Memory testing by linear checks

Prof. M. Karpovsky, Mem. I.E.E.E.

Indexing terms: *Errors and error analysis, Memory testing*

Abstract: In this paper we consider methods of error detection, location or correction in ROM or RAM by systems of orthogonal linear equality and inequality checks and by the Rademacher transform. Implementations and error detecting, locating, correcting capabilities of these techniques for ROM and RAM testing are also described.

1 Introduction

In this paper we shall consider the problems of error detection, location and correction for read-only memories (ROM) and for random-access memories (RAM).

It is well known that memory testing is one of the bottlenecks of the computer industry. In this paper we shall use the functional testing approach [10, 22, 23] for memory testing.

The errors we shall consider may result from a wrong masking in ROM, stuck-at and intermittent faults in cells and in the address decoder, bridgings between cells and wrong decoding of an address. Pattern-sensitive and transient errors will not be discussed in this paper.

To solve these problems, we shall use the techniques developed in Reference 1–5, 17 and 21. These techniques are based on error-correcting codes, Walsh transforms and least-absolute-error polynomial approximations.

Let $Z = (Z_1, \ldots, Z_n)(Z_i \in \{0, 1\})$ be an address of the cell containing the data $f(Z)$. We denote by G the set of all binary n-vectors, and we shall consider G as a group with respect to the operation $\oplus$ of componentwise addition mod 2.

In References 1–4 the methods of error detection based on linear equality checks,

$$\sum_{\tau \in T} f(Z \oplus \tau) - C = 0 \quad \text{for every } Z \in \{0, 1, \ldots, 2^n - 1\} \tag{1}$$

were developed.

In eqn. 1, T is a subgroup of G and C is a constant (we use the same letter to denote an integer and its binary representation). The verification of whether expr. 1 is satisfied constitutes the error-detection method. In the case $T = G$, the check of expr. 1 is the well known control-sum check. In References 1 and 2 the technique based on Fourier transforms over finite groups was proposed for the construction of optimal checks so as to minimise the cardinality of a check set T. Advantages and limitations of the error-detection method based on the linear checks of expr. 1 and their error-detecting capability were also considered in Reference 1.

The problem of error correction in a program or device computing $f(Z)$ by a system of linear checks was studied in References 2 and 3. Decoding methods for the results of these checks (syndromes), complexity of the decoding, error-detecting and error-correcting capabilities for a system of checks of expr. 1 were also considered in References 2 and 3.

In Reference 4 the general properties of the minimal check set $T(f)$ for the given function f were studied. Minimal sets $T(f)$ for the important case when f is a polynomial were also described in Reference 4. In particular, it was shown that if $f(Z)$ is a polynomial of degree d, then $T(f)$ may be chosen as a linear error-correcting code which is dual to any code detecting d independent errors. Methods of error detection based on linear equality checks or expr. 1 may be effectively used in the case where $f(Z)$ is an integer for every $Z \in \{0, 1, \ldots, 2^n - 1\}$, but very few noninteger functions have nontrivial checks of this type ([4, 5]). Reference 5 suggests the use of the following linear inequality checks for error detection in the case of noninteger computations:

$$\left| \sum_{\tau \in T} f(Z \oplus \tau) - C \right| \leqslant \varepsilon \quad \text{for every } Z \in \{0, 1, \ldots, 2^n - 1\} \tag{2}$$

where T is a subgroup of G, C is a constant and $\varepsilon \leqslant 0$ is a small constant (the checks of expr. 1 is a special case of expr. 2 with $\varepsilon = 0$). The general properties of a minimal check set $T = T(f, \varepsilon)$ for the given function f and ε, methods of construction for these minimal check sets, the advantages and limitations of inequality checks, and their error-detecting capability were considered in Reference 5.

In this paper we shall use systems of equality and inequality checks for error-detection, location and correction in a ROM and a RAM. In Appendix 9.1, the optimal equality checks are given for the standard computer blocks. Optimal inequality checks for analytical functions in the case $n = 23$, $\varepsilon = 5 \times 10^{-3}$ are given in Appendix 9.2. (For the sake of completeness, we include, in Appendixes 9.1 and 9.2, some results from References 2 and 4.)

In Section 2 we shall discuss the application of the linear equality checks of expr. 1 to the correction of single and double errors in a ROM. (The definition of the multiplicity of an error will be given later.) This technique is efficient in the case where f is an analytical function and $f(Z)$ is an integer for every Z.

Section 3 will be devoted to the location of single and double errors in a ROM containing values of noninteger functions. In this case we shall use the systems of inequality checks of expr. 2.

Error-correcting and error-locating techniques developed in Section 2 and 3 are efficient only in the case where $f(Z)$ is an analytical function. We shall propose, in Section 4, another approach for error detection/correction in a ROM containing random data based on the so-called 'Rademacher transform' [7]. This approach will provide us with a simple method for the correction of single errors and detection of double and triple errors.

In Section 5, we shall apply the ROM testing techniques developed in Sections, 2–5 to the testing of a RAM. In Section 6 we compare these techniques with some well known techniques [10–14] widely used in RAM testing.

The testing methods proposed in this paper may be

Paper 3243E (C2) received 1st August 1983

The author is with the Department of Computers, System and Electrical Engineering, College of Engineering, Boston University, 110 Cummington Street, Boston, MA 02215, USA

(ii)*b* If $S_1(v) \neq S_2(v)$, $S_1(v) = S_2(w)$ (see Fig. 2*b*), then $Z^{(2)} = w$, $e(Z^{(2)}) = S_1(w) = S_1(v)$, and there exists $v' \in v \oplus T_2(v' \neq v)$, such that $e(v) \neq 0$ and $Z^{(1)} = v'$.

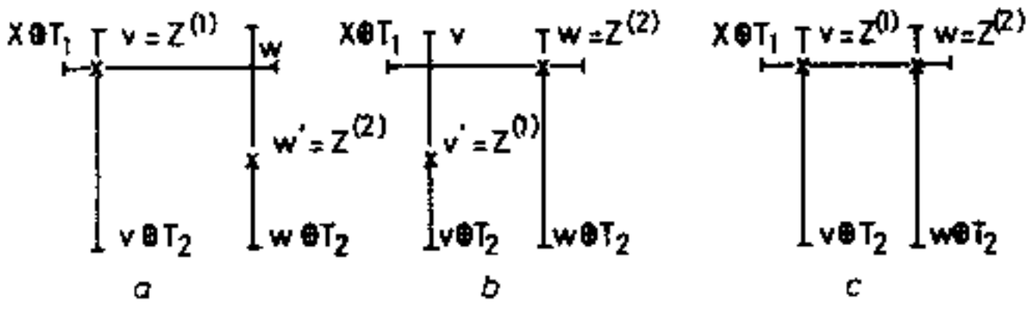

Fig. 2 *Correction of the double error* $e(Z) = \delta_{Z,Z^{(1)}} e(Z^{(1)}) + \delta_{Z,Z^{(2)}} e(Z^{(2)})$

(ii)*c* If $S_1(v) \neq S_2(v)$, $S_1(v) \neq S_2(w)$ (see Fig. 2*c*), then $Z^{(1)} = v$, $e(Z^{(1)}) = S_2(v)$, $Z^{(2)} = w$, and $e(Z^{(2)}) = S_2(w)$.

The block diagram of the algorithm for the correction of single and double errors is represented by Fig. 3.

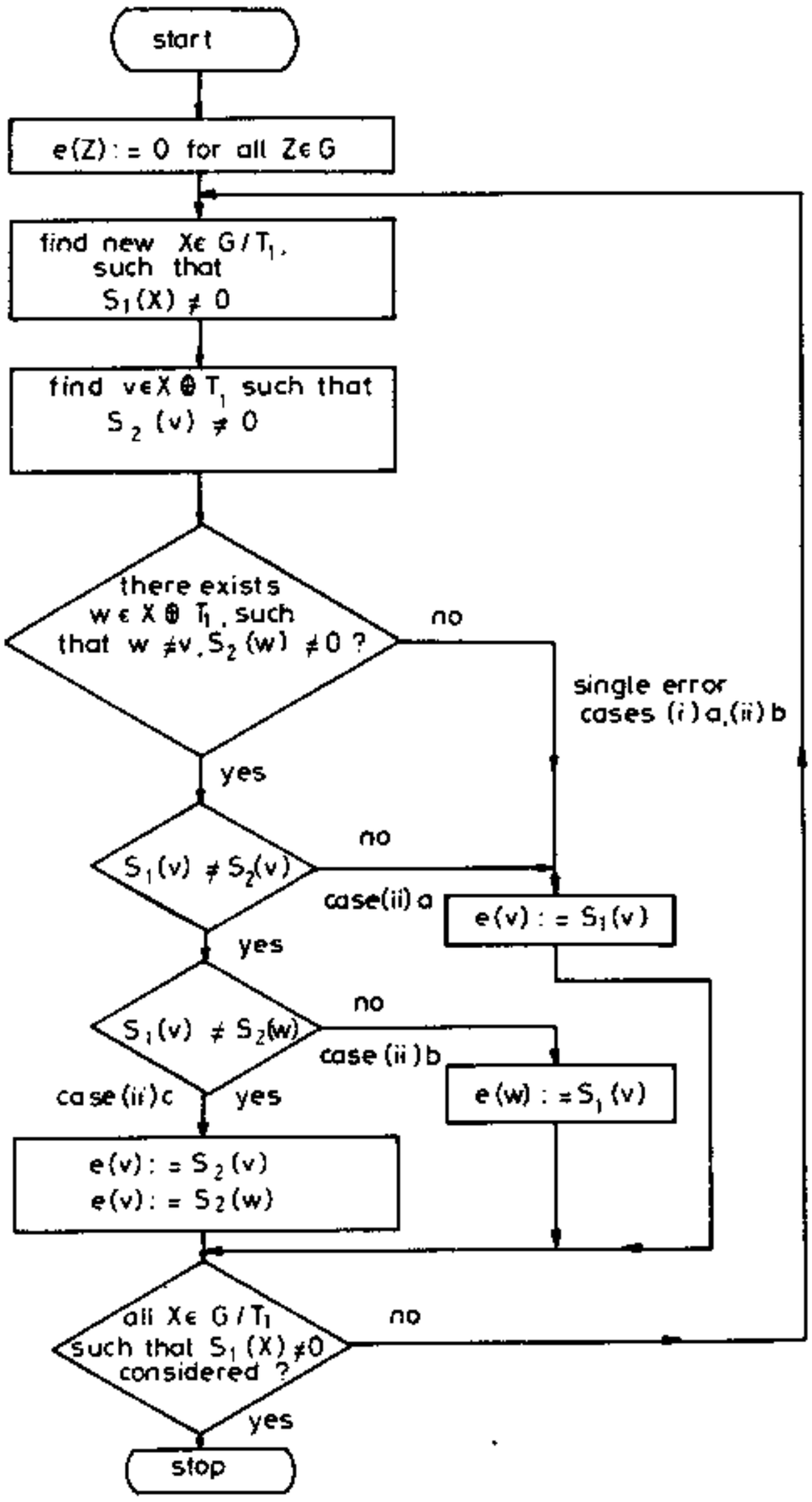

Fig. 3 *Block diagram of the algorithm for the correction of single and double errors by two orthogonal equality checks*

2.3 Complexity of error correction

We shall now estimate the time for the correction of errors in a ROM by the number N of read operations.

(i) If there is no error, then

$$N = N_0 = |G| = 2^n \tag{9}$$

(ii) If there is a single error then

$$N = N_1 \leqslant 2^n + |T_1||T_2| \leqslant 2 \times 2^n \tag{10}$$

(iii) If there is a double error then

$$N = N_2 \leqslant 2^n + 2|T_1||T_2| \leqslant 3 \times 2^n \tag{11}$$

Denote P_i the probability of an error with the multiplicity i ($P_0 + P_1 + P_2 = 1$, we suppose that for any $i > 2$, $P_i = 0$). Thus, we have for the expected number $\bar{N}$ of read operations

$$\bar{N} = P_0 N_0 + P_1 N_1 + P_2 N_2 \leqslant 2^n + |T_1||T_2|(P_1 + 2P_2) \tag{12}$$

As a weaker upper bounder for $\bar{N}$ one can use the formula

$$\bar{N} \leqslant 2^n(1 + P_1 + 2P_2) \tag{13}$$

2.4 Error-correcting capability

Let us describe now the error-correcting capability of two orthogonal linear checks. First, we note that all single errors are corrected by the algorithm of Fig. 3. All double errors satisfying expr. 8 are also corrected by this algorithm. Denote by m the number of bits in a memory word. Then, for the big m, we have, from expr. 8, the following estimations on the probability $P^c(2)$ of the correction of double errors:

$$P^c(2) \geqslant 1 - |T_1||T_2|2^{-n-m} \tag{14}$$

In the case when a double error results in a distortion of data in two binary cells (bitwise errors) we have to replace 2^m by $2m$ in expr. 14. Expr. 14 illustrates the good error-correcting capability of two orthogonal checks. It also follows from expr. 14 that to maximise the error-correcting capability, we have to choose T_1 and T_2 for the given $f(Z)$ with the minimal $|T_1||T_2|$. This will also provide with the minimal complexity of the error-correcting procedure (see exprs. 9–12). Results of the computer experiments illustrating a high error correcting capability of this algorithm are given in Tables 1 and 2. Here, $P^c(l)$ is the percentage of errors with multiplicity l corrected by the algorithm. In these experiments, first we write in the RAM $f(Z)$ ($f(z)$ is the data written in a cell with the address Z), then randomly generate locations and magnitudes of errors, distort cor-

Table 1 : Error-correcting capability of two checks

	$P^c(l)$				
Function $f(Z) = f(Z_1, Z_2)$ / l	$Z_1 + Z_2$	$Z_1 - Z_2$	$Z_1 Z_2$	$Z - 1$	$Z^2 - Z - 1$
2	100	100	100	100	100
3	99.90	99.90	97.00	99.93	58.06
4	99.90	99.90	88.86	99.90	23.46
5	99.33	99.26	76.43	99.56	6.36

Results of computer experiments on error-correcting capability of two orthogonal equality checks ($n = m = 8$; for every l and every $f(Z)$ 3000 experiments have been made to estimate $Pc(l)$).

Table 2: Bitwise errors

	$P^c(l)$				
Function $f(Z) = f(Z_1, Z_2)$ / l	$Z_1 + Z_2$	$Z_1 - Z_2$	$Z_1 Z_2$	$Z - 1$	$Z^2 - Z - 1$
2	99.93	99.83	99.13	99.83	98.50
3	99.70	99.53	94.93	99.46	54.63
4	99.43	99.13	84.96	99.08	20.96
5	98.70	98.60	71.43	98.43	6.26

Results of computer experiments on error-correcting capability of two orthogonal equality checks ($n = m = 8$; for every l and every $f(Z)$ 3000 experiments have been made to estimate $Pc(l)$)

Table 3: Error-locating capabilities of two orthogonal inequality checks

Function $f(y)$	m	$m - \log_2\lvert T_1\rvert - \log_2\lvert T_2\rvert$	$P^L(2)$	$P^L(3)$	$P^L(4)$	$P^L(5)$
$\frac{1}{\sqrt{y}}\sin\left(\frac{\pi}{2}\sqrt{y}\right)$	12	9	99.9	99.6	99.3	99.0
$\cos\left(\frac{\pi}{4}\sqrt{y}\right)$	12	4	97.0	84.0	72.5	56.0
$\left(\frac{\pi}{2} - \sin^{-1} y\right)\sqrt{1-y}$	15	3	94.4	66.8	42.3	22.0

Results of computer experiments on error-locating capabilities of two orthogonal inequality checks (an error with multiplicity l distorts data in l memory cells; $n = m$; $y = 2^{-m}Z$, $Z \in \{0, 1, \ldots, 2^m - 1\}$) for every l and every $f(y)$ 3000 experiments have been made to estimate $P^L(l)$)

Experimental estimations on probabilities $p^L(l)$ of location for errors with multiplicity l by the algorithm of Fig. 4 are given in Table 3.

In this Table, $y = Z2^{-m}$, $n = m$, $Z \in \{0, 1, \ldots, 2^m - 1\}$ and 3×10^3 experiments with randomly generated locations and magnitudes of errors have been made to compute $p^L(l)$ for every $l = 2, 3, 4, 5$ and every $f(Z)$.

4 Correction of single errors and detection of multiple errors in ROM by the Rademacher transform

4.1 Rademacher transform

The techniques described in Sections 2 and 3 are efficient in the case when the function stored in the ROM is analytical. We shall describe, in this Section, another approach which can be used for a ROM containing random data. It will provide us with the simple method for the correction of all single errors and the simultaneous detection of all double and triple errors. This will require, for a ROM with n address bits, $n + 1$ additional memory cells and 2^n read operations.

We shall define the Rademacher transform $(F_0, F_1, \ldots, F_n)$ for the function $f(Z)(Z = (Z_1, \ldots, Z_n), Z_i \in \{0, 1\})$ as

$$F_i = \sum_{Z=0}^{2^n-1} f(Z)R_i(Z) \quad i = 0, \ldots, n \tag{20}$$

where $R_0(Z) = 1$, $R_i(Z) = (-1)^{Z_i}$ are the Rademacher functions which are the subset of the Walsh functions [7]. (The Rademacher-Walsh functions are widely used in logical design, error-correcting codes and fault-tolerant computing [1, 4, 5, 7–9, 16].)

4.2 Error correcting capability

We shall use the Rademacher transform (expr. 20) for the correction of single errors and detection of multiple errors. As a result of the error $e(Z)$, our function $f(Z)$ is replaced by $\tilde{f}(Z) = f(Z) + e(Z)$; in this case, by the syndrome $S = (S_0, \ldots, S_n)$ of an error $e(Z)$, we mean

$$S_i = \sum_{Z=0}^{2^n-1} (f(Z) + e(Z))R_i(Z) - F_i$$

$$= \sum_{Z=0}^{2^n-1} e(Z)R_i(Z) \quad i = 0, \ldots, n \tag{21}$$

By error correction we mean the computation of an error function $e(Z)$ by the syndrome $S = (S_0, \ldots, S_n)$.

We note that for the single error $e(Z) = \delta_{Z, Z^{(1)}}e(Z^{(1)})(Z^{(1)} = (Z_1^{(1)}, \ldots, Z_n^{(1)}), Z_i \in \{0, 1\})$

$$S_0 = e(Z^{(1)}),\ S_i = e(Z^{(1)})R_i(Z^{(1)}) = e(Z^{(1)})(-1)^{Z_i^{(1)}} \tag{22}$$

Thus, any two single errors have different syndromes, and all single errors may be corrected by the analysis of the syndrome vector $S = (S_0, \ldots, S_n)$. For the computation of a syndrome we have to compute the Rademacher transform (expr. 20) of a function $f(Z) + e(Z)$ stored in the ROM. It is easy to check that by the Rademacher transform all single, double and triple errors can be detected, and there exist errors with multiplicity equal to four which cannot be detected. For example, for the error

$$e(Z) = \delta_{Z, 0^n} + \delta_{Z, 0^{n-2}12} - \delta_{Z, 0^{n-1}1} - \delta_{Z, 0^{n-2}10}$$

$$(n \geqslant 3)\left(0^i = \frac{0 \cdots 0}{i}\right)$$

we have $(S_0, \ldots, S_n) = (0, \ldots, 0)$, and this error cannot be detected.

We note also that for any multiplicity $l \geqslant 4$ of errors, we have, for the probability $p^d(l)$ of error-detection by the Rademacher transform,

$$p^d(l) \geqslant 1 - (2^m - 1)^{-1} \tag{23}$$

where m is the number of bits in a memory word.

4.3 Correction and detection of errors

The software implementation of the correction of single errors and the detection of multiple errors by the Rademacher transform is illustrated by the block diagram of Fig. 5. This algorithm requires 2^n read operations and $n + 1$ redundant memory cells $A_0, A_1, \ldots, A_n$. (All errors in

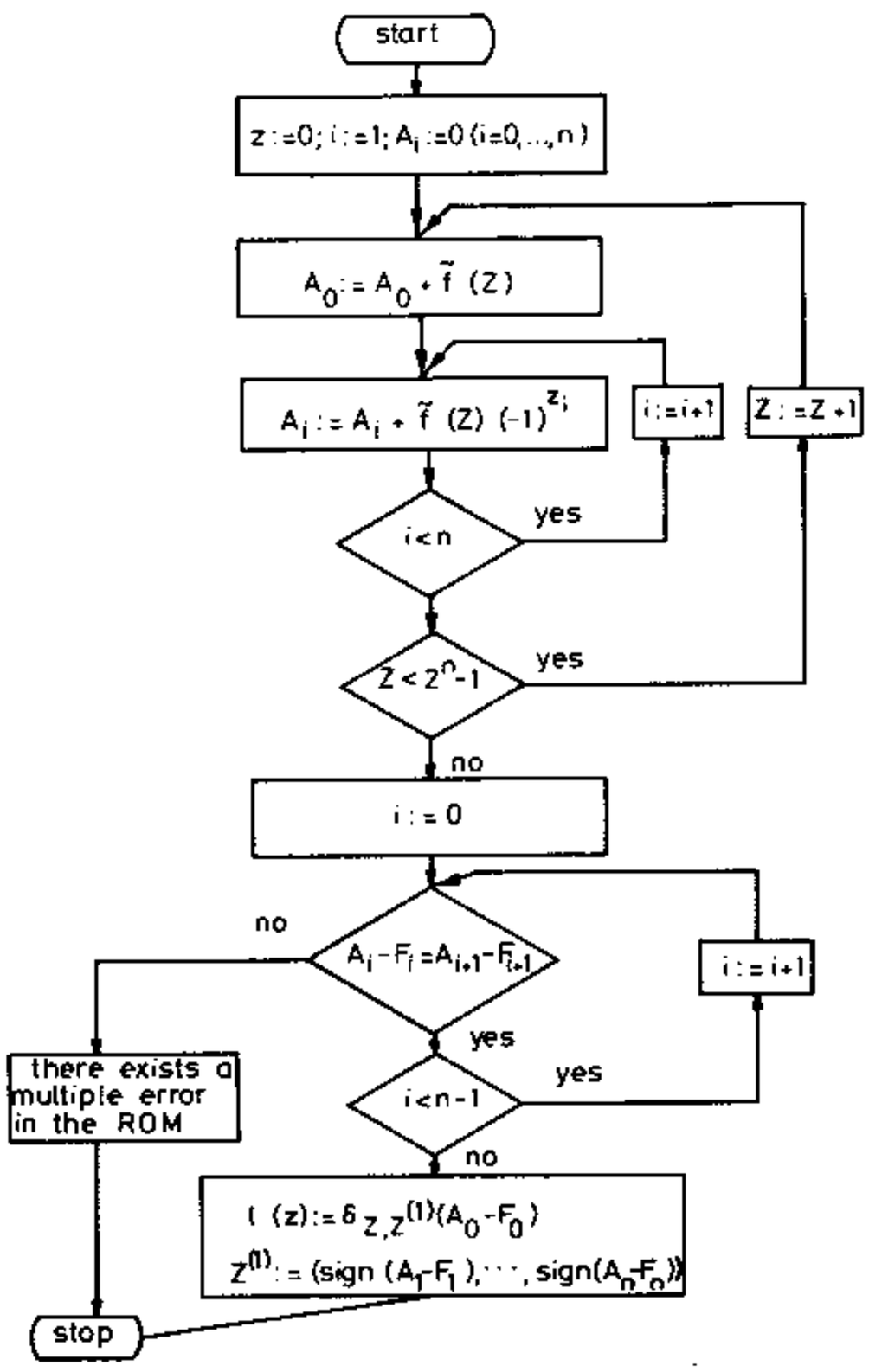

Fig. 5 *Block diagram of the algorithm for the correction of single errors and the detection of multiple errors by the Rademacher transform;* $(\tilde{f}(z) = f(z) + e(z))$

$$\text{sign } X = \begin{cases} 0, & X \geqslant 0 \\ 1, & X < 0 \end{cases}$$

The algorithm for computation of the Rademacher transform based on exprs. 24 and 25 requires $3(2^n - 1) - n$ additions and subtractions. This algorithm is very similar to the fast Walsh transform algorithm [7].

We note again that the computation of the Rademacher transform has to be executed only once for the given function $f(Z)$, and these computations may be considered as a part of the test-generation procedure.

5 Detection and location of errors in a RAM

5.1 Testing of a RAM

All three previous techniques based on linear equality checks, inequality checks and the Rademacher transform can be used for error detection, location or correction in a RAM.

To implement this, we first have to choose the function $f(Z)$ which we shall write in the RAM. Since the expected testing time $\bar{N}$ for linear equality or inequality checks increases linearly with the increase of $|T_1||T_2|$ (see exprs. 12, 13 or 19), it is expedient for these checks to choose $f(Z)$ in such a way that $|T_1||T_2|$ will be minimal. This will provide also the maximal error-correcting (locating) capability of checks (see expr. 14).

After choosing $f(Z)$, in a case of equality or inequality checks, we have to construct the corresponding checks for this $f(Z)$ using the techniques from References 1–4 or 5.

Next, for any $Z = 0, 1, \ldots, 2^n - 1$ (n is the number of bits in an address), we write in a cell with an address Z the value $f(Z)$. After this we scan out the memory and compute a syndrome (see exprs. 4, 17 or 21). Then we detect, locate or correct errors by the analysis of a computed syndrome using the algorithms described in previous Sections. If the implementation of a decoding algorithm analysing a computed syndrome requires N read operations, then the total number of read and write operations for a testing of a RAM is $N + 2^n$.

5.2 Complexity and error detecting capability

Comparing the decoding algorithms represented in Figs. 3–5, we can see that all three algorithms have about the same running time, but the linear-equality-checks algorithm (Fig. 3) have the maximal error-correcting capability. Thus, it is expedient to use orthogonal linear equality checks for RAM testing.

The best choice of $f(Z)$ to minimise $|T_1||T_2|$ for equality checks is $f(Z) = C$, where C is a constant for all $Z \in G$, but in this case we cannot detect, locate or correct errors in the address decoder or bridgings between cells.

The next best choice to provide error correction for decoding errors is to choose $f(Z)$ as a linear function, for example,

$$f(Z) = f_1(Z) = 2^{m-n}\left(\frac{2^n - 1}{2} - Z\right)$$

where m is the number of bits in a memory word, $m \geqslant n$. In this case, we minimise $|T_1||T_2|$; thus, minimising the number of read operations (see exprs. 10 and 12) and maximising the error-correcting capability of equality checks (see expr. 14).

To correct all single stuck-at errors in memory cells we have also to repeat our procedure for

$$f(Z) = f_2(Z) = 2^m - 1 - 2^{m-n}\left(\frac{2^n - 1}{2} - Z\right)$$

(for any Z the binary representation of $f_2(Z)$ is the componentwise negation of the corresponding representation for $f_1(Z)$). For

$$f_1(Z) = 2^{m-n}\left(\frac{2^n - 1}{2} - Z\right)$$

it is expedient to use the following orthogonal equality checks:

$$\left.\begin{aligned} &f_1(Z) + f_1(Z \oplus 1^n) = 0 \quad 1^n = \underbrace{1 \cdots 1}_{n} \\ &f_1(Z) + f_1(Z \oplus 101^{n-2}) + f_1(Z \oplus 011^{n-2}) \\ &\qquad + f_1(Z \oplus 110^{n-2}) = 0 \end{aligned}\right\} \tag{26}$$

For $f_2(Z) = 2^m - 1 - f_1(Z)$, we can use the same T_1 and T_2 as for $f_1(Z)$, and we have the following two checks:

$$\left.\begin{aligned} &f_2(Z) + f_2(Z \oplus 1^n) = 2(2^m - 1) \\ &f_2(Z) + f_2(Z \oplus 101^{n-2}) + f_2(Z \oplus 011^{n-2}) \\ &\qquad + f_2(Z \oplus 110^{n-2}) = 4(2^{m-1}) \end{aligned}\right\} \tag{27}$$

For the expected number $\bar{N}$ of read and write operations for the correction of single and double errors in a RAM, we have, from exprs. 12, 26 and 27,

$$\bar{N} \leqslant 2(2^{n+1} + 8(P_1 + 2P_2)) \sim 2^{n+2} \tag{28}$$

and, for the probability $P^c(2)$ of the correction of double errors, we have, from expr. 14,

$$P^c(2) \geqslant 1 - 2^{-n-m+3} \tag{29}$$

If we are interested only in the error detection, but not the error correction, we again can use the same two orthogonal checks for $f_1(Z)$ and $f_2(Z)$. In this case we have to check whether $S_1(Z) \neq 0$ for at least one $Z \in G/T_1$ or $S_2(Z) \neq 0$ for at least one $Z \in G/T_2$. This will require $N = 2^{n+2}$ read and write operations. By doing this we shall detect all single and double errors, and for a probability $P^d(l)$ of the detection of an error with the multiplicity $l \geqslant 3$, we have

$$p^d(l) \geqslant 1 - (2^m - 1)^{-2} \tag{30}$$

For bitwise errors, all single and double errors are detected and for $l \geqslant 3$, we have

$$p^d(l) \geqslant 1 - (2m - 1)^{-2} \tag{31}$$

We note that many stuck-at faults in cells of a RAM and at the outputs of the address decoder and bridgings between output lines of the address decoder may result in unidirectional errors [18] such that $e(Z) \geqslant 0$ for all Z (or $e(Z) \leqslant 0$ for all Z). For example, all and (or 'or') bridgings between rows in a RAM address decoders and faults that affect power supply or read/write circuits result in unidirectional errors [18, 20]. (Each row in a RAM corresponds to a cell storing a value of $f(Z)$.) All unidirectional errors with any multiplicity will be detected. Since any bridgings between two rows results in a distortion of at most two values of $f(Z)$, it follows from expr. 29 that almost all these bridgings will be corrected. We note also that many errors in the address decoder may be detected and/or corrected by two orthogonal checks. We shall say that there exists an error with the multiplicity l in the address decoder if there exists $Z^{(1)}, \ldots, Z^{(l)} \in \{0, 1\}^n$ such that the output of the RAM is $f(Y^{(i)})$ for the address $Z^{(i)}$, $Y^{(i)} \neq Z^{(i)} (i = 1, \ldots, l)$. In this case,

$$e(Z) = \sum_{i=1}^{l} \delta_{Z, Z^{(i)}}(f(Y^{(i)}) - f(Z^{(i)})) \tag{32}$$

9 Appendixes

9.1 Optimal equality checks for some basic hardware components

N	Device	Function f implemented by a device	T	C
1	And gate	$f(Z_1, \ldots, Z_n) = \prod_{i=1}^{n} Z_i$	$\{0, 1\}^n$	1
2	Or gate	$f(Z_1, \ldots, Z_n) = \bigvee_{i=1}^{n} Z_i$	$\{0, 1\}^n$	$2^n - 1$
3	Parity checker	$f(Z_1, \ldots, Z_n) = \bigoplus_{i=1}^{n} Z_i$	$\{0^n, 0^{n-1}1\}$	1
4	Majority voter	$f(Z_1, \ldots, Z_n) = \begin{cases} 0, & \|Z\| \leqslant 0.5(n-1) \\ 1, & \|Z\| > 0.5(n-1) \end{cases}$ $\left(n = 2S + 1, \ \|Z\| = \sum_i Z_i\right)$	$\{0^n, 1^n\}$	1
5	Voter with the threshold equal to 2 $(n = 2^\alpha - 1)$	$f(Z_1, \ldots, Z_n) = \begin{cases} 0, & \|Z\| \leqslant 1 \\ 1, & \|Z\| > 1 \end{cases}$	$(2^\alpha - 1, 2^\alpha - \alpha - 1)$ Hamming code	$2^\alpha - \alpha - 2$
6	PLA (product terms are orthogonal; AND gates have at most d inputs)	$f(Z_1, \ldots, Z_n)$	$V^\perp(n, d+1)$	$\sum_Z f(Z) \times \lvert V^\perp(n, d+1)\rvert^{-1}$
		$= \bigvee_i f_i(Z_1, \ldots, Z_n)$, where $f_i(Z_1, \ldots, Z_n)$ is a product of at most d literals, and for any $Z = (Z_1, \ldots, Z_n)$, $f_i(Z)f_j(Z) = 0$	$V^\perp(n, d+1)$ is a dual code to a maximal code with the Hamming distance $d + 1$.	
7	Autonomous linear feedback register [19] with initial state $(Z_1, \ldots, Z_n)$	$f_t(Z_1, \ldots, Z_n) = A^t \begin{pmatrix} Z_1 \\ Z_n \end{pmatrix}$; $A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ a_1 & a_2 & a_3 & \cdots & a_{n-1} & a_n \end{pmatrix}$ $a_i \in \{0, 1\}$; A^t is a tth degree of A; all computations are in $GF(2)$	$\{0^n, A^{-t}1^n\}$ (A^{-t} is an inverse over $GF(2)$ of A^t)	$2^n - 1$
8	n-bit shifter	$f(C_l, C_r, Z_1, \ldots, Z_n) =$ $\begin{cases} (Z_1, Z_2, \ldots, Z_{n-1}, Z_n); C_l = C_r = 0 \\ (Z_2, Z_3, \ldots, Z_n, 0); C_l = 1, C_r = 0 \\ (0, Z_1, \ldots, Z_{n-2}, Z_{n-1}); C_l = 0, C_r = 1; \\ (0, 0, \ldots, 0, 0); C_l = C_r = 1 \end{cases}$	$\{000^n, 001^n,$ $100^n, 101^n$ $010^n, 011^n,$ $110^n, 111^n\}$	$5 \times 2^{n-1} - 4$
9	m-bit counter	$f(Z_1, \ldots, Z_n) = \sum_{i=1}^{n} Z_i; n \leqslant m$	$\{0^n, 1^n\}$	n
10	m-bit up and down counter	$f(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$ $= \sum_{i=1}^{n} X_i - \sum_{i=1}^{n} Y_i; n \leqslant m$	$\{0^n, 1^n\}$	0
11	n by 1 multiplexer	$f(S_0, S_1, \ldots, S_\alpha, Z_1, \ldots, Z_n)$ $= Z_i$ iff $\sum_{j=0}^{\alpha-1} S_j 2^{\alpha-1-j} = i; n = 2^\alpha$	$\{0^n 0^\alpha, 0^\alpha 1^n\}$	1
12	Adder	$f(X, Y) = X + Y$ $X, Y \in \{0, 1\}^n$	$\{0^{2n}, 1^{2n}\}$	$2(2^n - 1)$
13	Subtractor	$f(X, Y) = X - Y$ $X, Y \in \{0, 1\}^n$	$\{0^{2n}, 1^{2n}\}$	0
14	Multiplier	$f(X, Y) = XY$; $X, Y \in \{0, 1\}^n$	$\{0^{2n}, 0^n1^n, 1^n0^n, 1^{2n}\}$	$(2^n - 1)^2$

9.3 Orthogonal checks for some numerical functions ($|T_1 < |T_2|$)

N	Name	Function	T_1	C_1	T_2	C_2	Comments
1	Multiplication	$f(X, Y) = XY$ $X, Y \in \{0, 1, \ldots, 2^n - 1\}$	$\{0^n, 1^n\}^2$ $= \{0^{2n}, 0^n1^n, 1^n0^n, 1^{2n}\}$	$(2^n - 1)^2$	$\{0^n, 101^{n-2}, 011^{n-2}, 110^{n-2}\}^2$	$4(2^n - 1)^2$	$A^2 = \{a_1a_2 \mid a_1, a_2 \in A\}$
2	Conversion from the binary to the inverted code [15]	$f(Z) = Z \oplus \text{shr } Z$ $\text{shr } Z = (0, Z_1, \ldots, Z_{n-1})$ $Z = (Z_1, \ldots, Z_n)$ $Z_i \in \{0, 1\}$	$\{0^n, 1010\ldots\}$	$2^n - 1$	$\{0^n, 10101\ldots, 0101, \ldots, 10^{n-1}\}$	$2(2^n - 1)$	
3	Conversion from the inverted to the binary code [15]	$f(Z) = \bigoplus_{i=0}^{n-1} \text{shr}^i Z$ $\text{shr}^0 Z = Z, \text{shr}^i Z = \text{shr}^{i-1} Z)$ $Z = (Z_1, \ldots, Z_n), Z_i \in \{0, 1\}$	$\{0^n, 10^{n-1}\}$	$2^n - 1$	$\{0^n, 1110^{n-3}, 010^{n-2}, 101^{n-3}\}$	$2(2^n - 1)$	
4	Conversion from the 2421 BCD code to the binary code [15]	$f(Z_1, \ldots, Z_n) =$ $\sum_{i=0}^{S-1} (Z_{n-4i} + 2Z_{n-4i-1} +$ $+4Z_{n-4i-2} + 2Z_{n-4i-3})10^i$ $n = 4S, Z_i \in \{0, 1\}$ $S > 2.$	$\{0^n, 1^n\}$	$10^S - 1$	$\{0^n, 1^40^41^{n-8}, 0^41^41^{n-8}, 1^80^{n-8}\}$	$2(10^S - 1)$	
5	Conversion from the excess 3 BCD code to the binary code [15]	$f(Z_1, \ldots, Z_n) =$ $\sum_{i=0}^{S-1} (Z_{n-4i} + 2Z_{n-4i-1} +$ $4Z_{n-4i-2} + 8Z_{n-4i-3} - 3)10^i$ $n = 4S, S > 2$	$\{0^n, 1^n\}$	$10^S - 1$	$\{0^n, 1^40^41^{n-8}, 0^41^41^{n-8}, 1^80^{n-8}\}$	$2(10^S - 1)$	
6	Linear function	$f(Z) = az + b$ $Z \in \{0, 1, \ldots, 2^n - 1\}$	$\{0^n, 1^n\}$	$a(2^n - 1) + 2b$	$\{0^n, 101^{n-2}, 011^{n-2}, 110^{n-2}\}$	$2a(2^n - 1) + 4b$	
7	Scalar product	$f(X^{(1)}, \ldots, X^{(m)}, Y^{(1)}, \ldots, Y^{(m)}) = \sum_{i=1}^{m} X^{(i)}Y^{(i)}$	$\{0^{nm}, 1^{nm}\}^2$	$m(2^n - 1)^2$	$\{0^{nm} \cup \{101^{n-2}\}^m \cup \{011^{n-2}\}^m \cup \{110^{n-2}\}^m\}^2$	$4m(2^n - 1)^2$	$A^m = \{a_1, \ldots, a_m \mid a_i \in A\}$
8	Rademacher function (see Section 4.1)	$f_i(Z) = (-1)^{Z_i}$ $Z = (Z_1, \ldots, Z_n)$	$\{0^n, 0^{i-1}10^{n-i}\}$	0	$\{0^n, 10^{i-2}10^{n-i}\}$	0	
9	Walsh function [7]	$f_{i_1, i_2, \ldots, i_s}(Z) = Z_{i_1} + Z_{i_2} + Z_{i_s}$ $(-1); 1 \leqslant S \leqslant n$	$\{0^n, 0^{i_1-1}10^{n-i_1}\}$	0	$\{0^n, 0^{i_2}10^{n-i_2}\}$	0	
10	Odd polynomial	$f(Z) = \sum_{i=1}^{S} a_i(0.5(2^n - 1) - Z)^{2i-1}$ $Z \in \{0, 1, \ldots, 2^n - 1\}.$	$\{0^n, 1^n\}$	0	linear span of rows of $H = (I_{d+1} \vdots J_{n-d-1}$ where $d = 2S - 1$	0	I_{d+1} is $(d + 1) \times (d + 1)$ identity matrix; J_{n-d-1} is $(d + 1) \times (n - d - 1)$ matrix of all ones
11	Polynomial of degree $d(d < n)$	$f(Z) = \sum_{i=0}^{d} a_i Z^i$ $Z \in \{0, 1, \ldots, 2^n - 1\}$	$V_1^{\perp}(n, d + 1)$	$2^n \sum_Z f(Z) \times \lvert V^{\perp}(n, d + 1)\rvert^{-1}$	$V_2^{\perp}(n, d + 1)$	$2^n \sum_Z f(Z) \times \lvert V_2^{\perp}(n, d + 1)\rvert^{-1}$	$V_1^{\perp}(n, d + 1)$ and $V_2^{\perp}(n, d + 1)$ are dual to codes $V_1(n, d + 1)$ and $V_2(n, d + 1)$ with distances at least $d + 1$ $V_1^{\perp}(n, d + 1) \cap V_2^{\perp}(n, d + 1) = 0^n$