

USE OF REDUNDANT BINARY REPRESENTATION FOR FAULT-TOLERANT ARITHMETIC ARRAY PROCESSORS

Vincenzo PIURI, Renato STEFANELLI

Department of Electronics - Politecnico di Milano
piazza L. da Vinci 32 - I20133 Milano - Italy

ABSTRACT

This paper presents a new approach to on-line error detection in arithmetic array processors for very high-computing applications. Use of redundant binary representation allows to design strongly fault-secure architectures with respect to unidirectional stuck-at faults on multiple gate input and/or output lines. On-line fault localization is also considered for fast array reconfiguration.

1. INTRODUCTION

Many applications (ranging from signal and image processing to matrix operations) need high arithmetic computing and special-purpose architectures for parallel computation on input data. Array processors are an example of these structures: they consist of a number (usually very high) of identical processing elements connected by a regular interconnection mesh [1][2]. Use of such systems has become very attractive due to the availability of VLSI and WSI implementation technologies at reasonable costs.

When critical applications are concerned, reliability, availability and continuous operations of these advanced architectures cannot be neglected. Different techniques may be found in literature to detect errors due to faults, to localize the faulty unit, to correct the result of computation and, possibly, to reconfigure the whole system by-passing the faulty elements.

The first approach consists in introducing redundancy [3] [4] [5] [6] in the array processor. In physical redundancy the same computation is performed by different processing elements, while in time redundancy it is performed at different time by the same unit; then, results are compared by a voter to check the error occurrence.

A second approach consists in coding [6] [7] [8] [9] [10] [11] [12]. Input data are substituted by a suitable coded representation and computation is performed on such modified data. Outputs are, then, checked by custom circuits to verify the congruence with the output code: whenever the output does not belong to that code, an error has occurred and a fault has been detected.

Main drawbacks of these general techniques may be the silicon area used by circuits and the computation delay introduced by information redundancy. These characteristics must be carefully considered and evaluated when a very high throughput is required by massive real-time applications and reasonable silicon area and power consumption are mandatory (e.g. radar systems or on-board computations for aerospace applications).

In this paper we propose the use of redundant binary representation [13] for data and a three-rail logic implementation [15] of processing elements to achieve on-line error detection in a class of arithmetic array processors for a wide spectrum of high-computing applications in digital signal and image processing and matrix operations. The redundant binary representation is a signed-digit

number representation in the fixed radix 2 which uses the digit set $\{-1, 0, 1\}$. Such approach allows to design very fast bit-parallel carry-save arithmetic units and a system which is strongly fault secure with respect to unidirectional stuck-at faults on multiple gate input and/or output lines [13] [14] [15]. These important features are achieved at a reasonable cost in terms of silicon area.

After a brief presentation of the structure of processing elements and the overall architecture of array processor considered here, we will discuss about the fault and error models for these systems. Then, we will introduce the redundant binary representation, the arithmetic operations and units, the encoding and decoding circuits and the error checking. Different approaches to on-line error detection are considered and discussed; some attractive architectures are proposed and evaluated by considering the silicon area, the computation time, the throughput and the error latency.

Error detection is the first requirement in designing fault-tolerant architectures: nevertheless, further information and characteristics are required to allow the system survival. In particular, the faulty units must be excluded from computation to assure the correctness of subsequent results. System reconfiguration [3] [5] needs the position of faulty elements in the array to identify the new interconnection for working processing elements. To increase system availability off-line testing after fault detection must be avoided; enhanced array architectures are proposed and evaluated for on-line fault localization, which is concurrently performed with the normal computation.

2. PROCESSING ELEMENT AND ARRAY ARCHITECTURES

An array is composed by a set of simple processing elements (PEs) connected by a proper network [1] [2]: the characteristics of these architectures and the algorithms executed depend on the structure of PEs and on the kind of interconnection network. Here we consider a regular interconnection mesh which allows to design a wide class both of linear and bidimensional arrays for digital signal/image processing and matrix operations (fig. 1). When bidi-

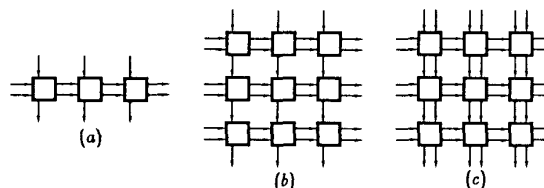


Fig. 1 - Array architectures: linear array(a), horizontal - computing (b) and bidirectional - computing (c) bidimensional arrays

mensional array are concerned, our approach to fault tolerance may be adopted to implement architectures in which the intermediate results of the computation flow only in horizontal direction or in horizontal and vertical directions according with the algorithm to be executed by the whole array.

Details of the interconnection network are not drawn in the previous figure because its characteristics and topology depend on the algorithm implemented and on the reconfiguration capabilities that the designer introduces into the structure. More reconfiguration alternatives are allowed, more complex the network becomes because the number of busses and switches or multiplexers increases. Anyway, the error-detection techniques discussed here are not affected by the adopted solution: in fact only the reconfiguration changes according with the interconnection network.

From the literature it is possible to extract a simple common structure of the PEs for the class of applications considered here (fig. 2) [1] [2].

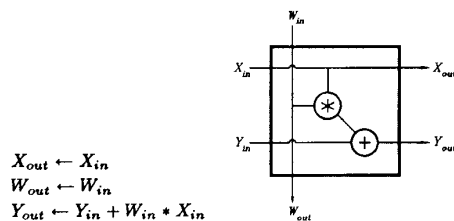


Fig. 2 - Processing element structure

Usually three inputs are supplied to the PE by the surrounding environment: one is the data coming directly into the array from digital devices connected to the array borders, the second one is the result of the computation performed by the adjacent element, the third one is the weight for combining the other input data. The outputs of each PE are the same input data from the digital devices, the weight and the result of the computation executed in the element itself. The basic computation performed in a PE is a linear weighted combination of the data from the digital devices connected to the array inputs and the result of the adjacent PE.

Often, along the data path, some shift registers are introduced to synchronize the data flow according with the nominal algorithm, as e.g. it is required by the evaluation of some implementations of the discrete Fourier transform. Besides, the direction in which the data flow may be changed for some classes of algorithms, without modifying the results presented here.

This simple structure, or other structures derived from it, may be used with a suitable interconnection network to execute a wide set of algorithms for different applications characterized by regularity (i.e. repetitive computations on large set of data) and high computing [1] [2] [13]. For example, in digital signal and image processing it is possible to perform FIR and IIR filtering, 1-D or 2-D convolution, discrete Fourier transform, and other more complex operations. In matrix arithmetic it is possible to compute matrix-vector multiplication, matrix-matrix multiplication, matrix triangularization, solution of triangular linear systems [1] [2] [10].

In this paper we consider bit-parallel carry-save arithmetic units to achieve a very high throughput and a low computation time for massive-computing real-time applications.

3. FAULT AND ERROR MODELS

To evaluate the array architectures and the features of redundant binary representation of data, it is necessary to define the fault model and the related error model which describes the effects of faults onto the computation.

In our studies we have adopted the classic fault models at gate level [10] [16] both for permanent, transient and intermittent faults; in particular, we consider stuck-at, stuck-on, stuck-open and bridging faults.

Related to such fault model, we consider a classic error model at gate level [9] [10]: the physical defects due to faults appear, if they are not masked by the actual input data, as some wrong bits in the result of the considered unit.

All faults that we consider may be reduced to *unidirectional* stuck-at faults on multiple gate output lines [14], i.e. to faults in which wrong bits are of the same type (all '0' or all '1'). Let F the set of faults: any fault $f_j \in F$ sticks multiple gate output lines at the same value (0 or 1). We assume that faults arise one at a time; $\langle f_1, f_2, \dots, f_n \rangle$ is the sequence of faults $f_j \in F$ occurring from the initial observation time till current time.

Since the union $\cup_{j=1}^k f_j$ of unidirectional faults, generally, is not a unidirectional fault, we assume that any fault cannot change the value which has already been stuck by a previous fault. A line which is stuck at 0 or 1 holds that value as long as the source of that error exists.

Before presenting our technique to design array processors with error detection features, some general definitions about self-checking capabilities of combinational circuits must be given [4] [14]. The output of a combinational circuit G when the input i is supplied and the fault f (or a combination of faults) occurs is $G(i, f)$. In our case the function $G(i, f)$ is the linear weighted input combination computed by array processors of the previous section.

During normal functioning, all circuits are fully working. Input data are coded in the chosen input code space I and supplied to the combinational circuit G . Under the fault-free condition $f = \lambda$, the output $G(i, \lambda)$ is correct and belongs to the output code space O . Under faults, non-codewords may be produced by G .

G is *fault secure* with respect to the set of fault F , if the output is either correct or is a non-codeword for all faults in F and all codeword inputs, i.e. $G(i, f) = G(i, \lambda)$ or $G(i, f) \notin O$, $\forall i \in I$ and $\forall f \in F$. G is *self-testing* with respect to F , if there is at least one codeword input that produces a non-codeword output for all faults in F , i.e. there is an $i \in I$ such that $G(i, f) \notin O$, $\forall f \in F$. G is *totally self-checking* with respect to F , if it is fault secure and self-testing with respect to F .

G is *code disjoint*, if it maps codeword inputs into codeword outputs and non-codeword inputs into non-codeword outputs when it is fault free, i.e. $G(i, \lambda) \in O \forall i \in I$ and $G(i, \lambda) \notin O \forall i \notin I$. G is a *totally self-checking checker* with respect to F , if it is totally self-checking with respect to F and code disjoint when it is fault free.

The self-testing property can be generalized to design easily totally self-checking circuits. G is *strongly fault secure* with respect to a fault sequence $\langle f_1, f_2, \dots, f_n \rangle$ with $f_j \in F$, if it is, $\forall i \in I$, either $G(i, \cup_{j=1}^k f_j) = G(i, \lambda)$ or $G(i, \cup_{j=1}^k f_j) \notin O$, where k is the smallest integer for which there is an $i' \in I$ so that $G(i', \cup_{j=1}^k f_j) \neq G(i', \lambda)$, otherwise, if there is no such k , let $k = n$. G is *strongly fault secure* with respect to F , if it is strongly fault secure with respect to all fault sequences whose elements belongs to F .

Every totally self-checking circuit is strongly fault-secure, while a strongly fault-secure circuit is either totally self-checking or can be easily converted into such a circuit [14]. In [14] [15] it was proved that arithmetic circuits can be easily designed to be strongly fault-secure and totally self-checking by using redundant binary representation. In particular, in [14] it was proved that *essentially inverter-free* circuits with *unordered output code space* are strongly fault-secure with respect to unidirectional stuck-at faults on multiple gate input and/or output lines. Essentially inverter-free circuits are circuits which does not contain inverters but in encoders. An output code space is called unordered if no element x of the space covers any other element y , i.e. if x has not one or more '1' where y has.

4. REDUNDANT BINARY REPRESENTATION

The redundant binary representation is a signed-digit number representation [13] in the fixed radix 2 with the digit set $\{-1, 0, 1\}$. A n -digit redundant binary integer x is represented by $[x_{n-1} \dots x_0]$, where $x_i \in \{-1, 0, 1\}$; its value in the decimal representation is $\sum_{i=0}^{n-1} x_i 2^i$. There are several ways to represent an integer in the redundant binary representation: such characteristic allows for designing fast arithmetic units. For simplicity sake, we will consider only integer numbers, but our approach can be extended to real number in the full-fractional representation.

4.1 Arithmetic units

Let's consider the *addition*. When two redundant binary integers x and y must be added, an intermediate selector $t_i \in \{-1, 0\}$ is evaluated for each position i as in fig. 3.a. Then, an intermediate carry $c_i \in \{0, 1\}$ and an intermediate sum $r_i \in \{-1, 0\}$ are computed for each bit i by means of the following relationship (fig. 3.b):

$$2c_i + r_i = x_i + y_i - 2t_i + t_{i-1}$$

Finally, a sum digit $s_i \in \{-1, 0, 1\}$ is obtained by adding r_i to c_{i-1} without propagating any carry, as it is shown in fig. 3.c. Therefore, each sum digit in position i depends only on the input bits in the same position and in the previous positions $(i-1)$ and $(i-2)$. Parallel addition can thus be performed in a constant time independent of the operand length.

		a				c	
		t_i	y_i			s_i	c_{i-1}
			-1 0 1				0 1
x_i	-1	-1	-1 -1	r_i	-1	-1	0
	0	-1	0 0				
	1	-1	0 0		0	0	1

		b	
		c_i, r_i	y_i
		$(t_{i-1} = -1)$	-1 0 1
x_i	-1	0,-1	0,0 1,-1
	0	0,0	0,-1 0,0
	1	1,-1	0,0 1,-1

		b	
		c_i, r_i	y_i
		$(t_{i-1} = 0)$	-1 0 1
x_i	-1	0,0	1,-1 1,0
	0	1,-1	0,0 1,-1
	1	1,0	1,-1 1,0

Fig. 3 - Addition in redundant binary representation

An implementation of such algorithm is shown in [15]; its computation time, i.e. the time required to produce the sum digits, is $\tau_{adder} = 6n\tau$. The time τ is the average switching time of gates in the considered technology; for simplicity sake we assume τ equal to the maximum of the switching time of logic gates.

The silicon area can be evaluated from experimental implementations, but it is strongly dependent on the adopted integration technology and process. Therefore, we prefer to remain at a logic level by considering the number of gates as estimator of the silicon area used by any actual VLSI or WSI implementation. Such measure is independent from the technology, even if it is strictly related to the silicon area. The number of gates A_{adder} for a redundant binary adder is $A_{adder} = 52n$.

Let's consider *multiplication*. The partial products must be generated from the input data bits in the redundant binary representation: n n -bit numbers are produced. Then, the partial products must be added together to achieve the final result. To increase the speed of this step, a binary tree of adder for redundant binary

representation can be used: in this tree all additions at each level are performed in parallel. The generation of partial products and the additions at each level in the binary tree are performed in a constant time independent of n ; since the tree is $\lceil \log_2 n \rceil$ depth, the computation time for adding the partial products is proportional to $\log_2 n$.

In [15] an implementation of this multiplication algorithm is presented. The time required to complete a multiplication is given by $\tau_{multiplier} = (1 + 6\lceil \log_2 n \rceil)\tau$, while the number of gates is $A_{multiplier} = 29n^2 + 26n$.

The arithmetic units presented in [15] are strongly fault secure with respect to unidirectional stuck-at faults on multiple gate input and/or output lines. In fact they are essentially inverter-free and their output code space is unordered [14]. In such implementation the two-rail logic and the three-rail logic have been adopted to achieve this feature. These design techniques constitute an approach at logic level in which two or three mutually exclusive conditions calculated in a circuit are encoded in the 1-out-of-2 or 1-out-of-3 code respectively. In particular, we encode each bit x_i of the redundant binary representation by three bits: we assign 100, 010 or 001 to such coded representation according to the value of x_i (-1, 0 or 1 respectively).

By connecting the previous arithmetic units, we can build the PE structure presented in section 2. The cascade of a multiplier and an adder with strongly fault-secure feature is still a strongly fault-secure circuit since all design constraints [14] are satisfied. In particular, no reconvergent paths are present in the PE since each output bit is computed by a different subset of gates. The computation time of such PE, i.e. the time required to produce the result, is $\tau_{PE} = (7 + 6\lceil \log_2 n \rceil)\tau$, while its number of gates is $A_{PE} = 29n^2 + 78n$.

The structure of PEs considered here allows for building large arrays which are strongly fault secure with respect to kind of faults assumed in section 3. In fact all constraints for strongly fault-secure logic networks [14] are fulfilled, i.e. the array is essentially inverter-free and the output code space of each row is unordered.

4.2 Encoder, decoder and checker

The input *encoder* transforms the binary representation in a corresponding redundant binary representation: it receives n input bits and generates the 1-out-of-2 code for each of them. The encoder circuit consists in n inverters: these are the only inverters in the whole array. The conversion time is $\tau_{encoder} = \tau$, while the number of gates of the encoder is $A_{encoder} = n$.

The result computed by each PE or by each row of PEs is represented as a redundant binary number in the 1-out-of-3 code. The equivalent binary number can be obtained by a *decoder*. Its output digits are encoded in the 1-out-of-2 code. The sequence of the first bits of each output digit is the final result in the usual binary representation (the *nominal output*), while the sequence of the second bits is the one's-complement of the final result itself (the *complemented output*).

A strongly fault-secure implementation of this decoder has been presented in [15]. The three-rail logic has been used to design this essentially inverter-free circuit with unordered output code space, as in the adder tree of multiplier. Three mutually exclusive conditions are computed by this circuit at each group of adjacent positions: the first one for generating a carry, the second one for propagating a carry and the third one for neither generating nor propagating a carry. All of them are coded in the 1-out-of-3 code adopted in arithmetic units. The number of gates of this decoder is $A_{decoder} = 2.5n^2 + 16.5n + 6$, while the computation time for conversion is $\tau_{decoder} = (n^2 - 3n + 8)\tau$. The *error checker* connected to a decoder verify that the nominal and the complemented results (which have been independently computed) are the one's complement each other. The error signal is generated starting from the $2n$ bit pair computed by the corresponding decoder. It is in the

1-out-of-2 code iff each of its input pairs is in the 1-out-of-2 code; thus, it indicates the error occurrence when they have the same value (either 00 or 11), otherwise (either 01 or 10) no error has been detected.

In [15] a totally self-checking checker based upon the two-rail logic has been proposed. This property has been proved in that paper for the cascade of a multiplier, a decoder and a checker: the structure of these circuits is the same assumed here. Nevertheless, the checker is still totally self-checking when a cascade of strongly fault-secure PEs is considered instead of the single multiplier: the proof can be easily achieved by means of the same reasoning of [15]. The number of gates of the checker is $A_{checker} = 6(2n - 1)$, while the error signal is produced in a time given by $\tau_{checker} = 2(\lceil \log_2 n \rceil + 1)\tau$.

5. ON-LINE ERROR DETECTION IN ARITHMETIC ARRAYS

Different architectures can be designed by using redundant binary representation of data to achieve on-line error detection in arithmetic array processors; in particular two approaches can be adopted:

- *local error detection*, in which computation is checked locally in each PE and only tested results are delivered from a PE to the adjacent ones,
- *array-level error detection*, in which the results are tested only at the array borders to decrease the silicon area required by checking circuits.

Let's briefly consider local error detection. Two levels of granularity could be designed:

- the output of each arithmetic unit in the PE could be verified,
- the output of the whole PE is checked.

The first solution allows for the *finest and immediate* error detection and fault localization because possible errors are detected directly at the output of each arithmetic unit; but a checker is required for each unit in the PE. Therefore, this solution is not feasible due to the too large silicon area for so many checkers: checkers' area is about double of the area for checked circuits. On the other hand, the identification of the fault arithmetic unit inside a PE is not useful because reconfiguration can connect only PEs to achieve a working array, not arithmetic units of different PEs. The second solution for local error detection requires only one checker at the output of each PE; nevertheless, it is too expensive as silicon area is concerned because checker is wide about as checked circuits. Thus, error detection at local level must be *a priori* discarded, even if it assures the lowest error latency.

Error detection at array level by using redundant binary representation of data is more attractive since it allows to design fast array processors (arithmetic units are carry-free) with fault-tolerant capabilities at a reasonable cost in terms of silicon area (checkers are appended only at array borders).

The architectures implementing this approach for linear and bidimensional arrays are shown in fig. 4. The input data X of the array are coded in the redundant binary representation by encoders at the leftmost border; the input partial sum Y of the leftmost PEs is set to 0 in the redundant binary representation. The input weights W of linear combination computed by the array are coded by encoder on the top side of the array itself. When properly synchronized, the arithmetic units inside each PE produce the final output result of the array by working on the redundant binary representation of data. The decoders connected at the rightmost border of the array transform the coded result back to the usual binary representation. These circuits output also the complement

of the nominal result. The checkers verify the occurrence of a fault in the whole system by testing the nominal output in the binary representation and its complement. Whenever these two data are not one the complement of the other, an error is detected. The fault related to such error has occurred in the row of PEs corresponding to the active check signal.

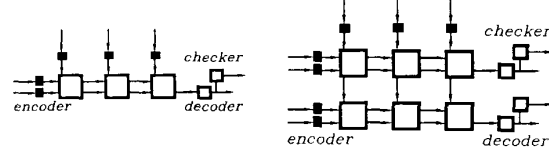


Fig. 4 - Architectures for array-level error detection

The silicon area required by these architectures can be estimated from the area of each simple component: PE, encoder, decoder, checker. Also in this evaluation we assume the number of gates as an estimator of the silicon area used by an actual VLSI or WSI implementation. Linear and bidimensional arrays have the following number of gates A_G respectively:

$$A_{G_l} = (29n + 79)nN + (5n + 59)\frac{n}{2}$$

$$A_{G_b} = (29n + 78)nMN + (5n + 59)\frac{n}{2}M + nN$$

where n is the number of bits of inputs X and W , M is the number of rows and N is the number of columns.

To compare the performance of our approach we have considered an experimental implementation based upon modular redundancy. The array architecture for the usual binary representation without on-line error detection capability has been doubled: fast arithmetic units (including carry look-ahead adders) have been adopted. Error detection consists in comparing the outputs of each array. In fig. 5 we give the percentage ratio between the number of gates for our approach and that for modular redundancy. This ratio is drawn versus the number n of input data bits and the number of PEs in the array: for simplicity sake, we consider $N \times N$ arrays.

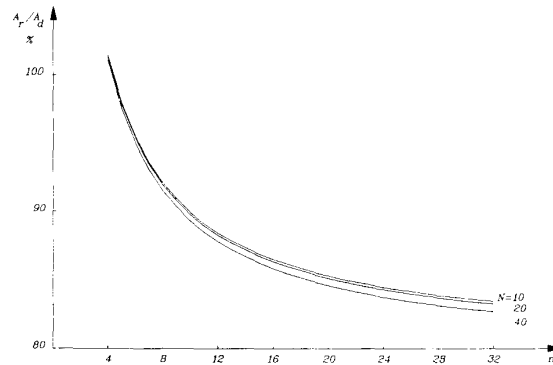


Fig. 5 - Percentage ratio of the numbers of gates

The computation time τ_G is the time between application of an input X at the leftmost border and delivery of corresponding final result at the rightmost border. As already said, by using redundant binary representation it is possible to design carry-free arithmetic units for the PEs. In this way the computation time is

greatly decreased: in fact the only circuit that cannot avoid carries is the decoder at array border. In our architectures for linear and bidimensional arrays it is given by:

$$\tau_C = [(7 + 6\lceil \log_2 n \rceil)N + (n^2 - 3n + 9)] \tau$$

where τ is the average switching time of gates in the considered implementation.

The *throughput* T of array processors is the number of results produced in one time unit. In the architectures proposed here, it is:

$$T = \min \left(\frac{1}{\tau_{PE}}, \frac{1}{\tau_{decoder} + \tau_{checker}} \right) =$$

$$\min \left(\frac{1}{7 + 6\lceil \log_2 n \rceil}, \frac{1}{n^2 - 3n + 10 + 2\lceil \log_2 n \rceil} \right)$$

where the first function is smaller for $n \leq 4$, the second one for $n \geq 5$.

The *error latency* τ_L is obviously greater in any architecture for on-line error detection at array level than in local error detection since the computation is checked only at array borders. In the worst case it is equal to the computation time of the array plus the checking time (i.e. the time required by checker to verify the array result):

$$\tau_L = [(7 + 6\lceil \log_2 n \rceil)N + (n^2 - 3n + 102\lceil \log_2 n \rceil)] \tau$$

6. CONCURRENT FAULT LOCALIZATION

The array architectures presented in the previous section are able to identify only the row containing the faulty PE, not such PE. In fact the output of each PE is not checked in error detection at array level, but only the results produced by the whole row are tested. To localize completely the faulty PE more information are required: in particular, the column containing such PE must be identified. The exact localization of the faulty PE is in fact mandatory to reconfigure the whole system.

Off-line testing could be adopted to obtain these information: during testing the normal activities of the array are suspended until reconfiguration has been completed. Nevertheless, concurrent fault localization becomes attractive whenever we need a fast identification of the faulty PE to achieve a high availability of the system. This feature can be added to the array architecture by inserting additional circuits: they will test intermediate results but preserving the structural regularity and without greatly increasing the overall silicon area.

First, let's consider linear arrays. In this case we must connect a decoder and a checker at the output of each PE, as we have done at the rightmost border of the array: this solution coincides with the architecture for local error detection. Therefore, it is too expensive in terms of silicon area since the additional circuits are larger than the PEs itself. A better solution consists in doubling each PE and in checking the outputs of each pair of PEs.

Attractive performances can be achieved in bidimensional array processors. The architecture able to localize faulty PE is shown in fig. 6. Instead of checking the output of all PEs belonging to a column like in local error detection, such outputs are added together to generate a redundant output Y' . This computation flows in the array orthogonally to the main computation.

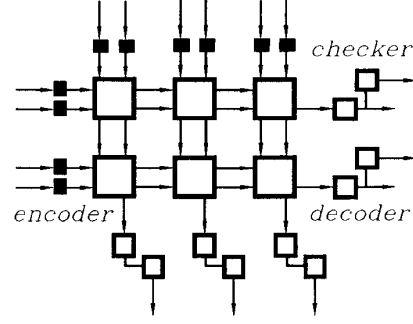


Fig. 6 - Fault localization in bidimensional arrays

To compute these check functions the basic PE structure must be modified by inserting another adder, as it is shown in fig. 7. The inputs of such adder are the output of the basic PE and the partial sum computed by the corresponding adder of the PE in the above row of the same column.

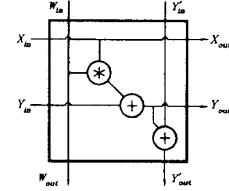


Fig. 7 - PE for fault localization

The fault and error models discussed in section 3 are still valid. On the other hand, the function Y' computed by each column does not depend on the corresponding function of the other columns, like the output Y of each row does not depend on the output Y of the other rows. In other words, there are no reconvergent data paths in the array. This allows for considering such structure as internal fanout-free [13] [14] with respect to the function computed by each row or column, even if the adder directly connected to the multiplier in each PE is involved in the evaluation of the functions of its row and column. Thus, like for the output Y , the redundant binary representation can be effectively adopted to check the output Y' of each column.

Therefore, the additional outputs Y' give information about the state of PEs belonging to the corresponding column; each of these outputs is tested in the same way of rows by connecting a decoder and a checker to the bottom side of the array. Checkers at rightmost border identify the row containing the faulty PE, while checkers at the bottom identify its column.

This architecture has the same computation time, throughput and error latency of the array for on-line error detection. Nevertheless, it requires a greater area because of the further adder which compute the output Y' and the additional decoders and checkers. The number of gates A_{G_b} is in fact:

$$A_{G_b} = (29n + 130)nMN + (5n + 59)\frac{n}{2}(M + N)$$

7. CONCLUDING REMARKS

On-line error detection in array processors for mission-critical applications constitutes one of the most important problems in designing fault-tolerant architectures. A new approach based upon redundant binary representation has been proposed and evaluated for a class of arrays which can be adopted in a wide spectrum of applications in digital signal and image processing and in matrix operations.

A careful implementation of this approach by means of three-rail logic has allowed to design a strongly fault secure architecture with respect to unidirectional stuck-at faults on input and/or output lines. On the other hand, high throughput has been preserved because all arithmetic units can be easily designed to be carry-save.

Finally, concurrent fault localization for fast array reconfiguration has been considered: enhanced architectures have been proposed to identify the position of faulty elements in the regular array concurrently with the nominal computation.

REFERENCES

- [1] H.T. Kung, "Why systolic architectures?", *IEEE Computer Magazine* (Jan. 1982)
- [2] S.Y. Kung, S.C. Lo, S.N. Jean, J.N. Kwang, "Wavefront array processors - Concept to implementation", *IEEE Computer Magazine* (July 1987)
- [3] R. Negrini, M. Sami, R. Stefanelli, "Fault tolerance techniques for array structures used in supercomputing", *IEEE Computer Magazine* (Feb. 1986)
- [4] D. Siewiorek and R. Swartz, *The theory and practice of reliable system design*, Digital Press (1982)
- [5] M.G. Sami, R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays", *Proc. IEEE* (May 1986)
- [6] J.A. Abraham, P. Banerjee, C. Chen, W.K. Fuchs, S. Kuo, A.L.N. Reddy, "Fault-tolerance techniques for systolic arrays", *IEEE Computer Magazine* (July 1987)
- [7] A. Avizienis, "Arithmetic error codes: cost and effectiveness studies for application in digital system design", *IEEE Trans. on Comp.* (Nov. 1971)
- [8] W.W. Peterson, E.J. Weldon, *Error-correcting codes*, ed. MIT Press (1972)
- [9] J.F. Wakerly, *Error detecting codes, self-checking circuits and applications*, ed. Elsevier-North Holland, New York (1978)
- [10] J-Y. Jou, J.A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing", *IEEE Proc.* (May 1986)
- [11] N.S. Szabo e R.I. Tanaka, "Residue Arithmetic and Its Application to Computer Technology", McGraw-Hill, New York, 1967
- [12] J. Massey and G. Garcia, "Error correcting codes in computer arithmetic", *Advances in information system science* (1972)
- [13] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic", *IRE Trans. Electron. Comput.*, vol. EC-10, (Sept. 1961)
- [14] J.E. Smith, G. Metze, "Strongly fault secure logic networks", *IEEE Trans. on Computer*, vol. C-27, (June 1978)
- [15] N. Takagi, S. Yajima, "On-line error-detectable high-speed multiplier using redundant binary representation and three-rail logic" *IEEE Trans. on Computer*, vol. C-36, (Nov. 1987)
- [16] R. Wadsack, "Fault coverage in digital integrated circuits", *The Bell Syst. Tech. Jour.*, vol. 57 (June 78)