question. In any case, it is clear that minimal length test sequences for M_2 are asymmetric in that they always contain runs of reads or writes of different lengths. This is unexpected in view of the symmetry of both M_2 and the SPSF fault model. It also suggests that the symmetry inherent in the test sequences produced by standard memory testing programs may not always be desirable. Asymmetric tests like $\mathscr{T}_2^*$ have the drawback that they require more complex test generation programs to produce them. It is easy, however, to construct near-minimal symmetric SPSF tests for M_2. One such test can be formed by inserting two reads after every write in (3). The resulting test is of length 48, which exceeds the minimum possible length by at most 50 percent. The complexity of the analysis required to construct $\mathscr{T}_2^*$ and prove its optimality suggests that the general SPSF test length minimization problem is very difficult.

It may be useful to extend the SPSF model by allowing a cell to have more than one neighborhood. For example, GALPAT and other "ping-pong" tests implicitly use 2-cell neighborhoods of the form C_i, C_j where for each i, j is allowed to range over the entire memory. In a similar way, we can apply an SPSF test such as $\mathscr{T}_2^*$ in ping-pong fashion to detect SPSF's involving all $\binom{n}{2}$ possible cell pairs C_i, C_j in M_n. The approximate length of such a test is $36\binom{n}{2} = 18(n^2 - n)$, which compares favorably with GALPAT II whose length is $8n^2 - 4n$ [1]. As noted earlier, it is difficult to compare the fault coverage of SPSF tests and commercial tests like GALPAT, since the latter do not have well-defined fault models.

References

[1] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems.* Woodland Hills, CA: Computer Science Press, 1976.
[2] J. P. Hayes, "Detection of pattern-sensitive faults in random-access memories," *IEEE Trans. Comput.*, vol. C-24, pp. 150–157, Feb. 1975.
[3] V. P. Srini, "API tests for RAM chips," *Computer*, vol. 10, no. 7, pp. 32–35, July 1977.
[4] J. Knaizuk, Jr. and C. R. P. Hartmann, "An algorithm for testing random access memories," *IEEE Trans. Comput.*, vol. C-26, pp. 414–416, Apr. 1977.
[5] J. Knaizuk, Jr. and C. R. P. Hartmann, "An optimal algorithm for testing stuck-at-faults in random access memories," *IEEE Trans. Comput.*, vol. C-26, pp. 1141–1144, Nov. 1977.
[6] S. M. Thatte and J. A. Abraham, "Testing of semiconductor random access memories," in *Proc. 7th Int. Fault-Tolerant Computing Symp.*, Los Angeles, CA, pp. 81–87, June 1977.
[7] R. Nair, S. M. Thatte, and J. A. Abraham, "Efficient algorithms for testing semiconductor random-access memory," *IEEE Trans. Comput.*, vol. C-27, pp. 572–576, June 1978.
[8] F. Harary, *Graph Theory.* Reading, MA: Addison-Wesley, 1969.
[9] S. W. Golomb, *Polyominoes.* New York: Scribner's, 1965.
[10] ——, "Tiling with polyominoes," *J. Comb. Theory*, vol. 1, pp. 280–296, 1966.

A Hardware Redundancy Reconfiguration Scheme for Tolerating Multiple Module Failures

STEPHEN Y. H. SU AND EDGAR DUCASSE

Abstract—**This paper deals with a method for designing a digital system which will remain operational in spite of the failure of some of its components. A scheme and its realization are presented for automatically reconfiguring a 5MR (five modular redundancy system or 5-input majority voting system) into a triple modular redundancy (TMR) system under a single or double module failures. The scheme can tolerate a double fault followed by a single fault which can neither be tolerated by a 5MR nor by a hybrid redundancy system with a TMR core. It uses no spare units and the circuit realization is relatively simple. The modular structure of the logic design for the proposed scheme should make the testing of the system easier. The scheme can be used in both binary and multivalued systems.**

Index Terms—**Fault-tolerant computing, fault-tolerant design, fault-tolerant systems, hybrid redundancy, multiple failures, multiple-valued logic, *N*-modular redundancy, redundancy scheme, self-repair systems, system design.**

Manuscript received August 22, 1977; revised July 23, 1979. This work is supported by the Division of Mathematical and Computer Science, National Science Foundation under Grant MCS 78-24323. The primary version of this paper has been presented at the 1975 International Computer Symposium, Taipei, Taiwan, Aug. 22-26, 1975.

S. Y. H. Su is with the Department of Computer Science, School of Advanced Technology, State University of New York at Binghamton, Binghamton, NY 13901.
E. DuCasse is with the Department of Information Systems, Lubin School of Business Administration, Pace University, New York, NY 10038.

I. Introduction

One method for achieving fault-tolerant computing is the utilization of redundant hardware components. Since the cost of hardware has been and is still going down rapidly, fault-tolerant design using redundant components has become more feasible than ever.

Basically, there are three types of redundancy techniques. 1) Static (masking massive) redundancy [1] masks the fault(s) by using a majority gate whose output is identical to the majority of the inputs. An example of this type of redundancy is triple modular redundancy (TMR) in which the outputs of three logic modules feed into a majority gate. As long as only one module fails, the output of the majority gate will have the same value as the two fault-free modules. Therefore, any single module failure will be masked. A system containing N logic modules whose outputs are connected to an N-input majority gate is called NMR or an N-input majority voting system (MVS). 2) Dynamic redundancy (stand-by spares) [2] uses one active module and several spare modules. Switching circuits are needed for detecting and switching out the faulty active module, and switching in a good spare module. 3) Hybrid redundancy [3] uses a combination of static and dynamic redundancy techniques. This type of network contains a majority voting system (called the "core") and some spare modules. Spares replace the faulty modules so that a fault-free majority voting system can be maintained until all the spares are used.

An adaptive voting scheme using a threshold gate has been presented by Pierce [4]. Goldberg *et al.* [5] have presented an interesting improved realization for switched-adaptive voting which uses a majority gate instead. Siewiorek and McCluskey [6] have presented an interactive cell switch design for hybrid redundancy. A study on fault tolerance of the iterative cell array switch has been made by Ogus [7].

The interesting idea of self-purging redundancy was introduced by Chandy *et al.* [8]. Losq [9] has recently carried out the logic design and calculated the reliability of self-purging systems. The basic idea of the self-purging technique is to let all modules participate in the voting. A faulty module is then switched off (produced a 0) and will no longer participate in the voting. The voter is a threshold gate. The self-purging technique is not suitable for tolerating multiple failures. For instance, consider a system with 5 modules. If the threshold of the voter is 3, the output

$$Z = x_1[x_2(x_3 + x_4 + x_5) + x_3(x_4 + x_5) + x_4x_5] + x_2[x_3(x_4 + x_5) + x_4x_5] + x_3x_4x_5.$$

If two modules fail (simultaneously or sequentially), say x_1 and x_2, then $x_1 = x_2 = 0$, and $z = x_3x_4x_5$; i.e., any double fault will change the majority gate to an AND gate. As a result, the system can no longer tolerate any stuck-at-0 fault. If the threshold of the voter is 2, then it cannot tolerate a simultaneous double stuck-at-1 fault.

Recently, two redundancy schemes for optimum multiple fault tolerance have been introduced by Losq [10]. The basic idea of the first scheme is similar to Pierce's scheme except that the threshold of the threshold gate has been changed from 1 to N. The second scheme forces the output of every other faulty module to 1 instead of 0. The drawback of the second scheme is that when two modules fail simultaneously, a 5-modular redundancy system will be changed to an OR gate which cannot tolerate any stuck-at-1 fault.

In this correspondence, a scheme and its realization are presented which will automatically reconfigure a 5-modular redundancy system into a TMR when two modules fail simultaneously. When only a single fault occurs, the system will become a TMR with a spare. The spare module will replace the faulty module in the TMR. The failure of a third module will be tolerated and there is a 50 percent chance that the system will tolerate a fourth failure.

This scheme is superior to the 5MR system since the scheme can tolerate either a double fault followed by a single fault or three single faults occurring one at a time which cannot be tolerated by the 5MR. The proposed scheme is better than the hybrid redundancy with a TMR core and two spares because the latter can only tolerate one single fault in the TMR core at any instant.[1]

This scheme can easily be generalized for handling a multivalued 5-modular system (see [11]).

II. The Reconfiguration Scheme

In this section, we present a method for reconfiguring a 5MR to a TMR. The same idea can be used for designing an NMR with any number of inputs. However, NMR's with $N \geq 7$ are seldom used. This scheme has been generalized for 7MR and the result is available from the first author.

A. Reconfiguration of 5MR

For a five-modular redundancy system, a maximum of two failures can be tolerated. Using the technique described in this section, it is possible to tolerate the failure of at least three modules.

Let us consider a 5 module system with binary variables $x_1, x_2, \cdots, x_5$ as the outputs for the modules, where $x_i \in \{0, 1\}$ for all i. If the x_i's feed into a majority gate, then the output of the majority gate z can be expressed as

$$\begin{aligned} z &= M(x_1, x_2, \cdots, x_5) \\ &= x_1(x_2(x_3 + x_4 + x_5) + x_3(x_4 + x_5) + x_4x_5) \\ &\quad + x_2(x_3(x_4 + x_5) + x_4x_5) + x_3x_4x_5. \end{aligned} \tag{1}$$

Substituting $x_1 = 0$ and $x_2 = 1$ into the above equation, we obtain

$$z = M(0, 1, x_3, x_4, x_5) = x_4x_5 + x_3x_4 + x_3x_5 = M(x_3, x_4, x_5) \tag{2}$$

where the term $x_3x_4x_5$ has been removed by absorption. From the above observation, since the majority function is symmetric, we see that a TMR is easily obtained from a 5MR by replacing any variable by 0 and any other variable by 1. For example, if module number 1 is faulty, by letting x_1 be a 0 and x_3 be a 1, we can obtain a TMR with inputs x_2, x_4, x_5.

Based on this idea, we can design a logic network for automatically locating single or double faults and reconfiguring a 5MR to a TMR. In this correspondence, we shall focus our attention on the case where the majority gate and reconfiguration circuit (A, B, and M in Fig. 1) are fault-free.

The block diagram for the automatic reconfigurable system is shown in Fig. 1. Block A consists of five "equivalence detectors."

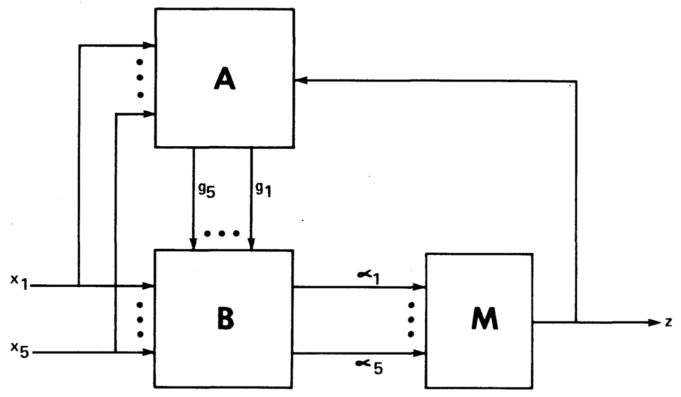

Fig. 1. Block diagram for the automatic reconfigurable system.

Each has two inputs, x_i and z, and one output g_i, where $i = 1, 2, \cdots, 5$. The output $g_i = 1$ if the ith module is faulty, that is, x_i has a logic value different from z. In this case, Block B inhibits the signal x_i and prevents x_i from transferring to α_i. Thus, α_i becomes stuck-at-0. This removes the faulty module from active participation in the voting. The lead $\alpha_{i+1(\text{mod } 5)}$ is forced to become temporarily stuck-at-1. Thus, we obtain a spare. The remaining α_i's being unaffected by the flip-flop, allow the x_i to transfer to α_i. If both the ith and kth modules fail either simultaneously or sequentially, Block B causes $\alpha_i = 0$, $\alpha_k = 1$ for $k = i + 1$ or $i + 2$ and $\alpha_i = 1$, $\alpha_k = 0$ for $k = i - 1$ or $i - 2$ and again a perfect TMR system results. Suppose x_i fails first, then $g_i = 1$ yielding $\alpha_i = 0$ and $\alpha_{i+1} = 1$. If x_k fails next, where $k = i - 1$, then $\alpha_k = 0$ and α_i changes from 0 to 1, and α_{i+1} changes from 1 to x_{i+1}.

B. Design of Block A

Fig. 2 shows the logic diagram for Block A. Initially all RS flip-flops are reset. If module i is faulty then z and x_i are different from each other and hence $S_i = 1$ sets flip-flop i when clock pulse p occurs. Therefore, $g_i = 1$.

C. Design of Block B

In this subsection, we present an informal approach to the design of network B by deriving the α's in terms of the x's and g's. If all modules are fault-free then $g_i = 0$ for all i and network B should cause $\alpha_i = x_i$ for all i—a direct transmission from x_i's to α_i's. The function α_i is equal to the disjunct of $x_i\bar{g}_i$ and terms which are "zeroes" when all $g_i = 0$ and which determine the behavior of α_i when faults are present. The system with single and double faults is considered below.

Since the majority function is a symmetrical function, the 5-module majority function can be expressed by the following equation:

$$\begin{aligned} M(x_1, x_2, x_3, x_4, x_5) &= \sum_{j=1}^{5} x_j x_{j-1(\text{mod } 5)} x_{j-2(\text{mod } 5)} \\ &\quad + \sum_{j=1}^{5} x_j x_{j-1(\text{mod } 5)} x_{j-3(\text{mod } 5)}. \end{aligned}$$

For example, all double faults affect one of the module pairs $(x_i, x_{i-1(\text{mod } 5)})$ or $(x_i, x_{i-2(\text{mod } 5)})$ since $(i - 3) - 2 = i(\text{mod } 5)$.

For ease of understanding, we shall design Block B by deriving informally the equation for the system with a single failure. Then we shall derive the equation for the system with double failures and show that the same equation can be used for the single failure case. In each case, the equation is constructed informally by asking the question: "What terms are needed for α_i in order to satisfy each case?" Formal derivation of the equations can be obtained from the first author.

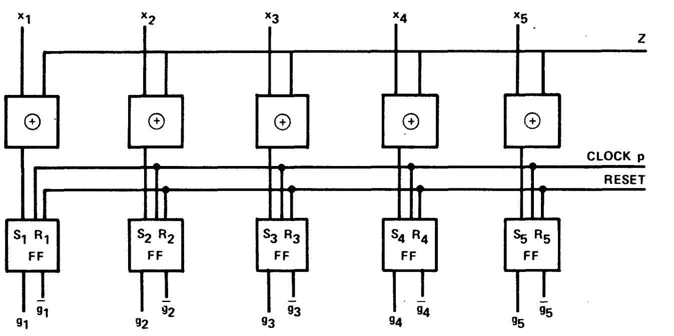

Fig. 2. Logic diagram for Block A.

1) *System with Single Failures*

For example, if the module m_1 is faulty, then network B should cause $\alpha_1 = 0$ and $\alpha_2 = 1$ and $\alpha_i = x_i$ for $i = 3, 4, 5$ so that the 5MR becomes a TMR. If module m_5 is faulty, then $\alpha_5 = 0$ and $\alpha_1 = 1$. Since the majority function is a symmetric function, network B can be synthesized using a ring-structure type of logic network. Therefore, α_i can be considered the same as $\alpha_{5+i(\text{mod } 5)}$, that is, the addition is to be taken as a modulo five addition, $\alpha_{i+5(\text{mod } 5)} = \alpha_i$. Fig. 1 can then be shown in more detail as Fig. 3. Block A in Fig. 2 is the logic network inside the five smallest circles in Fig. 3. Blocks B_1 to B_5 in Fig. 3 constitute Block B of Fig. 1. The center point of the circles in Fig. 3 represents the output of the majority gate z.

In general, if the ith module is the only faulty module, then $g_i = 1$ and $g_j = 0$ for $j \neq i$. We want network B to satisfy three requirements: 1) $\alpha_i = 0$, 2) $\alpha_{i+1(\text{mod } 5)} = 1$, and 3) $\alpha_k = x_k$ for all $k \neq i, i+1$. We now derive the equation for α_i as a function of the g_i's and x_i's. Recall that g_i is the only one out of five g_i's whose value is logic 1. Thus if g_i is an implicant of α_{i+1} (i.e., g_{i-1} is an implicant of α_i), then requirement 2) $\alpha_{i+1(\text{mod } 5)} = 1$ will be satisfied. The requirements 1) and 3) will be satisfied if α_i contains the product term $x_i\bar{g}_i$. Therefore, the following equation satisfies all three requirements:

$$\alpha_i = x_i\bar{g}_i + g_{i-1}. \tag{3}$$

2) *System with Double Faults*

Due to the ring structure of the reconfiguration network as shown in Fig. 3, any two "nonadjacent" faulty modules in the set $\{m_1, m_2, m_3, m_4, m_5\}$ can be considered as separate by a "distance" of 2, i.e., if one faulty module is m_i, then the other one can be considered as m_{i-2}. Suppose modules number 1 and number 4 are faulty, then we can consider $i = 1$ and $i - 2(\text{mod } 5) = -1(\text{mod } 5) = 4$. Therefore, if there is a double fault among the modules, we need only consider two cases.

Case 1: Nonadjacent Double Fault

The $(i-2)$th and ith modules are faulty. The term $g_i g_{i-2}$ appearing in a disjunction representing α_i will guarantee that $\alpha_i = 1$.

Case 2: Adjacent Double Fault

If module i and module $i - 1$ fail, the term $g_{i-1} \cdot \overline{\sum_{k \neq i-1,i} g_k}$ will assure $\alpha_i = 1$.

If neither module $i - 1$ nor i fails, the term $x_i\bar{g}_i$ will guarantee (in the absence of terms other than those described above) that $\alpha_i = x_i$. The following representation for α_i will thus assure the desired performance of the reconfiguration network:

$$\alpha_i = g_{i-1} \cdot \overline{\sum_{k \neq i-1,i} g_k} + x_i\bar{g}_i + g_i g_{i-2}. \tag{4}$$

The logic diagram for realizing α_i (each of the five square blocks B_i in Fig. 3) is shown in Fig. 4.

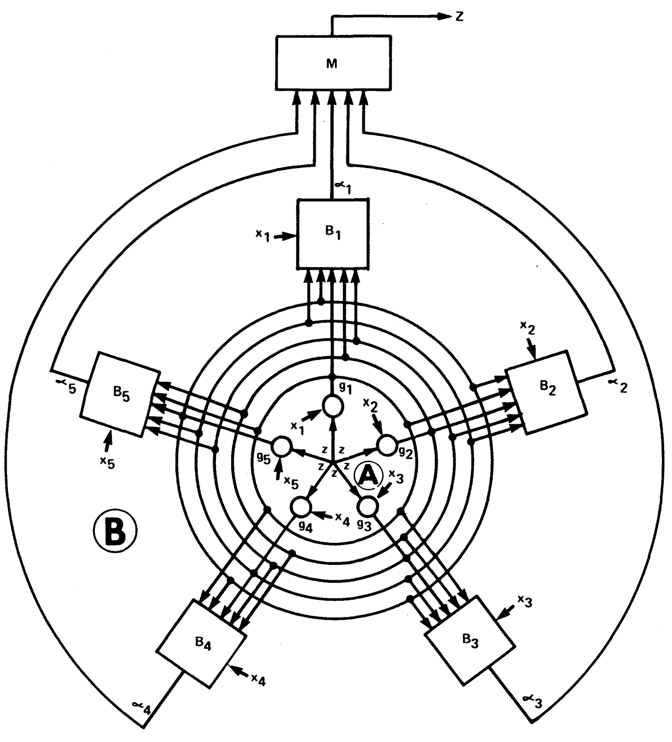

Fig. 3. Detailed diagram for the system in Fig. 1.

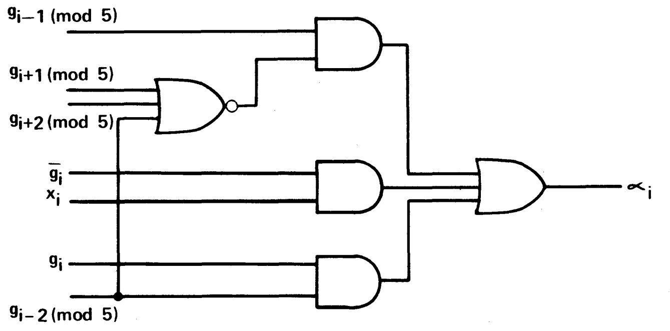

Fig. 4. Logic diagram of each B_j in Fig. 3.

Example 1: To verify the above equation, let us consider the case where both m_1 and m_3 are faulty. Substituting $g_1 = g_3 = 1$ and $g_2 = g_4 = g_5 = 0$ into (4), we obtain

$$\alpha_1 = g_5\overline{(g_2 + g_3 + g_4)} + x_1\bar{g}_1 + g_1 g_4$$
$$= 0 + 0 + 0 = 0$$

and

$$\alpha_3 = g_2\overline{(g_4 + g_5 + g_1)} + x_3\bar{g}_3 + g_3 g_1$$
$$= 0 + x_3 \cdot 0 + 1 \cdot 1 = 1.$$

Let us now verify that (4) works for a 5MR with a single faulty module. For a single fault, one and only one g, say g_i, is equal to 1, while the other g's are 0's. In this case, $g_k = 0$ for all $k \neq i - 1, i$, and in (4) the term

$$\overline{\sum_{k \neq i-1,i} g_k} = 1. \tag{5}$$

Also,

$$g_i g_{i-2} = 0. \tag{6}$$

TABLE I

Faults	Effect on Proposed Scheme	Effect on 5MR	Effect on Hybrid Redundancy with TMR Core
1	Reconfigure to TMR	None	None
2 simultaneously	Reconfigure to TMR	None	Failure
2 in sequence	Reconfigure to TMR	None	None
2 simultaneously, followed by 1	OR gate or AND gate, OK	Failure	Failure
3 in sequence	OR gate or AND gate, OK	Failure	None
1 followed by 2	Failure	Failure	Failure
3 simultaneously	Failure	Failure	Failure
4 after surviving 3 s-a-1	OK if AND gate (50% of time)	Failure	Failure
s-a-0	OK if OR gate (50% of time)	Failure	Failure

Substituting (5) and (6) into (4), we obtain (3). Therefore, (4) is valid for either single or double faults.

Example 2: In a 5-input MVS, if module 1 fails, then $g_1 = 1$ and $g_2 = g_3 = g_4 = g_5 = 0$. From (4), $\alpha_1 = 0$, $\alpha_2 = 1$, $\alpha_3 = x_3$, $\alpha_4 = x_4$ and $\alpha_5 = x_5$. If module 4 fails next, then substituting $g_1 = g_4 = 1$ and $g_2 = g_3 = g_5 = 0$ into (4), we obtain $\alpha_1 = 1$, $\alpha_2 = x_2$, $\alpha_3 = x_3$. $\alpha_4 = 0$ and $\alpha_5 = x_5$. Therefore, the system reconfigures correctly with the three fault-free modules 2, 3, and 5 forming a TMR system.

Suppose the third module now fails. The system will change the majority gate in the TMR to an OR or an AND gate. This is shown below.

a) If module 3 fails, then substiuting $g_3 = g_4 = g_1 = 1$ and $g_2 = g_5 = 0$ into (4), we obtain $\alpha_1 = 1$, $\alpha_2 = x_2$, $\alpha_3 = 1$, $\alpha_4 = 0$, and $\alpha_5 = x_5$. Hence $z = x_2 + x_5$. The system will tolerate a fourth module failure as long as x_2 or x_5, has a stuck-at-0 failure.

b) If module 2 or 5 fails, by the same procedure as in a), we can see that the system will be reconfigured from a majority gate into an AND gate. Thus a fourth failure can be tolerated as long as it is a stuck-at-1 fault.

D. Advantages of the Proposed Scheme

From the system described above, we see that the proposed fault-tolerant digital system allows the system to reconfigure from 5MR to TMR following the failure of the first module. Initially this might strike the reader as being absurd since a majority gate with five inputs will tolerate the malfunction of not only one, but even two participating modules. The reconfiguration is accomplished in such a way, however, as to effectively hold in reserve, or make into a spare, the fault-free module which is removed from active participation in the subsequent vote at the same time that the faulty unit is suppressed. This is done by forcing the output of the faulty module, say, the ith module, to be stuck-at-0 and the output of the module next to it, that is, the $(i + 1)$th module to be stuck-at-1. Thus, the 5-input NMR becomes a TMR with the $(i + 1)$th module as a spare. If a second module, that is, one of the modules in the TMR should then develop a fault, it is seen that the system frees the $(i + 1)$th module and employs it in place of the newly found malfunctioning unit. In this way, then, the network effectively employs a spare unit when in actuality it had no spares at all. Thus, after two modules have failed sequentially, we are left with a perfect TMR system.

The proposed system can tolerate more failures than 5MR and hybrid redundancy systems.[1] This is shown in Table I. For example, in the presence of a double fault, the system can reconfigure a 5MR to a perfect TMR system. This means that the third module failure can be tolerated. Furthermore, the TMR will be reconfigured into an AND gate or an OR gate in the presence of the third failure. If the probability of stuck-at-1 failures is the same as the probability of stuck-at-0 faults, then by the proposed scheme, there is a 50 percent chance of tolerating a fourth module failure.

The second advantage is that the circuit realization of the proposed scheme is simpler than the schemes presented in the existing literature. From Figs. 2 and 4, we see that to implement the switch, only five gates[2] (or 6 NAND gates) and one flip-flop are required. Eight gates and one flip-flop are required for each module in the iterative cell switch [6]. The scheme shown in [9] requires four gates and two flip-flops per module plus a T flip-flop and an OR-gate with fan-in of N.

A third advantage is that the structure of the switching network for reconfiguration is highly modular. For example, both Blocks A and B contain five identical components. This means that the testing of these components should be easier.

E. Disadvantage of the Proposed Scheme

Although the approach taken in the proposed scheme for a five module system can be used for deriving equations for an N-module system where $N \geq 7$ and N is odd, the complexity of Block B grows with N. Such a drawback is not serious since NMR's with $N \geq 7$ are seldom used in a practical environment.

References

[1] J. von Neumann, "Probabilistic logic and synthesis of reliable organisms from unreliable components," *Automata Studies* (Annals of Mathematical Studies, no. 34), C. E. Shannon and J. McCarthy, Eds. Princeton, NJ: Princeton University Press, 1956, pp. 43–98.

[2] W. G. Bouricius, W. C. Carter, and P. R. Schneider, in "Reliability modeling for self-repairing computer systems," in *Proc. Assoc. Comput. Mach. Annu. Conf.*, pp. 295–309, 1969; also, IBM Rep. RC-3278.

[3] F. P. Mathur and A. Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair," in *Proc. Spring Joint Comput. Conf.*, 1970, pp. 375–383.

[4] W. H. Pierce, "Adaptive vote-takers improve the use of redundancy," in *Redundancy Techniques for Computing Systems*, Wilcox and Mann, Eds. Washington, DC: Spartan Books, 1962.

[5] J. Goldberg, K. N. Levitt, and R. A. Short, "Techniques for the realization of ultra-reliable spaceborne computers," Final Rep.—Phase 1, Project 5580, Stanford Res. Inst., Menlo Park, CA, Sept. 1966.

[6] D. P. Siewiorek and E. J. McCluskey, "An iterative cell switch design for hybrid redundancy," *IEEE Trans. Comput.*, vol. C-22, pp. 290–297, March 1973.

[7] R. C. Ogus, "Fault tolerance of the iterative cell array switch for hybrid redundancy," *IEEE Trans. Comput.*, vol. C-23, pp. 667–681, July 1974.

[1] Here the effects of compensating failure (a stuck-at-0 compensates for a stuck-at-1 as far as the voter is concerned) are not considered.

[2] EXCLUSIVE-OR gate is not counted since it is required in all three schemes.

[8] K. M. Chandy, C. V. Ramamoorthy, and A. Cowan, "A framework for hardware-software tradeoffs in the design of fault-tolerant computers," in *Proc. Fall Joint Comput. Conf. AFIPS*, pp. 55-63, 1972.
[9] J. Losq, "A highly efficient redundancy scheme: Self-purging redundancy," *IEEE Trans. Comput.*, vol. C-25, pp. 569-578, June 1976.
[10] ——, "Redundancy scheme for optimum multiple fault tolerance," Stanford Univ. Digital Syst. Lab. Tech. Note 33, Jan. 1974.
[11] S. Y. H. Su and E. DuCasse, "A reconfiguration scheme for tolerating multiple failures in digital systems," in *Proc. 1975 Int. Computer Symp.*, vol. II, pp. 216-222, Taipei, Taiwan.

A Digital Quarter Square Multiplier

EVERETT L. JOHNSON

Abstract—**An application of the quarter square multiplication technique used in analog computing is proposed for digital multiplication. Significant savings in storage requirements for ROM-implemented product tables are demonstrated. A two's complement multiplication circuit utilizing the digital quarter square technique is presented.**

Index Terms—**Absolute value circuit, digital quarter square multiplication, high-speed digital multiplication, ROM product tables.**

Introduction

Large-scale integrated (LSI) circuit technology has permitted the development of iterative circuit solutions to the design of high-speed multipliers [1], [2]. A survey of current high-speed multipliers and the theory of their operation has been presented by Waser [3]. In this correspondence a suggestion for an alternative approach will be presented. This alternate approach involves a variation of the straightforward application of ROM look-up tables for obtaining the product of two binary numbers.

The multiplier and multiplicand form the address which points to the memory location containing their product. The amount of storage required in an ROM implemented multiplier increases as 2^M where M is the sum of the number of bits in the multiplicand and multiplier. The large amount of storage required is due to the redundancy in the products stored, i.e., the product 6 would be stored in four different locations as the result of 2×3, 3×2, 1×6, and 6×1. It will be demonstrated in the following sections that a significant reduction in the storage requirements can be realized by utilizing ROM's and an analog computer multiplication technique called quarter square multiplication.

Quarter Square Multiplication

The quarter square multiplication technique is easily demonstrated algebraically as

$$xy = \tfrac{1}{4}\{(x+y)^2 - (x-y)^2\}. \tag{1}$$

It involves generating the sum and difference of two numbers and subtracting the squared difference from the squared sum as shown in (1). The product is then one quarter of the resultant difference—thus the name quarter square. This is similar to the technique suggested by Logan [4] who proposed that digital multiplication be performed by accumulating squares via the Binomial Theorem:

$$AB = \tfrac{1}{2}\{(A+B)^2 - A^2 - B^2\}. \tag{2}$$

Equation (2) requires one more squaring device than (1). Logan proposed an LSI squaring device which would be used to perform the indicated squares in (2). A value of 10 input bits was placed as the upper limit on feasible LSI squaring circuits (1971 Technology). For squares involving more than 10 bits Logan proposed a double precision algorithm using (2) as a base.

For the digital quarter square (DQS) multiply scheme proposed in this correspondence memory sizes of $2^{N+1} \times 2N$ will be required for products of N bit words. Values of N up to 16 may be accommodated with current LSI memories and even larger word length products should be possible with VLSI without resorting to double precision techniques.

Fig. 1 shows the block diagram of a single ROM implementation of a product circuit. For the product of two N-bit words, $2^{2N}2N$-bit words are required to store the complete product table.

In the following discussion, it is assumed that x and y are positive and that $x \geq y$. Fig. 2 shows the basic block diagram of the DQS multiplier. The sum of x and y may result in a value of $N+1$ bits. The difference will always be representable by N bits. ROM 1 contains $(x+y)^2/4$ and ROM 2 contains the two's complement of $(x-y)^2/4$.

Fig. 3 shows the symbolic square of a 4 bit quantity A. Note that the least significant two bits of the squared quantity have possible values of 00 or 01. The division by 4 which is required by the DQS algorithm is accomplished by not storing the two least significant bits of each of the squared quantities. No loss of significance in the resultant product is incurred.

Comparison of the storage requirements of the two techniques demonstrates the significant reduction in storage requirement in the digital quarter square multiplier. The ratio of the single ROM requirement to the DQS bit requirement is

$$\frac{\text{Single ROM Bits}}{\text{DQS Bits}} = \frac{2^{2N} \times 2N}{2^{N+1} \times 2N + 2^N \times (2N-2)} = \frac{2^N}{3 - \dfrac{1}{N}} \approx \frac{2^N}{3} \text{ for large } N. \tag{3}$$

For an 8 bit multiplier the single ROM technique requires 64k of 16 bit words, the DQS technique requires 512 sixteen bit words and 256 fourteen bit words. Certainly the DQS technique makes ROM implemented multipliers for short word length economically feasible. However, there is a slight reduction in the speed of the DQS technique compared to the single ROM caused by the addition required before and after accessing the ROM's.

Two's Complement Multiplication

Fig. 4 shows the block diagram of a DQS configuration which allows multiplication of two's complement numbers. All restrictions of the previous section are removed on the sign and relative

Manuscript received January 10, 1979; revised September 18, 1979.
The author is with the Department of Electrical Engineering, Wichita State University, Wichita, KS 67208.