

USE OF $A \cdot N + B$ CODES FOR FAULT-TOLERANT BIT-SERIAL ARRAY PROCESSORS

Vincenzo Piuri

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

ABSTRACT

In this paper an approach to error detection in bit-serial array processors by means of $A \cdot N + B$ coding is presented: the optimal choice of such code is also discussed. New methodologies to design array processors with high error detection capability, low silicon area consumption and low computational overhead are proposed and evaluated.

1. INTRODUCTION

The high computation needs of many applications (ranging from signal and image processing to matrix operations) often require special-purpose architectures which allow parallel computation on input data. Array processors are an example of these systems: they consist of a number (usually very large) of identical processing elements connected by a regular interconnection grid [Kun82] [Kun87].

The continuing growth of interest in such architectures for massive computation, in particular when critical applications are concerned, points out the problem of their reliability and continuous operation. High reliability and availability of arrays may be achieved by means of fault-tolerance techniques, which usually involve error detection and correction, fault localisation and system reconfiguration. The fault tolerance of the system and its overall performance depend on the effectiveness and efficiency of such activities: some of them may be delayed or omitted to reduce computational overhead, when a bounded number of errors is acceptable in the considered application (e.g. a wrong row on the display of a radar system).

Two primary approaches to fault tolerance may be found in literature: redundancy (either physical or time redundancy) [Neg88] [Abr87] and coding [Abr87] [Avi71] [Pet72] [Wak78]. In physical redundancy the same computation is performed by different processing elements and their outputs are compared. In time redundancy the same computation is performed at different times in the same or in different processing elements.

In coding, input data are substituted by their coded representation. Computation is performed on such data and outputs are checked by custom circuits that verify for congruence with a proper output code. A second approach for concurrent error detection based on data coding consists in building additional input data from the uncoded input data and in manipulating them like a checksum of the basic computation [Abr87]. This technique is not a general solution, but it is strictly related to the algorithm executed by the array.

These fault-tolerance techniques may generally present two drawbacks with respect to their effectiveness, efficiency and realisability: silicon area for additional circuits may be relevant and delays required by redundant computation may become unacceptable.

In this paper we study the use of $A \cdot N + B$ codes to detect errors due to faults in bit-serial array processors: these codes define a linear transformation of the data space on which arithmetic units work [Pet72] [Avi71]. The array architecture is presented: bit-serial arithmetic units are considered since they allow implementation of large arrays on a single chip or wafer. The compactness of each unit and the reduced number of interconnections with respect to the correspondent bit-parallel solution save a very great amount of silicon area. On the other hand, it has been shown, e.g. in [Neg85], that bit-serial array architectures can often give a higher throughput than bit-parallel architectures since delay due to interconnection wires between processing elements on different chips is drastically reduced by interconnecting the smaller processing elements on the same wafer.

The use of $A \cdot N + B$ codes is then introduced: some criteria are given for identifying the best code to protect a given number of bits minimising the silicon area and the computation overhead. For some classes of values of the coefficients A and B some low-cost encoder and decoder circuits are presented.

Different architectures are proposed and evaluated to detect errors and localise the faulty modules both for linear and for bidimensional arrays; two approaches are considered:

- error detection inside each processing element verifies locally the computation before delivering the results to the adjacent processing elements,
- error detection at array level verifies the correctness of the computation only at the array borders, saving a great amount of silicon area.

2. ARRAY AND PROCESSING ELEMENT

An array is composed of a set of simple processing elements (PEs) connected by a proper network [Kun82] [Kun87]. The characteristics of these architectures and the algorithms executed depend on the structure of PEs and on the interconnection network. Here we consider a regular interconnection mesh which is applicable to a wide class both of linear and bidimensional arrays for digital signal/image processing and matrix operations. This approach to fault tolerance in bidimensional arrays may be adapted to architectures in which the intermediate results of the computation flow only in horizontal direction or in horizontal and vertical directions according to the algorithm being executed by the whole array.

From the literature it is possible to extract a simple common structure of the PEs for the class of applications considered here (fig. 1). 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 properly synchronise the data supplied according to the target algorithm, as e.g. it is required by the evaluation of the discrete Fourier transform.

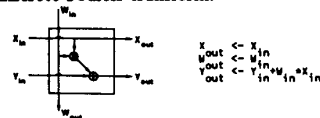


Fig. 1

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, e.g. in digital signal and image processing and in matrix arithmetic.

To minimise the silicon area required by the PE and, thus, to implement large array processors on a single chip or on a wafer, we consider in this paper only bit-serial arithmetic units, even if our approach can be adopted also when bit-parallel units are concerned. Under this assumption, an adder consists simply of a single full adder whose carry output is fed back into its carry input after one clock delay. Different kinds of bit-serial

multipliers have been proposed in literature; in the research here presented, we have carefully studied Dadda's architecture [Dad85] because of its regularity and speed, even if the most of the achieved results can be extended to other structures. For the sake of simplicity we present our technique to achieve fault tolerance by means of $A \cdot N + B$ coding by considering integer input data represented in two's-complement notation.

3. FAULT AND ERROR MODELS

To evaluate the array architecture and the features of $A \cdot N + B$ coding with regard to fault tolerance, we have adopted the classic fault models at gate level [Abr86] both for permanent, transient and intermittent faults; in particular, we consider stuck-at faults, stuck-on, stuck-open and bridging faults. Related to such fault model, we consider a classic error model at gate level [Abr86]: 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. Thus, the actual result may be viewed as the sum of the correct result plus the error.

In a bit-parallel unit each output bit is produced by a different subset of devices of the unit itself: therefore, when circuits are carefully designed, each single fault generates a known set of errors whose values are powers of two [Pet72], while multiple faults produce errors given by the sum of the errors due to each single faults. Under these assumptions it is possible to design a code and an architecture that is able to detect and correct single and multiple errors. In bit-serial arithmetic units all output bits are computed by the same devices iteratively: therefore, any simple deterministic relationship between fault and induced error is lost. In this case correction of the wrong result becomes impossible on deterministic basis since we cannot uniquely identify the error and the correcting term which has to be added to the actual output to achieve the correct result. On the other hand, we found that also error correction on probabilistic basis is not feasible under our assumptions about fault model. To such purpose we simulated on computer the behavior of arithmetic units when different faults occur: we have found that no relationship may be defined between faults and erroneous results.

4. $A \cdot N + B$ CODES

$A \cdot N + B$ codes constitute an important class of non-separate codes [Pet72] [Wak78], i.e. codes where subsets of bits cannot be processed in parallel by arithmetic units. The arithmetic unit does not manipulate the input numbers N directly, but each of its target inputs is modified via the linear transformation $A \cdot N + B$ and then delivered to the arithmetic unit which executes the target operations in the transformed data space. The output of such operations may be moved back to the target data space via the inverse transform.

By means of $A \cdot N + B$ transformation, some codes with different interesting properties of error detection may be defined [Pet72], e.g. cyclic codes, sphere-packed codes and Mandelbaum-Barrows codes. However these particular properties do not hold in bit-serial arithmetic under general fault conditions; therefore we consider only the general characteristics of $A \cdot N + B$ codes.

The functions computed by arithmetic units for coded input data and the structure of such circuits are the same as units working on the uncoded input data. The only difference consists in the number of bits processed. If N^* is the maximum considered input number, $\lceil \log_2 N^* \rceil$ bits are used in the first case, while $\lceil \log_2 (A \cdot N^* + B) \rceil$ bits are required by coded data.

This difference implies that the computation time is greater with coded data than with target data because of the additional bits which must be treated and the time required to perform data transformation.

On the other hand, the silicon area for array processors with coded data is increased by

- the encoding circuits, that transform the target inputs of the array into the corresponding coded representation;
- the decoding circuits, that transform the coded outputs back into the uncoded representation;
- the additional storage elements required in each PE to record the longer partial results of multiplication;
- the additional bit-serial adder, to allow the connection of adjacent PEs when an $A \cdot N + B$ code is adopted; for this code, in fact, the output of the basic adder in the PE belongs to the $A \cdot N + 2B$ code:

therefore the constant $-B$ must be added to such output to obtain a code compatible with the following PE, i.e. to connect the PE to the adjacent one.

The area for encoding and decoding circuits may be partially reduced by a careful choice of the code's coefficients. For example, if we adopt a simple $A \cdot N$ code ($B = 0$), the output data are still coded in a code similar to the input one: in particular, such output code is the same as that the inputs for addition, while for multiplication with both inputs coded the first coefficient is A^2 . This property allows to connect a PE to adjacent ones directly without inserting either decoders or encoders.

By adopting $A \cdot N + B$ codes, the error detection in bit-serial arithmetic units and, thus, in array processors consists of verifying that the results computed by each unit or by the whole array belong to the proper code. This check may be easily performed by modifying slightly the decoding circuits since we must verify that the result minus the second constant B of the output code is divisible by the first constant A of the output code.

4.1 About the optimum $A \cdot N + B$ codes

To fully exploit the error detection capabilities of $A \cdot N + B$ code and to minimise the requirements of such technique (computation delay and silicon area), a careful choice of the coefficients A and B is necessary.

Let's first consider $A \cdot N$ codes, i.e. the coefficient B is assumed null. This is the simplest approach which allows saving of silicon area and computation time compared with the more general class of $A \cdot N + B$ codes. In $A \cdot N$ codes different error detection capabilities may be achieved by choosing the coefficient A . A class of such codes consider $A = 2^k$; it is possible to show that such choice is not adequate to achieve fault-tolerance performance [Pet72]. In fact the coded data may be obtained simply by shifting the input N for k bits on the left: in this case the correctness check degenerates to verifying that the k least significant bits of the encoded number are null. Such condition does not ensure the proper coverage for all errors that influence any bits but the first k bits, which do not contain useful information. Therefore, this class of coefficients is not suited to minimise costs and maximise the error coverage.

The following criterion may be adopted to perform an optimal choice for an $A \cdot N$ code with respect to the silicon area and the computation time. If A and N are represented by means of k and n bits respectively, the coded representation $A \cdot N$ of the input N requires $n + k$ bits at most. The best value of A is the one that maximises the distance between the representation of two successive numbers [Wak78]. Therefore, the optimum value of A is the greatest number whose binary representation requires k bits, i.e. $2^k - 1$. In fact, the best performance is achieved when the detection probability is maximum, i.e. when the undetection probability is minimum. This happens when the coefficient A is maximum.

Better results may be achieved if the uncoded input data are a subset of all n -bit configurations. In this case the optimisation might reduce the number of bits required by the coded representation of the input data: nevertheless, no simple and general relationship was found between the maximum input number and the coefficient A to maximise the error detection capabilities and to minimising the silicon area and the computation time.

If many inputs of a PE are null, the test on PE outputs may not detect any error and, thus, the fault latency may increase dangerously. Besides, if a fault is not detected as early as possible and other faults arise, detection capabilities might be decreased by the joined interaction of the effects of faults on the computation.

To minimise the fault latency in arithmetic units we adopt the more general class of $A \cdot N + B$ codes [Pet72]. In fact, to reduce the fault latency it is necessary to avoid the presence of null code word: in the $A \cdot N + B$ code a non-null coefficient B allows for this. Nevertheless, some problems arise in the implementation of arithmetic units using this data coding because the outputs may be in a representation different from the input code or from similar ones.

For example, let's consider an adder: if the inputs are represented in the $A \cdot N + B$ code, the output is in the $A \cdot N + 2B$ code, i.e. in a code similar to the input one. The multiplication has usually a different behavior. In fact, it is: $P = (AX + B)(AY + B) = A^2(XY) + AB(X + Y) + B^2$. It cannot be transformed in a coded representation of the product XY , like $C \cdot XY + D$, without subtracting the first order term $(X + Y)$.

Still, some classes of $A * N + B$ codes may be identified in which the code representation of the multiplier output behaves as a $C * N + D$ code. Let's consider the finite field in which operations on n -bits numbers are performed: if we assume that no overflow occurs during computation, the usual arithmetic operations performed in such field may be viewed as operations in a base b of a residue number system [Ssa67]. In particular, the output of the multiplication is in a $C * N + D$ code when the inputs are in a $A * N + B$ code if $R_b[A^2(XY) + AB(X + Y) + B^2] = R_b[CKY + D]$, that is if $R_b[AB] = 0$. Such condition is satisfied, for A and B different from zero, if $AB = b$.

From this property some $A * N + B$ codes may be identified to achieve the goal previously discussed by choosing the value of the base b . For example, if c bits are used for the coded word, the greatest base b in which residues are represented in c bits is 2^c . In this case the coefficient A is equal to a power of two since it must be different from 1: but, as already said, it is not a value suited for error detection.

A second class of $A * N + B$ codes may be defined if we accept eventually to discard some codewords, i.e. to reduce the range of input data. Let's consider a base $b = 2^c - s$. For $s > 0$, this base allows a pair of values for A and B to be chosen so that A is not a power of two. The error detection capabilities may be exploited with maximum performance of silicon area and computation time when the value of b is maximum, i.e. when $b = 2^c - 1$. In tab. 1 some pairs of values for A and B are given when different numbers of bits are assumed.

c	A	B	c	A	B	c	A	B	c	A	B
4	1	3	5	1	7	6	1	31	7	15	29
4	2	6	5	2	14	6	2	6	14	28	56
4	3	5	6	3	11	7	3	9	5	23	46
4	3	5	6	4	8	7	4	4	11	19	38
4	3	5	6	5	5	7	5	3	13	17	34
4	3	5	6	6	2	7	6	2	17	11	22
4	3	5	6	7	1	7	7	1	23	7	29
4	3	5	6	8	0	7	8	0	31	0	39
4	3	5	6	9	0	6	9	0	31	0	39
4	3	5	6	10	0	5	10	0	31	0	39
4	3	5	6	11	0	4	11	0	31	0	39
4	3	5	6	12	0	3	12	0	31	0	39
4	3	5	6	13	0	2	13	0	31	0	39
4	3	5	6	14	0	1	14	0	31	0	39
4	3	5	6	15	0	0	15	0	31	0	39
4	3	5	6	16	0	0	16	0	31	0	39
4	3	5	6	17	0	0	17	0	31	0	39
4	3	5	6	18	0	0	18	0	31	0	39
4	3	5	6	19	0	0	19	0	31	0	39
4	3	5	6	20	0	0	20	0	31	0	39
4	3	5	6	21	0	0	21	0	31	0	39
4	3	5	6	22	0	0	22	0	31	0	39
4	3	5	6	23	0	0	23	0	31	0	39
4	3	5	6	24	0	0	24	0	31	0	39
4	3	5	6	25	0	0	25	0	31	0	39
4	3	5	6	26	0	0	26	0	31	0	39
4	3	5	6	27	0	0	27	0	31	0	39
4	3	5	6	28	0	0	28	0	31	0	39
4	3	5	6	29	0	0	29	0	31	0	39
4	3	5	6	30	0	0	30	0	31	0	39
4	3	5	6	31	0	0	31	0	31	0	39

Tab. 1

When different values of the pair of coefficients A and B are allowed to protect the same number of bits, some criteria should be defined for selecting the best combination. But a simple rule cannot be identified since conflicting cost functions are usually considered. In fact large value for the coefficient A allows for high rates of error detection since the codewords are far from each other, but only a reduced range of input data can be coded with the given number of bits. On the other hand, a greater number of bits requires a greater silicon area and an high computation time. Therefore, the choice of the code must take into account the application of the array processor and, thus, the available silicon area and the timing constraints for the computation.

4.2 Encoders, decoders and checkers

To generate the $A * N + B$ coded representation of the inputs it is necessary to multiply them by a constant A and to add another constant B . The circuit implementing this multiplication is simpler than a usual multiplier since one input is known a priori. Some compact and fast technique can thus be adopted for the encoder when particular classes of coefficients are assumed.

Let's consider the encoder for an $A * N$ code in which A has been chosen according to the criteria previously presented, i.e. $A = 2^k - 1$. The coded input P is given by $P = A * N = (2^k - 1) * N = 2^k N - N$. The circuit performing such operation is shown in fig. 2. Its silicon area is $A_{fa} + (k+1)A_{ff}$, where A_{fa} and A_{ff} are the area of the full adder and of the flip-flop respectively. The total computation time is $[k + \log_2 N^*]r$, where r is the clock period and N^* is the maximum representable uncoded number. The proper alignment of input data to perform the transformation can be achieved by means of the sign extension rule [Dad85], i.e. the most significant bit of the uncoded input data must be supplied again as input during the last k clock periods.

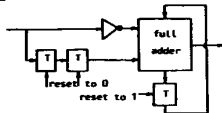


Fig. 2

Similar approaches can be adopted for other values of the coefficient A in an $A * N$ encoder, but no general structure was found for such cases: each value should thus be carefully studied.

As the $A * N + B$ codes are concerned, the encoder may be obtained by connecting a serial adder to the previous encoder to add the constant B . The bits of B can be supplied one at time by a shift register whose output is feedback into its input. To save silicon area only one of such registers can be implemented in the array processor: its output must be broadcasted to all encoders that need it.

To transform the output of the array processor back to the uncoded representation it is necessary to design fast and compact decoder circuits. Besides, to check the correctness of computation we must verify that a proper constant divides the result of the computation itself or the difference between the result and the constant B , according with the adopted code. To this purpose we present here some approaches which effectively implements the division by the first constant (A or A^2 according with the selected architecture): the decoder can be implemented directly by one of such circuit, while the error detection requires a very simple checker (which is constituted by few AND/OR gates) that verify the output of some bits of the decoder.

Let's consider the $A * N$ codes ($A = 2^k - 1$); the input P to the decoder is $P = A * N = (2^k - 1) * N = 2^k N - N$. Thus, the decoded output may be obtained from $N = 2^k N - P$. Even if N is a priori unknown, its k least significant bits may be computed from the corresponding bits of the number P , while the other bits may be evaluated from the remaining bits of P and the bits of N previously decoded.

The bit-serial architecture which executes this operations is shown in fig. 3. Its silicon area is $A_{fa} + (k+1)A_{ff}$, while the total computation time is $[k + \log_2 N^*]r$.

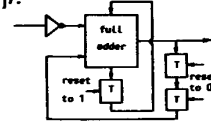


Fig. 3

Error detection consists of checking if the result of PE's computation is divisible by A . By using the sign extension rule, such test consists in verifying if the k most significant bits of the decoder output are equal to the bit whose weight is $2^c - k - 1$, where c is the number of bits of the codewords.

If the array outputs are represented in an $A^2 * N$ code (with $A = 2^k - 1$), some particular decoder circuits may be designed. A first approach implements such decoder by performing two subsequent divisions by A by means of the decoding circuit shown in fig. 3. A simple extension of this circuit allows for a self-checking decoder. If the inputs belong to the code, the k most significant bits of the first division are equal to the most significant bit of the product $A * N$; therefore, it is not useful to feed them to the second divider. If we supply the k least significant bits of the input to the second divider during the last k clock-steps, we obtain a self-checking decoder. If the k last bits generated by the second divider differ from the k least significant bits computed by the first divider, a fault occurred in one of the two dividers.

The second technique to test and decode the result of a computation by means of a divider by A^2 consists in the direct division. In this case a custom circuit must be designed for each value of coefficient A because there is not a general algorithm, as in the previous circuit for the class of $A = 2^k - 1$.

Decoders for $A * N + B$ and $A^2 * N + B$ codes may be designed by connecting the input of the previous decoders to a bit-serial adder in which the constant B is subtracted from the coded input data.

5. ERROR DETECTION AT LOCAL LEVEL

Use and performances of $A * N$ codes in local error detecting are presented here: the computation of each individual PE is directly checked before its results are delivered to adjacent PEs.

Two levels of granularity can be designed for local error detection:

- the output of each arithmetic unit in the PE may be verified,
- the output of the whole PE is checked. The first approach allows the finest and faster error detection and fault localisation because the error is detected directly at the output of each arithmetic unit; in this case a checker must be added to each unit in the PE.

Nevertheless, even if the application of array processor is critical, more relaxed constraints are acceptable about fault latency and localisation. In particular, at most we need to identify the faulty PE, not the faulty unit inside the PE; in fact the reconfiguration process considers the whole PE and it is not able to use parts of different faulty PEs to build a working PE. Therefore, we do not further consider this first solution which requires a silicon area too large.

The second approach achieves error detection at local level but with an acceptable cost, even if error latency may increase for the reduced capability of localising faults, i.e. for the decreased number of check points. Instead of checking the outputs of all arithmetic units, it is possible to insert the proper checkers only at the outputs of the whole PE before supplying them to the adjacent PE (fig. 4).

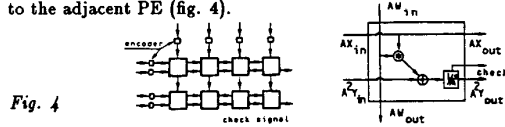


Fig. 4

The area of the PE is increased by the additional area required by each arithmetic unit to work with coded words (as in the previous case), and by only one checker connected at the PE output. The redundant area is halved with respect to the first solution; besides, the incidence of the checkers is considerable when there are very few PEs in the array. Output checking may be designed so that no computational delay is introduced at array level: in fact, data computed by arithmetic units of a PE can be concurrently delivered to the adjacent PE and verified while the PE itself computes the subsequent output data. Only the encoders and the decoders at the array borders delay the output of results from the array processor. Another source of delay with respect to the computation of the array for uncoded data is intrinsic to the arithmetic units: each of them must work on more bits than the correspondent units for uncoded data. The probability of detecting errors due to faults when $A \cdot N + B$ codes are adopted has been evaluated. To such purpose we have extensively simulated with a computer program the behavior of this architecture when different faults arise. The error latency is slightly greater than in the previous solution because the check on the output of the multiplier is delayed after the addition and performed together with the check on the addition itself.

Some results of our analysis of the performances of $A \cdot N + B$ codes that can be achieved via this first PE architecture are shown in fig. 5. There we have drawn the percentage increase of the silicon area of a PE and the percentage increase of its computation time with respect to the architecture without error-detecting capabilities: these characteristic parameters have been evaluated by considering the number of gates and their switching time in some experimental implementations. Detection probability is also given. All these parameters are drawn versus the number n of bits of uncoded words, the number k of redundant bits and the number of PEs in the array ($M \times M$ arrays are considered).

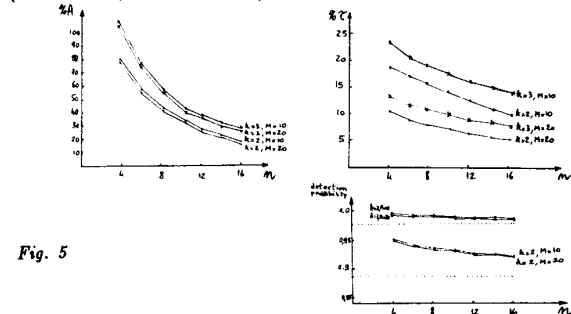


Fig. 5

The last remark about these architectures related to the capability of localising faults. The same signal, that points out the presence of an error due to a fault in a PE, allows the faulty PE to be localised. In fact these error signals can be directly carried out of the array and tested: the active signal identifies uniquely the faulty PE; nevertheless, this solution does not allow to design a regular structure of the array processor. A better approach consists in identifying separately the row and the column of the faulty PE in the array. To identify the row a combined signal is generated in each PE by or-ing the error signal of the PE itself and the combined signal

coming from the previous PE in the row; a similar signal manipulation is performed along the columns to identify that containing the faulty PE.

6. ERROR DETECTION AT ARRAY LEVEL

A system-level approach is proposed and evaluated to reduce the additional silicon area, without decreasing the detection probability drastically. In such approach concurrent error detection via $A \cdot N + B$ codes is achieved by checking the outputs of the PEs only at the array borders, while the PEs inside the array do not contain any checker.

The computation delay is equal to that introduced in local error detection, but error latency increases because the errors can be detected only at the array borders.

6.1 Linear arrays

The simplest solution for a linear array processors is shown in fig. 6. The input data of the linear array are coded in a suited $A \cdot N + B$ code, chosen according with the required detection performance; then, the PEs execute their computation as in the basic array without error detection capabilities: the only difference is the number of bits that must be treated and, thus, the computation time and the dimensions of the multipliers. The decoder/checker is connected at the output of the last PE of the array; when the output does not belong to the selected code, an error has been detected in the array.

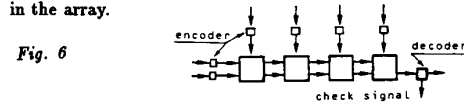


Fig. 6

This solution allows a low increase of silicon area with respect to the array for uncoded data because only one checker is introduced in the structure; as in the architectures for local error detection, a constant area is used by encoders at the array borders, while other constant additional area is required in each PE to treat input data having a higher number of bits than uncoded data. Let's note that the influence of the checker in the area increase of each PE decreases and becomes neglectable as the number of PEs increases. In fig. 7 performance of this architecture are shown. The area increase with respect to the architecture without error-detecting capabilities has been evaluated from the number of gates of some experimental implementations, the detection probability has been estimated by simulating on computer the behavior of the architecture presented in fig. 6.

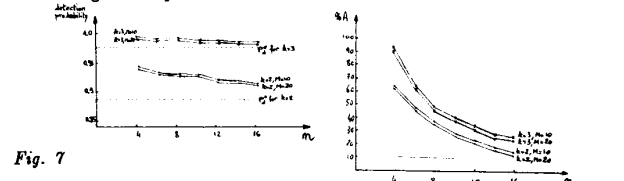


Fig. 7

This solution, so attractive for different characteristics, is not able to supply any information about the position of the faulty PE in the array. Thus, to reconfigure the array an off-line testing of the PEs must be executed to identify the faulty element and to exclude it from the computation. A fast identification of the faulty PE and a high availability of the system may be achieved by means of additional circuits. For linear arrays, this second solution is equivalent to insert one checker at the output of each PE, as proposed in the second approach of the previous section.

6.2 Bidimensional arrays

Concurrent error detection in bidimensional array processors may be achieved by extending the first solution presented for linear arrays, as it is shown in fig. 8.

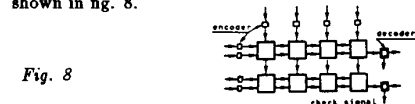


Fig. 8

The input data are coded by using the proper encoder circuit and are supplied to the array. Also in this architecture PEs are similar to the PEs of the array without error detection capabilities: they differ only in the number of treated bits. The checkers are connected to the outputs of PEs at the border of the array, while no correctness test is performed in the other PEs.

As in the linear array, this approach allows a low increase of silicon area with respect to the array for uncoded data because only one checker is

added for each row of the array. A constant area is used by encoders at the array borders, while other additional area is used in each PE because the inputs have a higher number of bits. The percentage of area increase due to decoders and to the higher number of bits treated in each PE is the same of the solution discussed for linear arrays: the incidence of the higher number of bits does not depend on the number of PEs in the array processor, while the area percentage for the decoders/checkers decreases when the number of PEs increases. Another fraction of the area increase is due to the encoders at the array borders: the percentage of such area obviously decreases when the number of PEs in the array increases. In fig. 9 the percentage of redundant area for each PE is drawn when this architecture is considered; detection probability behaves as in the first architecture for linear arrays.

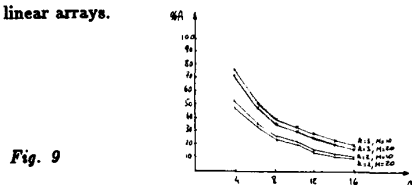


Fig. 9

Whenever an error is detected, this solution is able to identify the row in which the fault occurs: since the output of each row is separately tested, we can identify the row of PEs containing the faulty unit by examining the signals generated by the checkers. But, the faulty PE cannot be identified because only the output of the whole row is verified. To overcome this problem it is necessary to introduce additional circuits in the array.

An attractive architecture, which allows this problem to be overcome, is shown in fig. 10a. Instead of checking the output of all PEs belonging to a column by introducing a checker for each PE, such outputs are added together to generate an additional output Y' of the array: in the array this computation flows orthogonally to the main computation.

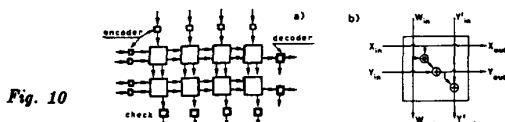


Fig. 10

To compute these sums the PE must be modified by inserting a bit-serial adder, as it is shown in fig. 10b; its inputs are the output of the PE considered until now and the partial sum computed by the corresponding adder of the PE in the above row of the same column.

The additional outputs Y' give information about the state of the PEs belonging to the corresponding column. Each of these outputs is tested by a checker identical to the circuits used for the rows of the array. The checkers on the right border of the array allow to identify the row in which there is a faulty PE, while the checkers on the lowest border identify the column in which it lies. This two information are enough to localise the faulty PE. This solution requires a greater area with respect to the previous architecture; in fact a little further area increase must be considered in each PE because of the additional bit-serial adder which compute the output Y' . In fig. 11 the percentage of the area increase for such architecture is given.

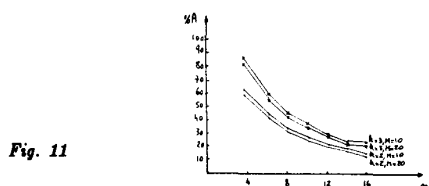


Fig. 11

The additional adder inserted in the PE of fig. 10b, protects the multiplier and the first adder, but it might not detect errors due to faults in the adder itself. If such circuit fails, only the column containing the faulty PE might be identified, not its row since the check on the rows is passed by all PEs.

To overcome this problem two solution are available. First, it is possible to insert a third bit-serial adder in the structure of the PE, as it is shown in fig. 12a. The output of this adder is the partial sum Y'' , which

is obtained by adding the output Y' computed in the PE itself and the output Y'' of the previous element in the row. If a fault occurs in the adder which generate Y' , it may be detected by examining the outputs Y'' at the borders of the array. Therefore, in this architecture the column of the faulty PE may be identified by examining the outputs Y' at the border of the array, while its row may be obtained from Y and, in case, from Y'' . Each PE is greater than the PE for uncoded data since it must treat inputs represented by means of a higher number of bits and it contains two additional bit-serial adder.

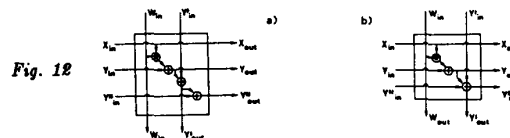


Fig. 12

A second solution is shown in fig. 12b: the two additional adders are here substituted by a unique adder with three inputs: it computes a checksum for each PE. Its inputs are the output Y , the checksum Y' of the upper PE and the checksum Y'' of the adjacent element on the left side. This structure detect errors as the previous one, but a careful hardware design may allow to save silicon area. In fig. 13 a comparison between the area increase of the architectures presented for bidimensional array processors when the same code is considered.

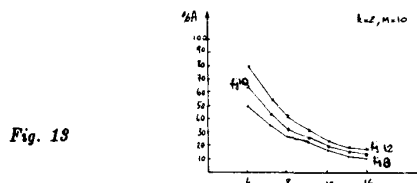


Fig. 13

We note in all architectures here proposed that input data may 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 computer driving array computation would store the position of the column (row) containing a faulty element to identify it as soon as possible.

7. CONCLUDING REMARKS

In this paper we propose and discuss the use of $A \cdot N + B$ coding technique to achieve a concurrent error detection in a wide class of bit-serial array processors for digital and image processing and matrix computation. The optimum choice of the code coefficients is also discussed to achieve high performance at the minimum cost in term of silicon area, computation delay and error latency.

$A \cdot N + B$ coding is considered both at a local level and at an array level. In the first case each PE is tested to detect errors; in the second case the check on the results is performed only at the outputs of the whole array. For both approaches different structures are proposed and evaluated with regard to silicon area, computation delay and fault localisation.

From all these architectures and our remarks, it is possible to conclude that $A \cdot N + B$ coding constitutes a good and effective solution for concurrent error detection and localisation in bit-serial array processors. A complete fault tolerance of such structures, i.e. the survival of the system after faults, may then be easily achieved by means of the reconfiguration techniques well known in literature: some of the architectures presented here are in fact able to supply on-line information about the position of the fault in the array, while others require an off-line test procedure to localise the faulty processing element.

REFERENCES

- [Abr86] J.A. Abraham, W.K. Fuchs, "Fault and error models for VLSI", Proc. IEEE, may 1986
- [Abr87] J.A. Abraham, et al., "Fault tolerance techniques for systolic arrays", Computer, july 1987
- [Avi71] A. Avizienis, "Arithmetic error codes: cost and effectiveness studies for application in digital systems", IEEE Trans. on Comp., nov. 1971
- [Dad85] L. Dadda, "Fast multipliers for two's complement numbers in serial form", ARITH-7, 1985
- [Kun82] H.T. Kung, "Why systolic architectures?", Computer, jan. 1982
- [Kun87] S.Y. Kung, et al., "Wavefront array processors - Concept to implementation", Computer, 1987
- [Neg85] R. Negrini, M. Sami, "Array architectures for signal processing", Journ. d'electronique, 1985
- [Neg88] R. Negrini, M. Sami, R. Stefanelli, Fault-tolerance through reconfiguration of VLSI and WSI arrays, MIT Press, to be published
- [Pet72] W.W. Peterson, E.J. Weldon, Error correctin codes, MIT Press, 1972
- [Sas67] N.S. Saabo, R.I. Tanaka, Residue arithmetic and its application to computer technology, McGraw-Hill, 1967
- [Wak78] J.F. Wakerly, Error detecting codes, self-checking circuits and applications, North Holland, 1978