

Fault-tolerant Techniques for Finite Ring Arithmetic Processors

M. Taheri, Student Member
G.A. Jullien, Senior Member
W.C. Miller, Member

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

Abstract

High speed attached processors are demanded in many real time applications; digital signal processing tasks are particularly suited to dedicated arithmetic processors. VLSI fabrication has made it possible to implement such processors and to enable them to operate at high speed. In some applications, reliability of processing results is of paramount importance and attention has to turn to fault-tolerant arithmetic processing systems. This will be increasingly the case as the designs become more compact and soft errors begin to be more predominant in the technology of sub μ geometries. This paper discusses a fault-tolerant mechanism for use with a new family of generic, finite ring, computational cells for systolic array processing of high speed DSP algorithms. The generic cell, and the fault-tolerant mechanism are based on a ROM/latch structure that is flexible, easy to design with and requires a low overhead fault detection circuit. The fault detection function is automatically distributed throughout the array, since the cell is used for all elements of the computation process. The fault correction circuitry is based on the use of the same generic cell, thus providing a universal solution to the protection of arithmetic processors.

Introduction

Intensive computational tasks (for example those required for many digital signal processing applications) are normally performed using an attached processor rather than using the host or executive system itself. These processors are usually designed for high throughput and the best architectures are of a parallel nature, where many arithmetic computations are performed at the same time. A particularly important class of architectures are systolic arrays where pipelining techniques distributed through many similar (or identical) processing elements ensure throughput rates governed only by the critical path through one computational element. As technology advances shrink device features, soft errors will be much more predominant and fault correction will be a necessity [22].

The use of residue arithmetic¹ for fault detection/correction is well known[2,14,24]. The interest in the RNS in the early days of computer technology was for its properties as a softly degradable numbering system. This was an important issue for the undependable components (vacuum tubes, relays) that were used in the early machines. The commercial use of transistors, and the associated increase in reliability, removed the necessity for such measures and so interest faded somewhat in the use of the RNS in error detection/correction systems. Over the past decade there has been a small but steady resurgence of interest in the use of RNS fault-tolerant systems[5,30], but the overhead involved is quite large[8]. The approach usually taken is to define a redundant residue system in which the independent operations over the various rings or fields of the system can be in error in any one of the channels (individual ring or field). Through sub-projection techniques, the faulty channel can be located and removed from the isomorphic mapping process that converts the RNS number back to the normal weighted magnitude number representation. The projections require extensive hardware, though the system of error detection also provides the most of the correction mechanism[8]. The fact that the RNS operates over entirely independent channels provides this natural ability for error isolation.

In this paper we re-examine fault-tolerant computations over finite rings and fields. We will show that using a new approach, in which fault detection is a separate, distributed, process within each computational element, that the fault-tolerant mechanism can be produced in a different and more efficient way than previous procedures. The natural error isolation of the RNS is preserved using this new technique.

Section I of the paper discusses a universal, finite ring/field arithmetic processing element which lends itself to VLSI fabrication techniques. Section

¹Residue Number System (RNS)

II discusses the modification required to the cell which enables the simple fault detection mechanism to operate. We also show, in this section, that fault avoidance and fault-tolerant techniques can be used for higher efficiency of the structure. A fault model for the single cell used as a building block in an entire processing structure, is based on the properties of the single cell and no block in the system need be more reliable than any other. Section III discusses a fault-tolerant system based on the structure of Section I.

We will start with some mathematical preliminaries.

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^[24]:

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

For cascades of operations, mixed ring (field) operations (such as the Chinese Remainder Theorem^[24] mapping) 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 $GF(m_k) = \{S: {}^a, {}^o\}$ $S = \{0, 1, \dots, m_k-1\}$

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

Where it is not obvious by context which ring the operation is being performed over, a subscript will be employed. Thus 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^[24]:

$$X = (x_0, x_1, \dots, x_{L-1}) \tag{1}$$

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:

$$A \overset{a}{\Delta} B \Leftrightarrow \{a_0^a b_0, a_1^a b_1, \dots, a_L^a b_L\}; \quad A \overset{o}{\Delta} B \Leftrightarrow \{a_0^o b_0, a_1^o b_1, \dots, a_L^o b_L\}$$

with $A, B \in R(M)$; $a_k, b_k \in R(m_k)$ and $R(M) \sim \overset{a}{\Sigma} R(m_k)$ (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, $\{m_i\}$, 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:

$$X = \sum_{k=1}^L \overset{a}{\Delta} \{ \overset{o}{m}_k \overset{o}{\Delta} [x_k^o (\overset{o}{m}_k)^{-1}] \} \tag{2}$$

with $\overset{o}{m}_k = M / m_k$, $X \in R(M)$ and $x_k \in R(m_k)$

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

Section I

Many matrix operations can be implemented using repeated multiply and add operations in a loop [11,12,18]. The operation performed is known as the inner product step processor (IPSP) [13] as shown in equation 1 and Fig 1.

$$Y_{out} = Y_{in} + (A_{in} \cdot X_{in}) \quad (1)$$

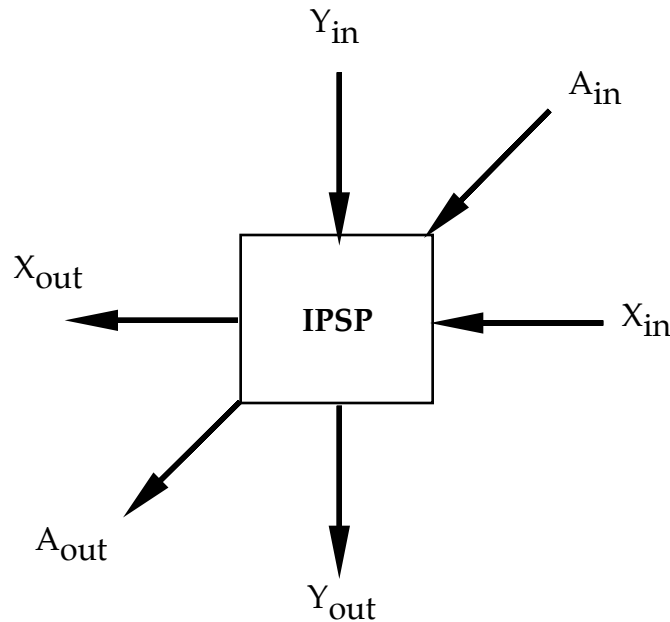


Fig 1 The Inner Product Step Processor (IPSP)

The processor takes input signals at the word level and produces a word level output. 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 an array structure, is used to process the required task. The IPSP can be sliced into bits^[3,16], 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 bit sliced IPSP (BIPSP) is shown in Fig 2.

The operation is described by:

$$y_{out} = y_{in} \oplus c_{in} \oplus (a_{in} \odot x_{in}) \quad (2a)$$

$$c_{out} = \text{Maj}(y_{in}, a_{in}, c_{in}) \quad (2b)$$

$$a_{out} = a_{in} \quad (2c)$$

$$x_{out} = x_{in} \quad (2d)$$

Several of these slices can be interconnected to form a word level IPSP [3,16,7].

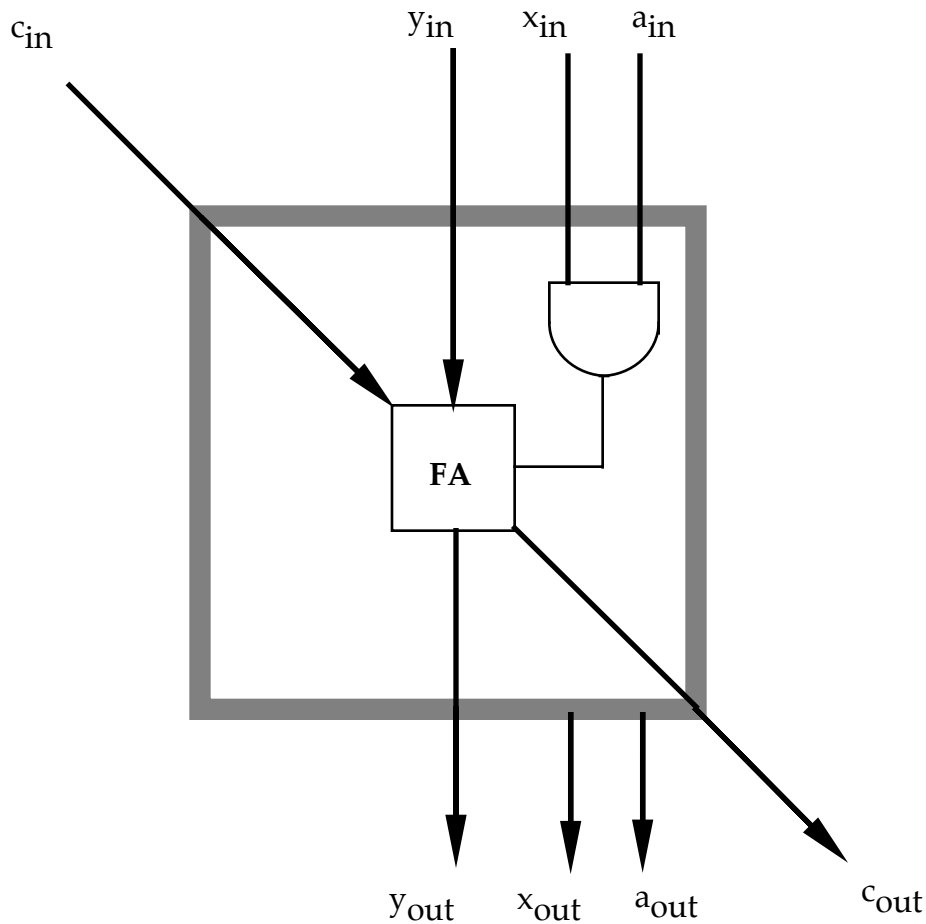


Fig 2. The Bit level IPSP (BIPSP)

Pipelining these simpler processors normally results in a much higher throughput compared to the throughput of the word level IPSP. A finite ring counterpart for the binary IPSP can not, for general rings, be constructed so easily. The most general approach is to use a ROM as a direct truth table implemented, as shown in Fig 3 [10].

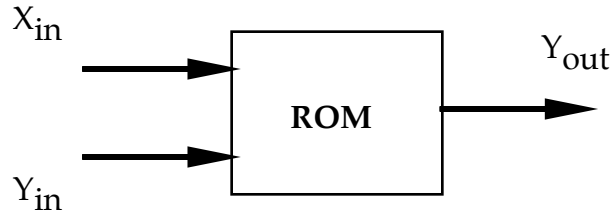


Fig 3. Single ROM implementation of the IPSP_m

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

$$Y_{out} = Y_{in} \circ_m [A_{in} \circ_m X_{in}] \quad (3)$$

All the inputs and outputs are B bit ring elements $Y, A, X \in R(m)$ with $B = \lceil \log_2 m \rceil$. 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 a finite ring, modulo m . It will be shown that the BIPSP_m has similar advantages over the IPSP_m as the BIPSP has over the IPSP. In addition the structure satisfies the requirements of modularity, homogeneity and local communication, sought after in 'good' VLSI designs. By repeatedly using a generic cell structure all closed computations can be performed with a parallel linear systolic array.

The operation of the BIPSP_m can be defined by¹:

$$y^{(i+1)} = y^{(i)} \circ_m [A_{in} \circ_m x^{[i]} \circ_m 2^i] \quad (4)$$

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

A possible implementation of a 4-bit BIPSP_m cell is shown in Fig 4.

¹At this point we will imply the 'm' subscript on the finite ring arithmetic operators by context.

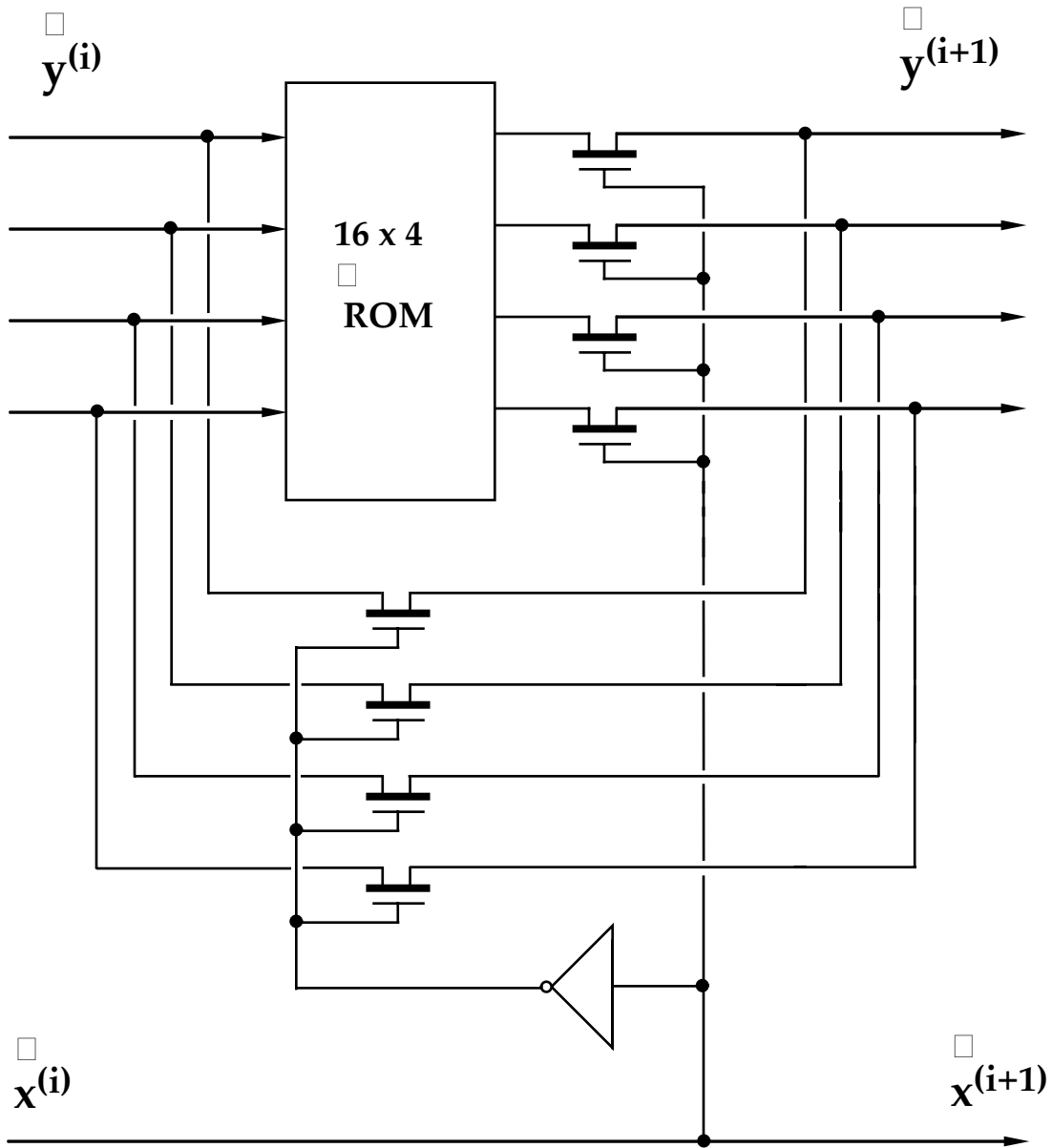


Fig 4 BIPSP_m implementation

The inputs to the cell are $y^{(i)}$ and $x^{(i)}$: the output is $y^{(i+1)}$. The cell contains a ROM of size mB bits and a set of steering switches; the ROM stores the operation of:

$$y^{(i)} \oplus [2^i \circ A_{in}]$$

The cell computes the following:

$$\text{For } x[i]=1: \quad y^{(i+1)} = y^{(i)} \circ [2^i \circ A_{in}] \quad (5a)$$

$$\text{For } x[i]=0: \quad y^{(i+1)} = y^{(i)} \quad (5b)$$

We can expand eqn (3) as:

$$Y_{out} = Y_{in} \circ \left[\sum_{j=0}^{B-1} \{A_{in} \circ x[j] \circ 2^j\} \right] \quad (6)$$

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 $BIPSP_m$ cells, as shown in Fig 5 for $B=4$.

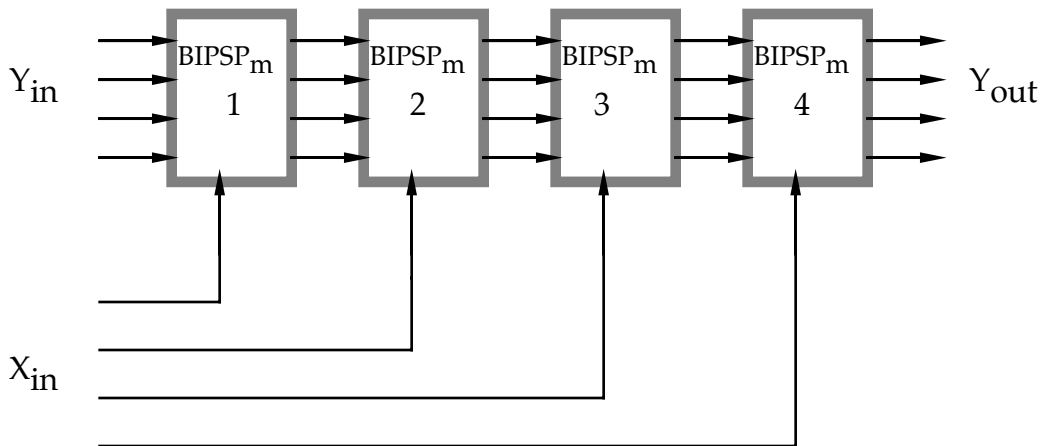


Fig 5 A 4-bit $IPSP_m$ using 4 $BIPSP_m$ cells

This technique only requires $B^2.m$ ROM bit locations compared to $B.m^2$ bits for the single ROM $IPSP$ structure. This yields significant savings for large m . The speed of operation will improve considerably by pipelining each $BIPSP_m$. If we modify the structure of Fig 4 by performing a cyclic rotation of the X data path, and include pipeline latches, the resulting array will contain only one

type of cell. The implementation is shown in Fig 6.

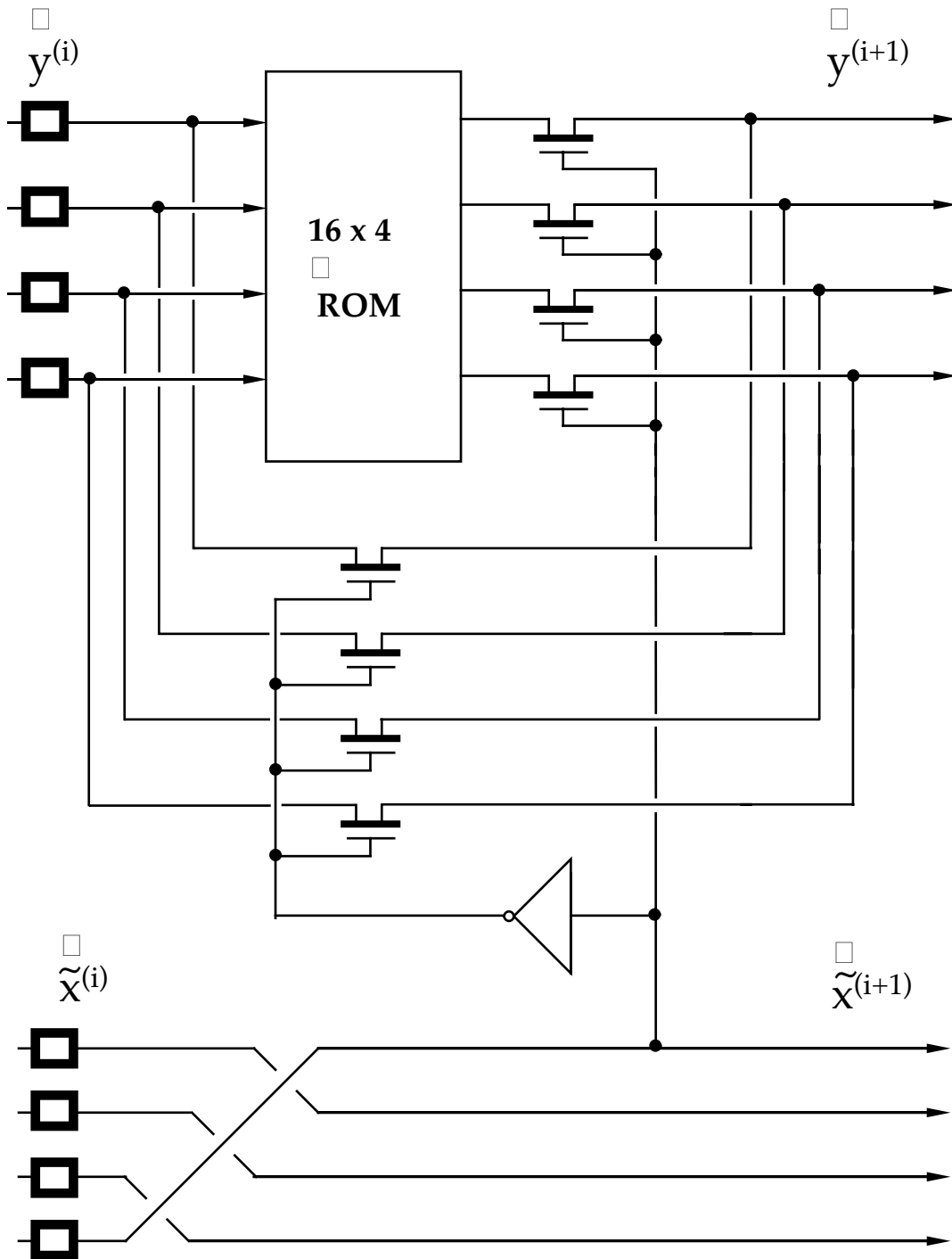


Fig 6 Universal BIPSP_m cell structure

The X sequence has been modified, notationally, to \tilde{X} to indicate the cyclic shift. Note that the entire X sequence is available at the output and that it is

in its original bit configuration after passing through B cells; i.e. $\bar{x}^{(0)} = \bar{x}^{(B-1)}$. An alternative scheme is to double the size of the ROM and feed $x^{[i]}$ as an extra address bit. There seems no merit in such a modification since both the area and access time of the ROM will increase. From our observations, the critical path through the steering circuitry is less than that through the ROM and so the ROM access time dominates the cell speed.

Section II

In this section the various types of errors that are encountered and appropriate approaches for designing reliable systems will be discussed. We will also present a simple modification to the BIPSP_m cell which enhances the operation in terms of error detection.

ERRORS

An erroneous output may be caused by fault(s) on the chip which may be the result of improper design, imperfect manufacturing process, etc. Detecting a high percentage of the faults in this category requires rigorous testing and is expensive and time consuming. Even satisfactorily tested chips may not be flawless or they may become damaged during operation. Transient effects, such as alpha particle penetration; cosmic radiation; transients on the power lines; general logic switching noise, are sources of soft errors and in some cases they may damage the chip permanently^[1,17,19]. A more reliable system can be designed by incorporating techniques which identify and correct erroneous outputs.

RELIABLE SYSTEMS

A reliable system provides correct results in spite of faults having occurred. Such systems can be designed based on one of the following strategies^[24]:

- 1) fault avoidance (fault intolerance)
- 2) fault-tolerance

1. Fault avoidance: Fault avoidance seeks to reduce the possibility of failures occurring in the system. This is achieved through the use of sophisticated design rules. Using a set of special 'high yield' design rules on some parts of the chip enables one to assume that those parts

are fault free; this sometimes allows more straightforward procedures to be used on the remainder of the chip in terms of detecting/correcting faults. Mangir and Avizienis^[15] discuss the yield improvements when this technique is used on the interconnections between fault-tolerant cells. Designing a highly reliable system based only on this philosophy, however, will be difficult and costly.

2. *Fault-tolerance:* Fault-tolerance is achieved by the use of redundancy, and is usually split into two categories: temporal redundancy, and physical redundancy. In temporal redundancy, once an error is encountered certain procedures are activated which re-compute the result. Physical redundancy, on the other hand, is achieved through the use of extra hardware^[21]. Fault masking is one of the approaches in this latter category. An example of the fault masking technique is to triplicate the original circuit; a reliable majority voter on the output decides between the end results^[28]. This leads to a three fold increase in hardware.

In this paper our main interest is in correcting isolated 'soft' errors, though our technique will also handle hard errors. We will combine fault avoidance techniques with physical redundancy to form a reliable signal processing system. Our approach is to use fault avoidance by constructing fault detection circuitry around the generic cell; the 'fault free' circuitry comprising a small fraction of the cell hardware.

A major component of the structure, introduced in section I, is the ROM table. Fault detection and correction techniques for semiconductor memories such as various coding strategies, have been reported in the literature^[19,29,17]. An efficient scheme has been described^[4] which uses three parity bits to detect faults in ROMs. In this section we will show how the use of two parity bits allows fault detection in the ROM and the steering switches when the ROM is used in a linear array^[25]. In this paper we use an established fault model^[4,6] and also assume only a single fault can occur in the cell at any time. Fig 7 shows the modified version of the cell. The ROM wordlength and input/output lines are increased by two bits; a checking circuit, consisting of two gates, has been added to the original circuit.

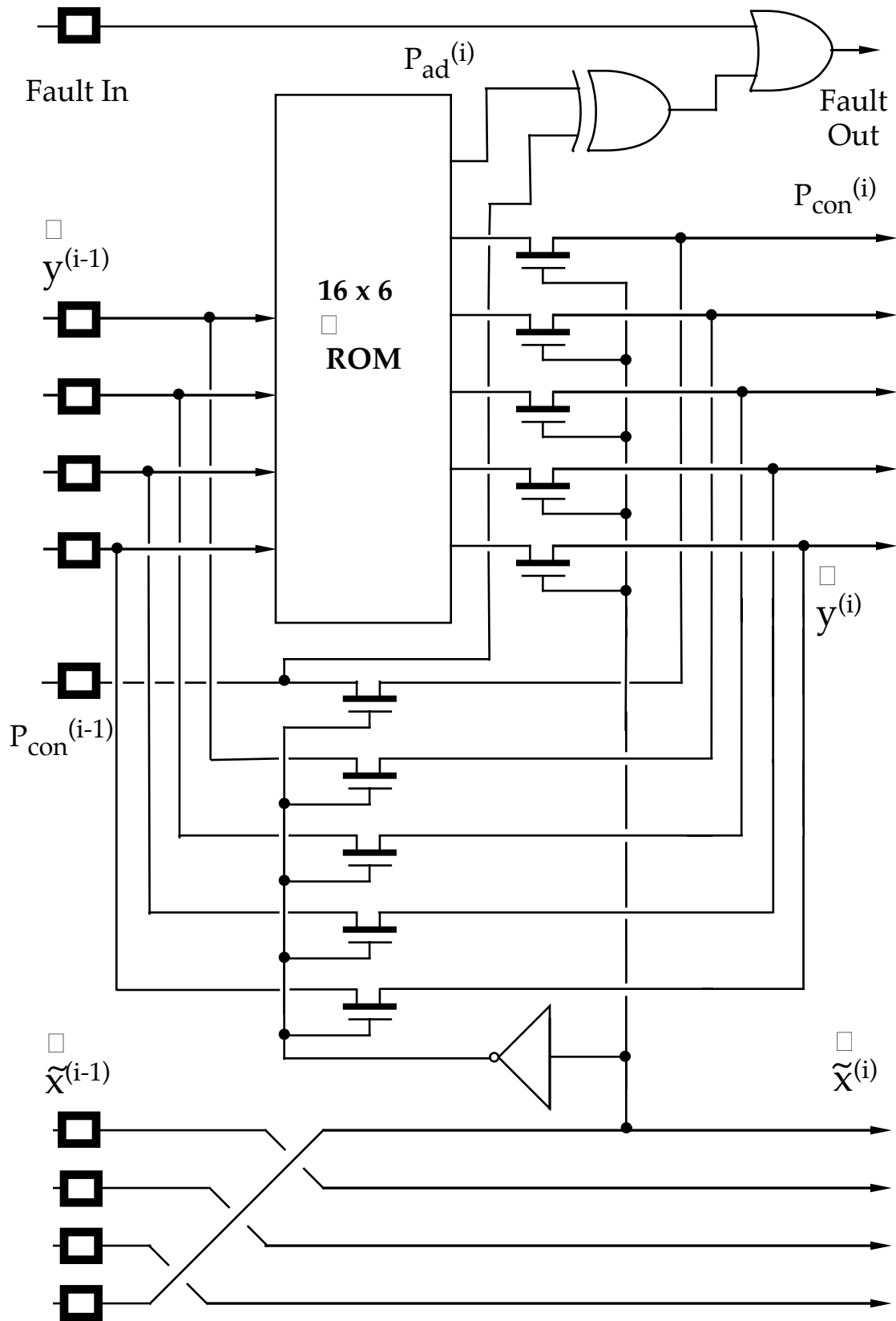


Fig 7 Systolic Cell with Fault Detection Circuitry

The inputs to the cell are a set of data lines $\{y^{(i-1)}, \bar{e}X^{(i-1)}\}$, a parity bit $P_{in}^{(i-1)}$ and a fault bit $F^{(i-1)}$. Input data lines to the i th stage of the systolic array, $y^{(i-1)}$, are used to look-up the output data $y^{(i)}$. At the same time, two parity bits are looked-up at the same location. The first is the output parity bit, $P_{out}^{(i)}$, which we refer to as the 'content parity check'. This parity check is calculated based on the following:

$$P_{con}^{(i)} = y_0^{(i)} \text{ a}_2 y_1^{(i)} \text{ a}_2 y_2^{(i)} \text{ a}_2 y_3^{(i)}$$

where $y_j^{(i)}$ is the j th bit of $y^{(i)}$. Note that a_2 is the exclusive-OR function. The second parity bit, $P_{ad}^{(i)}$, which we refer to as the 'address parity check', is generated as follows:

$$P_{ad}^{(i)} = y_0^{(i-1)} \text{ a}_2 y_1^{(i-1)} \text{ a}_2 y_2^{(i-1)} \text{ a}_2 y_3^{(i-1)}$$

In the zero error case, the fact that the address parity check in any cell is equal to the content parity check in the previous cell is the key to the operation of this fault detection structure. The fault detection circuitry generates the fault signal:

$$Fault\ Out = Fault\ In \text{ a}_2 \{ P_{ad}^{(i)} \text{ a}_2 P_{con}^{(i-1)} \}$$

where *Fault Out* is true if there has been a fault detected in any of the previous cells. This signal is pipelined through the array with the fault detection logic acting in a 'daisy chain' fashion. The error signal travels along with the processed data so that at the end of the chain the erroneous output samples are synchronously flagged.

In order to determine the detection properties for singly occurring faults, we can partition the cell into four different categories and investigate each separately as below:

A) Consider the elements which only contribute to the generation of data bits $y^{(i)}$. These elements are the memory planes which store this information, the lines to and from steering switches and the latches on these lines. A fault in any one of these elements (by assumption) will change only one output bit, therefore the check circuit will flag these and the fault will be detected.

B) Consider the elements which only contribute to the generation of parity bits. These elements are the parity memory planes and the fault flag, *Fault Out*. Since we have assumed a maximum of one failure in

the cell the only fault situations that can occur with these elements are erroneous detection of a fault when the data bits are in fact valid or no fault detection when the data is in error. In the first case the false signal causes the cell result to be flagged as erroneous; this does not jeopardize the operation since, by assumption, all the other channels, including any redundant channel, are operating correctly. The other case requires both an output fault and check circuit fault at the same time. By the single fault assumption this can not occur.

C) Consider the elements which contribute to the generation of both the fault flag *Fault Out* and the output data $y^{(i)}$. These elements are the ROM row or column decoder. Chen et al^[4] have shown that the address of each incorrectly selected location will differ by exactly one bit from the address of the correct location. In the following theorem we show that a faulty decoder, with a single fault, will be detected.

Theorem 1: Let the input to the decoder be the set $A=\{a_L, a_{L-1}, \dots, a_k, \dots, a_0\}$ and assume that the faulty decoder misinterprets one bit of the input data, this fault will be detected.

Proof: Suppose that the input set selects location R with an address parity of Pa . Assume, as a result of the fault, that the location selected is $\bar{A}=\{a_L, a_{L-1}, \dots, \bar{a}_k, \dots, a_0\}$ with a parity bit, $\bar{P}a$. Since the set A and \bar{A} differ in one bit, then their parities, Pa and $\bar{P}a$ will not be equal and the fault will be detected.

D) The $X^{(i)}$ bits and their corresponding latches are not checked in this approach; however, if another parity bit is introduced and an exclusive-OR function implemented with this bit and one of the $X^{(i)}$ bits in each cell, as shown in Fig. 8, then these errors may also be detected.

We note that the cell is capable of detecting errors even if the input, $X^{(i)}$, is zero. The reason for this is that the ROM still generates the parity bit outputs even if the steering switches by-pass the ROM for the $y^{(i)}$ data.

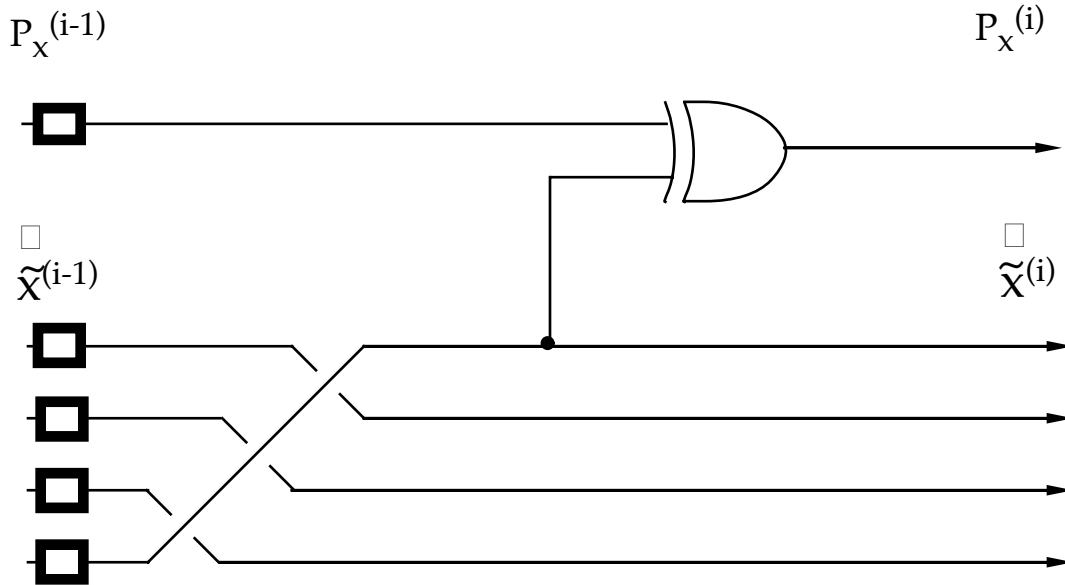


Fig 8 Parity fault detection circuitry for $X^{(i)}$ data

Section III

In this section we present an example of a fault-tolerant system based on the fault detection architecture described above. We will first review the 'classical' fault detection and correction techniques using RNS representations.

REVIEW OF RNS FAULT-TOLERANT TECHNIQUES

A standard residue number system is represented with a set of relatively prime moduli $\{m_1, m_2, \dots, m_L\}$.

The dynamic range of the system is $M = \prod_{i=1}^L m_i$.

A redundant RNS (RRNS) is defined with a moduli set augmented by r redundant moduli

$$\{m_1, m_2, \dots, m_L, m_{L+1}, \dots, m_{L+r}\}$$

The legitimate range is $M = \prod_{i=1}^L m_i$ and the total range is $M_T = \prod_{i=1}^{L+r} m_i$.

Any number belonging to the interval $[M, M_T)$ is illegitimate^[5]. Mandelbaum^[14] proved that a redundant residue number with r redundant moduli will be able to detect r errors and correct $r/2$ errors. Barasi and Masetrini^[2] showed that an error in any of the residue digits in a RRNS maps the number into the illegitimate range, they also showed that, in an RRNS with $r=2$, if a single residue digit is in error only one of the possible combinations of $L+1$ out of $L+2$ moduli maps into the legitimate range and the remainder map into the illegitimate range. Etzel and Jenkins^[5] used the digits from a mixed radix conversion of the number to check the viability of the projections. Later, Jenkins^[8] designed a self checking error checker (the device which implements both error detection and error location).

The main purpose of the error checker is not so much for error correction, but rather isolation of the faulty modulus^[8]. Basically, the error checker projects the represented number into various subspaces in the RNS algebra using a mixed radix conversion (MRC) algorithm, and checks the (MRC) digits generated in order to detect the faulty channel. The main problem is that the hardware required for the checking and detection is cumbersome and as prone to errors as the system it is protecting! If we implement an RNS system based on the simple cell described in Section II, then error detection (over a subset of the error space) can also be implemented in a straightforward fashion.

It is important to show that this error detection scheme can be used in a correction scheme. The following theorem shows that a single error can be corrected with the information restricted to knowing the residue in error.

Theorem 2: A redundant residue system with one redundant modulus ($r=1$) allows correction of one error if the erroneous modulus is discarded.

Proof: Let a legitimate number, X , be represented as:

$$X = (x_1, x_2, \dots, x_L, x_{L+1})$$

in an RRNS with a moduli set of $\{m_1, m_2, \dots, m_L, m_{L+1}\}$, where m_{L+1} is the redundant modulus, $m_{L+1} > m_i$ for $i=1, 2, \dots, L$ and the dynamic range of the system is $M = \prod m_i \forall i \leq L$. Assume there is an $X' < M$ which can be represented by the same set of residue digits, with the erroneous digit, x_j , removed. Then $X' = (x_1, x_2, \dots, x_{j-1}, x_{j+1}, \dots, x_L, x_{L+1})$.

Since $M < \prod_{i=1, i \neq j}^{L+1} m_i$, we also know that $X = (x_1, x_2, \dots, x_j, x_{j+1}, \dots, x_L, x_{L+1})$.

Since the mapping is one-to-one and onto, X and X' must be identical.

This theorem basically allows us to use the information provided by the fault

flag in each modulus and reduce the redundancy from $r=2$ to $r=1$.

As a simple example, consider a system with 3 non-redundant moduli and one redundant modulus. The systolic cells, discussed in Section I, are used to generate a processing array; e.g. a FIR filter with encoding from binary^[26,27]. The fault flag at the end of the linear systolic structure reflects the validity of the output; this flag is fed into a redundant RNS to binary decoder where each data output is examined in order to find three error free channels out of the four. This is accomplished by examining the fault flags accompanying each data line. By logically ORing the fault flags from each combination of 3 out of the 4 flags, the first OR output that is logical zero (i.e. all flags register correct data) represents the correct decoder output. This output is again the logical OR of the outputs from two parallel channels in the decoder (see the next discussion). Fig 9 shows the redundant decoder block diagram for this example.

It is important that the decoder be at least self checking, otherwise the fidelity of the entire system is in question. Ideally the generic fault detecting cell, discussed in this paper, will be used to perform decoding, leading to a unified implementation procedure over the entire processing system. With this in mind we will examine the existing decoding methods.

Principally there are two methods for converting from RNS to WMS; the Chinese remainder theorem (CRT) and the mixed radix (MRC) technique. Both techniques have been examined extensively in the literature^[23,24,30]. The CRT usually requires either a modulo M adder (where M is the dynamic range of the RNS being converted) or, in the case of a recent fractional conversion scheme^[23], an L -input binary adder that is $\lceil \log_2 M \rceil + L$ bits wide. The extra L bits are required to correct for round-off errors in each of the fractional elements being summed. Both approaches require a summer with at least the full dynamic range of the number system, and are clearly not suitable candidates for our small dynamic range generic cell. The use of a large dynamic range connected element (in this case an adder), also introduces problems in error detection and correction which our approach, so far, has overcome. The MRC technique naturally retains computations over the individual rings of the RNS being converted. In its traditional form it requires a triangular array of interacting residue calculations (the height of the triangle is L), and the result is obtained in a mixed radix WMS. The radices are the same as the RNS moduli; this choice is crucial for the MRC to work successfully. Jenkins^[8] presented a fault-tolerant procedure based on the MRC in which both physical and temporal redundancies are used. The technique projects all the possible $L+1$ out of $L+2$ residues in sequence and checks for a projection which falls in the legitimate interval. The encoder has a self checking design; however, finding the correct output may require up to L iterations.

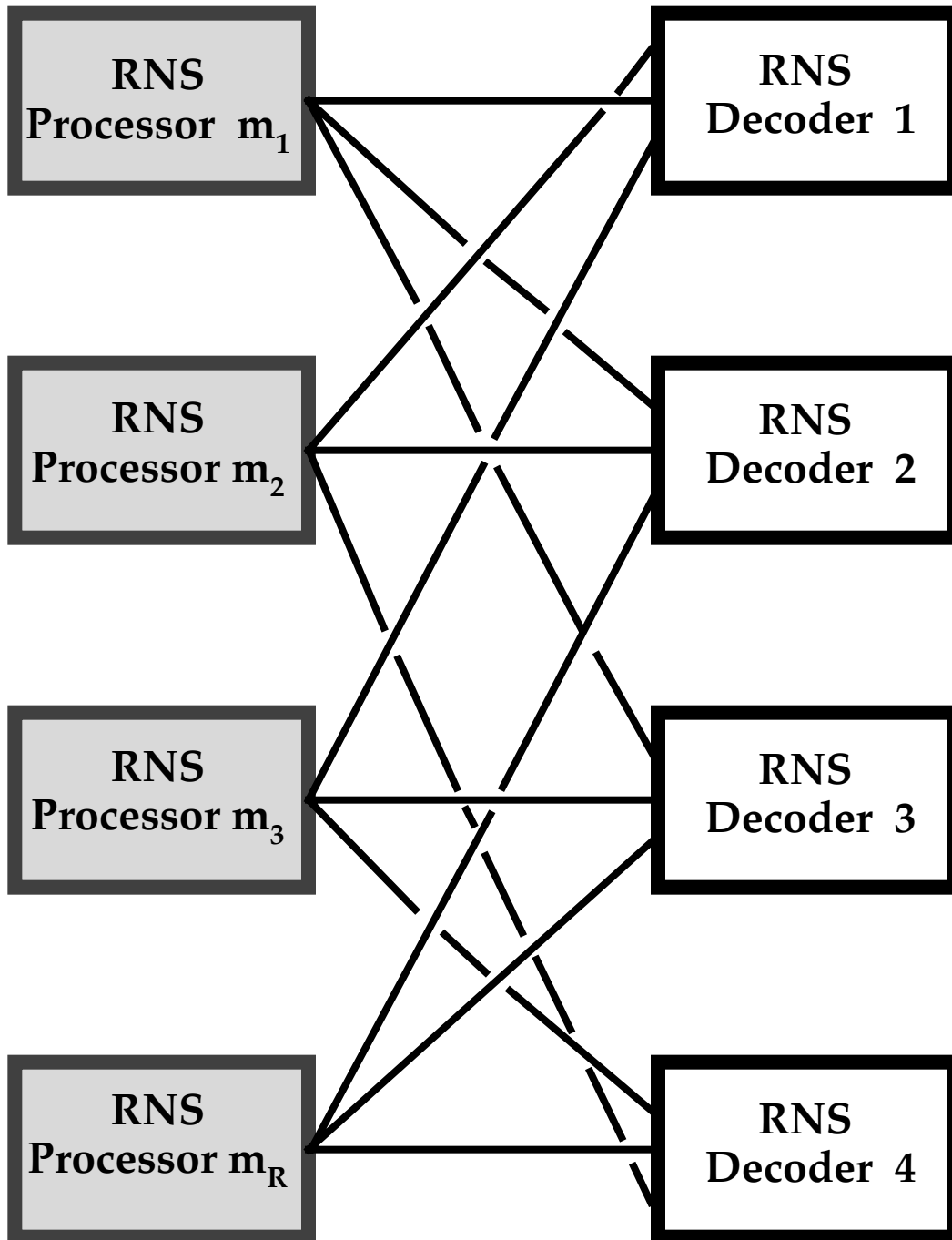


Fig 9 Redundant Decoder for $L=3$ $r=1$

The most natural vehicle for implementation, using our systolic cell, is found in a recently disclosed scheme^[20]. This technique allows computation over the individual rings and also produces the output in bit-sliced binary. The basic concept is to repeatedly perform base extension to a modulus of the form 2^t ; in this way the binary output is computed in slices of t bits. This technique lends itself to implementation, as shown in Fig 10, for our example system of $L=3, r=1$.

Each decoder in Fig 9 is represented in detail in Fig 10; note that the structure of each decoder is identical. Each decoder consists of a linear systolic array of B cells. There is some redundancy between the blocks of the different decoders and it is possible to reduce the total number of blocks implemented.

The contents of each block have the form:

$$R = \{A \text{ }^a_m B\} \text{ }^o_m K$$

where A and B are variables and K is a constant multiplier. This is amenable to bit slicing and so each block in fig 10 is capable of being sliced. For 5-bit moduli and a base extension power of two of 32, the contents of the blocks for decoder 1 are given in the following:

$$\begin{aligned} B_1 &= \{X_2 \text{ }^a_{m_2} [-X_3]\} \text{ }^o_{m_2} \{m_3^{-1}\} \\ B_2 &= \{X_1 \text{ }^a_{m_1} [-X_3]\} \text{ }^o_{m_1} \{m_3^{-1}\} \\ B_3 &= \{B_1 \text{ }^o_{32} m_3\} \text{ }^a_{32} X_3 \\ B_4 &= \{B_2 \text{ }^a_{m_1} [-B_1]\} \text{ }^o_{m_1} \{m_2^{-1}\} \\ B_5 &= \{B_3 \text{ }^a_{32} \{B_4 \text{ }^o_{32} m_2 \text{ }^o_{32} m_3\}\} \\ B_6 &= \{X_2 \text{ }^a_{m_2} [-B_5]\} \text{ }^o_{m_2} \{32^{-1}\} \\ B_7 &= \{X_3 \text{ }^a_{m_3} [-B_5]\} \text{ }^o_{m_3} \{32^{-1}\} \\ B_8 &= \{B_6 \text{ }^a_{m_2} [-B_7]\} \text{ }^o_{m_2} \{m_3^{-1}\} \\ B_9 &= \{B_8 \text{ }^o_{32} m_3\} \text{ }^a_{32} B_7 \\ B_{10} &= \{B_7 \text{ }^a_{m_3} [-B_9]\} \text{ }^o_{m_3} \{32^{-1}\} \end{aligned}$$

The output is obtained as the concatenation of B_{10}, B_9, B_5 (15 bits) in descending order of significance. For this example $L+r=4$ of these decoders, each operating independently, are required. A single fault occurring in any one of the moduli channels, or in the decoder, is flagged as an error at the end of the decoding operation. The correct output (in the case of no error, all of the outputs are the same and correct) is indicated as such by its flag.

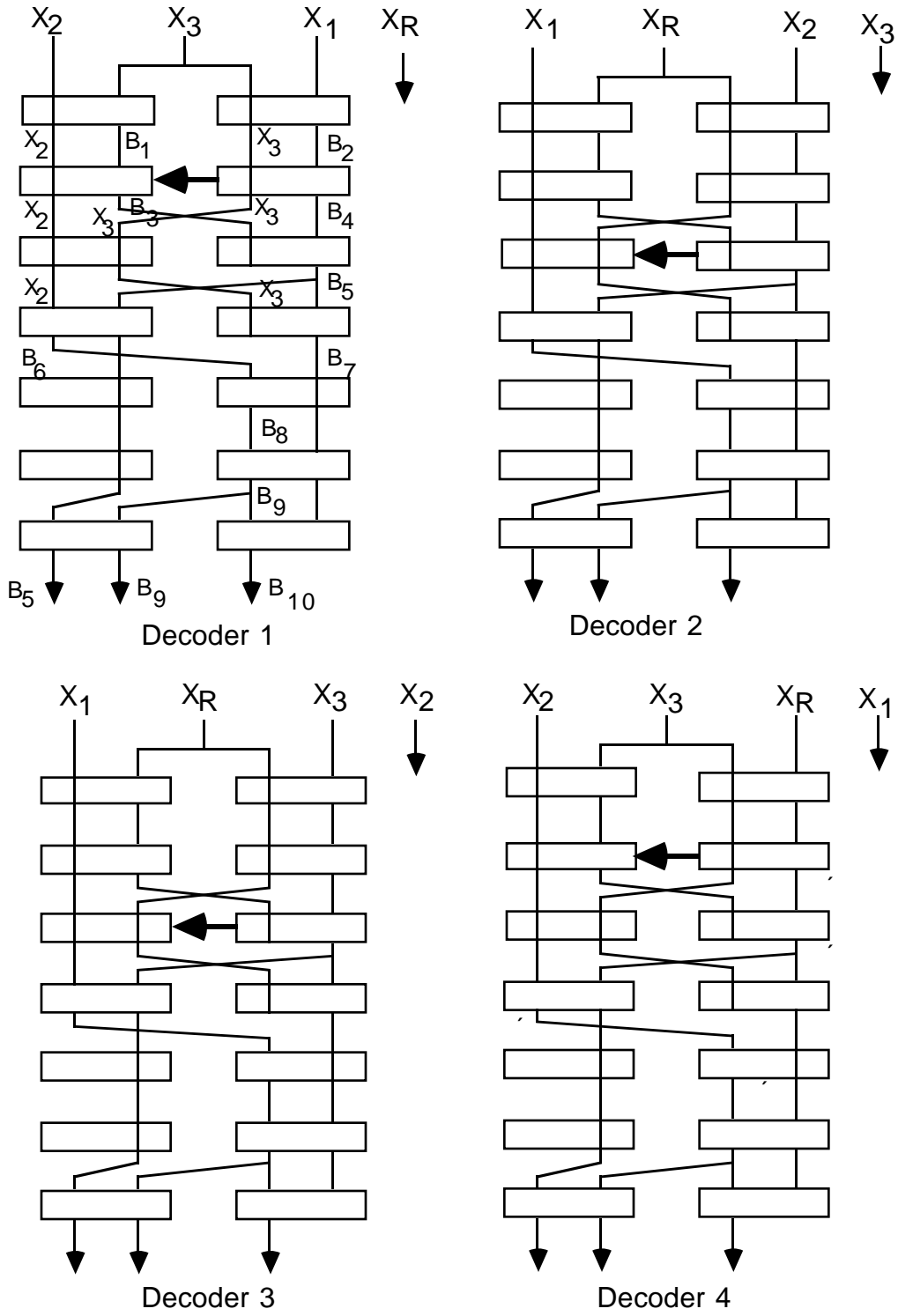


Fig 10 Example of Fault-tolerant decoder for $L=3$, $r=1$

The redundancy required by the system is:

- z 2 extra bits per word in each ROM of the cell
- z 2 extra steering switches per cell
- z 2 two-input gates per cell
- z 1 extra processing channel
- z L extra decoders

In some case it is required to monitor the different sections of the system for later improvement. The scheme described above only supplies this information at the end of the operation; however, by including simple binary counters, clocked by the *Fault Out* signals at different locations, the entire pipeline array can be monitored. A permanent or soft error which affects one or several stages of one module can be corrected since each stage performs operations on different samples at the same time. This fact allows several stages of the array to become faulty for a period of time and still enables correct results to be recovered. The only restriction is that the computation on any sample can fail, at most, in one modulus for the same set of clock periods.

CONCLUSIONS

This paper has described the construction of a processing cell which is capable of implementing the function of a fixed multiplier inner product step processor. The processor is ideally suited for many high speed digital signal processing requirements. We have shown how a simple parity check fault detection circuit can be constructed around the cell, and have demonstrated that it is capable of detecting all single faults within the cell. In a linear systolic array, the cells communicate the fault information through a pipelined 'daisy chain' structure, which allows the fault flag to accompany the data through the array. The paper has also described techniques for producing fault-tolerant systems by the use of redundant decoding techniques. The redundancy required by the system is detailed and a technique for monitoring of long term system faults has also been briefly described.

REFERENCES

- [1] J.A. Abraham and W.K. Fuchs, "Fault and Error Models for VLSI", (invited paper), Proceedings of IEEE, Vol. 74, No. 5, May 1986, pp. 639-654.
- [2] Ferruccio Barsi and Piero Maestrini, "Error Correcting Properties of Redundant Residue Number Systems", IEEE Trans. on Computers, Vol. C-22, No. 3, March 1973, pp. 307-315.
- [3] 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.
- [4] C. Y. Chen, W. K. Fuchs, and J.A. Abraham, "Error Detection in PLAs and ROMs", Proc. of IEEE Inter. Conf. on Computer Design, ICCD'85, October 1985, pp. 525-529.
- [5] Mark H. Etzel and W.K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters" IEEE Trans. on Acoustic, Speech, and Signal Processing, Vol. ASSP-28, No. 5, October 1980, pp. 538-545.
- [6] W.K. Fuchs and J.A. Abraham, " A Unified Approach to Concurrent Error Detection in Highly Structured Logic Arrays", Proc. 14th Int. Symp. on Fault-Tolerant Computing, June 1984, pp. 4-9
- [7] M. Hatamian and G.L. Cash, " High Speed Signal Processing, Pipelining, and Signal Processing", Proc. ICASSP, April 1986, pp. 1173-1176.
- [8] W. Kenneth Jenkins, " The Design of Error Checkers for Self-Checking Residue Number Arithmetic", IEEE Trans. on Computers, Vol. C-32, No. 4, April 1983, pp. 388-396.
- [9] Niraj K. Jha and Jacob A. Abraham, " Totally Self-Checking MOS Circuits Under Realistic Physical Failures", Proc. of IEEE Inter. Conf. on Computer Design, ICCD'84, October 1984, pp. 665-670.
- [10] 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.
- [11] H.T. Kung, " Why Systolic Architectures?", IEEE Computer Magazine, Vol. 15, No.1, Jan. 1982, pp. 37-46

- [12] 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.
- [13] C.E. Leiserson, "Area Efficient VLSI Computation", Ph.D. Dissertation, Dept. Computer Science, Carnegie-Mellon University, Oct. 1981.
- [14] David Mandelbaum, "Error Correction in Residue Arithmetic", *IEEE Trans. on Computers*, Vol. C-21, No. 6, June 1972, pp. 538-545.
- [15] T.E. Mangir and A. Avizienis, "Fault-Tolerant Design for VLSI: Effect of Interconnect Requirements on Yield Improvements of VLSI Design", *IEEE Trans. on Computers*, Vol. C-31, No. 7, July 1982, pp.609-616.
- [16] J.V. McCanny and J.G. McWhirter, "Implementation of Signal Processing Functions using 1-bit Systolic Arrays", *Electronic Letters*, Vol. 18, No. 6, March 1982, pp. 241-243.
- [17] Robrt J. McEliece, " The Reliability of Computer Memories", *American Scientific*, Vol. 252, No. 1, January 1985, pp. 88-95.
- [18] D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays", *Proc. of IEEE*, Vol. 71, No. 1, Jan. 1983, pp. 113-120.
- [19] David B. Sarrazin and Miroslaw Malek, "Fault-Tolerant Semiconductor Memories", *IEEE Computer Magazine*, Vol. 17, No. 8, August 1984, pp. 49-56.
- [20] 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*.
- [21] Daniel P. Siewiorek, " Architecture of Fault-Tolerant Computers", *IEEE Computer Magazine*, Vol.17, No.8, August 1984, pp. 9-18.
- [22] D. Siewiorek, "Architecture of Fault-Tolerant Computers", in *Fault-Tolerant Computing: Theory and Techniques*, Dhiraj K. Pradhan editor, Prentice-Hall 1986, pp. 417-466.
- [23] M.A. Soderstrand, C. Vernia, and Jui-Hua Chang, " An Improved Residue Number System Digital-to-Analog Converter", *IEEE Trans. on Circuits and Systems*, Vol. CAS-30, No.12, December 1983, pp.903-907.
- [24] N.S. Szabo and R.I Tanaka, "Residue Arithmetic and its Applications to Computer Technology", MacGraw-Hill, New York, 1967.

- [25] M. Taheri, G.A. Jullien, and W.C. Miller, " Fault Detection in RNS Systolic Arrays", IEE Electronics Letters, Vol. 23, No. 4, Feb. 1987, pp. 165-166
- [26] M. Taheri, G.A. Jullien, and W.C. Miller, " Systolic ROM Arrays For Implementing RNS FIR Filters", IEEE Intr. Conf. on Acoustics, Speech, and Signal Processing, ICASSP'87, April 1987,pp. 771-774.
- [27] M. Taheri, G.A. Jullien, and W.C. Miller, "High Speed Signal Processing Using Systolic Arrays Over Finite Rings", Submitted for publication in IEEE Journal of Selected Areas in Communications.
- [28] Yuval Tamir and Carlo H. Sequin, "Design and Application of Self-Testing Comparators Implemented with MOS PLA's " IEEE Trans. on Computers, Vol. C-33. No. 6, June 1984, pp. 493-506.
- [29] R. Michael Tanner, " Fault-Tolerant 256K Memory Designs", IEEE Trans. on Computers, Vol. C-33, No. 4, April 1984, pp. 314-322.
- [30] Fred J. Taylor, "Residue Arithmetic: A Tutorial with Examples", IEEE Computer Magazine, Vol. 17, No.5, May 1984, pp. 50-62.