J. Arlat
W. C. Carter

Implementation and Evaluation of a (*b,k*)-Adjacent Error-Correcting/Detecting Scheme for Supercomputer Systems

This paper describes a coding scheme developed for a specific supercomputer architecture and structure. The code considered is a shortened (b,k)-adjacent single-error-correcting double-error probabilistic-detecting code with $b = 5$, $k = 1$, and code group width $= 4$. An evaluation of the probabilistic double-error-detection capability of the code was performed for different organizations of the coding/decoding strategies for the codewords. This led to the selection of a system organization encompassing the traditional feature of memory data error protection and also providing for the detection of major addressing errors that may result from faults affecting the interconnection network communication modules. The cost of implementation is a limited amount of extra hardware and a negligible degradation in the double-error-detection properties of the code.

Introduction

The requirement of superfast performance for large-scale supercomputing systems has led to the development of new types of highly parallel architectures: tens (or more) of processors concurrently working on different parts of the same problem. This trend has been necessitated by the speed limitations of computer technologies, such as gallium arsenide or Josephson junction devices, which can increase computer speed only by about a factor of 10 with current uniprocessor architectures [1, 2], whereas multiprocessor architectures have the potential of providing much larger speed increases, from 100 to 1000 times the current speed.

With these new architectures, the analysis of systems such as fusion power reactors, turbulent flow around ships or planes, and weather analysis and prediction would be possible on a three-dimensional basis, applications that are practically unmanageable with today's computer performance. Also, it should be noted that the solution of such 3-D models would require not only ultrahigh speed but also a very large primary memory of capacity up to 10^{10} words [3]. With such large-capacity memories, the problem of data protection against the effects of errors becomes one of great significance requiring particular attention.

The study presented in this paper constitutes a contribution to this problem; it has been developed in the framework of a large ongoing project, currently funded in France, for research and development in supercomputer systems. A core machine is currently being developed to serve as a basis upon which more powerful systems can be built. It is a *multiple-instruction-stream, multiple-data-system* (MIMD) machine intended for scientific computation and designed to act as an array processor when connected to a host computer system. This machine is expected to provide an average computation speed of more than twenty million floating-point operations per second (20 MFLOPS). The development of a machine aimed at an average computation speed in excess of 100 MFLOPS and featuring more than 10^8 words of primary memory is being planned.

Clearly, with such a structure, the error-tolerance characteristics of main memory have to be carefully investigated. Along these lines, a preliminary study was performed to identify and apply possible solutions in the case of the core machine. Although more restricted in size, this structure features most of the characteristics to be considered for the development of a coding scheme that could be applied in future upgraded systems. This study and the results obtained are presented in this paper.

For this computer system, the choice of the code to protect information depends more upon the structural features of the core machine than upon the main memory features, so the core computer system will be briefly described. As shown in

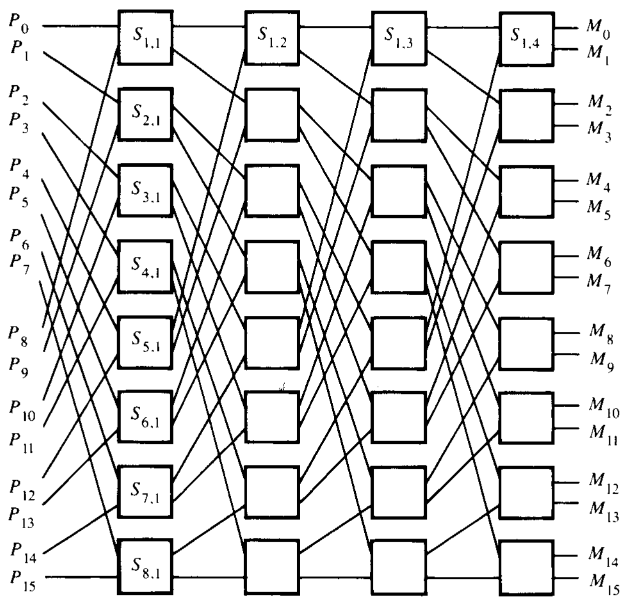

Figure 1 Symmetric $N \times N$ Omega network, with $N = 16$.

Figure 1, the core machine consists of 16 processor modules (P_0, P_1, $\cdots$, P_{15}) interconnected, by means of a symmetric Omega-type interconnection network [4], to 16 memory modules (M_0, M_1, $\cdots$, M_{15}) which constitute the primary memory of the system. The processor modules are composed of standard array processors. Each such processor has a small local memory protected by a single-error-correcting, double-error-detecting (SEC-DED) modified Hamming code [5–7]. The Omega-type network was selected in order to provide an acceptable tradeoff between the amount of hardware and the necessary control complexity, flexibility, rearrangeability, delay, etc. [8, 9]. The memory modules are characterized by a capacity of 1M words of 64 bits each. These modules are composed of four banks of 64 data modules, each data module one bit wide and containing 256K bits. Initially, a SEC-DED modified Hamming code was envisioned for data protection.

In practice, the conceptual network of Fig. 1 is modified as indicated in **Figure 2.** Each network ensures bidirectional switching of B lines for each terminal module; these lines represent address, data, control bits, etc. Simultaneous switching of these B bits may be obtained by the superposition of B identical 1-bit-wide slices. However, for efficiency, specific circuits allowing the switching of b bits in parallel have been developed [10]. The network can thus be viewed as the superposition of $\lceil B/b \rceil$ slices ($\lceil B/b \rceil$ represents the least integer greater than or equal to B/b) logically equivalent to the $N \times N$ Omega network illustrated in Fig. 1 (where $N = 16$).

The actual structure being implemented is characterized by $N = 16$, $B = 100$, and $b = 4$. Each slice is made up of four identical stages composed of eight (8×8) switching modules, each of them being able to take two states: the through state and the cross state. The total number of switching modules in general is $(N/2)\ \lceil B/b \rceil\ \log_2 N$, so in this case 800 switching modules are used.

The main structural modifications intended in order to reach higher performance levels are 1) increasing the number of primary memory modules to about one hundred and providing double access capability, 2) including a stage of specific computing modules, each connected to a memory module, working in single-instruction-stream, multiple-data-stream (SIMD) mode, and 3) extending the network connecting primary memory modules with the array processor modules working in MIMD mode.

The major influences of these modifications on the problem of primary memory protection are related to 1) the increase in the number of memory modules and in the resulting capacity of the primary memory, and 2) the increase in the size of the network. Both factors would tend to make the system much less reliable and available. It thus appears that basically the same problems have to be solved for both structures in order to derive an efficient error-correcting scheme. This allows methods which may provide sufficiently high system availability for the upgraded machine to be tested and evaluated on the core machine.

Determining the error correction and detection techniques for the data transmission and storage subsystem of the core system began with a study of the effects of the failures of single components. Both the local memories in the sixteen processors and the memory modules will be implemented by 1-bit-wide data modules; therefore, single failures will produce single errors, and standard Hamming SEC-DED codes would suffice. However, as stated previously, for circuit efficiency in the Omega network, four bits are switched in parallel. Thus, a single component failure in one slice of the Omega network could produce any one of the fifteen possible error patterns in the four bits of data, address, or control information being transmitted through that slice. Therefore, a single-bit SEC-DED code is inappropriate for transmissions between the processors and the main storage modules. A solution is to use a code which is SEC-DED for groups of four bits [11].

A practical implementation consideration was that the planned SEC-DED code for the 64-bit-wide data words in the main memory uses eight check bits, and any code chosen for implementation should use a comparable number of check bits, as well as a comparable amount of circuitry and number of logic levels. In addition, the code chosen must have error correction and detection power near that of a single-bit SEC-DED code, but over groups of four bits.

The usual solution for error correction and detection over groups of b bits is to use a Reed-Solomon b-adjacent code

[11, 12]. The implementations of encoding/decoding circuitry in [11] and [12] are based on arithmetic in the finite field $GF(2^b)$ and are serial in nature. Cocke [13] showed that the encoding/decoding operations could be performed in the prime field [usually $GF(2)$; thus ordinary Boolean algebra can be used], and Bossen [14] provided an efficient parallel implementation of such codes. The practical difficulty with such b-adjacent codes is that if b-bit-wide SEC-DED is desired, then $3b$ check bits are necessary and the maximum data length for the $3b$ check bits is

$$m_{\max} = b(2^b - 1). \tag{1}$$

If $b = 4$, then 12 check bits are necessary, and only 60 data bits can be covered by such a code. Carter, Wadia, and Hsieh [15, 16] introduced (b,k)-adjacent codes with a parallel encoding technique. These codes use $2b + k$ bits for a SEC code and $3b + 2k$ bits for a SEC-DED code for groups of data bits which are $b - k$ bits wide. As pointed out in [16], this type of code was primarily designed to overcome the inherent data length limitation of b-adjacent codes. A (b,k)-adjacent SEC code has a code group width $q = b - k$, $2b + k$ check bits, and the maximum number of data bits handled is

$$m' = q(2^b - 1). \tag{2}$$

The addition of k check bits increases the maximum number of data bits that can be handled by more than a factor of 2^k. If $b = 5$ and $k = 1$, then data in 4-bit-wide groups will be corrected, and the maximum length of data which may be corrected is 4(31) = 124 bits.

The difficulty of detecting errors in two distinct b-bit-wide groups may be solved as follows. It is well known that error-correction codes operating on shortened data have a probability of detecting multiple errors which are not corrected. The ability of b-adjacent and (b,k)-adjacent codes to perform such detection was discussed in [15, 17] and was seen to be quite high. The double-error detecting ability of a shortened SEC code depends on the shortened code chosen from the SEC code of maximum length. For a (b,k)-adjacent SEC code of length qd there are

$$\binom{2^b - 1}{d} \text{ possible choices.}$$

A shortened (b,k)-adjacent SEC code with $b = 5$ and $k = 1$ can correct all 15 error patterns in 4-bit-wide groups, can cover as information the 84 bits needed for parallel transmission of the 64 data bits and 20 address bits used in the core machine, can detect many double errors involving two 4-bit-wide data groups (with arbitrary data patterns), and uses only nine check bits for each word in main memory. Since such a code satisfies all the practical system requirements for availability, except possibly high enough probability of double-error detection, it was decided to investigate the properties of such a code, devise and compare specific codes, and determine if

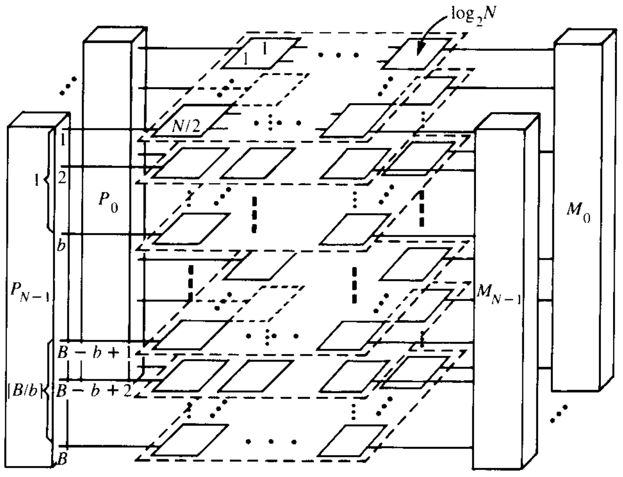

Figure 2 Structure of the system.

the probabilistic four-adjacent double-error detection would be sufficient to fulfill the core system availability requirements, considering the relative probability of occurrence of single and multiple four-adjacent errors. The shortened length chosen must have length 4 × 21. There are

$$\binom{31}{21} = 44\ 352\ 165 \text{ possible choices for such a code.}$$

In the following sections of this paper we first describe the procedure for generating the specific code as well as that of exploiting the syndrome to identify the error. The double-error-detection properties of the code are then examined, using a systematic investigation of the error combinations that allow evaluation of the double-error-detection probability. Following that, different implementations of the code are examined and evaluated on the basis of their double-error-detection capabilities. These implementations differ by 1) the coding scheme, which may include only data lines or both data and address lines in the generation of the check bits, the latter taking into account faults affecting the commutation modules that convey the address lines in the network; and 2) the decoding scheme, which, depending on the use of the redundancy of the code, may correct all bits of the codeword or may correct data and check bits only, to satisfy different architectural requirements.

Description of the code

- *General background*

Let $q = b - k$ be the code group bit width for a (b,k)-adjacent code. The parity check matrix for such a code can be written as

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_q & \mathbf{I}_q & \cdots & \mathbf{I}_q & \mathbf{I}_q & \mathbf{O}_{q,b} \\ \mathbf{C}^0_{b,q} & \mathbf{C}^1_{b,q} & \cdots & \mathbf{C}^m_{b,q} & \mathbf{O}_{b,q} & \mathbf{I}_b \end{bmatrix}, \tag{3}$$

Table 1 Correspondence between shortened and complete matrices.

$C'^{i}_{5,4}$	(i)	0	1	2	3	4	5	6	7	8	9	10
C^{j}_{5}	(j)	0	1	2	3	4	30	29	28	27	17	18
$C'^{i}_{5,4}$	(i)	11	12	13	14	15	16	17	18	19	20	$\cdots$
C^{j}_{5}	(j)	16	19	26	5	6	7	21	20	8	9	$\cdots$

where $\mathbf{I}_q$ and $\mathbf{O}_y$ represent the identity and null square matrices of dimension y, $\mathbf{O}_{y,z}$ corresponds to the y,z-dimension null matrix, and the $\mathbf{C}^{i}_{b,q}$, $0 \le i \le m$, represent the b,q-dimension matrices obtained by deleting the same set of k columns from the b,b matrices $\mathbf{C}^{j}_{b,b}$, $0 \le j \le 2^b$, corresponding to the powers over $GF(2)$ of the nonzero elements of $GF(2^b)$ [15, 16].

These matrices represent the distinct powers of the companion matrix $\mathbf{C}_b = \mathbf{C}^{1}_{b,b}$ of the primitive polynomial of degree b,

$$p_b(x) = t_0 + t_1 x + \cdots + t_{b-1}x^{b-1} + x^b,$$

$$\mathbf{C}_b = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & t_0 \\ 1 & 0 & & 0 & 0 & t_1 \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 0 & 0 & \cdots & 0 & 1 & t_{b-1} \end{bmatrix}, \tag{4}$$

and are characterized by the following properties:

1. $\mathbf{C}^{i}_{b} = (\mathbf{Y}^{i}, \mathbf{Y}^{i+1}, \cdots, \mathbf{Y}^{i+b-1})$, where the $\mathbf{Y}^{j}$, $0 \le j \le 2^b - 2$, represent the column vectors of the successive powers of the root $p(x)$: Y.
2. $\mathbf{C}^{i}_{b}\mathbf{Y}^{j} = \mathbf{Y}^{i+j}$, for $i, j, 0, 1, \cdots, 2^b - 2$.
3. If $\mathbf{Y}^{i} + \mathbf{Y}^{j} = \mathbf{Y}^{m}$, then $\mathbf{C}^{i}_{b} + \mathbf{C}^{j}_{b} = \mathbf{C}^{m}_{b}$, for $i, j, m \in \{0, 1, \cdots, 2^b - 2\}$.

These properties of the companion matrix allow for an easy derivation of its distinct powers, since $w = 2^b - 1$, $\mathbf{C}^{0}_{b} = \mathbf{C}^{w}_{b} = \mathbf{I}_b$ [16].

• *Construction of the code and derivation of the decoding expressions*

As previously mentioned, a consideration of the possible errors which may occur because of the architecture of the supercomputers being designed shows that a single fault in the Omega-type interconnection network can cause from one to four faults in a group of four adjacent bits in a memory unit. Thus, for single-error correction, any of the 15 possible error patterns in such four-adjacent groups must be corrected. The supercomputers have a word length of 64 bits, which must be divided into 4-bit code groups. A standard b-adjacent code with $b = 4$ cannot be used since the maximum data length for such a code is 60 bits. Thus a (b,k)-adjacent code with $b = 5$ and $k = 1$ was chosen. This code handles code data groups of width $q = b - k = 4$, and has a maximum data length of 124 bits. This flexibility in length is exploited, as described in subsequent sections.

The specific check matrix is easily deduced from the general matrix (3) and can be written as

$$\mathbf{H}'' = \begin{bmatrix} \mathbf{I}_4 & \mathbf{I}_4 & \cdots & \mathbf{I}_4 & \mathbf{I}_4 & \mathbf{O}_{4,5} \\ C'^{0}_{5,4} & C'^{1}_{5,4} & \cdots & C'^{r}_{5,4} & \mathbf{O}_{5,4} & \mathbf{I}_5 \end{bmatrix}, \tag{5}$$

where r is determined by the number of data bits that are considered by the relation $r = \lceil m/b \rceil - 1$, $(r \le 30)$.

The companion matrix is derived from the primitive polynomial

$$p(x) = 1 + x^2 + x^5, \tag{6}$$

and all the distinct powers can be easily obtained following the previously described method. In practice, the choice of the $\mathbf{C}'^{i}_{5,4}$ among the $\mathbf{C}^{j}_{5}$ can be made arbitrarily, provided that in all cases the same column is deleted. Nevertheless, it has to be noted that this choice presents a direct influence on 1) the hardware implementation complexity of the code, and 2) its multiple group error detection abilities. The constraints related to the latter condition cannot be simply identified at this stage and are addressed in the next section. Thus, the selection of the matrices was based on the first influence only and ended in the correspondence between matrices C^{j}_{5} and $C'^{i}_{5,4}$ indicated in **Table 1,** where the $C'^{i}_{5,4}$ are actually obtained by deletion of the fifth column in the associated C^{j}_{5}. Note that only $C'^{i}_{5,4}$ matrices, which are used in the different implementations of the code, have been considered here.

The logical equations for the generation of the check bits can be derived easily from the matrix $\mathbf{H}''$. In the following, B_i is used in place of $C'^{i}_{5,4}$, for conciseness of the presentation; furthermore, the symbols + and Σ are used to represent addition modulo-2, i.e., exclusive-or, while the symbols $\cdot$ and $\cap$ represent the logical AND operation, v and U represent the logical OR, and $\overline{}$ represents negation.

$$c_j = \sum_i d_{ij}, \qquad 0 \le j \le 3,$$

$$(c_4, c_5, c_6, c_7)^T = \sum_i \mathbf{B}_i(d_{i0}, d_{i1}, d_{i2}, d_{i3})^T, \tag{7}$$

where $(d_{i0}, d_{i1}, d_{i2}, d_{i3})^T$ represents the transposed vector of the data bits of group i, for $i = 0, 1, \cdots, r$.

The bits of the syndrome vector (s_0–s_7) are then identified by

$$s_j = c'_j + \sum_i d'_{ij}, \qquad 0 \le j \le 3,$$

$$(s_4, s_5, s_6, s_7, s_8)^T$$
$$= (c'_4, c'_5, c'_6, c'_7, c'_8)^T + \sum_i \mathbf{B}_i(d'_{i0}, d'_{i1}, d'_{i2}, d'_{i3})^T,$$
$$i = 0, 1, \cdots, r, \qquad (8)$$

where d'_{ij} and c'_k, $0 \le k \le 8$, have been used instead of d_{ij} and c_k to account for possible errors that could affect data and check bits. Considering an error vector $(e_{j0}, e_{j1}, e_{j2}, e_{j3})^T$, corresponding to any error combination that affects data group $j, j = 0, 1, \cdots, r$, then the following expressions apply:

$$(s_0, s_1, s_2, s_3)^T = (e_{j0}, e_{j1}, e_{j2}, e_{j3})^T, \qquad (9)$$

$$(s_4, s_5, s_6, s_7, s_8)^T = \mathbf{B}_j(s_0, s_1, s_2, s_3)^T. \qquad (10)$$

Equation (9) allows the determination of faulty bits in a group that can be identified by use of Eq. (10); the associated group pointer is determined by

$$G_j\colon (s_4, s_5, s_6, s_7, s_8)^T = \mathbf{B}_j(s_0, s_1, s_2, s_3)^T. \qquad (11)$$

It has been shown [16] that if an error exists in group j, $G_j = 1$ and $G_i = 0$ for all $i \ne j$. Correction of data bits of group i is then obtained from the following equation:

$$d_{ijc} = d_{ij} + G_i s_j, \qquad 0 \le j \le 3. \qquad (12)$$

Error control on check bits can be implemented according to the following relations. An error vector $(e_0, e_1, e_2, e_3)^T$ affecting check bit group 1 (c_0–c_3) results in

$$U_j s_j = 1, \qquad \bigcap_k s_k = 0, \qquad 0 \le j \le 3, \qquad 4 \le k \le 8,$$

which leads to the determination of the associated group pointer as

$$G_{c1} = (s_0 \vee s_1 \vee s_2 \vee s_3)\cdot(\bar{s}_4\,\bar{s}_5\,\bar{s}_6\,\bar{s}_7\,\bar{s}_8). \qquad (13)$$

In the same way, it can be verified that pointers for check bit group 2 (c_4–c_7) and group 3 (c_8) are defined as

$$G_{c2} = (\bar{s}_0\,\bar{s}_1\,\bar{s}_2\,\bar{s}_3)\cdot(s_4 \vee s_5 \vee s_6 \vee s_7)\bar{s}_8\,,$$

$$G_{c3} = \bar{s}_0\,\bar{s}_1\,\bar{s}_2\,\bar{s}_3\,\bar{s}_4\,\bar{s}_5\,\bar{s}_6\,\bar{s}_7\,s_8\,. \qquad (14)$$

Correction of the check bits is then obtained using the following equations:

$$c_{jc} = c_j + G_{c1}\cdot s_j = c_j + s_j\,\bar{s}_4\,\bar{s}_5\,\bar{s}_6\,\bar{s}_7\,, \qquad 0 \le j \le 3,$$

$$c_{kc} = c_k + G_{c2}\cdot s_k = c_k + s_k\,\bar{s}_1\,\bar{s}_2\,\bar{s}_3\,\bar{s}_8\,, \qquad 4 \le k \le 7,$$

$$c_{8c} = c_8 + G_{c3}\cdot s_8 = c_8 + \bar{s}_0\,\bar{s}_1\,\bar{s}_2\,\bar{s}_3\,\bar{s}_4\,\bar{s}_5\,\bar{s}_6\,\bar{s}_7\,s_8\,. \qquad (15)$$

Evaluation of the double-error-detection probability

In order to evaluate the efficiency of the code, it is important to characterize its ability to identify multiple error conditions, i.e., conditions that affect more than one group of the codeword. Furthermore, multiple-error-detection ability strongly relies on the amount of redundancy of the code that is not explicitly used for correction. As a consequence of the (b,k) code organization, no systematic way is known to determine the probability of detection of multiple errors [16].

The determination of this probability requires that a detailed study of the error combinations be performed. As a result of the inherent complexity of the procedure, and of the relative probability of multiple errors, the study reported here has been limited to the consideration of double errors only. The procedure consists in the characterization and enumeration of the various double-error combinations that would result in a wrong correction.

For example, if one considers a double-error combination $(e_{i0}, e_{i1}, e_{i2}, e_{i3})$ and $(e_{j0}, e_{j1}, e_{j2}, e_{j3})$ affecting groups i and j, respectively, the problem is to identify the cases when these two errors induce an erroneous correction on group k; i.e., they can be viewed as a single equivalent error $(e_{k0}, e_{k1}, e_{k2}, e_{k3})$. Furthermore, it has to be noted that the error combination consisting of $(e_{i0}, e_{i1}, e_{i2}, e_{i3})$, $(e_{j0}, e_{j1}, e_{j2}, e_{j3})$, $(e_{k0}, e_{k1}, e_{k2}, e_{k3})$, and all other error bits equal to zero, forms a valid codeword. Such an approach is investigated in detail hereafter for each error combination affecting the various groups of codewords. For better readability of the presentation, errors affecting data bits and check bits are considered separately and joint conditions are derived when possible.

- *Errors affecting data bit groups only*

Basic conditions for miscorrection can be stated as

$$e_{im} + e_{jm} = e_{km} \ne 0, \qquad 0 \le m \le 3, \qquad (16)$$

where subscripts i, j, and k identify distinct data bit groups only. Furthermore, pointer G_k has to be activated ($G_k = 1$), i.e.,

$$\mathbf{B}_k(s_0, s_1, s_2, s_3)^T = (s_4, s_5, s_6, s_7, s_8)^T. \qquad (17)$$

Considering relation (8) which characterizes the syndrome vector, it follows that $s_m = e_{km}$, and that

$$(s_4, s_5, s_6, s_7, s_8)^T = \mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T + \mathbf{B}_j(e_{j0}, e_{j1}, e_{j2}, e_{j3})^T,$$

which leads to

$$\mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T + \mathbf{B}_j(e_{j0}, e_{j1}, e_{j2}, e_{j3})^T$$
$$= \mathbf{B}_k(s_0, s_1, s_2, s_3)^T. \qquad (18)$$

Equations (16) and (18), as expressed above, are explicitly considering that the double-group error $(e_{im})^T$, $(e_{jm})^T$, $0 \le m \le 3$, is equivalent to the single-group error $(e_{km})^T$, which induces miscorrection of group k. Nevertheless, two more analogous conditions have to be considered, which can be obtained by rotation of subscripts i, j, and k, i.e.,

$$(e_{jm})^T + (e_{km})^T \rightarrow (e_{im})^T,$$

$$(e_{km})^T + (e_{im})^T \rightarrow (e_{jm})^T.$$

One should note that this remark holds for the other cases that are investigated hereafter in this section.

• *One single error which affects the check bit groups*
The cases corresponding to the three groups forming the check bits are described successively.

Error affecting the first group (c_0–c_3)
In this case, the conditions to be considered can be expressed as

$$e_{im} + e_{jm} = e_m \neq 0, \qquad 0 \leq m \leq 3; \tag{19}$$

and from (8), it follows that $s_m = 0$, which leads to

$$\mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T + \mathbf{B}_j(e_{j0}, e_{j1}, e_{j2}, e_{j3})^T = (0, 0, 0, 0, 0)^T. \tag{20}$$

From a practical point of view, it is important to note that conditions (19) and (20) can be expressed analogously as relations (16) and (18) respectively, provided that 1) the error vector affecting the first check bit group is represented with the same notation as for data bit groups, and the subscript 16 is introduced, i.e., $e_m = e_{16m}$, and 2) the null matrix $\mathbf{O}_{5,4}$ is included in the set of values of the shortened "**B**" matrices, i.e., $\mathbf{B}_{r+1} = \mathbf{O}_{5,4}$. The number of double-group error combinations satisfying the conditions stated by relations (17) and (19) is denoted $N_{\overline{\text{ded}_1}}$.

Error affecting the second group (c_4–c_7)
It can be easily verified that this case corresponds to

$$e_{im} + e_{jm} = 0, \qquad 0 \leq m \leq 3; \tag{21}$$

and from relation (8), it follows that

$$\mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T + \mathbf{B}_j(e_{j0}, e_{j1}, e_{j2}, e_{j3})^T = (e_4, e_5, e_6, e_7, 0)^T,$$

which leads to

$$(\mathbf{B}_i + \mathbf{B}_j)(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T = (e_4, e_5, e_6, e_7, 0)^T. \tag{22}$$

Error affecting the third group (bit c_8)
The conditions that need to be satisfied here are defined by

$$e_{im} + e_{jm} = 0, \qquad 0 \leq m \leq 3, \tag{23}$$

and

$$\mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T + \mathbf{B}_j(e_{j0}, e_{j1}, e_{j2}, e_{j3})^T = (0, 0, 0, 0, 1)^T;$$

that is,

$$(\mathbf{B}_i + \mathbf{B}_j)(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T = (0, 0, 0, 0, 1)^T. \tag{24}$$

It has to be noted that conditions imposed by relations (22) and (24) can be considered jointly, provided that the error vector considered covers the error combinations just identified. Hereafter, $N_{\overline{\text{ded}_2}}$ is used to denote the number of such combinations.

• *Two errors which affect the check bit groups*
Here also, we distinguish the consequences of the errors affecting the various check bit groups.

Errors affecting groups 1 and 2
According to the form of the check matrix $\mathbf{H}''$, the conditions for miscorrection correspond to

$$e_{im} = e_m, \qquad 0 \leq m \leq 3,$$
$$\mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T = (e_4, e_5, e_6, e_7, 0)^T. \tag{25}$$

Errors affecting groups 1 and 3
The conditions to be verified correspond to

$$e_{im} = e_m, \qquad 0 \leq m \leq 3,$$
$$\mathbf{B}_i(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T = (0, 0, 0, 0, 1)^T. \tag{26}$$

Errors affecting groups 2 and 3
According to the inherent properties resulting from the construction of the code, it is important to note that any double-error combination $(e_4, e_5, e_6, e_7, 0)$, $e_8 = 1$, affecting groups 2 and 3, represents an error vector on five adjacent bits that is perfectly identifiable and is thus correctable. It follows that such a form of error cannot induce erroneous correction on another single group of a codeword; in effect, in this case, the syndrome bits s_0–s_3 are all equal to zero.

However, the benefit that can be expected from this added double-error-correction ability, resulting from the consideration of bits c_4–c_8 as a "single" group, has to be weighted with respect to the following points: 1) only 15 combinations on the total set of possible double-group error combinations are concerned, 2) due to the physical implementation of the system (groups of four bits), such a "single" error event is significantly less likely to occur than any actual single error event, and, moreover, 3) this "single" error is as likely to occur as any double error affecting other groups, which induces a tremendous increase in the number of miscorrections in the cases where these two errors have the same characteristics as a "single" error affecting check bits c_4–c_8. Such cases can be identified by the following relations:

$$e_{im} + e_{jm} = 0, \qquad 0 \leq m \leq 3,$$
$$(\mathbf{B}_i + \mathbf{B}_j)(e_{i0}, e_{i1}, e_{i2}, e_{i3})^T = (e_4, e_5, e_6, e_7, 1)^T, \tag{27}$$

which are definitely much less restrictive than the just-mentioned limitation to 15 correctable combinations.

This explains the reason for limiting the correction ability of the code to *single* groups. Therefore, no case for miscorrection has to be considered for double errors affecting check bit groups 2 and 3, as they result in the activation of the two associated pointers and thus can be detected. The number of non-detected double-error combinations which affect the check bit groups is denoted $N_{\overline{\text{ded}_3}}$.

Determination of the double-error-detection probability

A program has been developed which includes the three sets of conditions previously defined, in order to identify $N_{\overline{ded_1}}$, $N_{\overline{ded_2}}$, and $N_{\overline{ded_3}}$. It is important to note that all other double-error combinations will be detected; hence, the double-error-detection probability is derived by reference to the total number of double-error combinations N_{de}. The determination of N_{de} is made easy by considering that the double-error combinations are characterized by the following: 1) if c_8 is safe, the selection of two error groups among the remaining $r + 3$ groups that constitute a codeword; and 2) if c_8 is faulty, the choice of one group among $r + 3$. In both cases, it is considered that each group consists of four-adjacent bits, which corresponds to 15 distinct combinations. It follows that N_{de} is given by

$$N_{de} = \binom{r+3}{2} \times 15^2 + \binom{r+3}{1} \times 15. \qquad (28)$$

Considering that all double-error combinations have a constant density function, the probability of double-error detection can thus be expressed as

$$P_{ded} = 1 - \frac{\sum_i N_{\overline{ded_i}}}{N_{de}}. \qquad (29)$$

The numerical evaluation of this probability is strongly related to the actual implementation of the code that was discussed previously. The description of three different coding schemes, as well as the evaluation of their respective error-detection ability, based on results introduced in this section, is presented in the next section.

Selection of a coding scheme

As implicitly stated in previous sections, a straightforward implementation of the code would consist in the consideration of data bits only ($m'_d = 64$) in the generation of check bits. However, the network also conveys $m'_a = 20$ address bits for selecting the memory word from each memory module. An extension of the monitored lines to these address lines is made possible without an increase in the number of check bits since $m'_d + m'_a < 124$. This is desirable in order to enhance the range of covered faults. **Figure 3** illustrates the general organization of the coding scheme for the two cases just discussed.

These schemes are investigated in detail subsequently with respect to their capability for error detection. Furthermore, an estimate of the hardware complexity of the considered implementations is indicated as well, based on the coding/decoding equations presented earlier.

Consideration of data bits only

Such a scheme allows for the correction of 1) all memory faults that affect any single four-adjacent-bit groups in a word, and 2) network faults affecting any single commutation module that conveys data and check bits. However, no detection is provided for errors on address bits resulting from either faults affecting the address decoding in memory modules or the commutation modules conveying the address lines. Despite this strongly restrictive range of covered faults, this scheme is interesting, as it is characterized by the more efficient ability to cope with double-group errors, and the associated figure for double-error-detection probability is used as an optimal reference benchmark for subsequently considered schemes. This property is directly related to the fact that only half of the redundancy potentially available is used.

In this case, $r = 15$, and thus the value of the total number of double-error combinations is easily derived from relation (28) and is equal to $N_{de} = 34\,695$. The associated numbers of nondetected double-error combinations derived from conditions introduced previously are given in **Table 2.** Application of relation (29) leads to $P_{ded} = 49.55$ percent. It is important to note that this value is independent of the mode of implementation of the decoding scheme, since, in this case, redundancy provided by the code has to be fully used to correct all codeword bits.

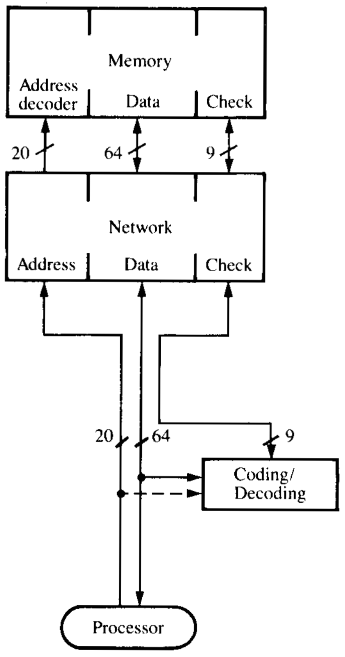

Figure 3 General coding/decoding scheme.

Table 2 Number of nondetected double errors—monitoring of data bits only.

$N_{\overline{ded_1}}$	$N_{\overline{ded_2}}$	$N_{\overline{ded_3}}$
14280	2835	390

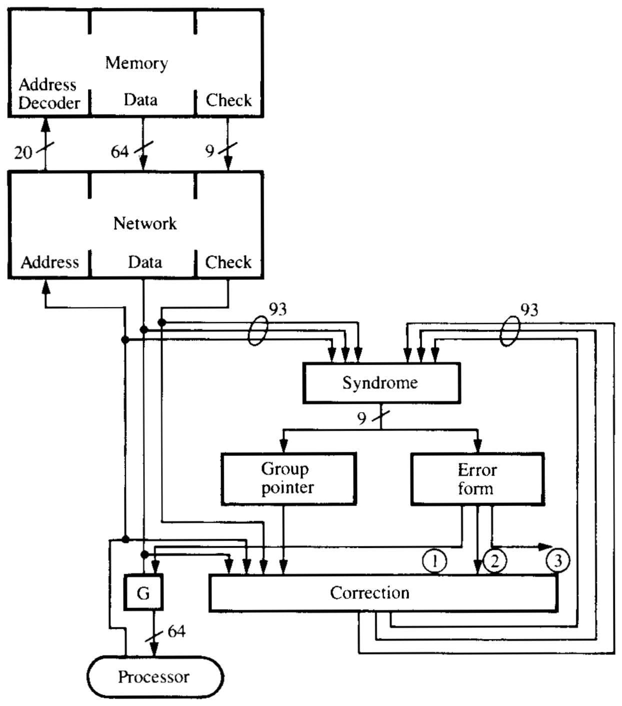

Figure 4 Basic implementation. [Note: Error forms are denoted by (1) no error, (2) single group error, and (3) noncorrectable errors.]

As far as the hardware complexity of this implementation is concerned, about 500 two-input logical functions were identified, among which almost three-fourths are exclusive-or functions. This value serves also as a reference benchmark for comparison with other implementations.

• *Inclusion of address bits*
The main advantage of the inclusion of the address bits in the generation of the code is the added ability to cope with addressing errors resulting from faults affecting both memory and commutation modules. Address bits are not stored, and only check bits corresponding to the data and the address of the word in which they reside are stored, as indicated in Fig. 3. Identification of addressing errors is then made possible when monitoring the syndrome vector. In practice, all addressing faults cannot be detected because of 1) the inherent limitation of code redundancy with respect to multiple faults, and 2) the different behaviors associated with read and write cycles. The first point has already been discussed; we stress here the difference that results whether the first manifestation of the fault occurs during a write cycle or a read cycle. This helps to identify the strengths and weaknesses of the scheme.

Let us consider first an addressing error occurring during a *write cycle* in the memory word of address X that results in a write in location Y; previous data and check bits contained in $Y(DC_Y)$ are overwritten by the new content (DC_X) that was addressed to X. In order to examine the consequences of this error on further read attempts, we distinguish two cases, considering that the initial fault is either permanent or transient.

If the fault is permanent, all further attempts to address X will attain Y, whose content is DC_X. No error indication will appear in the processing of the syndrome, which is acceptable since expected data have been reached. On the other hand, any attempt to address Y in order to retrieve DC_Y will deliver DC_X, which makes reference to address X in the memorized check bits. Thus, the concatenation of address bits sent Y and delivered information DC_X constitutes an invalid codeword (in the limit of the inherent redundancy of the code), and the error can be identified.

Transient addressing faults considered here correspond to faults that vanish before any further read cycle is performed. In this case, it is still possible to reach location X, but retrieved content DC'_X corresponds to the old value of the variable that has been stored in Y by error; such an error cannot be identified by the proposed scheme, as concatenation of X and DC'_X is a valid codeword. However, any attempt to access the data initially stored in $Y(DC_Y)$ induces the same consequences as in the case of permanent faults.

In the case of an addressing fault, either transient or permanent, occurring during a *read cycle* of location X that results in the access of the content of Y (DC_Y), it can be verified that concatenation of retrieved content DC_Y and address sent X constitutes in both cases a nonvalid codeword and is thus identifiable.

It is also important to note that another consequence of the inclusion of the address bits in the generation of the check bits is constituted by the reduction of the double-error-detection probability of the code; this follows from the related increase of 1) the number of groups to process to generate the check bits, which leads to $r = 20$, and 2) the number of double-error combinations to account for, which is equal to $N_{\text{ded}} = 52\,270$. However, as opposed to the case where only data bits are considered, it is important to note that, due to the specific consequence of address faults that were just identified, only the detection and diagnosis of faults that affect address bits are of practical interest. Two implementations of the decoding policy have been considered, and these are now described and evaluated.

Basic implementation
The basic principle of this implementation is illustrated in **Figure 4.** As previously discussed, during a memory read cycle, check bits related to the address sent and data read from memory are compared to the stored check bits in order to obtain the syndrome vector. Syndrome analysis allows for 1)

identification of the error form, and 2) generation of the group pointers according to equations presented previously. Study of the different error forms results in the distinction of three principal cases identified in Fig. 4.

1. No error indication appears and data are directly delivered without entering the correction circuit.
2. A single group error is diagnosed and all the bits (data + address + check) pass through the correction circuit to restore the erroneous group; corrected bits are then sent back to the syndrome generator and equality to zero of the output is tested before corrected data are delivered to the processor module. This feedback-type correction is known as the wrap-around correction [18]; it ensures that the corrected word constitutes a valid codeword and allows detection of faults that could occur in the correction state.
3. This case corresponds basically to noncorrectable errors (double error, for example); an interrupt signal has to be generated in order to process this case by investigation of error characteristics. Note also that this form of error diagnosis must include any error (yet single) in the address bit groups.

According to this implementation, for which correction of *all* bits is performed before data are delivered in the case of single-error indication, conditions for miscorrection resulting from double errors that were presented previously are still valid, provided that 1) five supplementary shortened matrices are included in the set of "$\mathbf{B}_i$" matrices, where $i = 0, 1, \cdots, r = 20$, according to the correspondence stated in Table 1, and 2) address bits are characterized by d_{jk}, with $16 \le j \le 20$, $0 \le k \le 3$. Moreover, as previously considered, this set may be extended with a null matrix denoted $\mathbf{B}_{r+1}$.

It follows that this implementation leads to the numbers of nondetected double-error combinations indicated in **Table 3.** Thus, the value of the probability of double-error detection is $P_{\text{ded}} = 34.27$ percent. This figure shows that the obtained benefit in addressing error identification, resulting from the inclusion of address bits in the generation of the check bits, results in a relative P_{ded} decrease of more than 30 percent in this implementation.

Considering the hardware overhead associated with this implementation, a basic estimation leads to 744 two-input functions, which corresponds to a relative increase of 48 percent with respect to the previous implementation.

Modified implementation

The major cause in the significant degradation of P_{ded} observed for the previous implementation lies in the constraint imposed by the wrap-around correction, which requires that correction be applied on all groups. A modification of the implementation which eliminates the wrap-around correction has been investigated in order to improve the probability of double-error detection. The modified implementation is described in **Figure 5.** Furthermore, two important points justify this modification: 1) it is not necessary that corrected check bits be delivered to the processor modules, and thus no correction circuit is needed for the check bits, and 2) activation of pointers corresponding to address groups has to initiate an error processing at the system level. The latter point is interesting since it involves a double error leading to erroneous identification of a single error on the address groups resulting in a detection decision (with possible erroneous diagnosis) but not in a miscorrection leading to system failure. It follows that conditions on nondetected double errors become more restrictive than the ones presented previously. These new conditions are now investigated.

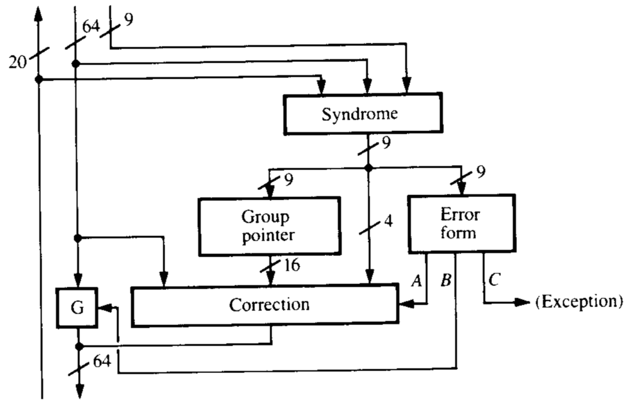

Figure 5 Modified implementation. (Note: Error forms are denoted by A, single error on data bits; B, no error or single error on check bits; C, single error on address bits or noncorrectable error.)

Table 3 Number of nondetected double errors—monitoring of data and address bits—basic implementation.

$N_{\overline{ded}_1}$	$N_{\overline{ded}_2}$	$N_{\overline{ded}_3}$
32208	4935	490

Modifications of conditions imposed by relations (16), (18), and (20) correspond to the limitation of variation of subscript k to $k \le 15$ and $k = 21$; furthermore, it is important to remember that these relations explicitly represent the case

$$i + j \rightarrow k; \qquad \forall\ i, j = 0, 1, \cdots, 21; \qquad i \ne j.$$

For the two other cases deduced by rotation, it follows that

$$j + k \rightarrow i, \quad i \le 15, \quad i = 21; \quad \forall\ j, k = 0, 1, \cdots, 21; \quad i \ne j,$$

and

$$k + i \rightarrow j, \quad j \le 15, \quad j = 21; \quad \forall\ k, i = 0, 1, \cdots, 21; \quad k \ne i.$$

Table 4 Number of nondetected double errors—monitoring of data and address bits—modified implementation.

$N_{\overline{ded_1}}$	$N_{\overline{ded_2}}$	$N_{\overline{ded_3}}$
24917	4149	462

Modifications of conditions (22) and (24) require consideration of only the following cases, where cg_i identifies check bit group i:

$$i + j \rightarrow \begin{Bmatrix} cg_2 \\ cg_3 \end{Bmatrix}; \quad \forall\ i, j = 0, 1, \cdots, 20; \quad i \neq j,$$

$$j + \begin{Bmatrix} cg_2 \\ cg_3 \end{Bmatrix} \rightarrow i, \quad i \leq 15; \quad \forall\ j, \quad i \neq j,$$

$$\begin{Bmatrix} cg_2 \\ cg_3 \end{Bmatrix} + i \rightarrow j, \quad j \leq 15; \quad \forall\ i, \quad j \neq i.$$

Finally, conditions (25) and (26) have to be restricted with respect to the variation of subscript i in the cases where the resulting error affects address bits, which leads to

$$i + cg_1 \rightarrow \begin{Bmatrix} cg_2 \\ cg_3 \end{Bmatrix}, \quad \forall\ i = 0, 1, \cdots, 20,$$

$$cg_1 + \begin{Bmatrix} cg_2 \\ cg_3 \end{Bmatrix} \rightarrow i, \quad i \leq 15,$$

$$\begin{Bmatrix} cg_2 \\ cg_3 \end{Bmatrix} + i \rightarrow cg_1, \quad \forall\ i = 1, 2, \cdots, 20.$$

The associated numbers of double-error combinations that will be miscorrected are given in **Table 4**; this leads to a probability of detection of double group errors equal to P_{ded} = 48.44 percent, which compares very well with the benchmark figure corresponding to the classical implementation of the code. This figure clearly shows the efficiency of the modified implementation.

The hardware overhead to consider in this case is characterized by an increase of 32 percent with respect to the estimation presented for the case where only data bits are monitored in the construction of the code.

These results, along with the address-error-detection abilities of this code, clearly show the efficiency of this modified implementation.

Conclusion

An efficient error-correcting scheme designed to be implemented on a supercomputer system has been presented. The problem of primary memory data error protection is crucial in such systems due to the unusually large capacity and the complexity and length of computations to be performed.

Based on structural considerations inherent in the system, the considered code is a shortened redundant (5,1)-adjacent code intended for single four-adjacent error correction. Such a code has been selected mainly for its ease of derivation and implementation. Different schemes for application of the code have been investigated and evaluated, taking into account both the range of faults covered and the double-error-detection property. From this study, a preferred scheme that includes both data bits and address bits in the construction of the code has been selected. In particular, it has been shown that such a scheme compares very well, on the basis of double-error-detection capability, with a classical coding scheme for which only data bits are considered in the generation of the check bits; the evaluation indicated a relative degradation of less than 2.5 percent. Moreover, another main advantage of this scheme is its ability to detect addressing errors. Quantitative evaluation of the efficiency of this added detection ability is difficult because, as it has been shown, it relies heavily on both the time characteristics of the errors and the sequencing of write and read cycles. Nevertheless, as indicated by the presented qualitative study, a large proportion of addressing faults can be detected; clearly these detection properties are limited by the inherent redundancy provided by the code.

As a closing remark, it is important to stress that such an error-correcting scheme, including both data and address bits in the generation of check bits, and characterized by an implementation based on the correction of data and check bits only, constitutes an efficient mechanism that allows for 1) correction of memory and network single four-adjacent errors, 2) detection of a large proportion of single four-adjacent addressing errors, and 3) detection of about half of double four-adjacent errors affecting either data, address, or check bits.

Note
J. Arlat's work was supported by SINTRA under Contract No. 84255/55295/AA/BS. W. C. Carter contributed to this study during his sabbatical leave at LAAS, CNRS, from August 1981 to February 1982.

References and note

1. R. Bernhard, "Computing at the Speed Limit," *IEEE Spectrum* **19,** No. 7, 26–31 (1982).
2. T. Nuyen Linh, "Survey of Semiconductor (Si, GaAs) and Josephson Technologies—Today and Tomorrow," *Proceedings, Computer Science Convention A*, SICOB, Paris, September 1982, pp. 1–7 (in French).
3. N. R. Lincoln, "Technology and Design Tradeoffs in the Creation of a Modern Supercomputer," *IEEE Trans. Computers* **C-31,** 349–362 (1982).
4. D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers* **C-24,** 1145–1155 (1975).
5. Hamming introduced an excellent class of SEC-DED codes [6]; however, almost all practical implementations of SEC-DED codes now use modified parity-check matrices of the type devised by Hsiao [7] because of their higher probability of detection of multiple errors and because of the hardware savings realizable in their implementation.

6. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.* **29,** 147–154 (1950).
7. M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM J. Res. Develop.* **14,** 395–401 (1970).
8. T. Y. Feng, "A Survey of Interconnection Networks," *Computer* **14,** No. 12, 72–78 (1981).
9. J. P. Shen and J. P. Hayes, "Fault Tolerance of a Class of Connecting Networks," *Proceedings, International Symposium on Computer Architecture,* La Baule, France, May 1980, pp. 61–71.
10. F. Larbey, "Crossbar and Transcoder Circuits: Technical Specifications," *Report No. 26.069,* Oct. 1981, available from SINTRA, 92602 Asnieres, France (in French).
11. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Indust. Appl. Math.* **8,** 300-304 (1960).
12. W. W. Peterson, *Error Correcting Codes,* MIT Press, Cambridge, MA, 1961.
13. J. Cocke, "Lossless Symbol Coding with Non-Primes, " *IEEE Trans. Info. Theory* **IT-5,** 33–36 (1959).
14. D. C. Bossen, "b-Adjacent Error Correction," *IBM J. Res. Develop.* **14,** 402–408 (1970).
15. W. C. Carter, E. P. Hsieh, and A. B. Wadia, "Multiple b-Adjacent Group Error Correction and Detection Codes and Self-Checking Translators Therefor," U. S. Patent 3,766,521, 1973.
16. W. C. Carter and A. B. Wadia, "Design and Analysis of Codes and Their Self-Checking Circuit Implementations for Correction and Detection of Multiple b-Adjacent Errors," *Proceedings, 10th International Symposium on Fault-Tolerant Computing,* Kyoto, Japan, October 1980, pp. 35–40.
17. W. G. Bouricius, W. C. Carter, E. P. Hsieh, D. C. Jessep, and A. B. Wadia, "Modeling of a Bubble Memory Organization with Self-Checking Translators to Achieve High Reliability," *IEEE Trans. Computers* **C-22,** 269–275 (1973).
18. W. C. Carter and C. E. McCarthy, "Design and Analysis of an Experimental Fault-Tolerant Memory System," *IEEE Trans. Computers* **C-25,** 557–568 (1976).

Received June 3, 1983; revised October 12, 1983

Jean Arlat *Laboratory for Automatics and Systems Analysis (LAAS), National Center for Scientific Research, 31400 Toulouse, France.* Dr. Arlat first joined the "Design and Validation of Dependable Computer Systems" team at LAAS in 1976. He prepared his Doctor-Engineer diploma (Ph.D.) in automatics, investigating the concept of diversification for the tolerance of hardware design faults, which he received in 1979 from the Toulouse National Polytechnic Institute. During 1979–80, he spent a postdoctoral year in the Computer Science Department of the University of California at Los Angeles, working on the evaluation of software fault tolerance strategies. Since the end of 1980, he has been back at LAAS working on dependability evaluation of fault-tolerant computer systems and has been involved with several contracts with French industries. Currently, he is engaged in the preparation of a "Docteur ès-Sciences" thesis focusing on the development of methods for the evaluation of the dependability of fault-tolerant systems including both modeling and experimental approaches. Dr. Arlat is a member of the Institute of Electrical and Electronics Engineers and the Fault-Tolerant Computing Technical Committee (FTC-TC), acting as a French correspondent for the FTC-TC Newsletter.

William Caswell Carter *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Carter joined IBM in the Research Division in 1959 at Poughkeepsie, New York. From 1959 to 1961, he was a member of the technical staff at the Research laboratory with responsibilities for cryogenic machine organization for Project Lightning and for planning a specialized programming system. From 1961 to 1966, he was in the Systems Development Division in charge of systems automation for IBM System/360. His responsibilities included System/360 diagnostic programs, design for System/360 recovery, diagnostics and automatic test generation, design automation for ROS-controlled computers, and computer systems evaluation and planning. In 1966, he rejoined the Research Division and continued work in the field of fault-tolerant computing. From 1966 to 1972, he worked on architecture, modeling, and design procedures. Since 1972, he has been studying microprogram validation techniques as well. From 1947 to 1952, he was at Aberdeen Proving Ground using the ENIAC to obtain new solutions for compressible fluid flow problems and was adjunct instructor at the University of Maryland and at Johns Hopkins University. From 1952 to 1955, he was head of the Systems Section, Raytheon Computer Department, with responsibilities for computer system evaluation, reliability and serviceability, methods of logic design, programming systems, and preparation of government proposals; and in addition, he was adjunct professor at Boston University until 1956. From 1955 to 1959, he was manager of the Systems Analysis Department, EDP Division, of Minneapolis Honeywell, with design automation added to his responsibilities. Dr. Carter received the B.A. (*magna cum laude*) from Colby College, Waterville, Maine, in 1938 and the Ph.D. in mathematics from Harvard University in 1947. Dr. Carter is a Fellow of the Institute of Electrical and Electronics Engineers and a member of the American Association of Rhodes Scholars, American Men and Women of Science, Who's Who in America, Association for Computing Machinery, Phi Beta Kappa, Society for Industrial and Applied Mathematics, and Sigma Xi. He was an ACM National Lecturer on combinational mathematics from 1962 to 1964 and 1967 to 1968 and an IEEE Computer Society Distinguished Visitor from 1973 to 1974. He was a member of the IEEE (AIEE) Computer Society Technical Committee on systems from 1954 to 1964 and helped form the Technical Committee on fault-tolerant computing in 1970. He was Vice Chairman of this committee from 1970 to 1972 and Chairman from 1973 to 1976. He has frequently served on FTCS program committees (Chairman 1971), is general Chairman of FTCS-11, and edited an IEEE TC special issue on fault-tolerant computing in 1973. He was named to the Computer Society Honor Roll in 1974 and was a member of the Computer Society Technical Interest Council from 1976 to 1978. Dr. Carter is a U.S. representative to IMEKO TC.10, Technical Diagnostics.