Testability Analysis and Behavioral Testing of the Hopfield Neural Paradigm

C. Alippi, Franco Fummi, V. Piuri, M. Sami, and Donatella Sciuto

***Abstract*—Testability analysis and test pattern generation for neural architectures can be performed at a very high abstraction level on the computational paradigm. In this paper, we consider the case of Hopfield's networks, as the simplest example of networks with feedback loops. A behavioral error model based on finite-state machines (FSM's) is introduced. Conditions for controllability, observability and global testability are derived to verify errors excitation and propagation to outputs. The proposed behavioral test pattern generator creates the minimum length test sequence for any digital implementation.**

***Index Terms*—FSM, functional TPG, neural network.**

I. Introduction

INCREASING complexity of VLSI systems has accelerated a trend to take into account testability and test generation throughout the synthesis process, since the highest abstraction levels [5]. This involves defining testability parameters, error models and test patterns based on technology-independent, purely *behavioral* information, where by "behavior" we denote the input–output mapping affected by the system, excluding any implication of the system's architecture or structure. Such high-level approaches are advocated as allowing compact system description and simpler test pattern generation algorithms. On the other hand, validity of behavioral approaches can be finally assessed only by evaluating the structural fault coverage achieved.

An interesting case concerns VLSI implementations of artificial neural networks (ANN's). Although a large number of silicon solutions have been proposed, testing issues have mostly been examined in the context of implementation rather than with reference to the algorithm implemented. Behavioral analysis of feed-forward neural paradigm has been discussed in [9], while impact of faults on the network behavior has been analyzed in [8]. In this paper, we present a solution for functional test pattern generation and we examine its validity with respect to actual coverage of structural faults.

The standard Hopfield network [7] consists of a layer of N recurrent neurons. In vector notation, neurons evaluate their own output as

$$X(t) = f(WX(t-1) + \Theta) \tag{1}$$

where $X(t-1)$ is the output (or *state*) vector of the neurons at the previous iteration, W is the $N \times N$ interconnection (synoptic) weight matrix, Θ is the bias vector and f the nonlinear activation function. We restrict the possible values assumed by the components of X to $\{0, 1\}$. The network evolves autonomously from the externally forced initial state (specified by external inputs) to a stable state defined as an *attractor*. During the learning phase, weights are determined with reference to a set of *desired* attractors but a number of *spurious* attractors may arise as well. Two different recall procedures have been presented in literature [7] as follows.

- *Parallel-Type:* Initial inputs are set simultaneously in all neurons, and the network evolves toward the final steady state. At each iteration, neurons outputs are simultaneously updated.
- *Serial-Type:* Initial inputs are set in all neurons simultaneously and subsequent neurons' state update is performed one neuron at a time in a given ordering (usually, circular).

We assume that learning has been perfected. In Section II we present a higher abstraction model of network behavior based on the finite-state machine (FSM) model. In Section III, testability is evaluated based on this model. In Section IV, a functional test approach is developed based on a well-known functional error model introduced for FSM's [9], on related test pattern definition and on implicit [2] techniques.

II. FSM Modeling of the Hopfield Paradigms

A Hopfield network can be modeled by a Moore-type FSM $\langle S, I, O, \delta, \lambda \rangle$ where we have the following.

- S Set of states of the machine (states $s \in S$ being coded as vectors X of the Hopfield network).
- I Input alphabet defined as the set of input vectors used to force initial states. After the initial setting the network evolves autonomously toward an attractor, thus the input alphabet is not involved in the definition of next-state function δ.
- O Output alphabet, $O = S$.
- δ δ: $S \rightarrow S$ for any state $s(t) \rightarrow S$ evaluates the next state $s(t+1)$ based on (1). Evaluation may either render $s(t) = s(t+1)$ (i.e., an attractor has been reached and the computation stops), or $s(t) \neq s(t+1)$ [(1) is applied again].
- λ λ: $S \rightarrow O$ is identical to δ.

A. The Parallel-Recall Hopfield Paradigm

Given any initial input vector coded by $i_k \in I$, the network's operation is deterministic, so that the associated FSM will evolve through a sequence of states $s_k^0, s_k^1, \cdots, s_k^a$, where s_k^a is the attractor associated with i_k. If during learning orthogonal patterns have been used, only desired attractors will be present; otherwise, spurious attractors may appear in the system's behavior.

Considering as an example the four-neuron Hopfield network characterized by weight matrix W in Fig. 1(a), application of $i_k = 0010$ induces the state sequence 0010, 1111, 1110 (steady state), represented by an oriented path in the state

Manuscript received June 29, 1995; revised September 30, 1997.

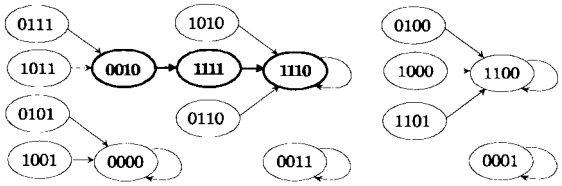
The authors are with the Department of Electronics and Information, Politecnico di Milano, Milano I-20133 Italy.

Publisher Item Identifier S 1063-8210(98)06000-4.

1063–8210/98$10.00 © 1998 IEEE

$$W = \begin{bmatrix} 2 & 2 & 0 & -2 \\ 2 & 2 & 0 & -2 \\ 0 & 0 & 2 & 0 \\ -2 & -2 & 0 & 2 \end{bmatrix}$$

(a)



(b)

Fig. 1. (a) Weight matrix of a Hopfield network and (b) its state graph.

graph of the equivalent FSM. Thus, the state transition graph of the equivalent FSM is acyclic, apart from self-loops on the attractor state.

Whenever the input pattern applied i_m coincides with an intermediate state s_k^i of a previously identified path, the state sequence created starting from s_k^i coincides with the subsequence $s_k^i, s_k^{i+1}, \cdots, s_k^a$ of the complete path (in our example, this happens if the input pattern is 1111).

If a different input pattern i_j is applied leading to the same attractor s_k^a, the path associated with such input will either converge with the previous one in the final state, or else share with it a final segment (obviously including the final state). For example, for attractor 1110, the first case is achieved starting from input state 1010, while the second one is generated by input state 1011. By applying all input patterns leading to the same attractor s_k^a, a directed acyclic subgraph is derived containing all and only the initial states sharing s_k^a as attractor. An example is the subgraph whose states converge to steady state 1110 in Fig. 1. This subgraph is similar to a n-ary tree; however, the oriented paths are not directed from the root (in our case, the attractor 1110) to the leaves, but in the reverse direction. Similar subgraphs are created for each attractor. The complete state transition graph of the FSM is the union of all above subgraphs and resemble a forest [see Fig. 1(b)].

This basic characteristic does not change even in the presence of spurious attractors. When only expected attractors are present in the neural network, the set of possible states is partitioned into disjoint subsets each associated with an attractor. Similarly, the state graph is partitioned into disjoint subgraphs: each of them is associated with an attractor and contains all and only the initial states leading to such attractor. Each possible state belongs to exactly one subgraph. When spurious attractors occur, the set of states is again partitioned into disjoint sets: in Fig. 1(b), only 1110 and 1100 are expected attractors while 0000, 0001, and 0011 are spurious ones. States not associated with expected attractors are partitioned into subsets associated with the spurious attractors. Where testing is concerned, spurious attractors and the related subgraphs will be treated in principle just as expected attractors and the related subgraphs.

All the above assumes that the whole input space of the Hopfield network is meaningful, i.e., that all 2^N binary configurations over X can be forced as initial states. The associated FSM then contains exactly 2^N states. If, on the contrary, the input space is partitioned into a set of feasible patterns I_A and a set of unacceptable patterns I_U, states associated with patterns in I_A may appear in the state graph in any position (either as leaves, nodes internal to a path or attractors). States associated with patterns in I_U never appear as leaves or roots, they will either be internal to a path or not appear at all in the state graph, which may therefore consist of less than 2^N states.

B. The Serial-Recall Hopfield Paradigm

Given an initial vector, the states of the N neurons are updated one at a time in a predetermined order so that the machine evolves through a sequence of N "intermediate" states. At the end of the sequence, a new "main" state is reached (all neurons have been updated) and the first neuron of the sequence is again ready to fire. The operation is repeated until an attractor is reached. We can see superposition of *two* synchronisms: a fine-grained one controlling firing of individual neurons within the sequence and a main one activating a new sequence. When an attractor is reached, the corresponding N intermediate states form a cycle. This mode of operation requires a slightly more complex FSM model. The FSM associated with the neural network contains N memory elements whose states are initially set to the input vector; a *controller* specifies—at each secondary clock cycle—which memory element must be updated. The controlled FSM produces a new state on the basis of the present state and of the information provided by the controller; an output vector is read only when a main state is reached.

In the equivalent FSM state diagram, states are defined as for the parallel-recall case, while transitions from one state to the next one are marked with the ordering number of the firing neuron. To describe creation of the state diagram, we refer to a simple example defined by

$$W = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 3 \\ -1 & -1 & 3 & 0 \end{bmatrix} \quad \Theta = \begin{bmatrix} 1.5 \\ 1.5 \\ -0.5 \\ -0.5 \end{bmatrix}. \tag{2}$$

The network has one true attractor, namely 1100 and the spurious attractor 0011. To derive the equivalent state diagram, we first choose the firing order for the neurons, e.g., the natural ordering from neuron 1 to neuron 4 (left to right). We set an initial input configuration (e.g., 0000) and enable the first neuron to fire; state 1000 (reached by an edge marked 1) is obtained; from this, firing of neuron 2 produces outputs 1100, firing of neuron 3 produces 1100 and that of neuron 4 outputs 1100. We know *a priori* that this is an attractor state, so that no further application of (1) is required; in general an attractor is identified whenever a cycle is generated by a firing sequence of the N neurons. Starting now from initial configuration 1000, firing of neuron 1 leads to state 1000, already present in the graph (the two states are marked with the same coding and are reached by firing of the same neuron). Thus, it is not necessary to pursue further the path, since confluence into the previous one is achieved. By iterating this procedure we end with the state diagram of Fig. 2 ("main" states are identified

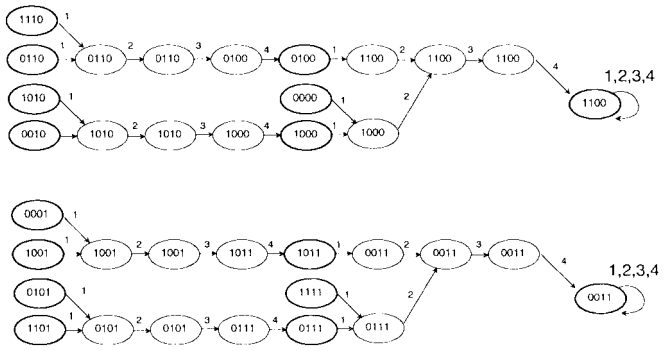


Fig. 2. Equivalent state diagram of the Hopfield network (2).

by a thick contour, "intermediate" ones by a thin one). This is actually the state diagram of the *controlled* FSM. Under the assumption that the firing order is kept unmodified throughout the network's operation, the state diagram is a disjoint graph. Each of these subgraphs is acyclic, excepting for the N-state cycle corresponding to the attractor.

III. Testability Analysis

A system's testability issue can be summarized by evaluation of *controllability* and *observability*. By *controllability* we denote the possibility of propagating arbitrary test patterns from the system's inputs to the component whose possible fault must be tested. By *observability* we denote the possibility of propagating the results produced by the component under test up to the system's outputs. Assessment of a system's behavioral testability allows the estimation of the gate-level fault coverage before logic synthesis is performed. If such a value, which represents an upper bound, is not satisfactory, suitable design guidelines can be adopted from the initial design steps thus avoiding iterations of the entire synthesis flow.

In the case of Hopfield networks, behavioral testability can be seen as the possibility of forcing the network into an arbitrary state and then verifying the correct transition to its next state. In the parallel-recall case, an arbitrary state can be applied as input configuration, i.e., the machine is completely controllable. In the serial-recall case procedure, if we assume that the controller is provided with a reset signal forcing it to its initial state, the machine is controllable as well. Therefore, controllability is always granted and independent of the adopted recall procedure. In the case of the parallel-recall procedure the state is directly read at the network outputs, so that the machine is completely observable. In case of the serial-recall procedure, observability holds for the controlled FSM, while the controller outputs cannot be directly propagated to the network outputs. However, a fault in the controller modifies the firing sequence of the neurons, which is observable at the outputs after a given time latency, as it will be shown for the test generation procedure. Therefore, also serial-recall networks are observable. Thus, any Hopfield network affords complete behavioral testability. Actual testability will then depend on the architectural and technological solutions chosen for machine implementation; behavioral testability constitutes an upper bound for lower-level (e.g., gate-level) testability (it is respected in particular if one-to-one mapping of operators onto components is adopted).

IV. Test Pattern Generation

For FSM functional test, the *single-state transition fault* model assumes that a fault generates an error which can affect one transition only, producing either a faulty state or a faulty output or both. It is assumed that the number of states does not increase as a consequence of a fault. In our specific case, since the FSM model describes all possible states of the neural network, we assume that gate-level faults cannot introduce new memory elements. In the most general case, functional testing of a FSM requires verification of all transitions of the machine. For each transition a test sequence must be identified, composed of three subsequences: a *justification sequence* driving the machine from a known state into the source state of the transition to be tested; the *input symbol* activating the transition and a *distinguishability sequence* distinguishing the correct destination state from any other (faulty) one. Unique input–output (UIO) sequences have been selected to discriminate the correct state from any other state of the machine [10]; it might be recalled that not all FSM's have UIO's for each state [11].

The FSM model of a Hopfield network allows a simplification of the test generation process. Each state of the FSM is characterized by an UIO sequence; parallel-recall based networks have UIO's of unit length since each state shows a different output value, while for serial-recall based networks, the UIO can be at most of length N, since it corresponds to the path from the given state to the next main state. In the following, we will illustrate the test pattern generation approach for serial-recall networks since it is more complex; when appropriate, we will show differences between the two recall schemes.

The test sequence for a transition outgoing from a main state does not require any justification sequence since the machine can be directly set to such a state. For transitions outgoing from intermediate states, the justification sequence consists of the input sequence which drives the FSM from a main state to the source state of the transition under test. After application of the transition under test, the UIO for its next state is concatenated. The UIO for a main state consists of the sequence leading to the next main state; there is no need for an explicit test sequence for transitions outgoing from intermediate states since they are implicitly tested by the test sequence of the transitions from the main states. For example, starting from 0110 (Fig. 2), in eight clock cycles (two complete firing sequences) attractor 1100 is reached; all transitions in the path are tested, since each of them is concatenated with its UIO sequence. Since any transition applied in the test sequence is always followed by its UIO sequence, that verifies its next state, it is assured that the path starting from a main state and reaching the next main state is always correct [3]. Start now from initial state 1110: firing of the first neuron leads to an already visited state (0110, reached by firing of neuron 1), so that it is not necessary to pursue the sequence until the attractor, but only for a length of four

states (the UIO). Furthermore, there is no need to create a test sequence from 0100, since it has been already explored, and thus tested, by activating the first path starting from state 0110. By applying path analysis, the FSM in Fig. 2 is completely tested by a set of sequences requiring a total of 56 clock cycles (instead of 88 cycles necessary to explore all paths).

The test generation procedure aims at identifying the minimum subset of initial states such that all transitions of the machine will be activated. To check all transitions, it is necessary to explore all paths of the FSM, starting from all possible initial states and terminating when the set of explored states (always considered after N steps, to account for completion of the serial-recall sequence) coincides with the set of the machine's states. The test generation algorithm performs such FSM traversal in an implicit way to identify the optimal test sequence; to reduce the test generation complexity, a symbolic exploration must be performed.

A. Implicit Implementation of FSM's

We assume that each element of alphabet I is a vector encoded by n Boolean variables $x_1 \cdots x_n$, (*input variables*). Similarly, the present state is encoded by k Boolean variables $s_1 \cdots s_k$ (*present state variables*), the output vector by m Boolean variables $z_1 \cdots z_m$ (*output variables*), and next state by k Boolean variables $t_1 \cdots t_k$ (*next state variables*). The transition induced by function δ can be expressed by its characteristic function and efficiently represented with binary decision diagrams (BDD's) [2]. Moreover, there are well known techniques [12] to enumerate all states of a FSM by implicitly traversing its transition relation. Unfortunately, classical traversal techniques cannot be employed in the case of Hopfield networks; the FSM representation here is not based on a transition relation but it is given in the form of a vector-matrix product and a threshold function, elements of the weight matrix and of the threshold vector being integer numbers instead of Boolean values. Integer and matrix operations with BDD's have been defined in [6] to face the problem of technology mapping.

Let $D_n\{-2^{n-1}, \cdots, 0, \cdots, 2^{n-1}-1\}$ be the set of integer numbers which can be represented with $n+1$ bits and f be the function mapping Boolean vectors of length m onto D_n; it is $f(\overline{x}) = \sum_{i=0}^{n} f_i(\overline{x})2^i$ where each f_i has value "0" or "1" and is represented as a BDD, and the vector symbol on x emphasizes that each integer number requires a vector of Boolean values to be represented to.

Arithmetic operations such integer valued functions can be implemented in terms of logical operation on BDD's. A $2k \times 2l$ matrix M over D_n can be represented as a n integer valued function M: $B_{k+l} \rightarrow D_n$ such that $M_{ij} = M(\overline{x}, \overline{y})$ where $\overline{x}$ represents the bit vector corresponding to i and $\overline{y}$ the bit vector for j. Matrices composed of integer values can then be represented as integer valued functions and thus implemented with arrays of BDD's. Moreover, matrix operations can be implemented in terms of logical operations on BDD's since they are based on the mathematical operations of addition and multiplication which can be implemented with BDD-based operations [6].

```
Unreachable_States(W[0:m], D[0:m], n)
{
   Input = CreateInput(n);
   Reached = ∅;
   Previous_Reached = not ∅;
/* repeat until no new main states are reached */
   while (Reached ≠ Previous_Reached){
      /* computes the next main states */
      for (i = 0; i < n; i++){
         Net_Next_State[0:m] = W[0:m] × Input;
         Next_State[0:m]=Net_Next_State[0:m]-D[0:m];
         Input = not Next_State[m];
      }
      Previous_Reached = Reached;
      Reached = Reached ∪ Input;
   }
   return(not Reached);
}
```

Fig. 3. Algorithm for the unreachable states identification.

B. Test Generation Algorithm

The proposed testing methodology consists of the following.

1) Identification of the set of unreachable states (US), composed of those main states which are not reachable from other main states. Test generation will be performed starting from this set of states since in this case the longest paths to the attractors will be obtained. Considering the example in Fig. 2, US $= \{1110, 0110, 1010, 0010, 0001, 1001, 0101, 1101\}$.

2) Computation of the test length for each main state in US is as follows:

   - for each main state $u \in$ US determine the main states traversed from u to the attractor;
   - for each pair of main states u_1 and u_2 which share a traversed main state m, assign as test sequence length for u_1 the entire length until the attractor, and as test sequence length for u_2 the length of the path from u_2 to m.

For instance, states 1110 and 0110 in Fig. 2 share the traversed main state 0100 in the path reaching attractor 1100. A test sequence of length $2N$ is associated with state 1110 and a test sequence of length N is associated with state 0110.

Identification of unreachable states without explicit representation of the FSM is implemented by the procedure Unreachable_States in Fig. 3. Such a procedure requires as inputs the BDD representation of the weight matrix W (the vector of BDD's $W[0:m]$), the threshold vector (the vector of BDD's $D[0:m]$) and the number of BDD's necessary for integer representation (n); it returns the BDD representation of the unreached states.

Complexity of the algorithm is proportional to the length of the longest path starting from an initial state and arriving to its attractor. The same algorithm holds for parallel-recall networks, where only one operation, instead of n, is necessary to compute the next reachable states. The result of the Unreachable_States procedure is a characteristic function, expressed through a BDD. Each minterm belonging to such characteristic function is extracted and included in the set US.

To perform the second step the number of states in US must be enumeratable. This is not restrictive since these states constitute the test sequence that must have finite size to be of practical use. Main requirement of this second phase of the testing approach is the storage of the already explored states in order to interrupt the test generation. An already explored state is represented by its n-bit output configuration and the additional information on the ordering number of the neuron whose firing produced the transition to the state. Given n neurons, this information requires $m = \log_2 n$ further bits. In case of a parallel-recall based network, these last m bits are not required.

For example, the reset state 1110 (Fig. 2) is represented by configuration 111000, where the last two bits represent the code of the first firing neuron. Therefore, a BDD $S(s, m)$ is generated by adding all states (both intermediate and main states) reached by states in US.

Initially, S is composed of a single node (the FALSE node) that represents the empty set. Starting from the current main state s in US the next state t is generated by the product of the weight matrix with the main state and by applying the nonlinear function over the result. If t already belongs to S the test sequence for s is interrupted, otherwise the reached state t is added to S and the exploration continues. The path examination is concluded when the attractor is reached, verification of the inclusion of t in S is performed by the *intersect* operation [2], whose complexity is linear with the number of nodes of the smaller BDD involved (in this case the BDD representation of t). Since the number of nodes of such a BDD is always at most $n + m$, the above verification is performed in a very efficient way. BDD's are used in this step of the test generation procedure to efficiently check the inclusion of a newly generated next state into the set of already explored states.

C. Experimental Results

The above methodology for test generation has been implemented in the program *HFunTest*, written in C^{++} on a Sun SparcStation10, based on the BDD library of CMU [1]. Experiments have been performed to validate the methodology on a set of Hopfield networks with different numbers of neurons. Each network (weight matrix and threshold vector) has been converted into a VHDL description input to an industrial synthesis tool to produce the gate-level description of the network (net-list). At the same time, *HFunTest* converts the network characterization into a BDD description and generates the test sequence. Finally, both net-list and test sequence are simulated by a sequential fault simulator to verify the stuck-at fault coverage.

In Table I circuits are characterized in terms of the number of neurons, gates, memory-elements and stuck-at faults. The following two columns of the same table report the length of the produced test sequence and the achieved stuck-at fault coverage. Since fault coverage includes detectable and undetectable faults, it is not an effective measure of the goodness of the proposed test methodology. Thus, the two smallest circuits have been analyzed with a redundancies removal program (Veritas [4]) to produce irredundant network implementations. On such circuits, the achieved fault coverage may be seen as fault efficiency (last column) and it better highlights the effectiveness of the proposed testing methodology.

TABLE I
EXPERIMENTAL RESULTS

name	#neu.	#gates	#F.F.	#s-a f.	T.L.	%F.C.	%F.E.
hopf1	4	110	6	258	126	81.8	98.5
hopf2	8	225	11	520	362	81.9	98.6
hopf3	24	692	29	1594	1330	81.3	-
hopf4	32	930	37	2120	1840	81.1	-

Due to their huge size, the largest proposed circuits cannot be analyzed by a gate-level test pattern generator even if it is based on implicit FSM traversal techniques while they can be tackled by the proposed testing methodology.

REFERENCES

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. IEEE DAC*, June 1990.
[2] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, pp. 79–85, Aug. 1986.
[3] G. Buonanno, F. Fummi, D. Sciuto, and F. Lombardi, "FsmTest: Functional test generator for sequential circuits," in *INTEGRATION, The VLSI Journal*. New York: Elsevier Science, 1996, vol. 20.3, pp. 303–325.
[4] H. Cho, S. Jeong, and F. Somenzi, "Synchronizing sequences and symbolic traversal techniques in test generation," *JETTA*, no. 4, pp. 19–31, Apr. 1993.
[5] V. Chickermane, J. Lee, and J. H. Patel, "Addressing design for testability at the architectural level," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 920–934, July 1994.
[6] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. C. Y. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," in *Proc. IEEE DAC*, June 1993.
[7] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1991.
[8] C. Alippi, V. Piuri, and M. Sami, "Sensitivity to errors in artificial neural networks: A behavioral approach," *IEEE Trans. Circuits Syst.*, vol. 42, June 1995.
[9] V. Piuri, M. G. Sami, and D. Sciuto, "Testability of artificial neural networks: A behavioral approach," in *The Journal Electronic Testing: Theory and Applications*. Kluwer Academic, 1995.
[10] K. K. Sabnani and A. T. Dahbura, "A protocol test generation procedure," *Comput. Networks*, vol. 13, no. 4, pp. 285–297, 1987.
[11] Y. N. Shen, F. Lombardi, and A. T. Dahbura, "Protocol conformance testing by multiple UIO sequences," *Protocol Specification, Testing and Verification*, vol. IX, pp. 131–143, 1990.
[12] H. J. Touati, H. Savoj, B. Lin, R. Brayton, and A. Sangiovanni-Vincentelli, "Implicit state enumeration of finite state machines using BDD's," in *Proc. IEEE ICCAD*, 1988, pp. 130–133.