4

RADC-TR-88-321
Final Technical Report
February 1989

FAULT TOLERANT VLSI DESIGN USING ERROR CORRECTING CODES

Syracuse University

C.R.P. Hartmann, P.K. Lala, A.M. Ali, S. Ganguly, G.S. Visweswaran

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-321 has been reviewed and is approved for publication.

APPROVED: Kevin A Kwiat

KEVIN A. KWIAT
Project Engineer

APPROVED: John J. Bart

JOHN J. BART
Technical Director
Directorate of Reliability & Compatibility

FOR THE COMMANDER: James W. Hyde III

JAMES W. HYDE, III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBRA)Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
N/A

5. MONITORING ORGANIZATION REPORT NUMBER(S)
RADC-TR-88-321

6a. NAME OF PERFORMING ORGANIZATION
Syracuse University

6b. OFFICE SYMBOL
(If applicable)

7a. NAME OF MONITORING ORGANIZATION
Rome Air Development Center (RBRA)

6c. ADDRESS *(City, State, and ZIP Code)*
Department of Computer & Information Sciences
Syracuse NY 13244

7b. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Rome Air Development Center

8b. OFFICE SYMBOL
(If applicable)
RBRA

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F30602-81-C-0169

8c. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
62702F	2338	01	PU

11. TITLE *(Include Security Classification)*
FAULT TOLERANT VLSI DESIGN USING ERROR CORRECTING CODES

12. PERSONAL AUTHOR(S)
C. R. P. Hartmann, P. K. Lala, A. M. Ali, S. Ganguly, G. S. Visweswaran

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM Mar 87 TO Mar88

14. DATE OF REPORT *(Year, Month, Day)*
February 1989

15. PAGE COUNT
72

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	04	
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

Fault Tolerant	Error Correction	Bus Circuits
Self-Checking Circuits	Error Detection	EDAC
On-Chip Fault	Microcircuits	VLSI

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Very Large-Scale Integration (VLSI) provides the opportunity to design fault tolerant, self-checking circuits with on-chip, concurrent error correction. This study determines the applicability of a variety of error-detecting, error-correcting codes (EDAC) in high speed digital data processors and buses. In considering both microcircuit faults and bus faults, some of the codes examined are: Berger, repetition, parity, residue, and Modified Reflected Binary codes. The report describes the improvement in fault tolerance obtained as a result of implementing these EDAC schemes and the associated penalties in circuit area.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
KEVIN A. KWIAT

22b. TELEPHONE *(Include Area Code)*
(315) 330-2047

22c. OFFICE SYMBOL
RADC/RBRA

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

EVALUATION

Very Large Scale Integration (VLSI) provides the opportunity to design fault tolerant microcircuits that have on-chip, concurrent error correction. Long before VLSI, error-correcting codes were used to provide error-free communication over noisy channels. Digital memories used coding techniques to provide correct data in spite of bit errors in stored data. By using techniques applicable to VLSI design and appropriate fault models, this task evaluates the use of error-detecting and correcting (EDAC) codes in high speed digital data processors and buses.

This report determines the applicability of a variety of EDAC codes such as: Berger, repetition, parity, residue, and Modified Reflected Binary (MRB) codes. The applicability of a code is determined by the demonstrated improvement in fault tolerance obtained by a particular coding scheme and the concomitant penalty in chip area or bus width needed to accommodate any redundant circuitry.

The results in this report substantiate the completeness of the evaluation. Although no single fault-tolerant technique using EDAC codes is a complete solution, this study provides the supporting data for making trade-off decisions.

Selecting a "best" technique for fault-tolerance that does not negatively impact performance or throughput requires consideration of many factors. This task has provided information that will allow intelligent consideration of EDAC codes as design options.

Kevin A. Kwiat

KEVIN A. KWIAT
Project Engineer

Table of Contents

List of Figures

Section 1. Introduction

The reliability of a circuit can be enhanced by employing the method of worst-case design, using high quality components, imposing strict quality control procedures during the assembly phase, and using extensive testing. However, such measures can significantly increase the cost. Furthermore, the effectiveness of these techniques is reduced because it is not possible to exhaustively test a complex circuit. Also, a transient fault would not be detected by these techniques. An alternative approach to improve the reliability is to incorporate self-checking facilities in a circuit; this allows the circuit to react "on the fly" to internal failures. For a given set of faults, a self-checking circuit either produces the correct outputs or indicates that the outputs are incorrect.

A wide variety of codes is available for possible use in self-checking circuit design, e.g., parity codes, Hamming codes, m-out-of-n codes, Berger codes, and residue codes. However, different codes have different error-detecting capabilities. In order to choose an error-detecting code for a circuit it is essential to know the effects of the faults under consideration on the outputs of the circuit. In general, a non-code word at the output of a self-checking circuit indicates the presence of a fault in the circuit.

In Section 2 we consider two schemes for the detection of an error caused by a single short in a 32-bit bus. The first scheme utilizes a modified form of the Berger code and is implemented by cascading ROMs. The second scheme is based on the time-multiplexing of information and tests vectors on the bus. As we will show, the test vectors required depend on the physical characteristics of the bus. A scheme for correcting a single bit error is also proposed.

Section 3 deals with the detection of single bit error in basic arithmetic operations, e.g., addition and multiplication. Three separate schemes for the design of self-checking adders have been considered. The overhead in terms of chip area and the fault detection capability of each scheme is evaluated. One of these schemes is based on the MRB (Modified Reflected Binary) code. This code is also used to design a self-checking pipelined multiplier.

Finally, Section 4 considers the application of a well-known code, the residue code, for the detection and correction of errors in fixed point adders.

Section 2. Self-Checking and Fault Tolerant Bus Design

A bus consists of one or more lines which transfer information and electrical power between the various components of a digital system. The concept of a common bus, carrying data from one part of a system to another is so obvious that it is difficult to imagine a complex digital system without one. The overall reliability of a bus-oriented system is heavily dependent upon the reliability of the bus. One way to improve the reliability of a bus is to make it self-checking. Two such schemes are proposed in this section. We assume that the system consists of only fully-complementary CMOS devices and that there are inverter banks at both ends of a bus.

Section 2.1. Nature of Bus Faults

Faults most likely to occur in a bus are those due to a bridging between two or more lines of the bus and those due to a break in one or more lines of the bus. We will restrict our attention to a single break (or open fault) and a single short between any number of lines.

Study of this type of bridging fault in CMOS circuits [Hart, 1987] has shown that these faults can produce WIRED-ANDing or WIRED-ORing effect depending on the values of the lines which are shorted. Moreover, all the shorted lines assume the same logical value; hence these faults produce unidirectional errors. If this were not true we would need at least one $0 \rightarrow 1$ transition and one $1 \rightarrow 0$ transition due to the fault; however, this violates the requirement that all shorted lines assume the same logical value as a result of the fault.

Consider a short between k lines of a bus. Let d denote the difference between the number of ones and zeros in these k-bits, i.e.,

$$d = \text{(number of ones)} - \text{(number of zeros)}.$$

This short can be modeled as shown in Fig. 1 where R_p is the on-resistance of a PMOS transistor, R_N is the on-resistance of an NMOS transistor, and V_S is the resultant voltage at the shorted lines. For $|d| \neq k$

$$\frac{V_S}{V_{DD}} = \frac{1}{1 + \left(\frac{R_P}{R_N}\right)\left(\frac{k-d}{k+d}\right)}$$

$$\text{and } \frac{\delta(V_S/V_{DD})}{\delta(d)} = \frac{\left(\frac{R_P}{R_N}\right)\frac{2k}{(k+d)^2}}{\left(1 + \left(\frac{R_P}{R_N}\right)\left(\frac{k-d}{k+d}\right)\right)^2} > 0.$$

V_S is a monotonically increasing function of d; hence the value of d which decides whether V_S is logic high or low would depend on k, R_P and R_N. Thus the maximum number of unidirectional errors also depend on k, R_P and R_N. Simulation results for values of k up to 8 show that the maximum number of unidirectional errors is $\lceil \frac{k}{2} + 1 \rceil$. Generally, for values of d which are not close to zero and for typical values of $\left(\frac{R_P}{R_N}\right)$, the zeros of the shorted lines get converted to ones if the number of ones is greater than the number of zeros, and vice versa. This implies that, in this situation, the short can be modeled as a majority function.

Section 2.2. Design of Self-Checking Buses

For any bus to be self-checking it should at least be able to detect unidirectional errors. To design an efficient scheme to make the bus self-checking we need to investigate the effect of unidirectional errors. A unidirectional error always changes the number of ones (or zeros) in the information present on the bus lines.

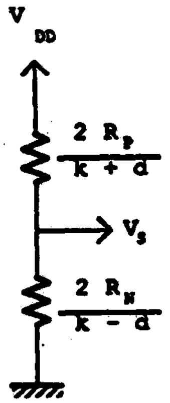


Fig.1 Effective Model for a Short in a Bus.

Consider a k-bit bus. We can define a partition π on the set of the 2^k possible data words. This partition has $k+1$ blocks where data words containing the same number of ones (or zeros) belong to the same block of π. This is illustrated in Table 1 where block B_i contains all data words with i ones. Since a unidirectional error always changes the count of ones, it can never erroneously convert a data word to another belonging to the same block of π. Thus, to detect unidirectional errors, we can append the same check bits to data words which have the same number of ones, and from now on we only need to consider data words shown in Table 1.

Block	Data
B_0	000...............000
B_1	000...............001
B_2	000...............011
$\vdots$	
$\vdots$	
$\vdots$	
B_k	111...............111

Table 1.

We now formalize the above concepts. Let X and Y be two binary k-tuples.

Definition 1. The Hamming distance between X and Y, $D(X,Y)$, is defined as the number of bit positions in which they differ.

Definition 2. The crossover from X to Y, $N(X,Y)$, is defined as the number of bit positions in which X is 1 and Y is 0.

Example:

$$\text{Let } X = 10011010 \text{ and}$$

$$Y = 00110111.$$

Then $D(X,Y) = 5$, $N(X,Y) = 2$, and $N(Y,X) = 3$.

Note that $D(X,Y) = N(X,Y) + N(Y,X)$. The following theorem gives the conditions required for a code to detect t unidirectional errors.

Theorem 1 [Bose,1985]. A code C is capable of detecting up to t unidirectional errors iff for all distinct codewords X and Y belonging to C at least one of the following conditions is satisfied:

(i) $D(X,Y) \geq t+1$;

(ii) $N(X,Y) \geq 1$ and $N(Y,X) \geq 1$. ■

Note that all binary tuples of the same block of π satisfy condition (ii) of Theorem 1.

Let us now obtain a lower bound for the number of check bits required by a separable code to detect t-unidirectional errors. Separable codes are those in which no decoding is required to extract the information bits from the codeword. Using condition (i) of Theorem 1 we can assign the same check bits to two blocks B_i and B_j of π (without loss of generality assume $i < j$) whenever $j-i \geq t+1$. Thus we can assign the same check bits to blocks B_i and B_{i+t+1} for $i = 0,1,2,\ldots(k-t-1)$. As a consequence, only blocks B_0 through B_t need to be assigned distinct check bits. Therefore, the lower bound for the number of check bits required by a separable code to detect t unidirectional errors is $\lceil \log_2(t+1) \rceil$. For $t = k$ the lower bound for the number of check bits is $\lceil \log_2(k+1) \rceil$.

Berger has proposed a technique to design a separable code, using $\lceil \log_2(k+1) \rceil$ check bits to encode k information bits, capable of detecting all unidirectional errors [Berg, 1961]. Thus Berger codes are optimum.

We are interested in designing a separable code for a bus with 32 information bits. As discussed earlier, we expect to have not more than $t = \lceil \frac{32}{2} + 1 \rceil = 17$ unidirectional errors. Thus the number of check bits r has to be at least

$\lceil \log_2(17+1) \rceil = 5$. The number of lines in the redundant bus is $n = k + r$. We now propose a scheme that will detect all unidirectional errors caused by a single short in a 37-bit bus consisting of 32 information bits and 5 check bits. Since in a bus with 37 lines we expect to have at most $\lceil \frac{37}{2} + 1 \rceil = 20$ unidirectional errors, this scheme uses the minimum number of redundant bits required by a separable code.

Our scheme is a modification of that proposed by Berger [Berg, 1961]. In the latter, the check bits are formed by taking the complement of the binary representation of the count of the ones in the information bits. For $k = 32$ the Berger code requires 6 check bits. We notice that all information vectors except the all-1 vector have the same MSB (most significant bit) in the check bit part of the code. We now delete this bit. We change the check bits for the all-1 information vector and for the information vectors with 31 ones to 00000 and 10000, respectively. These correspond to the last two rows of Table 2.

32 information bits	5 check bits
0 0	1 1 1 1 1
0 01	1 1 1 1 0
0 011	1 1 1 0 1
$\overbrace{0\ \ldots\ldots\ldots 0}^{17}\ \overbrace{1\ldots\ldots\ldots 1}^{15}$	1 0 0 0 0
001.................... 1	0 0 0 0 1
01 1	1 0 0 0 0
11 1	0 0 0 0 0

Table 2.

Notice that information vectors which have 31 ones and 15 ones have the same check bits. Moreover, only the code words corresponding to these information vectors do not satisfy condition (ii) of Theorem 1. These code words, denoted by X and Y, are shown below where the check bits are given in parentheses:

$$X: \ 0000\ 0000\ 0000\ 0000\ 0111\ 1111\ 1111\ 1111\ (10000)$$

$$Y: \ 0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ (10000)$$

We must now show that a single short cannot change X to Y and vice versa. Note that X has 21 zeros and 16 ones and Y has 5 zeros and 32 ones. To change X to Y, 16 zeros have to be shorted with a certain number of ones (say m) to result in a total of 32 ones. Notice that $m \leq 16$. The resultant voltage at the shorted lines, V_S, caused by shorting 16 zeros with m ones is

$$V_S = \frac{V_{DD}}{1 + \left(\frac{R_P}{R_N}\right)\left(\frac{16}{m}\right)} \quad ,$$

where R_P and R_N are the on-resistances of a PMOS and an NMOS transistor, respectively. Note that V_S is an increasing function of m. Thus to obtain the maximum value of V_S we choose $m = 16$; however, since R_P is typically 2 to $2 \cdot 5$ times R_N, the resultant V_S is still logic low. Thus a single short cannot change X to Y. By a similar analysis it can be shown that a single short in Y cannot change 16 ones to zeros, and hence Y can never change to X.

Consider a code word Z from the coding scheme shown in Table 2. Let Z be different from X or Y defined earlier. No unidirectional error, irrespective of its length, can change Z to another valid code word W belonging to Table 2 since $N(Z,W) \geq 1$ and $N(W,Z) \geq 1$. Thus the proposed coding scheme will detect the errors caused by any single short in the 37 $(= 32 + 5)$ bit bus. Fig. 2 shows a block diagram for the proposed implementation. In this figure $C(X)$ denotes the check bits corresponding to the information vector X. Notice that ROM A and ROM B give complementary output values for the same input. This enables us to use a totally self-checking two-rail checker [Lala, 1985] to compare the ROM outputs. There are many possible ways, including the use of PLAs, to implement the two-rail checker [Lala, 1985]. Fig. 3 shows one possible implementation of the two-rail checker.

According to our earlier analysis, any error produced by a single short in the bus will be detected; moreover, since a single error is a trivial case of a unidirectional error, any fault that produces a single error in the bus, e.g., a single line stuck open, will be detected. Moreover, the two-rail checker is totally self-checking with respect to all single stuck-at faults. Let us now consider faults in the ROM. These can be classified into two categories:

(i) Faults for which there exists an input X that causes ROM A (or B) to output $C(Y)$ where $C(Y) \neq C(X)$.

(ii) Faults for which there does not exist any input X that causes ROM A (or B) to output $C(Y)$ where $C(Y) \neq C(X)$.

For the former category, in the absence of any other error, the two-rail checker will detect the error caused by the ROM. Note that the latter category constitutes an undetectable fault; however, even in the presence of such an undetectable fault any fault which causes a unidirectional error in the bus is still detectable. The above scheme can be modified to one in which the information and check bits are transmitted on two physically separate buses. The block diagram for this is shown in Fig. 4; however, this does not detect all errors caused by separate single shorts in both buses.

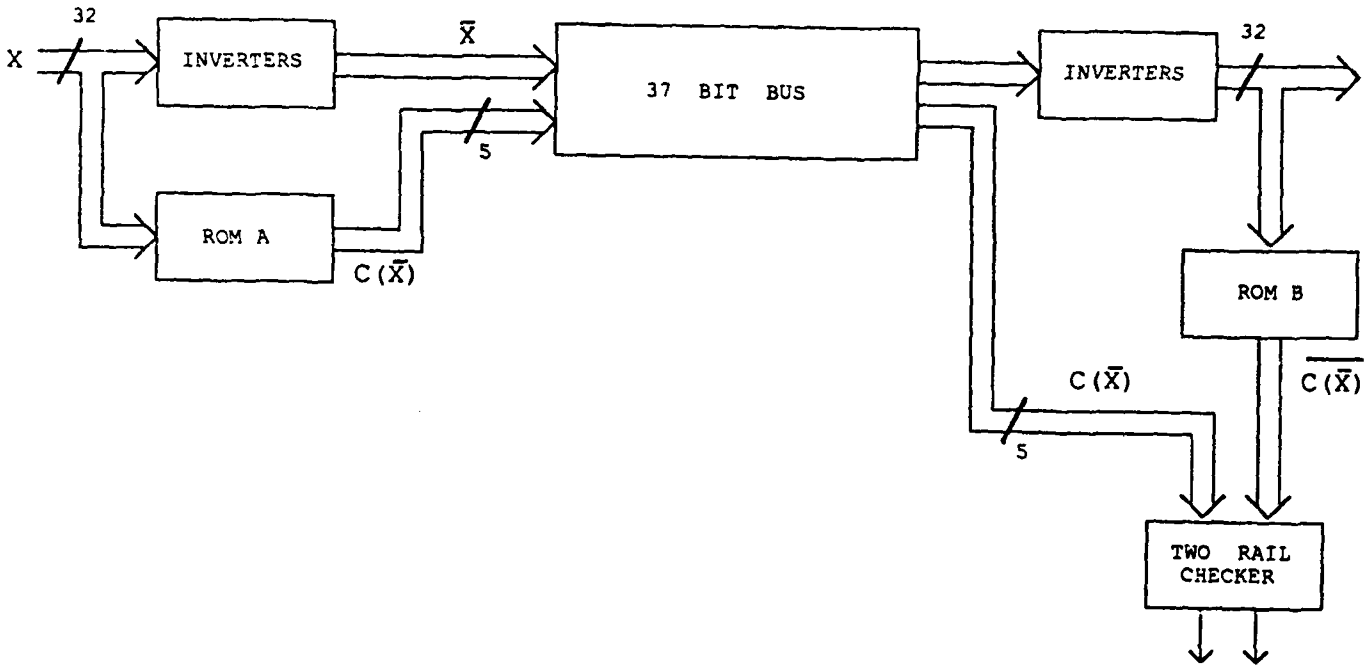
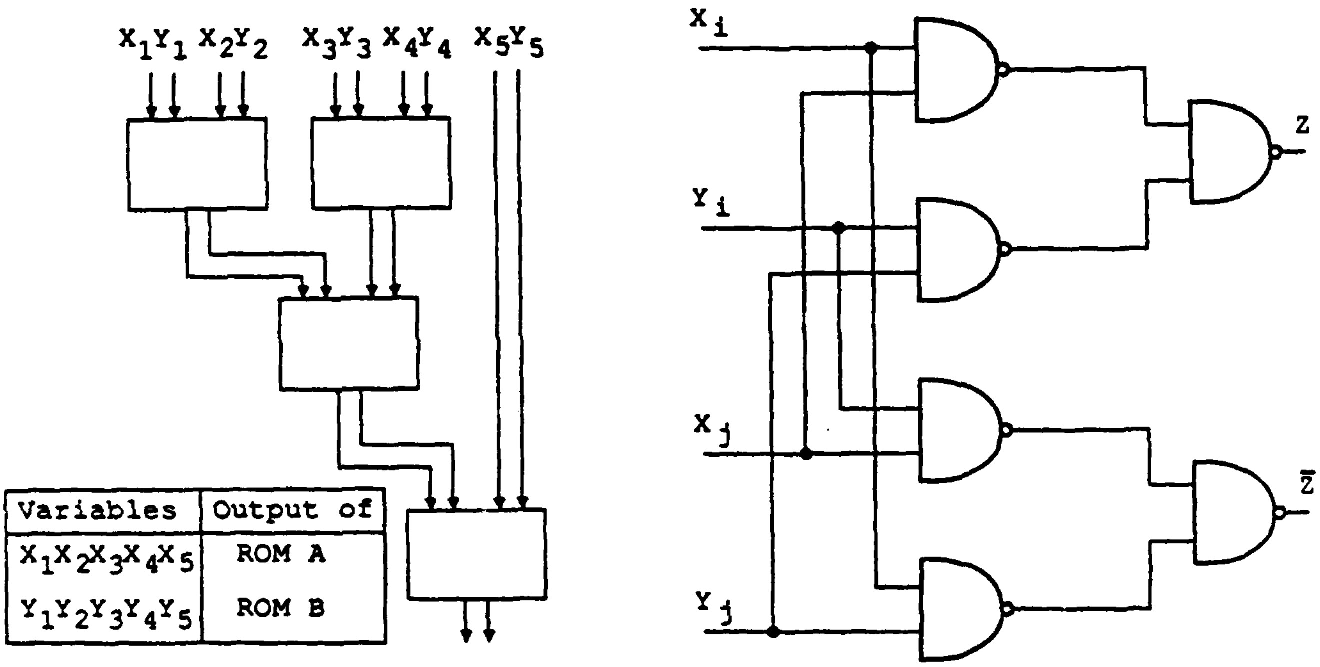


Fig.2 Block Diagram For Proposed Implementation.



Variables	Output of
$X_1X_2X_3X_4X_5$	ROM A
$Y_1Y_2Y_3Y_4Y_5$	ROM B

(a) Design of a 5 input pair two rail checker using 2 input pair checker blocks.

(b) Gate level representation of a two input pair two rail checker.

Fig.3 A Two Rail Checker Implementation.

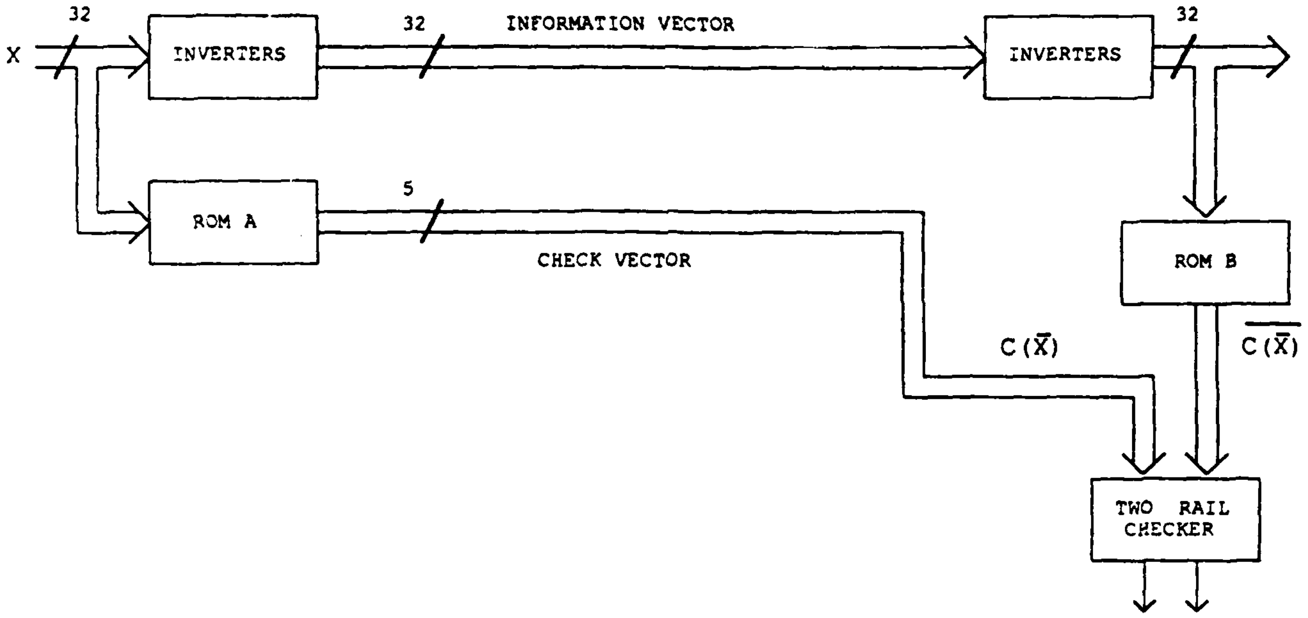


Fig.4 Modified Scheme with separate Buses for Information and Check bits.

Another possible strategy is to use the same 32-bit bus to transmit the information and test vectors. These test vectors are inputs that detect single shorts in the bus. The feasibility of this time-multiplexed scheme would depend on whether the bus is physically "flat" or not. We define a "flat" bus as one in which a short between any two lines always shorts these lines with all the lines physically located between them.

Flat Bus

For a flat bus a single test vector consisting of alternating 0's and 1's (i.e., $0101\cdots 01$) is sufficient to detect all errors due to single shorts in the bus. This is because a single short would cause all the shorted lines to transmit all zeros or all ones and hence the error can be detected.

Fig. 5 shows a possible implementation of this scheme. The control input S of the multiplexer selects the information vector when $S = 0$ and the test vector when $S = 1$. The input L is used to ensure that only the test vectors are latched into the flip-flops. Fig. 6 shows the timing diagram for the operation of one cycle of this scheme. For the scheme to be feasible it must satisfy the constraint

$$\frac{1}{F} > t_m + t_b + t_{FF} + t_c.$$

Using Schottky TTL components we estimate that $t_m = 5ns$, $t_{FF} = 5ns$ and $t_c = 20ns$. Therefore, systems with a clock rate of 25 MHZ and buses with delay of $8ns$ or less would operate correctly. We note that this scheme would not detect certain stuck-at faults. If time constraints permit, we can transmit the test vector and its complement. This will enable us to increase the error-detecting capability to include single stuck-at faults in the lines of the bus and at the output of the D flip flops.

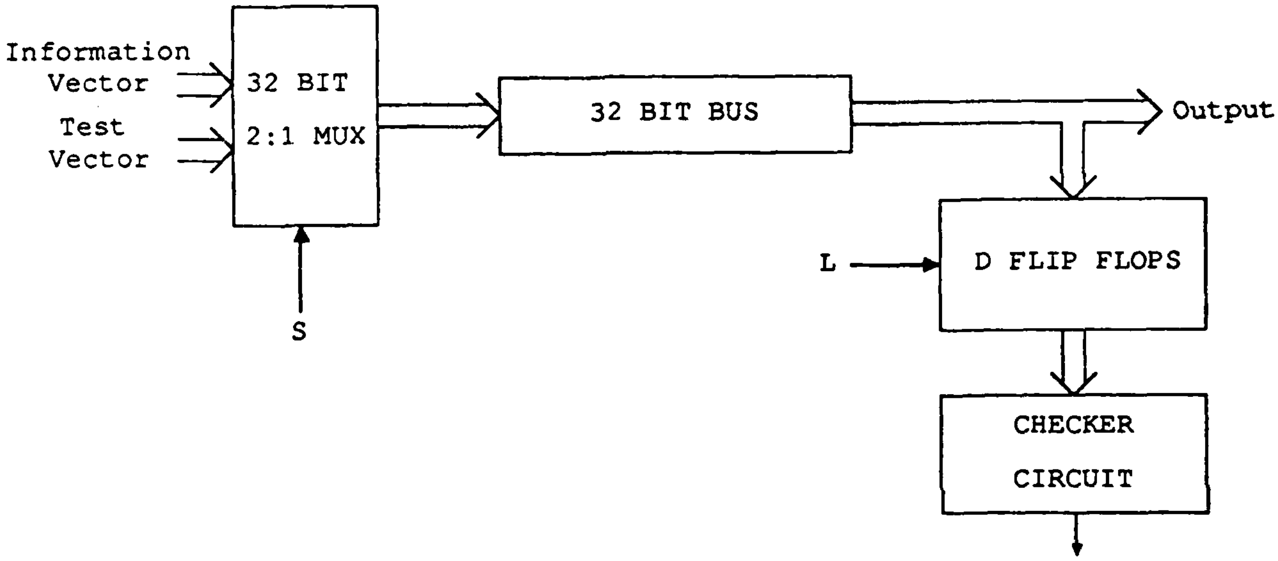
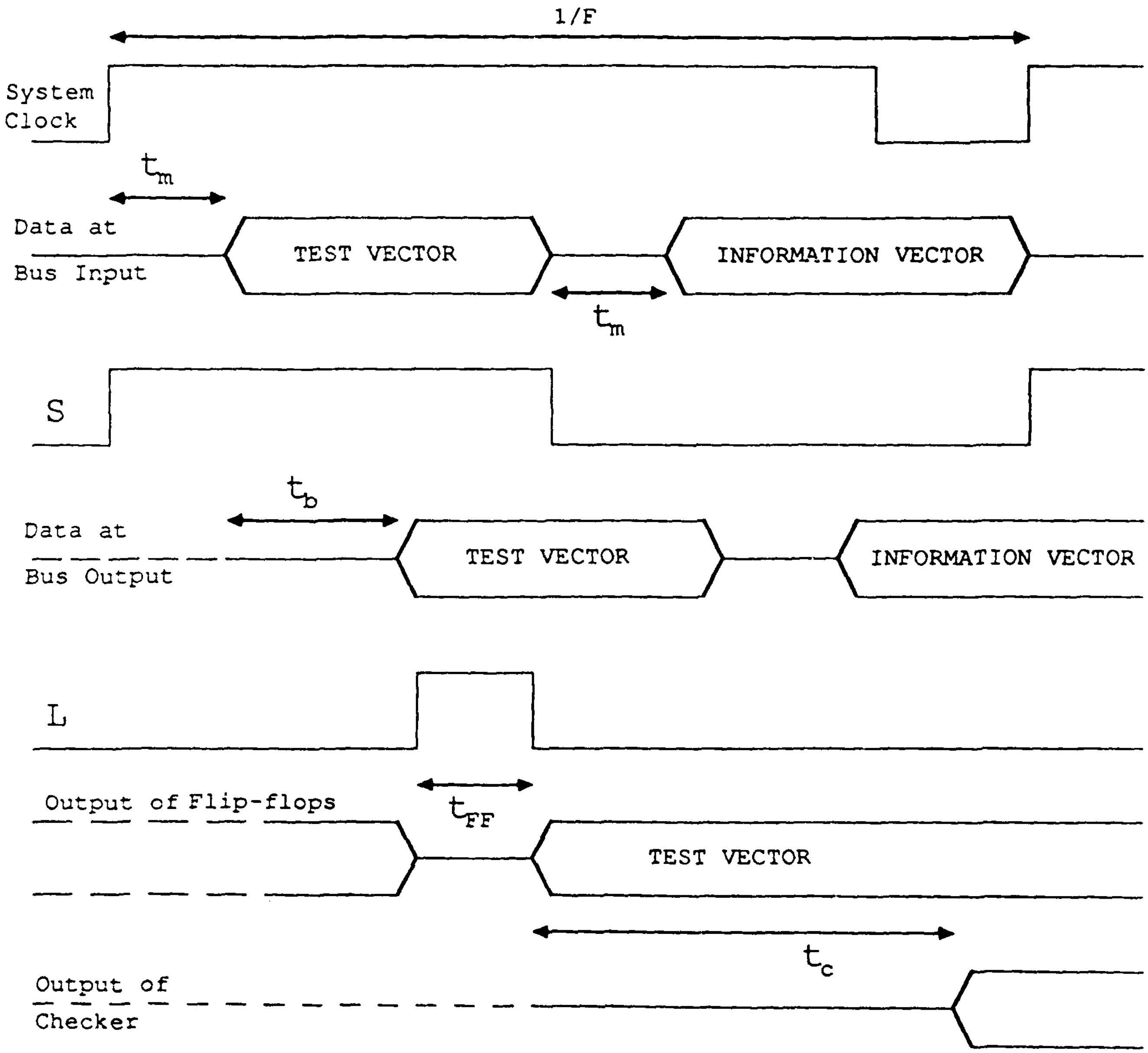


Fig.5 A Time Multiplexed Error Detecting Scheme for a Flat Bus.



KEY:

t_m :Multiplexer Delay

t_b :Bus Delay

t_{FF}:Flip-flop Delay

t_c :Checker Circuit Delay

F :Frequency of System Clock

Fig.6 Timing Diagram for the Scheme depicted in Fig.5

Non-Flat Bus

Since the bus is not flat, any two lines may be shorted together. To be able to detect a single short between any two lines we must apply a test vector which transmits different values on these two lines of the bus. Thus, to detect all possible single shorts between any two lines we must transmit a set of test vectors such that for any given pair of lines we have at least one test vector which transmits different values on this pair of lines. This test set will also detect all possible single shorts because for a short across any given t lines there exists at least one test vector in this set which does not transmit the same value on these t lines.

Theorem 2. Let S be a set of vectors of length n. The members of S are such that for all possible integral values of i and j, $i \neq j$ and $1 \leq i, j \leq n$ we can find at least one vector in S which differs in its ith and jth component. Under these conditions the cardinality of $S \geq \lceil \log_2 n \rceil$.

Proof. Consider the two-dimensional array where each row of the array consists of one element of S. It is necessary that every two columns of the array be distinct so that it satisfies the required condition. The minimum number of rows required to get n distinct columns is $\lceil \log_2 n \rceil$. ■

For $n = 32$, consider a two-dimensional array whose columns consist of all the 32 distinct 5-tuples. The rows of this array form a set of test vectors with minimum cardinality which satisfy the condition of Theorem 2.

If we want to implement a time-multiplexed scheme for a 32-bit non-flat bus, we have to transmit 5 test vectors for every information vector. For a system clock of 25 MHZ it is not possible to meet the timing constraints involved in transmitting six vectors in one clock cycle. However, we could transmit a different test vector in five consecutive clock cycles.

Section 2.3. Design of SEC-DED Buses

We are now interested in designing a scheme for single error-correcting (SEC) and double error-detecting (DED) buses. We assume that the bus is the only source of errors. Hsiao has proposed a class of SEC-DED codes [Hsia, 1970]. In order to design a SEC-DED scheme for a 32-bit information bus we use the (39,32) SEC-DED code constructed by Hsiao. The parity check matrix **H** for this is shown in Table 3.

$$\mathbf{H} = \begin{bmatrix} 111111110000001000010010100000111000000 \\ 000010011111111100100100100001000100000 \\ 000100000001000011111111001101100010000 \\ 001000100010010110000000111111110001000 \\ 011001010100100100001111011010000000100 \\ 100001101000111011111000000010000000010 \\ 110110001111000001000001010100010000001 \end{bmatrix}$$

Table 3. Parity Check Matrix

Fig. 7 shows a possible implementation of the SEC-DED scheme. The Check Generator generates the seven check bits from the information bits to be transmitted on the bus in accordance with the parity check matrix. These seven bits are transmitted in a separate bus. The Syndrome Generator first generates the check bits corresponding to the information bits D0-D31 received at the output of the bus. It then compares these check bits with the received check bits transmitted by the 7-bit bus and generates the 7 syndrome bits $S1$ through $S7$. The structure of one of the seven cells that constitute the Check/Syndrome Generator is shown in Fig. 8. The Check Generator generates the check bit $C1$. It does not contain gate G. The Syndrome Generator consists of the Check Generator and gate G. It generates the Syndrome bit $S1$ by comparing the received check bit $\hat{C}1$ with the check bit $C1$

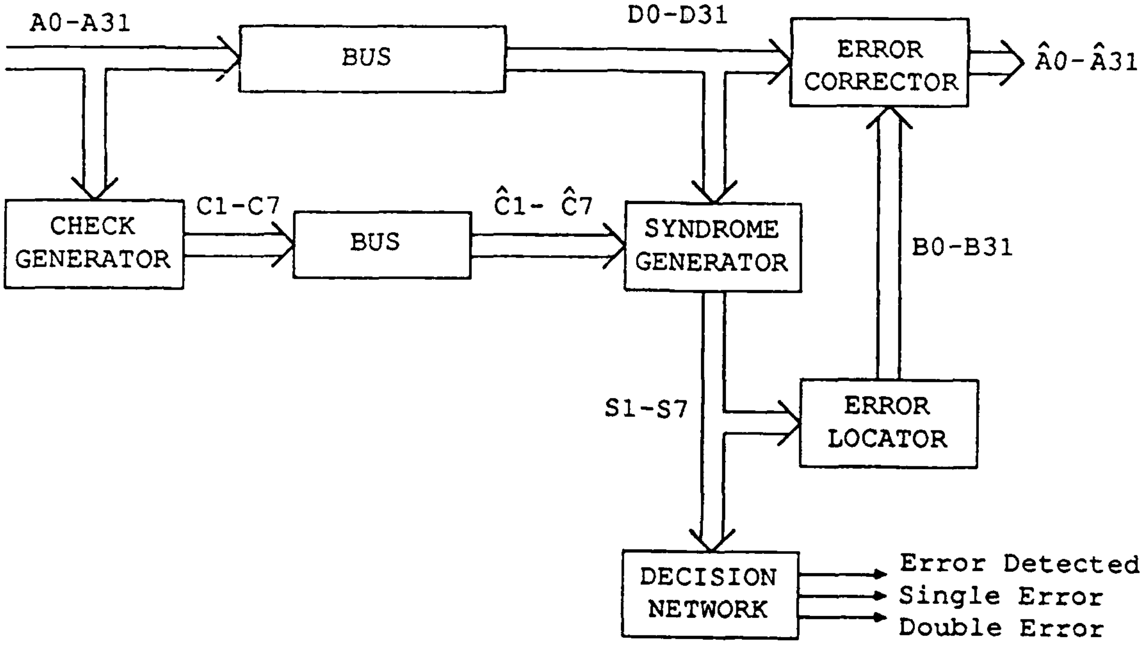


Fig.7 SEC-DED Scheme.

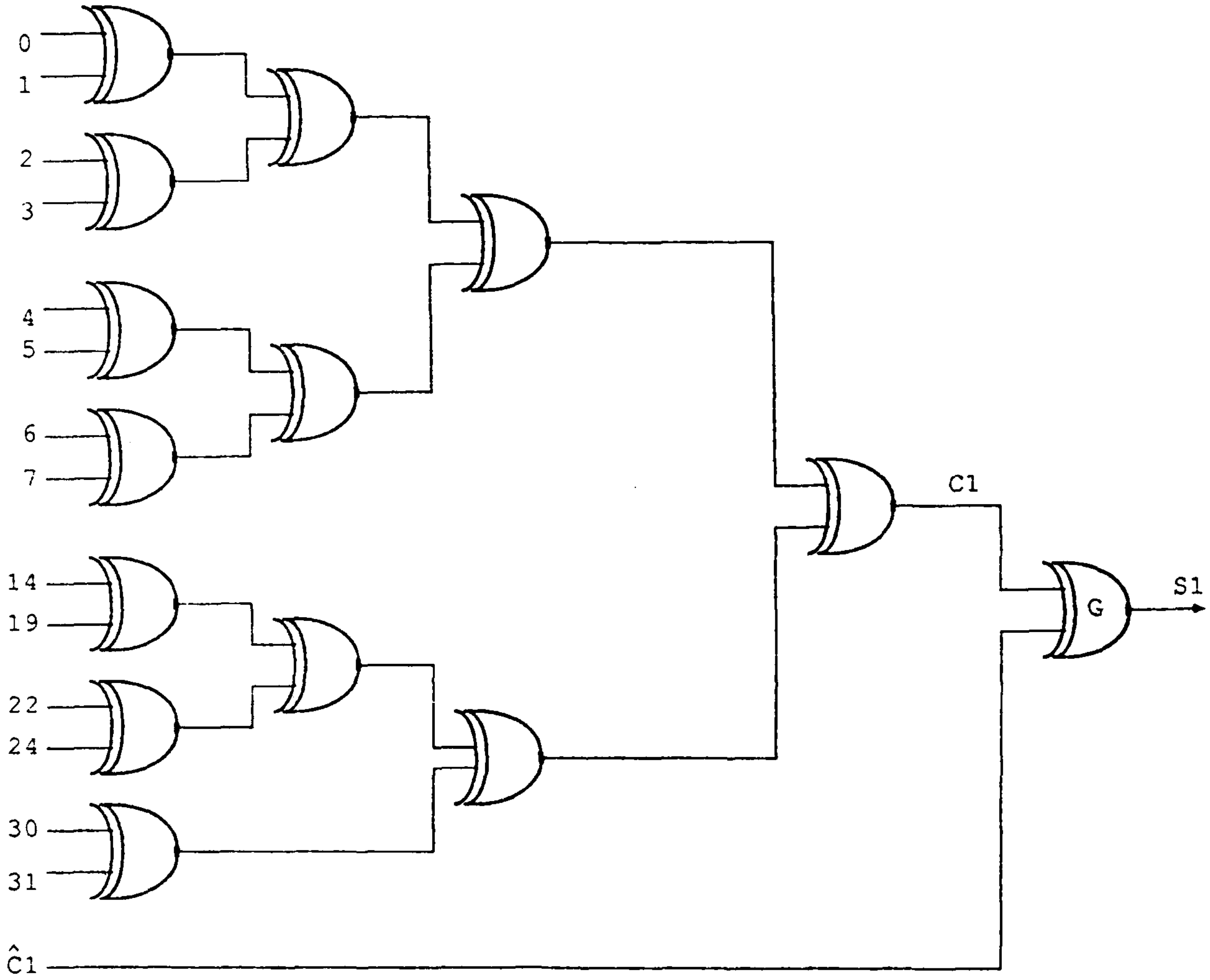


Fig.8 Check/Syndrome Generator.

generated from the received information bits $D0 - D31$. It can be easily shown that if there is a single error in the information bus, then exactly three syndrome bits will be 1. Moreover, if there is a single error in the transmitted check bits, then only one syndrome bit will be 1. It can also be shown that an even number of errors results in an even number of ones in the syndrome bits. Based on these properties, the Error Locator can be implemented by using 32 three-input AND gates as shown in Fig. 9. If there is a single error in bit i, Di, of the received information vector, then the error locator would cause Bi to be 1 and all other outputs to be 0. Accordingly, the Error Corrector consists of 32 two-input EX-OR gates as shown in Fig. 9. The purpose of the Decision Network shown in Fig. 7 is to detect the presence of correctable errors. It examines $S1$ through $S7$ and indicates whether $\hat{A}0$ through $\hat{A}31$ should be accepted as correct information. If it flags a single error or no error detected, then $\hat{A}0$ through $\hat{A}31$ is equal to $A0$ through $A31$ given that, at most, one error has occurred. If it flags double error, then $\hat{A}0$ through $\hat{A}31$ should not be accepted since an even number of errors has occurred. An implementation of the Decision Network is given in Fig. 10.

Section 2.4. Conclusion

In this section two schemes have been proposed to detect errors caused by a single short in a 32-bit bus. The first scheme is based on a modified form of the Berger code. The check bits are generated from the 32 information bits by using a number of ROMs. The proposed scheme requires a tree of ROMs because a 32-bit address ROM is not commercially available. While cascading the ROMs in a tree structure care must be taken to ensure that the timing constraints of the system are met.

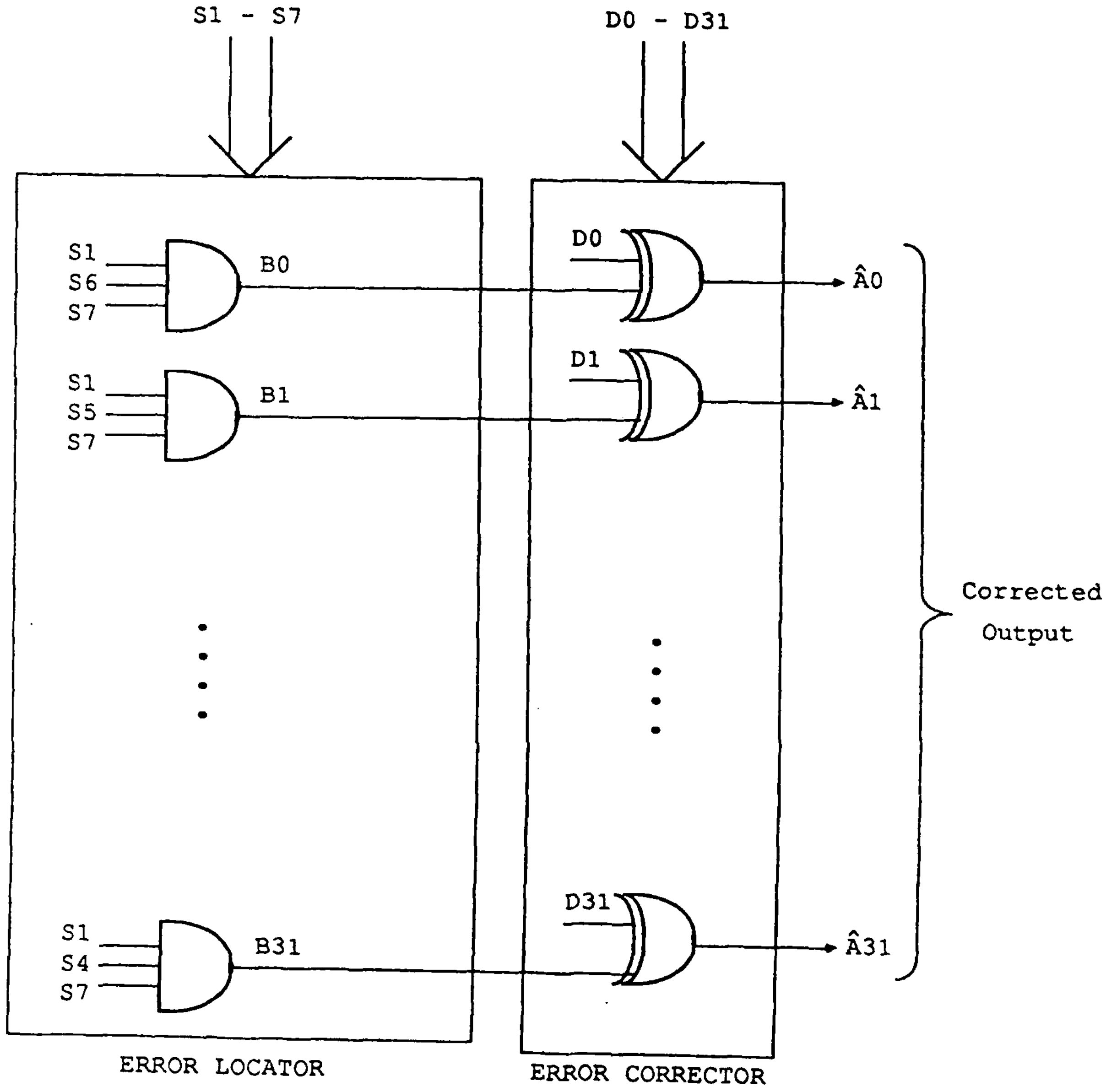


Fig.9 Correction Network.

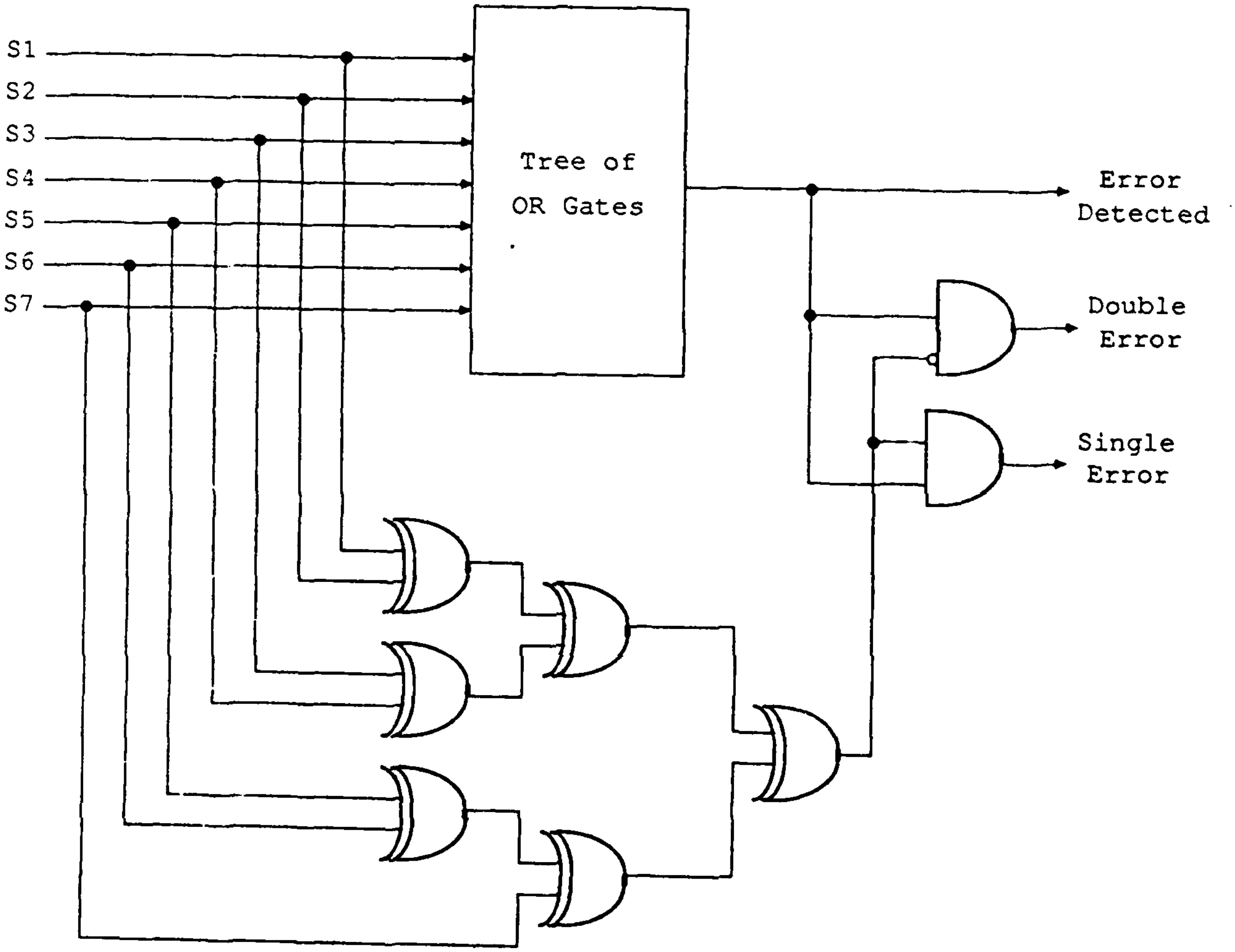


Fig.10 Decision Network.

The other scheme considered for detecting a single short in a 32-bit bus is to time-multiplex both the information bits and the test vectors on the bus. If the bus is "flat," then the test vector basically consists of alternate 0's and 1's. For a non-flat bus, to detect all possible single shorts between any two lines we must transmit a set of test vectors such that for any given pair of lines we have at least one test vector which transmits different values on this pair of lines. For an n-bit bus we need at least $\lceil \log_2 n \rceil$ test vectors and it is therefore impractical to transmit these vectors in one clock cycle.

Finally, a scheme for correcting a single-bit error and simultaneously detecting a double-bit error in a 32-bit bus is also proposed.

Section 3. Design of Self-Checking Arithmetic Circuits

In this section we study several schemes for the design of self-checking adders. Each scheme was evaluated by studying its fault coverage and comparing its area requirements to those of a non-redundant adder. VLSI layouts were generated and simulations were performed to verify the correctness of design for all the schemes. The adder schemes studied were:

(*i*) Duplicated Adder;

(*ii*) Parity Prediction Adder [Prad, 1986];

(*iii*) MRB (Modified Reflected Binary) Code Adder [Luca, 1959].

Furthermore, we also discuss how a self-checking multiplier can be designed using the MRB adder cell.

A fully complementary MOS non-redundant adder cell has been used for comparison of area requirements of the various schemes. Also, the duplication and parity prediction adder cells were built using this cell. The logic equations for the non-redundant adder cell are

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + c_i(a_i + b_i).$$

Where a_i and b_i are the input bits to be added, c_i is the input carry from the previous stage, and s_i and c_{i+1} are the sum and carry outputs, respectively. The circuit diagrams and the layout used to design the non-redundant adder cell are given in Figs. 11 and 12, respectively.

Section 3.1. Duplicated Adder

Duplication is a well known method used for purposes of fault detection. The output of a circuit and its duplicate copy are compared to detect the presence of errors. In the case of an adder, the s_i (sum) output of each cell and its duplicate

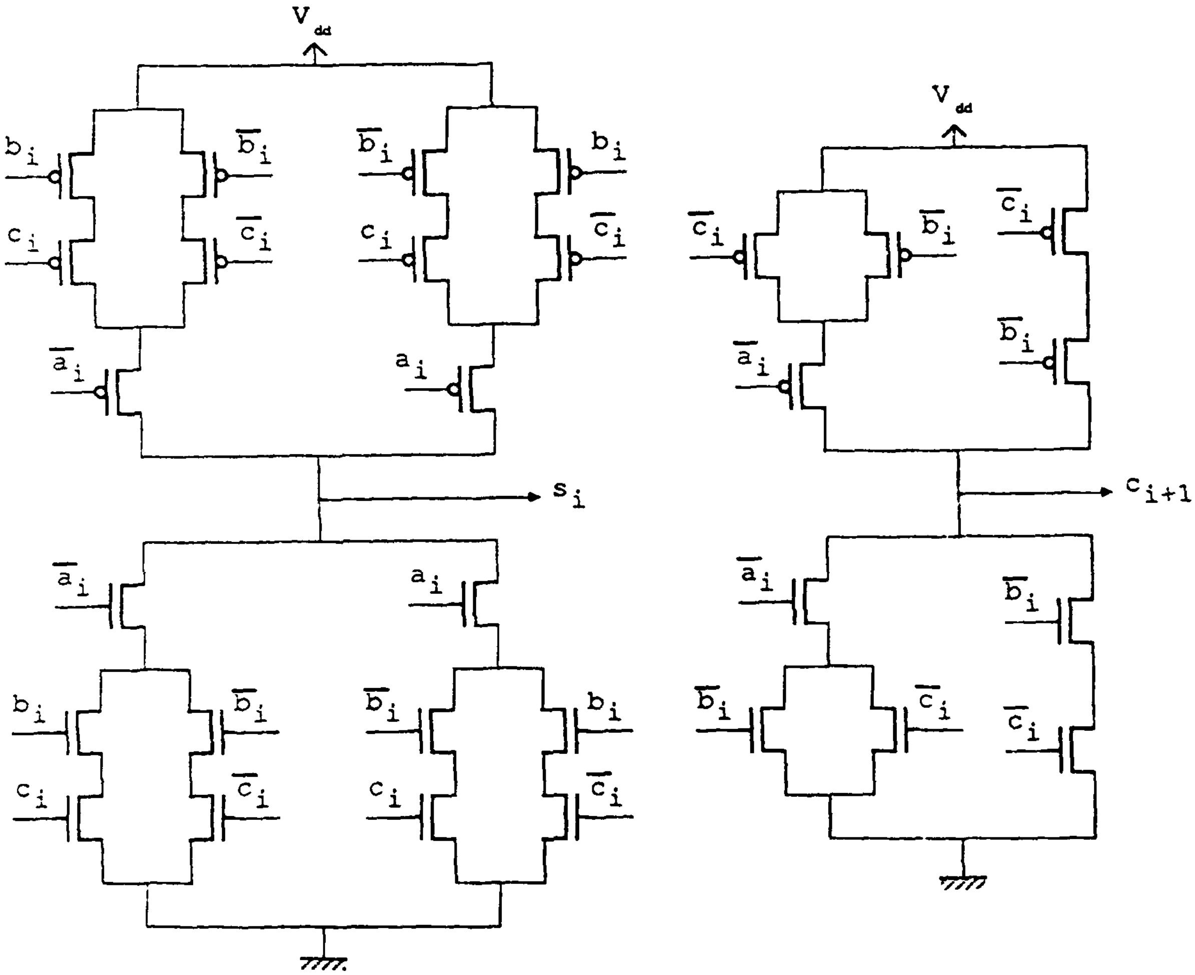


Fig.11 Circuit Diagrams for the Nonredundant Adder Cell.

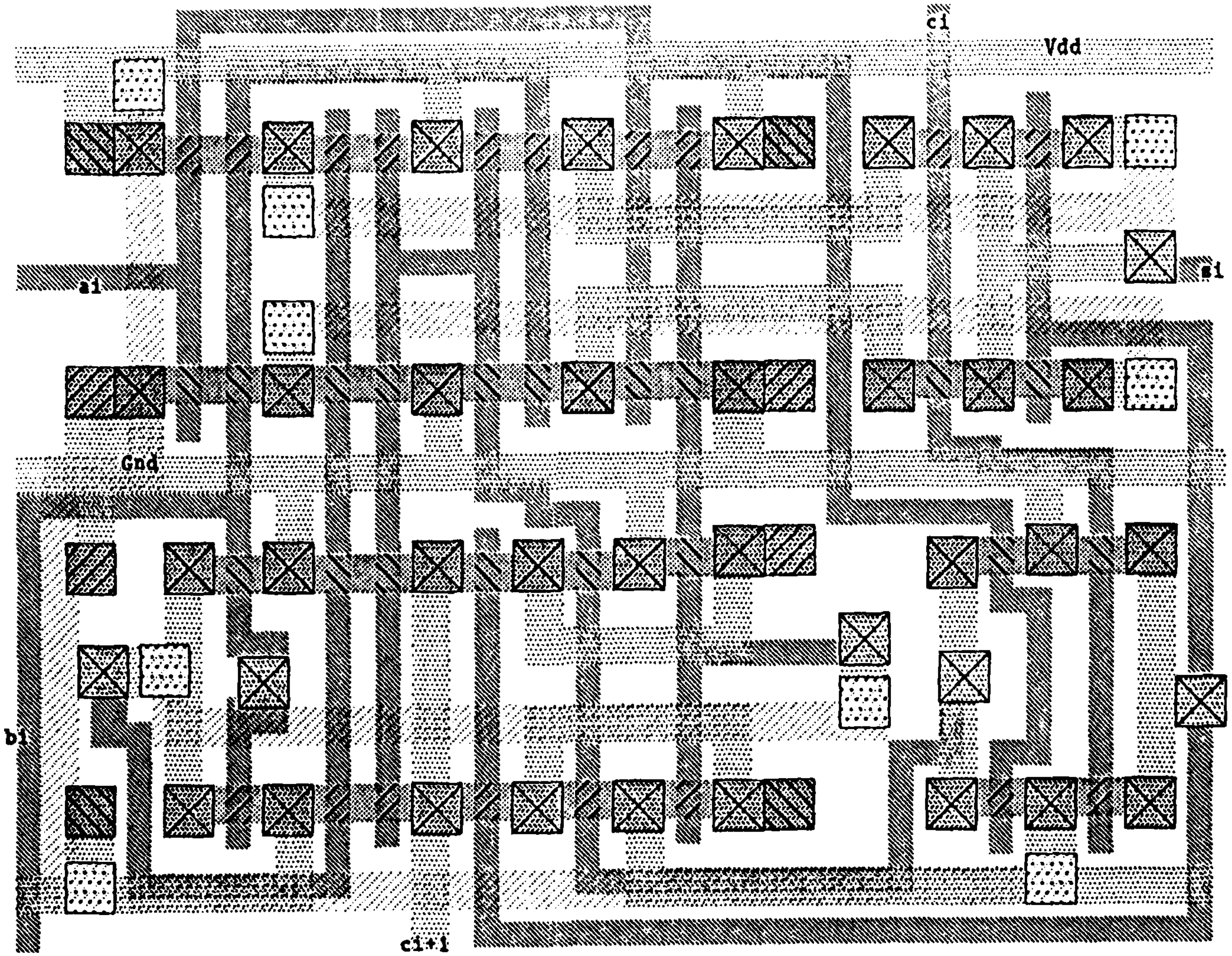


Fig. 12 Layout of Nonredundant Adder Cell

version are compared using an EX-OR gate. A tree of OR gates can now be used to propagate an error signal indicated by one or more EX-OR gates. This method would not detect several single stuck-at faults, e.g., any stuck-at-0 fault at the output of an EX-OR gate. In order to overcome this deficiency, we propose a duplication scheme which uses a totally self-checking two-rail checker to detect errors in the adder. An implementation of our scheme for an n-bit adder is shown in Fig. 13. Note that the inputs and outputs of the duplicate version of each cell are complements of those of the original cell. This allows us to use a two-rail checker. All single stuck-at faults except those at the primary inputs (which cannot be detected by any scheme) can be detected by this scheme. The layout for a composite cell, consisting of a non-redundant adder cell, its complementary version, and a two-rail checker cell, is shown in Fig. 14.

Section 3.2. Parity Prediction Adder

In this section we study the parity prediction adder proposed by Tohma [Prad, 1986]. This scheme requires the parity bits for the numbers to be added. We assume that these parity bits are already available. Let

$$\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0, a_p) \text{ and}$$

$$\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0, b_p)$$

be two n-bit numbers along with their parity bits a_p and b_p. The result of adding the n-bit numbers is

$$\mathbf{s} = (c_n, s_{n-1}, \ldots, s_0)$$

where c_n is the carry-out from the last bit position. The parity bit for $\mathbf{s}$ is s_p where

$$s_p = c_n \oplus s_{n-1} \oplus s_{n-2} \oplus \cdots \oplus s_0. \qquad (1)$$

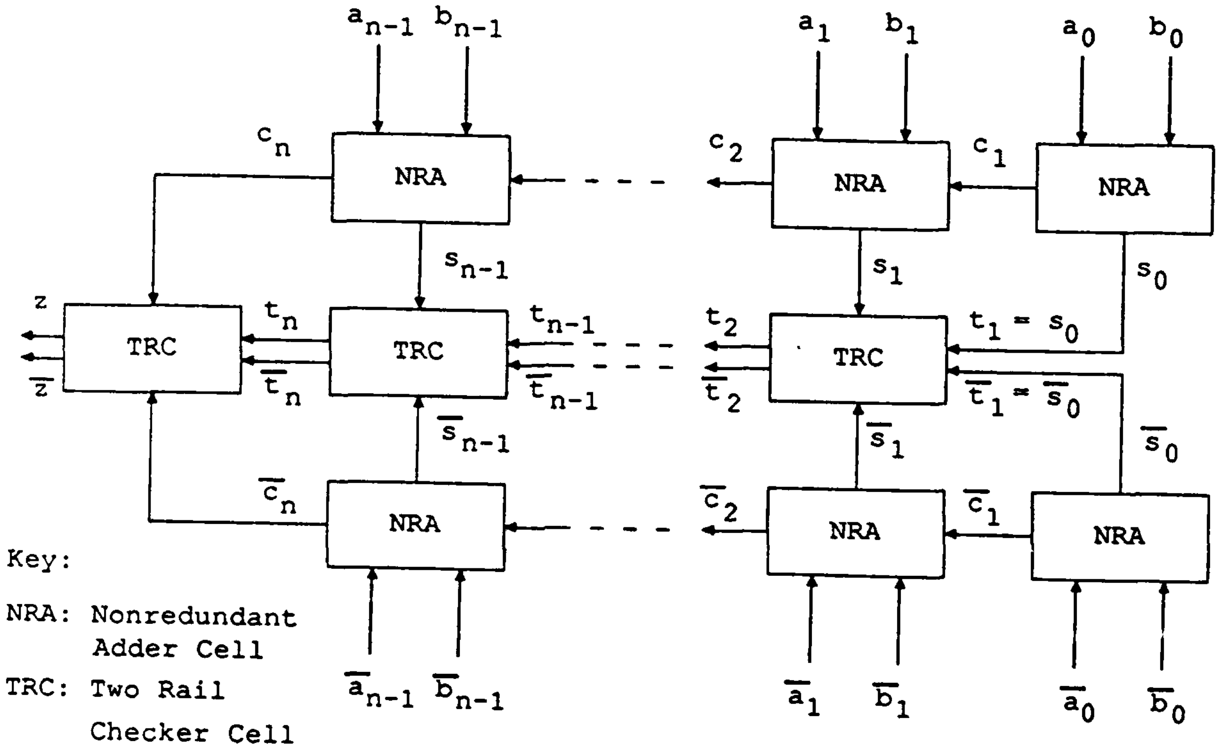


Fig.13 Duplicated Adder Scheme.

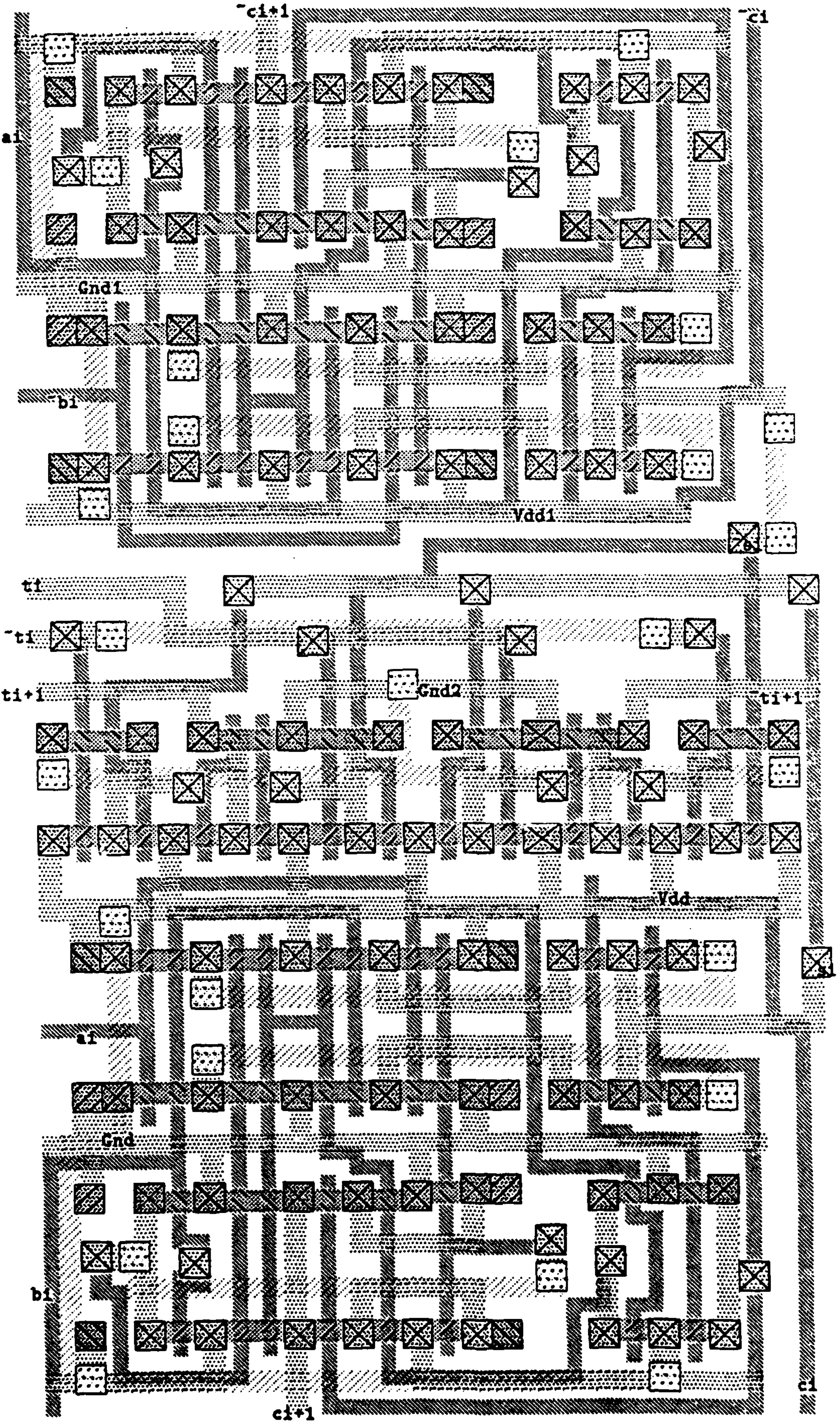


Fig. 14 Layout of Duplicated Adder Cell

Note that since $s_i = a_i \oplus b_i \oplus c_i$ we can rewrite s_p as

$$\begin{aligned} s_p &= (a_{n-1} \oplus a_{n-2} \oplus \cdots \oplus a_0) \oplus (b_{n-1} \oplus b_{n-2} \oplus \cdots \oplus b_0) \oplus \\ &(c_n \oplus c_{n-1} \oplus \cdots \oplus c_1) \qquad (2) \\ &= a_p \oplus b_p \oplus (c_n \oplus c_{n-1} \oplus \cdots \oplus c_1). \end{aligned}$$

In this scheme s_p is generated in two different ways, using Equations 1 and 2. These two values of s_p are compared to detect the presence of errors in the adder; however, if the same c_i is used in generating both values of s_p, then errors in c_i cannot be detected. Tohma suggested the use of duplicate circuits to generate two independent values of c_i. One is used for computing s_p using Equation 1, and the other using Equation 2. An implementation for this scheme is shown in Fig. 15. We have used the circuit diagrams shown in Fig. 11 to generate each adder cell of this scheme.

As shown by Tohma, this scheme detects all single stuck-at faults. Note that during normal operation each EX-OR gate receives all the four possible input patterns and, hence, is exhaustively checked. Thus, this scheme is self-checking with respect to all possible single stuck-at faults. Fig. 16 shows a composite cell for this scheme which consists of the adder cell and the two corresponding EX-OR gates.

Section 3.3. MRB Code Adder

This scheme is based on the Modified Reflected Binary (MRB) Code proposed by Lucal [Luca, 1959]. MRB Codes are simply Gray Codes with a parity bit appended in the least significant position. If

$$\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)$$

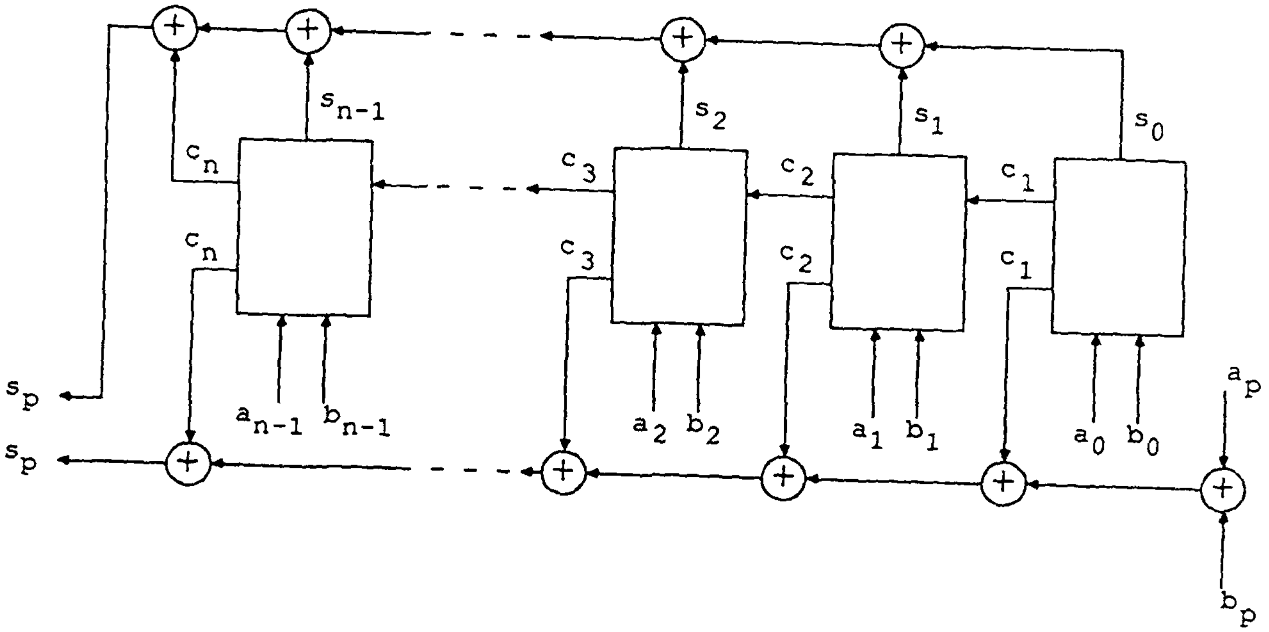


Fig.15 Parity Prediction Adder.

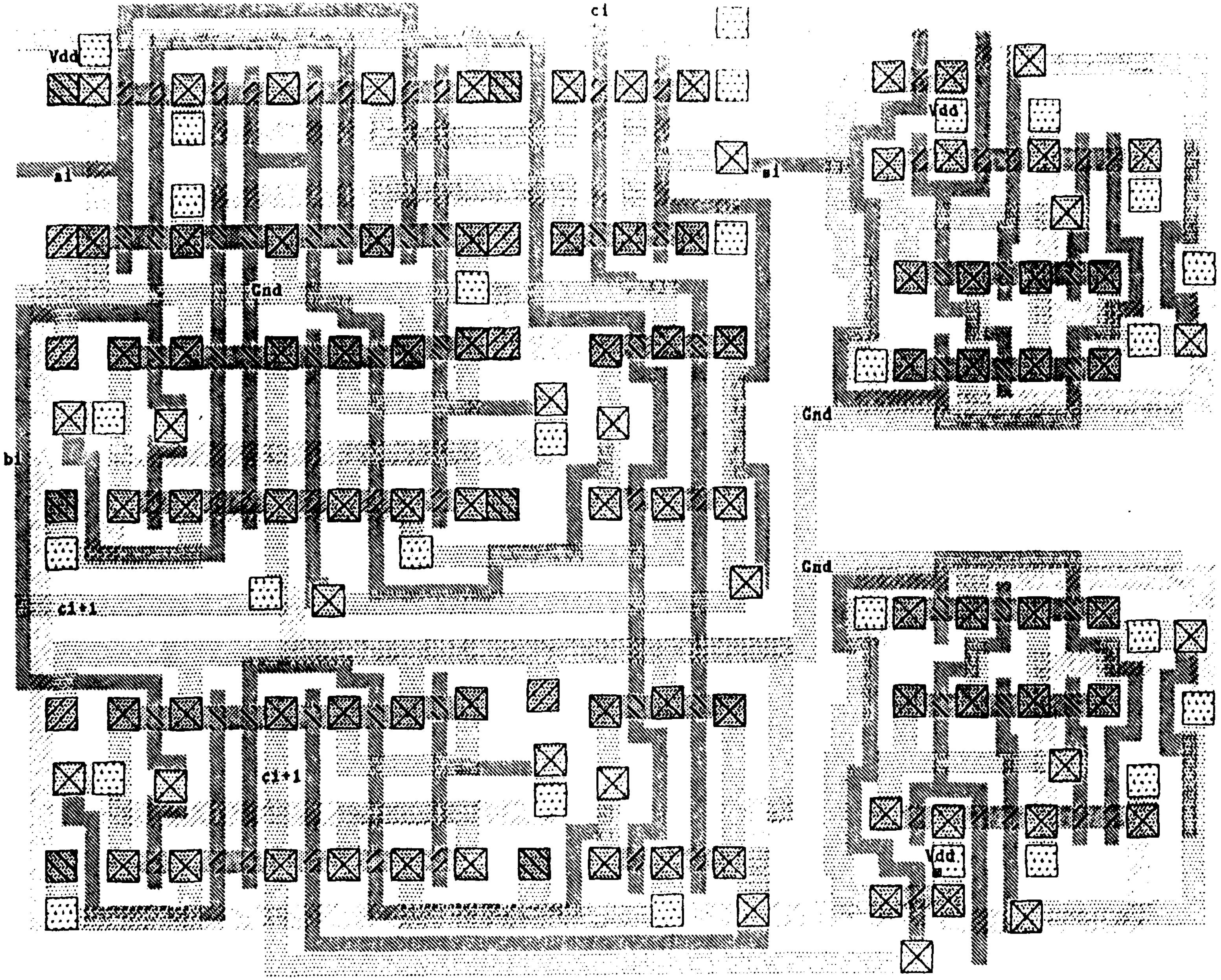


Fig. 16 Layou' of Parity Prediction Adder Cell

is an n-bit binary number, then the corresponding $(n+1)$ bit MRB representation is

$$\mathbf{m} = (m_n, m_{n-1}, \ldots, m_0),$$

where

$$m_0 = b_0$$

$$m_i = b_i \oplus b_{i-1} \text{ for } i = n-1, n-2, \ldots, 1$$

$$m_n = b_{n-1}.$$

Reconversion of $\mathbf{m}$ to $\mathbf{b}$ can be accomplished by

$$b_0 = m_0$$

$$b_i = m_i \oplus b_{i-1} \text{ for } i = n-2, n-3, \ldots, 1$$

$$b_{n-1} = m_n.$$

Note that encoding procedure for MRB codes is non-recursive, whereas the decoding procedure is recursive; however, the latter is not a limitation in the case of addition because of the serial nature of the operation.

We now introduce a set of addition rules so that addition of two MRB coded numbers **A** and **B** will yield the MRB coded sum **S**. These rules are described in Lucal's original paper [Luca, 1959] and are restated below for sake of completion.

(i) The first step, after writing one addend below the other with binary points aligned in the usual fashion, is to group the 1's into pairs. Reading from right to left, column by column, we pair the 1's as they appear, ignoring the 0's. The grouping may be indicated by encircling the two 1's of each pair (see example in Fig. 17). Three different types of pairs may be distinguished. A "horizontal pair" consists of two adjacent 1's in the same addend. A "vertical pair" comprises two

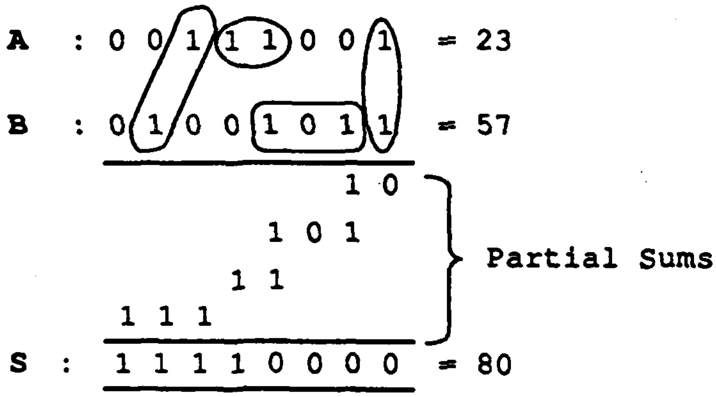


Fig.17 Example of MRB addition.

1's lying in the same column. A "diagonal pair" comprises two 1's which lie in different addends and also in different columns. The term "*ab* pair" will be used to designate any pair which is either vertical or diagonal (i.e., which contains a 1 from **A** and a 1 from **B**). In cases where the second 1 of a pair must be chosen from two 1's in the same column, we take the 1 which is in the same addend to form the pair (i.e., to form a horizontal rather than a diagonal pair).

(*ii*) Next we form the partial sum corresponding to each pair as follows:

a) For a horizontal pair, the partial sum is to have 1's in the two columns occupied by the 1's of the pair. Zeros may be placed in any intervening columns.

b) For a diagonal pair, the partial sum is to have 1's in the two columns occupied by the pair and also a 1 in the next column to the left of the leftmost 1 of the pair.

c) For a vertical pair, the partial sum is to have simply a 1 in the next higher-numbered column. (A zero may be placed in the column of the pair if desired.)

(*iii*) The sum S is then obtained through addition modulo two of the partial sums.

The proof for the procedure outlined above can be found in Appendix I of [Luca, 1959].

We now design an MRB adder cell to implement the rules described above.

The basic adder cell operates on two digits at a time—the two digits of **A** and **B** lying in a single column, beginning with the rightmost column. The information which must be passed along from one column to the next (that is, from one basic adder cell to the next) concerns the pairing of the 1's and the partial-sum carry to the next column when an ab pair is completed. This information may be conveyed by two binary digits, which we denote E and F.

Fig. 18 shows the block diagram of an MRB adder. We assume that we have to add two n-bit binary numbers so the corresponding MRB representations have $n+1$ bits; therefore, the resultant sum contains, at most, $n+2$ bits denoted by S_0 through S_{n+1}.

There are four possible states with respect to the pairing of the 1's after each successive cell has been inspected. These four states may be represented by the E and F digits as follows:

$E = 0$ and$F = 1$ indicates that an ab pair has just been completed and that a 1 should be carried to the next cell.

$E = 1$ and $F = 1$ indicates that the first 1 of the next pair has appeared in A.

$E = 0$ and $F = 0$ indicates that the first 1 of the next pair has appeared in B.

$E = 1$ and $F = 0$ indicates that the preceding pair (if any) has been completed, no carry is required, and the first 1 of the next pair has not yet appeared.

The above encoding information and the addition rules described earlier are used to generate the truth table for each adder cell. This is shown in Table 4.

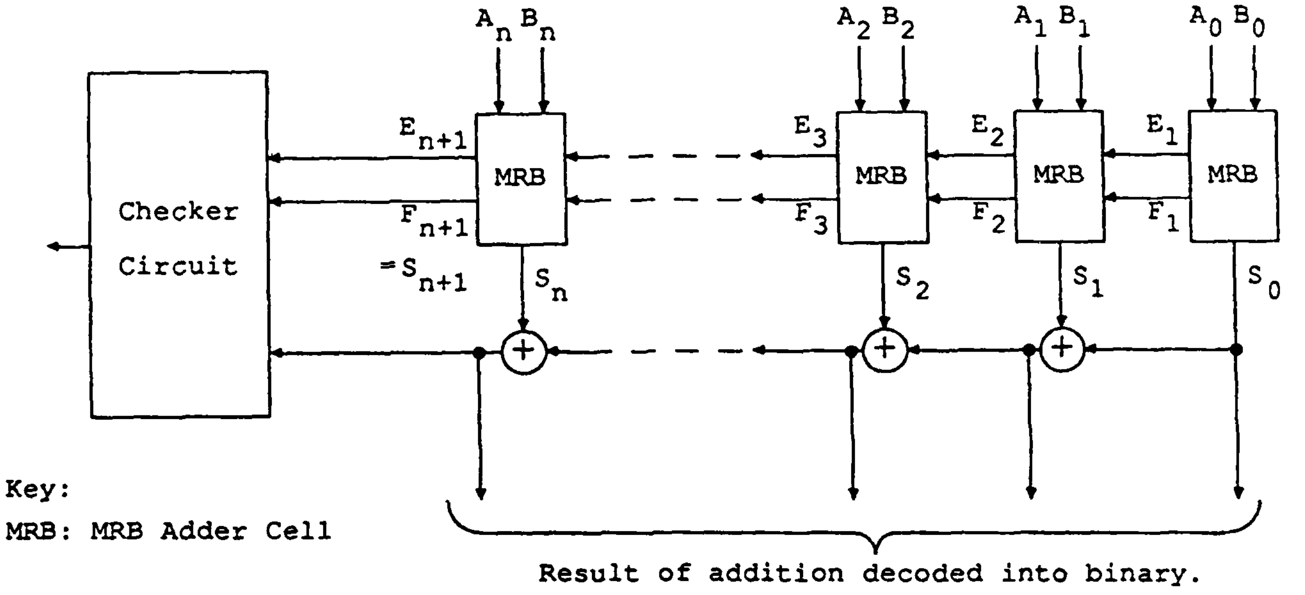


Fig.18 MRB Addition Scheme.

E_k	F_k	A_k	B_k	S_k	E_{k+1}	F_{k+1}
0	0	1	1	0	1	1
0	0	0	1	1	1	0
0	0	1	0	1	0	1
0	0	0	0	0	0	0
0	1	1	1	1	0	1
0	1	0	1	0	0	0
0	1	1	0	0	1	1
0	1	0	0	1	1	0
1	0	1	1	0	0	1
1	0	0	1	1	0	0
1	0	1	0	1	1	1
1	0	0	0	0	1	0
1	1	1	1	0	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0
1	1	0	0	0	1	1

Table 4. Truth Table of Adder Operation

Defining the intermediate signals x_k, y_k, v_k, and w_k as:

$$x_k = A_k \oplus B_k$$

$$y_k = \bar{E}_k \cdot F_k$$

$$v_k = E_k + F_k$$

$$w_k = E_k \cdot F_k$$

the following minimized expressions are obtained from the truth table:

$$S_k = x_k \oplus y_k$$

$$E_{k+1} = B_k \oplus v_k$$

$$F_{k+1} = A_k \oplus w_k.$$

The layout of a composite MRB adder cell, which also includes three EX-OR gates for encoding of inputs and decoding of output, is shown in Fig. 19. Note that in Fig. 18 the decoder circuit is also used to obtain the parity of the MRB sum. As

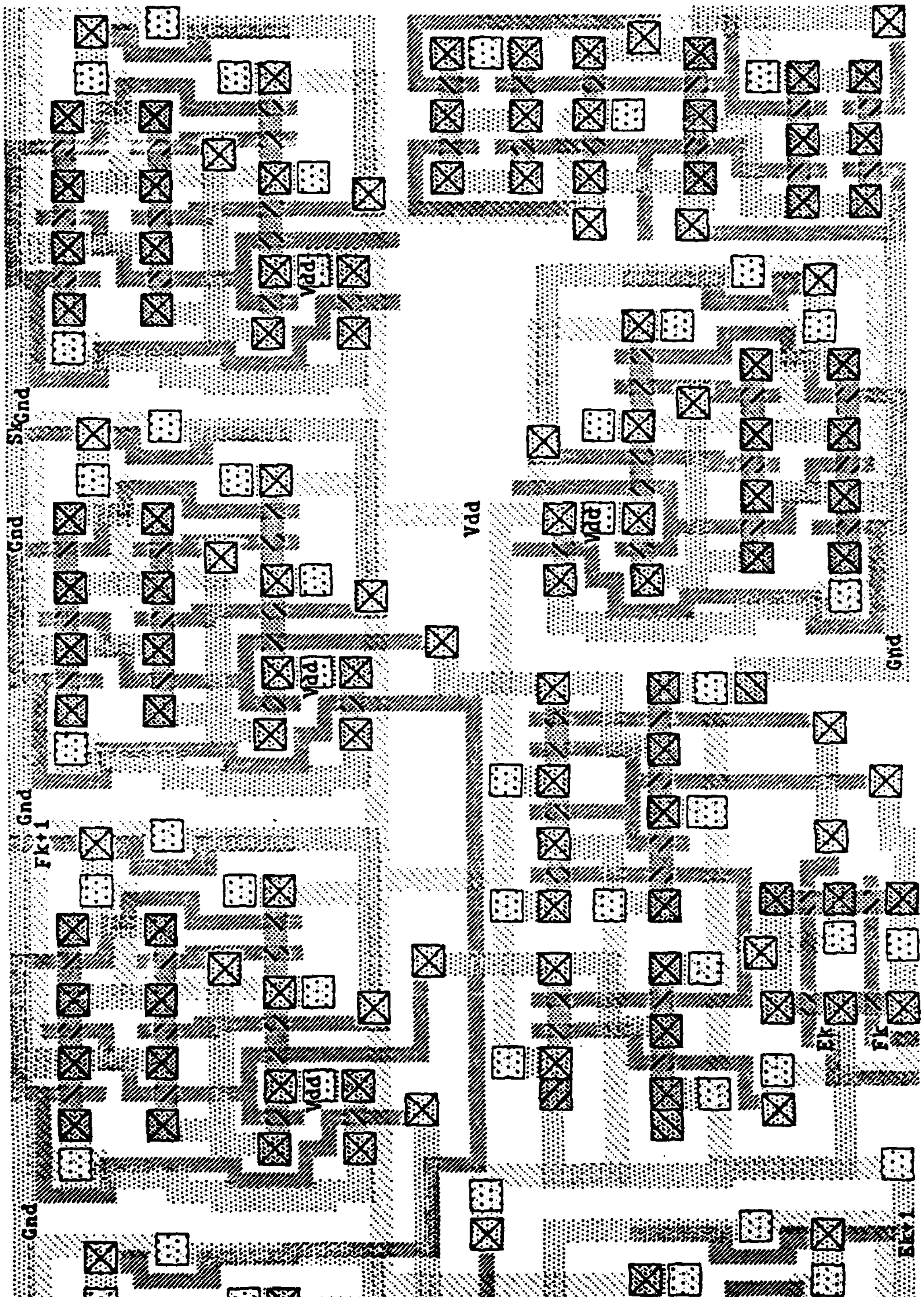


Fig. 19 Layout of MRB Adder Cell

in the case of the Parity Prediction Adder, this tree of EX-OR gates is self-checking with respect to all possible single stuck-at faults.

During normal operation the outputs E_{n+1} and F_{n+1} (see Fig. 18) are 01 or 10 and

$$S_{n+1} = S_n \oplus S_{n-1} \oplus \cdots \oplus S_0.$$

If these conditions are satisfied in the presence of any single stuck-at fault, except those at the primary inputs, then the output of the adder is correct. Again, we emphasize that stuck-at faults at the primary inputs cannot be detected by any scheme.

Section 3.4. Comparison of Adder Schemes

The area requirements of the layouts illustrated in Figs. 12, 14, 16, and 19 are shown in Table 5.

Adder Cell	Dimensions in μm	Area in $(\mu m)^2$	Ratio
Nonredundant	$145 \cdot 5 \times 111 \cdot 0$	$16 \cdot 15 \times 10^3$	1
Duplicated	$162 \cdot 0 \times 295 \cdot 5$	$47 \cdot 87 \times 10^3$	$2 \cdot 96$
Parity Prediction	$183 \cdot 0 \times 229 \cdot 5$	$41 \cdot 99 \times 10^3$	$2 \cdot 60$
MRB	$172 \cdot 5 \times 318 \cdot 0$	$54 \cdot 86 \times 10^3$	$3 \cdot 39$

Table 5

For purposes of performance evaluation we define the "confidence level" of a circuit as the probability of the event that either the circuit is fault free or a detectable fault has occurred in the circuit. Note that in a nonredundant circuit the confidence level is equal to the reliability of the circuit.

We now compare the confidence level of a nonredundant adder cell, CL(NRA), with that of a duplicated adder cell, CL(DA). In our comparison we make the following assumptions:

(*i*) Stuck-at faults are the only source of errors in the circuit.

(*ii*) The stuck-at faults considered are only those that occur either at the gate of any transistor or at the input or output of any CMOS gate.

(*iii*) The occurrence of stuck-at-faults are statistically independent events.

(*iv*) All the possible single stuck-at faults have the same probability of occurrence.

To calculate the confidence level of the adder cells we count the number of nodes in each cell which are potential sites for stuck-at faults. Let p be the probability that a stuck-at fault occurs at any node of the circuit. Our count gives the following expressions:

$$\mathrm{CL(NRA)} = (1-p)^{94}$$

$$\mathrm{CL(DA)} \geq (1-p)^{260} + 256\, p(1-p)^{259}.$$

Note that the expression given for CL(DA) is a lower bound because certain multiple faults will be detected by this scheme. For a circuit to be reliable, the value of p should be extremely small. In such a situation the lower bound given above is a good estimate of CL(DA). Fig. 20 shows a plot of CL(NRA) and CL(DA) for values of p less than 5×10^{-3}. Note that CL(DA) is greater than CL(NRA) for values of p less than 5×10^{-3}, while the trend is reversed for p greater than 5×10^{-3}.

Section 3.5. MRB Multiplier

In this section we propose a pipelined multiplication scheme using the MRB adder developed earlier. The MRB cell was chosen to implement the multiplier because, as shown in Appendix I, only one of the two operands need to be encoded in MRB representation. This would reduce the size of the MRB cell and, hence, that of the multiplier.

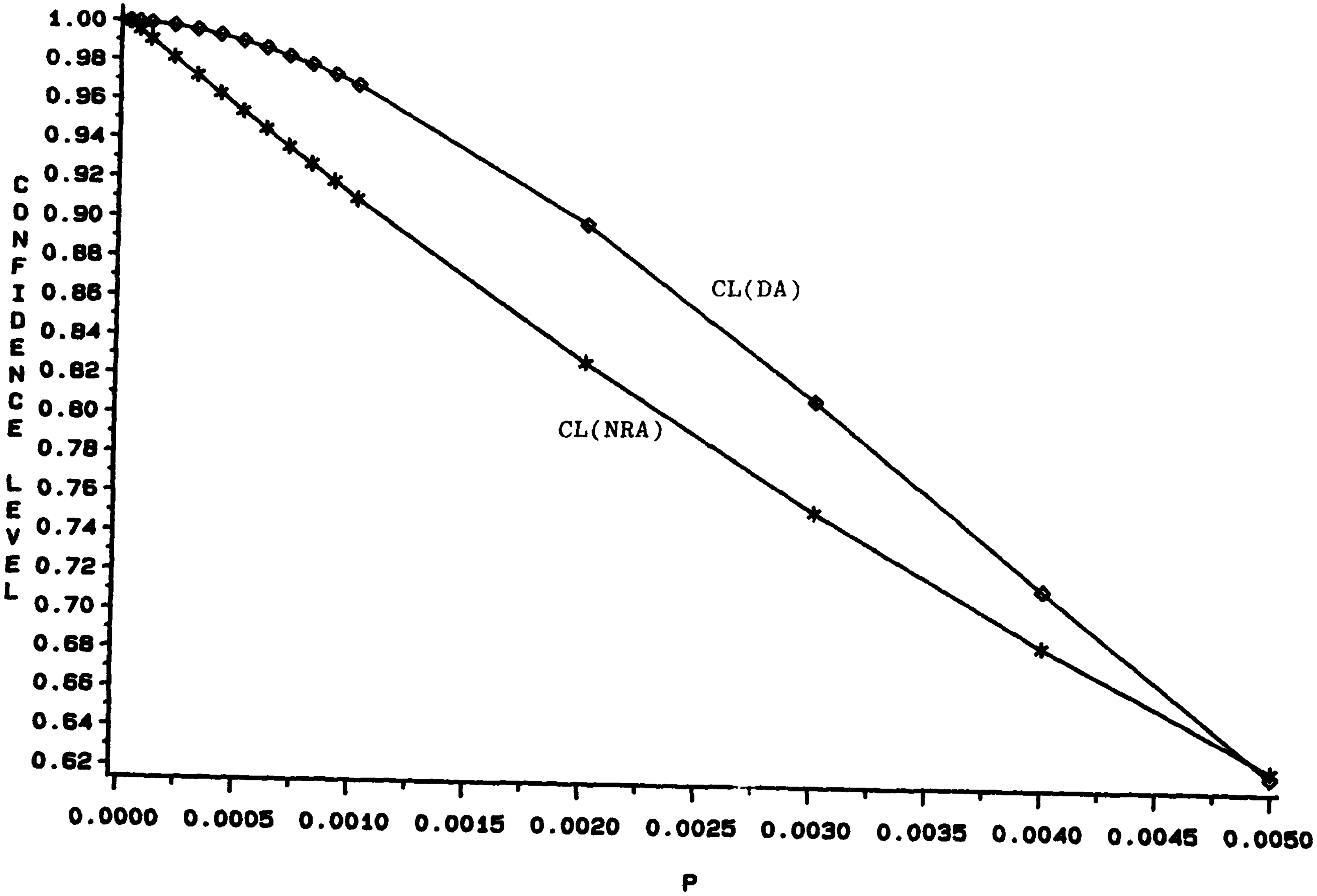


Fig. 20 Confidence Level of Adder Circuits

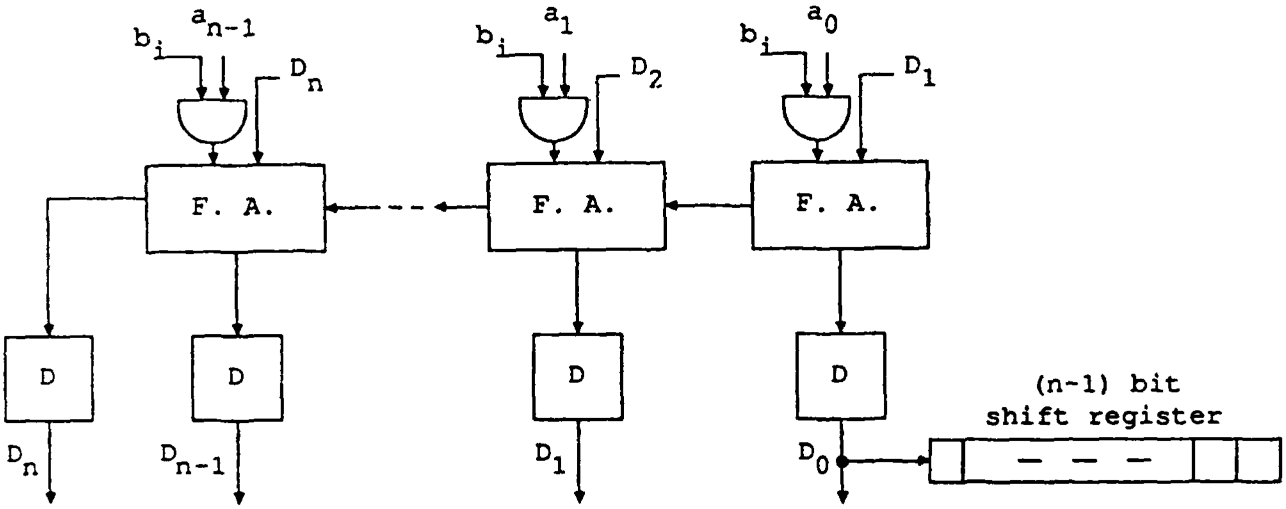
Fig. 21 shows the block diagram of a pipelined binary multiplier where $(a_{n-1}, a_{n-2}, \ldots, a_0)$ and $(b_{n-1}, b_{n-2}, \ldots, b_0)$ are the numbers to be multiplied. Bit b_i is used to generate the partial product at the ith stage by adding $(a_{n-1}, a_{n-2}, \ldots, a_0)$ when $b_i = 1$ and adding $(0, 0, \ldots, 0)$ when $b_i = 0$ to the partial product generated at the $(i-1)$th stage. This partial product is stored in the D flip flops and the shift register. After n steps, the final product is stored in the $(n+1)$ D flip flops and $(n-1)$ bit shift register.

Fig. 22 shows the block diagram of a pipelined MRB multiplier. The principle of operation is similar to that of the binary multiplier discussed before. In Fig. 22 $(A_n, A_{n-1}, \ldots, A_0)$ is the MRB representation of a binary number $(a_{n-1}, a_{n-2}, \ldots, a_0)$. The partial products in this scheme are in MRB representation. E_0 is the complement of the parity of the partial product bits stored in the shift register. The value of E_0 is required by the MRB adder cell to generate successive partial products.

This scheme will detect all single stuck-at faults except those at the primary inputs.

Section 3.6. Conclusion

In this section we studied several schemes for the design of self-checking arithmetic circuits. We have concluded that all the self-checking adder schemes studied require considerable area overhead. In particular, the MRB adder scheme requires the maximum overhead because of the additional circuitry needed for encoding and decoding. MRB codes could provide an attractive technique for designing self-checking systems if the encoding and decoding are done at the primary inputs and outputs of the system, whereas all internal data is MRB coded.

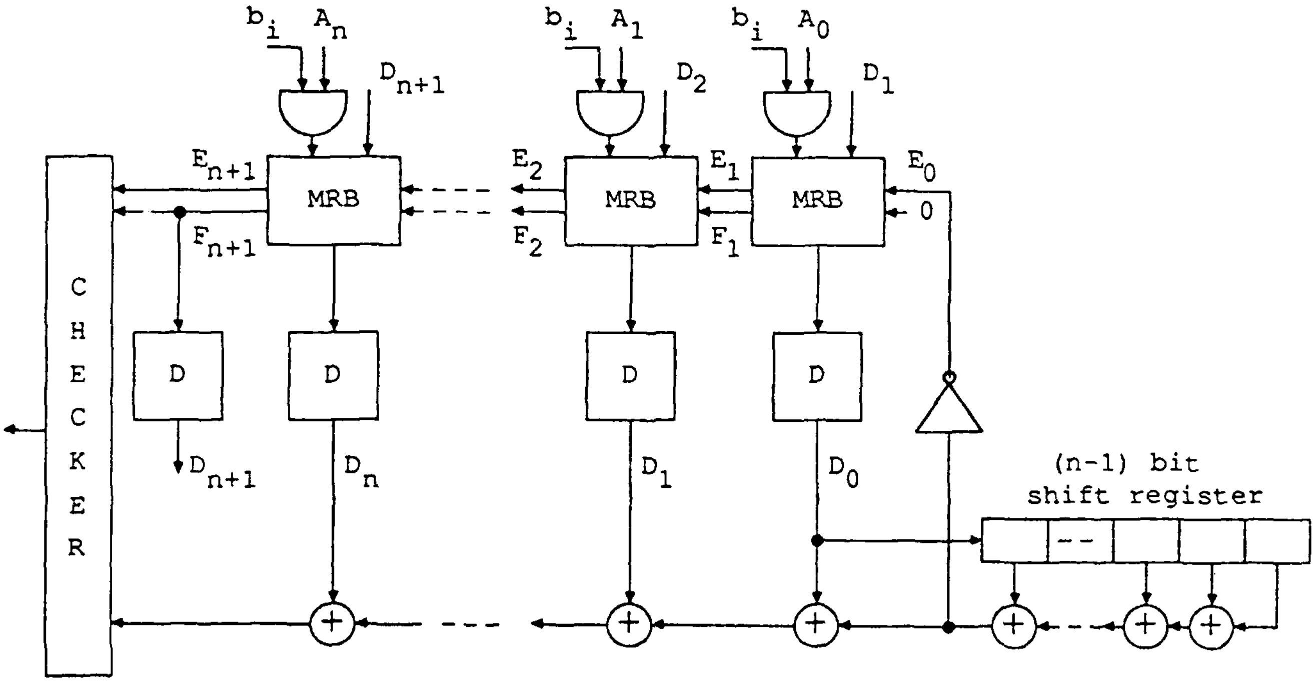


Key:

F.A.: Full Adder Cell

D: D Flip-Flop

Fig.21 Pipelined Binary Multiplier.



Key:
MRB: MRB Adder Cell
D: D Flip-Flop

Fig.22 Pipelined MRB Multiplier.

We have also proposed a technique for designing self-checking pipelined multipliers in which only one of the multiplicands needs to be MRB coded.

Section 4. Residue Codes

In this section we investigate the use of residue codes to design self-checking and fault-tolerant adder circuits.

An error is said to occur in the addition operation if the actual output $\mathbf{Z}'$ of the adder differs from the expected value $\mathbf{Z}$. The error pattern $\mathbf{E}$ is defined as

$$\mathbf{E} = \mathbf{Z}' - \mathbf{Z}.$$

If $\mathbf{Z} = (z_{n-1}, z_{n-2}, \ldots z_0)$ and $\mathbf{Z}' = (z'_{n-1}, z'_{n-2}, \ldots z'_0)$, then $\mathbf{E} = (e_{n-1}, e_{n-2}, \ldots, e_0)$, where $e_i = z'_i - z_i \;\; \forall i = 0, 1, \ldots, n-1$. Note that $e_i \in \{-1, 0, 1\}$. For example, if $\mathbf{Z}' = 110001$ and $\mathbf{Z} = 101101$, then $\mathbf{E} = 01\bar{1}\bar{1}00$ where $\bar{1}$ denotes -1 in the error pattern.

For every pattern $\mathbf{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$, $x_i \in \{-1, 0, 1\}$, we define $\delta(\mathbf{x})$ as

$$\delta(\mathbf{x}) = \sum_{i=0}^{n-1} x_i 2^i.$$

Note that $\delta(\mathbf{E}) = \delta(\mathbf{Z}') - \delta(\mathbf{Z})$ and $\delta(\mathbf{E})$ is called the error value. This mapping, δ, of error patterns into error values is a conversion of the n-tuples into integral values and is not a one-to-one correspondence. As examples, the error patterns $01\bar{1}\bar{1}00$, $001\bar{1}00$, and 000100 all correspond to the same error value $\delta(\mathbf{E}) = 4$.

We now show that the concepts of Hamming weight and distance [Pete,1972] are not appropriate for dealing with arithmetic errors. Suppose we wish to add 0001 and 0111; the correct result is 1000. However, if a single fault changes the first number to 0000 the result is 0111, whose Hamming distance from the correct result is four. Thus the Hamming weight of an arithmetic error can be considerably larger than the number of single bit failures needed to produce it. This motivates the definition of the binary arithmetic weight.

Definition. The binary arithmetic weight of an integer N, denoted $W(N)$, is the minimum number of terms in an expression of the form

$$N = a_1 2^{j_1} + a_2 2^{j_2} + \cdots + a_k 2^{j_k}$$

where $a_i \in \{-1, 1\}$. This expression is said to be in minimal form.

For instance, the decimal number 31 has a binary representation 11111, but this is certainly not in a minimal form. Its minimal form is $10000\bar{1}$ and thus $W(31) = 2$. Note that such a minimal representation is not unique. For example, $W(13) = 3$ and the integer 13 has two minimal representations given by 01101 and $10\bar{1}01$.

The general problem is to design an adder that can detect (correct) all error patterns whose binary arithmetic weight is less than or equal to some given integer t. One way to accomplish this task is to use residue codes [Rao, 1974].

The residue code (modulo $A, A \neq 0$) corresponding to any integer N is $[N, |N|_A]$ where $|N|_A$ is the remainder formed when N is divided by A. In other words, there exist integers q and $|N|_A$ such that

$$N = qA + |N|_A; \;\; 0 \leq |N|_A < A.$$

Section 4.1. Error Detection Using Residue Codes

In this section we investigate the use of residue codes for designing self-checking adders. The block diagram of such an adder with addends N_1 and N_2 is shown in Fig. 23. In the fault-free situation it can be easily shown that

$$|N_1 + N_2|_A = \big||N_1|_A + |N_2|_A\big|_A.$$

We now investigate the conditions under which error detection is possible. If only one of the inputs to the error detector is greater than or equal to A, then the fault

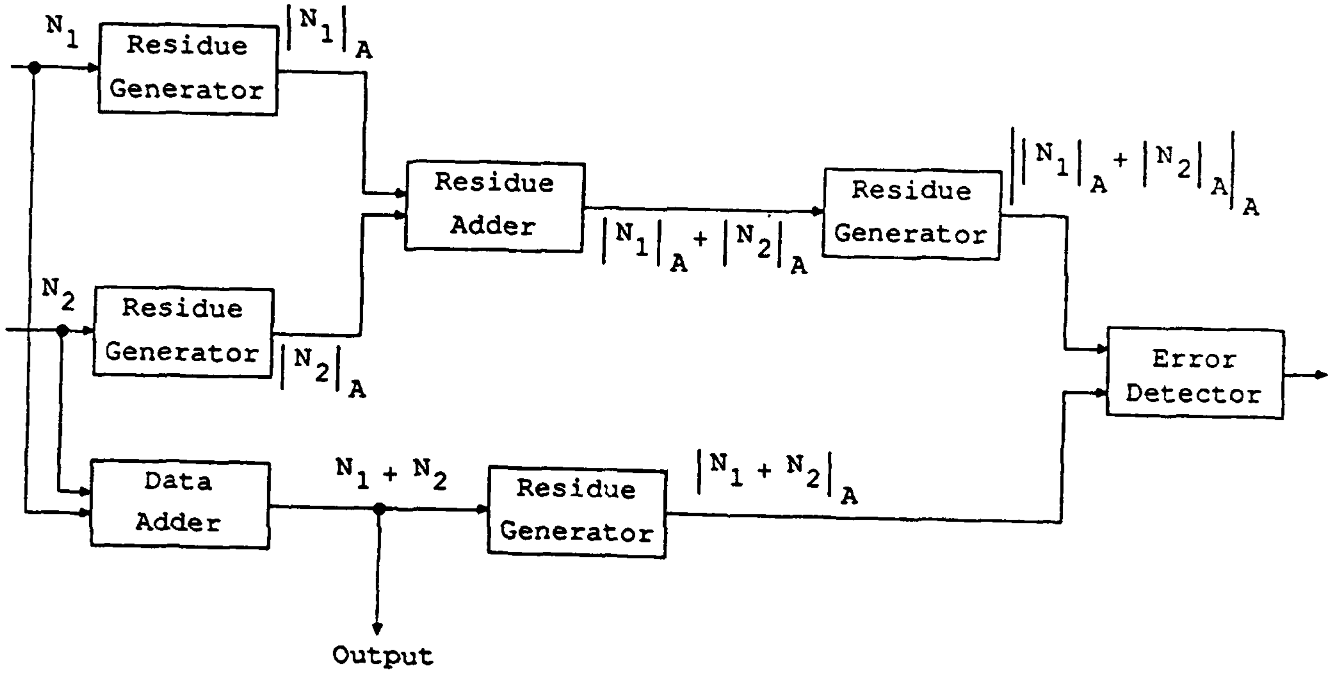


Fig.23 Residue Code Adder

is always detected. First, let us consider only faults in the data adder causing its output to be $N_1 + N_2 + E_D$. This error cannot be detected if and only if

$$|N_1 + N_2 + E_D|_A = \big||N_1|_A + |N_2|_A\big|_A.$$

It can be easily shown that the above condition is equivalent to

$$|E_D|_A = 0.$$

An analogous result is obtained if the fault is in the residue generation following the data adder.

If the fault is in the residue adder or in the corresponding residue generators, then the input from this half of the circuit to the error detector is

$$\big||N_1|_A + |N_2|_A + E_R\big|_A.$$

This error cannot be detected if and only if

$$|N_1 + N_2|_A = \big||N_1|_A + |N_2|_A + E_R\big|_A.$$

It can be shown that this condition is equivalent to

$$|-E_R|_A = 0.$$

Let us now assume that a fault occurs either in the data adder part or in the residue adder part of the circuit, but not in both. Furthermore, we are interested in detecting faults that manifest as errors such that either $W(E_D) = 1$ or $W(-E_R) = 1$. The importance of detecting these errors is that for every E_D (or E_R) satisfying $W(E_D) = 1$ (or $W(-E_R) = 1$) there always exists a single stuck-at fault that also causes the same E_D (or E_R). Note that the set of faults that causes $W(E_D) = 1$ or $W(-E_R) = 1$ is circuit dependent.

The necessary and sufficient condition for these errors to be detected is $A \neq 2^j$ where $j \in \{0, 1, \ldots, n\}$ where n is the number of bits in each addend. Residue generation involves the division operation which is inherently more complex than addition. Hence in designing a scheme to check addition we should choose a value of A that causes the residue generation operation to be as simple as possible. As suggested in [Rao, 1974], A should be of the form $2^c - 1$ where the integer $c > 1$. In this case $|N|_A$ is found by repeated addition of c bits of N with end-around carry. Furthermore, Wakerly has described a scheme for designing a modulo 3 residue tree without addition [Wake, 1978]. Implementation of an error- detecting scheme using these residue generators requires approximately an overhead of 40% more than that required by a duplication scheme when both schemes are implemented using two input gates and the residue generators are non-trivial, i.e., $A \leq 2^{n+2} - 2$. Moreover, the duplication scheme detects all single stuck-at faults. Thus duplication is more efficient than residue code scheme given in [Wake, 1978], both in terms of fault coverage and gate count.

In conclusion, we remark that if $A > 2^{n+2} - 2$, where n is the number of bits of each addend, the scheme given in Fig. 23 reduces to duplication.

Section 4.2. Error Correction Using Residue Codes

In this section we study the application of residue codes to the design of fault-tolerant adders.

We first show that the class of residue codes discussed in the previous section cannot be used for error correction. For purposes of error correction we define the syndrome S as the difference (modulo A) between the inputs of the error detector of Fig. 23. If the error is in the data adder part of the circuit, then

$$S = \Big| \big|N_1 + N_2 + E_D\big|_A - \big| |N_1|_A + |N_2|_A \big|_A \Big|_A = |E_D|_A.$$

If the error is in the residue adder part of the circuit, then

$$S = \left|\left|N_1 + N_2\right|_A - \left||N_1|_A + |N_2|_A + E_R\right|_A\right|_A = |-E_R|_A.$$

Note that there are errors E_D and E_R such that $W(E_D) = W(-E_R) = 1$ and $|E_D| = |-E_R|$, hence we cannot correct errors with binary arithmetic weight equal to 1.

So we now introduce the concept of bi-residue codes, which is a generalization of residue codes. The bi-residue code corresponding to any integer N is $[N, |N|_A, |N|_B]$ where A and B are positive integers. A possible scheme for an error-correcting adder using bi-residue code is shown in Fig. 24. As in the previous section, the residue generators are assumed to be implemented using the scheme in [Wake, 1978]. In this case, such a scheme for single-error correction requires an overhead of at least 100% more than that required by a Triple Modular Redundancy (TMR) scheme when both are implemented using two input gates and the residue generators are non-trivial, i.e., $A, B \leq 2^{n+2} - 2$. Hence TMR is more efficient than bi-residue codes in the design of single error correcting adders, both in terms of fault coverage and gate count.

We end by re-emphasizing that the mapping between faults and error patterns is circuit dependent and that if $A, B > 2^{n+2} - 2$, then the scheme shown in Fig. 24 reduces to TMR.

Section 4.3. Conclusion

In this section we studied the feasibility of using residue codes for the detection and correction of errors in binary fixed point adders. The inherent complexity of the residue generation operation substantially increases the area overhead. Our study reveals that the design of single error detecting and correcting schemes using residue codes fare poorly in comparison to duplication and TMR, respectively.

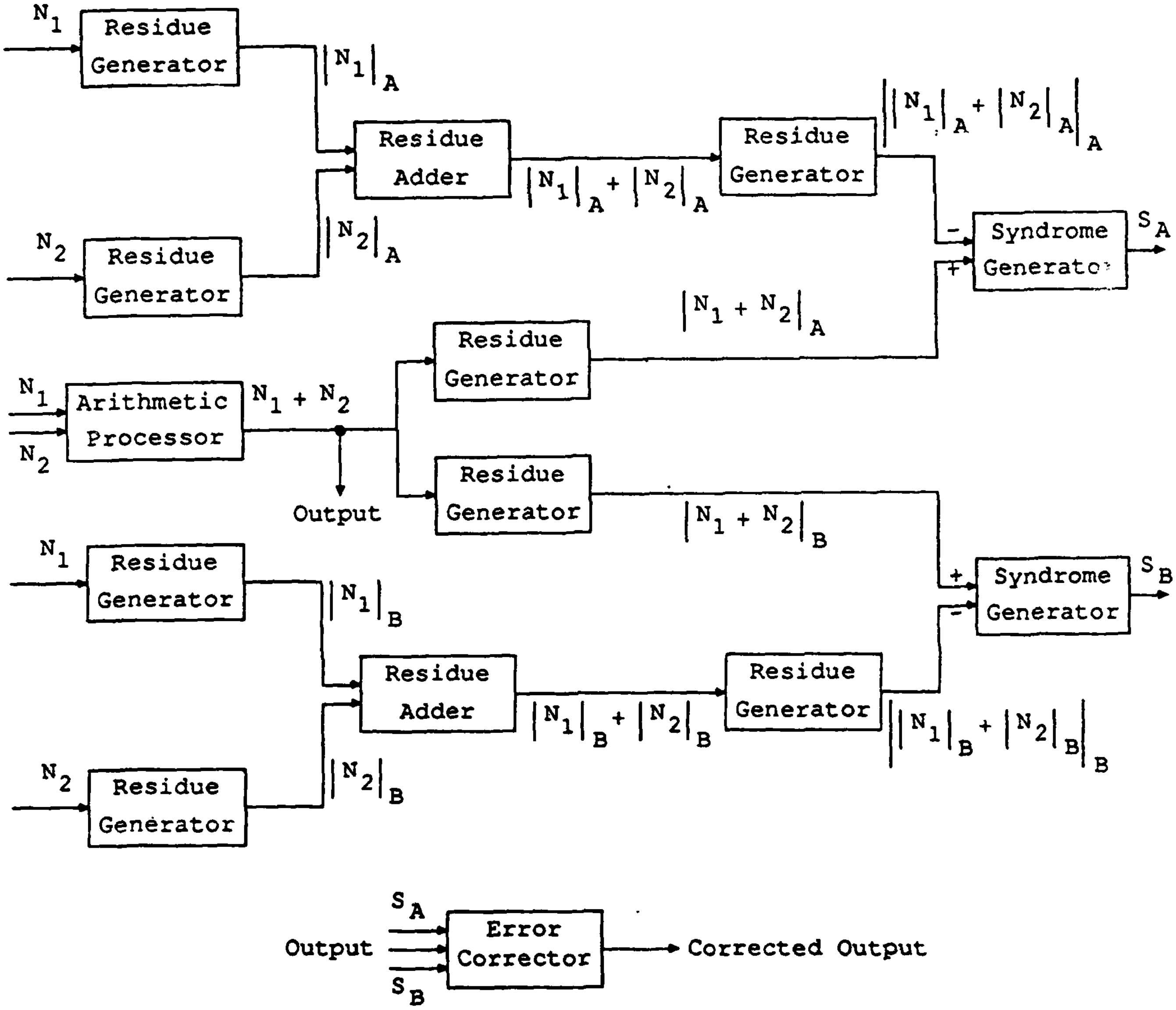


Fig.24 Error Correction using Biresidue Codes.

Section 5. Conclusion

In this report two schemes have been proposed to detect errors caused by a single short in a 32-bit bus. The first scheme is based on a modified form of the Berger code. The check bits are generated from the 32 information bits by cascading a number of ROMs. The second scheme considered for detecting a single short in a 32-bit bus is to time-multiplex both the information bits and the test vectors on the bus. If the bus is "flat," then the test vector basically consists of alternate 0's and 1's. For an n-bit "non-flat" bus we need at least $\lceil \log_2 n \rceil$ test vectors, and it is therefore impractical to transmit these vectors in one clock cycle. Also, a scheme for correcting a single-bit error and simultaneously detecting a double-bit error in a 32-bit bus is proposed.

In this report we also studied several schemes for the design of self-checking arithmetic circuits. We have concluded that all the self-checking adder schemes studied require considerable area overhead. In particular, the MRB adder scheme requires the maximum overhead because of the additional circuitry needed for encoding and decoding. MRB codes could provide an attractive technique for designing self-checking systems if the encoding and decoding are done at the primary inputs and outputs of the system, whereas all internal data is MRB coded. We have also proposed a technique for designing a self-checking pipelined multiplier in which only one of the multiplicands needs to be MRB coded.

Finally, we studied the feasibility of using residue codes for the detection and correction of errors in binary fixed point adders. The inherent complexity of the residue generation operation substantially increases the area overhead. Our study reveals that the design of single error detecting and correcting schemes using residue codes fare poorly in comparison to duplication and TMR, respectively.

Appendix I

Let

$$A = (a_{n-1}, a_{n-2}, \ldots, a_0) \text{ and}$$

$$B = (b_{n-1}, b_{n-2}, \ldots, b_0)$$

be the n-bit binary representation of two integers to be multiplied. Thus, the MRB representation of A, denoted by $M(A)$, is

$$M(A) = (a_{n-1}, a_{n-1} \oplus a_{n-2}, a_{n-2} \oplus a_{n-3}, \ldots, a_1 \oplus a_0, a_0).$$

Let $2^j M(A)$ denote the j-bit left shift of $M(A)$, i.e.,

$$2^j M(A) = (a_{n-1}, a_{n-1} \oplus a_{n-2}, \ldots, a_1 \oplus a_0, a_0, \underbrace{0, 0, \ldots, 0}_{j \text{ zeros}}).$$

It can be easily shown that $2^j M(A) = M(2^j A)$ where $2^j A$ represents the j-bit left shift of the number A.

In order to show that the block diagram of Fig. 22 computes the correct result we must show that

$$b_0 2^0 M(A) +_M b_1 2^1 M(A) +_M \cdots +_M b_{n-1} 2^{n-1} M(A) = M(A * B),$$

where $+_M$ and $*$ denote MRB addition and binary multiplication, respectively. Using the fact that $2^j M(A) = M(2^j A)$, we may write

$$\begin{aligned}
& b_0 2^0 M(A) +_M b_1 2^1 M(A) +_M \cdots +_M b_{n-1} 2^{n-1} M(A) \\
&\qquad = M(b_0 2^0 A) +_M M(b_1 2^1 A) +_M \cdots +_M M(b_{n-1} 2^{n-1} A) \\
&\qquad = M(b_0 2^0 A + b_1 2^1 A + \cdots + b_{n-1} 2^{n-1} A) \\
&\qquad = M\left(A * \sum_{i=0}^{n-1} b_i 2^i\right) \\
&\qquad = M(A * B) \qquad \blacksquare
\end{aligned}$$

References

[Berg, 1961] Berger, J. M. "A Note on Error Detecting Codes for Assymmetric Channels." *Information and Control*, vol. 4, March 1961, pp. 68–73.

[Bose, 1985] Bose, Bella, and Der Jei Lin. "Systematic Unidirectional Error Detecting Codes." *IEEE Transactions on Computers*, Nov. 1985, pp. 1026–1032.

[Hart, 1987] Hartmann, C. R. P., and Parag K. Lala. "Fault Model Development for Fault Tolerant VLSI Design." Final Report for RADC Contract No. F30602-81-C-0167, Task N-6-5788, October 1987.

[Hsia, 1970] Hsiao, M. Y. "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes." *I.B.M. Journal of Research and Development*, 1970, pp. 395–401.

[Lala, 1985] Lala, P. K. *Fault Tolerant and Fault Testable Hardware Design.* Prentice Hall International, 1985.

[Luca, 1959] Lucal, H. M. "Arithmetic Operations for Digital Computers Using Modified Reflected Binary Code." *IRE Transactions on Electronic Computers*, Dec. 1959, pp. 449–458.

[Pete, 1972] Peterson, W. W., and E. J. Weldon, Jr. *Error Correcting Codes.* 2nd ed. MIT Press, 1972.

[Prad, 1986] *Fault Tolerant Computing—Theory and Techniques*, Vol. I. Ed. Pradhan, D. K. Prentice Hall, 1986.

[Rao, 1974] Rao, T. R. N. *Error Coding for Arithmetic Processors.* Academic Press, 1974.

[Wake, 1978] Wakerly, John. *Error Detecting Codes, Self-Checking Circuits and Applications*. North Holland, 1978.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.