Cellular automata based pattern generator for testing RAM

D. Roy Chowdhury
I. Sen Gupta
P. Pal Chaudhuri

Indexing terms: Cellular automata, Characteristic matrix, Pattern sensitive faults, RAM testing

Abstract: The paper presents a new strategy for parallel testing of RAM. A cellular automata (CA) based test pattern generator for detecting pattern sensitive faults (PSFs) in random access memories is also reported. An 8-cell one dimensional three neighbourhood CA has been extended to the five-neighbourhood case preserving the criterion of local connections. By changing the neighbourhood relation, all the 64 patterns for detecting five-neighbourhood PSFs can be generated by loading two seeds only. The method can be easily extended for detecting PSFs of any neighbourhood.

1 Introduction

Recent development and commercial fabrication of high density random access memory (RAM) integrated circuits have been greatly motivated because of the growing needs of RAMs in computer systems. The number of memory cells on an integrated circuit has quadrupled every two to four years, from the initial 1 Kbit RAM to the present 16 Mbit RAM. Owing to this increased density, the testing of these chips have become costly as well as time consuming. A RAM chip mainly consists of an array of memory cells, an address decoder, memory address register (MAR), memory data register (MDR) and read/write logic (sense amplifiers, write drivers, etc.) (Fig. 1). A wide variety of physical faults can occur in any

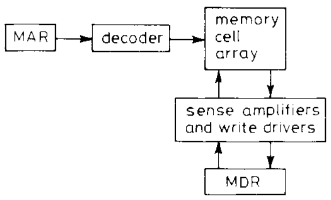


Fig. 1 *Organisation of a RAM chip*

of these subsystems. The test procedures for the detection of these faults can be classified as DC parametric testing, AC parametric testing, and functional testing [2]. In this paper, we restrict out attention to functional testing of the memory aray only.

Paper 8830E (C2), first received 4th September 1990 and in revised form 6th January 1992

The authors are with the Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur-721302, India

For an n-cell memory array, the complexity of the functional testing of all possible faults is 2^n, as every cell is to be checked for 0 and 1 for all possible states of remaining cells. Since this is practically not feasible, some restricted fault models are usually considered. Three fault models for RAMs are most widely used. These are:

(*a*) *The 'stuck-at fault' model:* The logic values on some signal lines or in one or more memory cells are permanently stuck at 1 or 0. These errors can occur in the memory data register, memory address register, address decoder, the memory cell array and the read/write logic. Several works have been reported on the detection of memory stuck-at faults, like the MSCAN method [2], the ATS procedure [11], the modified ATS procedure [14], etc.

(*b*) *The 'coupling fault' model:* A pair of memory cells i and j are said to be coupled if a transition from 1-to-0 or 0-to-1 in one cell (say i) of the pair changes the state of the other cell (say j). Various test procedures such as column bars, marching 1s and 0s and galloping 1s and 0s (GALPAT) have been proposed [2]. These algorithms have complexities ranging from $O(n)$ to $O(n^2)$ depending on the fault coverage. Later, more efficient test procedures having complexity $O(n)$ with more comprehensive fault coverage have been reported [21, 24].

(*c*) *The 'pattern sensitive fault' (PSF) model:* This category of faults refer to the case that a memory cell do not function properly whenever a particular pattern is stored in other cells, or whenever another memory cell changes state.

General PSFs are intractable in practice [8]. However, some restricted (and also realistic) PSF models allow for the generation of efficient test sequences [9, 23]. Even though these algorithms have linear complexity, they have an $O(k2^k)$ constant multiplier, where k is the neighbourhood size. Subsequent works have been reported which improve upon this complexity by using parallel testing strategies [12, 13].

Here, in this paper, the key ideas of parallel testing strategies are further extended and a new scheme of cellular automata (CA) based built-in self-test design of RAM is proposed. Since a test set for detecting PSFs automatically detects stuck-at and coupling faults, so we concentrate on the detection of PSFs only. The key concepts introduced in the present work are

(i) A new CA-based test pattern generator covering all PSFs is introduced.

(ii) A simple hardware is proposed at the output of the test pattern generator in order to reduce the number of test patterns to be written on RAM cells.

(iii) Cells on multiple numbers of rows are accessed simultaneously by using a simple current sharing circuit in conjunction with some additional control for test mode.

This method has been found to be more efficient and elegant with respect to modularity and cascadibility. Moreover, the time complexity is comparable with existing methods.

2 CA preliminaries

Traditionally, cellular automata (CA) have been widely used as a method of modelling *self-organising* systems like biological self-reproduction, fully discrete dynamic systems etc. [4]. Recent renewed interests in CA have developed mainly due to its simple computation theory and attractive parallel processing capabilities.

In Reference 26 the state transition behaviour of a wide variety of CAs has been studied, and the CAs have been classified into four broad groups:

Class 1: CAs which evolve to a homogeneous final global state.

Class 2: CAs in which each state lie in some cycle (periodic behaviour).

Class 3: CAs exhibiting chaotic or pseudo-random behaviour.

Class 4: CAs having complicated localised and propagating structures.

A CA basically consists of a discrete lattice of sites (cells) which can assume values 0 or 1. The cells evolve in discrete time steps according to some deterministic rule that depends only on local neighbourhood. Each cell consists of a storage element (flip-flop) and a combinational logic realising the next state function, as shown in Fig. 2.

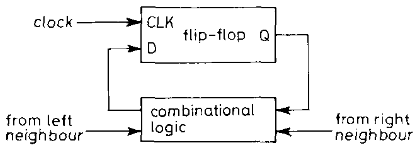


Fig. 2 *A typical CA cell*

Consider an 1-D CA where all the cells are arranged in a linear array. From the viewpoint of ease of implementation, the next state of a particular cell is assumed to depend on itself and on its two neighbours (3-N dependency).

Mathematically, the state x_i of the ith cell of a 1-D 3-N CA at time $t + 1$ is given by

$$x_i^{t+1} = f(x_i^t, x_{i-1}^t, x_{i+1}^t)$$

where x_i^t denotes the state of ith cell at time t, and f is called the *rule of the automata*. f is a boolean function of three variables, so there are 256 possible rules.

Example 1: Consider a 3-neighbourhood CA with the next-state function for cell i represented as

Neighbourhood state:	111	110	101	100	011	010	001	000
Next state:	1	0	0	1	1	0	0	1

On the top row all 8 possible values of neighbourhood are given, and below each is given the value achieved by the central cell on the next time step according to the rule. The bottom row taken as a binary number and converted into its decimal equivalent, is the rule, 153.

Alternatively, the next-state transition function (rule) of a 1-D CA cell can be expressed algebraically. In this notation, x_0, x_1 and x_{-1}, respectively, denote the state of a particular cell, and those of its right and left neighbours. Then the next state of the cell can be expressed as a Boolean function f:

$$x_0^{t+1} = f(x_0^t, x_{-1}^t, x_1^t)$$

Definition 1: If the next state function f is an EXOR function, then the rule is said to be linear.

Definition 2: If all the cells obey the same rule, then the CA is said to be uniform; otherwise it is hybrid.

Definition 3: If the end cells of the CA are connected to constant logic 0, then the CA is said to have 'null boundary', but if the leftmost and rightmost end cells are connected in a loop then it is 'periodic boundary'.

Efficient characterisation of 1-D CA based on matrix algebra has been proposed in Reference 6, where the global state of the CA is generated by a linear operator. For an n-cell 1-D CA the linear operator is an $n \times n$ binary matrix whose ith row corresponds to the neighbourhood relation of cell i, and the operation is a simple modulo 2 matrix multiplication. The operator is termed as the *characteristic matrix* of the CA, and is denoted by T.

If $f_t(x)$ represents the state of the automata at time t, then the next state, i.e. the state at time $(t + 1)$ is

$$f_{t+1}(x) = T \times f_t(x)$$

In general,

$$f_{t+p}(x) = T^p \times f_t(x)$$

Example 2: The four cell hybrid null boundary 1-D CA with the rule (90, 150, 150, 90) is shown in Fig. 3.

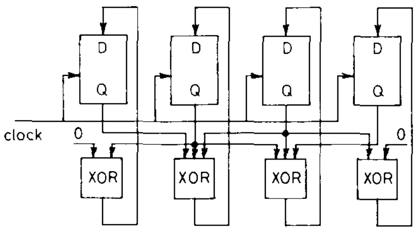


Fig. 3 *A one-dimensional CA*

Here,

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

If the present state of the CA is $f_t(x) = [1\ 0\ 1\ 0]^t$ then the next state is

$$f_{t+1}(x) = T \times f_t(x)$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

In this paper, the concepts developed in Reference 6 for the 3-N case have been generalised to the extended neighbourhood case. The dynamic behaviour of such an extended neighbourhood CA has been investigated to generate a particular class of unit distance code, which has been applied to detect *pattern sensitive faults* (PSF) in random-access memories (RAM).

3 Characterisation of extended neighbourhood CA having dynamic behaviour

3.1 Extended neighbourhood

One of the limitations of a 1-D CA employing a 3-N function is its relatively limited computational ability. If the number of neighbourhood cells on which the next state of a cell can depend is increased, the number of possible rules or functions realisable by the CA also are enhanced.

Suppose that we want to increase the neighbourhood size of a given 1-D CA. One way of achieving this is to reorganise the cells of the CA in a two-dimensional plane in such a way that the neighbourhood set of a particular cell is a superset of that of the corresponding cell in the original CA, although still preserving the local neighbourhood property. An efficient way to get the extra neighbours is to arrange (pleat) the 1-D array in the form of a spiral so that the CA becomes narrower in the linear direction and correspondingly gain in thickness. This is illustrated below with the help of an example.

Example 3: Consider a 1-D CA comprising of 9 cells as shown in Fig. 4*a*, where each cell excepting the boundary cells has 3-neighbourhood functions. Fig. 4*b* depicts another CA with an extended neighbourhood function.

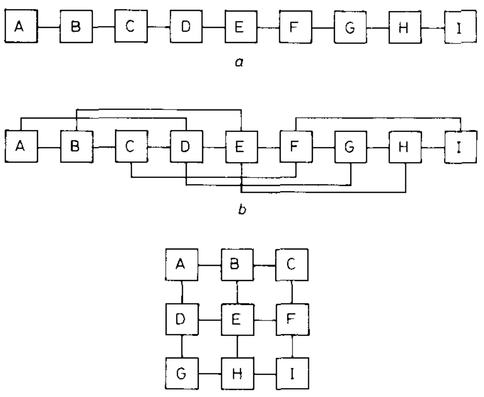


Fig. 4 *A one-dimensional CA comprising 9 cells*
a Original 3-neighbourhood CA
b Extended neighbourhood CA
c Physical cell layout to implement *b*

We redraw this configuration as one in which the same cells are arranged into two bit planes (Fig. 4*c*). In this configuration, the neighbourhood of the cell E (here five, as compared to three earlier) are no more than one cell distance away from E, even though some are in the other bit plane.

3.2 Dynamic behaviour of characteristic matrix

The characteristic matrix T mainly contains two sets of functional information: (i) the neighbourhood dependence function, and (ii) the state transition function.

These functions are constant for a particular CA. So a CA can be characterised by a unique *characteristic matrix T*. Now, if one of the above two functions changes with time then the characteristic matrix will also change dynamically. By changing the neighbourhood relation of one or more cells the T matrix can be modified easily. A CA is said to have *dynamic behaviour* if it can be configured to have different T matrices at different instants of time.

3.2.1 Group properties of dynamic CA: If a CA under the transformation of the operation with T forms a cycle of length n, then

$$[f_{t+n}(x)] = [T]^n[f_t(x)] = [f_t(x)]$$

That is,

$$[T]^n = I$$

or,

$$\det [T] = 1.$$

A CA with such a property is called a *group CA*. Some results related to group CA have been proved in Reference 6 which are stated below without any proof.

(*a*) A CA is a group CA if $\det [T] = 1$.

(*b*) A group CA has cycle lengths of p (or factors of p) with a nonzero starting state if and only if $\det [T^p + I] = 0$.

We now consider a CA whose characteristic matrix T changes dynamically with time. Suppose that the CA alternately uses rule given by a characteristic matrix T_1 for some cycles of time, and that by matrix T_2 for other cycles of time. If both T_1 and T_2 obey the characteristics of group CA, then we can write,

$$T_1^n = I \quad \text{and} \quad T_2^m = I$$

for some m and n.

Definition 4: If T_i and T_j denote the characteristic matrices of a CA having dynamic behaviour, then the 'composite matrix' is defined as $T_{ij}^{xy} = T_i^x T_j^y$, where x and y are some integers.

Lemma 1: If n and m, respectively, denote the lengths of two group CAs whose characteristic matrices are T_1 and T_2, then their composition T_{12}^{ab} will also constitute a group CA, provided either (i) $a = n$, $b = 1$, or (ii) $a = 1$, $b = m$.

Proof: Consider the case $a = n$ and $b = 1$. From the group properties of CA, it follows that

$$T_1^n = I \quad \text{and} \quad T_2^m = I$$

If $f(x)$ denote the present state of the CA, then

$$T_1^n f(x) = f(x)$$

or

$$T_2^m T_1^n f(x) = f(x)$$

or

$$T_2^{m-1}(T_2(T_1^n f(x))) = f(x)$$

or

$$T_2^{m-1}(T_1^n T_2(T_1^n f(x))) = f(x)$$

or

$$T_2^{m-2} T_2 T_1^n T_2 T_1^n f(x) = f(x)$$

or

$$(T_2T_1^n)(T_2T_1^n)\cdots \text{ to } m \text{ terms } = f(x)$$

So the CA under the operation of dynamic T forms a cycle of length $(n + 1)m$, i.e. it constitutes a group CA of length $(n + 1)m$. Now all the subcycles of length $(n + 1)$ can be made distinct if we choose T_2 in such a way that the $(n + 1)$th state of each subcycle are distinct from the previous n states. Similarly, the proof for the condition $a = 1, b = m$ can be arrived at. *Q.E.D.*

4 Detection of PSFs in RAM

As discussed in Section 1, we shall restrict our attention to the detection of PSFs in RAM. Two commonly used restricted PSFs, shown in Fig. 5, are:

Type 1: 5-cell neighbourhood, i.e. every cell depends on itself and its four neighbours (top, bottom, left and right).

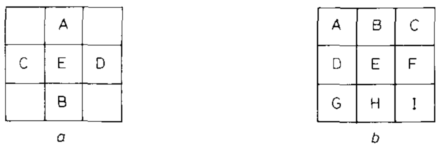


Fig. 5 *Neighbourhood dependencies*
a Type 1: 5-neighbourhood; E—base cell; $\langle ABCD \rangle$ — neighbourhood
b Type 2: 9-neighbourhood; E—base cell; $\langle ABCDFGHI \rangle$ — neighbourhood

Type 2: 9-cell neighbourhood, i.e. every cell depends on itself and its eight neighbours (top, bottom, left, right and the four diagonals).

In the present work we concentrate on the Type 1 category of faults.

There are two classes of PSFs caused by transition write operations affecting the base cell of a given neighbourhood. These are:

- *Active neighbourhood PSF (ANPSF):* The base cell changes with a transition write in one of its neighbours when the other neighbours contain a certain pattern of 0 and 1.
- *Passive neighbourhood PSF (PNPSF):* The contents of a cell cannot be changed from 0 to 1 (or 1 to 0) when a certain pattern exists in the neighbouring cells.

Two steps have to be executed to test the base cells for ANPSFs and PNPSFs:

Step 1: Sensitise every fault using the appropriate neighbourhood pattern.

Step 2: Read the state of the base cell after each sensitising pattern to detect any fault.

The number of patterns needed to sensitise all ANPSFs of a neighbourhood of size k is $(k - 1)2^k$ because each one of the $(k - 1)$ neighbours of the base cell must exercise both the 1-to-0 and 0-to-1 transitions, while the other $(k - 1)$ cells take all the possible binary values. Table 1 gives the sequence of patterns needed to sensitise all ANPSFs of size five. It can be observed that the test patterns form a unit distance code where two successive test vectors differ only in one bit position.

Many different procedures have been proposed [12, 13] for the efficient generation of the sensitising patterns. In the next section we present a CA based structure to generate the sensitising patterns in proper sequence.

Table 1: Test vector sequence to detect PSFs

0 0 0 0	0 0 0 0	1 0 1 0	1 0 1 0
0 0 0 1	0 1 0 0	1 0 1 1	1 1 1 0
0 0 1 1	1 1 0 0	1 0 0 1	0 1 1 0
0 1 1 1	1 1 0 1	1 1 0 1	0 1 1 1
1 1 1 1	1 1 1 1	0 1 0 1	0 1 0 1
1 1 1 0	1 0 1 1	0 1 0 0	0 0 0 1
1 1 0 0	0 0 1 1	0 1 1 0	1 0 0 1
1 0 0 0	0 0 1 0	0 0 1 0	1 0 0 0
0 0 0 0	0 0 0 0	1 0 1 0	1 0 1 0
0 0 1 0	1 0 0 0	1 0 0 0	0 0 1 0
0 1 1 0	1 0 0 1	1 1 0 0	0 0 1 1
1 1 1 0	1 0 1 1	0 1 0 0	0 0 0 1
1 1 1 1	1 1 1 1	0 1 0 1	0 1 0 1
1 1 0 1	0 1 1 1	0 1 1 1	1 1 0 1
1 0 0 1	0 1 1 0	0 0 1 1	1 1 0 0
0 0 0 1	0 1 0 0	1 0 1 1	1 1 1 0

5 Test pattern generation using dynamic CA

We now present a method to generate the code sequence depicted in Table 1 using an 8-cell CA as shown in Fig. 6*a*. To make the neighbourhood localised, the cells are arranged in a two dimensional plane as depicted in Fig. 6*b*. Additionally, to achieve dynamic behaviour (discussed later) the two rows are driven by distinct clock signals ϕ_1 and ϕ_2.

In this arrangement, the top row follows rule 170 with periodic boundary condition. Only one cell of the top row contains a 1 at any particular instant. This row basically contains the information regarding which bit position is changed in two successive patterns being generated. In effect, the top row is simply a cyclic left shift register.

The lower bit plane is driven by the outputs of the top row cells. A particular cell in the bottom row changes state when the corresponding neighbouring cell of the top row contains 1. The rule followed by the bottom row can be expressed algebraically as $x_0 \oplus x_{-4}$, with respect to Fig. 6*a*.

Thus the characteristic matrix of the 8-cell CA is given by

$$T_1 = \begin{bmatrix} 0\,1\,0\,0 & 0\,0\,0\,0 \\ 0\,0\,1\,0 & 0\,0\,0\,0 \\ 0\,0\,0\,1 & 0\,0\,0\,0 \\ 1\,0\,0\,0 & 0\,0\,0\,0 \\ 1\,0\,0\,0 & 1\,0\,0\,0 \\ 0\,1\,0\,0 & 0\,1\,0\,0 \\ 0\,0\,1\,0 & 0\,0\,1\,0 \\ 0\,0\,0\,1 & 0\,0\,0\,1 \end{bmatrix}$$

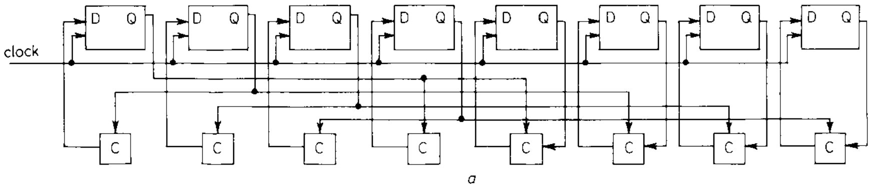
It can be shown that $T_1^8 = I$, which implies that this CA is a group CA of length 8.

We now consider another characteristic matrix T_2 in which cells of the bottom row do not depend on the cells of the top row; they depend only on themselves. The corresponding characteristic matrix is

$$T_2 = \begin{bmatrix} 0\,1\,0\,0 & 0\,0\,0\,0 \\ 0\,0\,1\,0 & 0\,0\,0\,0 \\ 0\,0\,0\,1 & 0\,0\,0\,0 \\ 1\,0\,0\,0 & 0\,0\,0\,0 \\ 0\,0\,0\,0 & 1\,0\,0\,0 \\ 0\,0\,0\,0 & 0\,1\,0\,0 \\ 0\,0\,0\,0 & 0\,0\,1\,0 \\ 0\,0\,0\,0 & 0\,0\,0\,1 \end{bmatrix} = \begin{bmatrix} T & 0 \\ 0 & T' \end{bmatrix}$$

It can be shown that $T_2^4 = I$, i.e. it is a group CA of length 4.

Here, the CA is made to change dynamically with time by changing its characteristic matrix. We choose the composite characteristic matrix as $T_1^8 T_2$, so that the CA is run for 4 subcycles, where in each subcycle the characteristic matrix is T_1 for 8 time steps and T_2 for 1 time step. So, according to Lemma 1, the CA forms a group CA of length $(8 + 1)4 = 36$. If two consecutive test vectors are identical then one of them is a dummy vector as it has no effect in testing. In that case, the similar test vectors are treated as one, applied for two time steps.



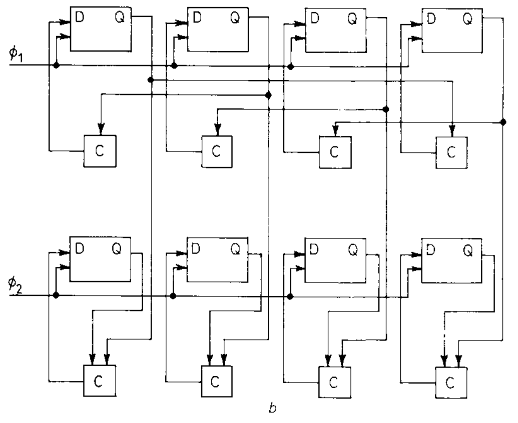


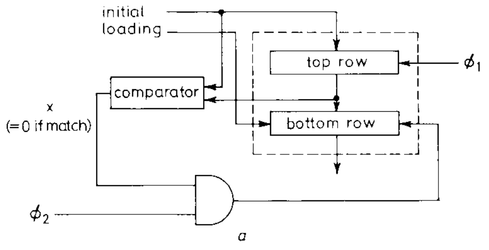
Fig. 6 *An 8-cell CA*
a 1-D organisation to generate the test vectors
b 2-D organisation of *b* with two clock signals

In our method, one part T' or T_2 is an identity matrix, which operates on the bottom row only. Although, the application of T_2 changes the global state of the CA, the bottom row do not change. It follows that the start of each subcycle produces a dummy state vector. Therefore, in the four subcycles there are four dummy state vectors. So effectively, we get $(36 - 4) = 32$ test vectors.

As discussed above, the CA generates 32 patterns in a single cycle. If the cycle is run with a new initial state then another set of 32 test patterns can be generated. By selecting suitable starting seed, the required 64 patterns (indicated in Table 1) can be generated. The control structure to achieve this is discussed next.

5.1 CA structure realising the dynamic behaviour

We now outline the design of a CA whose behaviour is represented by the composite characteristic matrix T given by $T_1^8 T_2$. The top and bottom rows of the CA are fed with distinct clock signals ϕ_1 and ϕ_2. However, the clock signal feeding the lower row is gated, which makes it possible to freeze the state of the lower row by disabling the clock ϕ_2. Disabling ϕ_2 effectively changes the characteristic matrix of the CA from T_1 to T_2. The two cycles of length 36 generated by the 8-cell CA are shown in Table 2, where the test pattern $\langle a_3, a_2, a_1, a_0\rangle$ is taken out from the bottom row of the CA. From the table it is clear that the process of enabling ϕ_2 for 8 cycles and disabling it for one cycle can be achieved by making $x = 0$ whenever the state of the top row is equal to its initial load value. The hardware circuitry of Fig. 7*a* is specifically meant to perform this control.



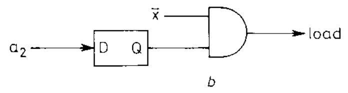


Fig. 7 *Generation of unit distance code*
a Realisation of dynamic *T*
b Loading of a new seed

Moreover, for the loading of the next seed, we see from the table that the end of the first cycle of length $(8 + 1)4 = 36$ can be detected by observing the pattern $\langle a_3, a_2, a_1, a_0\rangle = \langle 0\ 1\ 0\ 0\rangle$ at the end of a T_1 subcycle of length 8. The circuit to generate this load signal is

Table 2: Test patterns generated by CA

Initial loading: 0 0 0 1 0 0 0 0

	a_3 a_2 a_1 a_0		a_3 a_2 a_1 a_0		a_3 a_2 a_1
0 0 0 1	0 0 0 0	0 0 0 1	1 1 1 0	0 0 0 1	0 0 1
0 0 1 0	0 0 0 1	0 0 1 0	1 1 1 1	0 0 1 0	0 0 1
0 1 0 0	0 0 1 1	0 1 0 0	1 1 0 1	0 1 0 0	0 0 0
1 0 0 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 0 0	0 0 0
0 0 0 1	1 1 1 1	0 0 0 1	0 0 0 1	0 0 0 1	1 0 0
0 0 1 0	1 1 1 0	0 0 1 0	0 0 0 0	0 0 1 0	1 0 0
0 1 0 0	1 1 0 0	0 1 0 0	0 0 0 0	0 1 0 0	1 0 1
1 0 0 0	1 0 0 0	1 0 0 0	0 1 0 0	1 0 0 0	1 1 1
0 0 0 1	0 0 0 0	0 0 0 1	1 1 0 0	0 0 0 1	0 1 1
0 0 1 0	0 0 0 0	0 0 1 0	1 1 0 1	0 0 1 0	0 1 1
0 1 0 0	0 0 1 0	0 1 0 0	1 1 1 1	0 1 0 0	0 1 0
1 0 0 0	0 1 1 0	1 0 0 0	1 0 1 1	1 0 0 0	0 0 0

Initial loading: 0 0 0 1 1 0 1 0

	a_3 a_2 a_1 a_0		a_3 a_2 a_1 a_0		a_3 a_2 a_1
0 0 0 1	1 0 1 0	0 0 0 1	0 1 0 0	0 0 0 1	1 0 0
0 0 1 0	1 0 1 1	0 0 1 0	0 1 0 1	0 0 1 0	1 0 0
0 1 0 0	1 0 0 1	0 1 0 0	0 1 1 1	0 1 0 0	1 0 1
1 0 0 0	1 1 0 1	1 0 0 0	0 0 1 1	1 0 0 0	1 0 1
0 0 0 1	0 1 0 1	0 0 0 1	1 0 1 1	0 0 0 1	0 0 1
0 0 1 0	0 1 0 0	0 0 1 0	1 0 1 0	0 0 1 0	0 0 1
0 1 0 0	0 1 1 0	0 1 0 0	1 0 1 0	0 1 0 0	0 0 0
1 0 0 0	0 0 1 0	1 0 0 0	1 1 1 0	1 0 0 0	0 1 0
0 0 0 1	1 0 1 0	0 0 0 1	0 1 1 0	0 0 0 1	1 1 0
0 0 1 0	1 0 1 0	0 0 1 0	0 1 1 1	0 0 1 0	1 1 0
0 1 0 0	1 0 0 0	0 1 0 0	0 1 0 1	0 1 0 0	1 1 1
1 0 0 0	1 1 0 0	1 0 0 0	0 0 0 1	1 0 0 0	1 0 1

shown in Fig. 7*b*. It may be noted that the *load* signal will be active only at the end of the cycle of length 36.

6 Application of test patterns

There exists a number of tiling strategies for a 5-neighbourhood memory array. Our main aim is to choose a particular strategy such that greater parallelism is possible at the time of test application. One such tiling of an (8 × 8) memory array is shown in Fig. 8. *E* is the base cell and *A*, *B*, *C*, *D* are its four neighbours. Many base cells with identical neighbourhood can be accessed simultaneously so that the test application time gets reduced drastically.

E	D	E	C	E	D	E	C
B	E	A	E	B	E	A	E
E	C	E	D	E	C	E	D
A	E	B	E	A	E	B	E
E	D	E	C	E	D	E	C
B	E	A	E	B	E	A	E
E	C	E	D	E	C	E	D
A	E	B	E	A	E	B	E

C	E	D	E	C	E	D	E
E	B	E	A	E	B	E	A
D	E	C	E	D	E	C	E
E	A	E	B	E	A	E	B
C	E	D	E	C	E	D	E
E	B	E	A	E	B	E	A
D	E	C	E	D	E	C	E
E	A	E	B	E	A	E	B

Fig. 8 *A tiling strategy*

6.1 Parallel testing strategy

There is ample scope for parallelism in memory testing since by modifying the row and column decoders, a number of rows and columns can be selected at the same time. Accordingly, disjoint neighbourhood patterns may be applied and observed concurrently. A method of partitioning the memory array into a number of blocks and then testing each memory block parallely is considered here.

From simulation results it has been found Reference 12 that when more than one cell is selected by accessing multiple columns on a single row, the memory operates correctly, i.e. the multiple cell access is similar to single cell access. However, if multiple rows are accessed the speed of write operation is degraded too much. For read operation the situation is complex; if all the accessed cells contain the same value then the read is correct, but if they differ, the contents of some of the cells change [12].

One solution to the problem of write time degradation is to use a larger driver with higher current driving capability, so that it may be able to drive many cells at a time. However, as the driver size is increased, it consumes more power and space. Also, this will add additional delay in normal mode.

6.1.1 Solution to the problem of multiple row selection: To circumvent this problem we introduce an alternative method [3] to increase the driving capability. We assume that the total memory array ($\sqrt{n} \times \sqrt{n}$) is partitioned into n/p number of ($p\sqrt{n}$) memory array blocks, where p is the number of rows in a block. Row i ($1 \leqslant i \leqslant p$) of each block can be accessed simultaneously in the test mode. Here, we assume a *four partitioned* memory array, i.e. $n/p = 4$. As stated earlier, our tiling strategy is such that every fourth column is identical. Hence the number of columns are partitioned into groups of four (Fig. 9).

The decoder circuit of Fig. 9 selects alternate fourth columns and the transistor circuits enclosed within the dotted box is used one per group. The bit lines of the memory array are connected to the sense/drive amplifiers through transistor switches *T*1, *T*2, *T*3, *T*4. The pass transistors *P*1, *P*2, *P*3 are inserted to facilitate mutual current sharing. In the test mode the pass transistors are turned on. That is, in test mode when any one of the four bit lines are selected, the other three drivers which are quiescent normally, now boosts the current through the selected bit line. In normal mode the select line is 0, i.e. the pass transistors are off and simply one out of four column is selected by the modified decoder.

6.1.2 Reduction of memory write operations: The test set which we are using to test for PSFs in memory have the unique property that successive test vectors differ only in one bit position. Therefore, if we can detect the bit which changes state between successive test vectors, it is sufficient to write only the changed bit. The inherent assumption is that the memory is static and retains its stored information until modified further. To detect the changing bits, we can use the circuit as shown in Fig. 10, which compares the present test pattern with the previous one and detects the bit which has changed.

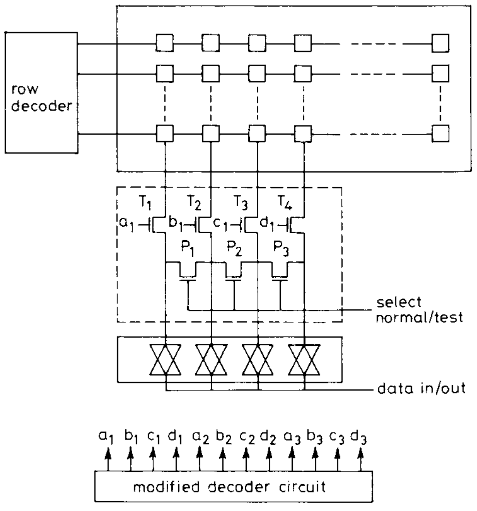


Fig. 9 *Parallel memory testing scheme*

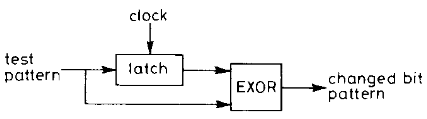


Fig. 10 *Detection of the changed bit*

As an example, let us consider a set of test patterns S_1, S_2, ..., S_m that are to be applied in sequence. Assume $S_1 = (0\ 0\ 0\ 0)$ and $S_2 = (0\ 0\ 0\ 1)$. Referring to Fig. 8, the test set is applied to the cells ($A\ B\ C\ D$), the neighbours of the base cell E. During application of the second pattern S_2, the comparator–decoder circuit compares S_1 and S_2 and decodes that bit which has changed, namely the bit corresponding to cell D. So to apply the second test pattern S_2, we write only the cell marked D instead of the four bits ($A\ B\ C\ D$). Obviously for m number of test patterns, the total number of writes is m instead of $4m$. So, in case of ANPSF, the total number of writes decreases from (4×256) to 256, and for PNPSF from (4×64) to 64.

For *dynamic RAM*, all the four bits ($A\ B\ C\ D$) of every test set has to be written into the neighbourhood cells during testing. It is evident from Fig. 8 that our tiling is such that the test bits A and B are in the same row, while C and D are in same other row.

From the previous discussion it is evident that multiple cells in the same row (i.e. multiple columns) can be accessed without any problem. So, in a test pattern if A and B are equal, then only p steps are required instead of $2p$ steps to write up all the As and Bs. A similar case exists for Cs and Ds. Now, if A and B differ (or C and D differ) then it takes $2p$ steps to write all As and Bs (or Cs and Ds). Table 3 shows the number of write operations required in our scheme. The last column of the table denotes the number of operations that would be necessary otherwise. p denotes the number of rows in a partition.

Table 3: Number of write operations

Pattern				Number of bit writes	
A	*B*	*C*	*D*	As per the proposed scheme	Normal
0	0	0	0	p	$2p$
0	0	0	1	$3p/2$	$2p$
0	0	1	1	p	$2p$
0	0	1	0	$3p/2$	$2p$
0	1	1	0	$2p$	$2p$
0	1	1	1	$3p/2$	$2p$
0	1	0	1	$2p$	$2p$
0	1	0	0	$3p/2$	$2p$

Hence for dynamic RAM two 2-bit comparators are used to compare A and B, and C and D. If the comparator checks that $A = B$ (or $C = D$) then the alternate columns are selected in a single row and are written in two steps rather than four steps as discussed in the above table. If ($A \neq B$) or ($C \neq D$) then they have to be written separately.

Using the comparator circuit for ANPSF the write time decreases from $(256 \times 2p)$ to $(96 \times 2p)$, and for PNPSF it decreases from $(64 \times 2p)$ to $(24 \times 2p)$.

6.2 The testing algorithms

Algorithm 1: Test for ANPSF
Step 1: Write 0 into all base cells.
Step 2: Repeat 64 times
(*a*) Write the current generated pattern into the cells.
(*b*) Read in parallel all the base cells.
Step 3: Write 1 into base cells.
Step 4: Repeat step 2.
Step 5: Interchange the role of base cells and A B C D cells and repeat steps 1, 2, 3, 4.

Time complexity for Algorithm 1:
(i) Writing all base cell — $4p$.
(ii) Writing all 128 test patterns — $130p$.
(iii) Read all base cells — $256p$.

Algorithm 2: Test for PNPSF
Step 1: Write 0 into all base cells.
Step 2: Repeat 16 times
(*a*) Write current test pattern in the neighbourhood.
(*b*) Read all base cells in parallel.
Step 3: Write 1 into all base cells.
Step 4: Repeat step 2.
Step 5: Interchange the role of base cells and the neighbourhood and repeat steps 1, 2, 3, 4.

Time complexity for Algorithm 2:
(i) Writing all base cells — $4p$.
(ii) Writing all 32 test paterns — $34p$.
(iii) Reading all base cells — $64p$.

7 Conclusion

A new strategy for parallel testing of RAM using a CA based test generation scheme has been reported. The advantage of the scheme is that it requires very simple hardware to generate the test patterns, and can be extended to PSFs of any neighbourhood size. Thus, if we want to generate n-bit code words with the same property as discussed, then $2n$ number of CA cells are required (organized in $2 \times n$ array). The number of code sequences to be generated is $n \times 2^n$. In that case, one loading produces $(n \times 2n)$ or $2n^2$ number of test vectors. So the number of loadings required is

$$\left\lceil \frac{n2^n}{2n^2} \right\rceil = \left\lceil \frac{2^{n-1}}{n} \right\rceil$$

For instance, test patterns for 9-neighbourhood PSFs ($n = 9$) can be generated by a 16-bit CA with a similar structure as discussed.

Several works have been reported [1, 5, 6] in the application of CA for a variety of test generation problems. Specifically, CA structures have been proposed which are capable of generating pseudo-random, pseudo-exhaustive, deterministic test patterns etc. The guiding motivation for the present work is to employ the same CA structure with dynamic T operator so that it can provide the BIST structure for memory testing. Work is being carried on to extend the capabilities of the dynamic CA to test the other memory subsystems and other types of memory array faults.

8 References

1 CHOWDHURY, D.R., DAS, A.K., MISRA, S., and CHAUDHURI, P.P.: 'Cellular automata — theory and applications', *JIETE*, 1990, **36**, (3/4), pp. 251–259

2 BREUER, M.A., and FRIEDMAN, A.D.: 'Diagnosis and reliable design of digital systems' (Computer Science Press, Potomac, MD, 1976)

3 CHOWDHURY, D.R., and CHAUDHURI, P.P.: 'Parallel memory testing: a bist approach', *in* Proceedings of the 3rd International Workshop on VLSI Design, Bangalore, India, 1989, pp. 373–377

4 CODD, E.F.: 'Cellular automata' (Academic Press, New York, 1968)

5 DAS, A.K., and CHAUDHURI, P.P.: 'An efficient on-chip deterministic test pattern generation scheme', *Euromicro J.*, 1989, **26**, pp. 195–204

6 DAS, A.K., and CHAUDHURI, P.P.: 'Efficient characterization of cellular automata', *Proc. IEE E*, January 1990, **137**, (1), pp. 81–87

7 SERRA, M., *et al.*: 'The analysis of one dimensional linear cellular automata and the aliasing properties', *IEEE Trans., Comput.-Aided Des.*, July 1990, **9**, (7), pp. 767–777

8 HAYES, J.P.: 'Detection of pattern-sensitive faults in random access memories', *IEEE Trans. Comput.*, February 1975, **24**, (2), pp. 150–157

9 HAYES, J.P.: 'Testing memories for single-cell pattern-sensitive faults', *IEEE Trans. Comput.*, March 1980, **29**, (3), pp. 249–254

10 HORTENSIUS, P.D.: 'Parallel computation of non-deterministic algorithms in vlsi'. PhD Dissertation (University of Manitoba, Winnipeg, Canada, 1987)

11 KNAIZUK, J. Jr., and HARTMANN, C.R.P.: 'An optimal algorithm for testing stuck-at faults in random-access memories', *IEEE Trans. Comput.*, January 1981, **30**, (1), pp. 1–17

12 LE, K.T., and SALUJA, K.K.: 'A novel approach for testing memory using bist techniques' *in* Proceedings of the International Test Conference, 1986, pp. 830–838

13 MAZUMDER, P., and PATEL, J.K.: 'Parallel testing for pattern-sensitive faults in semiconductor RAMs', *IEEE Trans. Comput.*,March 1989, **38**, (3), pp. 394–407

14 NAIR, R.: 'Comments on an optimal algorithm for testing stuck-at faults in random access memories', *IEEE Trans. Comput.*, March 1979, **28**, (3), pp. 258–261

15 HORTENSIUS, P.D., McLEOD, R.D., PRIES, W., MILLER, D.M., and CARD, H.C.: 'Cellular automata based pseudo-random number generators for built-in self-test', *IEEE Trans. Comput.-Aided Des.*, August 1989, **8**, (8), pp. 842–859

16 HORTENSIUS, P.D., McLEOD, R.D., and CARD, H.C.: 'Cellular automata circuits for built-in self test', *IBM J. Res. & Dev.*, March/May 1990, **34**, (2/3)

17 McLEOD, R.D., HORTENSIUS, P.D., and CARD, H.C.: 'Parallel pseudo-random number generation for vlsi systems using cellular automata', *IEEE Trans. Comput.*, October 1989, **38**, (10), pp. 1466–1473

18 McLEOD, R.D., HORTENSIUS, P.D., and CARD, H.C.: 'Cellular automata based signature analysis for built-in self-test', *IEEE Trans. Comput.*, October 1990, **39**, (10), pp. 1273–1283

19 McLEOD, R.D., HORTENSIUS, P.D., CARD, H.C., and PRIES, W.: 'Importance sampling for using computers using one-dimensional cellular automata', *IEEE Trans. Comput.*, June 1989, **38**, (6), pp. 769–774

20 SCHNEIDER, R., McLEOD, R.D., HORTENSIUS, P., and CARD, H.C.: 'Calbo — cellular automata logic block observation' *in* Canadian Conference on VLSI, November 1986

21 THATTE, S.M., NAIR, R., and ABRAHAM, J.: 'Efficient algorithms for testing semiconductor random-access memories', *IEEE Trans. Comput.*, June 1978, **27**, (6), pp. 572–576

22 MISRA, S., and CHAUDHURI, P.P.: 'Characterisation of additive cellular automata and its application for deterministic test pattern generation' *in* Proceedings on VLSI India, 1989

23 SUK, D.S., and REDDY, S.M.: 'Test procedures for a class of pattern-sensitive faults in semiconductor random-access memories', *IEEE Trans. Comput.*, June 1980, **29**, (6), pp. 419–429

24 SUK, D.S., and REDDY, S.M.: 'A march test for functional faults in semiconductor random access memories', *IEEE Trans. Comput.*, December 1981, **30**, (12), pp. 982–985

25 THANAILAKIS, A., PRIES, W., and CARD, H.C.: 'Group properties of cellular automata and vlsi applications', *IEEE Trans. Comput.*, December 1986, **35**, (12)

26 WOLFRAM, S.: 'Statistical mechanics of cellular automata', *Rev. Mod. Phys.*, July 1983, **55**, (3), pp. 601–644