FAULT-TOLERANT SYSTOLIC ARRAYS: AN APPROACH BASED UPON RESIDUE ARITHMETIC

Vincenzo PIURI

Department of Electronics, Politecnico di Milano
Piazza L. da Vinci 32, I20133 Milano, Italy, tel.(39)(2)23993528

ABSTRACT

Much attention has been recently given to VLSI and WSI processing arrays: systolic arrays are often adopted to execute a wide class of algorithms, e.g for matrix arithmetic or signal and image processing.

In this paper a fault-tolerant architecture is proposed to allow reliable computation of systolic arrays by using physical redundancy and residue number coding. Such architecture supplies also information for fast reconfiguration.

1. INTRODUCTION

Array architectures consist of a number (usually very high) of identical processing elements connected by a regular interconnection grid [11,13]; these characteristics make arrays well suited for VLSI and WSI integration. They are extensively studied for a wide spectrum of high-computing applications ranging from signal and image processing to matrix operations to relational data base support.

The continuing growth of interest in such computing architectures, their great complexity, their VLSI or WSI integration and their use in dangerous or critical applications give more importance to the problem of reliability of their computation. Since each processing element in such a system executes a part of the whole computation, the failure of one of them may lead to uncorrect results. The systolic array has to work correctly even if some processing elements are faulty [4,6,7]; besides, it has to supply enough information to allow an easy, fast reconfiguration of the whole array when a permanent fault is detected in a processing element [15].

High reliability and availability of arrays may be achieved by means of fault-tolerance techniques. Two main methodologies may be found in literature: the first approach is redundancy (either physical or time redundancy) [17], the second one is coding [3,14,22]. In physical redundancy additional circuits are added to the basic architecture to allow error detection and correction, while in time redundancy the computation is repeated more times and results are compared. These solutions present two drawbacks: silicon area for additional circuits may be great and delays required by redundant computation may became unacceptable. On the other hand, by properly coding input and output data it is possible to detect and correct errors due to faults; but, also in this case, the greater number of bits introduces execution delays and additional area occupation.

Whenever a fault is detected, it may be masked by correcting the results or it may be ignored and the array may be reconfigured to exclude the faulty processing element [15]. Permanent faults may be usually treated by using off-line testing and reconfiguration; while, transient faults and critical applications require techniques which are able to assure the correct functioning of the whole system and to supply data for the identification of faulty elements allowing a fast reconfiguration.

In this paper a mixed approach to fault tolerance for processing arrays is presented: physical redundancy due to replication of computational elements is associated to the adoption of a residue number code [19,20] for input and output data. Redundancy allows to detect and correct errors and to generate data for reconfiguration, while residue data representation bounds the increase of silicon area and the computation delay. In this way it is possible to exploiting the properties of modular redundancy without replicating the whole processing element. This approach to fault tolerance was firstly proposed in [16,18] for multipliers; here, it is shown how it may be adapted to processing arrays. Error detecting and correcting architectures are proposed for linear and bidimensional systolic arrays: additional circuits are also introduced to allow a fast identification of faulty elements and, then, an easy reconfiguration. The definition of the error models and the choice of the bases of the residue number system are discussed: constraints upon them and mathematical properties are presented to assure the correction of outputs.

Fault-tolerance may be locally introduced in arrays: for example, it is possible to design fault-tolerant processing elements or it is possible to repeat computation on different neighboring elements and compare results. This approach requires a wide silicon area and high delays; moreover, unregularity may generate difficulties in circuit design.

A more profitable approach is the definition of globally fault-tolerant arrays: in this case the whole array is replicated and results at the edges of the array are examined to detect errors due to faults. In particular, the basic architecture that is here considered consists of a usual processing array, in which operations among integers are performed in binary representation without truncation or rounding, and of a residue network able to correct errors and to outline the faulty elements. In the residue network a set of processing arrays perform the same operations of the usual binary array, but using residue representations of data: the results of the binary array are compared with the results of the residue arrays to detect errors and, then, a correction newtwork allows to correct the results of the binary array. The residue network may identify also the faulty processing elements for reconfiguration.

Several architectures may be designed by using this approach to achieve different degrees of fault tolerance; for example, the following policies with different complexity and cost, may be adopted:

- error detection followed by reconfiguration;
- error correction without reconfiguration (one or two faults may be tolerated in the whole array);
- correction of the first error, detection of the second error and, then, reconfiguration;
- error correction and reconfiguration after the first or the second fault.

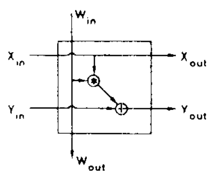
The main features of proposed architectures are: the strong regularity of the whole structure (allowing an easy VLSI or WSI implementation), the bounded increase of computation time and of silicon area due to redundancy, the minimization of hard-core (i.e. the unprotected area).

2. ARCHITECTURE OF PROCESSING ELEMENTS

An array is composed by a set of simple processing elements connected by a proper network [11,13]: thus, the characteristics of these architectures depend on the properties of processing elements and on the kind of interconnection network. Before presenting the overall architecture and its properties of self-checking and fault-tolerance, it is necessary to discuss the structure of each processing element to introduce its features, which allow to achieve the desired behavior of the whole array.

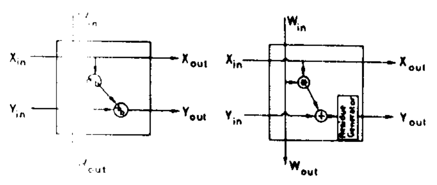
The fault-tolerant processor array is composed by processing elements of two kinds: the elements of the first kind operate in binary representation and belong to the binary array, while the elements of the second kind belong to a residue arrays of residue network and use a representation of data in a suited Residue Number System [19,20]. Here, the structure of such processing elements is briefly described.

From literature it is possible to extract a simple common structure of the processing elements belonging to the binary array (see fig. 1) [11,13]. Usually three inputs are supplied by the environment external at the element: one is the data coming from the host computer into the binary array, 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 element are the input from the host, the weight and the result of the computation executed in the element itself. Computation in a processing element is a linear weighted combination of the data from the host computer and the result of the adjacent element: data from the host is multiplied by proper weight and, then, added to the result of the adjacent element. The direction in which data flow may be changed to implement some classes of algorithms, without modifying the results of this paper.



$X_{out} \leftarrow X_{in}$
$W_{out} \leftarrow W_{in}$
$Y_{out} \leftarrow Y_{in} + W_{in} * X_{in}$

Fig. 1 - Binary processing element



) b)

$X_{out} \leftarrow X_{in}$
$W_{out} \leftarrow W_{in}$
$Y_{out} \leftarrow R[Y_{in} + R[W_{in} * X_{in}]_b]_b$

Fig. 2 - Residue processing element

By using this structure it is possible 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 [11]. In digital signal and image processing this processing element may be adopted to implement, via a suited interconnection network, FIR or IIR filtering, 1-D or 2-D convolution, discret Fourier Transform, and other operations [9]. Some problems in matrix arithmetic may be solved with such processing elements, e.g. matrix-vector multiplication, matrix-matrix multiplication, matrix triangularization, solution of triangular linear systems [10].

The processing elements of residue arrays are similar to the processing elements of the binary array; they are structurally equivalent to binary elements, but they execute all the operations in the adopted Residue Number System [19,20]. Therefore, the binary multiplier and the binary adder of fig. 1 are substituted by devices working in residue representation (fig. 2.a).

2.1. Implementation of the residue processing elements

Many techniques were proposed to implement residue arithmetic devices [19,20]; for example, the simplest circuit is a ROM or PLA, whose inputs are the operands and the output is the result of the operation. Nevertheless, these structures are highly area consuming. Very compact solutions may be designed for particular bases, such as low-cost bases [1]. For other bases special-purpose devices may be considered. A general solution for a modulo-b adder consists in adding to the inputs the constant ($2**\log_2(b-1)-b$): if a carry is generated, the result has to be corrected by adding b to it and discarding the new carry; this structure is compact and fast.

An alternative solution is shown in fig. 2.b; multiplication and sum are performed by usual binary devices: the final result is the input of a correcting circuit whose output is the residue representation of its input.

Anyway, the area used by the residue processing element is smaller than the area of the binary element, because the number of bits of data is less. On the other hand, the additional delay in computation, due to modulo correction, is shorter than the reduction of computation time due to the minor number of bits: therefore the result of residue element will be approximately available at the same time of the result of binary processing element.

3. ERROR MODELS OF PROCESSING ELEMENTS

Before discussing the detection and correction mechanisms in the array architecture, it is necessary to define the error model, i.e. the influence of faults on computation, for processing elements.

Let us assume that a fault (stuck-at-0 or stuck-at-1) in a gate of a processing element causes in the result an additive error of the type "power-of-2" [16,18]: i.e. for any possible fault, the correct result is derived from the output of the processing element algebraically added to the amount of error.

The number of bits affected by a fault depends on the considered device; in this paper it is always assumed that a fault can only produce a single error. If P is the correct result and P' the wrong result, it is $P'=P\pm K_i 2^i$, where $K_i \in \{0,1\}$. The exponent "i" is related to the position of the fault device, while the sign of the error depends on the input data. The coefficient K_i depends on the input data; a fault in a processing element may not appear in the output of that element: error detection depends also on the input data that can mask the error. In this case the fault remains latent until it will be detected.

An error may be identified by examining the difference between the correct result and the wrong one. When a set of bases $\{b_k | k=1..n\}$ is considered, the n-tuple of the residues of such difference is called the syndrome $s_{\pm i}$ of an error:

$$s_{\pm i}=\{\sigma^k_{\pm i} | \sigma^k_{\pm i}=R[P'-P]_{b_k}=R[\pm K_i 2^i]_{b_k} , k=1..n\}.$$

Since the error in some arithmetic units may depend on the input data, also the syndrome may be related to the input data [18]. Usually, circuits with one, two or three different syndromes for each error are considered; these errors will be referred as single-syndrome errors, double-syndrome errors and triple syndrome errors respectively.

If two faults occur, double errors may be introduced in the result of the processing element. The errors are still additive as for the single error case, i.e. they are always sums of powers of 2 [18]. The global error is

$$P'-P = \pm K_i 2^i \pm K_j 2^j .$$

In this case the syndrome $s_{\pm i,\pm j}$ is given by:

$$s_{\pm i,\pm j}=\{\sigma^k_{\pm i,\pm j} | \sigma^k_{\pm i,\pm j}=R[\pm K_i 2^i \pm K_j 2^j]_{b_k} , k=1..n\}.$$

Note that, for particular values of the input data, the syndrome due to two faults may be equal to a syndrome due to a single fault or may not detect any error (power of 2 with same magnitude and opposite sign). The bases of the Residue Number System have to be chosen so that the previous events are detected.

If more than two faults occur, there are multiple errors in the output of the processing element. The error is a linear combination of powers of 2 with coefficients equal to ±1; also in this case masking or coincidence with syndromes due to minor number of faults are possible.

When two or more errors are considered, they may be partitioned in two classes [18]:

- <u>consecutive permanent errors</u> (or, briefly, consecutive errors), due to permanent faults which takes place in different times. The syndrome produced by the first failure may be stored in a register: correction of a second permanent failure may be performed more easily.
- <u>"any kind" errors</u>, i.e. due to permanent, transient or intermittent faults occurring either in the same time or in subsequent times.

Generally, it is possible to assume that errors are consecutive since the probability of contemporaneous occurrence of more faults is very low: this fact allows a very efficient error detection and correction.

Many arithmetic units satisfy the above error models, e.g. adders (SN7483, Lai and Muroga [12]), multipliers (Dadda [5], Wallace [21], Jayshree and Basu [8]) and more complex networks of full-adders. The unique constraints are that data flow must pass only once through each full-adder and all bits has to be processed in a parallel fashion, so that a fault in one of them produce an error equal to a single power of 2.

Every processing element described in the previous section satisfies these error models, because it is a full-adders network in which a data word passes once through each of full-adder. The models are correct also if buffers or latches are introduced in the processing elements: in fact a stuck-at fault of a bit produce an error of the type power of 2 at the output of such devices. Note that all the assumptions are still satisfied when the processing elements are connected in an array: therefore, a fault in a processing element produces an error equal to a power of 2 at the edges of the array (if input data do not mask it) and error models discussed above hold.

4. ARCHITECTURES FOR FAULT-TOLERANT ARRAYS

The processing elements previously described may be connected by a interconnection network to obtain an array, i.e. a regular architecture for high-computation applications [11,13]. When a fault occurs in a processing element, the computation may produce erroneous outputs. In this section an approach is proposed to achieve fault-tolerant arrays by means of modular redundancy and the properties of Residue Number Systems: these architectures are able to detect and, possible, correct the errors due to a bounded number of faults when given constraints are satisfied. To allow the survival of the array when more faults occur, the host computer, which drives its computation, must stop the array and reconfigure it. Reconfiguration [7,15] consists in reconnecting the processing elements to exclude faulty ones by sending suited signals to the interconnection network. Then, the computation may be restarted. The capability to output the correct results, even if faults occurred (but less than the upper bound), reduces the frequency of the reconfigurations and, therefore, increases the availability of the system.

To decrease the time required by reconfiguration and to assure a greater availability of the system, it is necessary that the architecture itself is self-checking, i.e. it has to identify the faulty element. In this way the host computer acquires the position of the faulty element, selects the new interconnection of processing elements and generates the corresponding signals for the interconnection network. Otherwise, it has also to test the processing elements to identify the fault: but this operation may require too much time.

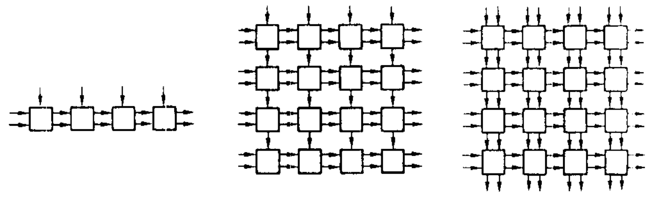

Linear array
Horizontal computing bidimensional array
Bidirectional computing bidimensional array

Fig. 3 - Different topologies for arrays

Different types of arrays are here considered: linear arrays and bidimensional array (fig. 3) [11,13]. Two structures are identified for bidimensional arrays, accordingly with the direction of transmission of the results. In horizontal-computing bidimensional arrays the results of computation are transmitted in horizontal direction, while in vertical direction only input data are passed. In bidirectional-computing bidimensional arrays the results of computation is passed horizontally and vertically to the adjacent processing elements.

Note that interconnection networks are not drawn in fig. 3 and in the following ones, since they depend on the reconfiguration capabilities introduced by the designer. Basic fault-tolerant architectures for linear arrays are presented: their behavior and their performance are discussed. Then, bidimensional arrays are considered; the solutions considered for linear arrays are extended to horizontally-computing bidimensional structures and an effective solution is proposed to identify the faulty processing element in the array. Characteristics and properties of fault-tolerant bidirectional-computing bidimensional arrays are derived immediately from horizontal-computing architectures: in fact in bidimensional-computing arrays also data passed vertically are the result of a computation performed in the processing elements, which are a simple extension of the processing elements presented above. Details on all these architectures will be discussed in the next section.

4.1. Linear arrays

Here, the architectures for fault-tolerant linear arrays are presented. The most simple solution to add fault-tolerance capabilities to the linear array of binary processing element is shown in fig. 4.
The binary array executes a computation receiving input data (integers in binary representation) from the host computer, which controls all activities. The same data are converted in a Residue Number System: each set of binary-to-residue converters generates the inputs for the corresponding residue array. Each residue array executes the same computation of the binary array, but all operations are in modulo: if no fault occurs, its output is the residue of the output of the binary array in the corresponding base.

The output of binary array is, then, converted in its residue representation by others binary-to-residue converters. The occurrence of errors may be detected by examining the output of the syndrome generator, i.e. the differences between each residue of the output of the binary array and the output of corresponding residue array. If all these differences are zero, no error occurs or, better, no error is detected: in this case the multiplexer controller in the error correction network selects the output of the binary array as the correct output.

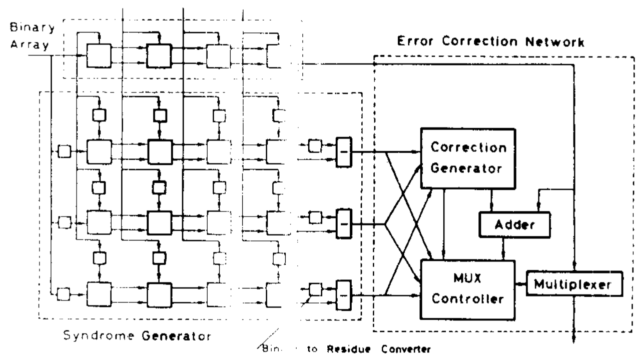


Fig. 4 - Fault-tolerant linear array

When at least one difference is not zero, an error is detected: its syndrome is the input of the correction generator in the error correction network. The correction generator produces the correction that has to be added to the output of the binary array to obtain the correct result of the computation. Then, the multiplexer controller selects the output of the adder as the final output of the fault-tolerant array. Note that in fig. 4 a very simple error correcting network is shown: it may be used only to correct single errors. Error correcting networks for more complex error models will be presented in the following section.

This architecture is able to tolerate a given number of faults according with the syndrome generator and the error correction network adopted. But no information is supplied about the position of the faulty processing elements in the array. As already said, such data are very important to allow a fast reconfiguration and to assure a high availability of the system. To overcome such problem, additional circuits are required: a possible solution is proposed in fig. 5. In this case the error correcting network is able to manage only single errors.

In this architecture the intermediate results of the computation, i.e. the outputs of each working processing element, are tested. The outputs of the binary elements are converted in the residue representation and the differences between the residue of the output of every binary element and the output of the corresponding residue element are computed. If all these differences related to processing elements in a given position in the arrays are zero, no error is detected: otherwise there is a faulty processing element in that position. Reconfiguration algorithm takes advantage of this information and excludes the faulty processing element.

Both these architectures minimize the hard-core, i.e. the unprotected area; in fact, it is reduced only to the last stage composed by the error correction network, while the syndrome generator itself is

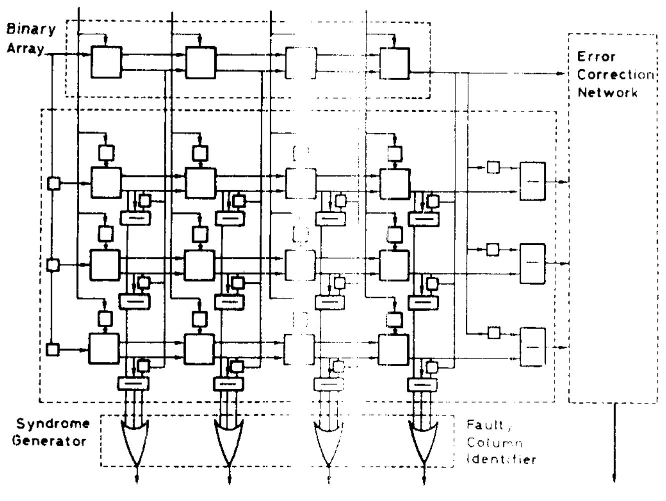


Fig. 5 - Linear arr[illegible] wi[illegible]h fault localization

protected from faults. The additional silicon area used to achieve a f[illegible]ult-tolerant system may be smaller than the are[illegible] re[illegible]uired by the classical modular redundancy if [illegible] careful hardware design is performed: moreover, the regularity of the architecture is well [illegible]uit[illegible]d for a VLSI or WSI implementation.

Let us consider how re[illegible]undancy influences the execution speed of the [illegible]ole architectures. Note that conversion delay may [illegible]e m[illegible]nimized by means of a careful hardware design [2]. When no fault arises in the arrays, the del[illegible] i[illegible]troduced by additional circuits is very low: in fact it is due to the residue conversion of t[illegible] result of binary array, the multiplexer controller and the multiplexer itself. When faults occur, the [illegible]ela[illegible] is due also to the error correction network: suc[illegible] ad[illegible]itional delay depends on the type of error mode[illegible] a[illegible]opted for the processing elements. Anyway, the ri[illegible] result is immediately available after having [illegible]dd[illegible] the correct[illegible]on; this is an advantage of the m[illegible]hod here proposed on techniques using Redund[illegible] [illegible]t [illegible]esidue Number Systems: in such approaches a time-[illegible]ons[illegible]ming conversion from the residue representation [illegible] t[illegible]e binary one is in fact required [2,20].

4.2. Bidimensional arr[illegible]s

To achieve higher comp[illegible]at[illegible]n speed, the processing elements may be connect[illegible]d [illegible] a bidimensional regular grid: here, some ar[illegible]hi[illegible]ctures achieving fault tolerance are propose[illegible] f[illegible] this configuration. A detailed description [illegible] [illegible]chniques are given for horizontal-computing b[illegible]di[illegible]nsional arrays; similar architectures and tech[illegible]qu[illegible] may be easily derived for bidirectional-compu[illegible]in[illegible] bidimensional arrays, in which also data flowin[illegible] v[illegible]tically in the structure are the result of the [illegible]om[illegible]utation performed in the processing elements.

The simplest architectu[illegible]e is shown in fig. 6; it is very similar to the s[illegible]lution presented for linear arrays in fig. 4. Th[illegible] b[illegible]nary array executes the algorithm assigned by [illegible]e [illegible]ost computer, while the residue arrays operate in [illegible] Residue Number System. The outputs at the e[illegible]e [illegible]f the binary array are converted in the resid[illegible] r[illegible]resenta[illegible]ion [illegible]nd compared with the outputs of [illegible]he residue arr[illegible]ys: if the resulting syndrome is zero, no error is detected; otherwise, the output of the binary array is corrected by the error correction network.

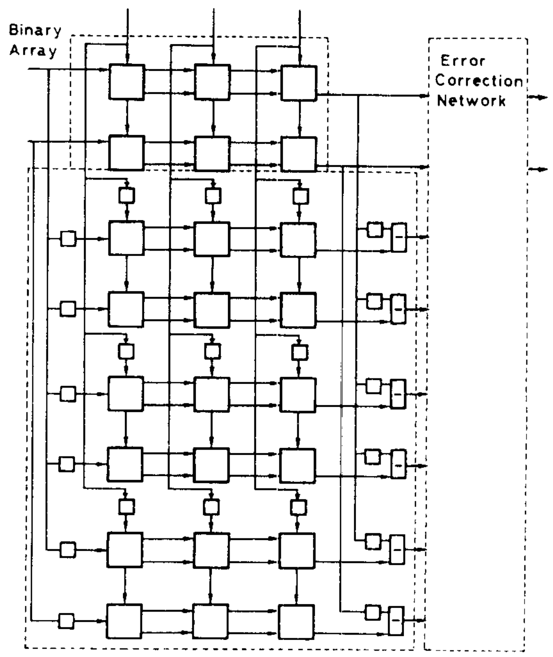


Fig. 6 - Fault-tolerant bidimensional array

Such network may also signal the row of the processor matrix containing the faulty processing element, in fact the output of each row is separately verified. But, the faulty element may not be identified because the whole output of the row is tested. To overcome this problem, an architecture like that presented in fig. 5 might be adopted. Nevertheless, the number of wires and the number of binary-to-residue converters are too high: therefore this solution is not suitable for VLSI or WSI integration.

To reduce the numbers of wires and converters the intermediate outputs might be latched and passed along the columns of the arrays: they should be compared only at the low edge of the arrays. In this case two clocks and register files (one for each column) are required: moreover, also this solution requires too much silicon area and it is not suitable for intergration.

An attractive architecture for fault-tolerant horizontal-computing bidimensional arrays based on residue arithmetic is shown in fig. 7: intermediate outputs of each column of processing elements are added together to generate the output Y'_{out}.

To compute such sums the basic processing elements have to be modified, as it is shown in fig. 8.a; in each processing element an adder is inserted: its inputs are the output of the basic processing element considered until now and the partial sum produced by the adder of the element in the previous row and in the same column.

Note that the sums Y'_{out} computed by the additional adders are obtained with an algorithm which is similar to the target algorithm executed by the array, since they are sum of products of the inputs X and W. Besides, all conditions on the data flow

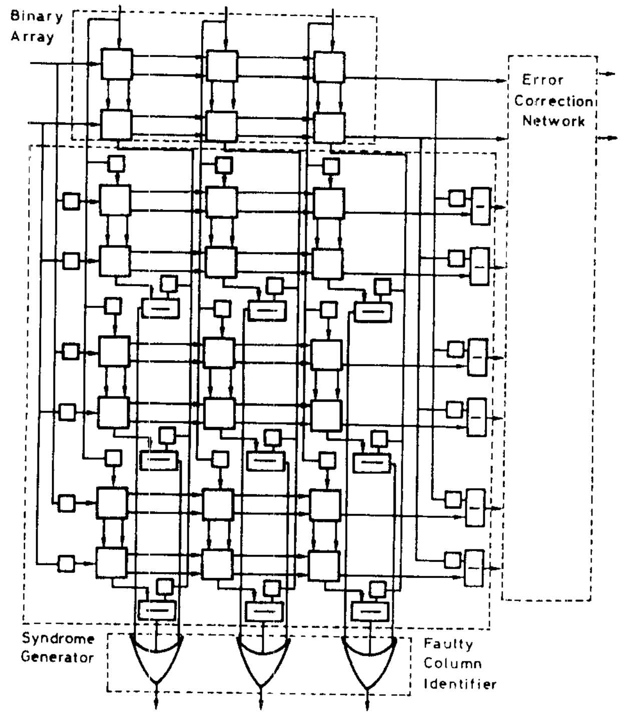
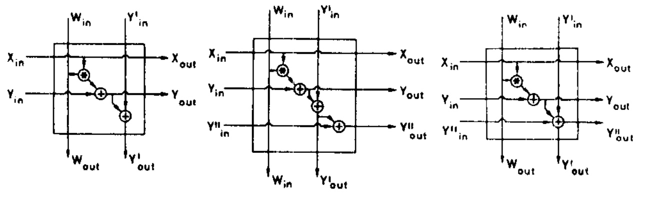


Fig. 7 - Bidimensional array with fault localization



a) b) c)

Fig. 8 - Processing element for fault localization

through the circuits are still satisfied: therefore, the error models are valid and, thus, these sums give information about the state of the processing elements of the corresponding column.

The additional sums produced by the binary array are converted in their residue representation; the set of the differences between the residue representation of one of those sums and the sums produced by the corresponding column in the residue arrays are similar to the syndrome generated by the syndrome generator of the architecture in fig. 6. The syndrome obtained at the right edge of the array allows to detect an error and to identify the row of the faulty processing element; in a similar way, the additional syndrome generated at the bottom of the array allows to identify the column of the faulty processing element. These two information are enough to identify uniquely the faulty element for a subsequent reconfiguration of the whole architecture.

It may happen that the input data of the array mask a fault in the row test, but not in the column test (or vice versa): in this case the fault has to be considered latent, even if the host should store the position of the column (row) containing a faulty element to identify it as soon as possible.

On the other hand, the additional adder of processing element in fig. 9 protects the multiplier and the first adder, but does not allow to detect errors due faults in the additional adder itself. If such circuit fails, only the column containing the faulty processing element may be identified, not its row since the check on the rows is passed by all elements. This may be acceptable since the output of the array is correct: but the capability to identify the column with the faulty element disappears.

To overcome this drawback two solutions are available. First, it is possible to add a third adder in the basic structure of the processing element as it is shown in fig. 8.b. The output of this adder is the value Y''_{out}; while its inputs are Y'_{out} (computed by the cell itself and the value Y''_{in} (equal to the output Y''_{out} of the previous cell). If a fault occurs in the second adder, it may be detected by examining the output Y''_{out} at the edges of the arrays, since also in this case all constraints on the structure of processing element and on data flow are satisfied. Therefore, the column of the faulty processing element may be identified by examining the output Y'_{out} at the bottom of the arrays; while the row may be obtained from Y_{out} and, eventually, Y''_{out}.

An alternative solution is shown in fig. 8.c. The two additional adders are here substituted by a unique adder with three inputs: its computes a check sum for each processing element. Its inputs are the target output function Y_{ot} of the processing element, Y'_{in} equal to the check sum of the upper processing element, Y''_{in} equal to the check sum of the leftmost processing element. This structure allows to detect errors in a similar way of the previous solution, but a careful hardware design may allow to obtain a smaller silicon area.

Fault tolerance may be easily introduced in bidirectional-computing bidimensional architectures by adopting solutions similar to those proposed for horizontal-computing architectures by adding another error correction network at the bottom of the binary array: error detection and correction are performed in the same way as in horizontal-computing bidimensional arrays.

5. ERROR CORRECTION

In this section details of the architectures for error detection and correction are presented and analyzed: for each type of error model, the constraints upon bases of the adopted Residue Number System are given to assure correction. First, error due to a single fault is considered; then, the occurrence of two faults is discussed. It has be pointed out that presentation deals only with the architectural aspects of error detection and correction, since theoretical foundations are already well-known [19,20].

5.1. Single error correction

The architecture which allows to detect a single error in a linear array is shown in fig. 9.a, where only one residue array is used in parallel with the binary array. The residue representation of data consists in the residue of the data in the selected base. The alternative solution presented in fig. 9.b (with a single residue array) is able to supply the position of the faulty processing element in the linear array for reconfiguration. Similarly, the architectures of figg. 10.a and 10.b (with only one residue array) may be adopted for single error detection in bidimensional architectures. Whenever the residue of a result of the computation of the whole binary array differs from the result of the computation of the whole residue array, the error due to a fault is detected and the host computer has to reconfigure the faulty array.

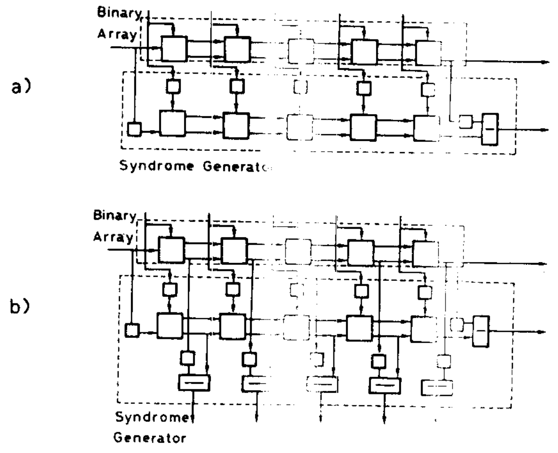


Fig. 9 - Single error detection in linear array

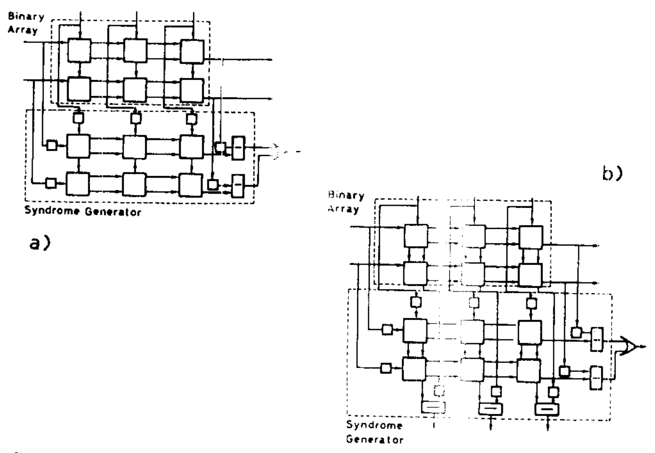


Fig. 10 - Single error detection in bidimensional arrays

Nevertheless, these architectures do not allow to correct the result of the binary array, since no enough information is available. As in usual structures based upon Triple Modular Redundancy [17], triplication of computation modules is required to correct a single error by voting their results. Thus, to tolerate a single fault the architectures presented in figg. 4, 5, 6 and 7 may be used, where two residue arrays are added to the binary array. The first architecture and the second one implement linear array, while the others are bidimensional architectures; the second architecture and the fourth one signal also the faulty processing element, allowing easier and faster reconfiguration.

When the data word of binary array has n bits, the syndromes of all powers of 2 with either positive or negative coefficient must be unique to correct a single error, i.e. the couple of bases (b_1,b_2) must satisfy [16,18]:

1.a) $s_{\pm i} \neq s_{\pm j}$, $\forall i=1,2,\ldots,n, \forall j=1,2,\ldots,n, \forall i\neq j$,

1.b) $s_{+i} \neq s_{-j}$, $\forall i=1,2,\ldots,n, \forall j=1,2,\ldots,n$.

In this case the correction generator is able to produce, without ambiguity, the correction which has to be added to the actual output of the binary array to obtain the expected result. The correction generator may be implemented by means of a combinatoric circuit (ROM, PLA or a custom circuit): its input is the syndrome, while its output is the correction.

The coverage assured by a couple of bases, i.e. the number of bits which the couple is able to correct, may be evaluated for odd bases with a simple algorithm presented in [16].

5.2. Double error correction

The architectures for single error correction are also able to detect two errors, but not to correct them, as in modular redundancy. Besides, if two errors occur, two residues in the syndrome may be zero, not allowing detection of the error. To avoid this problem, it is possible to select a couple of bases generating a non-zero syndrome for any combination of two errors; but the required bases have high values, i.e. residue processing elements are too great. An alternative solution consists in adopting a triple of bases such that no more than one residue is null for any couple of errors.

To correct two errors, three residue arrays are required in the structures presented in figg. 4, 5, 6 and 7. Also in this case the architectures of figg. 5 and 7 allow a faster reconfiguration in comparison with the other two structures.

As already said in the section about error models, there are two types of double errors: consecutive and "any kind" errors. Accordingly with the type of the double-error model, it is possible to design the correction generator and to identify the constraints to which the bases must satisfy to allow the error correction.

Let us consider "any kind" errors; the correction generator is similar to that designed for single error correction: it consists of four combinatoric circuits (such as ROMs, PLAs or a custom circuits), one for double errors, the others for single errors. To correct single and double errors, it is necessary that the triple of bases satisfies the following constraints which allow to identify uniquely the error in n-bit outputs [16,18]:

1. any couple of bases must be able to correct any single error;
2. $S' \cap S'' = \emptyset$, where $S'=\{s_{\pm i}\}$ is the set of the syndromes due to single errors and $S''=\{s_{\pm i,\pm j}\}$ is the set of the syndromes due to two errors;
3. the syndromes in S" must be unique, i.e. all combinations of powers of two must have different syndromes,
$$s_{\pm i,\pm j} \neq s_{\pm h,\pm k} \text{ for } (i\neq h \vee j\neq k) \wedge (i\neq k \vee j\neq h),$$
$$\forall i=1,2,\ldots,n, \forall j=1,2,\ldots,n,$$
$$\forall h=1,2,\ldots,n, \forall k=1,2,\ldots,n;$$
4. if a double error takes place, one residue can be zero at the most.

The architectures for correction of consecutive errors are more interesting because the probability of contemporaneous faults is very low in comparison with the probability of consecutive faults, even if the latter architectures are not able to manage transient or intermittent faults. The structure of correction generator and the constraints on the triple of bases for consecutive errors depend on the number of different syndromes produced by a permanent fault.

Let us consider single-syndrome errors. They may be managed by the correction generator shown in fig. 11. When no error occurs in the arrays, the signal α_0 allows to select the output of the binary array directly. As soon as the first error arises, it is recognized by a suited combinatoric circuit (as in single error architectures): such circuit produces the corresponding correction, which is stored with the syndrome in a register. The signal α_1 controls selection of the correct output. Until no other fault occurs, the output of binary array is corrected by the above combinatoric circuit, but corrections and syndromes are not stored. When a second fault is detected, the circuit for single errors is not able to manage it. In this case, the difference between the stored syndrome and the syndrome generated after the second fault identifies the new correction in a second combinatoric circuit; such second correction is added to the sum of the stored correction and the output of the binary array to obtain the correct result. The signal α_2 selects the correct output. This technique allows to implement compact correction generators: in fact the combinatoric circuit for "any kind" errors is surely greater than the circuits required by two consecutive single-syndrome errors.

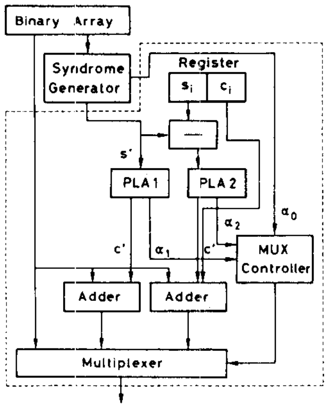


Fig. 11 - Correction generator for single-syndrome consecutive errors

To correct two consecutive single-syndrome errors, the triple of bases must satisfy the following constraints [16,18]:

1. any couple of bases must be able to correct any single error;
2. $S' \cap S'' = \emptyset$;
3. the syndromes in S' must be unique, i.e. all powers of two must have different syndromes; this condition allows to identify the second error when the previous one holds;
4. if a double error takes place, one residue can be zero at the most.

Let us consider now circuits in which a fault produces double syndrome: the triple of bases must satisfy the same constraints of single-syndrome case [16,18]. The correction generator is similar to the circuit presented for single-syndrome errors, but there are two registers to store the possible syndromes of the first error detected and their corrections (fig. 12). The signals α_1 and α_2 allow to select the suited correction for each output after the first fault, while the signal α_3 select correct output after the second fault. Also in this case the silicon area required by correction generator is smaller than the area for "any kind" errors.

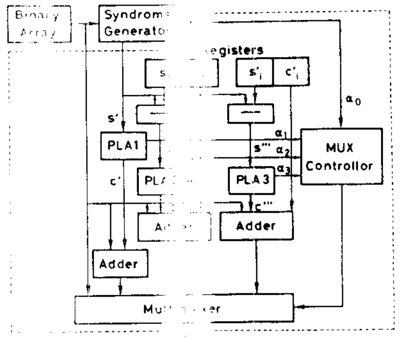


Fig. 12 - Correction generator for double-syndrome consecutive errors

Finally, when triple-syndrome errors are considered, the triple of bases must satisfy the same constraints of single-syndrome errors and the following one [16,18]:

5) $S''_i \cap S''_{-i} \cap S''_{i+1} = \emptyset$
where $S''_x = \{s_{xj} | j=1,2,\dots,n\}$ $\forall x \in \{i,-i,i+1\}$.

The structure of correction generator is an upgrade of the circuit for double-syndrome errors. Nevertheless this more complex structure and the degradation in performance can justify the choice of full-adders with a low fan-out, i.e. with no more than two different syndromes for any fault.

To compute the coverage assured by a triple of bases, i.e. the number of correctable bits, properties of residues of powers of 2 have been investigated and a computer program has been written considering odd bases and double-syndrome errors [16]. The results obtained are presented in tab. 1: when more triples supply the same coverage, the triple with the minimum sum of the squares of the bases has been selected.

The architectures discussed above may be extended to consider the cases of three or more errors. But the silicon area required to implement such architectures is too great in comparison with benefits obtained by error correction.

6. CONCLUDING REMARKS

The great interest in array architectures, their great complexity and their use in critical applications give more importance to the problem of reliability of their computation. High reliability and availability of arrays may be achieved by means of fault-tolerance techniques. Two main methodologies may be found in literature: the first approach is redundancy (either physical or time redundancy), the second one is coding.

Bases			Single e		r using pairs		2 err. consec
			correc i				
			of bas s				
i	j	k	ij	[illegible]	jk	MIN	
3	13	23	12	[illegible]	132	12	10
7	11	13	30	[illegible]	60	12	12
7	11	19	30	[illegible]	45	18	13
9	11	17	15	[illegible]	40	15	15
7	11	23	30		110	30	16
11	19	21	45		18	18	18
5	11	19	20		45	20	20
7	23	25	33	[illegible]	150	33	21
3	23	29	22	[illegible]	150	22	22
7	11	17	30	[illegible]	40	24	24
13	23	31	132		55	[illegible]	25
15	19	29	36		150	28	28
7	11	29	30	[illegible]	140	30	30
13	19	23	36	[illegible]	150	36	31
5	23	27	44	[illegible]	150	36	32
11	13	23	60	[illegible]	132	60	34
11	13	19	60	[illegible]	36	36	36
11	19	23	45	[illegible]	150	45	38
11	17	19	40	[illegible]	72	40	40
13	23	29	132	[illegible]	150	42	42
11	23	27	110	[illegible]	150	45	45
9	23	25	66		150	60	49
9	23	29	66		150	66	51
9	25	29	60		70	60	56
23	25	27	220	[illegible]	180	[illegible]	57
11	35	49	60	[illegible]	84	60	58
11	23	35	110		132	[illegible]	60
17	19	47	72	[illegible]	250	[illegible]	61
23	29	35	250	[illegible]	84	[illegible]	63
21	23	29	66		250	[illegible]	66
9	29	47	84	[illegible]	250	[illegible]	67
23	25	29	220	[illegible]	70	[illegible]	70
17	19	23	72		198	[illegible]	72
23	37	59	250	[illegible]	250	[illegible]	75
11	23	49	110	[illegible]	231	[illegible]	82
23	29	39	250	[illegible]	84	84	84
19	23	29	198	[illegible]	250	198	89
23	25	37	220	[illegible]	90	90	90
23	33	47	110	[illegible]	230	110	110
11	37	49	180	[illegible]	250	[illegible]	111
27	41	49	180	[illegible]	250	126	120
31	37	53	180	[illegible]	234	180	121
23	25	59	180	[illegible]	250	220	173
23	47	53	220	[illegible]	250	[illegible]	191
29	47	59	250	[illegible]	250	[illegible]	228
45	47	53	250	[illegible]	250	[illegible]	238
19	53	59	250	[illegible]	250	[illegible]	250
23	47	59	250	[illegible]	250	[illegible]	250
25	49	59	250	[illegible]	250	[illegible]	250

Tab. 1 – verage

In this paper a mixed appro h to fau t tolerance for processing arrays was pres ted: physical redundancy due to replication of putational elements is associated to the adoption f a resi ue number code for input and output da . Redund ncy allows to detect and correct errors nd to ge erate data for reconfiguration, while re due data representation bounds the increase of silicon area and the computation delay. In t way it is possible to exploiting the properti of mod ar redundancy without replicating the w process g element.

Error detecting and corr ting ar itectures are proposed for linear and bidimensional systolic arrays: additional circui are als introduced to allow a fast identificat of fault elements and, then, an easy reconfi ration nd an high availability of the syst The er or models are defined and the problem of e choice of the residue number system is presente .

Finally, single and multi error co rection is also discussed. An interestin esult is the negligible increase of global exec time when no fault occurs in the architectur ter a re onfiguration of the arrays. Otherwise, th mputatio is degraded by the error correction net , in pa ticular by the adder before the multiplex neverth ess, a careful hardware design of su circui s may limit degradation.

REFERENCES

[1] D.P.Agarwal, "Modulo 2^n+1 Arithmetic Logic", IEEE J.Electronic Circuits and Systems, vol.2, no.6, Nov.1978

[2] Alia, Barsi e Martinelli, "A Fast VLSI Conversion Between Binary and Residue Systems", Information Processing Letters, 18, 30, Mar. 1984

[3] A. Avizienis, "Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design" IEEE Trans. on Comp., Nov. 1971

[4] Y.H. Choi et al., "Fault Diagnosis of Reconfigurable Systolic Arrays", Proc. ICCD 1984

[5] L. Dadda, "Some Scheme for Parallel Multipliers", Alta Frequenza 34, May 1965

[6] D. Fussel, P. Varman, "Fault Tolerant Wafer Scale Architectures for VLSI", Proc. 9th Symp. Comp. Architecture, Apr. 1982

[7] K.H. Huang, J.A. Abraham, "Fault Diagnosis of Reconfigurable Systolic Arrays", IPPC, Aug. 1984

[8] T. Jayshree, D. Basu, "On Binary Multiplication Using the Quarter Square Algorithm", IEEE Trans. on Comp., Sept. 1976

[9] W.K.Jenkins, "Recent Advances in Residue Number Techniques for Recursive Digital Filtering", IEEE Trans. ASSP, vol. ASSP-27,no.1, Feb. 1979

[10] H.T. Kung, C.E. Leiserson, "Systolic Arrays for VLSI", Sparse Matrix Proc. 1978, Society for Industrial and Applied Mathematics, 1979

[11] H.T. Kung, "Why Systolic Architectures?" Computer, Vol. 15, No. 1, Jan. 1982

[12] H.C. Lai, S. Muroga, "Minimum Parallel Binary Adders with NOR (NAND) Gates", IEEE Trans. on Computer, Sept. 1979

[13] F.T. Leighton, C.E. Leiserson, "Wafer Scale Integration of Systolic Arrays", Proc. 23th Ann. Symp. Found. Computer. Sci., 1982

[14] J.L. Massey, G.N. Garcia, "Error-Correcting Codes in Computer Arithmetic", Advances in Information Systems Science, 1972

[15] R. Negrini, M. Sami, R. Stefanelli, "Fault Tolerance Techniques for Array Structures Used in Supercomputing", Computer, Vol. 19, Feb. 1986

[16] V. Piuri et al., "Residue Arithmetic for a Fault Tolerant Multiplier: the Choice of the Best Triple of Bases", Presented at EUROMICRO '86 Symposium, Venice, Italy, Sept. 1986

[17] D.P. Siewiorek, R.S. Swarz, "The theory and practice of reliable system design", Digital Press, 1982.

[18] R. Stefanelli, M. Annaratone "A Multiplier with Multiple Error Correction Capability", Arith-6, Aarhus Denmark 1983

[19] N.S.Szabo e R.I.Tanaka, "Residue Arithmetic and Its Application to Computer Technology", McGraw-Hill, New York, 1967

[20] F.J.Taylor, "Residue Arithmetic: a Tutorial with Examples", IEEE Trans. on Computers, May 1984

[21] C.S. Wallace, "A Suggestion for a Fast Multiplier", IEEE Trans. on Comp., Feb. 1964

[22] W. Watson, C.W. Hastings, "Self-Checked Computation Using Residue Arithmetic", IEEE Proc. 54(12), Dec. 1966