

A Partitioning Approach to Improve Reconfigurable Neuron-inspired Online BIST

Ali Shahabi, S. Behdad Hosseini, Hassan Sohofi and Zainalabedin Navabi
Electrical and Computer Engineering, School of Engineering Colleges, Campus 2
University of Tehran, 1450 North Amirabad, 14395-515 Tehran, Iran
{shahabi, behdadh, h_sohofi}@cad.ece.ut.ac.ir, navabi@ece.wpi.edu

Abstract— Two of the most challenging issues in online testing are deriving a general tester scheme for various circuits and reducing the area overhead. This paper presents a novel reconfigurable online tester using artificial neural networks to test combinational hardware. Our proposed BIST architecture has the capability of testing a number of arbitrary sub-modules of a big design simultaneously by time-multiplexing between them. Output partitioning method is proposed as a powerful technique to reduce neural network training time and the tester area overhead. Our experimental results show that after proper partitioning the average area overhead is reduced by 16% in data-path and 33% in memory area. Also average fault detection latency has been improved by 14%.

Keywords- *On-line Testing, Self-Checking, Digital Logic Modeling, Digital Neural Networks, Partitioning.*

I. INTRODUCTION

Great innovations in semiconductor technology and industry have made digital system testing more and more challenging [1]. Tendency to System On Chip (SoC) realizations and thus growth in transistor density in order to meet current intricate and high performance system requirements has increased the likelihood of existence of permanent and transient faults during working period of a chip [2]. Furthermore, complex circuits are getting harder to test using external test equipments because of limited available number of IO ports. All of the above motivations justify Self-Test as one of the approaches to tackle the testing issues of digital systems. In this technique, called Built-In Self Test (BIST), a dedicated hardware is added to system and some provisions have to be considered in the original design to check the validity of the system during its life time.

In Online BIST, the correctness of Circuit Under Test (CUT) is checked without interrupting its normal operation by constantly monitoring the input/output. Exertion of online test techniques is inevitable in fault sensitive applications [3] where other DFT techniques, such as Offline BIST are not feasible due to the impossibility of temporary suspension of CUT.

Different approaches for online testing have been proposed. The most challenging problem is the imposed area overhead of the tester. Generality of the method or ability to be applied to a wide range of circuits is another important issue in this domain. To the best of our knowledge, efficient on-line test techniques (in terms of imposed area overhead) have been proposed and used for specific applications and limited circuit types. In the next section we will review some of those methods which are related and can be compared to our work.

In this paper, we improve a neuron-inspired reconfigurable online BIST previously presented in [4]. In that approach, a digital circuit defined by its truth table or logic equations, is completely modeled by an artificial neural network (simply Neural Network or NN). In other words, a NN is being trained to produce exactly the same outputs of the CUT for all inputs.

The NN-based architecture is capable of being reconfigured to model and test different combinational designs (*e.g.*, parts of a complex circuit) on the fly. A new configuration is obtained by adding data needed to represent a new CUT into our architecture which are acquired from the corresponding trained NN.

We have incorporated a variety of techniques to find the most compact NN with the ability of exactly modeling the behavior of a given CUT. Our solution especially effective for big circuits is partitioning their output signals into a number of disjoint sets and then finding a suitable NN for each partition. Our results show that if proper partitions are exploited, the overall areas of these NNs are less than one big NN when implemented using our reconfigurable architecture. Overall training time is also reduced when partitioning technique is utilized. In some cases, the latency to fault detection is improved too. In each partition, a number of promising NNs capable of modeling the hardware are obtained by excessive training and simulation sessions, and the most area-efficient NN will be selected.

The rest of this paper is organized as follows. Previously published methods for online testing and their characteristic as well as utilization of NNs in digital testing domain are briefly reviewed in Section II. In this section we also compare our approach to previous solutions when applicable. Section III presents the basics of NNs and several important aspects on their hardware implementation. We also show how a NN can be used to model a digital hardware. Our proposed approach to obtain and set up an optimized NN-based online BIST for a given set of circuits is thoroughly explained in Section IV. Section V shows the experimental results and the effect of different parameters on area and performance of the tester. Finally, Section VI concludes the paper and summarizes our proposed method.

II. RELATED WORKS

The increasing appearance of soft errors usually caused by ionizing radiation in the state-of-the-art chips put another big challenge in the area of hardware testing. In this section, the existing literature on online testing to detect soft errors will be reviewed and compared to our method. Then previous usages of NNs in digital test domain will be presented.

In [5] the authors first emphasized the significant difference between online memory testing [6] and online logic testing [7]. A variety of techniques using redundancy (*e.g.*, duplication, coding, online BIST) were surveyed and a new online error detection approach in logic circuit using logic implication was introduced. Despite the wide usage of logic implications in some application such as logic synthesis optimization, extraction of these implications for large circuit might be complex and time consuming. In our method the behavior of a CUT is used to make an exact model while in [5] they benefit from the internal structure of the CUT.

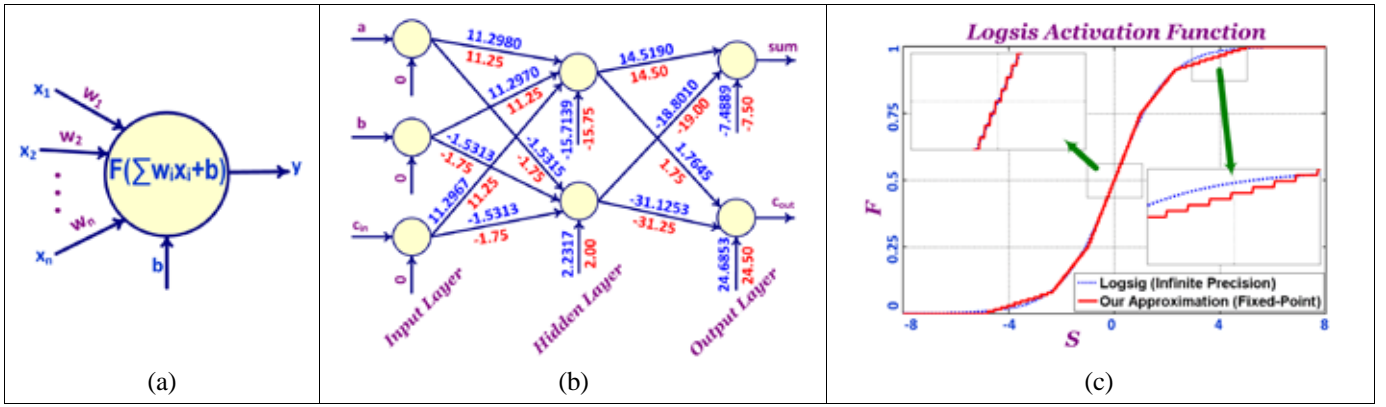


Figure 1. (a) A typical neuron (b) A Feed-forward neural network to model a full-adder (c) Logsig Activation Function (AF)

The authors of [8] have used Viterbi coding to design Finite State Machines (FSMs) to facilitate online error detection. Although, the area overhead of this technique is relatively less than the previously published self-checking FSM structures, it requires re-synthesis of the original circuit and also its applicability is confined to a limited range of digital hardware designs. Yang et al. in [9] have proposed an enhanced logic BIST architecture for online testing. Since for periodic and online testing, importance of stuck-at-open defects that arise due to the wear and tear of the circuit becomes exalted, the authors paid attention to this category of faults. They proposed a new set of tests to detect this kind of faults in addition to a suitable BIST architecture. Considering just open faults, apparently, reduces the generality of this approach. In [10] a concurrent BIST based on the deterministic test set has been proposed to detect permanent faults and prevent their accumulation. Although, their method adjusts a deterministic test set by reducing the number of specified bits to diminish the hardware, the imposed area overhead cannot be neglected and in some cases it exceeds 100%.

Time-Multiplexed Online Checking (TMOS) is introduced in [11] as an effective way to reduce area overhead by dynamically reprogramming a dedicated region of a FPGA for testing another part of the CUT. However, a time lag in producing tester data is inevitable which results in latency to the fault detection process. Our idea to minimize area overhead by increasing delay in the tester is similar to their method.

Artificial neural networks, because of their capability to solve NP-Complete problems, also establish interesting and useful approaches in digital system test domain. In [12], a feed-forward NN trained by resilient error back-propagation has been presented to detect defects in analog integrated circuits. Chakradhar et al. presented a test generation algorithm for single and multiple faults in a specific class of combinational circuits, called (k, k)-circuits, using a bidirectional NN model of the original circuit [13]. In [14] a new automatic test pattern generation technique, in which stuck at open faults are considered, has been presented by means of NN models. Ortega et al. in [15] has shown a Generalized Hopfield Neural Network (GHN) can be used to design the checking circuitry of a concurrent testable circuit. Finding a solution for the checking circuit of the system, that could be capable of detecting as many error as possible with minimum hardware complexity, is done by mapping the problem into a NN.

III. ARTIFICIAL NEURAL NETWORKS

This section contains a concise introduction to the artificial neural networks, or simply NNs. First we briefly explain their fundamentals and the concept of training. Then we show how a

NN can be used to model a digital design. Finally NN implementation issues in hardware will be discussed.

A. Preliminaries

Neural networks are non-Von Neumann computational paradigms capable of solving complex non-linear problems. NNs have been successfully used in optimization, control and recognition applications [16]. They are composed of simple and similar elements called neurons. Each neuron (see Fig. 1.a) accepts a number of inputs and performs a function (usually some form of sigmoid) on summation of all inputs and a bias value. A directed weighted connection, called a link, might exist between two neurons. Each link value (produced from the source neuron) is multiplied by its weight prior to enter as an input to the destination neuron.

A group of neurons is called a layer where there is no link among them. Feed-forward network is one of the most common topologies of NNs (Fig. 1.b) which consists of one input layer, usually one hidden layer and an output layer. The number of neurons of input and output layer is equal to the input and output signals, respectively. The computational capacity of feed-forward neural networks is dependent on the number of hidden layer neurons. The real power of NNs comes from two features: inherent parallelism and the ability to learn.

Training in NNs is the process of adjusting weights in order to make outputs of the network for a set of inputs closer to their target output *i.e.*, expected correct output. An epoch is defined as the exhibition of the whole training data once to the network. Usually, many epochs are applied in learning procedure in order to get correct output from the network and converge the network coefficients (*i.e.*, weights and biases) further to their optimum points. Several learning algorithms have been proposed for different application types and network topologies. Back-propagation [16] is a popular learning algorithm in feed-forward networks. It first computes the errors of outputs to target data for a given input. Then this error value is propagated backward from outputs to input neurons considering relative contribution of each neuron to the final error. The calculated error for each weight is then used to tune the weight (by using a learning rate constant).

B. Digital Hardware Modeling via a Neural Network

This section describes how a digital hardware can be modeled by means of a neural network. The first step is to establish a relation between inputs/outputs of the hardware and NN. Although our modeling approach is similar to those presented in [14] and [15], our mechanism to obtain an optimal networks differs. In [17], the authors have proved that for any arbitrary combinational digital circuit, a neural network exists that is capable of modeling the circuit.

Our NNs are Feed-forward networks with one input, one hidden and one output layer. Assume a given circuit has n bits of input and m bits of output. Our NN will have n input lines, each of which connected to one neuron forming the input layer. Because inputs of NNs are supposed to be real values, each input is padded with zeros in fractional part to form a fixed-point representation of '0' or '1'. Minimum number of required neurons in hidden layer is obtained by an algorithm explained in the next section. Output layer will contain m neurons, and their outputs should be rounded to obtain one output bit. Output is considered as '0' if it is below or equal to 0.5, and is regarded as '1' when it is greater than 0.5. All internal values as well as weights and biases are real numbers with fixed-point representation. To clarify the computations, imagine a full adder as an example. After applying our training algorithm, it is determined that minimum number of required hidden neurons is 2. Acquired NN along with weights and biases is depicted in Fig 1.b (precise and fixed-point values are shown in blue and red, respectively).

C. Implementation of a Neural Network in Hardware

Since the main problem of a hardware-implemented NN is its area and because the functionality of all neurons is alike, just one neuron is sufficient to realize the whole network. Also in this approach, we are able to implement all NNs (having different input, hidden, output neurons and weights) by the same neuron. The major drawback of this approach is the delay between applying the inputs and getting the outputs. It will result in missing some CUT inputs/outputs. Nevertheless, as stated before, online testing is applied to circuits during their working hours, thus, as will be shown later, it is probable that missed faults be caught later.

From hardware point of view, a neuron consists of three components: multiplier, accumulative adder and a non-linear activation function (AF). In addition to those, a controller is needed as well as a ROM to store weights and biases and a RAM to keep intermediate values. The imposed area can be further reduced by using a sequential multiplier. The AF used in our NN and in trainings, is Logsig which is shown in Fig. 1.c (infinite precision). Among its implementations, Piecewise Linear Approximation of a Nonlinear Function (PLAN) [18] is chosen because of its acceptable error factors and its efficient hardware implementation. Fig. 1.c also depicts the outputs of our PLAN implementation with seven fractional bits.

To further reduce the area overhead of the tester, fixed-point representation is used in NN [19] to hold weights, biases and all intermediate data. The main issue is how to decide bit-width for data and arithmetic units (See Section IV.B.2)).

IV. PROPOSED ONLINE BIST METHODOLOGY

This section is dedicated to explain our methodology for online testing. First of all, the architecture of the proposed online BIST hardware is illustrated. Then a general flow for extraction of necessary data to feed our hardware is described and investigated thoroughly.

A. Reconfigurable Hardware Architecture

In this section, the architecture of our BIST is presented. Its primary purpose is to calculate the outputs of any desirable NN described in its ROM. The outputs of a NN are used to verify correctness of corresponding CUT afterwards. The data-path of the implemented online BIST is shown in Fig. 2. The hardware of our BIST architecture contains a small RAM, a ROM, one saturate adder, one sequential multiplier, one Logsig AF, and some logic for holding IO and validity checking of the output.

All values through the data-path are in fixed-point representation.

When an input value is selected for testing, it is loaded into a circular shift register. For neurons on the hidden layer, each bit of the input value in this register should be shifted out and multiplied by the corresponding weight value located in the ROM rows. These intermediate results will be stored in RAM. For neurons on the output layer, the results of the hidden layer from RAM are multiplied by the corresponding weights from the ROM. The multiplier is sequential and activates a ready signal when the result is ready. After the truncation of the multiplication result, this value is accumulated into a register using a saturation adder. When all input values according to the structure of a neuron are multiplied and accumulated in the register, the result must be confined by passing through the AF module. If the neuron is related to the hidden layer, the output of the activation function should be written to the RAM rows for later use. If the neuron is an output layer's neuron, the two most significant bits of the activation function's output (according to the fixed-point representation) should be checked to see if the output value is zero or one. Obtained bit value is then compared to the related output bit value of the CUT which is shifted out from a shift register at the output. If both bits are the same, the output of the circuit is fine; otherwise an error signal is activated to indicate that the CUT is faulty.

Controller is responsible to issue appropriate control signals for proper order of computations. For making the controller as simple as possible, we have arranged biases and weights in a special sequential manner and also adjusted order of neuron computations to avoid random access and address calculations to ROM and RAM. Data required for calculating the output of the first hidden neuron is first stored in ROM, followed by data of the second hidden neuron and so on. After hidden neurons, coefficients of output neurons are kept in a same manner. In this case, if we calculate output one by one and in order, the RAM is also accessed sequentially.

In order to make the same hardware usable for online

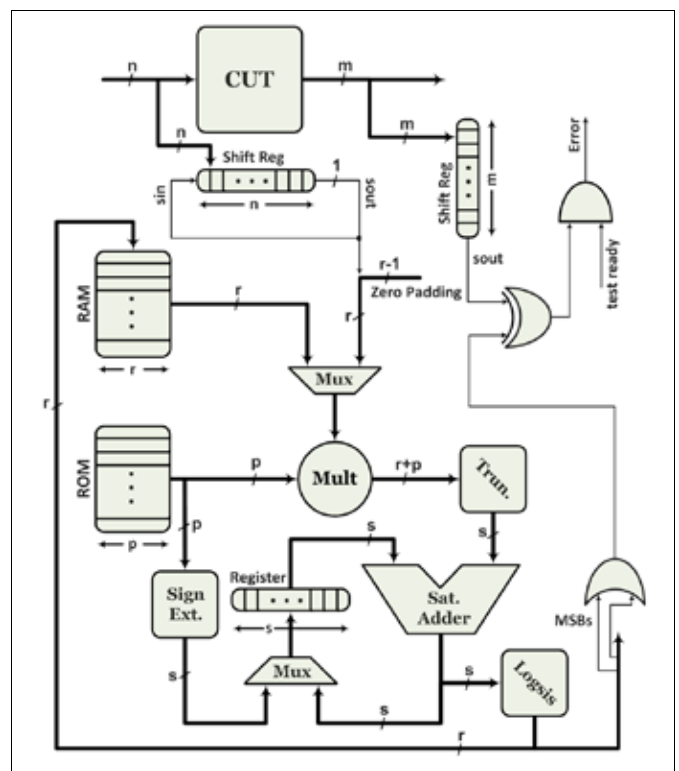


Figure 2. Data-path of the proposed BIST architecture

testing of different CUTs or partitions (see Section B) of a CUT, the required bit-width for different parts of the data-path is re-calculated by maximizing corresponding bits in all CUTs and partitions. The final NN is also made reconfigurable by storing weights and biases of all previously obtained bit-true NN models in a single ROM. A pointer to the start location of coefficients of the current CUT/partition, in addition to the number of inputs, hidden, and outputs neurons are also stored in a separate controller ROM. The reconfigurable hardware needs a final step to switch into another configuration, on the fly, by appropriate multiplexing of CUTs inputs/outputs. We implemented the policy of regularly switching (and testing) between CUTs. Other policies can be practiced easily by adjusting a constant value stored in controller which determines how many inputs/outputs should be tested for a CUT before switching to the next one.

B. Generation Flow of Optimim Neural Networks

Our approach to acquire the best NN for a given combinational CUT in addition to the corresponding BIST parameters is shown in Alg. 1. Description of the CUT is fed to this algorithm using two text-files that show the expected outputs of the circuit for all possible inputs. `FINDBESTNETWORK` uses five major steps to produce required data: 1) Partitioning outputs of the CUT, 2) Finding minimum hidden neurons for each partition, 3) Gathering some different valid networks, 4) Running a bit-true simulation for each network to calculate required bit-widths, and 5) Choosing the best possible network among the candidates for each partition. In the following sub-sections the whole procedure will be described in detail.

1) Training a Neural Network

The first and the most time consuming step in modeling the behavior of a circuit (or partition) using NNs is the training stage. As stated before, input and expected (target) data is generated from the truth-table or logic equation of the CUT. Finding a minimum number of hidden neurons to minimize the trained network and subsequently the area overhead is a major difficulty in our training flow. Our experiences show that the more outputs of the circuit, the more time must be spent to find the number of minimum neurons.

By partitioning the expected data, training time and area overhead will be reduced simultaneously. The expected data is a two dimensional matrix where its rows show the circuit outputs for an input (corresponding row in input training data). Each partition will contain some of the bits of the output, *i.e.* some columns of that matrix. To obtain the best partitions, we first sort the columns using `SORTHAMMINGDIST` in Hamming distance order (*i.e.*, summation of bitwise difference of corresponding values). By doing so, each partition will contain similar outputs that facilitates network learning by focusing just on differences. Using this matrix and proper number of parts, `PARTITIONOUTPUTS` provide best possible partitions from the output data. Now, we can train a separate network for each partition and find the number of minimum neurons by which the trained NN can emulate a part of original circuit, exactly.

To find minimum number of required hidden neurons (`FINDMINIMUMNEURONCOUNT`) for each partition, we start with two hidden neurons and go up by powers of two. During the learning procedure, the network is constantly tested by applying input data to see whether it is trained enough to exactly model the given circuit or not. Exact modeling means that for all inputs, the network output (after rounding, see Section III.B) have to be exactly the same as the expected value. If the NN is not trained after a parametric maximum

```

FINDBESTNETWORK(in_file, exp_file, parts)
begin
  [train_in train_exp] = READTRAINDATA(in_file, exp_file);
  sorted_exp = SORTHAMMINGDIST(train_exp);
  train_exp_set = PARTITIONOUTPUTS(sorted_exp, parts);
  for each partition p in 1 to parts do
    part_train_exp = train_exp_set(p);
    n = FINDMINIMUMNEURONCOUNT(train_in, part_train_exp);
    network_set = GATHERDIFFERENTNETS(n);
    for each valid network net from network_set do
      width = ROUGHESTIMATIONFORNETWORKWIDTH(net);
      while ( ISVALID(net) )
        TUNENETWORKWIDTH(net, width);
      end while;
      cost = CALCULATECOST(net);
      if ( ISMINIMUM(cost) ) then
        KEEPNETWORK(net);
      end if;
    end for;
  end for;
end;

```

Algorithm 1. Proposed algorithm to obtain optimum neural network

number of epochs, we assume that the number of hidden neurons is not enough. After finding an upper number of hidden neurons, it is decreased gradually until the minimum number of required neurons is obtained.

Since initial weights in the generation phase of NNs are chosen randomly, it is most probable that two equal NNs in terms of hidden neuron numbers behave completely different in training procedure. Using previously acquired minimum number of hidden neurons, `GATHERDIFFERENTNETS` collects different trained networks. Obviously these networks have different weights and hence impose different area overhead in our hardware model.

2) Bit-True Simulation

A Precise NN is a network without any constraints on the precision of internal data. Clearly because of un-acceptable area overhead, it is infeasible to realize it in hardware. Thus we have to reduce the accuracy of the internal computations, as far as the final outputs (after rounding) remain intact. This hardware is called the bit-true model of the original trained NN. In order to find optimum bit-widths, we replace all of the arithmetic computations in the NN with our hardware implemented bit-true models. Since the developed hardware components in HDLs are unconstrained, *i.e.*, they can be instantiated with any bit-widths, all of their corresponding software functions are parameterized as well. All arithmetic hardware units, *e.g.*, saturate adder, multiplier, and AF-PLAN have been verified for different bit-widths.

The bit-true simulation is conducted to extract the necessary bits to represent weights, biases and intermediate results. First we use a rough estimation to extract the necessary bits for integer and fractional parts. Maximum integer bits for weights and biases can be calculated based on their highest and lowest values. Maximum fractional bits are acquired based on the maximum precision provided by AF-PLAN which is seven in our case. Then we try to decrease integer and fractional bits until the output of the simulation goes wrong. This point denotes the actual minimum bits required to correctly model the given CUT (`TUNENETWORKWIDTH`). For example, consider the full-adder again (See Section III.B). Figure 1.b also shows the value of weights and biases (by red color) with respect to our bit-true model using five integer bits (excluding sign bit) and two fractional bits. For each network gathered

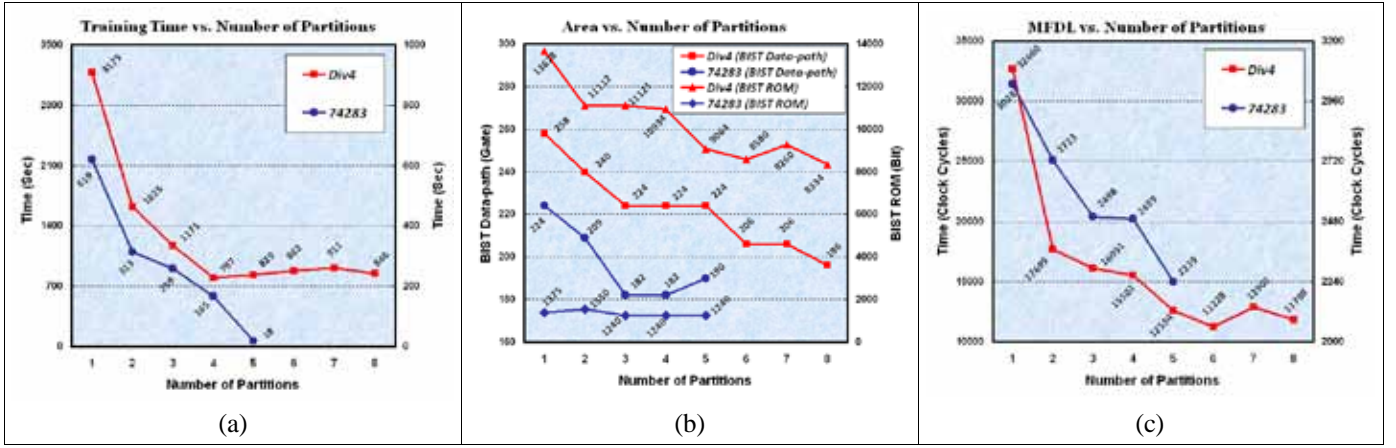


Figure 3. Effect of partitioning on (a) Training time (b) BIST Area overhead (Data-path and ROM) (c) Mean fault detection latency

from the previous step, the required bit-width is extracted as described before. Among those, the NN having minimum bit-width (CALCULATECOST) is selected for each partition or CUT.

V. EXPERIMENTAL RESULTS

To assess the proposed technique, we have considered a variety of combinational and full-scanned sequential circuits. For each circuit an online BIST architecture based on presented approach is generated. Also different partition numbers have been applied for each circuit. Finally, all of them will be combined and tested in a separate reconfigurable NN.

Mean Fault Detection Latency (MFDL) is used as a factor of test quality. It defines the mean time between appearance of a fault and its detection by online tester. For a given CUT, all of the stuck-at faults have been injected one by one and the time (in clock cycles) between its injection and detection is measured. MFDL is the mean of above calculated times when all of the faults are injected many times in a random manner. Input patterns are also generated randomly. Note that faults which are impossible to detect were excluded from the fault lists. Because no test pattern exists to detect this category of faults, it is impractical for an online BIST to recognize them without access to the internal parts of the CUT, for example by inserting some controllability and accessibility points.

We first investigate the effect of partitioning on two typical circuits. Fig. 3.a shows the training time as a function of partition numbers. Training time is the total time needed to partition the outputs and find the minimum number of hidden neurons capable of modeling the partition. Due to the fact that addition of one neuron to a NN contributes exponentially to the training time (because of exponential growth in weights) it is expected that training a number of small NNs will be faster than a big NN as depicted in the figure. In Fig. 3.b the BIST data-path area besides the required ROM for different number of partitions of the former circuits are shown. Since the RAM holds temporary data and it does not greatly vary between different partitions (see TABLE I), we excluded them from the figure. As shown in the figure, the area of data-path and ROM are almost reduced by increasing the number of partitions until the addition of more partitions imposes extra overhead fading the effect of small NNs. Fig. 3.c denotes the effect of partitions on the MFDL of the circuits. Since the total size of small NNs are usually less than one big NN, the expected data is generated more swiftly and the time to detect a fault using the same input pattern will be reduced as well. For CUTs in TABLE I, the average area overhead of the BIST was reduced by 16% in data-path and 33% in memory bits when the best partitions were utilized. Average MFDL was also decreased by 14%.

The circuits chosen to be used as benchmarks and the extracted results are reported in TABLE I. Each row shows the obtained data for each benchmark. The first group of columns denotes each benchmark specification (*i.e.*, name, short description, the number of input and output ports). Then the benchmark is partitioned into a number of parts. The next three columns show the number of partitions, required hidden neurons and training time, respectively. Required bits for the different parts of our architecture with respect to Fig. 2 are given in the third group of columns in TABLE I. We use the notation of $sx.y$ to show signed fixed-point format that uses x bits for the integer part and y bits for the fractional part. Likewise $ux.y$ is used to denote an unsigned fixed-point format. The required bit-widths for RAM, ROM and internal bus are reported in columns 7 to 9 of the table, respectively.

Finally, the last group of columns provides various measurements to facilitate evaluation of the partitioning approach. Each benchmark with the extracted BIST parameters was modeled using 'VHDL' or 'Verilog' hardware description languages and then, synthesized on an ASIC library. The imposed area overhead of the BIST architecture is reported in terms of its data-path gates and needed memory bits. Since the implementations of memories are completely technology and synthesis dependent we reported their area in bits, to be fair. For instance, RAM can be located in dedicated FPGA RAM area or realized as dynamic RAMs. ROM can be a look-up table or minimized when regarded as Boolean functions.

The last two rows, RNN (Reconfigurable NN), show the characteristics of the BIST that tests all of the foregoing circuits. First RNN uses the NN of one-partitioned CUTs. The second one utilizes the best partition configurations of benchmarks. Using the data of TABLE I we conclude that in spite of the unacceptable area overhead of our BIST for small CUTs, the overall overhead is reasonable when all of them are combined or for big circuits. We synthesized the CUTs and the total area (315 in gates) is smaller than the area of RNNs. It can be explained by the fact that the area of our reconfigurable BIST remains almost intact regardless of the CUTs.

VI. CONCLUSION

Artificial neural networks can be utilized to model any arbitrary combinational hardware defined by truth-tables or logic equations. In this paper, we presented an online reconfigurable BIST architecture by using neural networks concept. Our proposed architecture permits switching to other configurations to model and test different CUTs on the fly. Similarity among computations in NNs is utilized to reduce imposed area overhead in expense of allowing more delay to

TABLE I. EXPERIMENTAL RESULTS

Benchmark Circuits			Neural Network Details			Data-Path Bit-Widths			Area Overhead			MFDL (Clock Cycles)
Name	Description	# of In/Out	# of partitions	# of Hidden Neurons (for each partition)	Training Time (sec)	RAM (r)	ROM (p)	Internal (s)	BIST			
									Data- path (Gates)	Memory(bits)		
RAM	ROM											
c17	ISCAS Benchmark	5/2	1	3	61	u1.2	s5.0	s5.0	132	9	156	264
			2	4 (2,2)	2	u1.1	s5.0	s5.0	123	4	204	267
7485	4-bit Comprator	11/3	1	3	138	u1.4	s6.3	s6.3	210	15	525	1014
			2	4 (2,2)	2	u1.3	s6.2	s6.2	190	4	520	996
74182	4-bit Lookahead Carry Generator	9/5	1	5	232	u1.2	s6.3	s6.3	191	15	800	1691
			2	5 (3,2)	64	u1.2	s5.2	s5.2	172	9	640	996
			5	10 (2,2,2,2,2)	1	u1.1	s4.0	s4.0	134	4	775	1231
74283	4-bit Binary Full Adder	9/5	1	8	619	u1.5	s6.4	s6.4	224	48	1375	3028
			3	10 (4,4,2)	259	u1.3	s5.2	s5.2	182	16	1240	2498
			5	10 (2,2,2,2,2)	18	u1.4	s5.1	s5.1	190	10	1240	2239
s27	ISCAS Benchmark (Full Scan)	7/4	1	3	71	u1.2	s5.1	s5.0	154	9	280	510
			2	5 (3,2)	36	u1.2	s5.1	s5.0	143	4	384	565
div4	4-bit Divider	8/8	1	57	3175	u1.7	s7.7	s7.7	258	342	13678	32660
			2	54 (6,48)	1625	u1.6	s7.4	s7.4	240	288	11112	17699
			4	58 (2,6,25,25)	797	u1.4	s6.4	s6.4	224	125	10934	15503
			6	50 (2,6,13,11,11,7)	882	u1.3	s6.3	s6.3	206	52	8580	11228
			8	54 (2,2,3,14,10,5,11,7)	846	u1.3	s6.2	s6.2	196	56	8334	11798
RNN	BIST of above circuits	11/8	6	79 (First row of CUTs)	N/A	u1.7	s7.7	s7.7	258	342	16814	46209
			22	87 (Best of CUTs)	N/A	u1.3	s6.2	s6.2	196	56	11457	31693

produce expected output. We also presented a partitioning technique based on Hamming distance on the circuit outputs to further reduce the area overhead of the BIST architecture. This technique also decreases the training time of the corresponding NN and the latency of fault detection. Our methodology to obtain the optimum NN among all of the candidates that can model the given hardware was also thoroughly explained. To determine the imposed overhead of each candidate, we conducted a bit-true simulation and synthesized obtained BIST architectures. The proposed technique does not imply re-synthesis for the previously designed circuits and also can test any arbitrary combinational logic by storing necessary data in the BIST ROM. Several measures to assess the proposed method were introduced and reported for some combinational circuits. We showed that proper partitioning improved the average area overhead and fault detection latency of our online BIST by 16% and 14%, respectively.

REFERENCES

- [1] Semiconductor Industry Association, International Technology Roadmap for Semiconductors, Edition 2008, available: <http://www.itrs.net/reports.html>.
- [2] Y. Zorian, S. Dey, and M. J. Rodgers, "Test of Future System-on-Chips," in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD'00)*, Nov. 2000, pp. 392-398.
- [3] A. Steininger, and C. Scherrer, "On the Necessity of On-line-BIST in Safety-Critical Applications – A Case-Study," in *Proc. 29th Annual Int. Symp. on Fault-Tolerant Computing (FTCS '99)*, 1999, pp. 208-215.
- [4] S. B. Hosseini, A. Shahabi, H. Sohofi and Z. Navabi, "A Reconfigurable Online BIST for Combinational Hardware Using Digital Neural Networks," Accepted to be published in *Proc. 15th IEEE European Test Symp. (ETS'2010)*, May 2010.
- [5] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar, "Using implications for online error detection," in *Proc. International Test Conference (ITC'08)*, 2008, page 24.2.
- [6] M. Choi, N. Park, F. Lombardi, Y.B. Kim, and V. Piuri, "Balanced redundancy utilization in embedded memory cores for dependable systems," in *Proc. 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'02)*, 2002, pp. 419-427.
- [7] P. Drineas and Y. Makris, "Concurrent fault detection in random combinational logic," in *Proc. 4th International Symposium on Quality Electronic Design (ISQED'03)*, 2003, pp. 425-430.
- [8] M. Li, S. Xu, E. Xia, and F. Wan, "Design of FSM with Concurrent Error Detection Based on Viterbi Decoding," in *Proc. 17th Asian Test Symposium (ATS'08)*, 2008, pp. 383-388.
- [9] F. Yang et al., "An Enhanced Logic BIST Architecture for Online Testing," in *Proc. 14th On-Line Testing Symposium (IOLTS'8)*, Jul. 2008, pp. 10-15.
- [10] M. A. Kochte et al., "Concurrent Self-Test with Partially Specified Patterns for Low Test Latency and Overhead," in *Proc. 14th IEEE European Test Symp. (ETS'2009)*, May 2009, pp. 33-58.
- [11] M. Gao, H. Chang, P. Lisherness and K. Cheng, "Time-Multiplexed Online Checking: A Feasibility Study", in *Proc. Asian Test Symposium (ATS'08)*, 2008, pp. 371-376.
- [12] V. Stopjakova, P. Malosek, D. Micusik, M. Matej, and M. Margala, "Classification of Defective Analog Integrated Circuits Using Artificial Neural Networks," *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 20, no. 1, pp. 25-37, 2004.
- [13] S. T. Chakradhar, V. D. Agrawal, and M. L. Bushnell, "Energy Minimization and Design for Testability," *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 5, no. 1, pp. 57-66, 1994.
- [14] Z. Zhang, R. D. McLeod, and W. Pedrycz, "A Neural Network Algorithm for Testing Stuck-open Faults in CMOS Combinational Circuits," *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 4, no. 3, pp. 225-235, 1993.
- [15] J. Ortega, A. Prieto, A. Lloris, and F.J. Pelayo, "Generalized Hopfield Neural Network for Concurrent Testing," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 898-912, 1993.
- [16] M. Ananda Ro and J. Srinivas, *Neural Networks Algorithms and Applications*, Alpha Science International Ltd., Part III, pp.157, 2003.
- [17] P. Zhongliang, C. Ling, L. Shouqiang, and Z. Guangzhao, "Neural Network Approach for Multiple Fault Test of Digital Circuit", in *Proc. 6th Int. Conf. on Intelligent Systems Design and Applications (ISDA '06)*, 2006, pp. 24-29.
- [18] H. Amin et al., "Piecewise Linear Approximation Applied to Nonlinear Function of a Neural Network," *IEE Proc. Circuits, Devices and Systems*, vol. 144, no. 6, pp. 313-317, Dec. 1997.
- [19] M. Moussa, S. Areibi, and K. Nichols, "On the Arithmetic Precision for Implementing Back-Propagation Networks on FPGA: A Case Study", Book Chapter in *FPGA Implementations of Neural Networks*, Springer, The Netherlands, pp. 37-61, 2006.