JOURNAL OF ELECTRONIC TESTING: Theory and Applications 8, 153–164 (1996)
© 1996 Kluwer Academic Publishers. Manufactured in The Netherlands.

A New Reconfigurable Test Vector Generator for Built-In Self-Test Applications

SAMIR BOUBEZARI AND BOZENA KAMINSKA
Electrical Engineering Department, Ecole Polytechnique of the University of Montréal, P.O. Box 6079, Station Centre Ville, PQ, H3C 3A7 Canada
bozena@vlsi.polymtl.ca

Received January 13, 1995; Revised October 27, 1995

Editor: V.K. Agarwal

Abstract. This paper proposes a new approach to designing a BIST Test Vector Generator (TVG) for random vector-resistant circuits based on reconfigurable Cellular Automata Registers (CARs). Each CAR configuration is constructed by combining rules 90 and 150 and the same approach can also be applied to the Linear Feedback Shift Register (LFSR). The TVG thus designed is able to produce 100% fault coverage with short test time at the cost of low area overhead. To achieve this objective, a new method called the Rank Order Clustering (ROC) method, is introduced in order to fix a number of inputs at certain values when generating pseudorandom vectors. It is shown that the ROC method is very simple and efficient in fixing inputs at these values in terms of complexity. Experimental results have been conducted to demonstrate the applicability of the proposed approach in terms of hardware size and test application time.

Keywords: cellular automata, test vector generator, built-in self-test, rank order clustering

1. Introduction

The Built-In Self-Test (BIST) offers solutions to several testing problems. It can test the chip internally so that the need for external testing equipment is greatly reduced. Moreover, the BIST has the potential of being fast and efficient since it tests the circuit at its own speed. The principal property that a BIST Test Vector Generator (TVG) should satisfy in order to guarantee a high quality of test is to obtain **very high fault coverage** by a **simple TVG** in an **acceptable interval of time**. To address this problem, a number of techniques have been developed. These techniques can be classified in three major groups: exhaustive testing, deterministic testing and pseudorandom testing.

In exhaustive testing, all the vectors for an n-input circuit are generated for every output. Although this technique results in coverage of all combinational faults, the test application time can become too long when n is over 25. Deterministic testing requires a preliminary test generation step. The cost of test generation can be offset by savings in test time resulting from a much smaller number of tests being applied and the certainty of fault coverage [1]. Given a set of deterministic test vectors (obtained by an ATPG tool), deterministic testing can be achieved in several ways. The conceptually simplest approach is to store the deterministic test vectors in a Read-Only Memory (ROM) [1]. A counter is used to cycle through the ROM addresses. However, for practical circuits, the ROM may be rather large.

Finally, pseudorandom testing requires a long test time to achieve the desired fault coverage. The number of test vectors applied is typically in the order of 10,000 to 1,000,000 and is related to the circuit's testability and the fault coverage required. This technique, however, has the potential to cost less in terms of hardware size and require less design effort than the preceding techniques. Moreover, it eliminates the need to store the deterministic vectors. The most popular TVGs used in pseudorandom testing are Linear Feedback Shift Registers (LFSRs).

A new type of pseudorandom generator has received some interest recently. This generator, called the Cellular Automata Register (CAR), is a cascade of small 1-bit finite-state machines, where the next state of a particular cell is determined by a linear combination of the present states of the cell and its two adjacent neighboring cells [2]. Two linear combinations have been found to be useful for CARs in BIST applications. These are rules 90 and 150. The advantages claimed for this TVG type (i.e., a CAR constructed by combining rules 90 and 150) are that it has only local connections between cells and that it produces vectors that are more truly random than those produced by an LFSR [3].

Among the test vector generation techniques discussed so far, pseudorandom testing is the one most frequently used in BIST applications. However, in many cases, a primitive LFSR [1] or a CAR configuration (constructed by combining rules 90 and 150) is considered insufficient to cover a large percentage of faults. In fact, there are some potential problems associated with the use of pseudorandom testing, such as the test length required to achieve the desired fault coverage and the detection of random vector-resistant faults.

One practical approach to addressing this problem is to use a mixed TVG. In the first step, a primitive LFSR or a CAR is used to generate a pseudorandom sequence to cover a large percentage of easily testable faults. Next, deterministic test vectors generated by an ATPG are used for the detection of all random vector-resistant faults. The cost of the ATPG can be offset by the certainty of a known fault coverage.

Some existing techniques have been proposed in which other TVG structures are used to improve fault coverage and to reduce test length in pseudorandom testing. All these techniques increase the fault coverage and reduce test length at the cost of increased area overhead. A technique proposed in [4] combines a ROM, a counter and an LFSR to generate a set of test vectors. Each ROM word is used as the starting state of the LFSR, however, the problem of determining the ROM words and the LFSR feedback configurations still remains with this technique.

In [5, 6], a CA is selected to cover a set of deterministic test vectors (obtained by an ATPG tool). This technique applied to the data path is based on linear CAs that can be toggled between two linear rule groups. The set of test vectors is then covered by traversing parts of some cycles of the state transition graph of each rule group. The authors used the idea of switching between the two rules in order to reduce the number of initializing vectors in the ROM.[1]

In [7], the authors used a multiple-seed LFSR as a TVG. However, this technique requires twice as much hardware as that required for a single LFSR. In addition, no experimental results were given to show the validity of this technique. A mixed TVG based on the reseeding of multiple polynomial LFSRs has been proposed in [8]. This technique is computation-intensive, where the deterministic test vectors (obtained by the ATPG) are encoded as the polynomial identifier and a seed. Finally, some BIST TVG techniques based on bit-fixing have been proposed in [9–11]. These techniques have provided good results for most circuits. However, the complexity of hardware and the algorithm of bit-fixing still remain limiting factors.

The objective of this paper is to propose a new TVG which produces 100% fault coverage with shorter test length at the increase of a very low hardware overhead when compared to an alternative BIST TVG solution. The new TVG has better performances over a mixed TVG when it is used to generate a pseudorandom and a deterministic sequence. In fact, using our approach, we can reduce the length of the pseudorandom sequence and the hardware used to store the deterministic test set.

The main contribution of our approach compared to the previous techniques is the use of a new method called Rank Order Clustering (ROC). In fact, by using ROC method, we can fix a number of inputs at certain values when generating pseudorandom vectors and thus the test length is reduced. ROC method has been used and applied to solve the problem of machine-component group formation in group technology [12]. It is shown that it is a very simple method in terms of complexity and very efficient to the proposed TVG approach.

The new TVG structure generates a sequence of pseudorandom vectors to detect all possible easily testable faults in an acceptable length of time and for the remaining faults, called hard faults, we use an ATPG to generate a set of deterministic test vectors to detect the given hard faults. Then the deterministic test set is partitioned into a set of deterministic subsequences such that each one contains a number of fixed inputs.

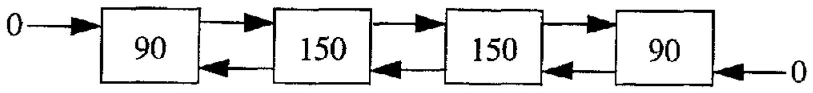

Fig. 1. An example of a 4-cell hybrid CAR with null boundary conditions.

The test set partitioning and input fixing is obtained by applying the ROC method. The resulting TVG is composed of a reconfigurable CAR used to generate either a fixed or a pseudorandom value and a ROM to store the fixed values, and is thus called a Fixed-Pseudorandom TVG (FPTVG). It is shown that the FPTVG structure yields better performances in terms of hardware size and test application time.

After this introductory section, Section 2 gives some basic definitions of CAs. Our approach is described in Section 3, where the FPTVG structure is presented. The experimental results based on benchmark circuits and area overhead evaluation are shown in Section 4. Finally, we conclude in Section 5.

2. Hybrid Cellular Automata Registers

CAs are defined as regular arrays of simple cells in an n-dimensional space. Generally, they can be characterized by four basic properties [2]: the cellular geometry, the neighborhood specification, the number of states per cell and the rule under which the CA cell computes its next state. A large variety of CA structures can be obtained by varying any of the above properties. The CA considered here is a binary one-dimensional array of cells. The neighborhood specification is defined by the Von Neuman neighborhood, which includes the physically nearest cells only. Furthermore, only linear rules are allowed. This means that the next state of a given cell can be computed from the previous state using only linear operations, that is, Exclusive-OR (addition over GF(2)) operators.

Definition 2.1 [2]. A CA in which all the cells are associated with the same rule is called a uniform CA, otherwise it is called a hybrid CA. Two linear combination rules have been found to be useful for CARs used as pseudorandom TVGs. The first linear rule (called rule 90) is where the next state of the cell is an XOR of the present state of its two neighbors. The second linear rule (called rule 150) is where the next state of a cell is an XOR of its own present state, and the present state of its two adjacent neighbors. The two linear rules are defined as follows:

$$\begin{aligned} &\text{Rule 90:} \quad x_i^{t+1} = x_{i-1}^t \oplus x_{i+1}^t \\ &\text{Rule 150:} \quad x_i^{t+1} = x_{i-1}^t \oplus x_i^t \oplus x_{i+1}^t \end{aligned} \tag{1}$$

where x_i^t denotes the state of the ith cell at time step (clock cycle) t. Figure 1 shows an example of a 4-cell hybrid CAR constructed by combining rules 90 and 150 with null boundary conditions.

A hybrid CAR can also be represented by a characteristic polynomial since it can be defined by a transition matrix. Pries et al. [2] have noticed that some combinations of rules 90 and 150 in a n-cell CAR will generate a maximum-length sequence of $(2^n - 1)$ different states, excluding the zero state. Zang et al. [13] have produced a table of minimal-cost CARs that produce a maximal-length sequence up to register length 150. In the rest of this paper, whenever a CAR is mentioned, we are referring to a hybrid, one-dimensional CAR constructed by combining rules 90 and 150 with null boundary conditions.

3. Basic Approach

Problem Formulation

Given a set of hard faults F (stuck-at) corresponding to a given circuit, find the minimal hardware size of a TVG implementation that can cover all given faults in F, with a minimal number of test vectors, i.e., in a minimal test time.

Motivation

In order to cope with the problem formulation, we have presented an approach for designing a deterministic TVG [14] through the use of nonlinear CA structures. The deterministic test vectors are partitioned into a set of equal-length subsequences such that each subsequence is generated by one nonlinear CA structure. Each subsequence is selected such that it covers a maximum number of deterministic test vectors. No ROM is used to store the initializing vectors since the last vector of a given subsequence is taken as the first vector of the next subsequence. Therefore, the resulting TVG structure is simply composed of a small counter and clock divider used to switch from one linear CA structure to another. However, since we have used nonlinear CA structures, switching between different nonlinear CAs results in large area overhead for a large test vector set.

The approach proposed here is an attempt to explore the best features of the technique presented in [14] and the pseudorandom testing technique. For our approach, a deterministic test vector set is first obtained by the ATPG tool to detect the hard faults. Then, using the ROC method, the test set is partitioned into a set of deterministic subsequences such that each subsequence contains a number of inputs fixed at certain values. In the following, we give the concept of the FPTVG technique. Note that the introduction of the ROC method in the TVG design is our original contribution.

Fixed-Pseudorandom TVG (FPTVG) Technique

It is known that a Pseudorandom TVG based on LFSR or CAR (constructed by combining rules 90 and 150) is very limited in its potential to achieve 100% FC for random vector-resistant circuits due to the long test length required to cover the random vector-resistant faults. This technique is an attempt to reduce the test length by achieving 100% FC at the cost of limited hardware overhead. The basic idea of this technique is to cover all possible easily testable faults by a pseudorandom sequence in an acceptable length of time and for all other remaining faults, called hard faults, we use an ATPG to generate a deterministic test vector set. Then, the deterministic test set is modified and partitioned into a set of deterministic subsequences in such a way as to have a number of inputs in each deterministic subsequence fixed at certain values, either to 0 or to 1. These inputs are kept to these values when generating pseudorandom test vectors, hence the pseudorandom test length will be decreased. The resulting FPTVG generates a set of equal-length pseudorandom subsequences to cover the original deterministic test set. The length of each pseudorandom subsequence is determined by the length of the first subsequence. Input fixing and deterministic test set partitioning are obtained using a new method called Rank Order Clustering (ROC). In the following we give first an example of bit-fixing in order to decrease the test application time. Next, ROC method is proposed as an efficient solution to bit-fixing and deterministic test partitioning.

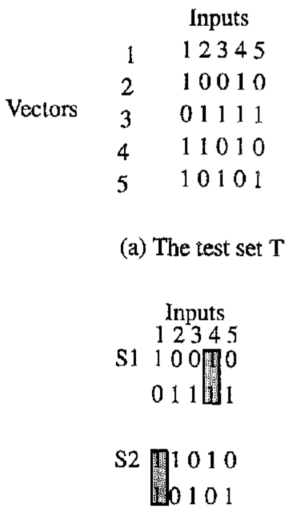
Inputs
Vectors
1 2 3 4 5
1 10010 / 2 01111 / 3 11010 / 4 10101 / 5
(a) The test set T
Inputs
1 2 3 4 5
S1 1 0 0 1 0
0 1 1 1 1
S2 1 1 0 1 0
1 0 1 0 1
(b) The two subsequences

Fig. 2. An example of a test set T.

Example 1. Consider a deterministic test set T obtained by an ATPG tool used to detect the hard faults of a given circuit. T is composed of 4 vectors, 5 bits each (Fig. 2).

The given test set, Fig. 2(a), presents no fixed input among the test vectors in T. If we split T sequentially into two deterministic subsequences, as shown in Fig. 2(b), the number of inputs (shown shaded) fixed at certain values in each subsequence is increased to one. The resulting FPTVG has to keep constant values (0 or 1) at certain inputs for a given period of test time while generating pseudorandom values for the remaining inputs. Therefore, the overall pseudorandom test length can be decreased by fixing some inputs at certain values during the pseudorandom generation process. For example, to cover the two subsequences S_1 and S_2 given in Fig. 2(b), the resulting FPTVG has to first fix the fourth input at value 1 and generate the pseudorandom values at the remaining inputs for an N-length pseudorandom subsequence in order to cover S_1. Next, the FPTVG will fix the first input at value 1 and generate pseudorandom values at the remaining inputs for the same N-length pseudorandom subsequence in order to cover S_2. Thus, a set of equal-length pseudorandom subsequences is generated with this technique by fixing a number of inputs at certain values (0 or 1).

We note that the position and the value of each input to be fixed change from one subsequence to another. Each cell of the resulting FPTVG has to be reconfigured to generate a fixed or pseudorandom value at each N-length pseudorandom subsequence, and these cells are thus called **reconfigurable cells**. However, not all cells of the resulting FPTVG have to be reconfigured since a certain number of cells generate only pseudorandom values, and we call these, **pseudorandom cells**. If we take the same Example 1 of the test set given in Fig. 2, to cover the two subsequences, the resulting TVG will be composed of 2 reconfigurable cells (the first and the fourth cell) and 3 pseudorandom cells for the remaining cells.

The basic FPTVG structure is given in Fig. 3 and consists of the following components:

1. A ROM, which is required to store the fixed values. The size of the ROM is proportional to the number of deterministic subsequences after partitioning the deterministic test set times the number of reconfigurable cells. In the worst case, the number of reconfigurable cells will be equal to the total number of the FPTVG cells. It is shown in the experimental results section that the number of reconfigurable cells is greatly reduced and so the ROM size as well.
2. Decoding logic (DL), which is used to reconfigure the corresponding cells such that each cell is able to generate a fixed or a pseudorandom value at each N clock cycles. Note that the DL outputs are used only for the reconfigurable cells since pseudorandom cells are not required to generate fixed values and are allowed to generate only pseudorandom values. The structure of a reconfigurable cell is shown in Fig. 4 in which a multiplexer is used to select between the fixed and the pseudorandom value.
3. A clock/divider, which is used to synchronize the ROM, DL and CAR such that for each ROM word, the DL configures the reconfigurable CAR cells to generate a pseudorandom sequence of length N by the CAR. The CAR configuration is selected such that it gives a maximal-length sequence.

In summary, the general concept of the proposed FPTVG consists of the following steps:

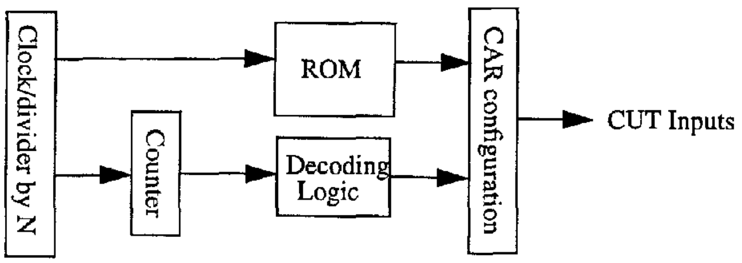

Fig. 3. The basic FPTVG structure.

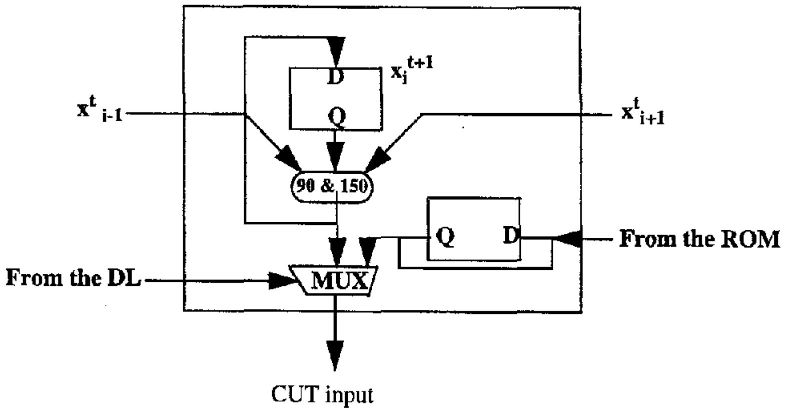

Fig. 4. The structure of a reconfigurable cell.

Step 1: Generate a sequence of pseudorandom test vectors to cover a large percentage of easily testable stuck-at faults.

Step 2: For the remaining undetected faults, called hard faults, use an ATPG to generate a deterministic test set which detects these faults.

Step 3: Partition the deterministic test set into a set of deterministic subsequences such that each one is composed of a number of fixed inputs.

Step 4: For each deterministic subsequence, fault simulate the FPTVG with fixed inputs against the undetected faults for N-length pseudorandom subsequence. If all faults are detected, go to Step 5, otherwise go to Step 2.

Step 5: Stop.

Since the main objective of our technique is to achieve 100% fault coverage and less test application time with limited hardware size, a compromise between two parameters (test application time and hardware size) is obtained. To accomplish this compromise, we consider two key parameters: the number of fixed inputs and the number of deterministic subsequences after partitioning the deterministic test set. Increasing the fixed inputs will increase the FPTVG hardware size and decrease the test application time. Note that the FPTVG hardware size is represented by the ROM, the DL and the number reconfigurable cells. However, if the number of fixed inputs decreases, the hardware size decreases by increasing the test application time. Hence, depending on the number of fixed inputs, the amount of FPTVG hardware introduced and the test application time can be controlled by the user specified parameter. In our technique, the number of fixed inputs is an important parameter to be controlled by a user so that it can meet varying design specifications. In the following section, we show how to partition the deterministic test set into a minimal number of deterministic subsequences in which each one contains a number of fixed inputs by using ROC method. First, we introduce a simple application of ROC method and our motivation for using this method for the FPTVG.

Rank Order Clustering (ROC) Method

ROC method has been used and applied to solve the problem of machine-component group formation in group technology [12]. It consists mainly of forming machines into groups and components into associates families. Figure 5(a) gives an illustration of the ROC

$$\begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline A & 1 & 1 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 1 & 0 & 1 & 0 \\ C & 1 & 1 & 0 & 1 & 0 & 0 \\ D & 0 & 0 & 1 & 0 & 1 & 1 \\ E & 1 & 1 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

Original matrix
(a)

$$\begin{array}{c|cccccc|} & 3 & 5 & 6 & 2 & 1 & 4 \\ \hline D & 1 & 1 & 1 & 0 & 0 & 0 \\ B & 1 & 1 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 1 & 1 & 1 \\ C & 0 & 0 & 0 & 1 & 1 & 1 \\ A & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

The new matrix after applying ROC
(b)

Fig. 5. An example of machine-component group formation using ROC method.

method application. A binary matrix is used in which rows are labelled from A to E and columns from 1 to 6. An entry 1 in cell (i, j) indicates that a relation exists between row (i) and column (j) whereas a zero entry means that it does not exist. Thus, it is clear from Fig. 5(a) that a relation exist between column 1 and rows A, C, and E. The main problem of this grouping is to partition rows and columns of the binary matrix into distinct groups. ROC method offers a solution to the above problem by applying simple transformations to the binary matrix (Fig. 5(a)). ROC method reads rows and columns as binary words and rearranges them so that they appear in the matrix ranked in descending binary order (counting from left to right for columns and top to bottom for rows). Figure 5(b) illustrates the new binary matrix by rearranging rows and columns in descending binary order. Now, two distinct groupings (B, D, 6, 5, 3) and (E, A, C, 2, 4, 1) emerge naturally along the diagonal of the binary matrix, and hence the grouping problem is solved. In the following section we show how ROC method is applied to the FPTVG technique.

Application of ROC Method to the Fixed-Pseudorandom TVG Technique

Given any binary matrix representing the deterministic test set, the ROC method consists of grouping 0's

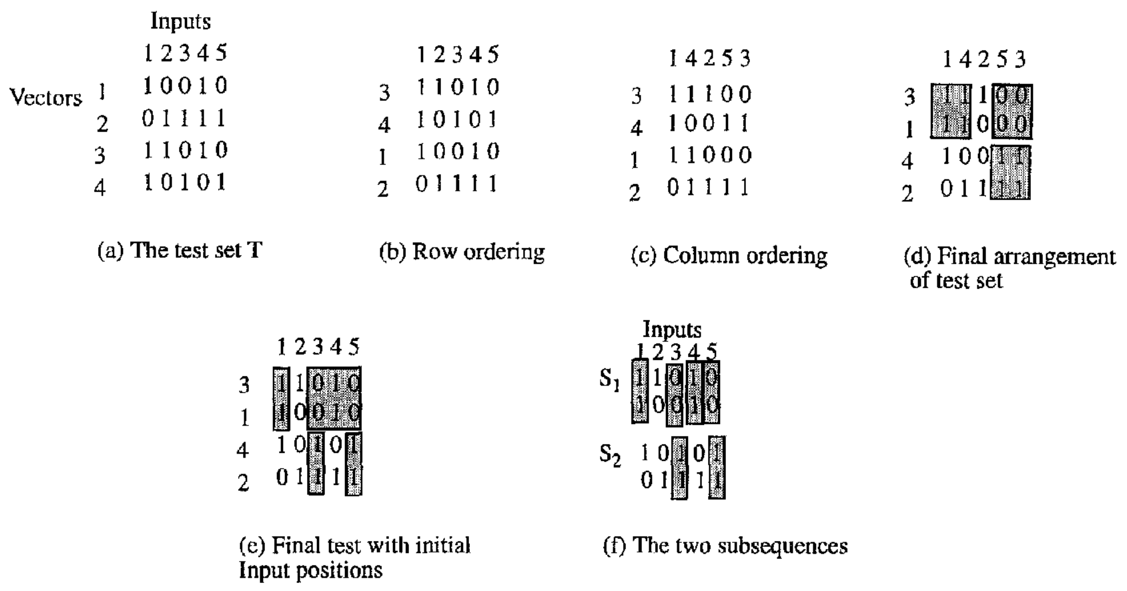

Fig. 6. Illustration of rank order clustering method in descending binary order.

and 1's by rearranging rows and columns in ascending or descending binary orders (counting from left to right for columns and top to bottom for rows). This transformation does not change the unordered original deterministic test set since we are concerned with the stuck-at faults. To see how this might be done, let us return to the test set given in Fig. 2 which is reported again in Fig. 6(a). For this example, we apply the ROC method by grouping 0's and 1's in descending binary order. To begin, we first rearrange the rows in descending binary order where the new test set is given in Fig. 6(b). Next, we take the matrix with rows rearranged (Fig. 6(b)) and we do the same transformation by considering columns. The new test set is given in Fig. 6(c). Now, we note that by rearranging columns, the descending order of rows has changed so that the same process is repeated until the rows and columns are respectively rearranged in descending binary order. The final test set is given in Fig. 6(d), and is composed of the set of clusters or groups of 1's and 0's shown selected. By replacing the initial positions of inputs of the original test set, we obtain the new test set in Fig. 6(e) with the same number of clusters of 0's and 1's. By splitting the final test as in Fig. 2, we obtain 4 fixed inputs in the first subsequence and 2 fixed inputs in the second subsequence (Fig. 6(f)). Therefore, the number of fixed inputs in each subsequence is increased when it is compared to the one in Fig. 2, and thus the overall random test length is decreased much more. However, the number of reconfigurable cells is increased to four (only the second cell generates a pseudorandom value), and therefore the FPTVG hardware size is increased. The number of fixed inputs differs from one subsequence to another. In our case we will take the average number of fixed inputs for all subsequences. For the example given in Fig. 6(f), the average number of fixed inputs is three.

ROC method can be also applied in ascending binary order. For the example given in Fig. 6, the descending binary order has been used. To apply the ROC method in ascending binary order to the same example of the test set given in Fig. 2, the number of fixed inputs will decrease compared to that with the descending binary order. The final test set after applying the ROC method in ascending binary order is given in Fig. 7.

To decide whether or not to use the ROC method in ascending or descending binary order, some simulation results are given in Table 1. Three random binary matrices were generated in C language. Each binary matrix is composed of 50 vectors of 20 bits each. For each binary matrix, we give the percentage of 1's in the corresponding matrix by counting the number on 1's over the total number of 1's and 0's. The average number of fixed inputs resulting from splitting sequentially the corresponding matrix is given by using the

Table 1. Simulation results using the ROC method.

		ROC (descending binary order)	ROC (ascending binary order)
Binary matrices	Percentage 1's [%]	Average number of fixed inputs	Average number of fixed inputs
M_1	70	14	8
M_2	30	7	13
M_3	50	11	10

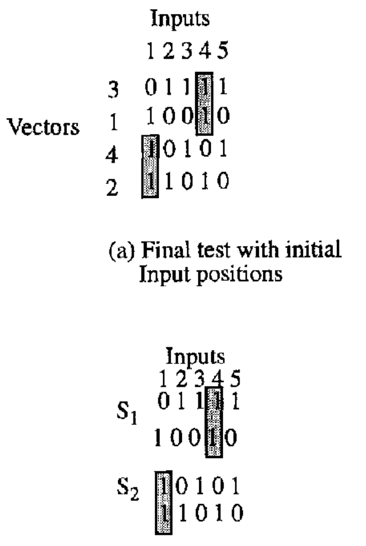

Fig. 7. The ROC method in ascending binary order.

ROC method in ascending and descending binary order. Note that each matrix is split sequentially into subsequences of length 5. It is shown that for a given binary matrix, if the number of 0's is greater (less) than the number of 1's, the average number of fixed inputs increases when ROC method is applied in ascending (descending) binary order. Therefore, depending on the percentage of 1's and 0's in the corresponding test set (represented as a binary matrix), we have to choose the best case of the ROC method in order to increase the average number of fixed inputs.

In summary, the algorithm of the ROC method consists of the following steps. This algorithm has the computational complexity of cubic order in the worst case, namely $O(mn(m+n))$, where m and n are the number of rows and columns respectively. However, by using efficient algorithms such as Quicksort or Mergesort, the overall complexity may be reduced to $O(mn(\log(mn))$, compared to $O(mn(m+n))$.

Rank Order Clustering Algorithm

Step 1: Given a binary matrix A_k, k is the number of iterations, set $k = 1$.
Step 2: Order A_k rows in ascending (or descending) binary order[2] and update the matrix A_k with the new arrangement.
Step 3: Repeat step 2 with A_k columns. $k = k + 1$.
Step 4: Repeat step 2 and step 3 until the matrix A_k is the same as A_{k-1}.

The Partitioning Technique

The problem of partitioning a deterministic test set into a set of subsequences such that each subsequence contains a number of fixed inputs is known as an NP-complete problem [10]. However, there exists a trade-off between the number of subsequences and the average number of fixed inputs. In our technique, we propose an algorithm to partition the test set based on the ROC method.

We have seen earlier that by applying ROC method to a given test set, the final test set arrangement consists of a set of clusters or groups of 0's and 1's. Thus, it is much easier to partition such a test set arrangement since it will just splitting the test set sequentially into a set of deterministic subsequences such that each subsequence contains a number of fixed inputs. This is the main advantage of the ROC method in partitioning the test set such that each subsequence contains the same average number of fixed inputs. As mentioned above, the average number of fixed inputs is determined as a user-specified parameter. Algorithm 1 gives the main steps involved in the construction of a set of deterministic subsequences with the required fixed inputs using the ROC algorithm.

Table 2. Experimental results for the FPTVG for a given subsequence length N_{seq}.

Circuits	No. of inputs	Percentage of fixed inputs = 20%				Percentage of fixed inputs [%] = 50%				Percentage of fixed inputs = 80%			
		N_{seq}	$\#nb_{\text{seq}}$	T_{app}	$\#R_{\text{cells}}$	N_{seq}	$\#nb_{\text{seq}}$	T_{app}	$\#R_{\text{cells}}$	N_{seq}	$\#n_{\text{seq}}$	T_{app}	$\#R_{\text{cells}}$
c2670	233	3,600	8	28.8 K	133	1,000	17	17 K	187	500	27	13.5 K	211
c7552	207	15,000	5	75 K	122	3,500	15	52.5 K	167	1,500	23	34.5 K	202
s420	34	3,000	2	6 K	7	1,500	3	4.5 K	19	700	4	2.8 K	22
s641	54	15,000	1	15 K	18	5,500	2	11 K	30	3,000	3	9 K	37
s838	67	30,000	3	90 K	20	10,000	8	80 K	49	4,000	13	52 K	62
s9234	247	15,000	10	150 K	109	6,000	19	114 K	187	2,500	28	70 K	233

Algorithm 1: Construction of Subsequences

Let $T_{\text{roc}} = \{V_j / j = 1, \ldots, m\}$ be a deterministic test set obtained by applying the ROC method algorithm. Let n_{fixed} be the average number of fixed inputs given as a user-specified parameter.

Step 1: Set $i = 1$, $j = 1$. Each deterministic subsequence S_i obtained after partitioning the deterministic test set must contain at least one vector. Take V_j from T_{roc}. Let this vector be x and mark it as covered in T_{roc}. Store it in S_i and set $j = j + 1$.

Step 2: Take V_j from T_{roc} as a successor vector to follow x in S_i. If the number of fixed inputs in S_i is less than or equal to n_{fixed}, mark V_j as covered in T_{roc} and store it to follow x in S_i and go to step 3, otherwise go to step 4.

Step 3: Let V_j be the new x and set $j = j + 1$. If there are still uncovered vectors in T_{roc}, go to step 2, otherwise go to step 5.

Step 4: Set $i = i + 1$, store V_j in S_i. Let V_j be the new x, set $j = j + 1$ and go to step 2.

Step 5: Stop.

4. Experimental Results

In this section, we give the experimental results for various benchmark circuits and the area overhead comparison for FPTVG approach. The algorithm from section 3 has been implemented using C language on Sun Spark Station 10. The circuits are from the ISCAS85 [15] and ISCAS 89 [16] benchmark sets.

An alternative BIST generator based on a ROM in which we store the entire deterministic test set is considered in order to compare the hardware requirements of the FPTVG.

Table 2 gives the experimental results for the FPTVG technique where only random vector-resistant circuits are considered. For each circuit, we give the number of inputs, the length of each pseudorandom subsequence, N_{seq}, the number of deterministic subsequences, ($\#nb_{\text{seq}}$) after partitioning the deterministic test set, the test application time, T_{app}, to achieve 100% fault coverage, and finally the number of reconfigurable cells ($\#R_{\text{cells}}$). Each of these parameters are given for three values of fixed-input percentage given as user-specified parameter. The percentage of fixed inputs is given as the number of fixed inputs over the total number of circuit inputs and the test application time is given by the ($\#nb_{\text{seq}}$) times the length N_{seq}. The length of each pseudorandom subsequence, N_{seq}, reported in Table 2, is selected such that it results in a minimal ($\#nb_{\text{seq}}$), ($\#R_{\text{cells}}$) and test application time.

From Table 2, for a given subsequence length, N_{seq}, the amount of FPTVG area overhead is controlled by the user-specified parameter given by the fixed-input percentage. Note that the FPTVG area overhead is proportional to ($\#nb_{\text{seq}}$) and ($\#R_{\text{cells}}$). In fact, by increasing (decreasing) fixed-input percentage, we increase (decrease) ($\#nb_{\text{seq}}$) and ($\#R_{\text{cells}}$) by decreasing (increasing) the test application time. Therefore, an acceptable balance must be obtained between overhead cost and test time. To determine the best balance of overhead cost and test time is a design issue which varies depending on the test priorities of the application. Note that the increasing of the N_{seq} decreases the FPTVG hardware size (represented by ($\#nb_{\text{seq}}$) and ($\#R_{\text{cells}}$)) at the expense of increasing in test application time.

In Table 3 we compare the hardware size of the FPTVG with the deterministic strategy based on the entire storage of the test set in a ROM. The FPTVG is basically measured by the corresponding ROM used to store the fixed values and the DL used to configure the

Table 3. Area overhead comparisons of the EPTVG.

	Deterministic testing		Percentage of fixed input = 20%		Percentage of fixed inputs = 50%		Percentage of fixed inputs = 80%	
Circuits	Test vectors	ROM (bits)	ROM (bits)	% of reduction	ROM (bits)	% of reduction	ROM (bits)	% of reduction
c2670	91	21203	2128	89.96	6358	70	11394	46.26
c7552	71	14697	1220	91.69	5051	65.91	9292	36.77
s420	6	204	28	86.27	114	44.11	176	13.72
s641	5	270	36	86.66	120	55.55	222	17.77
s838	34	2278	120	94.73	784	65.58	1612	29.23
s9234	89	21983	2180	90	7106	67.67	13048	40.64

reconfigured cells. In this comparison we will consider the DL as a ROM with its size given by (#R_{cells}) multiplied by (#nb_{seq}). Hence the total memory required for the FPTVG is ($2 \times R_{\text{cells}} \times nb_{\text{seq}}$). The ROM size for the deterministic strategy is given by the number of deterministic test vectors times the number of circuit inputs. The hardware comparison in Table 3 is given for three values of the percentage of fixed inputs (20%, 50% and 80%). It is shown that the size of the ROM in FPTVG is greatly reduced when compared to the ROM size of the deterministic testing. We note that by increasing the percentage of fixed inputs, this reduction decreases since we increase the FPTVG hardware size.

The comparison with other approaches like [9, 11] is a difficult task. In fact, in our approach, the number of fixed inputs is a user-specified parameter (see Table 2) which is not the case in [9, 4]. Therefore, the comparison in terms of test application, the number of fixed inputs and area overhead depends on the value of the user specified parameter (the percentage of fixed inputs).

Discussion

As mentioned in the abstract, the same approach can be applied to the LFSR. To define a LFSR, the only requirement is to specify which stages feed back to the first stage via an XOR gate circuit in which a DL is used to select a given polynomial by means of the control switches. It was reported in [3] that a CAR cell needs approximately double the area than a LFSR cell. Hence, the area overhead resulted from a CAR can be decreased if LFSR has been used. However, the benefits derived out of the CAR-based test structures may outweigh the area penalty, particularly for application employing LFSR with large number of cells and feedback paths. In fact, for larger implementations, the area of LFSR grows rapidly as a function of the number of cells than a CAR. In addition, the advantage in speed of a CAR over a LFSR is even more significant as the CAR approach is not expected to become slower, whereas the LFSR approach will become appreciably slower, since the delay grows linearly with the number of cells. Another, significant advantage of CAR is their modularity and cascadability, i.e., the physical length of the of CAR can be decreased or increased by simply adding or subtracting cells. This means that a CAR-based test vector generator is very appropriate for incorporation in CAD tool.

5. Conclusion

A new approach to designing fixed-pseudorandom TVG based on reconfigurable CARs has been presented. The use of a FPTVG for random vector-resistant circuits is an attempt to fix a number of fixed inputs at certain values (0 or 1). These inputs are kept constant at these values when generating pseudorandom vectors. Bit-fixing is computed efficiently by using the ROC method. The amount of hardware overhead introduced by the FPTVG is controlled by the percentage of fixed inputs. In comparison with the deterministic strategy, the FPTVG is better in terms of hardware size.

Acknowledgment

The authors thank Dr. Samir Lejmi for his helpful comments on the mathematical formulation of ROC method.

Notes

1. In this technique, each initialization vector stored in the ROM corresponds to a cycle generated by a CA rule group.
2. Rows and columns having the same binary value are ordered randomly in the order reading from left to right for columns and from top to bottom for rows.

References

1. P.H. Bardell, W.H. Mcanney, and J. Savir, *Built-In Self-Test for VLSI: Pseudorandom Techniques*, John Wiley and Sons, New York, 1987.
2. W. Pries, A. Thanailakis, and H.C. Card, "Group Properties of Cellular Automata and VLSI Applications," *IEEE Transactions on Computers*, Vol. C-35, pp. 1013–1020, December 1986.
3. P.D. Hortensius, R.D. Mcleod, W. Pries, D. Miller, and H.C. Card, "Cellular Automata-Based Pseudorandom Number Generator for Built-In Self Test," *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 8, pp. 842–858, August 1986.
4. V.K. Agarwal and E. Cerny, "Store and Generate Built-In Self-Testing," *Proc. FTCS 11*, June 1981, pp. 35–40.
5. J. Van Sas, F. Cattoor, and H. De Man, "Cellular Automata Based Self-Test for Programmable Data Paths," *IEEE International Test Conference*, 1990, pp. 769–778.
6. J. Van Sas, F. Cattoor, and H. De Man, "Optimized BIST Strategies for Programmable Data Paths Based on Cellular Automata," *IEEE International Test Conference*, 1992, pp. 110–119.
7. J. Savir and W.H. McAnneey, "A Multiple Seed Feedback Shift Register," *IEEE International Test Conference*, 1990, pp. 657–659.
8. S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vectors Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE International Test Conference*, 1992, pp. 120–129.
9. I. Pomeranz and S.M. Reddy, "3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 12, pp. 1050–1058, 1993.
10. S. Pateras and J. Rajski, "Cube-Contained Random Patterns and Their Application to the Complete Testing of Synthesized Multi-Level Circuits," *IEEE International Test Conference*, 1991, pp. 473–482.
11. M.F. AlShaibi and C.R. Kime, "Fixed-Biased Pseudorandom Built-In Self-Test for Random Pattern Resistant Circuits," *IEEE International Test Conference*, 1994, pp. 929–938.
12. J.R. king and V. Nakornchai, "Machine-Component Group Formation in Group Technology: Review and Extension," *International Journal of Production Research*, Vol. 20, pp. 117–133, 1982.
13. S. Zang, D.M. Miller, and J.C. Muzio, "Determination of Minimal Cost One-Dimensional Linear Cellular Automata," *IEE Electronics Letters*, Vol. 27, No. 18, 1991, pp. 1625–1627.
14. S.Boubezari and B. Kaminska, "Cellular Automata Synthesis Based on Precomputed Test Vectors For Built-In Self Test," *IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, November 1993, pp. 578–583.
15. F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Circuits and a Target Translator in FORTRAN," *IEEE ISCAS*, pp. 663–698, June 1988.
16. F. Brglez, D. Bryan, and K. Kozminski, "Acceerated ATPG and Fault Grading via Testability Analysis," In *Dig. Int. Sym. Circuits and Systems*, pp. 1929–1934, 1989.
17. S. Wolfram, "Statistical Mechanics of Cellular Automata," *Rev. Modern Physics*, Vol. 55, pp. 601–644, 1983.
18. S. Wolfram, "Universality and Complexity of Cellular Automata," *Physica*, Vol. 10D, pp. 1–35, 1984.
19. M. Khare and A. Albicki, "Cellular Automata Used for Test Pattern Generation," In *Proc. of ICCD'87*, 1987, pp. 56–59.
20. W. Daehn and J. Mucha, "Hardware Test Pattern Generation for nuilt-In Testing," *IEEE International Test Conference*, 1981, pp. 110–113.
21. S.B. Akers and W. Jansz, "Test Embedding in a Built-In Self-Test Environment," *IEEE International Test Conference*, 1989, pp. 257–263.
22. R. Dandapani, J.H. Patel, and J.A. Abraham, "Design of Test Pattern Generator for Built-In Self-Test," *IEEE International Test Conference*, 1984, pp. 315–319.
23. M. Serra, T. Slater, J.C. Muzio, and D.M. Miller, "The Analysis of One Dimensional Cellular Automata and Their Aliasing Properties," *IEEE Transaction on Computer Aided Design of Circuits and Systems*, Vol. 9, No 7, pp. 767–778, July 1990.
24. K.D. Aloke and P.P. Chaudhuri, "Vector Space Theoretic Analysis of Additive Cellular Automata and Its Application for Pseudoexhaustive Test Pattern Generation," *IEEE Transactions On Computers*, Vol. 42, No. INSKA, "Cellular Automata Synthesis Based on Precomputed Test Vectors for Built-In Self-Test," *IEEE International Conference on Computer-Aided Design*, November 1993, pp. 578–583, Santa Clara, CA.
25. M. Serra, D.M. Miller, and J.C. Muzio, "Linear Cellular Automata and LFSR's are Isomorphic," *Proceedings of the Third Technical Workshop: New Directions for IC Testing*, Hlifax, Nova Scotia, Canada, 1988, pp. 213–223.
26. B. Ayari and B. Kaminska "BDD_FTEST: Backtrack Test Generator Based on Binary Decision Diagram Representation," To appear in *IEEE Transactions on Computer Aided Design of Circuits and Systems.*
27. I. Pomeranz and S.M. Reddy, "A Learning-Based Method to Match a Test Pattern Generator to a Circuit-Under-Test," *IEEE International Test Conference*, 1993, pp. 998–1007.
28. D.J. Neebel and C.R. Kim "Inhomogeneous Cellular Automata for Weighted Random Pattern Generation," *IEEE International Test Conference*, 1993, pp. 1013–1022.
29. H.K. Lee and D.S. Ha, "An Efficient Forward Fault Simulation Based on the Parallel Pattern Single Fault Propagation," *Proc. of International Test Conference*, 1991.
30. Cadence VlSI Design Software.
31. The CMOS4S Standard Cell Library, as distributed by Canadian Microelectronics Corporation, 1990.
32. H.K. Lee and D.S. Ha., "Atalanta: An Efficient ATPG for Combinational Circuits," Technical Report, 93–12, Electronic Testing Group, Dept. of Electrical Eng., Verginia Polytechnic Institute and State University, Blacksburg, Virginia.
33. A. Kusiak, A. Vannelli, and K.R. Kumar, "Grouping Problem in Scheduling Flexible Manufacturing Systems," *Robotica*, Vol. 3, pp. 245–252, 1985.

Samir Boubezari received the B.Sc. degree in Electrical Engineering from Constantine University, Algeria, and the M.Sc. degree in Electrical Engineering from École Polytechnique of Montréal University, Canada.

He is currently a Ph.D. student at the École Polytechnique of Montréal where he worked as a research assistant in the Electrical and Computer Engineering Department. His current research interests include built-in self-test, synthesis for testability and design for testability. He is a Student Member of the IEEE Computer Society.

Bozena Kaminska received the M.Sc. and Ph.D. degree in Electrical Engineering from Warsaw University of Technology, Poland.

She is an associate professor in the Department of Electrical Engineering at the École Polytechnique of Montréal University, Canada. Her research interests include various aspects of analog and digital testing, design for testability, built-in self-test, and high-level synthesis. She is a Member of the IEEE Computer Society. She serves on the editorial board of the *Journal of Electronic Testing: Theory and Applications.*