

Redundant finite rings for fault-tolerant signal processors

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

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

ABSTRACT

Redundant Residue Number Systems (RRNS) have been proposed as suitable candidates for fault tolerance in compute intensive applications. The redundancy is based on multiple projections to moduli sub-sets and conducting a search for results that lie in a so-called *illegitimate* range. This paper presents RRNS fault tolerant procedures for a recently introduced finite polynomial ring mapping procedure (modulus replication RNS). The mapping technique dispenses with the need for many relatively prime ring moduli, which is a major draw-back with conventional RRNS systems. Although double, triple, and quadruple modular redundancy can be implemented in the polynomial mapping structure, polynomial coefficient circuitry, or the independent direct product ring computational channels, for error detection and/or correction, this paper discusses the implementation of redundant rings which are generated by (1) redundant residues, (2) spare general computational channels, or (3) a combination of the two. The first architecture is suitable for RNS embedding in the MRRNS, and the second for single moduli mappings. The combination architecture allows a trade-off between the two extremes. The application area is in fault tolerant compute intensive DSP arrays.

Keywords: redundant residue number systems, polynomial ring mapping, fault tolerance, DSP

1. INTRODUCTION

Redundant Residue Number Systems have been proposed, for many years, as suitable candidates for fault tolerance in compute intensive applications. The redundancy is based on multiple projections to moduli sub-sets and conducting a search for results that lie in a so-called *illegitimate* range¹. A variety of channel correction schemes are available based on this concept. The major draw-back to such schemes lie in the need to increase sets of relatively prime moduli, which do not lend themselves to exact hardware replication based on the variety of ring moduli chosen, and so increase design (and any replacement hardware) overhead.

The approach taken by traditional RRNS fault tolerant systems, is to effectively generate overlapping sets of moduli, each set capable of handling the computational dynamic range requirements. The overlapping sets are produced by increasing a base set of moduli, $(m_1 \dots m_L)$, by a number of redundant moduli, $(m_{L+1} \dots m_{L+R})$, such that $m_1 < m_2 < \dots < m_{L+R}$,

$\prod_{i \in (\mu_1 \dots \mu_\psi)} m_i \geq M$, where M is the required computational dynamic range, and the $\{\mu_i\}$ are sets of L modulus indices where

the total number of such sets is $\frac{(L+R)!}{L!R!}$. The results of computations, over each of the sets of moduli, are referred to as projections. Clearly if each of the projections yield identical results, then no fault has occurred. If some of the projections differ, then there are well established theorems to a) locate the channel in error and b) determine which is the correct projection (result). For

example, defining a *legitimate* range of $\prod_{i=1}^L M_i$, it can be shown that a channel error will produce a result, over the extended moduli set, $(m_1 \dots m_{L+R})$, outside of this legitimate range. Theorem 1 establishes the detection/correction properties of such a system.

Theorem 1.² The redundant residue number system, as described above, will detect R errors and correct $\left\lfloor \frac{R}{2} \right\rfloor$ errors. \square

The underlying theme of multiple projections can also be extended to the Modulus Replication RNS (MRRNS)³ (introduced recently by two of the authors) which uses novel polynomial ring mapping procedures for inner product type computations. The mapping allows large dynamic range computations to be performed over many replications of the same modulus, or possible small finite ring RNS mappings of this common modulus⁴. This finite polynomial ring mapping scheme allows independent bit-level systolic implementations of inner product matrix computations, with replicated independent bit streams⁵. Data are encoded as polynomial coefficients and then mapped to small finite independent rings over which algorithmic computations take place. Multi-indeterminate polynomial ring mappings allow complex number data manipulations without resorting to the traditional approach of using the quadratic residue number system. This alleviates the problem of limited moduli choices available with the 4K+1 constraint, and also produces homogeneous architectures, well suited for VLSI implementations.

2. THE MODULUS REPLICATION RNS (MRRNS)

This is a brief introduction to the MRRNS. More details can be found in the literature³.

2.1. Quotient rings

We let $R[X]$ denote the ring of polynomials in the indeterminate $X: R[X] = \left\{ \sum_{k=0}^n a_k X^k : a_k \in R, n \leq 0 \right\}$. If X_1, X_2, \dots, X_k are indeterminates then we define the ring $R[X_1, X_2, \dots, X_k]$ to be the ring of multivariate polynomials in the indeterminates.

The quotient ring $R[X]/(g(X))$ is defined to consist of all elements of the form $f(X) + (g(X))$, with $f(X) \in R[X]$. The more usual way of considering the quotient ring is to consider sums and products of polynomials reduced according to the equation $g(X) = 0$, that is, to consider the remainder after division by $g(X)$.

2.2. Polynomial binary integers

The essence of the MRRNS is the mapping of polynomial representations of the binary data into quotient rings and evaluating these polynomials at all possible combinations of roots of the ideal; effectively mapping to a direct product ring of many individual copies of the base ring (replication of identical ring calculations).

The data are represented by polynomials of the form given in eqn. .

$$s = \sum_{i_1=0}^{d_1} \sum_{i_2=0}^{d_2} \dots \sum_{i_n=0}^{d_n} s_{i_1 i_2 \dots i_n} 2^{(i_1 \beta_1 + i_2 \beta_2 \dots i_n \beta_n)} \quad (1)$$

where the data word width is $B = \beta_0 > \beta_1 > \dots > \beta_n$ with the proviso that $d_i + 1 \geq \beta_{i-1} / \beta_i$ for $0 < i \leq n$.

2.3. Mapping to the direct product ring

Each of the binary weights, 2^{β_i} , is now represented by an indeterminate, X_i , in a polynomial ring $R[X_1, X_2, \dots, X_n]$. This mapping is usually just wiring. By carefully selecting the polynomial factors of the ideal, we evaluate the quotient ring $R[X]/(g(X))$ at the roots of the ideal to generate the direct product ring map.

2.4. Complex data

It is possible to map complex binary data by reserving an extra indeterminate to represent the complex operator. We can also invoke quadratic residue rings and perform the QRNS map prior to the direct product ring map.

2.5. RNS embedding

In computing the DSP algorithm over each of the replicated rings, we may also embed a limited moduli set RNS in order to perform the ring computations. An interesting use of such an embedding is the use of the replicated modulus, $M = 105 = 3 \times 5 \times 7$, with only 3-bit ring calculations over the 3 moduli RNS⁴. The embedding can be performed before or after the ring replication. This choice can have implications on the final computational architecture.

2.6. Reverse mapping

Although not all of the forward mappings are isomorphisms, we carefully control the mapping parameters so that all the maps can, in practice, be reversed. If we do not use RNS embedding, then there is no integer Chinese Remainder Theorem computation required. The inverse polynomial mapping (polynomial CRT) takes place over the replicated ring (an inner product) and so does not involve a large modulus adder. The reverse polynomial substitution to obtain the weighted output can be performed with binary adders, and binary scaling can be used to great effect⁶.

3. MRRNS FAULT TOLERANT ARCHITECTURES

Redundant rings can be utilized to construct fault tolerant architectures for the MRRNS mapping procedure. In general, the number of faults that can be detected and/or corrected is proportional to the number of redundant rings formed. There are two distinct approaches investigated in this paper that produce redundant rings for fault tolerance: The first approach is based on redundant residues⁷ which accompany a limited RNS embedding within the MRRNS mapping. The second approach is based on using a spare general computational ring, and is most useful when a single replicated modulus is used. A combination of these two approaches is possible where a wide variety of faults can be detected and recovered without interruption to the data flow. The following subsections discuss each architecture in detail.

3.1. Architecture I

In this first architecture, redundant residues are used to detect and correct faulty residues. This architecture is useful where there is a certain amount of limited RNS embedding (i.e. inefficient with a single modulus MRRNS). The polynomial mapping is performed within the residue channel and thus any faults occurring in the mapping will be detected and corrected. A considerable advantage of the RRNS, is its ability to recover faults with the redundancy depending only on the number of redundant residues. The overhead, by definition, is the ratio of redundant residues to non-redundant residues, which differs from, for example, triple modular redundancy⁸ which always possesses a 200% overhead. In light of this, the efficiency of this architecture will depend on the number of factors of the polynomial ring modulus that are used for the RNS embedding.

Fig. 1. shows the general structure of this architecture. The first block at the top (*binary to polynomial*), converts the data from binary representation to polynomial representation where the equality given by eqn. (2) must hold.

$$\sum_{k=0}^{O_i} C_k X^k = \sum_{i=0}^{N-1} a^{[i]} 2^i \quad (2)$$

where C_k is the polynomial coefficient, X is the polynomial indeterminate, O_i is the input polynomial order, $a^{[i]}$ is the i th bit of the input sample, A , and N is the number bits in the input sample. It is possible to align the polynomial representation with the binary representation so that the conversion process reduces to simple wiring (e.g. $X = 2$ and $C_k = 0$ or 1)⁵. The second block of Fig. 1. performs binary to residue conversion on the polynomial coefficients, as given by eqn. (3).

$$C_{k,i} = |C_k|_{m_i} \quad \forall k, i \quad (3)$$

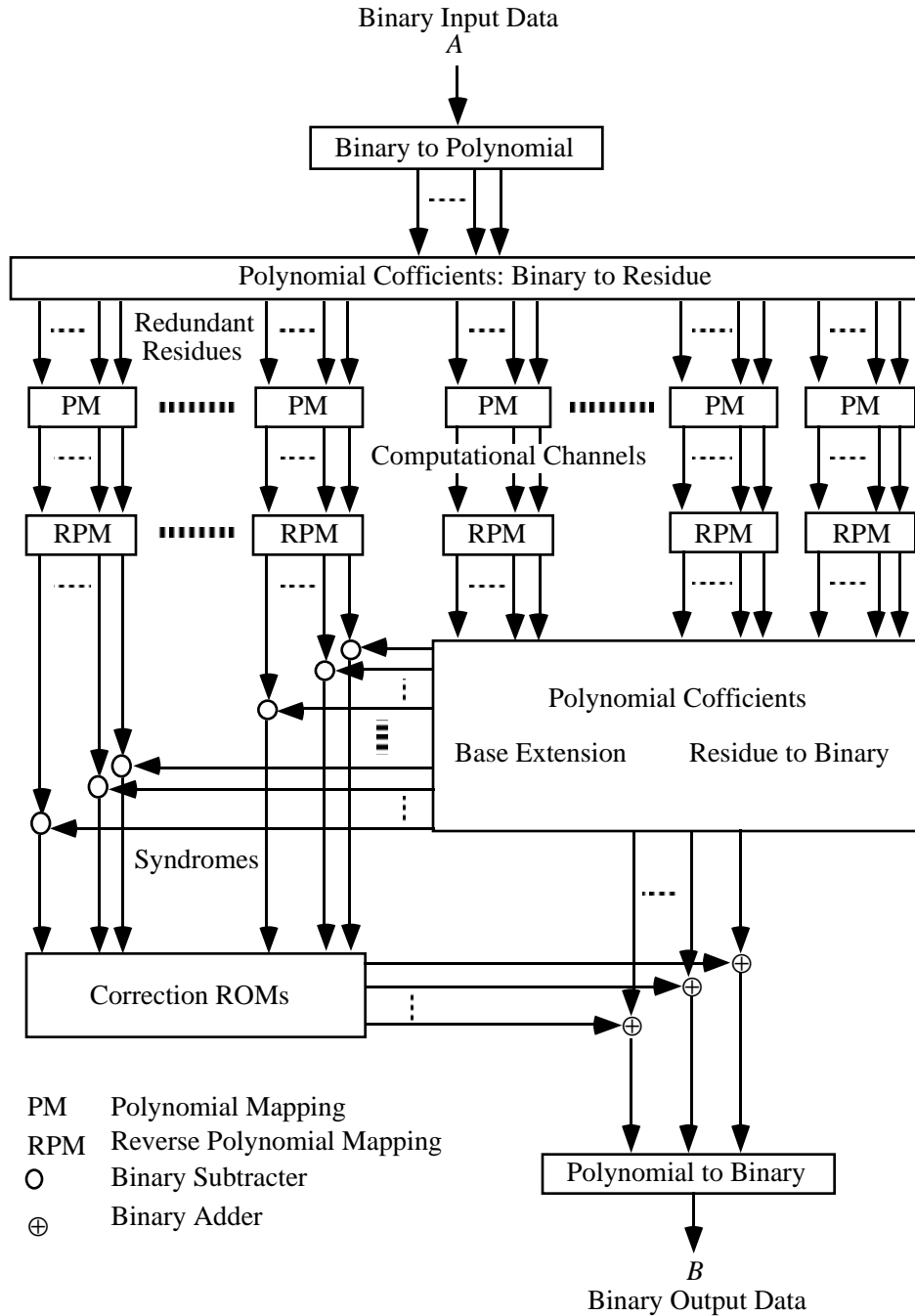


Fig. 1. Architecture I

If we choose a moduli set such that the input polynomial coefficients are always smaller than the least modulus, this block also reduces to simple wiring. The blocks labeled *PM* perform the polynomial map within the residue channel. They utilize the isomorphism that exists between the quotient ring and the direct product rings given by eqn. (4).

$$Z_{m_i}[X]/(g_i(X)) \cong \prod_j Z_{m_j} \quad (4)$$

$Z_{m_i}[X]$ is the mapped polynomial, $(g_i(X))$ is the ideal whose degree must be greater than the degree of the output polynomial, and $\prod_j Z_{m_i}$ is the direct product ring (j copies).

The polynomial map is an evaluation map where the polynomials are evaluated at the roots of the ideal. This completes the forward mapping stage and the DSP algorithmic computations take place here over small finite rings. The reverse mapping stage starts with the reverse polynomial mapping block (*RPM*) which simply reverses the polynomial mapping using the isomorphism of eqn. (4). We now employ mixed radix conversion (MRC), given by eqn. (5) (residue to mixed radix) and eqn. (6) (mixed radix to binary), with base extension performed on the polynomial coefficients to produce binary representations⁹.

$$d_{k,i} = \left(c_{k,i} \otimes_{m_i} \prod_{l=1}^{i-1} (m_l)^{-1} \right) \oplus_{m_i} \sum_{j=0}^{i-1} \left(-d_{k,j} \otimes_{m_i} \prod_{l=j}^{i-1} (m_l)^{-1} \right) \quad (5)$$

$$C_k = \sum_{j=0}^{L-1} 2^j \sum_{i=0}^{N-1} d_{k,i}^j p_i \quad (6)$$

$d_{k,i}^j$ is the j th bit of the i th mixed radix digit for the k th polynomial coefficient, $p_0 = 1$, and $p_i = \prod_{l=0}^{i-1} m_l$. The base extension

part will provide the necessary polynomial coefficients modulo the redundant residues, which will be compared to the actual results generated by the redundant residue channels to obtain the syndromes. A set of ROMs store all possible correction values to the output polynomial coefficients based on the syndrome values as address inputs. The contents of these ROMs can be found by simulating all possible errors and calculating correction values to maintain valid outputs (with two redundant residues, each possible single error has a unique set of syndromes). Fig. 2. shows the arrangement of the correction ROMs and the number and size of these ROMs is dictated by the number of redundant moduli, R , the number of output polynomial coefficients, the number of bits of the moduli set, and the number of bits of the computational dynamic range. For the system described in section 4, for instance, each ROM shown in Fig. 2. will have a 10 bit address input and a maximum of 23 bits per output word.

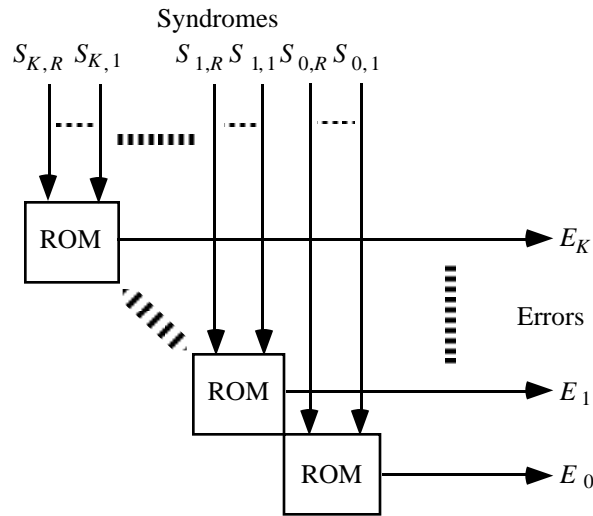


Fig. 2. Correction ROMs

Since the exponential growth of the ROM is due to address inputs, it represents a reasonable hardware solution to this correction architecture, where the input address width is based on the dynamic range of each residue channel and the output word width is based on the complete computational dynamic range. We may also consider using some recent results in minimizing ROM tables based on their contents¹⁰; these techniques allow minimization to about 12-bits using current sub-micron fabrication technologies¹¹.

An example, illustrated in Fig. 3. shows a minimized ROM core for a modulo 17 ring function. Each output bit has a separate minimized tree structure, as shown outlined in the figure. The lower 4-bits have been minimized to about 20% of their original transistor count, but the 5th bit (at the left of the figure), which only changes state for 1 out of the 17 possible states, has been reduced to less than 5% of the original transistor count. Since the minimization is based on graph theoretic pattern matching¹², hidden decompositions, which may not be discovered using algebraic decompositions on ROM table functions, are uncovered and used to reduce the look-up table area.

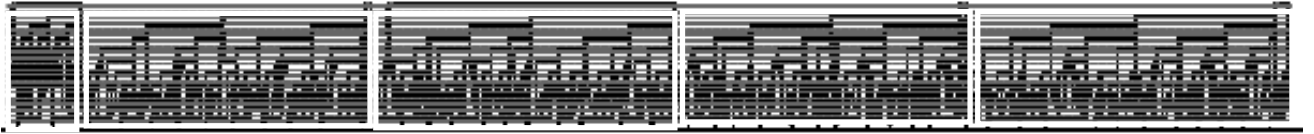


Fig. 3. Core of a Mod 17 function minimized ROM table

The last block converts the data from polynomial representation to binary representation by simply adding the polynomial coefficients with the proper weights, based on the polynomial indeterminate given by eqn. (7).

$$B = \sum_{k=0}^{O_o} X^k C_k \quad (7)$$

Here O_o is the order of the output polynomial. If the indeterminate is a multiple of 2 then binary adders can be used to obtain the final binary output.

The architecture detects/corrects errors as follows: if an error occurs within a residue channel, the polynomial coefficients generated by the base extension will not match those output from the redundant residues, and hence non zero syndromes will result. Based on the value of the syndromes, the ROM will look up stored correction values. The polynomial coefficients will be corrected by adding these correction values, with no interruption to the data flow. There are other techniques that utilize redundant residues for fault tolerance¹³, but the one adopted in this paper seems to be the most efficient to detect and correct one faulty residue. It is possible to correct more than one faulty residue with more than two redundant residues; however, in this paper, we restrict ourselves to two redundant residues.

3.2. Architecture II

The architecture, shown in Fig. 4., is based on building a spare general computational channel that can replace any given faulty computational channel.

This is most useful for a single modulus MRRNS mapping. If the non-redundant channels use general computations (i.e. no fixed coefficient multipliers), then the spare channel is identical to any of the non-redundant channels. If the non-redundant channels take advantage of fixed coefficient multiplications, then the spare channel will be larger to accommodate general multiplication. The spare channel is activated using a set of transmission gates and uses a ROM to extract the appropriate multiplicative and additive coefficients that correspond to the faulty channel.

The error detection scheme is implemented in each channel, at the circuit level, by utilizing a parity bit¹⁴. Fig. 5. shows the general structure of a bit slice computational cell used to construct the DSP algorithm in the computational channels.

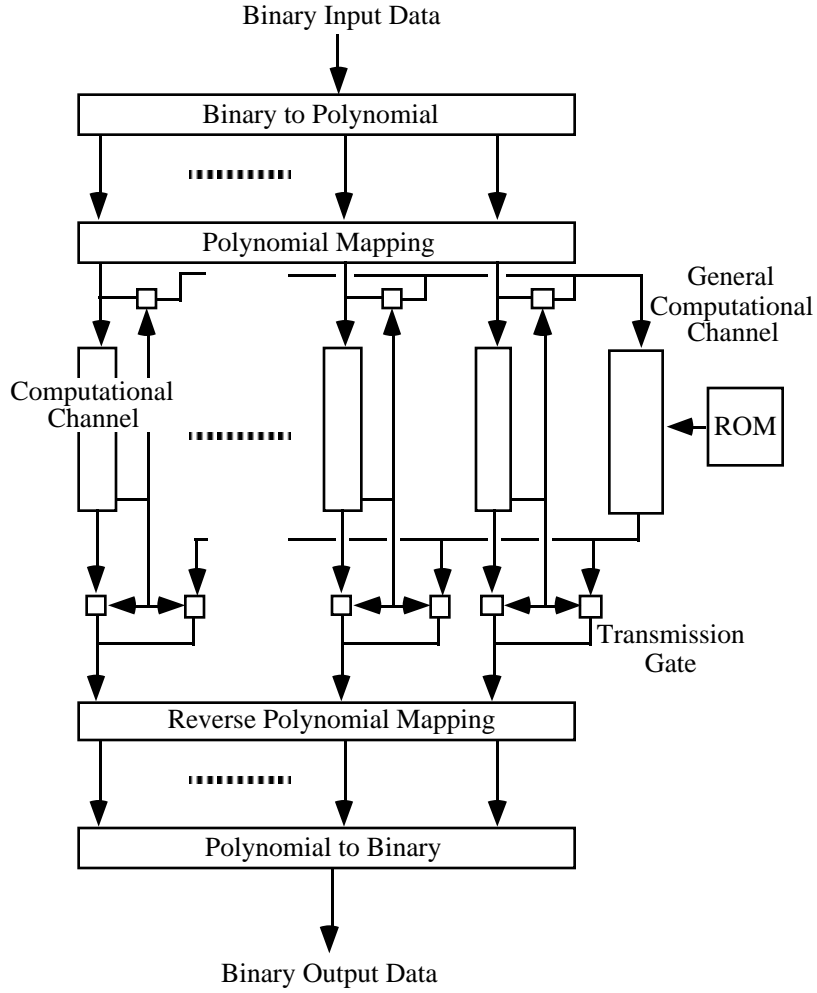


Fig. 4. Architecture II

For clarity, Fig. 5. shows only four bits for inputs X and Y. The cell uses two parity bits, a content parity check, P_{con} , and an address parity check, P_{add} , to check the parity of the look-up cell (or steered) output. Any discrepancy in the two parity bits will result in a fault flag being generated and travelling with the erroneous data. The pipeline latches are not included in the diagram; they may be placed at the input or the output data lines.

The values of the parity bits are obtained from eqn. (8) and eqn. (9).

$$P_{con}^{(i)} = \sum_{j=0}^{N-1} \oplus_2 y^{[j](i)} \quad (8)$$

$$P_{add}^{(i)} = \sum_{j=0}^{N-1} \oplus_2 y^{[j](i-1)} \quad (9)$$

N is the number of bits of the input Y. A fault is held and propagated through the fault flag bit by comparing the parity check bits of each processing cell and ORing the result with the flag, as in eqn. (10).

$$FaultOut = FaultIn \vee \left\{ P_{add}^{(i)} \oplus_2 P_{con}^{(i-1)} \right\} \quad (10)$$

If the fault flag is true at the end of the computational channel, then a fault has occurred in at least one of the processing cells.

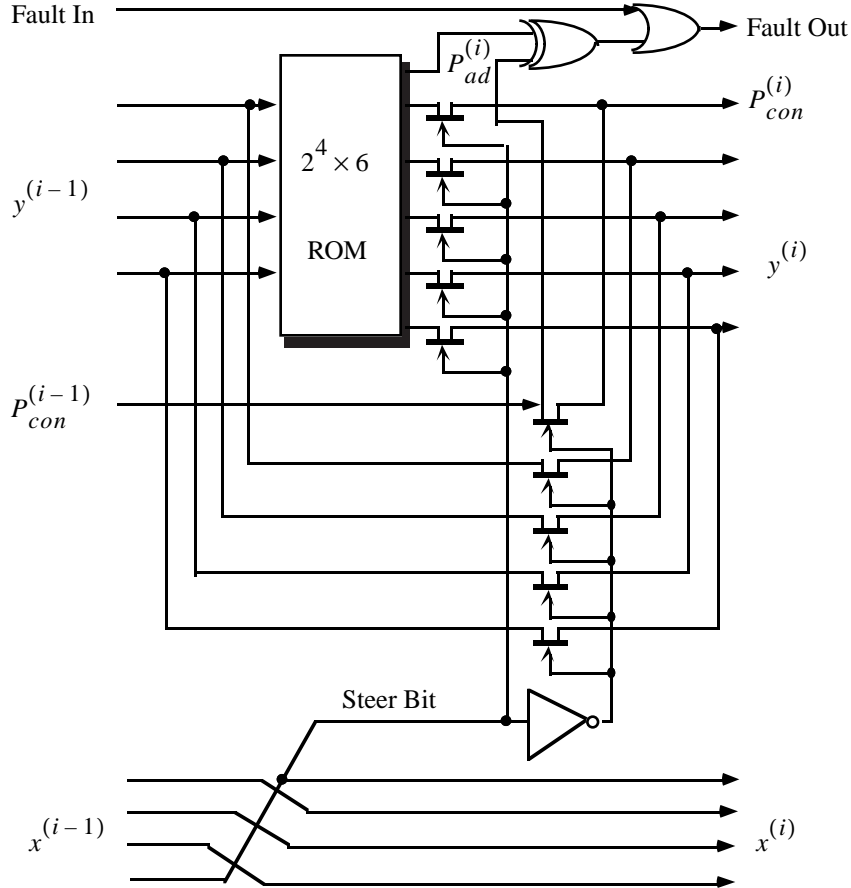


Fig. 5. Bit slice cell with fault detection

The flag is then used to switch the transmission gates such that the data of the faulty channel is directed to the spare channel so the output of the spare channel replaces the output of the faulty stage.

The overhead associated with the fault detection circuitry depends on the number of bits required to represent the channel residues. For a 5-bit residue, the overhead, for a conventional 2-phase dynamic ROM circuit, is about 25%. In the VLSI layout shown in Fig. 6., using the MOSIS scalable CMOS technology, the fault detection circuitry is completely contained in the outlined section on the right of the figure. This section includes the extra 2-bits required for each of the 32 ROM words, the fault detection logic and the extra two data latches.

Note that we can assign several spare channels to replace distributed faulty channels. One of the draw-backs of this architecture is that the polynomial mapping stage has to be error free. This problem is alleviated by Architecture III.

3.3. Architecture III

This architecture combines both techniques and is shown, at a reduced detail level, in Fig. 7.

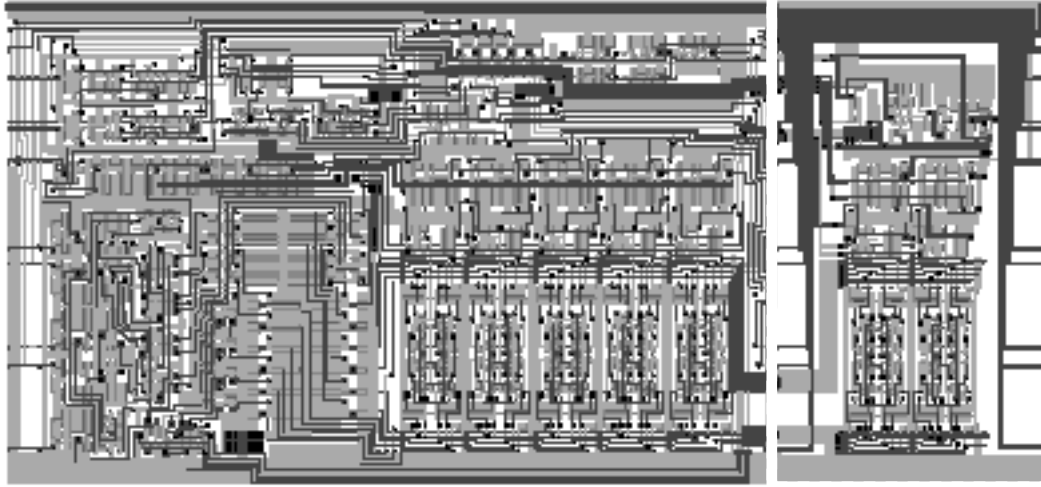


Fig. 6. Layout of a 5-bit fault detecting cell

Redundant residues are utilized to detect and correct faulty residues while spare general computational channels are distributed to replace faulty channels within the polynomial mapping. This architecture is of importance because: (1) its implementation complexity and hardware overhead are approximately the sum of those given by architectures I & II. (2) the fault tolerance of this architecture is greater than the sum of that offered by architectures I & II. This added fault tolerance is based on the fact that, in practice, hard faults are usually separated both in the spatial domain and the temporal domain.

Let us assume that one of the computational channels fails due to a hard fault. The general computational channel will replace the faulty channel and the faulty channel is ignored. Once the replacement channel has filled its pipeline, the architecture has recovered its level of fault tolerance, and the output data will only depend on the non-redundant residues. Multiple hard faults can be corrected successfully without interruption to the data flow if they are separated by a full computational channel pipeline temporal distance and a sufficient spatial distance such that they lie in different general computational channel zones.

4. RESULTS AND COMPARISONS

It is cumbersome task, if not impossible, to produce a fair comparison among architectures with different trade-offs. Nevertheless we have attempted, below, to put forward a comparison based on practical worst case assumptions. Table 1 summarizes the silicon area overhead, the fault tolerant aspects, and the VLSI implementation aspects of the three architectures. The table shows that Architecture III is the most immune to hard and soft faults that are spaced spatially or temporally with a modest silicon area overhead (compared to triple modular redundancy).

The following is a list of the assumptions made to produce Table 1.

Assumptions for architecture I & III:

- 5-bit moduli set.
- Five non redundant residues (e.g. 17, 19, 23, 25, and 27) and two redundant residues (e.g. 29 and 31).
- The residue channel is very large such that the silicon area of the architecture is proportional to the number of residues.

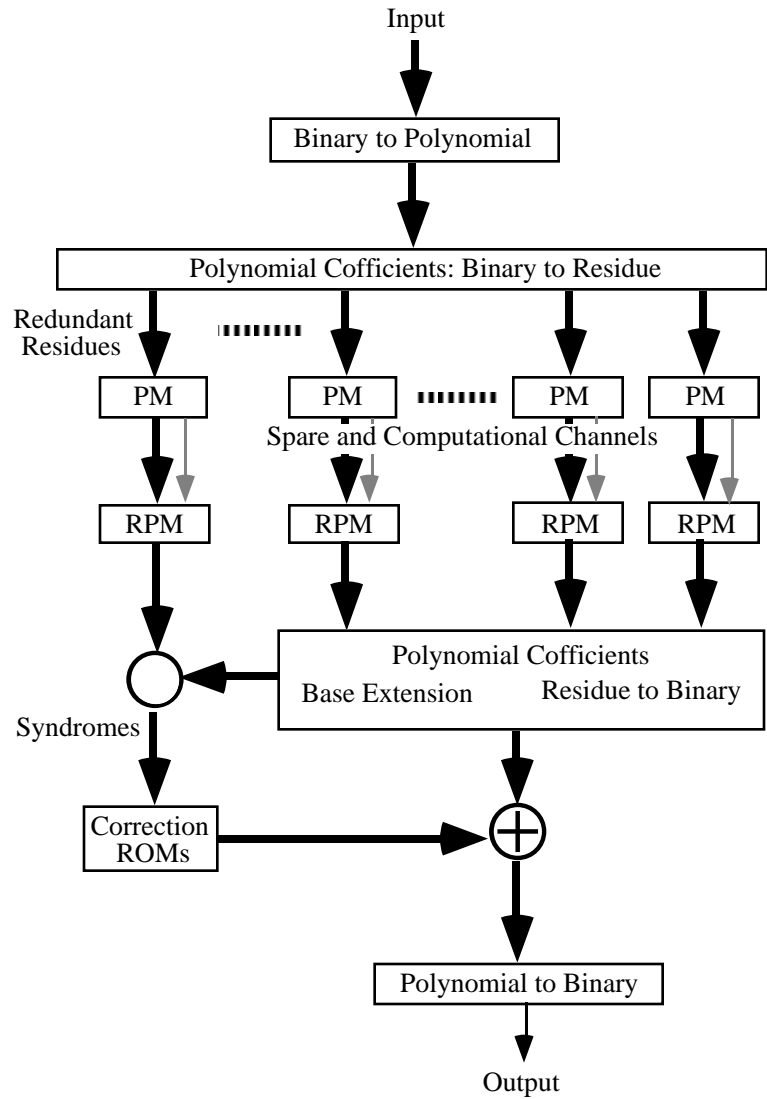


Fig. 7. Architecture III

Assumptions for architecture II & III:

- One general computational channel for every five channels.
- Five data bits per word (per residue digit).
- Two redundant bits per word within the computational channel for error detection.
- The computational channels assumed to be very large compared to the transmission gates and control interconnections.
- The general computational channel is 2.5 times larger than the computational channels if a fixed coefficient multiplication, FCM, is employed (e.g. fixed impulse response FIR filters) and is identical to the computational channels if a general multiplication, GM, is employed (e.g. general convolver).

The channel silicon area is proportional to word size.

	Silicon Area Overhead	Fault Tolerance Aspects	VLSI Implementation Aspects
I	40% % bits utilized=70%	Can detect and correct any single residue without interruption to the data flow. Can detect and correct soft errors provided not more than one faulty residue per pipeline period without interruption to the data flow.	The expected VLSI implementation advantages of using the MRRNS are preserved such as cell replication and local interconnections.
II	90% if employing FCM 60% if employing GM	Can detect and correct one faulty channel in every five channel zones. A full computational channel of pipeline data is lost during fault correction.	If GM is employed, the computational and the spare channels within the residue channel are completely identical and VLSI replication is possible. If FCM is employed, replication of the computational channels within the residue channel is still possible with minor adjustments. The spare channels are substantially different.
III	130% if employing FCM 100% if employing GM	Can detect and correct faulty computational channels without interruption to the data flow. Assumptions: the faulty channels are more than five channels apart spatially within the same residue channel and the faulty channels are separated by more than a full computational channel pipeline data temporally. If the entire residue channel is in error then this architecture has similar properties to Architecture II. If all the general computational channels are utilized then this architecture has similar properties to Architecture I.	All VLSI advantages of the above two architectures are imported to this architecture.

Table 1: Comparisons among the fault tolerant architectures

5. CONCLUSIONS

In this paper we have discussed three architectures for applying fault tolerance to the recently introduced MRRNS polynomial mapping strategy. We usually embed residue number system coding within the polynomial ring structure, and thus fault tolerant techniques developed for RNS can be easily adopted for the new mapping. Redundant residues are implemented to detect and correct faulty residues with no interruption to the data flow. The regularity of the computational channels, generated by the polynomial mapping (operating on the same modulus), enables the use of a single spare general channel that can replace any faulty channel that lies within its spatial zone. Different fault tolerant techniques can be combined in the same architecture to detect and correct various types of faults. The techniques can compensate for each other and can add an extra dimension to the fault tolerant ability of the structure.

6. ACKNOWLEDGMENTS

The authors acknowledge financial assistance from the Natural Sciences and Engineering Research Council of Canada, and the Micronet Network of Centres of Excellence. The authors are also indebted to the Canadian Microelectronics Corporation for laboratory equipment, design software, and fabrication services.

7. REFERENCES

1. M. A. Soderstrand, W. K. Jenkins, G. A. Jullien and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, 1986.
2. D. Mandelbaum, "Error correction in residue arithmetic," *IEEE Trans. Computers*, Vol. 21, pp. 538-545, June 1972.
3. N. M. Wigley and G. A. Jullien, "On modulus replication for residue arithmetic computations of complex inner products," *IEEE Trans. Computers*, Vol. 39, No. 8, pp. 1065-1076, August 1990.
4. N. M. Wigley and G. A. Jullien, "Large dynamic range computations over small finite rings," *IEEE Trans. Computers*, Vol. 43, No. 1, pp. 78-86, January 1994.
5. S. S. Bizzan, G. A. Jullien, N. M. Wigley, W. C. Miller, "Integer mapping architectures for the polynomial ring engine," *Proceedings of the 11th IEEE Int. Symp. on Computer Arithmetic*, Windsor, Ont., pp. 44-51, July 1993.
6. G.A. Jullien, N.M. Wigley, J. Reilly, "VLSI Implementations of Number Theoretic Concepts with Applications in Signal Processing", Invited chapter in *Handbook of Statistics* N.K Bose and C.R. Rao editors, (Invited), CRC Press, 1993.
7. R. J. Cosentino, "Fault tolerance in a systolic arithmetic processor array," *IEEE Trans. Comput.* Vol. 37, No. 7, pp. 886-890, July 1988.
8. W. K. Jenkins, B. A. Schnaufer, and A. J. Mansen, "Combined system-level redundancy and modular arithmetic for fault tolerant digital signal processing," *Proceedings of the 11th IEEE Int. Symp. on Computer Arithmetic*, Windsor, Ont., pp. 28-35, July 1993.
9. A. Baraniecka, G.A. Jullien, "On decoding techniques for residue number system realizations of digital signal processing hardware," *IEEE Trans. Circuits and Systems*, Vol. CAS-25, pp. 935-936, 1978.
10. G. A. Jullien, W.C. Miller, R. Grondin, L. Del Pup, S. Bizzan and D. Zhang, "Dynamic Computational Blocks for Bit-Level Systolic Arrays," *IEEE J. Solid-State Circuits*, Vol. 29, No. 1, pp. 14-22, 1994.
11. P. Zhou, J.C. Czilli, G.A. Jullien, and W.C. Miller, 1994, "Current Input TSPC Latch for High Speed, Complex Switching Trees.", *Proceedings of the International Symposium on Circuits and Systems*, London, June, 1994 (In Print).
12. Jullien, G.A., Miller, W.C., Grondin, R., Wang, Z., Zhang, D., Del Pup, L., and Bizzan, S. "Woodchuck: A Low-Level Synthesizer for Dynamic Pipelined DSP Arithmetic Logic Blocks." *Proceedings of the International Symposium on Circuits and Systems*, 1, pp.176-179, 1992.
13. J. D. Sun, H. Krishna, and K. Y. Lin, "A superfast algorithm for single-error correction in RRNS and hardware implementation," *Journal of VLSI Signal Processing*, 6, pp. 259-269, 1993.
14. M. Taheri, G. A. Jullien, and W. C. Miller, "High speed signal processing using systolic arrays over finite rings," *IEEE Trans. Selected Areas in Comm.*, Vol. 6, No. 3, pp. 504-512, 1988.