

ERROR CORRECTION BY  
ALTERNATE-DATA4 RETRY

John J. Shedletsky

Technical Report No. 113

May 1976

This work was initiated while the author held an IBM Graduate Fellowship and was supported in part by the National Science Foundation under Grant G J-40286.

**DIGITAL SYSTEMS LABORATORY**  
**STANFORD ELECTRONICS LABORATORIES**  
**STANFORD UNIVERSITY • STANFORD, CALIFORNIA**



ERROR CORRECTION BY ALTERNATE-DATA RETRY

John J. Shedletsky\*

Technical Report No. 113

May 1976

Center for Reliable Computing  
Digital Systems Laboratory  
Departments of Electrical Engineering and Computer Science  
Stanford University  
Stanford, California

\*This work was initiated while the author held an IBM Graduate Fellowship and was supported in part by the National Science Foundation under Grant GJ-40286.



ERROR CORRECTION BY ALTERNATE-DATA RETRY

John J. Shedletsky

Technical Report No. 113

May 1976

Center for Reliable Computing  
Digital Systems Laboratory  
Departments of Electrical Engineering and Computer Science  
Stanford University  
Stanford, California

ABSTRACT

A new technique for low-cost error correction in computers is the alternate-data retry (ADR). An ADR is initiated by the detection of an error in the initial execution of an operation. The ADR is a re-execution of the operation, but with an alternate representation of the initial data. The choice of the alternate representation and the design of the processing circuits combine to insure that even an error due to a permanent fault is not repeated during retry. Error-correction is provided at a hardware cost comparable to that of a conventional retry capability.

Sufficient conditions are given for the design of circuits with an ADR capability. The application of an ADR capability to memory and to the data paths of a processor is illustrated.

INDEX TERMS: Alternate-date retry, fault neutralization, morphic circuits, recoverable, self-checking, availability.



## INTRODUCTION

The increasing use of on-line computer systems has imposed strict requirements of computer reliability and availability. A computer hardware failure in a data management, interactive computing or control environment can lead to unacceptable costs in aggravation, idleness or even human life.

Efficient recovery procedures are essential to the reliability required of such systems. Recovery is defined [Carter, 1970] as the continuation of system operation with data integrity after a fault occurs. A successful recovery from a hardware failure must correct all errors due to the failure to insure continued system operation with data integrity. This error-correction must be accomplished quickly and automatically if high system availability is to be maintained.

Repair and program rollback is a typical recovery method for correcting undetected errors caused prior to the detection of a fault. The entire state of a process is copied into secondary storage at pre-selected points called checkpoints [Chandy, et al, 1975]. After repair of the fault, data integrity is restored by a program rollback to the assumed error-free checkpoint copy of the data. The total recovery time includes the time to repair the fault, to load the checkpoint data, and to re-execute the portion of the program included in the rollback.

Certain properties implemented by logic design can be used to enhance availability by reducing the total recovery time. The ~~self-checking~~ self-checking property guarantees that a fault in a circuit is eventually detected and that no undetected errors due to the fault can be propagated. The self-checking property obviates the need for rollback by eliminating the possibility of any undetected errors prior to the detection of the fault [Shedletsy, 1976]. Thus, the recovery time for a self-checking system is only the time to repair the fault and to restart without rollback.

Automatic repair of a permanent hardware fault has typically been achieved by automatic circumvention of the faulty hardware.

E-

Standby sparing [Avizienis, 1971] or simple amputation [Fox, 1975] are examples of circumvention. Both cases imply substantial hardware redundancy. Error-correcting codes and specifically, voting on the outputs of three or more identical modules [Dickinson, et al, 1964] are examples of ways to provide automatic error-correction without repair. In any case, the hardware cost of automatic error-correction has been prohibitive, except in special or critical applications. Voting, for example, increases the hardware cost by more than threefold.

Instruction or task retry is another recovery method for automatic error-correction [Carter, et al, 1974]. This technique offers the advantage of quick repair time since it is automatic, but without the disadvantage of excessive hardware cost. Unfortunately, retry has only been applicable to the correction of errors caused



by transient or intermittent faults.

This paper proposes the alternate-data retry (ADR) as a fundamentally new technique for automatic error-correction. An ADR provides the error-correction for permanent faults found in existing hardware-redundant schemes, but at low hardware cost over that of a retry scheme. The example in Figure 1 illustrates the basic principle invoked by an ADR.

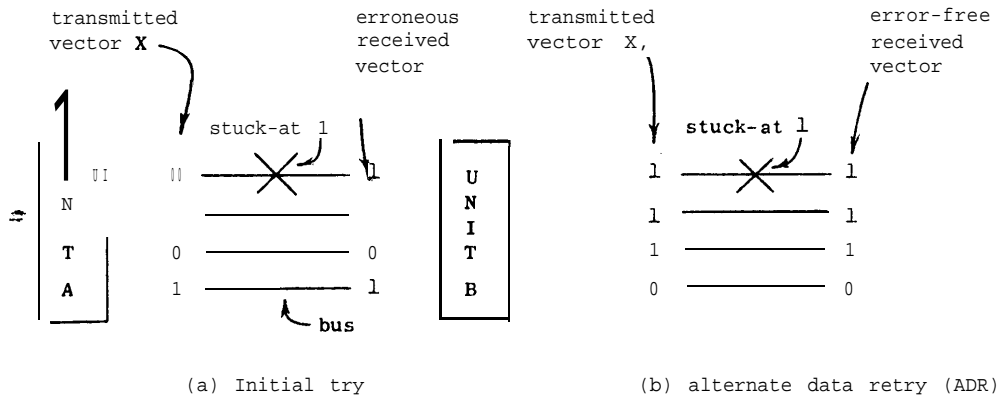


Fig. 1. - Fault neutralization during an ADR.

In Figure 1, unit A sends the odd-parity vector  $\langle 0001 \rangle$  over the bus to unit B. Due to the stuck-at fault, unit B receives the erroneous, even-parity vector  $\langle 1001 \rangle$  and signals an error. For the retry, unit A sends the alternate data-representation  $\langle 1110 \rangle$ , which is received correctly despite the fault. In this example, both the vector  $\langle 0001 \rangle$  and its complement  $\langle 1110 \rangle$  represent the same message and convey the same information to unit B.

A logical stuck-at fault is neutralized by the signal value of a line if the signal value is equal to the stuck-at value.

Every message in an ADR system is represented by more than one binary vector. After causing an error with the initial representation <0001>, the fault in Figure 1 is neutralized by the alternate representation <1110> during the ADR. While previous methods of error-correction seek to mask an error resulting from the application of data, an ADR applies an alternate representation to neutralize the fault so that no error is ever generated. Like any retry, an ADR is also effective against transient and intermittent faults. Since an ADR changes the pattern of data, it should also be effective against pattern-dependent faults.

This paper approaches the ADR as a candidate for possible use in low cost, highly available systems. Since only detected errors are corrected by an ADR, the self-checking property is assumed. In this way, all errors due to a fault are detected and corrected by an ADR before data contamination can occur.

The following section formally describes the system of message representation and circuit properties necessary for an ADR capability. Other sections provide guidelines for the design of circuits possessing these properties. A simple example of a self-checking processor with an ADR capability is given in the last section. This processor can achieve automatic error-correction by an ADR for any single fault (and many multiple faults) that may occur in the data paths or the memory.

## CIRCUIT PROPERTIES FOR AN ADR CAPABILITY

A message is conveyed by the pattern of logic values in a binary vector. Determinism is preserved in a digital system since each binary vector represents one and only one message. The binary vector <0101> for example, represents the message "5". Each message is represented by only one vector in a simplex system of message representation, but by more than one vector in a morphic system of message representation.

Let R be a relation on the set of binary vectors in a morphic system of message representation. Define R such that one vector is related to another if both vectors represent the same message. The relation R is an equivalence relation that partitions the vectors into morphic equivalence classes (MEC's). There is a one-to-one correspondence between MEC's and messages. Since messages are represented by only one vector in a simplex system, there is also a one-to-one correspondence between MEC's and simplex vectors.

For any circuit h mapping input vectors x to output vectors h(x) in a simplex system, there is a corresponding circuit H mapping input vectors X to output vectors H(X) in a morphic system.

Definition. The circuit H is morphic\* to the circuit h when vector X being a member of the MEC corresponding to simplex vector x implies that H(X) is a member of the MEC corresponding to h(x).

---

\*The circuits defined in [Carter, 1972] are a special case of this general definition.

We refer to  $h$  as the simplex circuit and  $H$  as the morphic circuit. There can be more than one morphic circuit corresponding to a given simplex circuit. A morphic circuit preserves the correspondence between a simplex and morphic system of message representation by mapping every member of each input MEC into members of the appropriate output MEC (Figure 2).

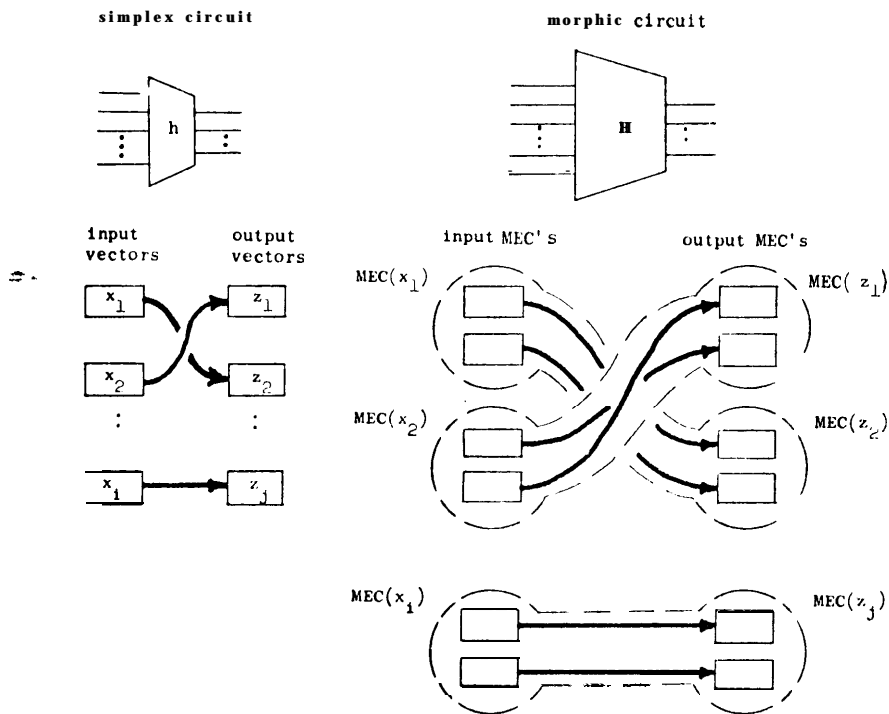


Fig. 2. - The mapping function realized by a simplex circuit  $h$  and a corresponding morphic circuit  $H$ .

The output  $H_f(X)$  of a faulty morphic circuit  $H$  is correct if  $H_f(X) = H(X)$ . The output  $H_f(X)$  is morphically correct if  $H_f(X)$  is equivalent to  $H(X)$  by the equivalence relation  $R$ .

Definition. A morphic circuit  $H$  is recoverable for a set of faults if for every fault in the set, the output  $H_f(X)$  is correct for at least one member  $X$  of every input MEC.

A broader definition of recoverability requires only that the output be morphically correct for at least one member of every input MEC. The following theorem 1 holds for the broader definition, but no practical advantage is gained. This paper assumes the **narrow**-sense definition above.

Example 1. Let every MEC in Figure 1 have just two members, an odd-parity code word and its (odd-parity) bit-complement. It follows that four messages can be represented by such a morphic/coded system. The bus in Figure 1 is morphic and recoverable for any single stuck-at fault.

The property of recoverability insures that a faulty morphic circuit produces a correct output for at least one member of each MEC. The retry strategy for error-correction is to apply alternate members of the same input **MEC** until a correct output is produced. An independent means of error-detection is assumed.

Algorithm 1: Error correction by ADR.

1. Apply an as yet untried member of the given input MEC.
2. If no error is observed in the output, then exit. Otherwise go to 3.
3. If all members of the input MEC have been tried, then signal a failure to correct and exit. Otherwise go to 1 (retry).

Theorem 1. If a circuit is recoverable for a fault set, and if an independent means of error-detection is provided, then algorithm 1 corrects any output error caused by a fault in the set.

Proof. The recoverability property guarantees that at least one member of the input MEC produces a correct output. A failure to correct in step 3 can only occur if the fault is not in the prescribed fault set.

Error-detecting codes provide the independent means of error detection required by algorithm 1. A codeword represents one and only one message while a noncode word represents no message. A noncode word indicates an error condition. A message is represented by more than one codeword in a morphic/coded system of message representation. A code is consistent with a morphic system if every member of each MEC is a code word and every code word is a member of some MEC (Figure 3).

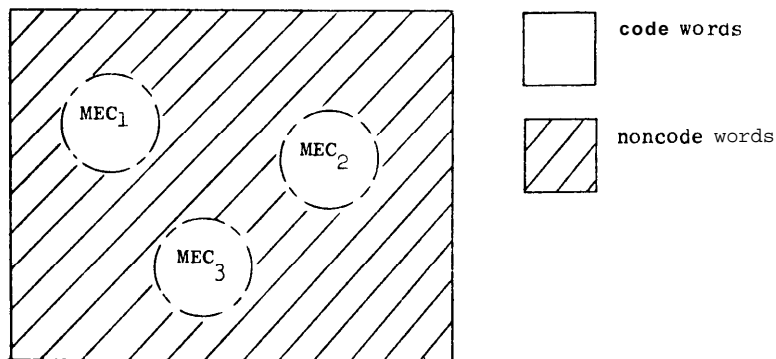


Fig. 3. - A consistent, morphic/coded system of message representation.

The following circuit properties for dynamic error-detection were introduced in [Carter and Schneider, 1968] and refined in [Anderson, 1971].

Definition [Anderson, 1971]. A circuit is secure for a fault set if for every fault in the set, the faulty circuit never produces an incorrect code output for code inputs.

Definition [Anderson, 1971]. A circuit is self-testing for a fault set if for every fault in the set, the faulty circuit produces a noncode output for at least one normally applied code input.

Definition [Anderson, 1971]. A circuit is self-checking if it is both secure and self-testing.

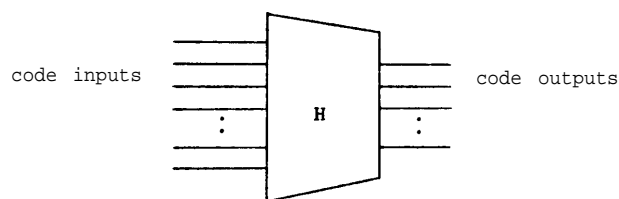


Fig. 4. - Amorphous circuit for use with a morphic/coded system of message representation.

If circuit H in Figure 4 is secure, then any output error can only force the output to a **noncode** word, which triggers an **error-detection**. A circuit must also be self-testing in an environment where maintenance is performed only after a fault is detected. If a circuit were not self-testing for a fault, then the fault would never be detected in normal operation. A second fault could occur

before the first was repaired, and the circuit might not be secure for the two faults combined.

Theorem 2 summarizes a list of circuit properties sufficient for immediate error-detection and then correction by ADR.

Theorem 2. Algorithm 1 can correct any error due to a fault in the circuit H in Figure 4 if the fault belongs to the set **F** and H is:

- (1) operated in a consistent morphic/coded system of message representation
- (2) morphic
- (3) self-checking for **F**, and
- (4) recoverable for **F**.

The design of self-checking circuits is discussed in detail in [Carter and Schneider, 1968] and [Anderson, 1971].



## DESIGN OF RECOVERABLE C-MORPHIC CIRCUITS

3 The choice of a morphic system of message representation depends on the set of faults that are most likely to occur. A circuit can only be recoverable for a fault if at least one member of each MEC can neutralize or mask the fault. Experience has indicated that most faults can be modeled as logical stuck-at faults. A single (stuck-at) fault forces the logic signal on a line to a constant 0 or to a constant 1 value.

Since only one member of a MEC is stored in a register or memory location, the other members must be easily derived by a hardware **translator**. Also, each MEC should have as few members as possible to minimize the maximum number of retrys.

The C-morphic\* system of message representation is distinguished by its equivalence relation R. In the C-morphic system, a vector X is related to a vector Y if  $X = Y$  or  $X = \overline{Y}$ . The **overbar** indicates bit-by-bit complementation. Every MEC has only two members, a vector and its bit complement.

The C-morphic system satisfies the considerations mentioned above. Every MEC has an optimal size of two. A translator is simply a multiplexer that chooses either the uncomplemented or the complemented output of each latch in a register. Also, design guidelines for recoverability developed in this section guarantee that at least one member of each MEC can neutralize or mask any single single fault.

---

\*The "C" implies complementation.

A code is consistent with a C-morphic system if the bit complement of every code word is also a code word. Several codes have this property, including the two-rail code<sup>\*</sup>, replication codes,  $n$  out of  $2n$  codes, and residue codes with checkbase  $A = 2^a - 1$  (low-cost residue codes) [Avizienis, 1969]. The bit complement of a binary linear block code word is also a code word if every row in the parity-check matrix [Peterson and Weldon, 1972] has an even number of ones. It follows that canonical hamming codes [Peterson and Weldon, 1972] and simple parity over an even number of bits are also consistent with the C-morphic system (see Example 1).

A Morphic circuit for use in a C-morphic system is a C-morphic circuit.

Example 2. A C-morphic identity circuit maps members of an input MEC into members of the same output MEC. Both circuits in Figure 5 are C-morphic identity circuits. There are many others.

A primary input line is an input line that does not fan-out, or an input line up to the point of fan-out. Any single fault in a primary input line is neutralized by one of a pair of complementary inputs. This proves lemma 3.

Lemma 3. A C-morphic circuit is recoverable for single faults affecting the primary input lines.

---

\* A two-rail code is complemented duplication.

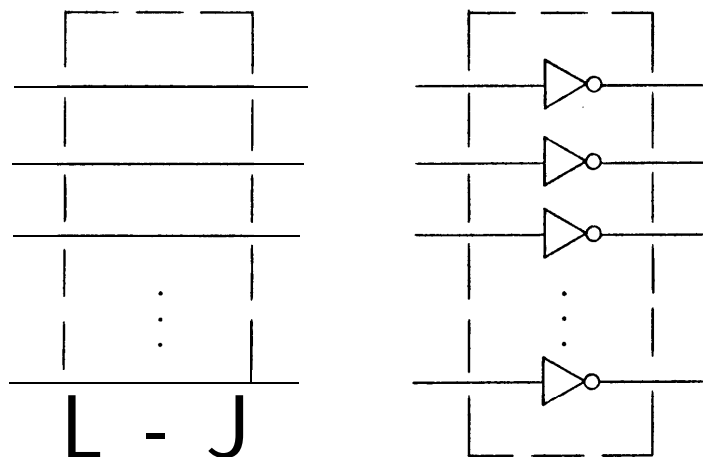


Fig. 5. - Two C-morphic identity circuits.

Any single fault in an output line of a C-morphic circuit is neutralized if and only if the complementary input vectors are mapped into complementary output vectors. The dual of a single or multi-output circuit  $g$  is denoted  $g^d$  and is defined  $g^d(x) \equiv \overline{g(x)}$ , where  $x$  is a normally applied input vector. A circuit  $g$  is self-dual if  $g(x) = \overline{g(x)}$ . The output vectors of a self-dual circuit are complementary for complementary input vectors. Both circuits in Figure 5 are self-dual. Lemma 4 follows.

Lemma 4. A C-morphic circuit is recoverable for single faults affecting the output lines if and only if the circuit is self-dual for the set of normally applied inputs.

Any self-dual circuit may be viewed as a C-morphic circuit, so the recoverability property applies. A self-dual circuit is recoverable for a fault set if for every fault in the set, the faulty circuit produces at least one correct output for every pair or normally applied, complementary input vectors.

Theorem 5. A network of self-dual modules, each recoverable for a fault set  $F_1$ , is recoverable for the fault set  $F$  where  $F$  is the union of all sets  $F_1$ . The set  $F$  includes all single faults affecting connections between modules.

Proof. A network of self-dual modules is self dual. The inputs to any module are complementary for complementary network inputs. The network is recoverable for any fault in a module for which the module is recoverable. Single faults in module connections are included in  $F$  by lemmas 3 and 4.

Theorem 5 permits the construction of a recoverable network by the interconnection of smaller, recoverable modules. The following development produces guidelines for the design of recoverable, self-dual circuit modules.

We define a four-valued logic for the analysis of faulty circuits [Roth, et al, 1967]. A line in a faulty circuit can assume one of the four logic values: 0, 1, D or  $\bar{D}$ . Table 1 describes the meaning assigned to each value.

Table 1 - A four-valued logic and assigned meanings.

	binary logic value in faulty circuit	binary logic value if circuit were fault-free
0	0	0
1	1	1
D	0	1
$\bar{D}$	1	0

Figure 6 shows the four-valued truth tables for an inverter, a line stuck-at 0, and a line stuck at 1. Figure 7 shows the four-valued truth table for a two-input AND gate and OR gate.

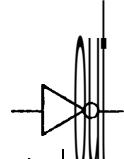
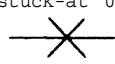
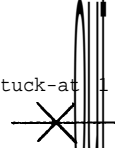


	stuck-at 0 	stuck-at 1 																														
<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th>in</th><th>out</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> <tr><td>D</td><td><math>\bar{D}</math></td></tr> <tr><td><math>\bar{D}</math></td><td>D</td></tr> </tbody> </table>	in	out	0	1	1	0	D	$\bar{D}$	$\bar{D}$	D	<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th>in</th><th>out</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>D</td></tr> <tr><td>D</td><td>D</td></tr> <tr><td><math>\bar{D}</math></td><td>0</td></tr> </tbody> </table>	in	out	0	0	1	D	D	D	$\bar{D}$	0	<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th>in</th><th>out</th></tr> </thead> <tbody> <tr><td>0</td><td><math>\bar{D}</math></td></tr> <tr><td>1</td><td>1</td></tr> <tr><td>D</td><td>1</td></tr> <tr><td><math>\bar{D}</math></td><td><math>\bar{D}</math></td></tr> </tbody> </table>	in	out	0	$\bar{D}$	1	1	D	1	$\bar{D}$	$\bar{D}$
in	out																															
0	1																															
1	0																															
D	$\bar{D}$																															
$\bar{D}$	D																															
in	out																															
0	0																															
1	D																															
D	D																															
$\bar{D}$	0																															
in	out																															
0	$\bar{D}$																															
1	1																															
D	1																															
$\bar{D}$	$\bar{D}$																															

Fig. 6. - Four-valued truth tables for an Inverter, a stuck-at 0 fault, and a stuck-at 1 fault.

	
AND	OR

<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th></th><th>AND</th><th>OR</th></tr> </thead> <tbody> <tr><td>0 0</td><td>0</td><td>0</td></tr> <tr><td>0 1</td><td>0</td><td>1</td></tr> <tr><td>0 D</td><td>0</td><td>D</td></tr> <tr><td>0 <math>\bar{D}</math></td><td>0</td><td><math>\bar{D}</math></td></tr> </tbody> </table>		AND	OR	0 0	0	0	0 1	0	1	0 D	0	D	0 $\bar{D}$	0	$\bar{D}$	<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th></th><th>AND</th><th>OR</th></tr> </thead> <tbody> <tr><td>1 0</td><td>0</td><td>1</td></tr> <tr><td>1 1</td><td>1</td><td>1</td></tr> <tr><td>1 D</td><td>D</td><td>1</td></tr> <tr><td>1 <math>\bar{D}</math></td><td><math>\bar{D}</math></td><td>1</td></tr> </tbody> </table>		AND	OR	1 0	0	1	1 1	1	1	1 D	D	1	1 $\bar{D}$	$\bar{D}$	1	<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th></th><th>AND</th><th>OR</th></tr> </thead> <tbody> <tr><td>D 0</td><td>0</td><td>D</td></tr> <tr><td>D 1</td><td>D</td><td>1</td></tr> <tr><td>D D</td><td>D</td><td>D</td></tr> <tr><td>D <math>\bar{D}</math></td><td>0</td><td><math>\bar{D}</math></td></tr> </tbody> </table>		AND	OR	D 0	0	D	D 1	D	1	D D	D	D	D $\bar{D}$	0	$\bar{D}$	<table border="1" style="border-collapse: collapse; width: 60px;"> <thead> <tr><th></th><th>AND</th><th>OR</th></tr> </thead> <tbody> <tr><td><math>\bar{D}</math> 0</td><td>0</td><td><math>\bar{D}</math></td></tr> <tr><td><math>\bar{D}</math> 1</td><td><math>\bar{D}</math></td><td>1</td></tr> <tr><td><math>\bar{D}</math> D</td><td>0</td><td>1</td></tr> <tr><td><math>\bar{D}</math> <math>\bar{D}</math></td><td><math>\bar{D}</math></td><td><math>\bar{D}</math></td></tr> </tbody> </table>		AND	OR	$\bar{D}$ 0	0	$\bar{D}$	$\bar{D}$ 1	$\bar{D}$	1	$\bar{D}$ D	0	1	$\bar{D}$ $\bar{D}$	$\bar{D}$	$\bar{D}$
	AND	OR																																																													
0 0	0	0																																																													
0 1	0	1																																																													
0 D	0	D																																																													
0 $\bar{D}$	0	$\bar{D}$																																																													
	AND	OR																																																													
1 0	0	1																																																													
1 1	1	1																																																													
1 D	D	1																																																													
1 $\bar{D}$	$\bar{D}$	1																																																													
	AND	OR																																																													
D 0	0	D																																																													
D 1	D	1																																																													
D D	D	D																																																													
D $\bar{D}$	0	$\bar{D}$																																																													
	AND	OR																																																													
$\bar{D}$ 0	0	$\bar{D}$																																																													
$\bar{D}$ 1	$\bar{D}$	1																																																													
$\bar{D}$ D	0	1																																																													
$\bar{D}$ $\bar{D}$	$\bar{D}$	$\bar{D}$																																																													

Fig. 7.- Four-valued truth tables for a 2-input AND gate and a 2-input OR gate.

Definition. The errors appearing on a line are D-monotonic ( $\bar{D}$ -monotonic) if, over the set of all possible inputs, the line never assumes the value  $\bar{D}$  (D). The errors are polytonic if the line can assume both values D and  $\bar{D}$ .

A fault is testable if it changes the function realized by a circuit.

Definition. A testable fault in a single output circuit is D-monotonic,  $\bar{D}$ -monotonic or polytonic if the errors on the output line are D-monotonic,  $\bar{D}$ -monotonic or polytonic respectively.

Only successor lines to a fault can assume the error values D or  $\bar{D}$ . Figure 6 indicates that a line stuck-at 0(1) can only assume an error value of D ( $\bar{D}$ ). A faulty line generates only monotonic errors. Polytonic errors in a circuit of AND, OR and inverter gates can only be generated at the output of an AND or OR gate. These m-input gates can only generate polytonic errors if at least one gate input line has polytonic errors, or at least two gate-input lines have monotonic errors of opposing values D and  $\bar{D}$ . In a circuit with a single fault, it follows that polytonic errors can only be generated at the output of an AND or OR gate if there is more than one path from the fault to the gate. Furthermore, the monotonic errors on at least two gate-inputs at the point of reconvergence must have opposing values. Only inverters can change D-monotonic errors to  $\bar{D}$ -monotonic errors and vice versa. Theorem 6 follows.

Theorem 6. A necessary condition for a single fault to be polytonic in a single output circuit of AND, OR, and inverter gates is at least two paths from the fault to the output, one containing an odd number of inverters and the other an even number of inverters.

Theorem 7. A single-output, self-dual circuit is recoverable for a monotonic fault.

Proof. Complementary inputs correctly produce complementary values on the output line. A monotonic fault can cause at most one of each pair of complementary output values to be incorrect.

Monotonicity of output errors is not a necessary condition for recoverability.

In a tree circuit, no gate outputs fan-out, although primary input lines may fan-out.

Corollary 8. A single-output, self-dual circuit is recoverable for single faults if the circuit is either:

- (1) a tree circuit of AND, OR, or inverter gates, or
- (2) a general circuit of AND or OR gates, or
- (3) a general circuit of AND, OR, or inverter gates such that the number of inverters in every path from a fan-out point to the output are either all odd or all even, for every fan-out point.

Proof. In every case but (1), single faults are monotonic by theorem 6. In case (1), all single faults are necessarily monotonic by theorem 6 except those in primary input lines that fan-out. The circuit is recoverable for single faults in input lines by lemma 3.

Theorem 5 and corollary 8 together form guidelines for the

design and verification of recoverable, C-morphic circuits. The conditions given in corollary 8 are sufficient to insure recoverability, but not necessary.

The following section discusses two specific codes consistent with a C-morphic system of message representation. The self-checking property is particularly easy to satisfy for circuits using the first code. The second code is particularly hardware efficient.



## SPECIAL CASES: TWO CONSISTENT CODES

The self-checking condition in theorem 2 is generally the most difficult to satisfy. A duplication code is a special case for which the self-checking property is much easier to attain.

A C-duplication code is formed in two steps. First form a C-morphic system of message representation by appending an invert bit  $i$  to an  $r$ -bit simplex vector. The uncomplemented (complemented) member of a MEC is that member with invert bit equal to  $0(1)$ . The second step is to duplicate each of the  $r + 1$  bits to form a  $2(r + 1)$  bit duplication code word.

A C-morphic circuit  $G$  for the C-duplication code is formed with two identical, C-morphic modules  $G1$  and  $G2$  (Figure 8).

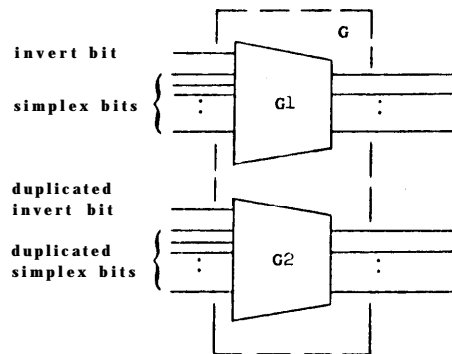


Fig. 8. - A morphic circuit for use with a C-duplication code.

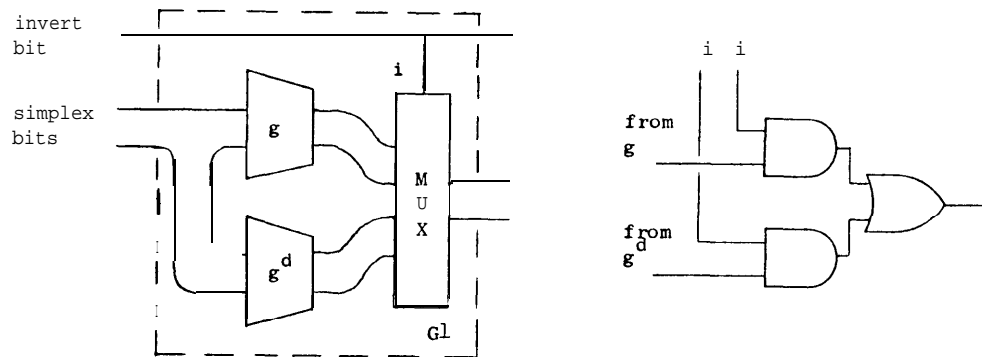
The C-morphic circuit  $G$  is easily verified to the self-checking; it is secure for any fault limited to  $G1$  or  $G2$ , and self-testing for any testable fault in  $G1$  or  $G2$ .\*

\* It is assumed that all  $2^{r+1}$  input vectors are normally applied to  $G1$  and  $G2$ .

Let  $g$  be the simplex circuit to which the morphic module  $G1$  corresponds. Module  $G1$  can be constructed as shown in Figure 9a.

Detail of the multiplexer in Figure 9a is shown in Figure 9b.

Module  $G2$  is identical to  $G1$ .



(a) Overall diagram.

(b) Multiplexer detail.

Fig. 9. - A C-morphic circuit corresponding to the simplex circuit  $g$ .

Module  $G1$  is self-dual even if the simplex circuit  $g$  is not. A careful analysis of the multiplexer verifies that  $G1$  is recoverable for single faults in the multiplexer. Furthermore, it is easy to verify that  $G1$  is recoverable for faults in the simplex circuit  $g$  and the dual  $g^d$ , and single faults in the primary input lines and output lines. We conclude that the module  $G1$  and hence the C-morphic circuit  $G$  is recoverable for single faults.

If the simplex circuit  $g$  is self-dual, then module  $G1$  can be constructed as shown in Figure 10.

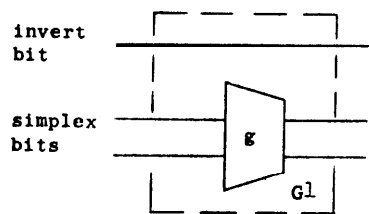


Fig. 10. - AC-morphic circuit corresponding to the self-dual, simplex circuit  $g$ .

The C-morphic circuit  $G$  is recoverable for single faults if  $g$  in Figure 10 is designed according to theorem 5 and corollary 8.

The circuit corresponding to a self-dual  $g$  in Figure 10 is much more economical in hardware than the circuit in Figure 9. Fortunately, many computer circuits are self-dual.

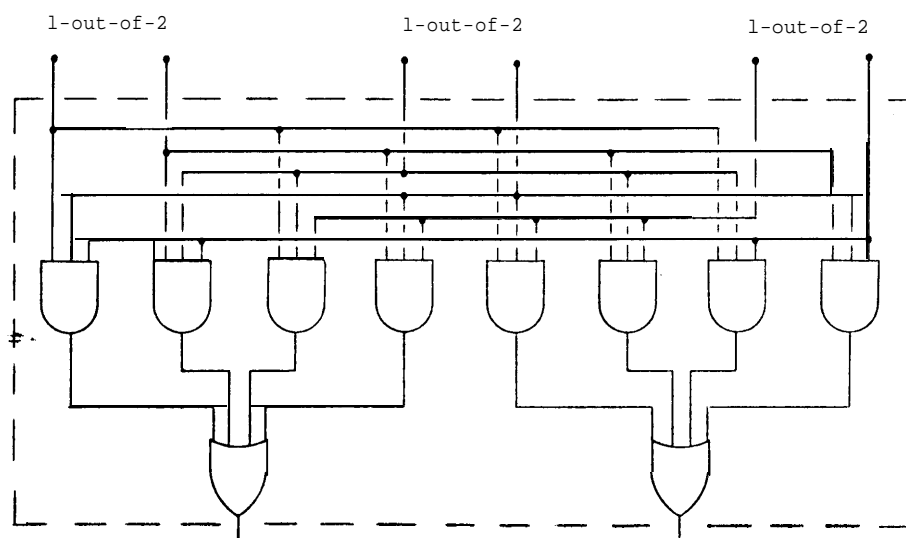
An error in the output of a self-checking circuit is detected by the observation of a **noncode** word. In fact, a **noncode** output is detected by a code checker. The checker for a C-duplication code checks for the equality of duplicated bits. Errors due to checker faults may also be detected, then corrected by an ADR if the checker satisfies the conditions of theorem 2.

To be secure for a single fault on an output line, the output of a checker must be coded. We choose a two-rail code for the checker output with the convention that a code output of  $\langle 01 \rangle$  or  $\langle 10 \rangle$  signals a no-error condition, while a **noncode** output of  $\langle 00 \rangle$  or  $\langle 11 \rangle$  initiates ADR.

To satisfy theorem 2, a checker may be viewed as a C-morphic circuit mapping code word members of every input MEC into code

word members of one output MEC,  $\{<01>, <10>\}$ .

The two-rail checker in Figure 11 was reported in [Anderson, 1971]. The checker is self-checking for single faults [Anderson, 1971], and self-dual for the set of normally applied inputs. It is recoverable for single faults by theorem 5 and corollary 8.



**Fig. 11.** - A recoverable, two-rail checker.

A two-rail checker for any odd number of two-rail inputs can be formed by constructing a tree of the checkers in Figure 11. A duplication checker is formed by adding inverters to one of each pair of two-rail inputs. A duplication checker constructed in this manner is self-checking for single faults [Anderson, 1971]. It is self-dual and recoverable for single faults by theorem 5.

The checker for 17 duplicated bits in Figure 12 satisfies the conditions of theorem 2.

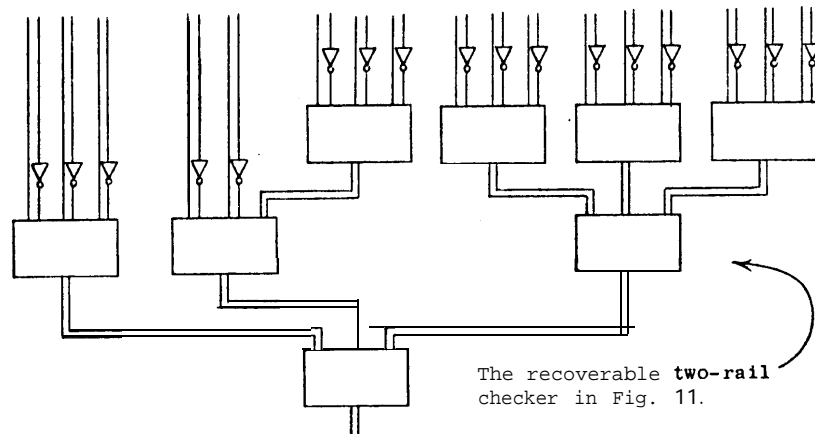


Fig. 12. - A recoverable checker for 17 duplicated signals.

The use of a duplication code makes self-checking design easier, but at a cost of duplicated hardware. A **single-error-detecting** parity code uses much less hardware.

The C-parity code is formed by appending an invert bit  $i$  and a parity bit  $p$  to a simplex vector. The simplex vector must have an even number of bits, so that the parity code word has an even number of bits. An even number of bits is required for consistency with the C-morphic system. The C-parity code may use even or odd parity over all bits. Assume odd parity.

Example 3. The message "5" is represented in a simplex system by the vector  $\langle 0101 \rangle$ . The C-parity code words representing "5" are  $\langle 100101 \rangle$  and  $\langle 011010 \rangle$ , where the parity and invert bits are the most significant and the next most significant bits respectively.

A two-level implementation of a 3-input exclusive-or (XOR) circuit is self-dual and recoverable for single faults by corollary 8. A C-morphic checker for an even-bit parity code can be constructed by forming two independent trees of the 3-input XOR circuits mentioned above. Any odd number of code word bits are inputs for one tree, while the remaining odd number of code word bits are inputs for the second tree. If the parity code uses even parity, then an inverter is added to the output of one of the trees.

A parity checker so constructed is self-checking for single faults [Anderson, 1971]. It is also self-dual and recoverable for single faults by theorem 5 and corollary 8.

## A C-MORPHIC PROCESSOR

This section illustrates the design of a dual, C-morphic processor system. The design guidelines for a C-duplication code apply, since error-detection is provided by comparing the results of two C-morphic processors operating in synchronization. The processors correspond to modules G1 and G2 in Figure 8. The correction of any error due to single faults in the data paths and the memory is achieved by an ADR.

An ADR achieves error-correction in an operation that applies stored binary vectors to a circuit for processing and observation of the result. This suggests that an ADR is best utilized at the level of primitive operations executed by a single microinstruction.

A microinstruction retry capability imposes design constraints additional to those imposed by the recoverability property. An important problem is the loss of an operand for retry due to the over-writing of a register [Maestri, 1972]. The approach taken here is to insert an intermediate latch at the output of the processing circuit. The result in the intermediate latch can be verified to be correct before it is returned to the registers.

We note here that a stuck-at fault can be neutralized only if the logic value assumed by a line is supposed to remain constant for the duration of an operation. This necessarily excludes faults on lines with changing logic values from the set of faults for which

a C-morphic processor is recoverable. Faults on clock and timing lines for example, cannot be neutralized.

Figure 13 is a simplified diagram of the data paths of a microprogrammed, C-morphic processor. Data paths are 16 bits plus 1 invert bit. An exception is the A and B busses which are 16 bits each plus 1 shared invert bit. The uncomplemented (complemented) output of each latch in a register is placed on the AU, BU, or RU (AC, BC, or RC) busses. Registers M and D are the memory address and memory data registers.

The execution of a micro-instruction is generally accomplished in two phases. In phase 1 operands are applied to the processing circuits via busses A and B and the result is stored in the intermediate latch IL. In phase 2 the result in the intermediate latch is applied to the shifter and the shifted result is stored in a destination register. Registers may be latch registers.

The C-morphic processor operates in two modes of message representation, controlled by the toggle flip-flop T. When  $T = 0(1)$ , messages are represented by the uncomplemented (complemented) member of each MEC. Recall that the invert bit of an uncomplemented (complemented) member of a MEC is 0(1). In normal operation, T is toggled at the completion of each micro-instruction, thus insuring a roughly equal proportion of uncomplemented and complemented representations in storage. Both representations must be normally applied to circuits to insure the self-testing property.



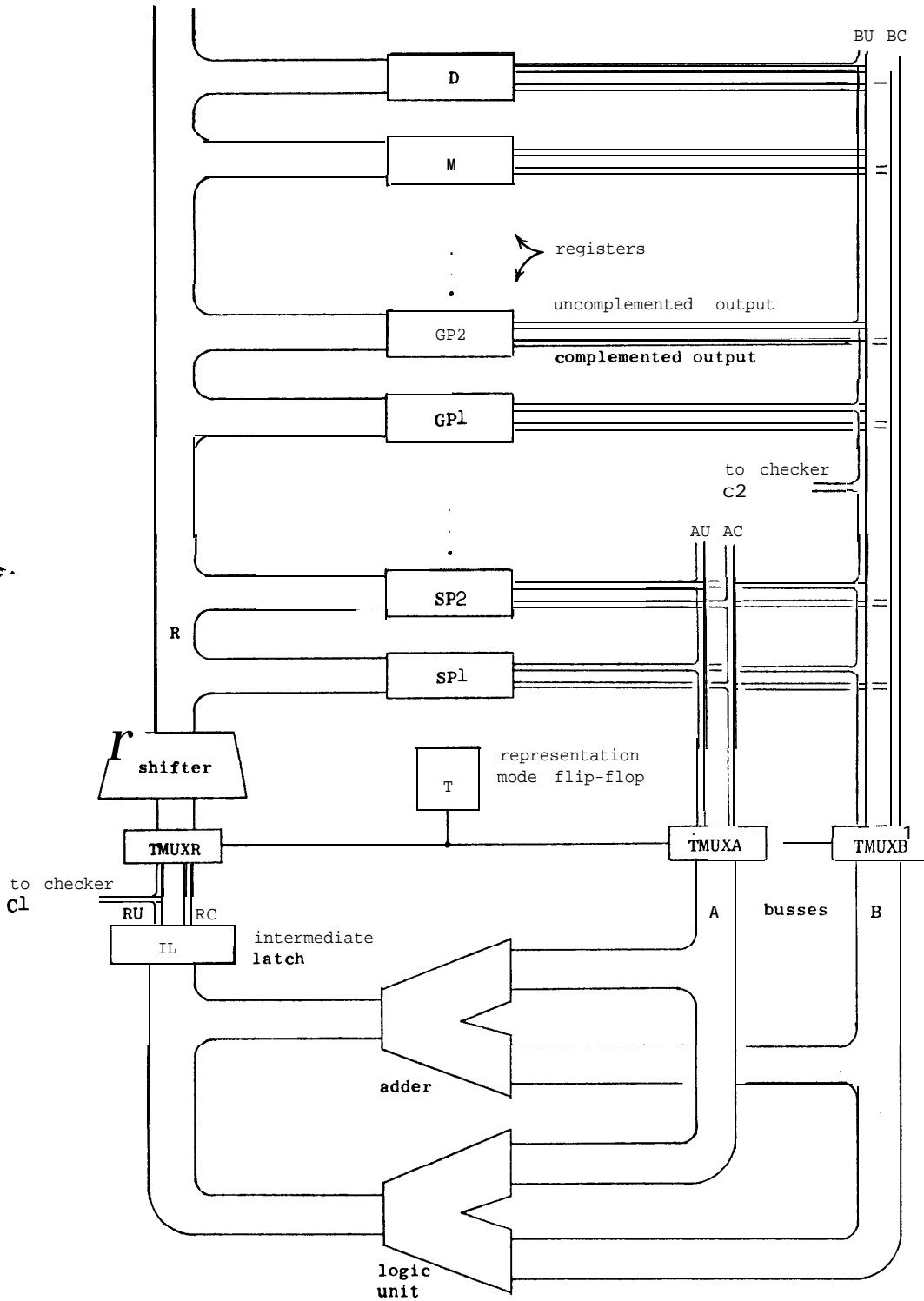


Fig. 13. - A simplified diagram of the data paths of a C-morphic processor.

The translation multiplexers TMUXA, TMUXB, and TMUXR are controlled by the flip-flop T. The buses A and B must both carry uncomplemented members when  $T = 0$  or both carry complemented members when  $T = 1$ , since A and B share the same invert bit. Figure 14 illustrates the detail of TMUXA and TMUXB. TMUXR is identical to TMUXB.

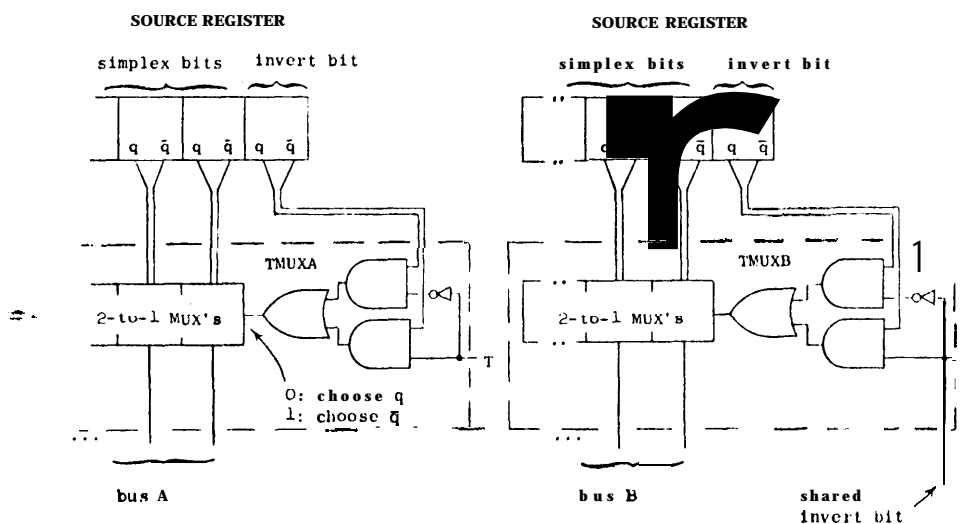


Fig. 14. - Detail of TMUXA and TMUXB.

It is easy to verify that if the latches of the source registers are in error-free states, then a single fault in Figure 14 causing an error in busses A or B when  $T = 0(1)$  will not cause an error when  $T = 1(0)$ .

The duplication checkers C1 and C2 are connected to the busses RU and BU in both processors. Figure 12 illustrates the design of the checkers.

The execution of a micro-instruction is described by algorithm 2.

Algorithm 2: Execution of a micro-instruction.

1. Execute phase 1 (T is 0 or 1)
2. If checker C1 indicates no error in the vector stored in IL, then go to 5.
3. Toggle T and re-execute phase 1.
4. If checker C1 still indicates an error, signal the failure of the ADR and exit.
5. Execute phase 2.
6. If checker C2 indicates no error in the vector stored in the destination register, then toggle T and exit.
7. Toggle T and re-execute phase 2.
8. If checker C2 still indicates an error, signal the failure of the ADR and exit.

The C-morphic processor is recoverable for any single fault in a data path from source registers to intermediate latch IL to checker C1 in phase one, or a data path from IL to destination register to checker C2 in phase two. The entire processor is actually recoverable for much more than just single faults. For example, the processor can continue to operate correctly with one stuck bit in every register, or with single faults in the adder and shifter.

Functional circuits for the C-morphic processor may be designed according to Figure 9 or Figure 10. Two examples follow.

The inputs to an adder bit position  $j$  are the operand bits  $A_j$  and  $B_j$  and carry-in  $C_j$ . The outputs are the sum bit  $S_j$  and the carry-out  $C_{j+1}$ . The adder functions are:

$$S_j = A_j \oplus B_j \oplus C_j \quad (1)$$

$$C_{j+1} = A_j B_j + A_j C_j + B_j C_j. \quad (2)$$

The carry bits  $C_0$  (least significant) through  $C_{n-1}$  in an  $n$ -bit adder form a carry vector.

Circuits implementing equations 1 and 2 are both self-dual. It follows that an  $n$ -bit adder is self-dual for the sum and the carry vector; the sum and carry vector are complemented when the operands and  $C_0$  are complemented.

Figure 15 illustrates a C-morphic, carry look-ahead adder.

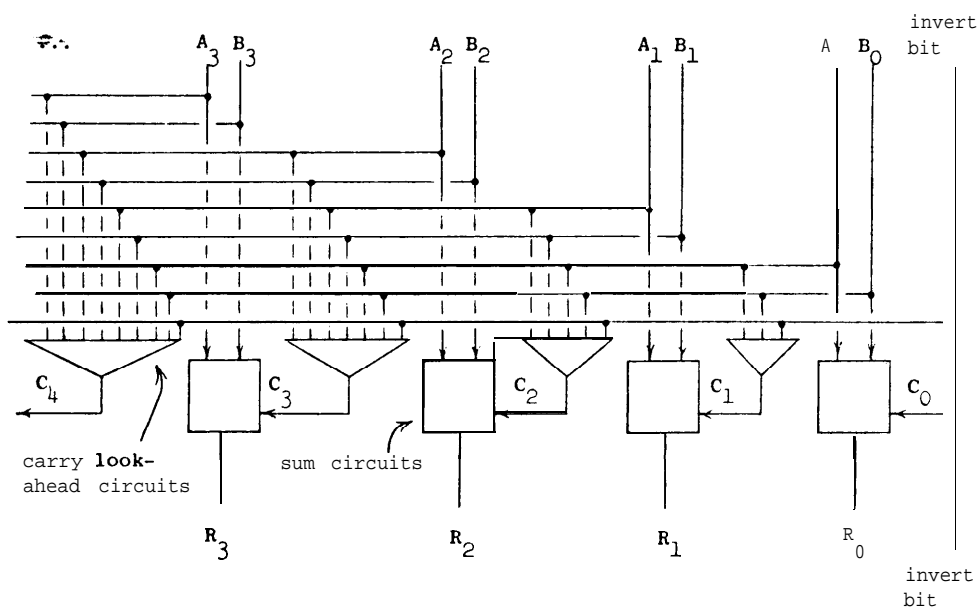


Fig. 15. - A C-morphic adder with carry look-ahead.

The carry look-ahead circuits [Morris and Miller, 1971] in Figure 15 are also self-dual. If the sum circuits and carry look-ahead circuits are designed according to corollary 8, then by theorem 5, the C-morphic adder in Figure 15 is recoverable.

Unlike an adder, the circuits for logic operations AND and OR are not self-dual. Figure 16 illustrates a C-morphic AND circuit designed according to Figure 9.

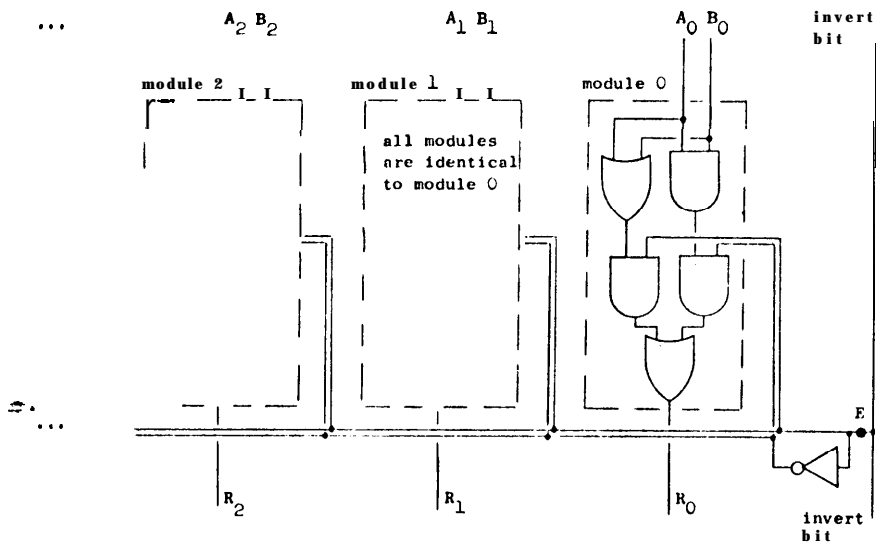


Fig. 16. - A C-morphic AND(OR) circuit.

Each module in Figure 16 is self-dual and recoverable for single faults. The entire C-morphic AND circuit is recoverable by theorem 5. A C-morphic OR circuit is formed by inserting an inverter at point E in Figure 16.

Faults sticking a single bit in a memory location can be neutralized by the following algorithm. The algorithm applies for any C-morphic system of message representation.

Algorithm 3: Fault-neutralization in memory.

1. Write the member W of a given MEC.
2. Read W immediately

3. If an error in storage has occurred, write the alternate member of the same MEC, W.

Algorithm 3 is implicitly used for storage in registers by the execution of algorithm 2. Algorithm 3 can correct errors only if the fault exists before an attempted storage.

Note that members of a MEC in the memory address register M must be translated into a simplex address for use by the memory. The C-morphic processor is necessarily not recoverable for faults in the output lines of the morphic-to-simplex translator.

Dual (non-morphic) processors have been used for the purpose of error detection in high-availability systems [Ressler, 1973].

When a mismatch is detected, each processor must interrupt and run programs for self-diagnosis. Since transient faults are commonly believed to be much more frequent than permanent faults, a retry capability is recommended for such systems. If an error due to a transient fault is corrected by retry, then a system interruption for self-diagnosis can be avoided.

Self-diagnosis by a permanently faulty processor is not guaranteed to be accurate. If both processors agree as to which is faulty, then the faulty processor is purged while the other continues. In this case, the error-detection capability is lost and data contamination can occur. If the processors disagree as to which is faulty, then the decision to purge the processor cannot be made reliably. Either choice risks data contamination.

An ADR capability enhances the availability of a dual processor system. An ADR is guaranteed to correct an error in the recoverable processor, restore duplication between processors, and allow the continuation of dual processor service. Thus a fault for which the processor is recoverable neither halts the dual system, nor forces it to operate in a single processor mode where error-detection is lost. As indicated, the C-morphic processor in Figure 13 can continue morphically correct operation even for certain multiple faults.

The additional hardware cost for this ADR capability, over that of a dual processor with a simple retry capability, is low. The cost in time is modest. Extra propagation delays are usually incurred due to the morphic circuits. Also, a permanent fault is expected to trigger frequent retrys. For example, a stuck-at fault on a bus line that is 0 half the time and 1 half the time causes an error half the time in one execution phase. Since an error-correction requires an ADR, the effective execution rate is reduced to 80 per cent of the normal rate, assuming equal execution time for both phase 1 and phase 2. Such degraded performance, however, may be preferable to a system shutdown or a contaminated data base.





## CONCLUSIONS

An alternate-data retry (ADR) is a fundamentally new way to achieve automatic error-correction in a high-availability system. An ADR capability insures the continuation of error-free service despite the occurrence of a fault, thereby allowing the postponement of repair until a more convenient time.

The application of an ADR capability to the memory and the data paths of a dual processor was illustrated. Other experimental designs indicate that such a capability may be extended to certain control lines and parts of the micro-program sequencer.

The additional hardware cost of an ADR capability over that of a single retry capability is relatively low. An investigation is continuing into the design of a single C-morphic processor that would satisfy the requirements for an ADR capability. Such a design might impose the recoverability property on the low-cost self-checking circuits discussed in [Wakerly, 1973]. The C-parity code is one candidate for use in such a processor. For example, a recoverable, C-morphic adder has been designed using the predicted parity techniques in [Sellers, 1968]. This adder is self-checking and recoverable for a suprisingly low hardware cost.

## REFERENCES

- [Anderson, 1971] Anderson, D.A., "Design of Self-Checking Digital Networks Using Coding Techniques," Tech. Rep. R-527, Coordinated Science Lab., Univ. of Ill., Sept. 1971.
- [Avizienis, 1969] Avizienis, A, "Digital Fault Diagnosis by Low-cost Arithemetical Coding Techniques," Proc. Purdue Centennial Year Symp. Infor. Processing 1, 1969.
- [Avizienis, 1971] Avizienis, A. "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," IEEE TC, Vol C-20, Nov. 1971.
- [Carter, 1974] Carter, W.C., et al, "Design of Serviceability Features for the IBM System/360," IBM J. Res. Develop., Vol. 8, No. 2, April, 1964.
- [Carter and Schneider, 1968] Carter, W.C., and P. R. Schneider, "Design of Dynamically Checked Computers," IFIP 68, Vol. 2, Edinburg, Scotland, Aug. 1968.
- [Carter, et al, 1970] Carter, W.C., et al, "Design Techniques for Modular Architecture for Reliable Computer Systems", IBM Res. Rep. RA 12, March, 1970.
- [Carter, et al, 1972] Carter, W.C., et al, "Computer Error Control by Testable Morphic Boolean Functions - A Way of Removing Hardcore", IBM Res. Rep. RC 3099, June, 1972.
- [Chandy, et al, 1975] Chandy, et al, "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," IEEE Trans. on Software Eng., Vol SE-1, March, 1975.
- [Dickinson, et al, 1964] Dickinson, M.M., et al, "Saturn V Launch Vehicle Digital Computer and Data Adapter," Proc. Fall Joint Comp. Conf., 1964.
- [Fox, 1975] Fox, J.L., "Availability Design of the System/370 Model 168 Multiprocessor," IBM Rep. TR 00.2590, Jan. 9, 1975.

- [Maestri, 1972] Maestri, G.H., "The Retryable Processor," Proc. Fall Joint Comp. Conf., 1972.
- [Morris & Miller, 1971] Morris, R.L., and J.R. Miller, Designing with TTL Integrated Circuits. McGraw-Hill 1971.
- [Peterson & Weldon, 1972] Peterson, W.W., and E.J. Weldon, Error Correcting Codes, Cambridge, MIT Press, 1972.
- [Ressler, 1973] Ressler, B.E., "Design of a Dual Computer Configuration for Redundant Computation," Tech. Rep. T-586, Draper Lab. MIT, June, 1973.
- [Roth, et al, 1967] Roth, J.P., et al "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits," IEEE TC, Vol. EC-16, Oct., 1967.
- [Sellers et al, 1968] Sellers, F.F., et al, Error Detecting Logic for Digital Computers, McGraw-Hill, 1968.
- [Shedletsky, 1976] Shedletsky, J.J., "A Rollback Interval for Networks with an Imperfect Self-Checking Property", Dig. 6th Int. Symp on Fault-Tolerant Computing, June, 1976.
- [Wakerly, 1973] Wakerly, J.F., "Low Cost Error Detection Techniques for Small Computers," Tech Rep. 51, Digital Sys. Lab., Stanford University, Dec., 1973.

