Behavioral Testability and Test Pattern Generation of the Hopfield Network Model

Cesare Alippi, Franco Fummi, Vincenzo Piuri, Mariagiovanna Sami, Donatella Sciuto
Dipartimento di Elettronica e Informazione, Politecnico di Milano
piazza L. da Vinci 32, I-20133 Milano, Italy

Abstract

The problem of testability and test pattern generation at the highest abstraction level, i.e., based on the network's behavior, is here considered for Hopfield networks. Complete testability is proved. A test pattern generation approach based on creation of an equivalent Finite State Machine is presented, functional test pattern generation having being proved to allow very high coverage of logic-level faults in the case of FSMs. An efficient algorithm, using BDDs, is finally described.

1. Introduction

Recent literature has made evident a trend towards qualitative evaluation of system testability at the highest, i.e. behavioral, abstraction model. Such an approach can provide guidelines for choice of subsequent implementation alternatives; recent published examples are provided, e.g. by [1, 2] and, in the neural network domain, by [3]. Even more relevant efforts have been dedicated to the problem of functional-level test pattern generation (usually, the term "behavioral" is in this case adopted only for microprocessors). The justification of such trend can be found both in the extreme complexity of VLSI devices, making conventional techniques unmanageable, and in the possibility of creating test patterns as far as possible independent of the final silicon implementation. In particular, the functional test generation approach has been proved very rewarding in the case of Finite-State Machines; test patterns derived for a purely functional error model and on the basis of the state table only have experimentally given very high fault coverage for different types of implementation technologies (e.g., random logic, PLA) and for related fault models [4, 5, 6].

In the present paper, the functional approach is adopted both to evaluate testability and to provide efficient test pattern generation for Hopfield networks. We start from the standard Hopfield paradigm definition [7]:

$$x_i(t) = f_i\left(\Sigma_{j=1}^{N}\ w_{ij}\, x_j(t-1) - \theta_i\right) \qquad (1)$$

where $x_i(t)$ is the output of the i-th neuron at time t, $x_j(t-1)$ the output of the j-th neuron at the previous iteration, w_{ij} is the interconnection weight between neurons i and j, θ_i is the bias of neuron i, and f_i is the non-linear activation function (usually taken as the signum or step function).

We assume that learning has been perfected, so the values of the weights w_{ij} are determined and the network is able to reach a steady output for any input pattern. Without loss of generality, we consider that the possible values of x_i are only $\{0, 1\}$. We restrict our present analysis to the parallel-type recalling procedure of the Hopfield networks: the initial input $x_i(t_0)$ is set in each neuron simultaneously; the network evolves towards the final steady state so that, at each iteration, the output updating is performed within each neuron simultaneously.

In our analysis, we distinguish between errors in the network's outputs due to faults and errors which are intrinsic in the neural paradigm due to incomplete classification and generalization capabilities provided by the actual learning procedure. In the presence of the second class of output errors, even if they are related to an undesired neural behavior, testability and testing can be considered as in the desired behavior since the undesired behavior is part of the whole nominal behavior generated by the adopted learning procedure. Therefore, we concentrate our attention on the identification of erroneous outputs due to faults.

No assumption is made on the implementation technology as well as on the supporting architecture, excepting for the request of explicit synchronization with respect to a time base.

The first problem in testing is the preliminary evaluation of the *testability* of the whole system at behavioral level [1, 2], i.e., the possibility of providing and propagating the necessary test patterns for error

0-7803-1901-X/94 $4.00 ©1994 IEEE

excitation (*controllability*) and the possibility of observing the results generated by the circuit under test (*observability*). In [3], we adopted a behavioral approach at the abstraction level of the neural operators for the case of multi-layered feed-forward networks. In Section 2, we discuss this figure of merit of the neural paradigm for Hopfield networks, i.e., in the presence of feedback loops.

A subsequent, even more important problem, is that of test pattern generation. Hopfield networks of large dimensions have already been implemented [8]; moreover, such different implementation approaches as analog, digital and mixed solutions have been presented in the literature. A low level test generation approach would involve separate test pattern generation for each implementation alternative; moreover, such procedure would be very complex both for a digital implementation (the Hopfield network is a sequential one) and, even more, for an analog implementation.

In Section 3, we introduce an FSM modeling approach for Hopfield networks. It will be seen that such model allows a very efficient test pattern generation with reference to the "single-transition" fault model adopted for FSM functional testing [9]. In fact, complete coverage can be reached by a procedure whose complexity is linear with the number of states. Thus, we achieve test patterns valid for any implementation satisfying to the above recalled restrictions (parallel recall, explicitly synchronous operation). Section 4 presents a detailed description of the behavioral test procedure generation.

2. Testability analysis

The testability issue for any system can be summarized as the possibility of total *controllability* and *observability*, where by controllability we denote the possibility of propagating arbitrary test patterns from the system's inputs to the component whose possible error must be tested, while by observability we denote the possibility of propagating the results produced by the component under test up to the system's outputs.

In the case of the Hopfield network, *behavioral testability* can be seen as the possibility of forcing the network into an arbitrary state and of verifying the correct transition to its next state. By nature, an arbitrary state can be applied as an input configuration, i.e., the machine is completely controllable.

In the same way, the outcome of the transition, i.e., the next state, can be directly read on the network outputs, so that the machine is completely observable.

As a consequence, ***any Hopfield network affords complete behavioral testability***. Actual testability will then depend on the architectural and technological solutions; 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).

3. FSM modeling of the Hopfield network

The classical definition of a finite state machine M is a quintuple $<S, I, O, \delta, \lambda>$, where S is the non-void and finite set of states, I is the finite input alphabet, δ is the next state function defined as δ: SxI $\rightarrow$ S and λ is the output function, defined as λ: S $\rightarrow$ O (we adopt here the Moore model). Operation of an FSM at the *functional* level is best described by a state diagram or (which is equivalent) a state table.

Gate-level testing of FSMs involves relevant difficulties in test pattern generation [11]; on the other hand, recent papers have proved that test approaches based on a *functional error model* - namely, the so-called ***single-transition fault model*** - and developing test patterns on the basis of the information provided by the state diagram only actually grant very good fault coverage even for lower-level (i.e., logic level) faults. We will now prove, first of all, that the Hopfield network is well representable by an equivalent FSM, and then we develop an FSM-based functional test generation approach valid for Hopfield networks.

Description of the Hopfield paradigm by using a FSM can be performed by introducing the following mapping between neural and FSM entities.

Each state of the Hopfield network, i.e., each output pattern x, is associated with one state s of the FSM; the output x_i of the i-th neuron coincides with the i-th bit of the state coding s.

The output alphabet O coincides with the set of states S. The input alphabet I can actually be taken to be the void set; in fact, operation of the Hopfield net proceeds by forcing an initial state (coinciding with the input pattern) and then having the network evolve in an *autonomous* way (without presenting any further input patterns), controlled by the synchronization signal only, until a steady state is reached. This justifies our assumption of a void input alphabet to the associated FSM model.

As for the next state function δ, defined now as δ: S $\rightarrow$ S, given any state s_i, the next state s_j is computed by application of equation (1). Two instances may occur; either $s_j \neq s_i$, and equation (1) is applied again at the

next step, or $s_j=s_i$, i.e., a steady state has been reached and computation stops until a new pattern is presented.
Considering the operation of a Hopfield network whose weight matrix W has been perfected and for which *parallel* recall approach is adopted, given any input pattern I_i^* the network operates in a deterministic way, so that the associated FSM will evolve through a sequence of states $s_j \equiv I_i^*$, s_i^1, s_i^2, ... s_i^a, s_i^a being the steady state associated I_i^*.
The output function λ is the identity function since the Hopfield output coincides with the state; therefore, the output alphabet is identical to the set of state coding.

When learning has been perfected, an n-input Hopfield network presents a number of *attracting states* (*attractors*), i.e. the steady states reached by the network when an input pattern is applied (usually, this involves transition through a number of non-steady states). If learning has been perfected by using center-of-class patterns all orthogonal to each other, only the desired attractors will be present [7]; otherwise, spurious attractors will in general appear in the system's behavior.

$$W = \begin{bmatrix} 2 & 2 & 0 & -2 \\ 2 & 2 & 0 & -2 \\ 0 & 0 & 2 & 0 \\ -2 & -2 & 0 & 2 \end{bmatrix} \qquad \text{(a)}$$

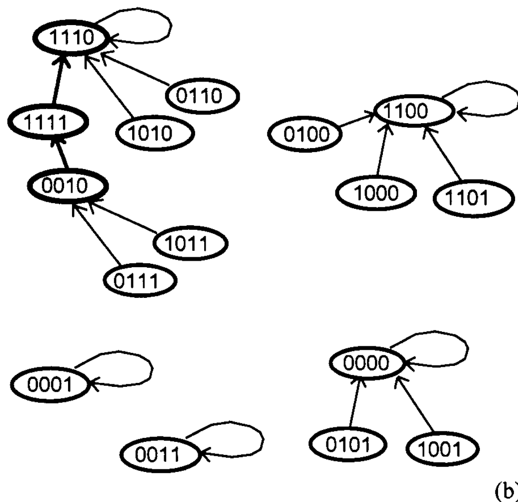


Figure 1 - The state graph for an Hopfield network: the weight matrix of the Hopfield network (a), the state graph (b).

Considering as an example a four-neuron Hopfield network characterized by the weight matrix W in fig.1a, application of the input pattern *0010* leads to creation of the state sequence *0010*, *1111*, *1110* (steady state), that can be represented as an oriented path in a state graph. Correct operation of the network grants that each state will be traversed only once. Thus, the state graph segment obtained is a simple directed path from the initial state s_i to the related attractor s_i^a.

Whenever we apply to the network an input pattern corresponding to any intermediate state s_i^k of a previously identified path, the state sequence created starting from s_i^k coincides with the subsequence s_i^k, s_i^{k+1}, ..., s_i^a of the complete path (this happens, e.g., if the input pattern is *1111*).

If another input pattern I_j is applied leading to the same attractor s_i^a, the directed path associated with such an 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 the steady state *1110*, the first case is achieved starting from the input state *1010*, leading to sequence *1010*, *1110*, while the second one is generated by the input state *1011*, leading to state *0010*.

By applying all input patterns leading to the same steady state s_i^a, an acyclic directed subgraph is built containing all and only the initial states sharing s_i^a as attractor. An example is given by the subgraph where the states converge to the steady state *1110* in figure 1. This subgraph is similar to an n-ary tree; however, the oriented paths are not directed from the root (in our case, the attractor *1110*) to the leaves, but they are in the reverse direction.

Similar subgraphs can be created for each attractor in the neural paradigm. Given the determinism of the parallel recall approach, these subgraphs are disjoint. The complete graph representing the FSM behavior is given by the union of all the above subgraphs; therefore, it is similar to a forest (see fig. 1b).
This basic characteristic does not change even in the presence of spurious attractors. When only expected attractors are present in the neural paradigm, the set of possible states is partitioned into disjoint subsets each associated with an expected 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 an attractor. Each possible state belongs to exactly one subgraph.
When spurious attractors occur in the neural definition, the set of states is again partitioned into disjoint sets: in figure 1, 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. As far as

definition of the FSM is concerned, there is no differences between these two types of subsets.

Note that, as far as testing is concerning, spurious attractors and the related subgraphs will be treated in principle just as expected attractors and the related states subgraphs.

All the about assumed that the whole input space of the Hopfield network was meaningful, i.e., that all 2^n binary configurations over $x_1 ... x_n$ could be forced as initial states. As a consequence the associated FSM contains exactly 2^n different states. If on the contrary the input space is partitioned into a set of allowable patterns X_A and a set of unacceptable patterns X_U, states associated with patterns in X_A may appeared in the state graph in any position (ether as leaves or as nodes internal to a path or as roots) while states associated with patterns in X_U may never appear as leaves or roots. They will ether be internal to a path or not appear at all and the state graph, which may therefore consist of less than 2^n states.

4. Test pattern generation for the FSM model of the Hopfield network

By analyzing the state transition graph of the FSM modeling the Hopfield network, we can derive a strategy for creating the test procedure. Testing of an FSM at behavioral level aims at verifying if all allowed transitions between states (i.e., all transitions defined in the state graph) are correctly performed. If we adopt a single state transition fault model, an error can alter only one transition either in its output and/or in its next-state [4].

Due to the characteristics of the Hopfield networks, the starting state s_i of the transition under test can always be reached and controlled. In fact, if the starting state belongs to X_A, it can be directly set as the initial state of the network at time t_0. Otherwise, a pattern in X_A associated with a state s_k and such that a path from s_k to s_i exists in the state graph must be selected and applied to the network. The path from s_k to s_i constitutes the *set-up sequence* for the transition under test. The application of a set-up sequence allows testing at the same time all traversed transitions, since each transition is directly observable.

After the network has been set in the starting state for the transition under test, testing such a transition means simply performing an iteration of the neural computation according to equation (1) and observing the output result. The transition has been correctly performed if the state reached coincides with the expected one.

The set of input patterns that must be applied to the network, to test all transitions in the state graph, is the *test procedure* of the network at behavioral level. Obviously, it is not unique due to the arbitrary order of examination of the transitions.

To minimize the time required to test a device implementing the Hopfield network, we need to minimize the number of patterns belonging to the test procedure. Each test pattern is composed of a set-up sequence (which can be empty) and the transition under test. A test pattern completely included into another one can be discarded; for example (see figure 1) test pattern *0111,0010*, which tests the transition outgoing from state *0111*, is completely included in the test pattern *0111,0010,1111,1110*, which tests the transition outgoing from state *1111*.

The minimum test set is composed of all the input patterns corresponding to the leaves: in this case the Hopfield network is constrained to reach the steady state and thus to traverse and test all admissible transitions.

In our example, the minimum test procedure is given by the following 11 patterns: *0001*, *0011*, *0100*, *0101*, *0110*, *0111*, *1000*, *1001*, *1010*, *1011*, *1101*.

Generation of the minimum test procedure implies complete exploration of the state graph of the FSM associated with the Hopfield network. An explicit description may require a large amount of memory and a long time to verify if a state has been already examined; these requirements exponentially grow according to the size of the Hopfield network (i.e., with the number of neurons). To minimize the storage and the computation time we suggest to adopt an implicit approach for describing and managing the FSM

Thus, the deterministic approach, previously described, is converted into a random approach that performs a pseudo-exhaustive analysis of the admissible input patterns (states). A possible test pattern is randomly generated and simulated until the steady state is reached. All the traversed transitions are thus tested and set-up sequences are no longer necessary. For example the randomly generated input *0010* (see figure 1) enables testing of all transitions between state *0010* and the steady state *1110*. If the randomly generated input (in this case *0010*) is not a leaf, it will be removed from the test set when one of its predecessors will be randomly generated (in this case *1011* or *0111*). Note that the direct generation of the input configurations corresponding to the leaves is not feasible due to the impossibility of explicitly describing the state transition graph of the FSM.

The use of Binary Decision Diagrams (BDDs) for testing [12, 13] is becoming more and more popular, in particular because BDDs allow to implicitly enumerate

FSM's states [14]. The following description, of the proposed algorithm, relates to the standard operations on Reduced Ordered BDDs (ROBDDs) defined in [10].

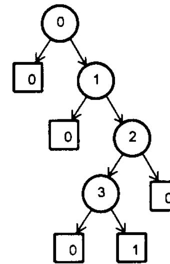
BDDs are binary trees composed of nodes representing *variables*; each node has a pair of outgoing edges representing its possible binary values (*1* or *0*); edges can connect a node to another one or to a leaf. A path starting from the root of the BDDs and ending into a TRUE (1) leaf can be seen as an *algebraic product*; all the variables, traversed by the path, assume the value of the associated incoming edges. For example, figure 2 (a) shows the BDD for the product *0010*; that can be derived looking at the unique path starting from the root (node 0) and ending into the TRUE leaf (with the convention of left edge equal to 1 and right edge to 0). A set of binary configurations may be represented by a single BDD built by *or-ing* the BDDs of the single configurations.

Let us define **S** as the BDD representing all possible states (i.e., 2^n if all input patterns are allowable). At the beginning of the algorithm, **S** is composed of a single node (the *TRUE* node) that represents the inclusion in **S** of all possible input configurations. Such a circumstance dramatically highlights the efficiency of BDDs in relation to explicit techniques for storing binary information: only one node represent 2^n objects. If only a subset of the possible 2^n configurations is allowed, the **S** set includes only the admissible inputs. Let us define **E** as the BDD representing the set of *already explored states*: at the first step of the algorithm **E** is the *FALSE* node since no node has been explored yet. Note that **E** is the complement of **S**: definition of two BDD's is useful to clearly describe the algorithm, but only one must be used in the actual implementation of the algorithm.

The proposed approach uses the activation function f to analyze the network evolution until the steady state is reached. At each clock tick, the network changes state or remains in the final steady state: our exploration algorithm generates a BDD **C** for the current state by using *n-1* times the *apply-and* operation on BDD's, as defined in [10] Such an operation performs the algebraic *and* operation between two BDDs. One represents the current product and the other the current variable. The complexity of the *apply* operation is quadratic with the dimension of the two involved BDD's, but in this case one of the two BDDs has size equal to *1* (the variable) and the number of nodes of the other one is smaller than *n*. Thus, the final complexity for the construction of **C** is $n(n+1)/2$.

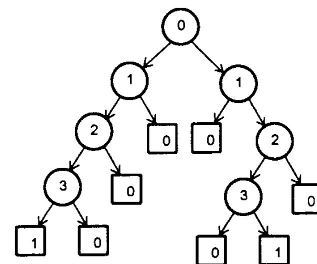
Whenever a new state is reached, its BDD **C** is compared with the set **E** of already explored states: if **C** is included in **E**, the exploration terminates and a new input pattern is selected; otherwise, the current state **C** is added to **E** by using the *apply-or* operation (see [10]) and a new current state is computed via the activation function f.

patterns: 0010



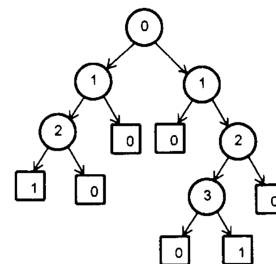
(a)

patterns: 0010
1111



(b)

patterns: 0010
1111
1110



(c)

Figure 2 - The BDD's trees for compact pattern generation: the tree after the first pattern (a), the second one (b), and the third one (c).

Verifying that **C** is included in **E** is performed by the *intersect* operation (see [10]). If the intersect operation is successful, and the current state is also a test pattern, such a pattern can be removed from the test set since it means the inclusion of the sequence, starting from **C**, into the sequence starting from the current input. Complexity of the *intersect* operation is linear with the number of nodes of the smaller BDD involved (in this case **C**). Since the number of nodes in **C** is always equal to n, the above verification is performed in a very efficient way.
Generation of a new input pattern can be easily performed by traversing, in a random way, **S** from the root to a leaf. Also complexity of this operation is proportional to n (length of every path in **S** is at most equal to n).

The use of BDDs allows to generate the test patterns by analyzing the set of states reached by each input pattern.

5. Concluding remarks

The approach here presented allows to design a test procedure for a Hopfield network independently of final implementation. Previous research has proved the functional approach chosen to be effective in the case of FSMs; thus, given the common logical fault model, the same rate of fault coverage can be deduced for digital implementations of Hopfield networks. Simulations will be carried out to check whether the same extends to fully analog implementations.

All results presented here relate to the parallel-recall approach; at present, research is under way to determine an efficient extension to the case of sequential recall.

6. References

[1] C.H. Chen, C. Wu, D.G. Saab, "BETA: behavioral testability analysis", *Proc. ICCAD*, Santa Clara, 1991

[2] M. Bombana, G. Buonanno, P. Cavallaro, D. Sciuto, G. Zara, "A multi--level testability assistant for VLSI design", *Proc. EuroDAC 92*, Hambourg, Germany, 1992

[3] V. Piuri, M. Sami, D. Sciuto, R. Stefanelli, "A behavioral approach to testability of neural networks", *Proc. IJCNN '92*, Baltimore, 1992

[4] K.T. Cheng, J.Y. Jou, "Functional Test Generation for Finite State Machines", *Proc. IEEE ITC*, 1990

[5] I. Pomeranz, S.M. Reddy, "On Achieving a Complete Fault Coverage for Sequential Machines Using the Transition Fault Model", *Proc. IEEE DAC*, 1991

[6] G. Buonanno, F. Fummi, D. Sciuto, "Functional Fault Model and Gate Level Coverage for Sequential Architectures", *Proc. IEEE ICCD*, 1993

[7] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Co., 1991

[8] *Proc. WCNN93,* Portland, 1993

[9] K.T. Cheng, J.Y. Jou, "A Single State-Transition Fault Model for Sequential Machines", *Proc. IEEE ICCAD*, 1990

[10] K.S. Brace, R.L. Rundell, R.E. Bryant, "Efficient Implementation of a BDD Package", *Proc. 27th ACM/IEEE DAC*, 1990

[11] M. Abramovici, M.A. Breuer, A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990

[12] S.B. Akers, "Binary Decision Diagram", *IEEE Transactions on Computers*, June 1978

[13] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, August 1986

[14] H.J. Touati, H. Savoj, B. Lin, R. Brayton, A. Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines using BDD's", *Proc. IEEE ICCAD*, 1988