A Multiplier with Multiple Error Correction Capability[1]

Marco Annaratone

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Renato Stefanelli

Dipartimento di Elettronica
Politecnico di Milano
Milano, Italy

Abstract

This paper presents a technique for increasing the reliability of arithmetic units. An error model is then presented; this model well represents the faulty behavior of many arithmetic units. The Residue Number System and its related properties are used in order to obtain a simple architecture (called Reliability Network, R-Net). The main characteristics of the presented technique are a significant reduction in the number of gates and a limited increase of global execution times. The extensive use of combinational logic makes it possible to implement the R-Net almost completely by means of Programmable Logic Arrays (PLA's). Finally, both the intrinsic regularity of the R-Net and its simple internal interconnection scheme make this approach suitable for a practical VLSI implementation.

1. Introduction

Most techniques commonly used in fault-tolerance (e.g. Triple Modular Redundancy, duplication with checker, input complementation as in Takeda [22] and so on) often result unsatisfactory when applied to arithmetic devices, due to their high cost (both in term of area and of design effort) and negative influence on the ultimate machine's performance.

Moreover, the design effort becomes very high when speed constraints are severe and complex architectures are required; this fact restricts reliable computation to special applications where cost can be considered a secondary constraint. In order to make reliable computation more available, it is necessary to reduce design's as well as devices' cost. If self-correcting units were available, the cost of design would be significantly reduced.

This paper deals with the design of reliable arithmetic units; the design of a fault-tolerant multiplier will be presented as an example. Residue number representation has been chosen for its powerful checking capabilities. Presently, a custom VLSI fault-tolerant multiplier, using N-channel, MOS technology ($\lambda = 1.5$ micron), is being developed.

[1]This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-81-K-1539.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

This research was also sponsored in part by the Consiglio Nazionale delle Ricerche, C.N.R., Italy.

[2]The "hard-core" is the area that is not protected against fault.

The approach presented in this paper features:

*strong regularity of the whole structure, which allows an extensive use of standard cells (PLA's and/or ROM's);

*limited increase of global execution time due to the redundant logic;

*minimization of the hard-core area[2].

2. General Architecture and Error Model

The Residue Number System (RNS) has been a well-known number representation for almost thirty years. Important contributions are those of Svoboda [21] and Garner [8] to which the reader is referred for a survey of residue algebra. Several studies have been performed, especially in the 60's; on the application of RNS to reliable computation, for example, by Rao ([18], [19] and [20]), Avizienis ([2], [3]), Watson [25], Garner [9], Brown [5] and Massey [14]. RNS has also been used to implement fault-tolerant arithmetic units with non-binary data representation (see [3], [25], [13], [15]). Error detection and correction codes based on RNS have been used in data transmission or mass storage (see [23], [16], [17], [10], [4]).

In the present paper, fault-tolerance is not achieved by self-correcting residue codes, but by a proper architecture (the *R*esidue-*Net*work, R-Net). The R-Net is a three-input structure: the two operands and the result of the arithmetic unit. The particular architecture depends on the error model, *which is here defined in an "additive sense"*, i.e. for any possible fault, the correct output is derived from the output of the arithmetic unit algebraically added to the amount of error.

A fault can produce different errors in the arithmetic unit output, namely:

1. *One-bit error*: when the amount of error E is:

$$E = \pm 2^i$$

This fault is peculiar of most adders when a "stuck-at" at the output of a gate occurs.

2. *Two (or more) bit error in any position*:

$$E = \pm 2^i \pm 2^j$$

3. *Two consecutive bit error*: for the first error we have:

$$E^I = \pm 2^i;$$

when a second error takes place we have:

$$E^{II} = E^I \pm 2^j;$$

in this case, however, E^I is known and *stored*. This case is *different* from the previous one; the R-Net does *not* have to compute a global correction, but only that of the second error. The above assumption holds only for permanent faults.

In this paper it is always assumed that the arithmetic unit can only produce a *single* error, i.e.:

$$E = \pm 2^i$$

Significant examples satisfying the error model are:

*a full-adder (e.g. the 7483);
*a multiplier implemented by Dadda's [6] or Wallace's approach [24] or by related approaches, as presented in [11];
*any network performing a function as $f(.) = A^2 + B^2$ (but *not* $f(.) = A * [B + A * B]$)

In the next section an R-Net capable of correcting a single error will be presented. The discussion will deal only with the architectural aspects of the problem, since the theoretical foundations are well-known (see [19], [20]). This architecture is the basic building-block for further improvements (i.e. multiple error correction).

3. Single Error Correcting Architecture

The architecture is shown in Fig. 1. It is derived from what is presented in [19] and overcomes the problem of the hard-core, by reducing it to the last stage (the MUX section). Let us consider now how the R-Net redundancy influences the speed of the overall unit.

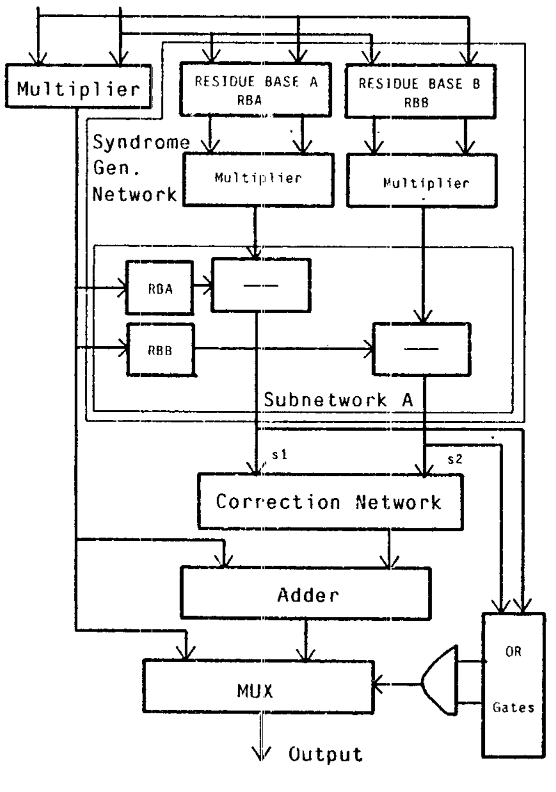

Figure 1: Architecture for single error correction.

When the arithmetic unit (in our example, the multiplier) is working correctly, the delay introduced by the redundant logic consists of the residue conversion of the result, the MUX control logic and the MUX itself. Note that the residue conversion of the multiplier output introduces a very limited delay, due to the right-to-left carry propagation inside the multiplier. Moreover, the subtractor *does not* introduce any delay. The approach is similar to that presented in [7], although the subtraction was not performed in that approach.

4. Multiple Error Correcting Architectures

As regards the occurrence of two (or more) errors, we shall examine two different instances:

**two or more errors of any kind*[3]: the proper architecture will be a suitable expansion of the basic structure shown in Fig. 1;
**two or more permanent errors* (stuck-at-0 or stuck-at-1 in any gate), which take place *at different times* (they will be called, from now on, "consecutive errors"): the first-error syndrome can be stored, thus making the correction of a second error simpler. In this case the R-Net will be significantly modified.

As already pointed out, the error is considered in an "additive sense" and always leads to an error which is power of two; therefore, the amount of error will be:

$$E = \pm 2^i \pm 2^j$$

In the sequel we shall not consider two errors with same magnitude and opposite sign (zero-error occurrence) and two errors with the same sign and magnitude, as they lead again to a single error:

$$E = 2^i + 2^i = 2^{i+1}$$

For the same reason, errors of this kind will not be taken into account:

$$E = \pm 2^i \mp 2^{i-1} = \pm 2^{i-1}$$

Threfore, the syndrome $s_{i,j}$ produced by two errors is simply:

$$s_{i,j} = [s_i + s_j]_b,$$

being s_i and s_j the syndromes of each error and b the residue base.

If a single error occurs, all the residues constituting the syndrome are different from zero; if two errors occur, a residue *can be zero*. For instance, if b = 17:

$$r_{i,i+4} = [2^i + 2^{i+4}]_{17} = 0$$

Two different approaches can be pursued to overcome the problem:

1. the selection of a couple of bases each of them producing a non-zero syndrome for any possible combination of two errors;
2. the selection of a *triple* of bases in such a way that, for any possible occurrence of two errors, only one residue can be zero.

The first approach is not efficient, because it compels us to use large-valued bases. The second approach is more attractive, because a fault in the syndrome computing network produces one non-zero residue: this error is, thus, detectable. A further consideration leads to prefer the second solution: if a fault producing the first error occurs in the syndrome computing network, the couple of error-free residues can still correct a second error generated by a fault in the multiplier.

5. Correction of Two Errors of Any Kind

The architecture is shown in Fig. 2. The syndrome generator computes s_1, s_2, s_3; if both the first and the second error take place in the multiplier, the multiplexer controller chooses among I', I'' and I''' (the

[3]The term "*any kind*" indicates either permanent or transient/intermittent errors and either contemporaneous or consecutive errors.

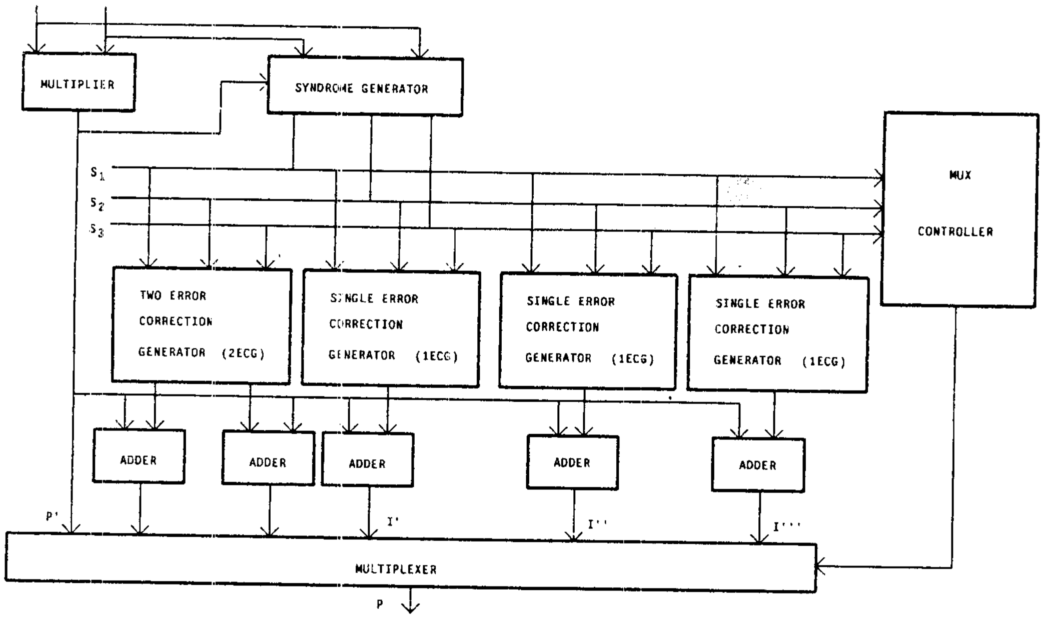
MULTIPLIER
SYNDROME GENERATOR
MUX
CONTROLLER
S_1
S_2
S_3
TWO ERROR CORRECTION GENERATOR (2ECG)
SINGLE ERROR CORRECTION GENERATOR (1ECG)
SINGLE ERROR CORRECTION GENERATOR (1ECG)
SINGLE ERROR CORRECTION GENERATOR (1ECG)
ADDER
ADDER
ADDER
ADDER
ADDER
P'
I'
I''
I'''
MULTIPLEXER
P

Figure 2: General architecture for two-error correction

two outputs of the "two-error-correction generator" will be used in the occurrence of two consecutive errors); if a fault occurs in one of the residue computing sections, the two error-free sections protect the unit against a second fault. In the sequel, only the "two-error-correction generator" will be dealt with, because the three sub-units called "single-error-correction generator" are identical to that of Fig. 1.

The triple of bases must satisfy the following constraints:

a. $S' \cap S'' = \emptyset$, where $S' = \{s_i\}$ is the set of syndromes related to single errors and $S'' = \{s_{i,j}\}$ is the set of syndromes related to double errors;
b. if a double error takes place, one residue can be zero at the most;
c. any couple of bases must be able to correct any single error.

For the *detection* of a third error, the following constraints must be satisfied:

a. $s_{i,j,k}$ must have one zero at the most;
b. $S''' \cap S'' = \emptyset$.

Some drawbacks are present, however:

1. a very large PLA has to be used to store all the correction patterns;
2. large-valued bases are to be considered to satisfy the constraint a..

For these reasons the R-Net needs to be modified; moreover, the possibility that two contemporaneous errors occur is negligible. An architecture able to overcome these problems needs further investigation of the faulty behavior of some basic arithmetic units.

6. Errors Produced by Permanent Faults

A fault does not necessarily appear at the output pins. In other words, its detection depends also on the input stream that can *mask* a fault (which is considered a "latent fault", until not detected).

A parallel multiplier is often designed as a "cascade" of two circuits: the first one, by means of an AND network, produces a "matrix" of partial products of the two operands; the second one performs the sum of these partial products (the rows of the matrix). The sum can be performed in two different ways:

a. reduction of the matrix into a two-row matrix with parallel counters (full-adders) (see [6], [24]) and subsequent sum of the two rows;
b. sum of the rows of the matrix by means of a regular, bi-dimensional array of full-adders.

A fault in the AND network (stuck-at-0 or stuck-at-1) produces, for different input operands, either a zero-error or a power-of-two error (s_i, i does not depend on the input data).

We shall consider now three different full-adders (four stages adders), namely:

**7483*;
**Lai and Muroga's full-adder* ([12]), LM-adder;
**a well-known full-adder scheme* (see Fig. 3), WK-adder.

A fault in a gate of the three adders always leads to an error which is a power of two, but with an input-dependent exponent: depending on the faulty gate, either one, two or three syndromes can be generated for all possible input patterns. We shall refer to these instances as, respectively, a *single-syndrome error*, a *double-syndrome error* and a *triple-syndrome error*.

The faulty behavior of the three adders is shown in Table 6-1 (s_i is, as usual, the syndrome of a 2^i error, while s_{-i} is the syndrome of a -2^i error). The LM and the WK adders may have a triple-syndrome error when a single fault occurs. This fact largely depends on the internal fan-out of these units; for example, the instance marked by a star occurs for a stuck-at-0 in the gate circumscribed by the block in Fig. 3; if this gate is duplicated, Fig. 4, the faulty behavior of the adder (WK_m) changes as shown in the fourth column of Table 6-1.

	Triple of residues		Single error correction with a couple of residues				Two error correction	
i	j	k	ij	ik	jk	Min.	"any kind"	cons.
3	13	23	12	22	132	12	9	10
5	7	19	12	36	18	12	12	12
5	11	19	20	36	45	20	10	20
3	23	29	22	28	308	22	11	22
7	11	17	30	24	40	24	14	24
5	19	43	36	28	63	28	12	28
7	11	29	30	84	140	30	17	30
11	13	19	60	45	36	36	13	36
11	17	19	40	45	72	40	12	40
13	23	29	132	42	308	42	13	42
5	23	59	44	116	>450	44	22	44
11	13	47	60	230	276	60	18	60
17	19	23	72	88	198	72	20	72
7	29	53	84	156	182	84	19	77
13	23	59	132	384	>450	132	25	82
19	23	29	198	252	>450	198	25	89
11	23	53	110	260	>450	110	29	92
23	29	37	308	396	126	126	39	126
29	31	53	140	182	260	140	20	140
11	37	59	180	145	>450	145	28	145
23	29	59	308	>450	>450	308	43	157
19	23	59	198	261	>450	198	28	166
19	29	53	252	>450	182	182	19	182
37	43	53	252	234	364	234	22	204
23	41	47	220	253	>450	220	30	220
29	47	59	>450	>450	>450	>450	44	228
43	47	53	322	364	>450	322	22	238
31	53	59	260	290	>450	260	45	240
23	47	59	253	>450	>450	253	43	>250
19	53	59	>450	261	>450	261	36	>250
37	47	59	>450	>450	>450	>450	39	>250
47	53	59	>450	>450	>450	>450	39	>250

Table 5-1: Correction range for various triples of residues.

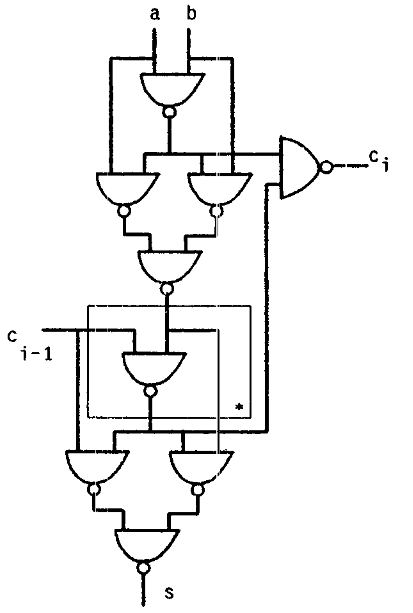

Figure 3: Original scheme: the block contains a gate with high fan-out

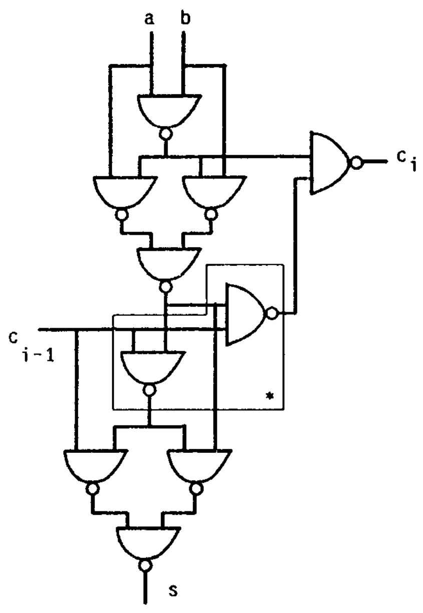

Figure 4: Modified scheme of the adder shown in Fig. 3

7. Two Consecutive Error Correcting Architectures[4]

The architecture depends on the number of different errors produced by a permanent fault. A structure including a device with a single-syndrome error (as Lai and Muroga's full adder with only a possible stuck-at-0) will be considered. Then a double-syndrome error device (as a multiplier implemented with 7483 full-adders) and a triple-syndrome error device (as a multiplier implemented with Lai and Muroga's full-adders or with full-adders as in Fig. 3) will be considered.

[4]In this section only *permanent* faults are considered.

SINGLE-SYNDROME ERROR

The architecture is shown in Fig. 5; the complete discussion of the behavior of the unit is presented in appendix. s is a triple of syndromes s_1, s_2 and s_3.

If the structure of Fig. 5 is compared with that of Fig. 1, properly extended by the considerations developed in section 4, it has two small PLA's able to correct a single error. The choice of the three bases,

	7483	LM-adder	WK-adder	WK_m - adder
stuck-at-0	s_i s_i or s_{-i}	s_i	s_i s_i or s_{-i} s_i or s_{i+1} s_i or s_{-i} or s_{i+1} *	s_i s_i or s_{-i} s_i or s_{i+1}
stuck-at-1	s_i s_i or s_{-i} s_i or s_{i+1}	s_i s_i or s_{-i} s_i or s_{i+1} s_i or s_{-i} or s_{i+1}	s_i s_i or s_{-i}	s_i s_i or s_{-i}

Table 6-1: Faulty Behavior of Four Adders

necessary to compute the syndrome, is not completely unconstrained, the following conditions having to be satisfied:

1. the syndrome triple s can have only one zero-element, two zeros indicating a fault in the R-Net;
2. any residue-pair must be capable of correcting a single error in the multiplier;
3. the elements of S' must be completely different from the corresponding elements of S", to distinguish a single error from a double error:

$$s_i \neq s_{j,k} \quad \forall i,j,k; \ -n \leq i,j,k \leq +n$$

4. All the elements of S' must be different, to be able to correct a single error:

$$s_i \neq s_j \quad \forall i,j; \ -n \leq i,j \leq +n$$

5. The i-th column $S''_i = \{s_{i,j}\}$ $(-n \leq j \leq +n)$ of the matrix S" must have all different elements in order to be able to correct the second error. However, this condition can be derived from the previous ones, in fact:

Proof

If $s_{i,j} = s_{i,k}$ then

$$[2^i + 2^j]_b = [2^i + 2^k]_b,$$

but $s_j = s_k$, and S' would have two equal elements.

DOUBLE-SYNDROME ERROR

The architectural approach is similar to one previously introduced (see Fig. 6). Only the core part of the structure is shown. Two registers (RSC1 and RSC2) store the two different syndromes s_i and s'_i and the corresponding corrections c_i and c'_i due to the first fault. The behavior of the unit is fully described in [1].

As in the previous section, the bases have to satisfy certain conditions. The same conditions related above are still valid, namely:

1. The $s_{i,j}$ syndrome does not present two zeros.
2. Any residue-pair must be able to correct one error.
3. $S' \cap S'' = \emptyset$, so as to distinguish single errors from double errors.
4. $s_i \neq s_j$, so as to distinguish all different single errors.
5. the column S'_j must have all its own different elements.

However, two new conditions have to be satisfied:

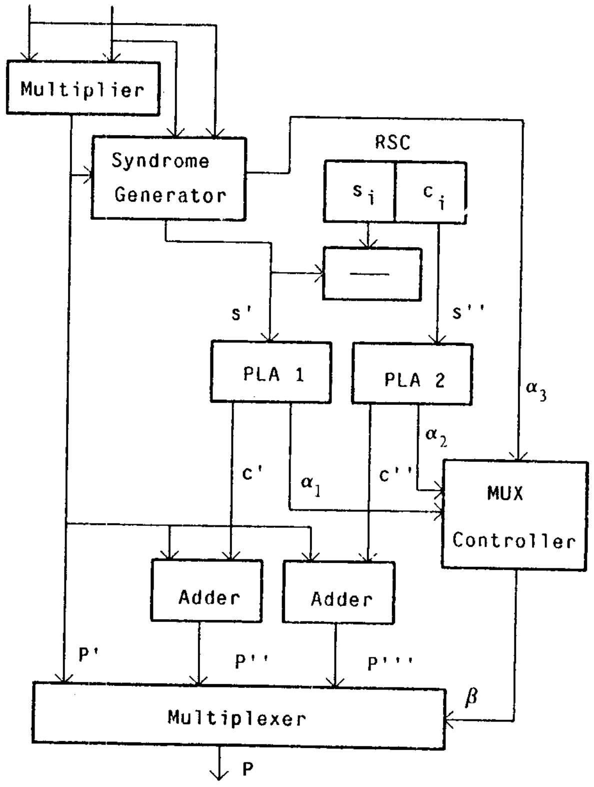

Figure 5: R-Net for two consecutive error correction: single-syndrome error family.

6. The elements of the two columns S''_i and S''_{i+1} must be completely different.
7. The elements of the two columns S''_i and S''_{-i} must be completely different.

Conditions 6. and 7. do not impose more constraints, since they can be derived from the previous ones, in fact:

***Condition 6.**

Proof

If $s_{i,j} = s_{i+1,k}$ then:

$$[2^i + 2^j]_b = [2^{i+1} + 2^k]_b$$

$$[2^j]_b = [2^{i+1} - 2^i + 2^k]_b =$$

$$= [2^i + 2^k]_b,$$

consequently:

$s = s_{i,k}$ and $S' \cap S'' \neq \emptyset$.

***Condition 7.**

Proof

If $s_{i,j} = s_{-i,k}$ then:

$$[2^i + 2^j]_b = [-2^i + 2^k]_b$$

thus

$$[2*2^i + 2^j]_b = [2^k]_b$$

and ,consequently:

$s_{i+1,j} = s_k$ and $S' \cap S'' \neq \emptyset$.

A problem can arise when the two external columns are considered (S''_n and S''_{-n}); in this case the number of bit of the result P' must be reduced by one unit.

In Table 5-1 some interesting cases are presented; it shows also some advantages, concerning the dimension of the multiplier, if compared with the approach referred in section 5. Both consecutive and "any kind" errors are taken into account.

TRIPLE-SYNDROME ERROR

The architecture is similar to that shown in Fig. 6 with one only difference in a register RSC3 and in a PLA 4. The selection of the triple of bases imposes that the above conditions must be still satisfied; a new condition, however, has to be considered:

The elements of the three columns S''_i, S''_{-i} and S''_{i+1} must be distinct.

This condition cannot be derived from the previous ones and reduces the number of result bits that can be corrected. The more complex architecture of the R-Net and the degradation in performance can justify the choice of full-adders with low fan-out, see Fig. 4.

Other tables, referring to single-syndrome and triple-syndrome cases, are shown in [1], together with the content of each PLA for all the three cases.

8. Three or more Consecutive Error Correction Architecture

All the assumptions presented in the previous sections are still valid; only a three-error case will be considered. For more than three errors it will be sufficient to extend the results. In order to compute the syndrome, *four* bases are necessary and must satisfy the following conditions:

a. The four bases must be able to correct *three* errors in the multiplier.

b. Any triple of bases, if an error takes place in the syndrome generator, must be able to correct two errors in the multiplier.

c. If a second error takes place in the syndrome generator, any couple of bases must be able to correct one error in the multiplier.

The structure able to cope with three errors of any possible kind ("any kind"-error) is simply an up-grading of the structure in Fig. 2; it needs one "error correcting generator (ECG)" to correct three errors, four

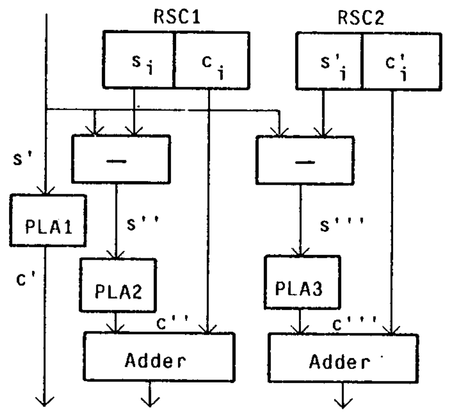

Figure 6: R-Net for two consecutive error correction: double-syndrome error family.

ECG's to correct two errors and six ECG's to correct one single error. This structure, thus, is fairly expensive. If we consider the errors as consecutive, the structure of Fig. 7 is obtained.

The four bases correct three consecutive errors s_i, s_j and s_k; the circuit of Fig. 7 is an extension of that of Fig. 6. The syndromes of the first two errors are stored, together the correspondent corrections c_i and c_j, in RSC1 and RSC2. After a second error has occurred, if a third error takes place, we have the following possibilities:

error	correction performed by
s_i	PLA 1
s_j	PLA 1
s_k	PLA 1
$s_{i,j}$	PLA 2 or PLA 3
$s_{i,k}$	PLA 2
$s_{j,k}$	PLA 3
$s_{i,j,k}$	PLA 4

The R-Net in Fig. 7 is fairly complex; the increase of hard-core may lead to a unit globally less reliable than a unit with a more limited correction range. A three errors correction can be therefore considered the limit of applicability of this technique.

9. Summary

The paper presents a methodology useful for designing arithmetic devices satisfying severe constraints of reliability. The goal was achieved by suitably matching the properties of the Residue Number System with the architectural aspect of the problem.

The main emphasis is in multiple error correction; we show that a three errors correcting architecture can be considered the limit of applicability of the technique, a wider correction range leading to unreasonably complex structures.

The architectural approach is particularly oriented to the VLSI design, by performing several functions with standard devices, such as PLA's, and featuring regular interconnection schemes. Moreover, 70% of the remaining structure is built out of one simple cell, i.e. a full-adder. A custom device is presently under development, using N-channel, MOS technology, $\lambda = 1.5$ micron. The unit is a 23 x 23 bit multiplier.

An interesting result of the approach described in the present paper is the negligible increase in global execution time, when the unit is working correctly. If a fault occurs, the speed is degraded by the adder before the multiplexer, which can be however carefully designed to guarantee O(*log n*) computation time.

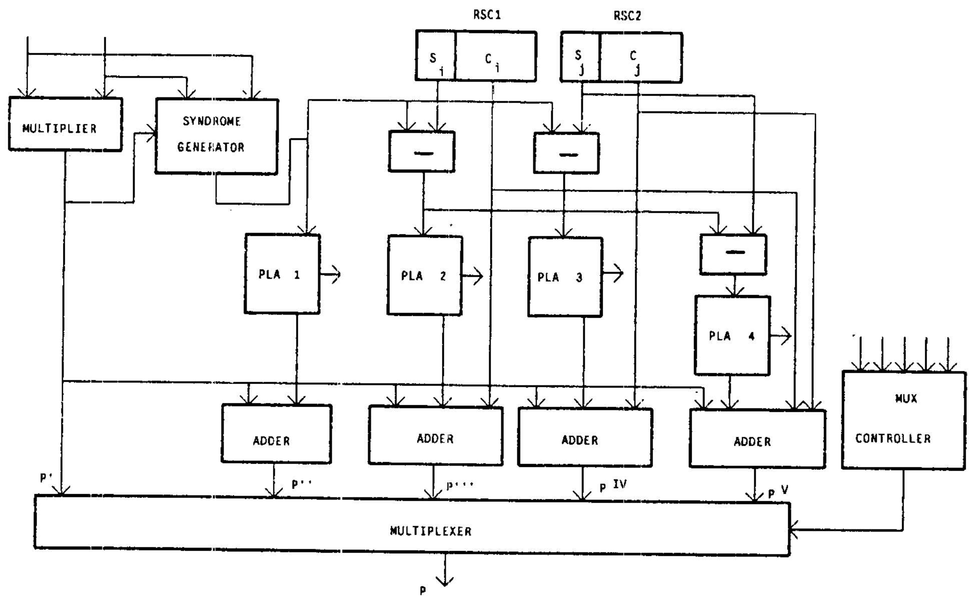

Figure 7: R-Net for three consecutive error correction

Some problems are still open and deserve further investigation. The faulty behavior of recoded multipliers should be studied: they are faster and smaller (but with more complex wiring and less regular structure) than the iterative cellular array multipliers we have taken into account. The faulty behavior of carry-lookahead adders should also be considered.

Only at that point, a whole fault-tolerant arithmetic unit, featuring acceptable performance, could be implemented in a single chip.

Appendix

The behavior of the architecture shown in Fig. 5 is now discussed, the assumption being that the multiplier belongs to the "single-error syndrome family".

a. *Correct behavior*: the three elementary syndromes are equal zero and $\alpha_3 = 0$; the multiplexer controller generates a signal $\beta = 0$ and the multiplexer chooses the P' input.

b. *First detection of an error* (obviously, single) in the multiplier (fault and related error taking place in the syndrome generation network will not be discussed, see section 4 and Fig. 2).
A syndrome $s' = s_i$ is produced: it contains three elementary non-zero syndromes s_{i1}, s_{i2} and s_{i3} ($\alpha_3 = 1$). The PLA 1 produces the correction $c' = c_i$ and the adder on the left produces the correct result P''; a signal α_1 is generated by PLA 1; this signal is one if and only if the syndrome s belongs to the set of addresses of the PLA 1 itself[5]. The multiplexer controller ($\alpha_3 = 1$, $\alpha_1 = 1$) selects the second input P''; the same time the syndrome s_i and the corresponding correction c_i are stored into the RSC register (note that some extra hardware, not shown in Fig. 5, has to be added: a bistable element is set to '1' at the first occurrence of an error; if $s \neq 0$ and the bistable element has a '0' stored, the unit is forced to copy s and c in RSC).

c. *Further single errors*: the operations are similar to those referred in **b.** (correction produced by PLA 1 and selection of the P'' input in the multiplexer). The new syndrome and correction, differently from the previous case, are not stored into the RSC register.

d. *Second error occurrence*: a syndrome s_j is produced if the first error has not influenced the result (zero-error); on the other hand, a syndrome $s_{i,j}$ is produced if both faults contribute to a wrong result. The first case is similar to point c., because it is considered as a single error. In the second case, PLA 1 does not recognize $s_{i,j}$ as a single error ($\alpha_1 = 0$) and, as:

$$s_{i,j} = s_i + s_j,$$

we have:

$$s'' = s_j \text{ and } c'' = c_j.$$

PLA 2 recognizes ($\alpha_2 = 1$) only the syndromes deriving from a single fault. The multiplexer controller receives $\alpha_3 = 1$ (two non-zero syndromes, at minimum), $\alpha_1 = 0$ (it indicates the presence of a syndrome $s_{i,j}$, i.e. a non-single-error) and $\alpha_2 = 1$ ($s_j = s_{i,j} - s_i$: single error); if such condition is met, it forces $\beta = 2$ in order to select the third input P'''.

e. *More than two errors*: if $\alpha_1 = 0$ and $\alpha_2 = 0$ then the error cannot be considered single and does not contain the i-th error; in this instance a third error has occurred; this error is not corrected but only detected.[6]

[5]The PLA 1 recognizes only all the single errors

[6]For some error occurrences, the PLA's can correct errors in a wrong way; this fact is caused by previous errors that took place in the PLA's; for this reason the PLA's need suitable protection. A possible solution is to store into any PLA the correction amount, *its residue* and the address of the proper decode section.

References

[1] Annaratone, M. and Stefanelli, R.
General Approach to Fault-tolerant Arithmetic Unit Design by Means of Multi-residue Codes.
1983.
Technical Report in preparation.

[2] Avizienis, A.
Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design.
IEEE Trans. on Computers C-20(11):1322-1331, November, 1971.

[3] Avizienis, A.
Low-Cost Residue and Inverse Residue Error Detecting Codes for Signed-Digit Arithmetic.
In *Proc. of the 5th Symposium on Computer Arithmetic*, pages 165-168. IEEE, IEEE, May, 1981.

[4] Bose, B. and Rao, T.R.N.
Unidirectional Error Codes for Shift Register Memories.
In *Proc. FTCS-10*, pages 26-28. IEEE, IEEE, October, 1980.

[5] Brown, D.T.
Error Detecting and Correcting Binary Codes for Arithmetic Operations.
IRE Trans. on Electronic Computers :333-337, September, 1960.

[6] Dadda, L.
Some Schemes for Parallel Multipliers.
Alta Frequenza 34:349-356, May, 1965.

[7] Gajsky, D.D. and Vora, C.
High-speed Modulo-3 Generator.
Electronics Letters 13(25):770-772, December, 1977.

[8] Garner, H.L.
The Residue Number System.
IRE Trans. on Electronic Computers :140-147, June, 1959.

[9] Garner, H.L.
Error Codes for Arithmetic Operations.
IEEE Trans. on Electronic Computers EC-15(10):763-770, October, 1966.

[10] Goto, M.
Rates of Unidirectional 2-Column Error Detectable by Arithmetic Codes.
In *Proc. FTCS-10*, pages 21-25. IEEE, IEEE, October, 1980.

[11] Jayshree, T. and Basu, D.
On Binary Multiplication Using the Quarter Square Algorithm.
IEEE Trans. on Computers C-25(9):957-960, September, 1976.

[12] Lai, H.C. and Muroga, S.
Minimum Parallel Binary Adders with NOR (NAND) Gates.
C-28(9):648-659, September, 1979.

[13] Liu, C.K. and Wang, T.L.
Error-Correcting Codes in Binary-Coded-Decimal Arithmetic.
IEEE Trans. on Computers C-27(11):977-984, November, 1978.

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

[15] Newmann, P.G. and Rao, T.R.N.
Error-Correcting Codes for Byte-Organized Arithmetic Processors.
IEEE Trans. on Computers C-24(3):226-232, March, 1975.

[16] Parhami, B. and Avizienis, A.
Application of Arithmetic Error Codes for Checking of Mass Storage.
In *Proc. FTCS-3*, pages 47. IEEE, IEEE, June, 1973.

[17] Parhami, B. and Avizienis, A.
Detection of Storage Errors in Mass Memories Using Low-Cost Arithmetic Codes.
IEEE Trans. on Computers C-27(8):302, August, 1978.

[18] Rao, T.R.N.
Error-Checking Logic for Arithmetic-Type Operations of a Processor.
IEEE Trans. on Computers C-17(9):845-849, September, 1968.

[19] Rao, T.R.N.
Biresidue Error-Correcting Codes for Computer Arithmetic.
IEEE Trans. on Computers C-19(5):398-402, May, 1970.

[20] Rao, T.R.N. and Garcia, O.N.
Cyclic and Multiresidue Codes for Arithmetic Operations.
IEEE Trans. on Information Theory IT-17(1):85-91, January, 1971.

[21] Svoboda A.
Rational Numerical System of Residual Classes.
Stroje Na Zpacovani Informaci , 1957.

[22] Takeda, K. and Tohma, Y.
Logic Design of Fault-Tolerant Arithmetic Units Based on the Data Complementation Strategy.
In *Proc. FTCS-8*, pages 348-350. IEEE, IEEE, October, 1980.

[23] Wakerly, J.F.
Detection of Unidirectional Multiple Errors Using Low-Cost Arithmetic Codes.
IEEE Trans. on Computers C-24(2):210, February, 1975.

[24] Wallace, C.S.
A Suggestion for a Fast Multiplier.
IEEE Trans. on Electronic Computers EC-13(2):14-17, February, 1964.

[25] Watson, R.W. and Hastings, C.W.
Self-Checked Computation Using Residue Arithmetic.
IEEE Proceedings 54(12):1920-1931, December, 1966.