Highly Regular, Modular, and Cascadable Design of Cellular Automata-Based Pattern Classifier

Santanu Chattopadhyay, Shelly Adhikari, Sabyasachi Sengupta, and Mahua Pal

***Abstract*—This paper enumerates a new approach to the solution of classification problems based on the properties of Additive Cellular Automata. Classification problem plays a major role in various fields of computer science, such as grouping of the records in database systems, detection of faults in VLSI circuits, image processing, and so on. The state-transition graph of Non-group Cellular Automata (CA) consists of a set of disjoint trees rooted at some cyclic states of unit cycle length-thus forming a natural classifier. First a scheme of classifying the patterns distributed into only two classes has been dealt with. This has been further extended for solution of the multiclass classification problem. The Multiclass Classifier saves on an average 34% of memory as compared to the straight-forward approach storing directly the class of each pattern. A regular, modular, and cascadable hardware implementation of the classifier has been presented which is highly suitable for VLSI realization. The design has been specified in Verilog and verified for functional correctness.**

***Index Terms*—Cellular automata, hardware classifier, pattern classifier, very large scale integration (VLSI) circuits.**

I. Introduction

THE ABILITY of machines to perceive their environment is limited. The apparent ease with which the vertebrates and insects perform their perceptual tasks is at once encouraging and frustrating. Psychological and physiological studies have provided many interesting facts, but nothing is sufficient to duplicate their ability by computers. A modest problem emerging in this field is pattern classification—the assignment of physical object or event to one or more prespecified categories [1]. However, the problem of classification can be identified in almost any field—be it the grouping of the genetic codes, chemical compounds, database records, or VLSI fault diagnosis. Neural networks is one of the modern methodologies for solving the classification problem. But the design of neural network-based circuits become far more intriguing and complex when the membership criteria becomes complicated. In this work, an elegant solution based on Cellular Automata (CA) has been reported for solving the generic classification problem. It has been found from the experimental results that the memory requirement in CA-based approach is much less as compared to the case where the class of individual patterns are stored directly. The scheme can be efficiently implemented by a highly regular, modular, and cascadable hardware structure—the three essential qualities the VLSI designers look for.

CA consist of a number of cells interconnected in a regular manner. *Von Neumann* [2] proposed a model of universal machines (a cellular space) involving 5-neighborhood cells each having 29 states. The theory of CA received consolidation by *Burks* [3] and considerable simplifications were subsequently introduced by Codd [4]. Wolfram [5]–[7] pioneered the investigation of CA as mathematical models for self-organizing statistical systems and suggested the use of a simple 2-state, 3-neighborhood CA with cells arranged linearly in one dimension. Each cell essentially comprises of a memory element and a combinational logic that generates the next-state of the cell from the present-state of its neighboring cells—*left*, *right*, and *self*. Martin [8] used polynomial algebraic tools to derive some characterization of *periodic boundary uniform* CA with identical CA rule applied to each of the cells. A new era of research on theory and applications of CA has been initiated with the work of Das [9]–[11], that dealt with analytical characterization of CA behavior based on matrix algebraic tools. This technique is capable of characterizing *periodic/null boundary* hybrid CA with different rules applied to different cells. A CA with nonsingular characteristic matrix is termed as *Group CA*, else it is called *Nongroup CA*. The state-transition behavior of group CA is characterized by the fact that every state has got a unique predecessor. Whereas in a nongroup CA, the number of predecessors is either zero, or 2^i, for some $i > 0$. Group CA has been studied extensively in [8], [10]–[14]. Many realistic applications have also been reported—*pseudo-random* [12], [13] and *pseudo-exhaustive pattern generation* [11], [15], *signature analysis* [14], [16], [17], *error-correcting codes* [18], and *cryptography* [19]. However, the study of nongroup CA behavior has not been given due attention in the past. A particular class of nongroup CA referred to as $D1^*$ CA is studied in details and used for synthesis of testable FSM [20]. Non-group CA has also been utilized to design efficient hashing functions [21].

In this paper, we have used a special class of nongroup CA whose state-transition diagram consists of a disjoint set of (inverted) trees rooted at states lying on cycles of length unity (Fig. 1). In the rest of the paper, such inverted trees are mentioned simply as trees. Let such a CA be loaded with any arbitrary bit pattern and allowed to run autonomously for a number of clock cycles equal to the depth of such trees. Naturally, the CA will evolve through a number of states and finally reach one of these cyclic states and remain there forever. It may be easily noted from Fig. 1, that all states lying on the same tree will reach the same cyclic state, whereas the states belonging to different trees will reach different ones. This has motivated us

Manuscript received May 17, 1999; revised April 12, 2000.

S. Chattopadhyay is with the Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati, India.

S. Adhikari is with Delsoft India Pvt. Ltd., Noida, Uttar Pradesh 201 303, India.

S. Sengupta is with Wipro Technologies, Sri Chamundi Complex, Bommanahali, Bangalore 560 068, India.

M. Pal is with WIPRO Infotech, Calcutta 700 071, India.

Publisher Item Identifier S 1063-8210(00)10103-9.

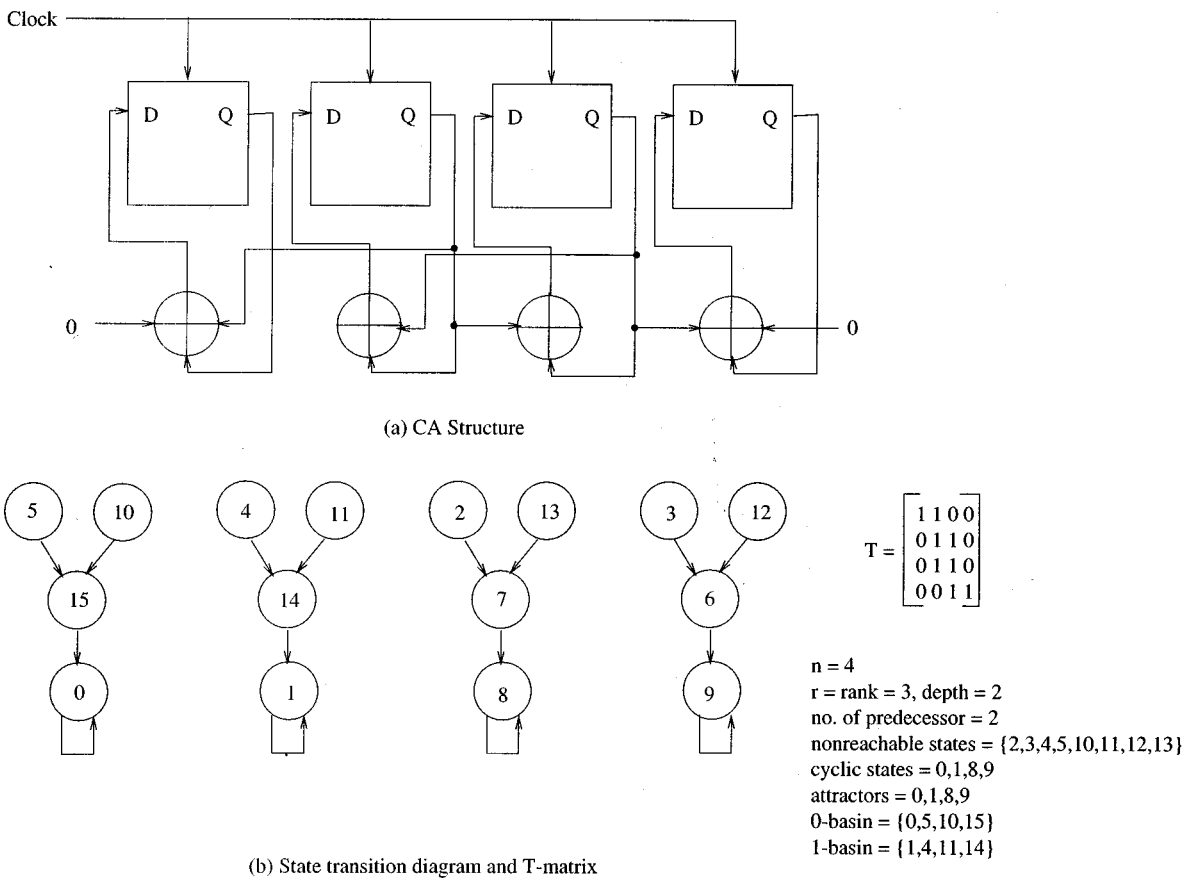

Fig. 1. Structure and behavior of a 4-cell CA.

to look into such tree structures as *pattern classifiers*—in which the states belonging to the same tree form a class. In our work, first a CA has been designed, which can classify a given set of patterns P into two classes P_1 and P_2 (say). Such a *Two-Class Classifier* has been extended to solve the multiclass classification problem in which a pattern set P is to be classified into k-classes. Finally a hardware implementation of the classifier is also presented. The highly regular, modular, and cascadable structure of the classifier makes it ideal for VLSI implementation. The design has been specified in Verilog and simulated for functional correctness.

Section II presents the preliminaries of CA along with some previously established relevant characterizations. Section III analyzes the behavior of a class of nongroup CA named as Multiple Attractor CA (MACA). Section IV utilizes the properties of MACA to build the pattern classifier with two mutually disjoint classes. Subsequently, the scheme is extended to handle nondisjoint larger number of classes. Section V deals with the hardware implementation of the two-class classifier. Section VI presents the experimental results. Section VII highlights two realistic applications of such a classifier.

II. CA Preliminaries

A CA consists of a number of interconnected cells arranged spatially in a regular manner [5]. In the most general case, each such cell can exist in m different states and the next state of any particular cell depends on the present states of k of its neighbors. Such a general CA is called an m-state k-neighborhood CA. In the present work we have used 2-state, k-neighborhood n-bit CA. Each of the cells is essentially comprised of a memory element built with a D flip-flop, a combinatorial logic that generates the next-state from the present state of itself and its neighbors. All earlier works [11]–[20], [22] employed CA with three neighborhood structure (*left, self, and right*) [see Fig. 1(a)]. However, in this work, we have relaxed the neighborhood restrictions for better solutions to the classification problem. Unrestricted neighborhood does not result in any penalty for software implementation of the CA-based scheme.

For an n-cell one–dimensional linear CA with XOR rules, it has been shown [10], [11] that the linear operator is an $n \times n$ Boolean matrix whose ith row specifies the dependency of the ith cell of the CA on other cells [Fig. 1(b)]. The next state of the CA is generated by applying this linear operator on the present CA state represented as a column vector. The operation is the normal matrix multiplication, but the addition involved is modulo-2 sum [that is, all operations are in GF(2)]. This matrix is termed as the *Characteristic Matrix* of the CA and is denoted by T. If f_t is a column vector representing the state of the automata at tth instant of time, then the next state of a linear CA is given by $f_{t+1} = T \times f_t$.

It has been proven in [10] that if the T matrix is nonsingular, then the CA is a *group CA*; otherwise it is a *nongroup CA*. In the state transition graph of a group CA all states are cyclic, each state has a unique predecessor and a unique successor state. Fig. 1 displays the state transition graph of a nongroup CA with

its characteristic matrix and other relevant information. In the state transition diagram of a nongroup CA [Fig. 1(b)], a state having at least one indegree is called a *reachable state*, while a state with no indegree is called a *nonreachable state*. Reachable states which lie on cycles are called *cyclic states*. A state which has a self loop is referred to as a *graveyard state* or *single cycle attractor*. In the present work, a single cycle attractor is simply referred to as an *attractor*. Thus, an attractor in the rest of the paper will be viewed as a cyclic state with unit cycle length. The maximum number of state transitions required to reach the nearest cyclic state from any nonreachable state is defined as the *depth* of the CA [Fig. 1(b)]. The set of states lying on the tree rooted at an attractor k is referred to as *k-basin*. In the rest of the paper we will be using both the terms "basin" and "tree" interchangeably. We define MACA to be the one in which each of the cyclic states is an attractor—that is, it lies in a cycle of unit length and there are multiple such attractors in the state-transition diagram. Thus the CA shown in Fig. 1 is a multiple attractor one with four attractors.

Theorem 1: The number of attractors in a Multiple Attractor CA is 2^{n-r}, where n is the number of cells of the CA, and r is the rank of the $T \oplus I$ Matrix, I being the $n \times n$ identity matrix [21].

Theorem 2: The number of predecessors in a Multiple Attractor CA is 2^{n-r}, where n is the number of cells of the CA, and r is the rank of the T Matrix [21].

III. Characterization of Multiple Attractor CA (MACA)

In this section we report some interesting properties of multiple attractor CA. Such CA is characterized by multiple single cycle attractors and trees rooted on such attractors in their state transition behavior. From Theorem 1 it can be noted that if the characteristic matrix of such an n-cell CA be T, and $rank(T \oplus I) = r$, then the number of attractor states is 2^{n-r}. Thus, by selecting the particular CA configuration with the required T matrix, it is possible to get a CA with varying number of attractors.

Next we state an important result regarding the pseudo-exhaustive nature of attractors.

Pseudo-Exhaustive Patterns: A set of patterns is said to generate k-bit pseudo-exhaustive patterns if there exists k bit positions containing all possible 2^k patterns. For example, the pattern set $\{0, 1, 4, 7\}$ generates 2-bit pseudo-exhaustive patterns $\{00, 01, 10, 11\}$ as shown below

$$\begin{matrix} \mathbf{0}00 \\ \mathbf{0}01 \\ \mathbf{1}00 \\ \mathbf{1}11 \end{matrix}$$

Theorem 3: In an n-cell multiple attractor CA with 2^m attractors, there exists m bit positions at which attractors give pseudo-exhaustive 2^m patterns [21].

Example: The 4-cell CA shown in Fig. 1 has attractors $0(\mathbf{0}000)$, $1(\mathbf{0}001)$, $8(\mathbf{1}000)$, and $9(\mathbf{1}001)$. The attractors generate pseudo-exhaustive patterns at bit positions 1 and 4. □

A. D1-MACA

An MACA having depth equal to one, is of special interest for our classifier design. Such an MACA is called a D1-MACA. D1-MACA has got some special features that have been exploited fruitfully in designing the classifier:

1) Since the depth is unity, to reach each attractor from a nonreachable state, the CA needs to be run for a single cycle only. As a result, the run time of the classifier to identify the class of a given pattern gets reduced.
2) It significantly reduces the computation necessary to search for the set of MACA that are required to design the classifier.

Theorem 4: For any d-depth MACA there exists a D1-MACA

Proof: It follows directly from the fact that if T be the characteristic matrix of a d-depth MACA, then the CA with characteristic matrix $T_1 = T^d$ is a depth-1 MACA. The new CA will have same set of attractors as the earlier ones, however all nonreachable states will be at depth unity only. □

A 4-cell 2-depth MACA with its characteristic matrix is shown in Fig. 2(a). The corresponding D1-MACA is shown in Fig. 2(b). Note that such a transformation may cause a change in the neighborhood of the cells of the CA, here in our example, the third cell from left in Fig. 2(b) has four neighborhood.

IV. MACA-Based Classification Strategy

An n-bit multiple attractor CA with m-attractors can be viewed as a natural classifier. It classifies a given set of numbers into m distinct classes, each class containing the set of states in the attractor basin. This can be envisaged as follows: when a CA is loaded with a number which is the bit-pattern whose class has to be determined, and is allowed to run in autonomous mode for a number of clock cycles equal to the depth of the CA, it evolves through a number of states and ultimately reaches an attractor state. As per Theorem 3 above, the positions giving the pseudo-exhaustive patterns will identify the class of the pattern uniquely. This pseudo-exhaustive field yields the memory address of the class, to which the pattern belongs to. This has been explained in Fig. 3.

A. Two-Class Classification

Let the given pattern set P be consisting of only two disjoint pattern classes P_1 and P_2. That is to say, all patterns in the set P belongs to either of P_1 or P_2. If the classifier classifies the pattern set then for any two patterns $x \in P_1$and $y \in P_2$ should lie on separate trees. Thus if T be the characteristic matrix of the D1-MACA performing the classification, then the following relations should hold:

$$\begin{gathered} \text{Relation R1: } \forall\, x \in P_1 \text{ and } \forall\, y \in P_2 \\ \Rightarrow T \cdot (x) \neq T \cdot (y) \\ \Rightarrow T \cdot (x \oplus y) \neq 0 \end{gathered}$$

Also to ensure that it is a depth 1 CA, the following relation must hold true:

$$\begin{gathered} \text{Relation R2: } T^2 = T \\ \Rightarrow T \cdot (T \oplus I) = 0 \end{gathered}$$

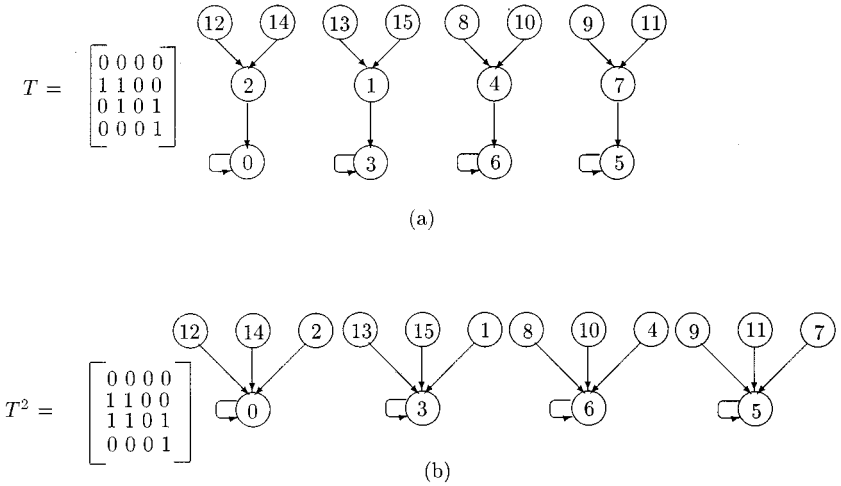

Fig. 2. (a) A 2-depth MACA and (b) corresponding 1-depth MACA.

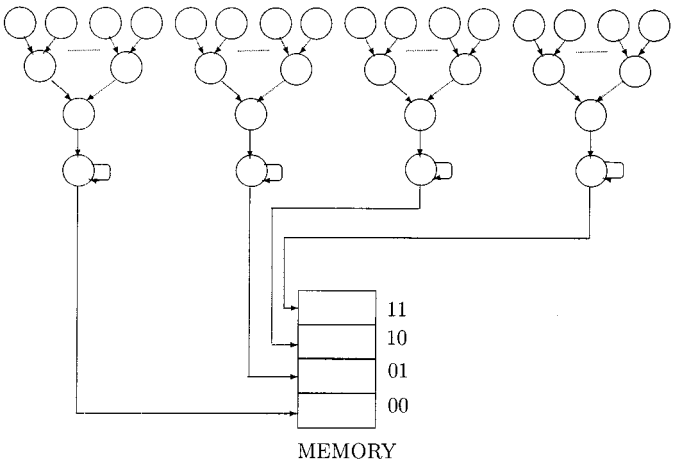

Fig. 3. CA-based classification strategy.

Thus, any CA having a characteristic matrix satisfying relations R1 and R2 is a classifier for the pattern set P. Each basin of the CA will contain patterns from either of P_1 or P_2, but not both. The two classes can be distinguished by a single bit, b [let $b = 1$ for P_1 and $b = 0$ for P_2].

1) Algorithm for a Two-Class Classifier: Consider that a given set of patterns P, the elements of which are to be classified into two disjoint classes *viz.* $P_1 = \{P_{11}, P_{12}, \cdots, P_{1n}\}$ and $P_2 = \{P_{21}, P_{22}, \cdots, P_{2m}\}$. From the above theoretical background, if T be characteristic matrix of the CA classifying them, then we must have

$$\forall P_{1i} \text{ and } \forall P_{2j} \Rightarrow T \cdot (P_{1i} \oplus P_{2j}) \neq 0, \text{ where } i = 1, \cdots, n \text{ and } j = 1, \cdots, m.$$

Moreover, to ensure that T is a D1-MACA, we should have $T^2 = T$. The algorithm to construct a classifier for two disjoint classes is noted in Fig. 4. A function to identify the class of a given pattern has been given in Fig. 5.

Function **Two-Class Classifier** is illustrated in the following example.

Example 2: Let us consider the 4-bit pattern set $\{0, 1, 2, 5, 6, 7, 10, 11, 15\}$ consisting of two classes *Class-1* $= \{0, 1, 6, 7, 15\}$ and *Class-2* $= \{2, 5, 10, 11\}$. Let T be the characteristic matrix of a CA in which patterns of *XOR-set* does not belong to the 0-*basin*. Assuming,

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix}$$

The corresponding equations, as per the *Relation R1*, are shown in Table I. Any solution to this system of equations which satisfies the *Relation R2* noted as $T^2 = T$, or $T \cdot (T \oplus I) = 0$ is a classifier for the classes P_1 and P_2. These equations can be solved by constructing the corresponding Binary Decision Diagram (BDD) and finding the satisfiability set of it. However, solving these equations directly is highly computation intensive. We now discuss the steps of the algorithm to construct our classifier.

The XOR-*set* in this case is $\{2, 3, 4, 5, 10, 11, 12, 13\}$. The XOR set expressed in binary is $\{0010, 0011, 0100, 0101, 1010, 1011, 1100, 1101\}$. To accomplish Step 3, since the number of 1's in each column is same, so we scan from right to left, and select the least significant column position first, thus the XOR elements 0011, 0101, 1011, and 1101 are first selected. Now observe that XOR elements 0010 and 1010 are yet unselected and has 1's in their second least significant bit position. So this column gets selected. Finally, 0100 and 1100 has 1's in the third least significant bit position. Hence, this column is also selected. Observe that the most significant bit position is never selected. Thus this bit has been ignored in our further discussions. Hence, as per Step 3 of the algorithm, $p = 3$. Now,

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

Function : TwoClassClassifier

Input

P_1 : set of patterns belonging to class-1

P_2 : set of patterns belonging to class-2

Output

The T-Matrix of the CA classifying class-1 and class-2

L : The look-up table containing the pseudo-exhaustive bits of the attractors, the trees of which hold patterns of P_1.

Assumption

P_1 and P_2 are disjoint

begin

1. Compute the XOR set $X = P_1 \oplus P_2$, by XORing each pattern of P_1 with each pattern of P_2. Let X = $\{ x_1, x_2, \ldots, x_k \}$
2. Convert each element x_i of XOR set X to binary. Let $x_i = b_{(r-1)i} b_{(r-2)i} \ldots b_{0i}$ where each pattern is r-bit.
3. Identify the minimum set of columns such that each x_i has got 1 in at least one of these columns. Let the columns be $\{ l_1, l_2, \ldots, l_p \}$.
4. Reorganise the pattern sets P_1 and P_2 so that the columns l_1 through l_p are the most significant ones. Let the modified XOR set be X′.
5. Create the matrix T_p as $p \times p$ identity matrix.
6. Try to merge some of the rows of T_p by Binary Decision Diagram (BDD) based optimisation technique, such that for the modified matrix also (say $T_{p'}$, $p' \times p$), $T_{p'}.X' \neq 0$.
7. Construct the T Matrix as
$$\begin{bmatrix} & 0 & \\ T_{p'} & & I_{p' \times p'} \end{bmatrix}$$
8. Let A_1 and A_2 denote the set of attractors trees of which contains patterns of P_1 and P_2 respectively, with cardinalities a_1 and a_2. If $a_1 < a_2$ then A = A_1 else A = A_2;
9. Store the pseudo-exhaustive bits of each element of A in L.

end.

Fig. 4. Algorithm to construct a two-class classifier.

Function : IdentifyClassOfPattern

Input

CA of the TwoClassClassifier.

p : the pattern whose class is to be determined.

L : the look-up table containing the pseudo-exhaustive bits of the attractors, the trees of which hold the patterns of P_1.

Output

The class in which the pattern p belongs.

begin

Load the CA with pattern p;

Run the CA with the pattern p for 1 clock cycle;

x = pseudo-exhaustive bits of the attractor in which the pattern p lies;

Search L for x.

if found

then pattern p lies in P_1;

else pattern p lies in P_2.

end.

Fig. 5. Function to identify class of a pattern in two-class classification.

Step 4 reorganizes the pattern set to the following:

Class-1:

$\{0\underline{000} \rightarrow \underline{000}0, 0\underline{001} \rightarrow \underline{001}0, 0\underline{110} \rightarrow \underline{110}0,$
$0\underline{111} \rightarrow \underline{111}0, 1\underline{111} \rightarrow \underline{111}0\} = \{0, 2, 12, 14\}.$

Class-2:

$\{0\underline{010} \rightarrow \underline{010}0, 0\underline{101} \rightarrow \underline{101}0, 1\underline{010} \rightarrow \underline{010}0,$
$1\underline{011} \rightarrow \underline{011}0\} = \{4, 10, 6\}.$

The modified XOR set X' is given by, $X' = \{010, 011, 100, 101\}$. Here the *least significant bit* position

TABLE I
CONDITIONS AND EQUATIONS FOR RELATION R1

$T \cdot 2 \neq 0$	$t_{13} + t_{23} + t_{33} + t_{43} = 1$
$T \cdot 3 \neq 0$	$(t_{13} \oplus t_{14}) + (t_{23} \oplus t_{24}) + (t_{33} \oplus t_{34}) + (t_{43} \oplus t_{44}) = 1$
$T \cdot 4 \neq 0$	$t_{12} + t_{22} + t_{32} + t_{42} = 1$
$T \cdot 5 \neq 0$	$(t_{12} \oplus t_{14}) + (t_{22} \oplus t_{24}) + (t_{32} \oplus t_{34}) + (t_{42} \oplus t_{44}) = 1$
$T \cdot 10 \neq 0$	$(t_{11} \oplus t_{13}) + (t_{21} \oplus t_{23}) + (t_{31} \oplus t_{33}) + (t_{41} \oplus t_{43}) = 1$
$T \cdot 11 \neq 0$	$(t_{11} \oplus t_{13} \oplus t_{14}) + (t_{21} \oplus t_{23} \oplus t_{24}) + (t_{31} \oplus t_{33} \oplus t_{34}) + (t_{41} \oplus t_{43} \oplus t_{44}) = 1$
$T \cdot 12 \neq 0$	$(t_{11} \oplus t_{12}) + (t_{21} \oplus t_{22}) + (t_{31} \oplus t_{32}) + (t_{41} \oplus t_{42}) = 1$
$T \cdot 13 \neq 0$	$(t_{11} \oplus t_{12} \oplus t_{14}) + (t_{21} \oplus t_{22} \oplus t_{24}) + (t_{31} \oplus t_{32} \oplus t_{34}) + (t_{41} \oplus t_{42} \oplus t_{44}) = 1$

TABLE II
MODIFIED EQUATION AND CONDITIONS FOR RELATION R1

$T \cdot 2 \neq 0$	$t_{12} + t_{22} + t_{32} = 1$
$T \cdot 3 \neq 0$	$(t_{12} \oplus t_{13}) + (t_{22} \oplus t_{23}) + (t_{32} \oplus t_{33}) = 1$
$T \cdot 4 \neq 0$	$t_{11} + t_{21} + t_{31} = 1$
$T \cdot 5 \neq 0$	$(t_{11} \oplus t_{13}) + (t_{21} \oplus t_{23}) + (t_{31} \oplus t_{33}) = 1$

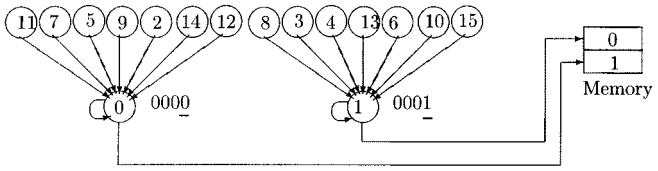

Fig. 6. Example of a two-class classifier.

hasbeen discarded, since it is always zero. To consider the possible merging of rows of T_p we form BDDs of the following operation:

$$\begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} \cdot X' \neq 0$$

A solution to set of equations shown in Table II is $t_{31} = 1$, $t_{32} = 1$. All other t_{ij}'s are equal to zero. Hence Step 6 of the algorithm will produce the matrix $T_{p'}$ (with p' being equal to 1) as, $T_{p'} = [110]$. Thus Step 7 constructs the T-Matrix as:

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

Rank of the above T-Matrix is unity and that of the $T \oplus I$ matrix is three. Hence, as per Theorem 1, there will be $2^{4-3} = 2$ attractors and, as per Theorem 2, each will have $2^{4-1} = 8$ predecessors. Fig. 6 shows the classifier constructed.

Thus we can conclude, that it is possible to design a Two-Class Classifier with a minimized memory requirement. Our strategy of two-class classification has been tested for randomly generated data sets, some of the results are enumerated in the Section VI. The two-class classifier discussed in this section can be viewed as a single stage classifier. It is a CA represented by the specified T matrix. For designing Multiclass Classifier, this scheme of single stage classification will be repeatedly employed that leads to a multi-stage classifier consisting of multiple CA, each CA corresponding to a single stage.

B. Multiclass Classification

In this section, we present the strategy for classifying patterns into more than two classes. Let the class of n-bit patterns P be consisting of k classes $P_1, P_2, \cdots, P_k$. In this k-class classifier the restriction regarding the disjoint nature of the data set has been relaxed, that is there can be common elements in the pattern sets $P_1, P_2, \cdots, P_k$. Algorithm for multiclass classification is noted in Fig. 7. Fig. 8 notes the function to identify the class of a given pattern.

Example 3: Let P_1, P_2, P_3, P_4 be four pattern classes which are not mutually disjoint. Consider that $P = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and $P_1 = \{1, 2, 3, 4\}$, $P_2 = \{2, 5, 7, 8\}$, $P_3 = \{6, 9, 5, 1\}$, $P_4 = \{0, 5, 6, 8\}$. Observe that the pattern 5 belongs to P_2, P_3, P_4. Our classifier sets an output register of 4 bits to $0, 1, 1, 1$ as the pattern 5 belongs to the above three classes. Similarly for pattern 8, which belongs to P_2 and P_4, the configuration of the output register will be $0, 1, 0, 1$.

In our strategy, we consider two temporary classes, *TempClass*$_1$ and *TempClass*$_2$ such that *TempClass*$_1$ $= P_1$ and *TempClass*$_2$ $= P_2 \cup P_3 \cup P_4 - P_1$. In our example, we have *TempClass*$_1$ $= \{1, 2, 3, 4\}$ and *TempClass*$_2$ $= \{2, 5, 7, 8\} \cup \{6, 9, 5, 1\} \cup \{0, 5, 6, 8\} - \{1, 2, 3, 4\}$. That is to say *TempClass*$_2$ $= \{0, 5, 6, 7, 8, 9\}$. At this point observe that *TempClass*$_1$ and *TempClass*$_2$ are mutually disjoint and hence a *Two-Class Classifier*, as proposed in the previous section can be used to identify the class to which a pattern p actually belongs to. If the pattern under test belongs to class C_1, then the corresponding bit of the output register, here the bit 1, is set by the CA, otherwise it sets the bit 1 to 0. In the second pass consider that *TempClass*$_1$ $= P_2$ and *TempClass*$_2$ $= P_1 \cup P_3 \cup P_4 - P_2$. In this way, after four passes, all the bits of the output register can be set/reset. □

V. HARDWARE REALIZATION

In this section, we discuss the hardware realization of the CA-based classification scheme. We first present a structure for two-class classification. The scheme being highly regular, modular, and cascadable, a number of such basic building blocks can be cascaded to design a Multiclass Classifier capable of handling any number of classes consisting of patterns of large bit size.

A. Hardware for Two-Class Classification

The basic building block for two-class classifier consists of a 16-bit *Pattern Register (PATT-REG)*, and an 8×16 *array of*

```
Function : MultiClassClassifier
    Input
        P_1, P_2, ..., P_k:  sets of patterns.
    Output
        The classifiers CA_1, CA_2, ..., CA_k.
        L :  The lookup table containing the pseudo-exhaustive bits
             of the attractors of each of the CA classifying the pattern
             set into P_1, P_2, ..., P_k.
begin
        Let TempClass_1 and TempClass_2 be two temporary sets of patterns.
        for i = 1 to k do
        begin
            TempClass_1 := P_i
            TempClass_2 := φ
            for j := 1 to k do
                TempClass_2 := TempClass_2 ∪ P_j;
            TempClass_2 := TempClass_2 - P_i;
            CA_i := TwoClassClassifier(TempClass_1, TempClass_2);
            L_i := Concatenation of the bit-streams representing the
                number of attractors (b bits) and the pseudo-exhaustive
                bits of attractors corresponding to TempClass_1 or
                TempClass_2 as identified by the TwoClassClassifier.
        end
        Merge the Look-up tables L_1, L_2, ..., L_k to form
            a single look-up table L.
end
```

Fig. 7. Algorithm to construct a two-class classifier.

```
Function : IdentifyClassOfPattern
    Input
        CA_1, CA_2, ..., CA_k :  the k classifiers.
        L :  The lookup table containing the pseudo-exhaustive bits
             of the attractors of each of the CA classifying the pattern
             set into P_1, P_2, ..., P_k.
        p :  the pattern whose class is to be identified.
    Output
        The k-bit OutputRegister with i^th bit set
        if the pattern p belongs to class P_i.
begin
    Clear OutputRegister
    for each CA_i [ ∀ i ∈ 1, ..., k ] do
        begin
            Load the CA_i with pattern p;
            Run the CA for 1 clock cycle;
            x = Pseudoexhaustive bits of the attractor of P;
            Search L for x;
            if found
                then OutputRegister[i] := 1;
        end;
    return OutputRegister;
end.
```

Fig. 8. Function to identify class of a pattern in two-class classification.

cells. It can handle patterns of size up to 16 bits, and can implement a CA-based classifier that has atmost 256 attractors. Of course, for larger sized patterns and/or higher number of attractors, two or more such blocks can be cascaded. The structure of the basic building block has been shown in Fig. 9. The details of each the components are outlined next.

1) *Pattern Register(PATT-REG):* The 16-bit pattern register to hold the pattern, whose class has to be identified.

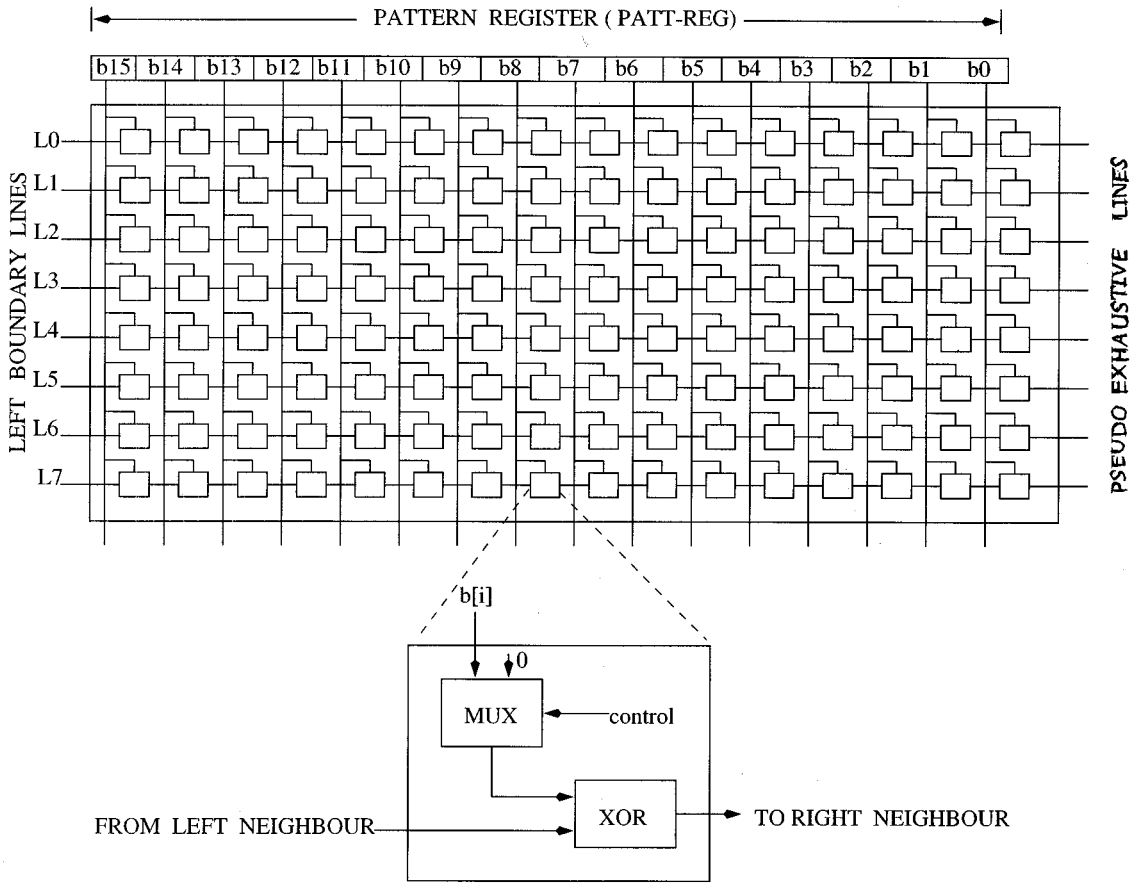

Fig. 9. Basic architecture of aCA-based two-class classifier.

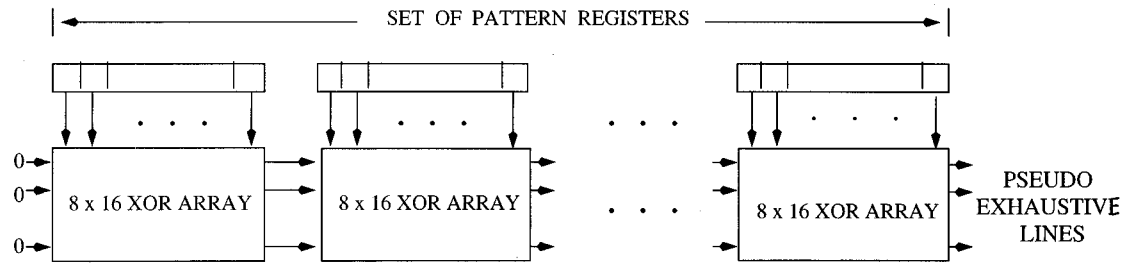

Fig. 10. Horizontal cascading of two-class classifiers.

2) *Array of Cells:* This is a regular array of 8×16 *cells*. Each row of the array realizes one of the nonzero rows of the T-matrix. It may be recalled that each nonzero row of the T-matrix of the CA implies the XOR of some of the bits of the pattern to be classified. Each cell consists of a multiplexer and an XOR gate. The control line of the multiplexer is programmed to select the bit b_i, if b_i is present in the corresponding row of T-matrix, otherwise it passes a "0." One of the inputs to the XOR-gate comes from the multiplexer, the other input comes from the *left neighboring cell*. Similarly, output of the XOR-gate goes to the *right neighbor*.
3) *Left Boundary Lines:* These lines act as the left neighbors of the leftmost cells. Normally, these should be set to zero. The lines help in horizontal cascading discussed later.
4) *Pseudo-exhaustive Lines:* These lines are the outputs of the rightmost cells. These lines will contain the pseudo-exhaustive bit pattern for the attractor, the basin of which contains the pattern to be classified. Hence, these lines can be used to search in the external lookup table. These lines can also be used for cascading.

B. Horizontal Cascading

To classify patterns of width more than 16 bits, two or more such building blocks can be cascaded horizontally. A typical horizontal cascading scheme has been shown in Fig. 10. The pattern to be classified is broken down into 16-bit subpatterns. Each of these subpatterns are fed to one of the cascaded building blocks. The *Pseudo-exhaustive Lines* of a block are connected to the *Left Boundary Lines* of the next block in the sequence.

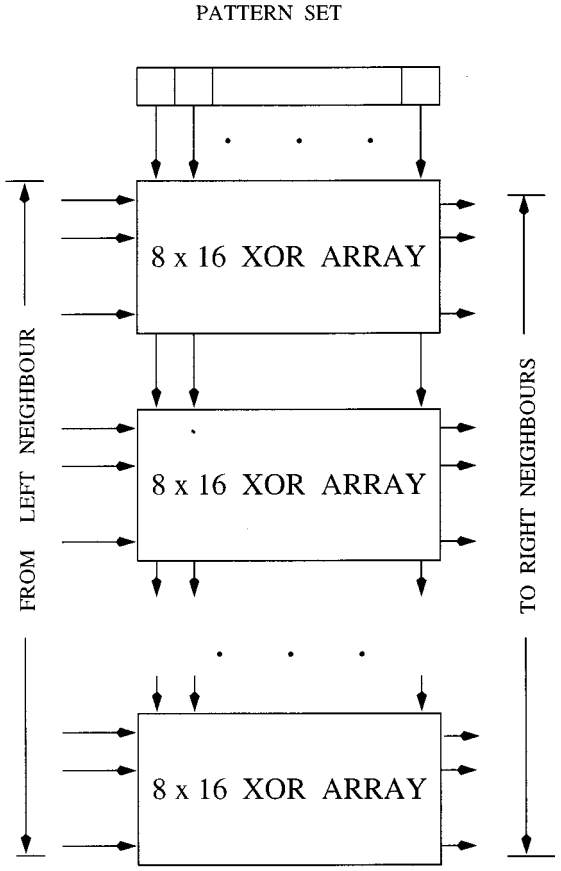

Fig. 11. Vertical cascading of two-class classifiers.

C. Vertical Cascading

Vertical cascading is required when the number of nonzero rows of the T-Matrix is more than eight. In this case, each of the building blocks realizes a subset of nonzero rows of T-Matrix. The pattern to be classified is fed to the *PATT-REG*. A typical vertical cascading scheme has been shown in Fig. 11.

D. A Combined Cascading Scheme

A combined cascading scheme containing both horizontal and vertical cascading has been shown in Fig. 12. This scheme can handle patterns of larger than 16 bits. As has been stated above the pattern is broken down into subpatterns of 16 bits and are applied parallely to the classifiers. Owing to vertical cascading, the classifier presented in Fig. 12 can handle those situations in which the number of nonzero rows of the T-Matrix is more than eight.

VI. Experimental Results

The classification technique has been implemented in C on a Dec Alpha workstation with 175 MHz clock frequency and 32 MB main storage. In order to judge the performance of the CA-based classifier, we have compared the memory requirement of the scheme with the **S**traight **F**orward **A**pproach (referred to as *SFA* in the remaining section) that stores the class of each pattern directly.

Let n_1 and n_2 be the number of patterns in the two classes P_1 and P_2 respectively, to be classified. Each pattern is assumed to be of b bits. In the straight-forward approach it suffices to note the class of all patterns of either P_1 or P_2. This requires $(\min(n_1, n_2) \times b)$ bits. An extra bit is required to determine whether the patterns stored are of P_1 or P_2. Hence, the total memory requirement is

$$M_{\mathrm{SFA}} = \min(n_1, n_2) \times b + 1.$$

On the other hand, let a classifier based on CA possess a_1 attractors containing patterns of P_1 and a_2 attractors containing patterns of P_2. If the number of pseudo-exhaustive bits of the attractors be m, then the total memory requirement is

$$M_{\mathrm{CA}} = \min(a_1, a_2) \times m + 1.$$

Table III notes some of the results for two-class classification on randomly generated data sets. The CA-based approach on an average requires 19.83% lesser memory than the straight-forward technique. For some cases, the saving in memory space is seen to be as high as 65.04%. However, in some situations, CA-based approach does not save any memory. This situation occurs when the patterns are such that the characteristic matrix for the classifier CA becomes equal to the identity matrix (all states are single cycle attractors). That is, the pattern bits cannot be combined effectively using XOR-logic to reduce the required

TABLE III
Results of Classification with CA-Based Two–Class Classifier on Randomly Generated Patterns

Pattern Size	Number of patterns in class-1(n1)	Number of patterns in class-2(n2)	M_{SFA} (bits)	M_{CA} (bits)	$(M_{SFA} - M_{CA})/M_{SFA}$ ×100%
7	18	14	99	79	20.20
	31	37	218	218	0
	50	25	176	176	0
	39	41	274	274	0
	71	30	211	211	0
8	123	23	185	185	0
	21	12	97	61	37.11
	31	9	73	49	32.88
	28	15	122	99	18.85
	17	15	121	106	12.40
9	18	20	163	97	40.49
	16	25	145	61	57.93
	20	30	181	103	43.09
	40	60	361	297	17.73
	60	90	541	541	0
10	15	25	151	217	0
	35	50	351	331	5.70
	60	40	401	352	12.22
	80	60	601	601	0
	60	90	601	601	0
11	15	25	166	118	28.92
	35	50	386	316	18.13
	65	45	496	451	9.07
	85	60	661	601	9.08
	100	100	1101	1101	0
12	15	25	181	127	29.83
	35	50	421	298	29.22
	65	45	541	451	16.64
	85	60	781	661	15.36
	100	100	1201	1097	8.66
13	35	50	456	265	41.89
	65	45	586	379	35.32
	85	60	846	601	28.96
	100	100	1301	1068	17.91
	120	135	1561	1345	13.84
14	35	50	491	265	46.03
	100	100	1401	1079	22.98
	120	110	1541	1178	23.56
	130	140	1821	1549	14.94
	170	200	2381	2353	1.18
15	15	25	226	79	65.04
	50	60	751	415	44.74
	100	100	1501	1068	28.85
	125	110	1651	1178	28.65
	160	140	2101	1795	14.56

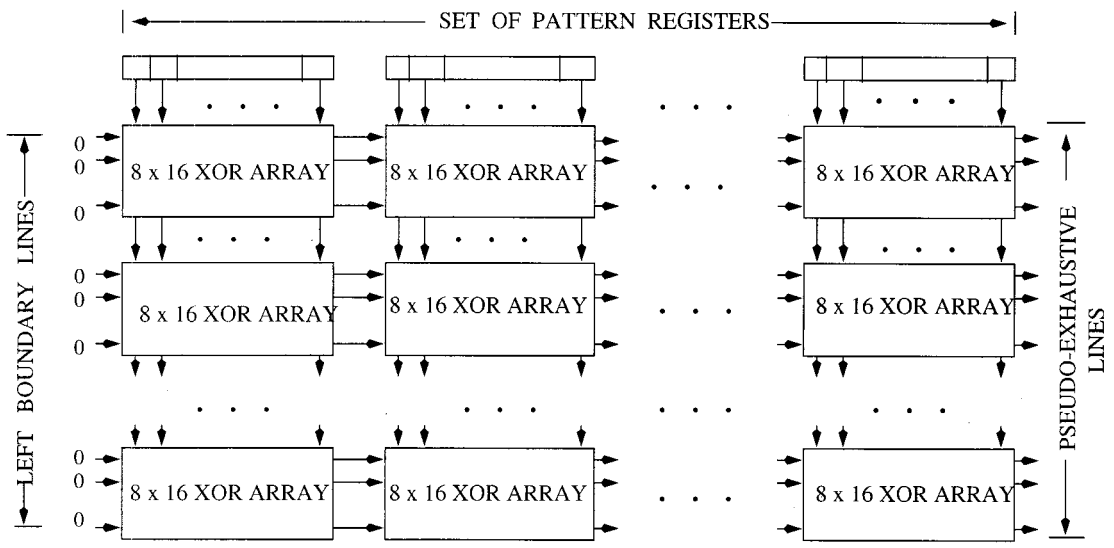

Fig. 12. Combined cascading of two-class classifiers.

TABLE IV
RESULTS OF CLASSIFICATION WITH CA-BASED MULTICLASS CLASSIFIER ON RANDOMLY GENERATED PATTERNS

Pattern Size	Number of patterns(n)	Number of classes(k)	M_{SFA} (bits)	M_{CA} (bits)	$(M_{SFA}-M_{CA})/M_{SFA}$ ×100%
7	95	3	855	577	32.51
	70	4	630	511	18.89
	125	5	1250	876	29.92
	82	7	820	619	24.51
	98	8	980	747	23.78
8	33	3	330	208	36.97
	51	5	561	339	39.57
	110	6	1210	967	20.08
	143	9	1716	1210	29.49
	187	12	2244	1582	29.50
9	507	4	5577	4603	17.46
	65	5	780	503	35.51
	447	6	5364	3993	25.56
	103	7	1236	835	32.44
	303	9	3939	2807	28.74
10	642	3	7707	6453	16.27
	893	5	11609	8985	22.60
	141	6	1833	1322	27.88
	184	7	2392	1840	23.08
	483	10	6762	4935	27.02
11	71	4	923	569	38.35
	175	5	2450	1747	31.22
	440	6	6160	4857	21.15
	168	7	2352	1697	27.85
	431	8	6034	4703	22.06
12	70	3	980	590	39.80
	188	5	2820	2008	28.79
	273	7	4095	2965	27.59
	179	8	2685	1861	30.69
	253	10	4048	2528	37.55
13	103	3	1545	985	36.25
	154	4	2310	1570	32.03
	135	6	2160	1204	44.26
	95	7	1520	797	47.57
	72	5	1152	572	50.35
14	127	4	2032	1192	41.34
	128	6	2176	1210	44.39
	73	5	1241	551	55.60
	119	8	2023	859	57.54
	88	10	1584	693	56.25
15	213	3	3621	2051	43.36
	120	6	2160	1164	46.11
	146	10	2774	1432	48.38
	140	4	2380	1157	51.39
	103	5	1854	901	51.40

number of bits. In this situation, probably a different encoding of pattern features will yield better results.

For multiclass classification, with k classes $P_1, P_2, \cdots, P_k$, having $n_1, n_2, \cdots, n_k$ patterns, respectively, the memory requirement of the straight-forward approach is

$$M_{\text{SFA}} = \sum_{i=1}^{k} n_i \times (b + \lg k).$$

That is to say, it stores all the patterns and their classes. On the other hand the CA-based approach will require memory for the individual Two-Class Classifiers. Since the number of attractors stored for the individual classifiers is not fixed, this information is also to be kept. Hence an additional $(b+1)$ bits will be required each of the k classifiers. Thus the total memory requirement will be

$$M_{\text{CA}} = \sum_{i=1}^{k} (\min(a_{i1}, a_{i2}) \times m_i + b + 1).$$

Table IV notes down the result for multiclass classification. On an average, the CA-based approach requires 34.51% lesser memory than the straight-forward strategy. In some situations, it can show upto 57.54% saving of memory.

VII. SOME PRACTICAL APPLICATIONS OF CA-BASED CLASSIFIER

In this section we present two applications of the CA-based pattern classifier for VLSI circuit testing.

1) *Analog Circuit Fault Detection:* Fault verification and testing of analog circuits can be an important practical application of the CA-based classifier, presented in [23]. As an example, we consider a low-pass filter having transfer function

$$T(s) = \frac{as^2 + bs + c}{ds^2 + es + f}$$

where the parameters a, b, c, d, e, f depends on R, C components of the circuit and the gain G_M. By varying these components of the circuit, the stability of the circuit may be controlled. Consider that $a_{\min}$, $b_{\min}$, $c_{\min}$,

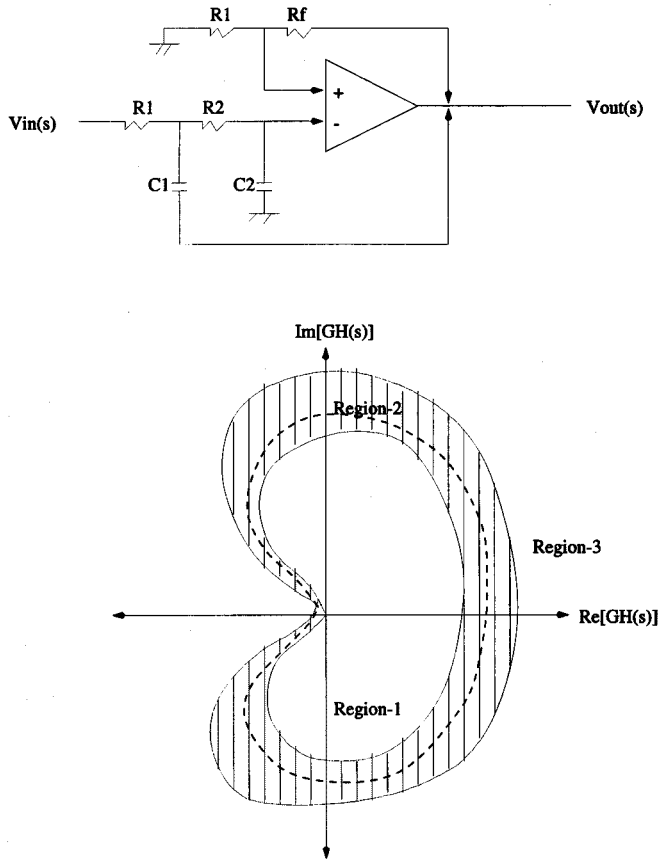

Fig. 13. Nyquist plot of a typical LPF.

$d_{\min}$, $e_{\min}$, and $f_{\min}$ as the minimum values of these parameters above which the circuit remains stable. Also let, $a_{\max}$, $b_{\max}$, $c_{\max}$, $d_{\max}$, $e_{\max}$, and $f_{\max}$ be the maximum values of these parameters below which the circuit remains stable. From the Nyquist plot, as presented in Fig. 13, it may be noted that the circuit is said to be stable if the Nyquist plot for a given set of values of a, b, c, d, e, f lies entirely in the region of the plot where the circuit remains stable. We can thus think of three classes,

- P_1: *Minimally Faulty Class*, comprising of points in region-1.
- P_2: *Non Faulty Class*, comprising of points in region-2.
- P_3: *Maximally Faulty Class*, comprising of points in region-3.

Thus, an analog circuit may be said to be fault free, if *all* the points in its Nyquist plot can be mapped to the stable region, that is to class P_2. It may be noted that as the boundary of the Nyquist plot is highly irregular, so the conventional hardware comparator circuits cannot be employed to solve this classification problem. Hence the CA-based classification approach presented in the current work can prove to be a better alternative.

2 *VLSI Circuit Fault Detection:* To reduce the complexity of testing, VLSI circuits are often partitioned into a set of disjoint/overlapping partitions. Response of the circuit corresponding to the faults in the different partitions are grouped separately, each group forming a class. The process helps in locating a probable faulty section of the circuit. Hence the classification scheme based on CA can be employed, which will point to faulty partition(s), given the faulty response of the circuit [24].

VIII. Conclusion

This paper enumerates an efficient classification scheme based on CA. It saves, on an average, 34.51% memory as compared to the direct storing of pattern classes. The design has been specified in Verilog and simulated for functional correctness. The regular, modular, and cascadable structure of the classifier makes the scheme ideal for VLSI implementation.

References

[1] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[2] J. V. Neuman, *The Theory of Self-Reproducing Automata*, A. W. Burks, Ed. Urbana, IL: Univ. Illinois Press, 1966.
[3] A. W. Burks, "Essays on cellular automata," Univ. Illinois, Urbana, IL, Tech. Rep., 1970.
[4] E. F. Codd, *Cellular Automata*. New York: Academic, 1968.
[5] S. Wolfram, "Statistical mechanics of cellular automata," *Rev. Mod. Phys.*, vol. 55, pp. 601–644, July 1983.
[6] ——, "Random sequence generation by cellular automata," *Adv. Appl. Math.*, pp. 123–169, 1986.
[7] ——, "Computation theory of cellular automata," *Commun. Math. Phys.*, vol. 96, pp. 15–57, 1984.
[8] O. Martin, A. M. Odlyzko, and S. Wolfram, "Algebraic properties of cellular automata," *Comm. Math. Phys.*, vol. 93, pp. 219–258, 1984.
[9] A. K. Das, "Additive cellular automata: Theory and application as a built-in self-test structure," Ph.D. dissertation, I.I.T. Kharagpur, India, 1990.
[10] A. K. Das and P. P. Chaudhuri, "Efficient characterization of cellular automata," *Proc. Inst. Elec. Eng.*, vol. 137, pt. E, pp. 81–87, Jan. 1990.
[11] ——, "Vector space theoretic analysis of additive cellular automata and its applications for pseudo-exhaustive test pattern generation," *IEEE Trans. Comput.*, vol. 42, pp. 340–352, Mar. 1993.
[12] P. D. Hortensius *et al.*, "Cellular automata based pseudo-random number generators for built-in self-test," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 842–859, Aug. 1989.
[13] P. D. Hortensius, R. D. McLeod, and H. C. Card, "Parallel pseudo-random number generation for VLSI systems using cellular automata," *IEEE Trans. Comput.*, vol. 38, pp. 1466–1473, Oct. 1989.
[14] ——, "Cellular automata based signature analysis for built-in self-test," *IEEE Trans. Comput.*, vol. 39, pp. 1273–1283, Oct. 1990.
[15] S. Nandi and P. P. Chaudhuri, "Additive cellular automata as on-chip test pattern generator," in *Proc. 2nd Asian Test Symp.*, Nov. 1993.
[16] A. K. Das, D. Saha, A. R. Chowdhury, S. Misra, and P. P. Chaudhuri, "Signature analyzer based on additive cellular automata," in *Proc. 20th Fault Tolerant Computing Systems*, U.K., June 1990, pp. 265–272.
[17] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller, "Analysis of one dimensional cellular automata and their aliasing probabilities," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 767–778, July 1990.
[18] D. R. Chowdhury, S. Basu, I. S. Gupta, and P. P. Chaudhuri, "Design of CAECC—Cellular automata based error correcting code," *IEEE Trans. Comput.*, vol. 43, pp. 759–764, June 1994.
[19] S. Nandi, B. K. Kar, and P. P. Chaudhuri, "Theory and application of cellular automata in cryptography," *IEEE Trans. Comput.*, vol. 43, Dec. 1994.
[20] D. Chowdhury, S. Chakraborty, B. Vamsi, and P. Chaudhuri, "Cellular automata based synthesis of easily and fully testable FSMs," in *Proc. ICCAD*, Nov. 1993, pp. 650–653.
[21] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata Theory and Applications: Vol. 1*. New York: IEEE Computer Society, 1997.
[22] D. R. Chowdhury, S. Basu, I. S. Gupta, and P. P. Chaudhuri, "Encoding and decoding of error correcting codes using cellular automata," in *Proc. VLSI Design*, Jan. 1992, pp. 133–136.

[23] K. Paul, A. Roy, P. Nandi, B. Roy, M. Purkayastha, S. Chattopadhyay, and P. P. Chaudhuri, "Theroy and application of multiple attractor cellular automata for fault diagnosis," in *Asian Test Symp.*, 1998.

[24] S. Nandi, S. Chattopadhyay, and P. P. Chaudhuri, "Theroy and application of cellular automata for fault diagnosis in VLSI circuits," in *Proc. VLSI*, Jan. 1996.

Santanu Chattopadhyay received the B.E. degree in computer science and technology from Calcutta University, Calcutta, India, in 1990 and the M.Tech. and Ph.D. degrees in computer science and engineering from Indian Institute of Technology, Kharagpur, in 1992 and 1996, respectively.

From July 1995 to February 1999, he was associated with Bengal Engineering College (Deemed University), Howrah, as a Lecturer in the Department of Computer Science and Technology. From February 1999 to May 2000, he was Computer Networking Manager at the Indian Institute of Technology, Kharagpur. In May 2000, he joined the Indian Institute of Technology, Guwahati, as an Assistant Professor in the Department of Computer Science and Engineering. He also coauthored the book *Additive Cellular Automata—Theory and Applications, Volume I* (New York: IEEE Computer Society). His research interests include theory and applications of cellular automata, logic design, high level synthesis, and circuit testing.

Shelly Adhikari received the B.E. degree in computer science and technology from Bengal Engineering College (Deemed University), Howrah, in 1998.

He is currently a Design Engineer working on EDA Projects at Delsoft India Pvt. Ltd., Uttar Pradesh, India. For a short time, he was with Kawasaki Research and Engineering Center, Fujitsu Ltd., Japan, where he was involved in the design of a 0.18-μm/500-MHz 64-bit RISC processor. His research interests include cellular automata and VLSI design.

Sabyasachi Sengupta received the B.E. degree in computer science and technology from Bengal Engineering College (Deemed University), Howrah, in 1998.

He is currently a System Engineer working on Sequennt's Dynix/ptx operating system at Wipro Technologies, Bangalore, India. His research interests include cellular automata and design of operating system.

Mahua Pal received the B.E. degree in computer science and technology from Bengal Engineering College (Deemed University), Howrah, in 1998.

She is currently a System Engineer working on knowledge in networking domain at Wipro Infotech, Calcutta, India. Her research interests include cellular automata, VLSI design, artificial intelligence, and neural network.