Predicting Signal Probabilities Using Neural Networks to Improve Test Point Insertion

by

Joshua Immanuel

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 14, 2019

Keywords: DFT, Artificial Neural Networks, Testability Analysis

Approved by

Spencer Millican, Chair, Assistant Professor Electrical and Computer Engineering
Thaddeus Roppel, Associate Professor Electrical and Computer Engineering
Ujjwal Guin, Assistant Professor Electrical and Computer Engineering

Abstract

This thesis presents an artificial neural network signal probability predictor for VLSI circuits that considers reconvergent *fan-outs*. Testability analysis techniques are useful in the insertion of testpoints to improve circuit testability. Unfortunately, reconvergent *fan-outs* in digital circuits creates inaccurate testability analysis. Conventional testability analysis methods like COP do not consider reconvergent *fan-outs* and can degrade test point quality, while more advanced methods can increase test point analysis time significantly. This study shows that the training and use of artificial neural network to predict signal probabilities increases post-test point insertion fault coverage compared to using COP, especially in circuits with many reconvergent *fan-outs* per gate.

Acknowledgments

I am grateful for the grace of Almighty God and His Providence that has benevolently guided me throughout my research. All the wisdom and strength to accomplish the tasks necessary for my research, I attribute to His grace and mercy towards me.

I am grateful to Dr. Spencer Millican, my research advisor, who guided me and provided me with all the resources required to do my research. He provided much needed advice and encouragement as I progressed in my research, enabling me to stay motivated and complete my thesis.

I am thankful to my parents for their prayers, support and encouragement throughout my masters.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

DFT	Design for Test
BIST	Built in Self-Test
ANN	Artificial Neural Network
CC	Controllability
CO	Observability
SP	Signal Probability
TP	Test Point
TPI	Test Point Insertion
FC	Fault Coverage
SA0	Stuck-at-0 Fault
SA1	Stuck-at-1 Fault
CUT	Circuit Under Test

CHAPTER I. INTRODUCTION

VLSI (Very Large-Scale Integration) chips have become commonplace in all industries. The wide adoption of the MOS transistor, developed in 1959, made VLSI possible. The first commercial MOS integrated circuit was introduced by General Microelectronics in the 1964 [1]. The initial MOS integrated circuit technology allowed for 10,000 transistors in each chip. This was developed into the later VLSI technology with over tens and hundreds of thousands of transistors in each chip.

The rapid rise in logic density of VLSI chips has also seen a rise in issues related to defects and testing of those chips. Failing to identify and eliminate defects in a chip leads to massive losses during manufacturing [2]. Selling a defective chip can lead to critical issues on the consumer end, especially in safety-critical applications. It is imperative to invest in testing VLSI chips to identify defective chips to detect defects and to eliminate faulty chips during manufacturing.

A defect in a VLSI chip is an unintended difference between its specifications by design and its hardware implementation [3]. They can occur on a chip either during manufacturing or during use. Some defects in a VLSI chip include parasitic transistors, surface impurities, dielectric breakdown and seal leaks [4]. A defect may or may not affect the function of a circuit. When a defect affects the function of a circuit, it is represented as a fault.

A fault is an abstract functional representation of a circuit defect [3]. While a defect is an imperfection in a circuit's hardware, a fault is an imperfection in its function caused by a defect. Various fault models have been proposed to model defects but the single stuck-at fault model is the most commonly used fault model [3] and is the fault model used in this thesis. This will be elaborated on in Chapter II.

Detecting faults is a critical part of VLSI testing. Stuck-at faults are detected when they are propagated to circuit outputs. This is done by generating a test for a circuit with an automatic test pattern generation (ATPG) tool. Faults which do not affect the input-output function of a circuit are redundant faults [3] and it can be removed from a circuit without affecting its function. Reconvergent *fan-outs* are circuit structures that can possibly cause redundant faults and are created when when a net fans out and reconverges at a further on in the circuit.

Testability analysis is a measure of difficulty to control or observe a net on a circuit which is acquired through a topological analysis of the circuit without test vectors [3]. The controllability (*CC*) of a net is the probability the net will be logic-1 when stimulus is applied. Thus, the *CC* of a circuit input is 0.5, presuming input stimulus is random. The observability (*CO*) of a circuit is the probability that a net in a circuit can be observed. The *CO* of a circuit output is always 1. Traditional testability analysis methods include COP [5] and SCOAP [6]. COP is explained in detail in Chapter II.

Calculating *CC* and *CO* values can identify the probability that a fault will propagate through a net and can be excited in a circuit. The *CC* value of a net is the probability that the net will have a value of logic-1 when stimulus is applied to circuit inputs. The *CC* value subtracted by 1 is the probability that the net will have a value of logic-0 when stimulus is applied to circuit inputs.

Problem Statement

Reconvergent *fan-outs* become a major challenge for conventional testability analysis. Since it is virtually impossible to reduce the amount of reconvergence due to intrinsic pin limitation in VLSI circuits, there is a need for a testability analysis method that takes into consideration the existence of reconvergent *fan-outs* without increasing time overhead. Accurate testability analysis

can be determined by circuit simulation with all possible vector combinations: however, this is not feasible for larger circuits as the computation time for such a simulation is exponentially large. Conventional testability analysis treats signals at reconvergent *fan-outs* as independent but this is not the case always.

Artificial Neural Networks (ANNs) are computer algorithms that model biological neural networks. They have been shown to perform better than traditional heuristic approaches in solving non-linear and difficult-to-solve problems. They can consider circuit structures to provide improved testability analysis compared to traditional methods in reasonable time. ANNs have been elaborated in detail in Chapter IV.

Thesis Contributions

This thesis presents an alternative to conventional testability analysis using ANNs to predict the *CCs* of the circuit under test (CUT). Unlike conventional strategies the ANN predictor considers reconvergent fanounts of the first degree in a circuit and overcomes a major shortcoming in the accurate prediction of a circuit's testability. It will be shown that these ANNs predict signal probabilities (SP) and create more accurate testability measures compared to conventional methods. The ANN-predicted *CCs* will then be used for test point insertion (TPI) and it will be verified that the ANN-based SP predictor improves FC compared to conventional testability analysis. The novel contributions of this thesis are as follows:

- An ANN that predicts circuit *CC* through training by random simulation.
- An analysis comparing the ANN predictor against COP in predicting true SP.
- The application of the ANN predictor to increase FC in TPI compared to conventional testability analysis.

Thesis Organization

The remainder of this thesis is organized as follows. Section II provides background information, terms, and definitions to better understand the problem at hand and solutions proposed. Section III reviews past work done by other researchers in the field of VLSI test in testability analysis. Section IV elaborates on neural networks, their advantages, and their application in the field of VLSI test. Section V describes the method by which the theoretical proposals were practically implemented in a simulation environment. Section VI briefly describes the experiments which were run to train and test the ANN. Section VII describes the results obtained and confirms that the method proposed in this thesis are a better alternative to conventional methods. Section VII proposes some future avenues of research and section VIII concludes this thesis.

CHAPTER II. BACKGROUND

Single Stuck-At Fault Model

The single stuck-at fault model assumes a circuit to be an interconnection of Boolean gates [3] and a stuck-at fault is assumed to affect only an interconnection between gates. There are two types of stuck-at faults, namely stuck-at-0 (sa0) and stuck-at-1(sa1). A line is sa1 if it is set to logic-1 irrespective of the function of its driving node. A line is sa0, if it is set to logic-0 irrespective of the function of its driving node.

The number of stuck-at faults in a circuit depends on the number of interconnections. For a circuit with n interconnections, there are $2n$ stuck at faults. Each interconnection has a sa0 and sa1 fault.

Figure 1: An example of a sa0 fault in an AND gate.

Figure 1 portrays a sa0 fault on an AND gate. The inputs to the AND gate are set to logic 1 and hence the expected output is logic 1. However, since line *A* is sa0, the output of the AND gate will always be logic 0.

Fault Detection

Fault detection requires that a fault be controlled and observed. This means that the fault must be excited by setting its net to a certain value and then the fault's effect must be propagated to a circuit output. The fault model determines the value of a line to excite the fault: a line that is sa0 must be set to logic-1 to detect the fault whereas it is set to logic-0 for sa1. Creating tests for fault detection can be done either using an ATPG tool or fault simulation.

ATPG

ATPG is a tool used to generate tests for a circuit to detect faults. A test generated by an ATPG for a fault must be able to detect and propagate the fault to an output. ATPG may initially apply random patterns that detect as many faults as possible and after this, tests are generated for the remaining faults. Popular ATPG tools include D-algorithm [7], PODEM [8] and FAN [9].

Fault activation and fault propagation are procedures applied to a fault to detect it. Fault activation sets a line to the opposite value of the type of fault. Once a fault is activated, it must then be propagated through logic gates to a circuit output. For fault propagation, all non-fault inputs of a gate need to be set to its non-controlling value. Eg., to propagate a fault through an AND gate, all other inputs of the AND gate must be set to logic-1, otherwise the output of the AND gate will be forced to logic-0 and the effect of the fault will be destroyed.

Fault Simulation

Fault simulation is a method of fault detection that consists of simulating a circuit with faults and then comparing results of the faulty circuit with a good circuit with the same patterns. This comparison shows which faults are detected under the given set of patterns. If the response of the good circuit and faulty circuit are different in the same pattern, the fault is detected.

FC is a quantitative measure of the testability of a set of test patterns. It is defined as the ratio of the number of faults detected by a test to the total number of faults. To get better tests which achieve high FC, fault simulation is used to check if the FC of a test is adequately high. Once a pre-determined FC threshold is reached (possibly 99.99%), test generation is stopped.

Redundant Faults

A fault is redundant if it does not affect the input-output function of the circuit. The removal of redundant faults can simplify a circuit. Redundant faults cannot be activated by applying test vectors, as they do not change the output response.

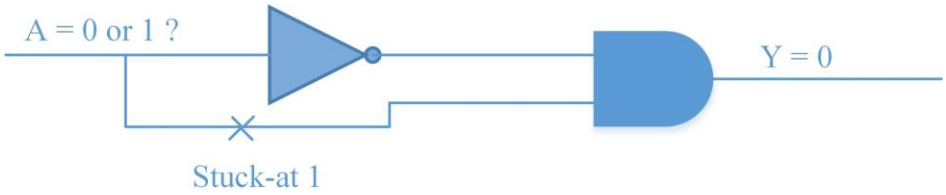

Figure 2: An example of a redundant sa1 fault in a circuit.

Figure 2 depicts a redundant fault in a logic circuit. To detect this fault A needs to be set to logic-1. However, doing so causes the output Y to be logic-0 because of the inverter. Setting A to logic-0 on the other hand, would mean that the fault is not excited. There is thus no possible vector to excite and propagate the fault to the output and it is thus a redundant fault.

COP

COP [5] is a common testability analysis tool which uses a probabilistic heuristic to determine circuit testability. COP predicts the *CC* of a net by usimg the input *CCs* according to the driving gate type. Observability can then be calculated by using these *CC* values. Thus, *CC* must first be calculated prior to observability.

The output *CC* of a gate in a circuit is a function of the combinational *CCs* of its input nets. Let the output *CC* of a gate be *OCC* and the input *CCs* be *ICC*. Thus, for an AND gate with two inputs, the output *CC* is the product of its input *CCs*. This is because the output of an AND gate is logic-1 only if all the inputs are logic-1 –

$$OCC = ICC1 * ICC2$$

Likewise, for an OR gate, the output *CC* is the product of each input *CCs* subtracted from one. This because for an OR gate the output is logic-1 if any of the gates are logic-1 –

$$OCC = 1-((1-ICC1) *(1-ICC2))$$

For an Inverter, the output controllability is merely its input controllability subtracted from one. This is because the inverter passes the opposite of its input to the output –

$$OCC = 1-ICC$$

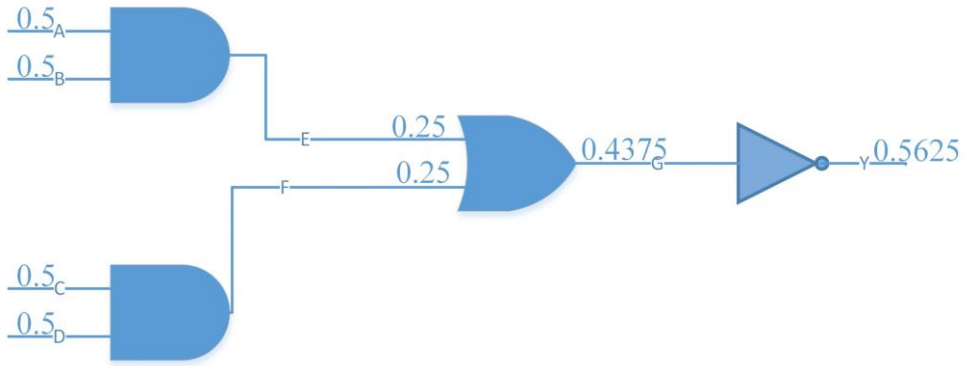

Figure 3: An example of COP based circuit controllabilities calculated for a simple combinational circuit.

For a buffer, the output *CC* is the same as the input *CC*. This is because a buffer merely allows a signal to pass through and thus does not affect its value.

Figure 3 illustrates the COP controllability calculations for an example combinational logic circuit. Note how the primary input *CCs* are 0.5 or 50%. This is because circuit inputs are presumed to be to random values. This initial value is then used to calculate the testability analysis for the rest of the circuit. For instance, the driver of *E* is an AND gate, hence the *CC* of *E* is the product

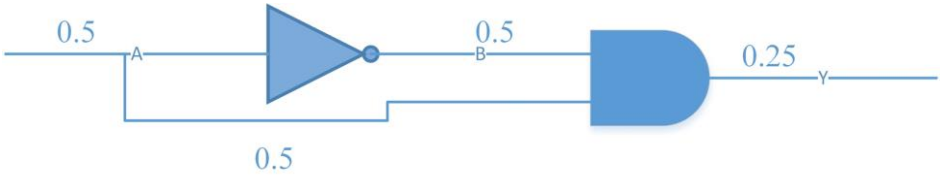

Figure 4: The case of the reconvergent *fan-out*, the shortcoming of conventional testability analysis.

of the *CCs* of *A* and *B*. For such a simple circuit, COP-calculated *CC* will be highly accurate in predicting true *CC*. This is because there are no reconvergent *fan-outs* and thus the *CC* values are truly a function of their driving node.

Ideally, the *CC* and *CO* values of each net are a function of its driver node but in a realistic circuit, the presence of reconvergent *fan-outs* can change this result. This non-linearity in the case of reconvergent *fan-outs* cannot be predicted by COP, as it uses a probabilistic heuristic that only considers the driver node and its inputs.

Figure 4 illustrates an example of the reconvergent *fan-out* creating inaccurate *CC* values. The COP-based *CC* values show that the output of the AND gate will be a logic-1 25% of the time but an analysis of the circuit would show that there is no input combination that can set the output to one. This circuit shows a shortcoming of the conventional COP-calculated testability analysis in certain circuits.

As is evident from the equations, COP solely depends on a gate and its inputs to calculate *CCs*. Thus, the heuristic to calculate *CCs* would be the same at any location in the circuit. Since, circuit structure is not part of this heuristic, reconvergent *fan-outs* can cause incorrect testability analysis results in circuits that have them.

Design for Test (DFT)

DFT is a wide range of design practices that improve and enable circuit testability. DFT practices include avoiding the use of synchronous logic feedbacks, making flip-flops initializable, and avoiding gates with a large number of fan-in signals and providing test control for difficult-to-control signals [10].

The presence of such structures can cause poor *CC* and *CO* and lower FC. It is possible for expert test engineers and designers to detect such structures on logic circuits. However, as the size

of a circuit increases it becomes infeasible to rely on manual examination to improve circuit structure. Human testability is also unpredictable and inconsistent.

Testability analysis can be used as a tool to guide DFT and test generation using ATPG [6]. Thus, it is imperative to have an accurate testability analysis metric for improved DFT.

Build in Self-Test (BIST)

BIST is a DFT technique that enables a circuit to test itself. BIST structures are capable of test vector generation and verifying its internal functionality. A BIST circuit includes a test pattern generator and an output response comparator that compares the expected output with the actual circuit output. It consists of circuitry that controls the tests called a test controller.

The exponential increase in logic-to-pin ratio in VLSI chips makes observing signals on chips increasingly difficult [11] but BIST provides a hierarchal method to divide a system-under-test into sub-assemblies that can be tested in a BIST cycle. BIST localizes circuit testing, eliminating many of the problems associated with system level testing [11].

BIST does not guarantee that all faults in a circuit are detected. Certain lines in a circuit may be hard-to-control and/or hard-to-observe. To help detect faults on such lines, TPs are used.

Test Point (TP)

A TP is a circuit modification that allows a net to be controlled or observed. They are logic gates that can improve controllability and observability. A control point sets a net to a constant value whereas an observe point enables the value on the net to be observed. TPs are only enabled when a circuit is under test mode and disabled otherwise. This means that TPs do not affect the regular functioning of the circuit unless it is in test mode.

A measure or heuristic is needed to evaluate the quality of TPs to select the best set of control and observe points in a circuit. This is done with the use of test point insertion algorithms.

Test Point Insertion (TPI)

TPI is the process of choosing the best set of control and observe points in a circuit to obtain the best FC possible. Probabilistic fault simulation has been suggested as a means for TPI in the past [12]. A probabilistic fault simulator computes fault detection probabilities using analytical equations. This is used to determine which signals receive control or observe points. Testability analysis methods like COP have been used to assist TPI in the past [13].

CHAPTER III. RELATED WORK AND MOTIVATION

Testability analysis has been used to improve circuit testability by improving TPI [14]. There exists a correlation between results obtained by testability analysis and the probability of detecting faults [15]. F. Brglez [5] demonstrated the testability of a digital circuits as a function of the difficulty of controlling circuit values from its inputs and observing them from its outputs. He proposed that the testability measures could be utilized to guide circuit design for improved testability and to enhance vector generation algorithms for better test generation.

Testability Analysis for Improved Circuit Testability

Conventional testability analysis does not take into consideration the existence of reconvergent *fan-outs*. J. Savir [16] demonstrated this shortcoming of conventional testability analysis by showing that it is not possible to conclusively determine the testability of a circuit solely based on controllability and observability values. Most industrial circuits have many *fan-outs*: in fact, over half of the nodes in conventional VLSI circuits are *fan-out* nodes, as demonstrated by Ratiu et al. [17]. Since VLSI circuits have a limited number of pins, it is consequential that the majority of *fan-out* branches will reconverge [18]. Large amounts of reconvergent *fan-outs* in a VLSI circuit makes it impractical for designers to provide information regarding such structures [17]. Knowledge of *fan-out* locations and the point of reconvergence can be useful to improve circuit testability. Roberts et al. [19] demonstrated the importance of locating all reconvergences in a circuit to improve testability analysis and presented an algorithm to detect all reconvergences. This algorithm however, was found to be insufficient as it does not detect reconvergent *fan-outs* in all cases [18]. It also requires *fan-out* data be provided to existing testability analysis methods to improve their performance. However, this adds to the time

complexity of testability analysis as it requires that conventional testability analysis be first calculated.

The use of testability analysis for enhancing circuit testability has been studied in detail in prior studies. Methods such as gate count, test vector size and controllability/observability matrices for linear sequential machines have been suggested in the past. Dejka [20] took these methods into consideration and compared them and it was found that all these methods have inadequacies. Gate count is an unreliable measure of circuit testability and most circuits used in industry are not linear sequential in nature, thus it is not possible to reliably use these measures as feedback to change circuit topology or the introduction of TPs.

Previous work attempted to provide a reliable testability analysis method. Stephenson et al. [21] analyzed circuits at the register transfer level to propose a testability metric normalized between zero and one. Brglez [5] presented a method of using combinational controllability and observability analysis of digital circuits. It was shown that these parameters could be used as feedback to design engineers to rearrange circuit topology to add TPs. This is because the obtained *CCs* and COs tell the design engineer details about faults that are hard-to-detect. The critical issue with Brglez's and Stephenson's method is that they don't take into consideration the existence of reconvergent *fan-out* branches in VLSI circuits. This can cause testability analysis results to not accurately model actual signal correlations in VLSI circuits that have reconvergent *fan-outs*.

Testability Analysis Methods Considering Reconvergent Fanouts

To overcome the issue of reconvergent *fan-outs* some methods have been proposed detect ing reconvergent *fan-outs*. Xu et al. [18] proposed an algorithm for identifying all reconvergent *fan-out* pairs and sites of reconvergence in a circuit with an algorithm that first detects all reconvergent pairs in a circuit and another algorithm that detects the *fan-out* branches that

reconverge at those sites. The time complexity of this algorithm was shown to be high ($O(n^3)$ at the least and $O(n^4)$ at the most). This implies that the algorithm will not provide a robust improvement to existing methods in all circumstances, especially in those where time is a constraint since an improvement to conventional testability analysis must also do so at a reasonable time to be applicable in an industrial environment.

Gu et al. [22] presented a testability analysis method that considers reconvergent *fan-outs* with an algorithm that finds a *fan-out* point and then checks if it is a reconvergent *fan-out*. This method does not present any empirical results to prove its effectiveness in application. There is a lot discussed about the theory and the algorithms used, but there has been no research into the usage of this method in industrial benchmark circuits to provide a reason to use it as a replacement to conventional methods.

Chang et al. [23] proposed a method that enhances the results obtained by COP and then improves the testability analysis results by using signal correlations. Their experiments showed that their method, TAIR provided higher accuracy than COP. However, TAIR adds to COP and requires that the regular COP results be obtained first before signal correlations are computed. This thesis' method computes signal correlations without any prior computation with COP, thus the runtime is comparable to COP, or as the results will show, faster than COP for some circuits.

Testability Analysis for TPI

Testability analysis methods have been used to determine the optimal selection of TPs to place at strategic locations to improve FC [14] and reduce area overhead [24]. However, the general shortcomings of conventional testability analysis impede the performance of TPI algorithm that uses it. Simulation of circuits with all possible vector combinations is the only means to provide accurate testability analysis for the TPI, but this is not practically feasible as the difficulty

to test a circuit for all possible vectors increases exponentially with an increase in circuit inputs. While it is possible to fully simulate all vector combinations in smaller circuits, most industrial VLSI circuits are too large to simulate all vector combinations.

CHAPTER IV. ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs) are a type of computer algorithm that models a biological neural network in a software or hardware. ANNs excel in solving problems that can be difficult to solve using traditional heuristic algorithms. This is because their training on the problem is done with a database of existing solutions (training data) till they achieve a high accuracy. In the past, the use of ANNs was highly restricted due to the lack of available training data and the lack of computational capabilities of older systems. At present, however, ANNs are an industrial standard for several applications.

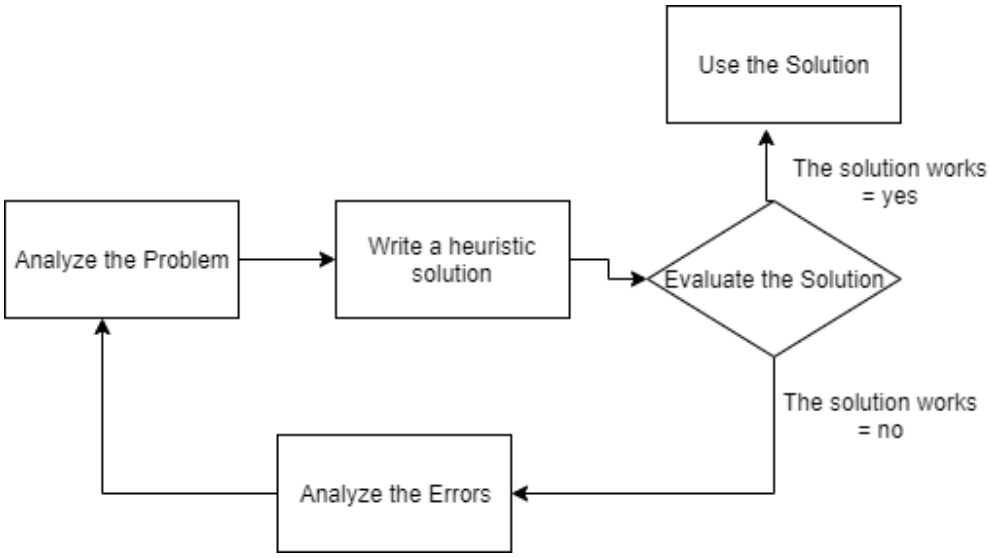

Figure 5: Traditional approach to solving a problem.

Figure 5 [25] depicts the traditional approach to solving a problem. This involves the step of studying the problem, figuring out possible solutions and heuristics. After this, rules are written to solve the problem. This may include either a formula or a complex probabilistic heuristic. The solution is then evaluated to test if it functions as required. If the heuristic solves the problem, it is chosen as the algorithm to the problem. If not, the solution needs to be analyzed again. The cycle is repeated till a solution is found.

Figure 6 [25] depicts the ANN method of solving a problem. This involves studying the problem but unlike traditional methods there is no need to write a heuristic or solution to the

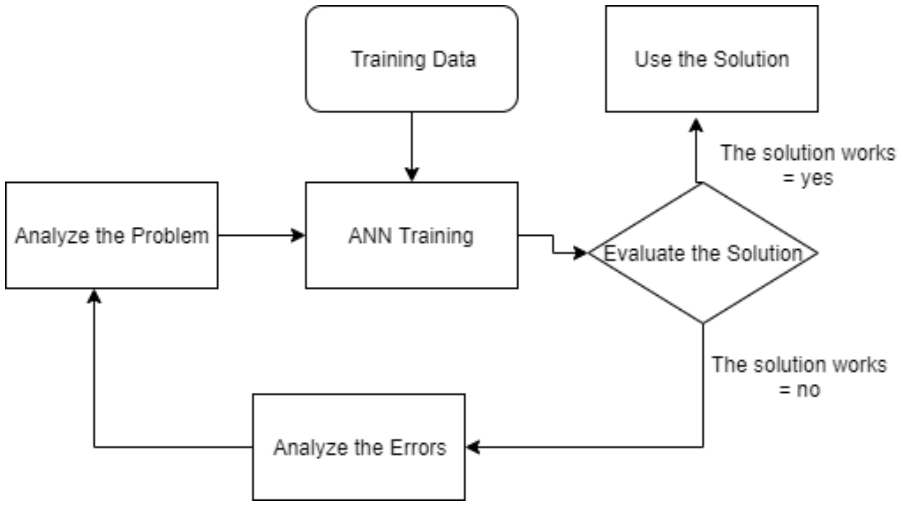

Figure 6: Neural Network Approach to Solving a Problem.

problem; rather, data is fed to the ANN model that trains itself from that data. This is used to come up with the algorithm. The algorithm is then analyzed either as good or bad. If an algorithm is good, it is chosen as the algorithm to the problem, if not the ANN keeps training itself till it reaches an optimal algorithm.

ANNs consist of input and output neurons separated by one or multiple layers of hidden layer neurons. The function of the hidden layer is to intervene between the input and output layer to increase accuracy [26]. A neuron is the base unit of an ANN that is its information processing unit. A single neuron consists of a set of connecting links called synapses, each of which has a weight parameter. These weight parameters can either be positive or negative. The input signals to a neuron are summed according to the synaptic weight of the neuron. An activation function is then used to limit the output amplitude of the neuron. A simple neuron is depicted in Figure 5.

Neurons can be structured in a diversity of architectures according to the learning algorithm used to train the neural network. A simple and commonly used architecture is the multi-layer

feedforward ANN. This architecture involves one or more hidden layers of neurons, an input layer, and an output layer. This is depicted in Figure 7 [27]. The scope of this thesis will be restricted to the use of the multilayer feed forward architecture. We will discuss this in detail in Chapter V.

ANNs need to be trained to accurately solve a problem. Training an ANN involves feeding it with a database of training data. The ANN then learns from the training data to predict outputs for a given set of inputs. This training process can take the neural network multiple iterations over a given set of training data till it reaches a certain accuracy. This process, depending on the architecture of the ANN and the size of the training data can be computationally intensive.

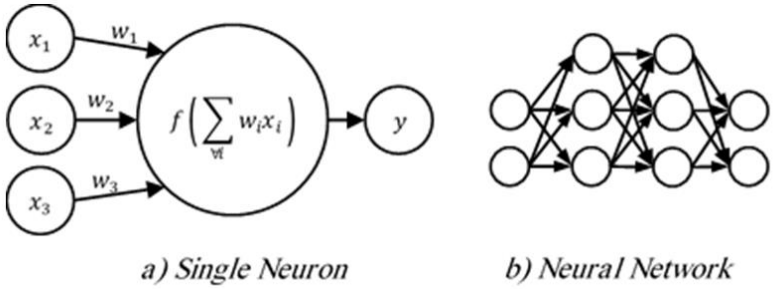

Figure 7: An example of a) a single neuron with input signals, input weights, and an activation function, and b) a neural network composed of multiple hidden layers. [20]

ANN Model Complexity

The size of the training database needed to generate a high quality ANN depends on the application and the complexity of the ANN model. There are two major issues that can occur with regards to the complexity of an ANN model –

- Overfitting – It is a condition that occurs in statistical models when the model captures the noise of the data. This is an issue that can occur in ANN training when the ANN model is too complex. This could primarily be due to a large width of input features that causes the ANN

model to take in the noise of the data. This results in a model that predicts with low bias but high variance. Overfitting can be prevented by breaking down the problem into smaller pieces.

- Underfitting – It is a condition in statistical models where the model is unable to classify the data. This occurs when an ANN model is too simple, or the width of the input features is too small. This implies that the neural network is unable to find the trend in the data provided to it. This results in a model that predicts with high bias but low variance. Underfitting can be prevented by providing sufficient input data and features.

The learning rate of an ANN is the size of the steps a model takes to correct itself. While training, if a neural network does not accurately predict an output feature, it corrects itself. It does that by changing its weights either positively or negatively. If the learning rate of a neural network is high, it will train quicker. However, this comes at the cost of lower ultimate accuracy. Having a smaller learning rate increases the time taken for training the ANN but increases its accuracy.

Supervised vs Unsupervised Learning

Supervised learning is method of training ANNs in which the training data consists of desired solutions to the problem, called labels [25]. Classification is an example of a supervised learning task. A spam filter is a supervised ANN classification algorithm: the ANN is fed with data from a lot of emails with a label of spam or ham. It trains itself with this data and classifies new emails based on this. Regression is another example of supervised learning: regression is used to predict a certain numeric value, like the price of housing. This is done by feeding the neural network with a set of features (location, square footage, rooms, etc.). The ANN can then learn to predict information when fed with relevant features. In this thesis we used a Regression based supervised learning model. This is further elaborated on in Chapter V.

Unsupervised learning is an ANN model that does not use any labels in its training data [25]. Clustering is an example of unsupervised learning. An ANN can be fed with a large dataset, and it can learn to find clusters of similar subsets within the dataset.

ANNs in VLSI Test

The application of ANNs to solve VLSI test problems is not new and ANNs have proved to serve as a solution for several NP-hard problems in VLSI test. For instance, in the field of fault diagnosis, Kabisatpathy et al. [28] demonstrated a pseudo-random test scheme that was based on ANNs. They proposed a technique that used the pseudorandom noise as a test pattern while using a model-based observer for fault diagnosis. Amnian et al. [29] proposed an ANN model for fault diagnosis in analog circuits: they used wavelet and Fourier transforms, normalization and principal component analysis to obtain the node voltages of VLSI circuits to generate training features for a neural network Catelani et al. [30] designed a backpropagation ANN that could identify the most likely faulty element responsible for the failure of a CUT. Madani et al. [31] presented multiple ANN models for fault diagnosis in analog circuits including back-propagation, learning vector quantization and radial basis function models. Meador et al. [32] demonstrated that while feedforward ANN classifiers did not improve diagnostic accuracy in integrated circuits they led to significant reduction in fault diagnosis costs once the network had been trained. Likewise in ATPG, Chakradhar et al. [33] proposed an ANN based algorithm that is suitable for massively parallel execution. In their method, the circuit is represented as a bidirectional network of neurons.

While ANNs have been used for several applications in VLSI test, to the best of the author's knowledge ANN algorithms have not been used directly in testability analysis for VLSI circuits. This provides an opportunity to implement ANNs to solve the current issues with

conventional testability analysis and take into consideration circuit structure to present more accurate testability analysis at a reasonable time.

CHAPTER V. METHOD

True CC

To get the true *CC* of every net in the circuit, it needs to be simulated with every possible input vector combination. Since this is not feasible and computationally intensive for larger circuits, it is necessary to obtain circuit testability with a high degree of certainty by simulating the circuit with many random vectors. For the purpose of these experiments this thesis has simulated all benchmark circuits 10000 random vectors.

Since SP is the probability that a net in the circuit will have the value of logic-1 when simulated with random vectors, it is obtained by dividing the number of times a net is logic-1 by the total number of simulations (10000).

ANN Model

The ANN needs a set of features to be trained on to predict the SP. Considering the ANN models discussed in Chapter IV, the regression based supervised learning model is ideal for the problem at hand. This is because regression is used in cases where the ANN is trained with supervised learning to predict a trend. *CC* values of each net in a circuit are a trend based on the structure of the circuit: the *CC* of a net is based on its driving node and reconvergent *fan-out* structures. Thus, training an ANN with enough data on nets, their structures and their *CC* values can make an ANN that can predict SPs for circuit nets.

Circuit Expansion

The training dataset of the ANN requires that the generated data be in a consistent order. For this, must be ensured that each node in the circuit has a constant number of inputs and outputs. The circuit file is modified to change all nodes that have more than two inputs to two inputs and

more than two outputs to two outputs. This is done without changing the functionality of the circuit. This has been previously suggested by Yang et. al. [34]

Input Features and Data Generation

The inputs to the ANN are a stream of floating-point and binary valsues that represent the SPs of nets and gate types respectively. A gate is represented as a binary stream of 8 bits, with all bits except one set to zero. The activated bit indicates the type of the gate. Table 1 describes binary stream for each gate type.

TABLE 1
BINARY STREAM FOR GATE TYPES

Gate Type	Binary Stream
AND	1,0,0,0,0,0,0,0
NAND	0,1,0,0,0,0,0,0
OR	0,0,1,0,0,0,0,0
NOR	0,0,0,1,0,0,0,0
XOR	0,0,0,0,1,0,0,0
XNOR	0,0,0,0,0,1,0,0
BUF	0,0,0,0,0,0,1,0
NOT	0,0,0,0,0,0,0,1

The proposed ANN considers reconvergent *fan-outs* of the first degree. To achieve this goal a backtracing algorithm is written that returns a list of nodes that are traced in the backward direction from the chosen net. This algorithm is described as Algorithm I. The backtracer object stores the root node `(RN)` that is the driver of the root net `(Rnet)` and a list of traced nodes `(TN)`. The nets of each level are stored as current level nets `(CLNet)` while the nodes are stored as current level nodes `(CLN)`.

```
Algorithm I – Backtrace
Inputs: Circuit c, Net n
Begin
Step 1: For the net n on circuit c find its driving node
(DN(n)).
            RN = DN (n)
Step 2: For the RN get a list of its inputs.
            CLNet = input nets of RN
Step 3: For every CLN get a list of its driving node
          CLN = CLNet(n)
Step 4: CLN = TN
End
```

Reconvergent *fan-outs* are then detected by checking if the `TN` input nets are the same as the either of the `Rnets`. If they are the same, then it indicates a reconvergent *fan-out* from that net to the root net. If a reconvergent exists a binary parameter is set to one else, it is set to zero. Not only does this detect all *fan-out* branches, but it also detects the actual location of reconvergence.

The type of gate, reonvergent *fan-outs* and SP values are collected for each net of the circuit that is not a primary input. If the driving node of a net is a NOT or a BUFFER gate, it is not used in data generation. This is because the output *CC* of a NOT gate is merely 1 subtracted from its input *CC* whereas the output *CC* is the same as the input *CC* for a BUFFER, thus making them irrelevant to train the ANN, as these can be calculated accurately for any case.

ANN Architecture

The ANN is fed with a thirty-eight feature input represented by floating-point values. Hence, the input layer of the ANN has thirty-eight neurons. Since the ANN model needs to predict the SP of a net, there is only need for one neuron in the output layer.

In this thesis, a single hidden layer was used for the ANN architecture. This is because it has been shown that in feedforward ANNs, a single hidden layer can represent a wide range of functions given appropriate input features [35]: this is called universal approximation theorem. Adding more hidden layers seldom improves the performance of an ANN as a single hidden layer

is enough to solve most problems. It was shown that the performance of multilayer feedforward ANN's as universal approximators was not restricted to choice of activation function [36].

There is no agreed standard on choosing the number of hidden layer neurons for an ANN. To select the optimal number of hidden layer neurons cross-validation is performed: cross-validation results showed that twenty-eight neurons in the hidden layer was the most optimal in giving higher training accuracy. Thus, the ANN architecture is 38-28-1, representing the input layer, hidden layer, and output layer neurons, respectively.

A multilayer feedforward ANN was designed in the Python language using the tensorflow libraries. The activation function chosen for this experiment is the sigmoid function. This is because the sigmoid function exists between 0 and 1 and is thus ideal for representing SP, since any kind of probability function exists between 0 and 1. The sigmoid activation function is represented as follows –

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

ANN SP Prediction

Once the ANN model is trained, it can be used to predict SP values of nets in a circuit. To do this, the trained ANN weights are stored in a matrix. Since there are three layers in the selected ANN model (1 input, 1 hidden, 1 output) in the 38-28-1 architecture, there are two weight matrices: the first weight ($W1$) is between the input layer and the hidden layer. It has 38 rows and 28 columns and the second weight ($W2$) is between the hidden layer and the output layer. It has 28 rows and 1 column.

For each net, the backtracer algorithm generates data that is stored as a stream of floating-point values in the vector. This input vector (IP) is thus a matrix of 1 row and 38 columns.

The first layer calculation is the activated matrix multiplication of IP with $W1$. This matrix H has 1 row and 28 columns. The second layer calculation is the activated matrix multiplication of H and $W2$. This multiplication result, O is the predicted SP of the net with input feature IP.

$$H = sigmoid(IP * W1)$$

$$O = sigmoid(H * W2)$$

TPI Methodology

Conventional TP architectures are used for this study since they are the most commonly used in industrial settings. A control TP sets a net on a circuit to a constant value when in test mode. There are two types of control TPs namely, control-0 TPs and control-1 TPs. Control-0 TPs set the value of a net on the circuit to a constant logic-0. A control-1 TP, on the other hand, sets a net on the circuit to a constant value of one.

This paper does not present a novel test TPI method but rather uses an existing TPI algorithm [13] to compare two distinct testability analysis methods. The original algorithm uses COP testability analysis to detect the probability of detecting a fault. This probabilistic detection of faults is used to maximize the FC of a circuit. The probability for fault detection (Pi) depends on the type of the fault, i.e., whether it is a sa0 or sa1 fault:

$$Pi = Controllability * Observability;\ For\ sa0$$

$$Pi = (1 - Controllability) * Observability; For\ sa1$$

$$FC = \left(\frac{1}{faults}\right) * \sum_{\forall i \in faults} Pi$$

The Pi for sa0 faults is a product of *CC* and *CO*. This is because a sa0 fault is excited by a logic-1 signal. Likewise, for sa1 faults, the Pi is the product of the *CC* subtracted from one and

the *CO*. This is because a sa1 fault is excited by a logic-0 signal. The FC is calculated as the ratio of the sum of probabilities of detection of all faults to the total number of faults.

CHAPTER VI. EXPERIMENT

Experimental Setup

All experiments were run on a high-performance workstation that is representative of an industrial environment. The workstation is equipped with an Intel i7-8700 processor and 8GB of RAM.

The framework of all experiments was a circuit simulation and test toolkit developed in the C++ programming language that represents circuits as a collection of interconnected nets and nodes. This structure models single stuck-at faults well since stuck-at faults exist on interconnections between circuit nodes as discussed in Chapter II. The circuit object acts as a wrapper that holds vectors of circuit nets, nodes, inputs and outputs. A net is a data structure used to model circuit interconnections that can have a single driver and one or multiple outputs. It is not possible for a net to have more than one driver as two or more gates cannot drive a single line in logic circuits. A node is a data structure used to model logic gates in a circuit. It can have multiple inputs and multiple outputs. It works in conjunction with nets. The toolkit consists of several tools that are useful for the experiment like a benchmark parsing, simulator, COP testability analysis calculator, TPI, fault simulation etc.

Data Generation

Training data is generated from the eleven circuits of the ISCAS-85 benchmarks. Every net in the benchmarks, excluding circuit inputs and nets driven by inverters and buffers, are used for data generation, resulting in a total of 11,118 data points to train the neural network. The generated data is stored in a comma-separated value (CSV) format.

ANN Training

The ANN is trained with the data generated from the ISCAS-85 benchmark circuits. The ANN is trained over the dataset for 225 epochs-iterations: an epoch is the number of training iterations equal to the size of the dataset, thus, in this case an epoch is 11,118 iterations. 225 epochs was enough to give a training accuracy of 99%. The weights generated while training the ANN are stored for later use during TPI.

ANN Prediction

The ANN weights are then read by a function that predicts the SPs of each net by running through each net in the circuit. This is done on the ITC-02 benchmark circuits on circuits b01 to b15. If a net is a circuit input it is assigned an SP of 0.5 by default. This can be changed to a more weighted value if desired, but the standard SP is 0.5. Likewise, for a net driven by an inverter, the SP is resolved as the SP of the input of the inverter subtracted from 1. For a net driven by a buffer, the SP of the buffer's input is propagated forward as is. The results are then compared with those of COP against SP by simulation with random vectors.

TPI Procedure

The predicted SP values are then used for TPI. This is done for the ITC 02 benchmarks from b01 to b15. The same is done with COP based SP prediction and the resultant FCs and TPI times are compared.

The constraints placed on TPI was to restrict it to a TP limit of 1% of the number of gates in the circuit and a time limit of two hours. To make a fair comparison, both methods use the same number of TPs for every benchmark.

CHAPTER VII. RESULTS

Table 2 shows the results of the ANN SP predictor compared to COP in the ITC-02 benchmark circuits. The three columns represent the accuracy of COP compared to SP by random simulation, the accuracy of the ANN predictor compared to SP by random simulation, and the increase in accuracy for the ANN predictor over COP in predicting SP by random simulation. The accuracy of both COP and the ANN predictor is calculated as a percentage of the absolute difference of the SP value by random simulation and the predicted SP value subtracted from 1.

The SP values by random simulation are representative of true testability of the net in the circuit. Experimental results showed that the use of 10,000 vectors for circuit simulation resulted in almost constant SP values for circuit nets, demonstrating the reliability of using SP values by random vector simulation as a representation of true circuit testability.

TABLE 2
COP VS NEURAL NETWORK ACCURACY COMPARISON

Bench.	COP Accuracy (%)	ANN Accuracy (%)	Accuracy Increase (%)
b01	98.4681	99.2385	0.7704
b02	98.3745	98.7649	0.3904
b03	99.8015	99.9016	0.1001
b04	99.9848	99.9945	0.0097
b05	99.9214	99.9590	0.0376
b06	99.3791	99.6840	0.3049
b07	99.9630	99.9820	0.0190
b08	99.8111	99.9544	0.1433
b09	99.8004	99.9000	0.0996
b10	99.8512	99.9284	0.0772
b11	99.9134	99.9590	0.0456
b12	99.9576	99.9787	0.0211
b13	99.7542	99.877	0.1228
b14	99.9970	99.9982	0.0012
b15	99.9914	99.9911	-0.0003

The results demonstrate that the ANN predictor performs consistently better than COP in predicting the SP values for all the tested ITC-02 benchmark circuits except b15. The set increase is attributed to the fact that reconvergent *fan-outs* that cause redundant faults are a small proportion of circuit nets. COP is generally accurate in predicting the circuit testability analysis in the absence of such redundant structures.

A possible reason why COP is slightly more accurate than the ANN in the b15 benchmark is because of its large size. The ANN uses a more complex heuristic to predict SP that considers the SP of nets and nodes 2 levels back, whereas COP's heuristic only uses the driver node and its input SPs. This can cause slight variation in prediction of results, especially in larger circuits.

Figure 8 demonstrates the case of the reconvergent *fan-out* in which the ANN results are significantly more accurate than COP in predicting SP by random simulation. There is no possible scenario in which the output of this circuit can be set to logic 1, hence the output *CC* must be 0.

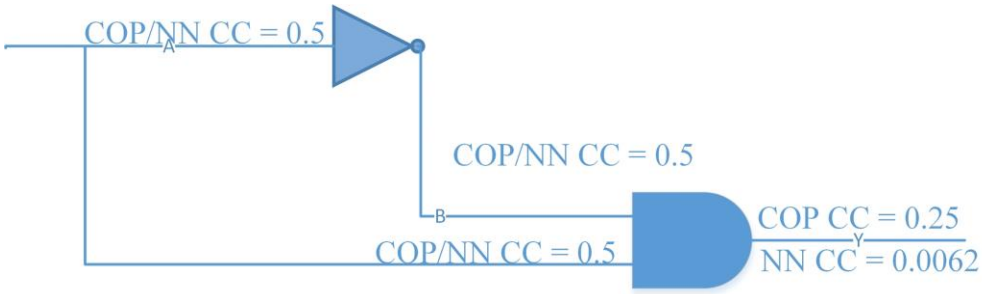

Fig 8: The case of the reconvergent *fan-out*. The NN is far more accurate than COP in representing testability analysis in a circuit with reconvergent *fan-outs*.

This SP of the output of the AND gate is 0 after 10000 random vectors are applied to the circuit. COP however predicts the output *CC* as 0.25, which implies that the net can be logic 1, 25% of the time. The ANN is close in its prediction of 0.62%. It is most likely instances like this that makes the ANN is more accurate than COP for most benchmark circuits in table 1.

Table 3 shows the results of TPI with 10,000 random vectors on the ITC-02 benchmark circuits. Only circuits whose original FC was lower than 100% are shown since it is irrelevant to insert TPs in a circuit that already has 100% FC.

The "Recfo" column shows the number of reconvergent *fan-outs* of the 10^{th} degree, i.e., 10 levels back from each net. While the ANN itself only considers reconvergent *fan-outs* of the 1^{st} degree, ANNs are complex algorithms that train themselves with the provided data, hence, it is possible that the ANN performs better even in nets with reconvergent *fan-outs* on higher degrees.

The columns “Ctrl TPs” and “Obs TPs” represent the number of control and observe points inserted for COP and the ANN predictor. The “TPI FC” columns indicate the FC of the circuit after TPI. Additional columns indicate the time (in seconds) for TPI. The Recfo/Gate column indicates the reconvergent *fan-outs* per gate in a circuit, which will be used as a metric of performance of the ANN over COP.

Table 3

Cop vs Neural Network Accuracy Comparison

Bench.	Gates	Recfo	TP Limit	COP Ctrl TPs	COP Obs TPs	ANN Ctrl TPs	ANN Obs TPs	Original FC (%)	COP TPI FC (%)	ANN TPI FC (%)	FC Increase (%)	COP TPI Time (s)	ANN TPI Time (s)	Time Decrease (s)	Recfo/Gate
b04	652	102	6	2	4	2	4	97.7074	99.4541	99.5669	0.1128	66	62	4	0.1564
b05	927	435	9	1	8	4	5	76.7460	78.7302	91.0482	12.318	203	194	9	0.4692
b07	383	23	3	2	1	2	1	97.7518	99.4604	99.2443	-0.2161	13	14	-1	0.0600
b11	726	105	7	0	7	7	0	95.3684	96.2105	96.5553	0.3448	50	46	4	0.1446
b12	944	100	9	3	6	3	6	94.1593	95.3628	99.4477	4.0849	244	226	18	0.1059
b14	9767	74	97	2	0	2	0	87.3098	88.1420	89.5209	1.3789	5292	4789	503	0.0075
b15	8367	1201	83	2	0	2	0	74.9869	97.104	99.9198	2.8158	4332	3843	489	0.1435

The results from Table 3 indicate the FC for ANN-based TPI is consistently higher than that for COP-based TPI for all benchmarks but one (b07). This discrepancy can be attributed to the low number of reonvergent *fan-outs* in this circuit, which is the lowest in the set of benchmarks. Even for this case, the decrease in FC is relatively small compared to increases elsewhere. b14 is a case where the FC does increase is relatively small. Again, this again is a circuit that has a very low reconvergent *fan-out* to gates ratio.

Table 3 also demonstrates that the ANN-based predictor takes a very similar amount of time as the COP-based predictor for TPI. As a matter of fact, it is significantly lower in certain circuits. This indicates that the use of a trained ANN for TPI does not add to the time complexity of the algorithm.

CHAPTER VIII. FUTURE WORK

Future studies must focus on expanding the ANN model to consider more circuit parameters, especially taking into consideration more levels back from each net. Eventually an ANN model can be designed that considers the whole circuit. This should provide an accurate testability analysis method. While the time complexity for training such an ANN could be very high, the trained model should still be reasonably time efficient. This would require more optimized neural network architectures to increase accuracy and decrease training time to account for the increase in input data width.

Another avenue of future research involves the prediction of true circuit observability using neural networks. This is a more complex problem than predicting true circuit controllability since true circuit observability requires more than random simulation of vectors.

CHAPTER IX. CONCLUSION

This thesis studied the use of ANNs for improving circuit testability by considering reconvergent *fan-outs*. It was shown that a testability analysis predictor using neural networks is more accurate than a COP-based predictor for testability analysis.

The ANN-based predictor was also shown to have the same time complexity as COP in the tested benchmarks, even showing better time performance in some of the benchmarks.

This method of testability analysis is then applied to TPI and it was shown that the ANN provides higher FC than the COP-based predictor. This is especially true of circuits with higher number of reconvergent *fan-outs* per gate.

CHAPTER X. REFERENCES

[1] A. S. Grove, E. H. Snow, B. E. Deal, and C. T. Sah, “Simple physical model for the space-charge capacitance of metal-oxide-semiconductor structures,” *J. Appl. Phys.*, vol. 49, pp. 2458–2460, 1964.

[2] I. Koren and Z. Koren, “Defect tolerance in VLSI circuits: Techniques and yield analysis,” *Proc. IEEE*, vol. 86, no. 9, pp. 1819–1838, 1998.

[3] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing*. 2000.

[4] M. . Howes and D. . Morgan, “Reliability and Degradation - Semiconductor Devices and Circuits,” *Wiley-Interscience*, 1981.

[5] F. Brglez, “On testability analysis of combinational networks.,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 1984.

[6] L. H. Goldstein, “Controllability/Observability Analysis of Digital Circuits,” *IEEE Trans. Circuits Syst.*, vol. 26, no. 9, pp. 685–693, 1979.

[7] J. P. Roth, “Diagnosis of Automata Failures: A Calculus and a Method,” *IBM J. Res. Dev.*, vol. 10, no. 4, pp. 278–291, 2010.

[8] P. Goel, “An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits,” *IEEE Trans. Comput.*, vol. C–30, no. 3, pp. 215–222, 1981.

[9] H. Fujiwara and T. Shimono, “On the Acceleration of Test Generation Algorithms,” in *IEEE Transactions on Computers*, 1983, pp. 64–69.

[10] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital systems testing and testable design, revised printing*. 1994.

[11] V. D. Agrawal, C. R. Kime, and K. K. Saluja, “A Tutorial on Built-ln Self-Test Part 1: Principles,” *IEEE Des. Test Comput.*, vol. 10, no. 1, pp. 73–82, 1993.

[12] J. Rajski and J. Tyszer, “Arithmetic Built-In Self-Test for embedded systems,” in *Upper Saddle River*, 1998.

[13] H. C. Tsai, K. T. Cheng, C. J. Lin, and S. Bhawmik, “Hybrid algorithm for test point selection for scan-based BIST,” in *Proc - Design Automation Conference*, 1997, pp. 478–483.

[14] S. Boubezari, E. Cerny, B. Kaminska, and B. Nadeau-Dostie, “Testability analysis and test-point insertion in RTL VHDL specifications for scan-based BIST,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1327–1340, 1999.

[15] V. D. Agrawal and M. R. Mercer, “Testability measures- what do they tell us?,” in *Proc. Int’l Test Conf*, 1982, pp. 391–396.

[16] J. Savir, “Good controllability and observability do not guarantee good testability,” *IEEE Trans. Comput.*, vol. C–32, no. 12, pp. 1198–1200, 1983.

[17] I. M. Ratiu, A. Sangiovanni-Vincentelli, and D. O. Pederson, “VICTOR: a fast vlsi testability analysis program.,” in *Proc. Intl. Test Conference*, 1982, pp. 391–396.

[18] S. Xu and E. Edirisuriya, “A new way of detecting reconvergent fanout branch pairs in logic circuits,” in *Proc. Asian Test Symposium*, 2004, pp. 354–357.

[19] M. W. Roberts and P. K. Lala, “Algorithm to detect reconvergent fanouts in logic circuits.,” *IEE Proc. E Comput. Digit. Tech.*, vol. 134, no. 2, pp. 105–111, 1987.

[20] W. J. Dejka, “Measure of testability in device and system design,” in *Proc. 20th Midwest Symposium on Circuits and Systems*, 1977, pp. 39–52.

[21] J. E. Stephenson, “A testability measure for register-transfer level digital circuits,” Carnegie-Mellon Univ., Pittsburgh, PA., 1974.

[22] X. Gu, K. Kuchcinski, and Z. Peng, “Testability measure with reconvergent fanout

analysis and its applications," *Microprocess. Microprogramming*, vol. 32, no. 1–5, pp. 835–842, 1991.

[23] S. C. Chang, W. Ben Jone, and S. Sen Chang, "TAIR: Testability analysis by implication reasoning," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 19, no. 1, pp. 152–160, 2000.

[24] J. S. Yang, N. A. Touba, and B. Nadeau-Dostie, "Test point insertion with control points driven by existing functional flip-flops," *IEEE Trans. Comput.*, vol. 61, no. 10, pp. 1473–1483, 2012.

[25] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017.

[26] S. O. Haykin, *Neural Networks and Learning Machines Third Edition*. Pearson, 2008.

[27] S. K. Millican, Y. Sun, S. Roy, and V. D. Agrawal, "Applying Neural Networks to Delay Fault Testing : Test Point Insertion and Random Circuit Training," in *IEEE Asian Test Symp.*, 2019, pp. 1–6.

[28] P. Kabisatpathy, A. Barua, and S. Sinha, "A pseudo-random testing scheme for analog integrated circuits using artificial neural network model-based observers," in *Proc. 45th Midwest Symposium on Circuits and Systems*, 2002, pp. 465–468.

[29] F. Aminian and M. Aminian, "Fault diagnosis of nonlinear analog circuits using neural networks with wavelet and fourier transforms as preprocessors," *J. Electron. Test. Theory Appl.*, vol. 17, no. 6, pp. 471–481, 2001.

[30] M. Catelani and M. Gori, "On the application of neural networks to fault diagnosis of electronic analog circuits," *Measurement*, vol. 17, no. 2, pp. 73–80, 1996.

[31] K. Madani, A. Bengharbi, and V. Amarger, "Neural fault diagnosis techniques for

nonlinear analog circuit," in *Proc. SPIE,* 1997, vol. 3077, pp. 491– 502.

[32] J. Meador, A. Wu, C. T. Tseng, and T. S. Lin, "Fast diagnosis of integrated circuit faults using feedforward neural networks," in *Proc : International Joint Conference on Neural Networks,* 1991, pp. 269–272.

[33] S. T. Chakradhar, M. L. Bushnell, and V. D. Agrawal, "Toward massively parallel automatic test generation," *IEEE Trans. Comput. Des. Integr. Circuits Syst.,* vol. 9, no. 9, pp. 981–994, 1990.

[34] Y. Sun and S. Millican, "Test Point Insertion Using Artificial Neural Networks," *Proc. IEEE Comput. Soc. Annu. Symp. VLSI, ISVLSI,* vol. 2019-July, pp. 253–258, 2019.

[35] G. Cybenko and G. Cybenkot, "Approximation by superpositions of a sigmoidal function.," *Math. Control Signals Syst.,* 1989.

[36] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks,* 1991.