Novel Test Point Insertion Applications in LBIST

by

Yang Sun

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
Dec 11, 2021

Keyword: Artificial Neural Network, Test Point Insertion, Logic Built-in Self-Test, Test Power

Approved by

Spencer K. Millican, Chair, Assistant Professor of Electrical and Computer Engineering
Vishwani D. Agrawal, Professor Emeritus of Electrical and Computer Engineering
Adit D. Singh, Godbold chair, Professor of Electrical and Computer Engineering
Ujjwal Guin, Assistant Professor of Electrical and Computer Engineering

Abstract

Pseudo-random stimulus is an established industry practice due to its simplicity and significant fault coverage. However, when applied to modern circuits, pseudo-random stimulus can fail to excite and observe *random pattern resistant* (RPR) faults. These faults become more common as logic circuitry becomes more complex, which naturally occurs with technology scaling. Many techniques attempt to detect RPR faults using *logic built-in self-test* (LBIST), including (1) modifying the pattern generator to create less "random" stimulus or (2) modifying circuits to make RPR faults more easily tested with random stimulus. This latter method, known as *test point* (TP) *insertion* (TPI), is frequently used in industry due to its ability to be implemented on post-synthesis circuit netlists with minimal effort from circuit designers. Optimal TPI is known to be an NP-hard problem, thus current TPI methods use heuristics to overcome computational barriers. *Artificial neural networks* (ANNs) are computing paradigms that present opportunities to increase solution quality and overcome computational barriers imposed by heuristic algorithms.

This dissertation studies ANN TPI algorithms. The methods include collecting training data, training ANNs, and analyzing ANNS to evaluate TPs. Experiments compare ANN-based TPI methods against equivalent heuristic algorithms in terms of circuit testability and computation time. Experimental results show ANN-based TPI can achieve high fault coverage with less execution time.

Another concern for *design-for-test* (DFT) engineers is high test power since high-power tests can cause false failures and circuits become less reliable when large and instantaneous power dissipation causes overheating. Thus, techniques must reduce test power while keeping fault coverage acceptable. Control TPs can reduce switching activity and keep lines stable, thus decrease power consumption during test. However, control TPs may also cause fault coverage to decrease,

thus some strategies must be used to balance power and fault coverage. This dissertation proposes power-targeting TPI. The method uses multi-phase strategies in power-targeting TPI to minimize the negative impacts on fault coverage. Experiments compare different numbers of TPI phases and finds the best number of TPI phases, and experimental results show power-targeting TPI can reduce test power while keeping fault coverage high.

Acknowledgments

Firstly, I am very grateful to my advisor Prof. Spencer K. Millican for his guidance. He inspired me a lot whenever I had problems with my research work. I also appreciate my other committee members: Prof. Vishwani D. Agrawal, Prof. Adit D. Singh, and Prof. Ujjwal Guin. They gave me lots of suggestions and helpful comments, so that I can modify and improve my dissertation research. Besides these professors, I would like to thank my team members, who helped me a lot in my research field. I couldn't have made a success in my research and other projects without their help. Last but not the least, I want to thank my family members and friends, who give support with their love during my whole study in Auburn.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

COP	Controllability Observability Program
CRF	Cost Reduction Factor
DFT	Design for Test
ECO	Engineering Change Orders
EDA	Electronic Design Automation
LBIST	Logic Built-in Self-Test
ML	Machine Learning
MPTI	Multi-Phase TPI
PDF	Path Delay Fault
PRPG	Pseudorandom Patten Generator
RPR	Random Pattern Resistant
SAF	Stuck-at Fault
SCOAP	Sandia Controllability/Observability Analysis Program
SoC	System-on-Chip
TDF	Transition Delay Fault
TP	Test Point
TPI	Test Point Insertion

Chapter 1

Introduction

Modern electronics in critical and high-assurance applications (e.g., self-driving cars, aerospace, and medical devices) have strict reliability requirements. Since defective devices create economic loss or catastrophic loss-of-life, manufacturing tests must be credible in detecting and preventing faulty behavior. To ensure the quality and correct operation of electronic products, tests must be done both during R&D (design verification & classification) and later in production (production/manufacturing testing). *Design-for-test* (DFT) describes the circuit design processes that make sure the required testing is possible to do, and preferably even easy to do.

DFT consists of *integrated circuit* (IC) design techniques that makes testing a chip possible and cost-effective by adding additional circuitry to the chip, adds testability features to a hardware product design to improve the controllability and observability of internal nodes. DFT makes it easier to develop and apply manufacturing tests to the designed hardware. DFT techniques include scan chains and *built-in self-test* (BIST) circuitry.

Logic built-in self-test (LBIST) [1] is a BIST technology in DFT that is commonly used for both manufacturing tests and post-manufacturing reliability checks [2]. LBIST uses on-chip stimulus generators, i.e., *pseudorandom pattern generators* (PRPGs) [3], [4] to stimulate circuit inputs and set circuit states while circuit outputs and states are observed. When complementing conventional test methods, LBIST can significantly increase fault coverage while decreasing test application time. With embedded LBIST, devices become testable with minimal functional interruption by saving the circuit state, applying test enable/disable signals, and then reloading the circuit state to resume the normal function.

A major challenge for LBIST is detecting *random pattern resistant* (RPR) faults [5]. RPR faults manifest in logic with many inputs when few input combinations can excite certain logic paths, and therefore pseudo-random tests often fail to excite and observe RPR faults. The prototypical example of an RPR fault is the output of a large logic gate; the probability of the output of a 32-input AND gate being logic-1 and exciting a stuck-at-0 fault at it's output is 2^{-32} (presuming all AND gate inputs are equally likely to be logic-0 or logic-1), which implies more than one billion pseudo-random patterns may be needed to excite the fault. Under the presence of RPR faults, applying LBIST becomes time-consuming and power-intensive, which means test costs increase and circuit reliability degrades.

A method to improve LBIST performance is modifying circuits with *test points* (TPs). TPs change circuit values or observe values in a circuit, thus making the detection of RPR faults easier. *Test point insertion* (TPI) techniques find high-quality TPs locations which improve fault coverage or reduce the required number of test patterns. Using TPs to increase random pattern effectiveness is well established in literature [6]–[8], but TPI methods still strive to improve their computational performance. Since the concept of TPI was proposed by Hayes and Friedman [6] in 1974, numerous algorithms have improved TPI performance by using fault simulation to classify RPR faults and propose TP locations [7] by using *automatic test pattern generation* (ATPG) [9], by using gradient optimizing techniques [8], and by incorporating many other nuances and using a variety of algorithms. All of these methods are less-than-optimal, which is necessary because optimal TP placement is a known computationally-infeasible problem [10]. Therefore, it is imperative to continue exploring TPI methods that select higher-quality TPs with less computational resources, since reduced execution time translates to increased TPI efforts, which in turn translates to higher-quality TPs being inserted into a circuit in a given amount time.

Unfortunately, the computational complexity of the algorithms that implement these techniques increases faster than the increasing size of logic circuits [10]. Since computational resources are in high demand by several *electronic design automation* (EDA) tool users during circuit development, designers must sacrifice testability or other circuit qualities if EDA engineers do not increase algorithm efficiency.

New computing methods, like *artificial neural networks* (ANNs), can increase algorithm efficiency and keep LBIST quality high. ANNs can solve complex problems, like image and speech recognition, and they can significantly increase the quality of existing algorithms while simultaneously decreasing computation time. Recently, ANNs have been applied to several EDA problems with noteworthy success [11], [12], but applying ANNs to DFT problems is in its infancy.

Another major challenge for LBIST is excessive test power. Excessive power not only discourages circuits in portable environments but also causes overheating which degrades performance and reduces chip life. Beyond chip functionality, excessive power during test increases manufacturing costs by requiring more expensive chip packaging or by reducing yield. During wafer test, wafer probes have current limits, and high switching activity during test increases power instability; this instability changes logic states and causes false failures, thus reducing yield. Existing TPI methods may reduce this high switch activity by inserting control TPs, but it may also come at the cost of reduced fault coverage. A new TPI procedure should be explored to reduce test power without affecting test quality, and TPs can be inserted after finding test power issues during silicon bring-up.

This dissertation explores using TPs to improve LBIST performance. Chapter 2 gives a survey on TP architectures and TPI methods and describes TPI methods for different purposes and their weakness. Chapter 3 introduces ANNs' history, structures, training methods and applications.

Chapter 4 and Chapter 5 present the methods of ANNs TPI targeting *stuck-at fault* (SAF) and targeting *transition delay fault* (TDF), respectively, including ANN creation, training and evaluation on SAT/TDF effectiveness and execution time. Chapter 6 futher develops the ANN for TPI, e.g., by exploring ANN input features, output labels, and a more detailed ANN parameters exploration. Chapter 7 demonstrates a procedure of power-targeting TPI with a multi-phase strategy and compares it against conventional fault-targeting TPI on SAF coverage, average power, and peak power.

Chapter 2

Background

2.1 TP Architectures

TPs are circuit modifications that change or observe circuit functions during test but do not change the circuit function when disabled [6], [7]. Conventional TPs are categorized into two types [13]: control TPs and observe TPs (as shown in Figure 2.1 (a), (b), and (d)). Control TPs are typically implemented using OR gates for control-1 TPs or AND gates for control-0 TPs (and NAND/NOR gates can be used at the output of inverters) [14]. During test, a test enable pin forces lines to their controlled values [15]. While not under test, this test enable pin is disabled and the circuit function does not change. The goal of control TPs is to increase the probability of exciting faults in a circuit and to make faults easier to observe by creating propagation paths to circuit outputs. Observe TPs change circuit observability by inserting fan-outs to circuit outputs, which makes faulty values on lines easily observed [16].

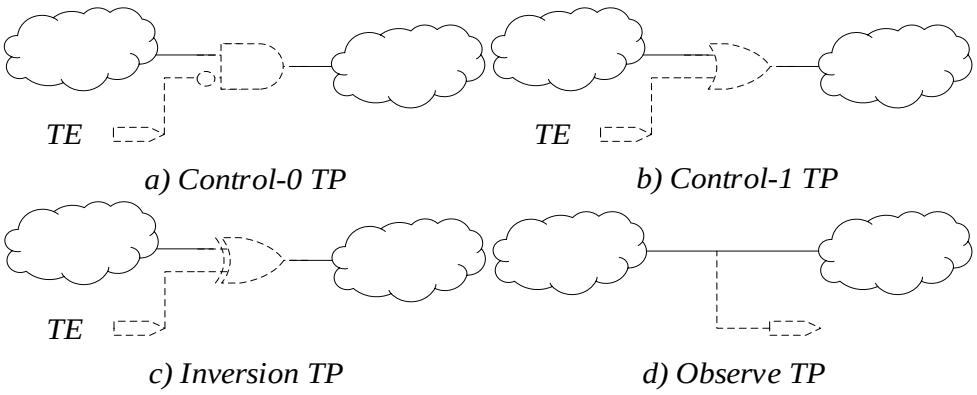


Figure 2.1: Logic-level implementations of control, inversion, and observe TPs.

The source of test enable and the output for observe points can either be a pin or a scannable latch [14]. Although test enable is most often modeled as a pin, implementing it as a circuit pin is

impractical given the high cost of circuit pins. Instead, additional TP "pins" are typically implemented as the outputs and inputs of scannable latches since a large circuit with many TPs and (latch-implemented) TP pins requires large area overhead. There are numerous articles on reducing TP pin/latch area overhead whilst using TP pins selectively to increase fault coverage, which is surveyed in Section 2.2.

Although effective at increasing SAF coverage, both control and observe TPs have detriments, hence TPI methods must carefully select TP locations and types. Since control TPs force lines to '0' or '1' when active, their controlled line can only be a single value when the TP is active: this prevents one SAF on the line from being excited. Additionally, active control TPs block the transmission of excited faults through the controlled line. Although observe TPs do not block faults like control TPs, osbserve TPs cannot detect RPR faults which are difficult to excite.

In contrast to control TPs, inversion TPs use inversions to change line values during test [17]–[21]. Inversion TPs are made with XOR gates and a test enable pin (shown in Figure 2.1(c)): when the test enable pin is active, the XOR gate becomes an inverter; otherwise the XOR gate acts as a buffer. In contrast to conventional control TPs which force lines to values, inversion TPs invert signal probabilities, i.e., if a line has an 80% probability of being logic-1, the line will have a 20% probability of being logic-1 with an active inversion TP. Because active inversion TPs do not force a single value, both stuck-at-0 and stuck-at-1 faults can be excited on active TP locations. Additionally, faults can propagate through inversion TPs to circuit outputs (unlike control TPs which block faults from propagating through) [21]. However, inversion TPs add more propagation delay, power, and overhead compared to control TPs [21], [22]. Additionally, RPR faults may require values to be forced to optimally increase fault coverage [21] that inversion TPs cannot perform.

2.2 TP Selection Architectures

Although TPs can significantly improve fault coverage, they can create significant area overhead, which in turn increases production costs and reduces yields due to larger die areas and fewer dies per wafer [23]. One study found chip area increased by 2.68% when using logic BIST, and TPs constituted 43% of this area increase [24]. It is therefore important to reduce the area overhead of TPs whilst keeping fault coverage high.

To reduce TP area overhead, some methods proposed sharing flip-flops or other existing circuit signals to reduce TP-controlling hardware [13], [17], [24]–[28]. Youssef et al. [13] and Nakao et al. [25] proposed sharing a single flip-flop for multiple test enable signals, which reduced the number of flip-flops that were required to implement control points. Yang et al. [24], [27] found more than half of TPs inserted were control points, so replacing dedicated test enable flip-flops with existing functional flip-flops reduced area overhead: suitable functional flip-flops could be found in each TP's fan-in region with a short distance from the control TP. Additionally, the test enable signals were only active in the test mode since the test enable signal was generated based on latch value combinations that could never occur in the functional mode, i.e., unused states of a finite state machine. Muradali et al. [26] proposed a self-drive TP that used test enable signals created from gate outputs already existing in the circuit, which eliminated the test enable signal generation. Similar to [26], [17] used pre-existing signals for test enable without the need for extra registers. Chang et al. [28] used controllability don't-cares to generate TP activation signals instead of a global test enable signal, which generated test enable signals locally and allowed TPs to be randomly activated: these controllability don't-cares were constant values in functional mode (i.e., circuit states which were accessible only through scan) and thus could only change values in test modes.

Other studies proposed reducing the number of TPs needed through various means. Basturkmen et al. [29], Tamarapalli et al. [30] partitioned circuit tests into multiple phases, and sub-sets of control TPs were activated during certain phases. This provided greater control over the interaction between control TPs and helped reduce the total number of TPs needed to obtain adequate fault coverage.

2.3 TPI Algorithms

TPI algorithms iteratively select TPs amongst a list of TP candidates: each iteration, the TP which increases the fault coverage the most whilst not violating other constraints (e.g., fault coverage, power, delay, etc.) is selected. Optimal TP placement in circuits with reconvergent fanouts is a known NP-hard problem [31], [32], thus most TPI approaches use heuristics to select TP locations, i.e., solve problem in a faster way by sacrificing optimality and accuracy.

Many TPI algorithms have been proposed in literature, and most algorithms performed the following steps to insert a single TP. First, fault simulation or approximate testability measured identified RPR faults. Second, candidate TPs were evaluated for their impact on fault coverage. Third, the TP with the highest positive impact on fault coverage was inserted into the circuit. This process was repeated until the number of desired TPs was inserted or the estimated fault coverage reached a pre-designated limit.

2.3.1 TPI Computational Difficulties

The challenge of TPI is placing the fewest number of TPs while maximizing fault coverage. Each TP requires logic circuitry, which in turn creates undesirable overheads: static and dynamic power, delay, and non-functional circuit area. Designers typically allocate budgets to TPs (and other DFT hardware), thus TPs must increase fault coverage to acceptable levels under these budgets.

Selecting optimal TP locations (and many other DFT problems) is a known NP-hard problem [32], thus existing TPI methods relied on heuristic approaches to select TP locations. Inserting T TPs into a circuit among T' candidate TPs creates $C(T', T)$ possible TP choices and finding the fault coverage impact of a choice requires computationally-intensive fault simulation. TPI heuristics address this by replacing fault simulation with less accurate fault coverage estimations [33], [34] and using greedy-algorithm approaches. The most common approach is iterative [33], [21] and [35]: the algorithm evaluates candidate TPs one at a time to find the one which one increases fault coverage the most, inserts it, and repeats this process until no more TPs are desired or needed.

Unfortunately, the computational complexity of iterative TPI grows faster than available computing resources. To insert a single TP, iterative TPI algorithms require evaluating every candidate TP, and the number of candidate TPs in a circuit is proportional to the size of the circuit (i.e., one or more candidate TPs can exist on every circuit line). Additionally, the complexity of TP-evaluating heuristics is linearly proportional to circuit size, and as circuit sizes increase, more TPs are needed to increase a circuit's fault coverage to acceptable levels, presumably at a rate linearly proportional to the circuit size (i.e., if the circuit size is doubled, meeting fault coverage goals requires twice as many TPs). Therefore, if circuit size grows by S, TPI complexity grows by $S \cdot T \cdot T'$. Presuming computer speeds increase with circuit complexity (i.e., algorithm performance increases at a rate of C), TPI time will grow at a rate of $S \cdot T \cdot T' / S = T \cdot T'$.

2.3.2 TPI Using Simulation

Using fault simulation to find undetected faults and then inserting TPs to detect these faults is a straightforward method of TPI. Iyngar et al. [36] inserted control TPs on gate outputs where faults were not excited while inserting observe TPs at the input of gates which blocked propagation.

Touba et al. [10] used backward path tracing to identify sensitized paths from undetected faults and selected a set of TPs for enabling the undetected faults to be detected. Ramakrishnan et al. [37], Menon et al. [38] used control TPs on undetected fault sites to sensitize a path to undetected faults in sequential circuit.

Several methods [29], [30] used probabilistic fault simulation to guide TP placement combined with greedy heuristics. Probabilistic fault simulation performed regular logic simulation to find signal probabilities and faults that were propagated in the circuit, and then used these probabilities to predict the probability any fault would be detected at a given location [30]. Tamarapalli et al. [30] used this method combined with a divide-and-conquer technique: probabilistic fault simulation was performed in phases, and at the end of each phase, TPs were inserted to target faults with the lowest detection probability. Basturkmen et al. [29] improved memory usage and TPI CPU time whilst marginally sacrificing TPI accuracy: instead of using logic simulation to determine all faults which could be detected on each circuit line (and the probability of each fault being detected), a representative of all faults at each fan-out location was chosen in order to reduce the number of faults to consider during TPI.

2.3.3 TPI Using Approximate Testability Measures

Fault simulation accurately quantifies fault coverage, but its computation complexity (in terms of CPU time and memory) is infeasible for modern circuits: to overcome this, numerous studies replace fault simulation with approximate testability measures, such as the *Sandia Controllability/Observability Analysis Program* (SCOAP) [39] and the *Controllability Observability Program* (COP) [40]. SCOAP is a linear complexity algorithm (relative to the number of logic gates in a circuit to analyze) which estimates the number of circuit inputs needed to force a logic-0/1 on a line, i.e., the *Controllability* (CC). Using these values, SCOAP can then

estimate the *Observability* (CO) of a line, which is the number of inputs that must be set to propagate a faulty value on a line to an observable output. SCOAP also includes the depth of a line in its controllability and observability estimations. Alternatively, COP predicts the probability a line will be logic-0/1 and the probability a line's value will be observed at a circuit output presuming random stimuli is applied to circuit inputs. COP values can directly be used to predict the probability of a fault being detected: $CC * CO$ for stuck-at-0 faults and $CC * (1 - CO)$ for stuck-at-1 faults, i.e., the probability a line is excited to the line's value is observed at a circuit output. Controllability and observability measures can therefore be used to identify hard-to-control and hard-to-observe locations in a circuit, and they can be used to predict the current fault coverage (with or without a TP) of a circuit without performing fault simulation. TPs can then be inserted based on this information.

Compared against exact fault simulation, testability measurements take substantially less time to calculate but lose accuracy for circuits with many reconvergent fanouts. However, experiments have suggested approximate testability measurements can be accurate enough for use in TPI for large designs [41]. Therefore, many TPI methods from literature [33], [41]–[45] used a cost function to estimate a TP's quality, with a typical example [42] provided below: F is a set of faults, and P_{d_j} is the probability the fault j is detected (calculated using COP).

$$U = \frac{1}{|F|}\sum_{\forall j \in F}\frac{1}{P_{d_j}}, P_{d_j} = \begin{cases} CC * CO, for\ stuck\text{-}at\text{-}1\ fault \\ CC * (1 - CO), for\ stuck\text{-}at\text{-}0\ fault \end{cases} \quad 2.1$$

Cost functions, such as U, are used as indicators of circuit testability, and many TPI algorithms attempt to maximize such cost functions during TPI. In this example, the value of U changes when a TP is inserted, and the difference in U before and after a TP is inserted is called the *actual cost reduction* (ACR) [42]. Gradient calculations [42] can select TPs with the largest ACR, but the computational complexity of finding the ACR for every TP is too high and

unpractical for modern circuits [41]. Therefore, the concept of a *cost reduction factor* (CRF) is introduced to approximate ACRs [45]. The algorithms which use CRFs and ACRs typically perform as follows: first, controllability and observability are calculated using an approximate testability measure, e.g. COP or SCOAP; second, the CRF/ACR for each TP in a set of candidate TPs is calculated, and TPs with a CRF/ACR below a given threshold are discarded; third, the ACR for remaining candidate TPs is calculated; lastly, the TP with the largest CRF/ACR is inserted.

When using a cost function to evaluate TPs, studies added nuances to select superior TPs or to reduce TPI CPU time. Bist et al. [43] selected TPs whose impact on timing slack and fault coverage were smaller and larger than a given threshold, respectively. Both Tsai et al. [41] and Bist et al. [33] proposed a hybrid cost reduction: after a TP was inserted, only faults with a large change in $1/P_{d_f}$ had their $1/P_{d_f}$ value recalculated using fault simulation (with other faults being calculated with testability analysis); the rational for this was the large changes in a CRF may be inaccurate. Nakao et al. [44] proposed three strategies for accelerating CRF-based algorithms: they removed TPs with redundant TPI-effective regions (i.e., regions where the same controllability (for control TPs) or observability (for observe TPs) were changed, chose the TP with the highest CRF (i.e., did not calculate an ACR), and reduced candidate TPs by selecting the first TP found to reduce the cost function (instead of calculating the ACR or CRF for all candidate TPs).

Beyond COP and SCOAP, other methods used additional/alternative cost functions or introduce additional constraints. Youssef et al. [13] and Gerstendörfer et al. [46] identified RPR faults using COP and created fault sectors: RPR faults were sorted by ascending logic levels, then control TPs targeted faults in ascending order and observation TPs targeted faults in descending order; this prevented the same fault being targeted by the same TP, which reduced the number of TPs required. Gerstendörfer et al. [46] used *test counts* (TCs) to complement COP-based TPI: the

TC of a line was the fewest number of tests that must pass through the line such that all faults in its fan-in cone would be tested, and TPs were selected in order of the most tests that must pass through the TP location. Geuzebroek et al. [8] proposed several cost functions using one or multiple test analysis measurements (COP, SCOAP, or TC): TPI was split into multiple stages, where each stage selected a cost function to target the current hardest test problem, i.e., detecting RPR faults, reducing test vectors, or a combination of the two. He et al. [47] [48] used an efficiency equation for TPs that evaluated the size of a TP fan-out/fan-in cone-of-influence and the number of undetected faults in this cone, which in turn was used to pick the TP with the highest efficiency. An estimation metric was then used to approximate the final area overhead and test coverage without TP insertion and synthesis. Chen et al. [49] proposed a new conditional testability measure to overcome COP's inability to account for reconverging fan-outs, thus increasing the accuracy of calculated cost functions.

Many methods incorporated non-fault coverage information (e.g., timing violations) into their cost functions [43] or efficiency equations [48], which are further discussed in the following sections.

2.3.4 TPI Using Multiple Measures

Some approaches utilized both fault simulation and testability measures to increase TP quality [50]–[52]. Sethuram et al. [50] reduced test vector counts and test generation time by considering layout and timing information for observe TPs. The cost function of an observe TP was the product of the total number of independent faults (i.e., faults which cannot be simultaneously detected by any single pattern) in the fan-in cone of the observe TP (which was found through fault simulation) and the minimum number of controlled primary inputs needed to propagate the independent faults to the TP location (estimated using SCOAP). Acero et al. [51]

and Moghaddam et al. [52] performed COP and fault simulation to calculate fault testability, propagated faults, and faults blocked by control TPs: a control TP cost function was composed of the controllability of blocked faults, and the cost function of observation TPs composed of observability of unobserved faults.

2.4 Modern Targets for TPI

Modern TPI not only targets increasing SAF coverage, but also considers extra constraints imposed by modern technologies, e.g., increasing delay fault coverage, reducing test power, reducing timing impacts. These constraints make TP selection much more difficult than before, but addressing them improves circuit performance [14].

2.4.1 Path Manipulation to Increase Path Delay Fault Coverage

A *path delay fault* (PDF) [53] occurs when any path's delay exceeds a circuit's designed clock speed, and PDFs model defects that cause cumulative propagation delays along a circuit path that exceed the circuit's specifications. Unlike SAFs, PDFs are defined by their environment: a PDF exists only within a certain range of operational clock speeds. PDFs are tested using a set-up vector to create the preconditions for a transition, and a second trigger vector to initiate the transition. Using specialized test hardware, a clock period greater than the operational clock period can be used to create the set-up vector and to apply the trigger vector, but the operational clock period must be applied after the trigger vector to properly capture a slow transition along a path. If the target output has not changed from its value after the set-up vector, then the circuit is faulty.

There are three problems associated with PDFs, which are problems TPs have attempted to address. First, the number of paths (and number of PDFs) in practically-sized circuits is too large for test tools to handle [54]. Second, the number of tests needed to detect all PDFs is too large [55]. Third, many PDFs in practical circuits are not testable [56]. To remedy this, TPs can

divide full paths into sub-paths, thus making paths easier to test and reducing the number of paths [57]. Additionally, it is easier to generate tests for shorter sub-paths compared to full-sized paths [57].

TPI methods have incorporated these observations into cost functions that represent the number of paths in a circuit (i.e., the TP that reduces the total number of paths in the circuit is iteratively chosen) [57]–[59]. Pomeranz et al. [58] used test point to divide the set of paths into a subset paths and reduce the number of paths to be tested.directly. Tragoudas et al. [57] added an additional constraint to the above cost function, i.e., the clock speed of the circuit under test: if a TP reduced the longest path in the circuit, the clock speed during test could be increased, thus decreasing test application time. Uppduri et al. [59] targeted non-robust-dependent faults (faults in functionally sensitize paths) [60], which reduced the fault set, thus reducing the number of TPs.

2.4.2 Test Power Reduction

The power consumption of digital systems is considerably higher in a test mode compared to functional modes. This is because during normal circuit operation, a relatively small number of flip-flops change their value each clock cycle, whilst in a test mode, a much larger number of flip-flops will change values, which results in excessive switching activity and current spikes [61]. Especially during self-test, power dissipation increases since random patterns desire as many nodes switching as possible so as to test for many faults [62]. If the peak power during test is too large, there will be a V_{dd} drop or ground bounce that can cause false-failures or device damage.

Some studies [61], [62] inserted TPs to reduce power consumption during test, but TP placement was restricted to flip-flop outputs. Gerstendörfer et al. [62] used modified shift registers that suppress activity at the output during shift operations: by adding NOR or NAND gates to the outputs of latches controlled by a test enable pin, latch outputs were forced to known values and

thus did not cause circuit switching. Sankaralingam et al. [61] proposed inserting TPs into a conventional full-scan circuit to keep peak power during scan below a given limit without decreasing fault coverage (with TPs being inactive during the capture cycles): a subset of scannable flip-flop outputs were forced to 0 and 1 during scan. First, cycle-by-cycle simulation identified which scan cycle's power consumption was greater than the specified limit. Second, an event-driven, selective trace simulation procedure [63] estimated the power reduction for every latch when its output was forced to 0 or 1, then latches were iteratively forced to reduce power consumption.

2.4.3 Timing Impacts of TPs

Timing is fundamental to modern digital electronics, as it provides synchronization that is necessary for error-free data transfer. Data must be stable before and after the clock edge to be reliably transferred. If not properly synchronized, there will be a host of design issues like, including timing hazards, metastability, and race conditions.

Inserted TPs may cause circuit timing violations that break proper circuit operation [23], and resolving these timing violations may require several tedious design iterations. Many attempts have been proposed [43], [64]–[66] to insert TPs to increase fault coverage without creating new timing violations. In Bist et al. [43] and Vranken et al. [66], timing analysis was performed before TPI to identify paths with small timing slacks, then TPI was performed after removing candidate TPs that reside on such paths. Tofte et al. [64] performed TPI without any constraints, then timing analysis was performed to remove TPs which caused timing violations. Roy et al. [65] performed TPI at the RTL-level (instead of the typical logical netlist level), which meant TPs were inserted before logic synthesis, which avoided later design iterations.

Chapter 3

Introduction to ANNs

Inspired by biological neural networks, ANNs are computing systems consisting of an extremely large number of simple processes (hardware processors, software functions, etc.) with many interconnections. ANNs are one type of model for *machine learning* (ML) and attempt to use the same principles of human thinking. Nodes in ANNs are artificial neurons that are computational models inspired by natural neurons [67]. Since the great potential of ANNs is high-speed processing provided in a massive parallel implementation, they are used in many fields [68]. ANNS are widely used in engineering: they are used as models of biological nervous system and "intelligence", as real-time adaptive signal processors, as controllers implemented in hardware in robots, and as data analytic tools [69].

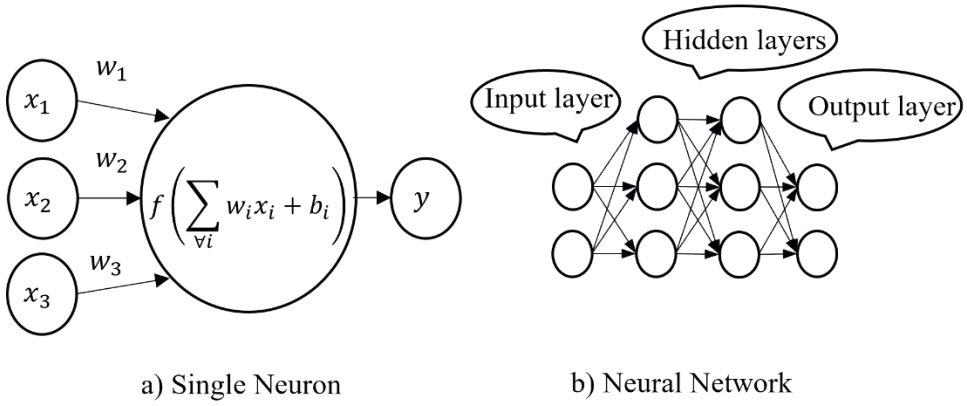
3.1 ANNs history

ANNs were a hot topic in artificial intelligence starting in the 1940s. Warren McCulloch and Walter Pitts opened the subject by creating a computational model for neural networks in the early 1940s [70]. Hebb was the first to define a learning rule to explain the behavior of networks of neurons in 1949 [71]. In the late 1950s, Rosenblatt developed the first perceptron learning algorithm [72]. In 1973, Dreyfus used backpropagation to adapt ANN parameters proportion to error gradients [73]. In 1992, max-polling was introduced to help improve ANN quality [74].

3.2 ANNs structures

Many ANN structures exist in literature, and Figure 3.1 illustrates the prototypical ANN structure. ANN architectures comprise an input layer, hidden layer(s), and an output layer. The input layer receives the input values and the output layer stores output values. Layers between the input and output layer are hidden layers, to which there can be a single or multiple layters. Each

layer can have one or more neurons. Connections between neurons have weights and biases. Neurons have activation functions that determine a neuron's input given its inputs. Many choices exist for neuron activation functions and neuron arrangements (the number of levels, neurons per level, etc.), and these hyperparameters are best optimized through trial-and-error.



a) Single Neuron b) Neural Network

Figure 3.1: An example of a) a single neuron, and b) the ANN prototypical structure

3.2.1 Neurons

ANNs are composed of artificial neurons similar to biological neurons. The input neurons collect feature values of the problem to solve, such as the pixels in images or letters in documents, or their inputs can be from the outputs of other neurons. The outputs of the final neurons return an answer to the problem, such as recognizing an object in an image.

3.2.2 Connections

ANNs consist of connections, with each connection providing the output of one neuron as an input to another neuron. Each connection has a weight (noted as w_i in Figure 3.1) and bias (noted as b_i in Figure 3.1) assigned to it, and these weights represent its relative importance [75]. A given neuron can have multiple inputs and single output connection.

3.2.3 Activation Functions

Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network and determines its output value. Activation functions also normalize the output of each neuron to a range between 1 and 0 or between -1 and 1. Activation functions can be divided into linear and nonlinear (e.g., Sigmoid, Tanh, Rectified Linear Unit (ReLU), or Softmax) [76].

3.3 Training ANNs

Once a network has been structured for a particular application, that network is ready to be trained. To start this process the initial weights are chosen randomly. Then, the training (or learning) begins. There are two approaches to training – supervised and unsupervised. In supervised training, training provides a set of both sample problem inputs and desired problem outputs. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights that control the network. In unsupervised training, the network is provided with inputs but not with desired outputs, and the training algorithm attempts to find trends in the data to apply to future possible inputs.

3.4 ANN applications

ANNs have been widely applied to real-world problems in business, engineering, language, and practical consumer products. In business, ANNs were used for credit scoring [77], financial analysis [78], stock performance prediction [79], etc. In engineering, ANNs were implied in aircraft component fault detectors [80], automotive guidance systems [81], robotics control system [82], etc. ANNs were utilized in language, e.g., in email classification and categorization [83] , named entity recognition [84], machine translation [85], etc.

ANNs also have many applications in the electronics field, including chip failure analysis [86], circuit chip layouts [87], prediction of electronic structure properties [88], etc. ANNs have also been used in DFT testing problems, e.g., scan-chain diagnostic [11], fault classifiers [12], and test pattern generation [89], [90].

Chapter 4

ANNs TPI Targeting Stuck-at Fault Coverage

This chapter introduces a method of applying ANNs to TPI for increasing SAF coverage while drastically decrease CPU runtime. Increasing TPI quality is essential for the modern logic circuit; the computational requirements of current TPI heuristics scale unfavorably against increasing circuit complexity and it is time consuming process. ANNs were applied to several EDA problems with noteworthy success [35], [11]. This gives the motivation to explore ANN to increase TPI algorithm efficiency and overcome the drawbacks of TPI heuristics.

Much of this chapter has been published by the author in [91].

4.1 Introduction to the Stuck-at Fault Model

A SAF is a particular fault model used by fault simulators to mimic a manufacturing defect within an integrated circuit. When a line is stuck it is called a fault: individual line is assumed to be stuck at logical ‘1’ or ‘0’, called stuck-at-1 fault and stuck-at-0 fault, respectively. A single stuck line is a fault model used in manufacturing testing; the model assumes that only one input or output on one gate will be stuck at logic-1 or logic-0 at a time. SAF coverage represents the percentage of all possible SAFs that a given test will detect. To test a SAF, a test vector applied to the circuit’s inputs must have at least one output pin different from fault-free outputs [92].

4.2 Proposed Method

The proposed method created an ANN that evaluated TPs using circuit probability information, i.e., COP controllability and observability values [40], which many other TPI methods ([33], [21] and [35]) used. Evaluating a single TP did not require re-calculating COP values: this was a noteworthy advantage of the ANN over TP-evaluating heuristics that required re-calculating values when evaluating a TP: calculating COP values for a circuit with G gates

requires $O(G)$ time. By performing COP once per TP insertion as opposed to once per TP evaluation, this reduced TP evaluation time from $O(G)$ to $O(1)$, which reduced the time to select a single TP from $O(T' \cdot G)$ to $O(T')$. However, for ANN TPI, long training time might negate this benefit, thus the overall effect on TPI time (with and without training) was explored in Section 4.8.1.

The proposed ANN evaluated TPs and could be used in any iterative TPI algorithm [33], [21] and [93]. Every iteration, the algorithm evaluated each individual TP to find the "best" TP (i.e., the TP which increased SAF fault coverage the most), and the algorithm inserted this TP into the circuit. This iterative selection and TP insertion continued until (1) the number of TPs inserted reached a pre-designated limit (representing hardware overhead), (2) the predicted fault coverage reached a pre-designated limit (i.e., no more TPs are necessary), (3) no TPs were predicted to increase fault coverage, or 4) a CPU time limit was reached.

4.3 ANN Input

The input features to the ANN TPI were circuit CC and CO values. These were the same values used by other conventional TPI algorithms [40], but the ANN used only pre-TP activation values. Using these values in such a manner presented several potential benefits. First, using only pre-TP activation values forgone the need to re-calculate CC and CO values for every TP and decreased TP evaluating time. This benefit could also be interpreted as allowing for more (or alternate) TPs to be inserted with identical computational effort, which in turn allowed for increased TP selection quality. Second, an ANN could find relations between values during training which heuristic algorithms may not take advantage of. For instance, the known issue of CC and CO values not considering fan-outs was ignored by many TPI algorithms, but a trained ANN could find and acted on such nuances automatically.

In contrast to conventional TP-evaluating heuristics, the ANN performed its evaluation on a transformed sub-circuit centered around a candidate TP location as opposed to estimating a TP's impact on the entire circuit. This was a consequence of using ANNs: the input size of an ANN must be of a pre-determined size, whereas logic circuits could be of any size and topology. In theory, the ANN could handle the largest possible circuit size or smaller by giving "unused" ANN inputs "default" values, but this required an infeasibly large ANN that is impossible to train. Instead, the ANN analyzed features around a TP's location by analyzing a sub-circuit *L* levels forwards and backwards of the indicated location, as Figure 4.1 illustrates.

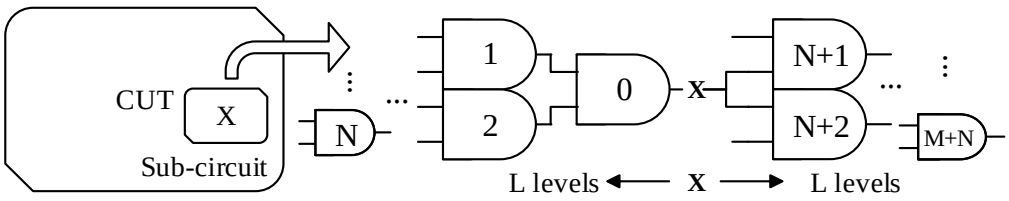


Figure 4.1: A sub-circuit size is represented by *L* levels. "X" marks the TP location.

The use of a sub-circuit presented a potential detriment of the ANN to explore: the ANN used less information for its TP-evaluating calculations and may return less accurate qualifications. The ANN would be as accurate as a heuristic or more using identical input features only if *L* is large enough to capture the entire circuit, but this created an infeasibly large ANN that was impossible to train. Section 6.3.2 explores whether analyzing sub-circuits decreases the ANN's performance compared to conventional TPI.

A nuance of the sub-circuits was they must have a particular circuit configuration (hence "transformed" in transformed sub-circuits): every gate must have at most two inputs and two fan-outs. So circuits had to be converted into a consistent fan-in/out structure before the ANN was used and trained. Any given circuit could be converted to this structure by replacing gates with more than two inputs with functionally equivalent copies and by adding buffers to fan-outs, as is

illustrated in Figure 4.2. After conversion, the circuit's logic function was not changed, and CC and CO values for lines and gates in the original circuit were same.

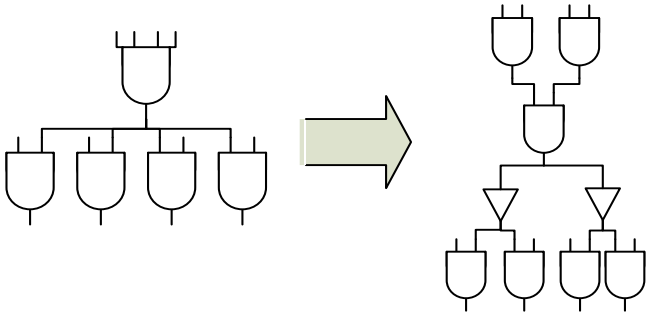
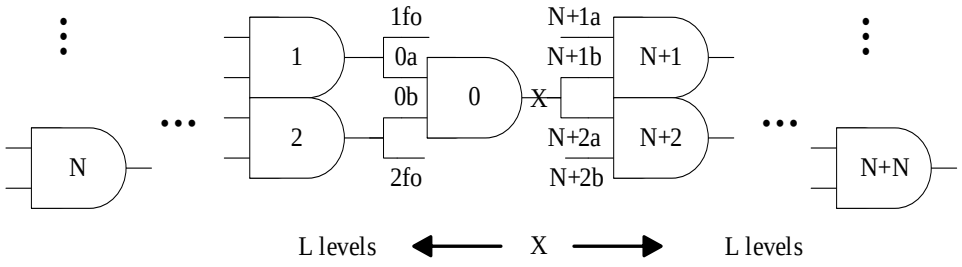


Figure 4.2: A conversion of a sub-circuit to the format of no more than two fan-in/fan-out per gate.

The input vector to the ANN was CC and CO values around a candidate TP location in a vectorized format, as is illustrated in Figure 4.3. The first value was the CC value of the candidate TP location. This was followed by the CC value of gate inputs (and fan-outs) feeding the TP location, then the CC value of gate inputs (and fan-outs) feeding these gates, etc., in a breadth-first order. This was repeated until L levels values were collected. This process was repeated starting at the candidate TP location moving forward in the circuit until L levels values were collected. When moving forward, the values of fan-ins to gates were also captured. CC values were followed with CO values in the same order. For gates with one input or one fan-out, "default values" replaced the values in the feature string: non-existent lines had 50% controllability and 0% observability, and non-existent gates, which were normally represented using a one-hot encoding (e.g., 0001 = AND, 0010 = OR, etc.), were replaced with a no-hot encoding (e.g., 0000).



NN Input Vector

$CC_0, CC_{1fo}, CC_{0a}, CC_{0b}, CC_{2fo}, CC_1, CC_2, \ldots, CC_{Nb}, CC_{N+1a}, CC_{N+1b}, \ldots, CC_{N+N-1}, CC_{N+N},$
$CO_0, CO_{1fo}, CO_{0a}, CO_{0b}, CO_{2fo}, CO_1, CO_2, \ldots, CO_{Nb}, CO_{N+1a}, CO_{N+1b}, \ldots, CO_{N+N-1}, CO_{N+N},$
$Gate\_0, Gate\_1, Gate\_2, \ldots, Gate\_N, Gate\_N+1, Gate\_N+2, \ldots, Gate\_N+N$

Figure 4.3: The ANN input features are CC, CO, Gate type.

4.4 Output Label

The output of the ANN was the impact on SAF coverage, a TP had on a circuit when it was inserted and activated: $\Delta FC_t = FC - FC_t$. In this formula, FC was fault coverage of sub-circuit with no TPs, and FC_t was fault coverage of sub-circuit with TP t active. Training values were found through fault simulation of a circuit before and after a TP had been inserted. This was a relatively computationally intensive process to perform, but every time it was performed more training data would be generated and the accuracy of the ANN would increase. To avoid the ANN being too specialized to one type of circuit, fault-simulated TP impacts were calculated for a diverse set of circuits.

4.5 Training Data Generation

The data generation process included generating input features and corresponding labels required to train the ANN. First, arbitrary locations were chosen in training circuits and sub-circuits were extracted from these locations. Second, CC and CO values (which require a single-pass calculation per training circuit to generate) and gate types (AND, NOR, etc.) in the sub-circuit were recorded.

The output label was generated by fault simulation, which was computationally demanding: fault simulating V vectors in a circuit with G gates and F faults requires $O(V \cdot G \cdot F)$ time, therefore collecting S training samples required $O(S \cdot V \cdot G \cdot F)$ time. Reducing V to a small number of vectors was not an option, since an LBIST test typically applied many vectors. To generate training output faster, two speed up techniques were used to reduce time.

The first training speedup technique was to apply fault simulation to sub-circuits in lieu of larger circuits with assistance from circuit probability information. Performing fault simulation on sub-circuits significantly reduced G and F (since the number of faults in a sub-circuit was proportional to the number of gates), which in turn reduced fault simulation time. However, truly random stimulus of sub-circuit inputs and direct observation of sub-circuit outputs was not realistic: under random circuit stimulus, sub-circuit inputs were not truly random, nor are sub-circuit outputs always observed. To account for this, the training data generation program calculated COP controllability and observability values once per training circuit. This additional $O(C)$ calculation time was negligible when taking a significant number of sub-circuit samples from a given circuit. Fault simulation weighted each sub-circuit input vector using these COP controllability values. Additionally, if a fault's effect reached a sub-circuit output, fault simulation probabilistically detected it using the COP observability values of sub-circuit outputs.

A second technique reduced training data generation time by eliminating redundant vectors. Simulating all $2^{|I_S|}$ input patterns (where $|I_S|$ was the number of inputs of the sub-circuit) for a sub-circuit took significantly less time compared to simulating 10K (or other common pseudo-random test lengths) random patterns for fault simulation for practical values of $|I_S|$, but even under many random vectors, all $2^{|I_S|}$ input patterns may not be applied. This was because the CC, CO values of inputs can prevent all patterns being applied, even when the number of patterns

applied was significantly greater than $2^{|I_S|}$. The probability of generating an input pattern based on CC and CO could be calculated as $p = \prod_{\forall i \in I} CC_i'$, where CC_i' is $1 - CC_i$ if a 0-value was needed or CC_i was a 1-value is needed. The probability of generating a pattern in V vectors was $P_i = 1 - (1 - p)^V$. Likewise, if a fault was observed on an output "o", the probability of the fault being observed on the pin in V vectors was $P_o = 1 - (1 - CO_o)^V$. By calculating these probabilities, every $2^{|I_S|}$ input vector was simulated at most once with probability P_i, and if the vector was applied, simulated faults would be considered detected with probability P_o.

4.6 Training

To make the ANN training process simpler, three separate, smaller ANNs were trained representing the three TP types: control-0, control-1, and observe TPs. The alternative to this was to have a single ANN with an extra "TP type" input, but this made the complexity of the ANN unnecessarily large. When evaluating a TP's quality, the appropriate trained ANN was used.

After training data was created, ANN training was performed under various ANN hyperparameters to minimize ANN error (in this study, mean squared error). Finding a truly optimal ANN for a given training data set is an NP-hard problem [94], and small changes in initial training conditions and ANN parameters can have significant impacts on the resulting ANN quality, thus a trial-and-error process was used to minimize this ANN error.

4.7 ANN TPI Flow

The use of a trained ANN in a TPI algorithm is analogous to the use of a TP-evaluating heuristic in an iterative TPI algorithm [33]. A trained ANN calculates a candidate TP's quality, and after doing so for every candidate TP, the highest quality TP is inserted. The entire TPI flow is shown in Figure 4.4.

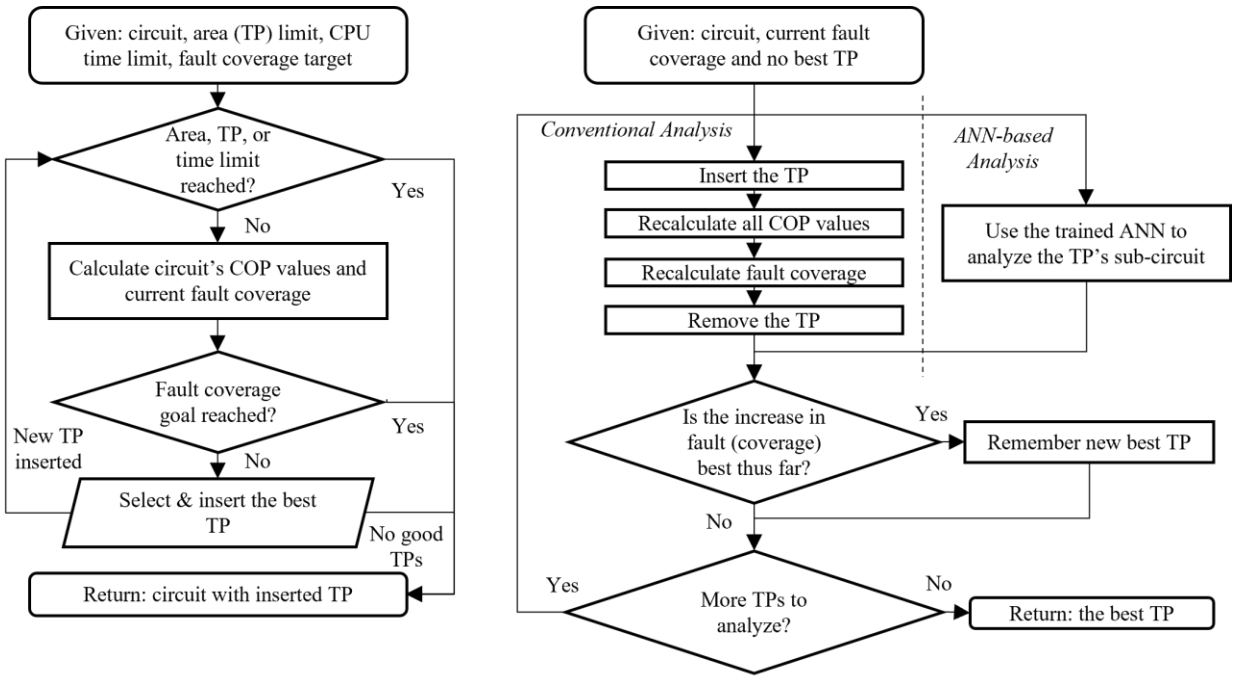


Figure 4.4: ANN TPI Flow.

4.8 Experiment Results

4.8.1 Experiment Setup

Industry-representative workstations performed fault simulation and TPI using original software. These workstations used Intel i7-8700 processors and possess 8 GBs of RAM, and all software was written in C++ and compiled using the MSVC++14.15 compiler with maximum optimization parameters. Original software was used in lieu of industry tools to obtain a fair comparison of the proposed ANNs against methods from literature: only code which analyzed the "quality" of a TP was changed between the methods thereby minimizing other sources of CPU time differences.

The ANN was trained by Python and TensorFlow. TensorFlow is an open-source symbolic math library developed by Google Brain [95], and it can be used across a range of tasks but has a

particular focus on training and evaluation deep neural networks. TensorFlow provides the TensorBoard function that stores statistical data and plots scalar figures for the training process.

The conventional TPI method used for comparison was from [33]. It [33] is an iterative TPI method that is representative of TPI implemented in industry, although industrial tools have many additional computation-time optimizations. Conveniently, the studied ANNs can directly replace the "quality" measuring subroutine in [33], which eliminated other sources of fault coverage and CPU time differences.

The benchmarks used in this study, both for ANN training and TPI evaluating, are given in Table 4.1, which were from the ISCAS'85 [96] and ITC'99 [97] benchmarks, which represent a wide range of industry-representative circuits. Table 4.1 provides the number of nets in each benchmark ("Num. nets") and the number of nets in the circuit after it was expanded ("Num. exp. nets") using the method resented in Section 4.3. If the benchmark was used for ANN training, the number of training TPs and sub-circuits extracted is given. From each sample/sub-circuit, the impact of a control-0, control-1, and observe TP was fault simulated using the method presented in Section 4.4. The total number of training samples/sub-circuits collected was 11,561.

Table 4.1: Training and Evaluation Benchmarks for TPI

	Benchmarks	Num. nets	Number. Exp. nets	Num. Samples
Training	c17	11	11	2
	c499	243	425	6
	c1355	587	881	13
	c1238	543	1016	14
	c2670	1426	1969	25
	c6288	2448	3376	52
	c7552	3719	5482	72
	b02	27	46	2
	b04	729	1221	16
	b06	50	93	2
	b08	179	304	4
	b10	200	342	5
	b12	1070	1900	26
	b14	10044	17719	242
	b15	8852	16920	232
	b17	32229	59818	817
	b18	114598	201844	2758
	b19	231290	405924	5549
	b20	20204	35993	491
	b21	20549	36916	503
	b22	29929	53564	730
Evaluation	c432	196	310	
	c880	443	682	
	c1196	561	978	
	c1908	913	1315	
	c3540	1719	2693	
	c5315	2485	4396	
	b01	47	80	
	b03	156	268	
	b05	962	1783	
	b07	433	706	
	b09	169	280	
	b11	764	1271	
	b13	352	543	

4.8.1 ANN Training Experiments

The first issue explored through ANN training was the impact of different ANN structures. As discussed in Section 3.2, many different parameters influence the quality of an ANN, including

ANN size (the number of neurons and the number of layers), ANN connectivity, training time, and neuron activation functions. These parameters were thoroughly explored through several parameter sweeps to find the configuration which minimizes the average error of the ANN. Figure 4.5 shows three trends of the control-1 TP, control-0 TP, and observe TP evaluation ANN. The figure shows the ANN accuracy as more training time was allocated, which shows that training effort was "saturated" after some point. All ANNs in this study were trained beyond this saturation point.

The second issue explored was the neuron's number impact on ANN accuracy. Figure 4.6 shows the accuracy of different ANNs for varying numbers of neurons, which shows more neurons may not produce a higher quality ANN. ANNs with different numbers of neurons (8,16,32,64,128) were used to train control-1 TP, control-0 TP and observe TP in the same literation, and the training error showed the ANN accuracy. The final parameters chosen were two hidden layers with 128 neurons in the first layer and a single neuron in the second layer. A ReLU activation function and the Adam Optimization method [98] were used for training the ANN.

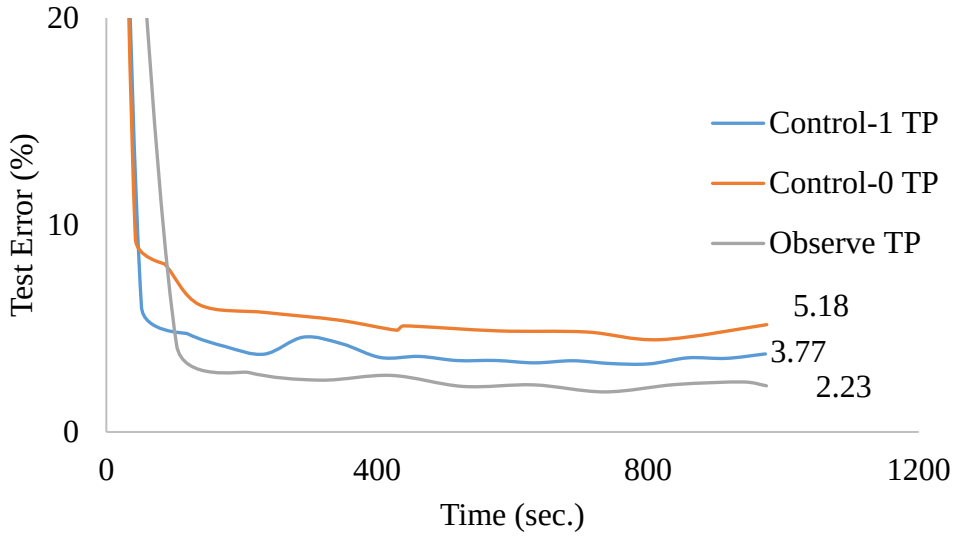


Figure 4.5: A plot of training time's impact on ANN accuracy.

The third issue explored was how the size of training data influences the quality of the ANN. Figure 4.7, which plots the control-1 TP ANN error with respect to training data size, shows that small training data size created an inferior ANN, but large data sizes increased ANN training time with minimal returns on ANN quality. Different sizes of training data were collected and used to train the ANNs: 0.1%, 0.5%, 0.9%, and 1.5% of all TP locations in all training circuits. From this, the ANN with the lowest error was chosen: the total training samples/sub-circuits was 11,561, the data collecting time was 33.6 minutes, and the ANN training time was 16.3 minutes. This 50-minute overhead was considered with evaluating the computational efficiency of ANN TPI.

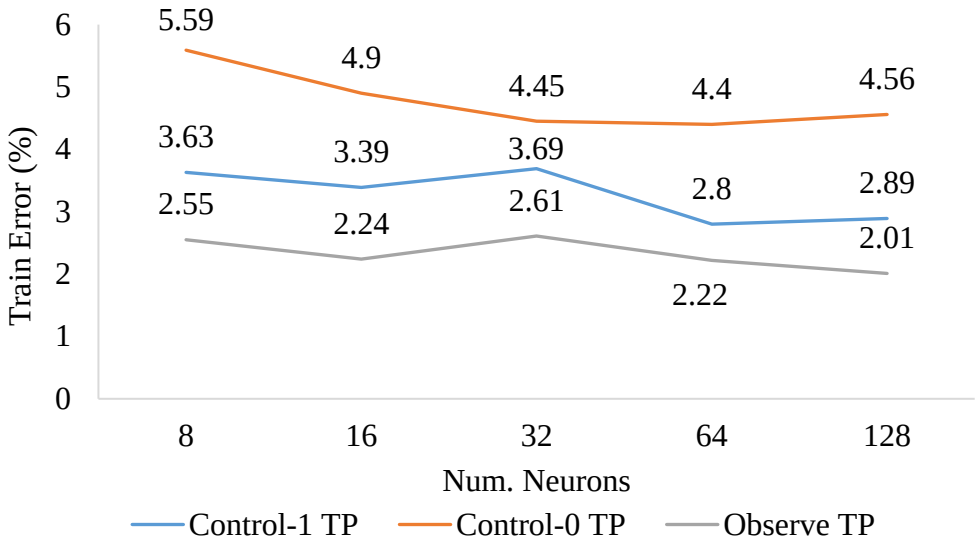


Figure 4.6: A plot on the number of neuron's on ANN accuracy.

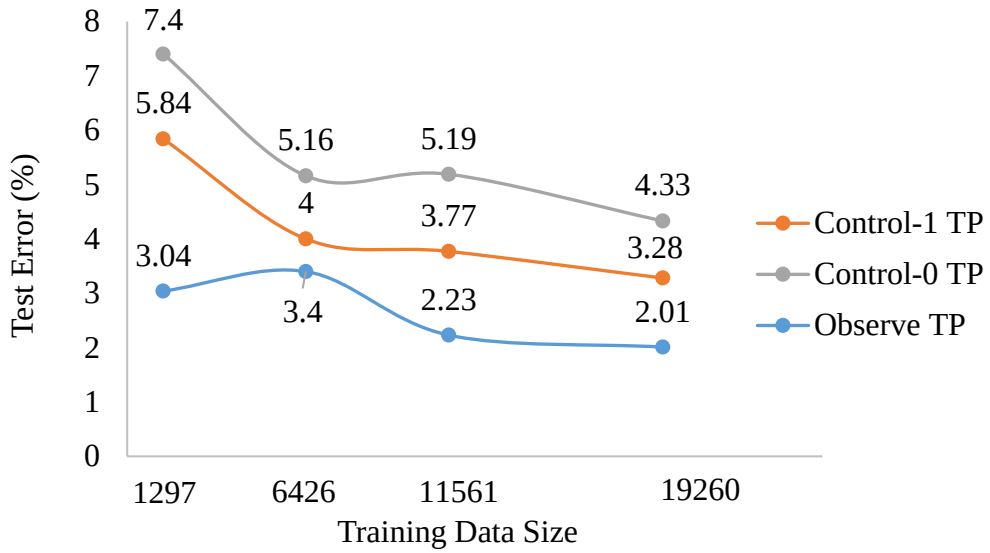


Figure 4.7: A plot of training data size on ANN accuracy.

4.8.2 TPI Experiment Results

The results of fault simulation on benchmark circuits are shown in Table 4.2. First, this table shows the final fault coverage for three variations of benchmark circuits: with no TPs, after ANN TPI was performed, and after conventional TPI [33] was performed, with the conventional TPI using the same information used for ANN training (i.e., neither the ANN nor conventional TPI have an information advantage). These three circuit variations were fault simulated with 100, 1,000, and 10,000 vectors to confirm the effect of TPs was consistent under different stimulus conditions. For both TPI methods, the number of TPs inserted is listed under "num. TPs", which was limited to 1% of the total number of nets or predicted fault coverage limits or negative TP impact limits were never reached for any benchmark. Second, the table lists the time required to perform TPI for both ANN TPI and conventional TPI [33]. To account for random stimulus variation, the average fault coverage was recorded amongst multiple trials: 100 trials for 100 vectors, 50 trials for 1,000 vectors, and 10 trials for 10,000 vectors.

Table 4.2 shows the fault coverage created from TPs inserted by the ANN TPI algorithm was decisively superior compared to conventional TPI. This is better illustrated in Figure 4.8, which gives the relative fault coverage obtained by the ANN TPI compared to conventional TPI. These results revealed several findings. First, the fault coverage of ANN TPI was frequently higher, with the only benchmark providing lower fault coverage being *b13*. Second, it was noteworthy that extreme differences in fault coverage favor ANN TPI, as is demonstrated by the benchmark *b05*.

Table 4.2: ANN TPI and Conventional TPI Targeting SAF Experimental Results

Benchmarks			c432	c880	c1196	c1908	c3540	c5315	b01	b03	b05	b07	b09	b11	b13
Num. TPs			1	3	5	8	16	23	0	1	9	3	1	7	2
Num. nets			196	443	561	913	1719	2485	47	156	962	433	169	764	352
Num. exp. nets			310	682	978	1315	2693	4396	80	268	1783	706	280	1271	543
Processor memory (MB)			4	11	21.7	154.5	170	227.1		3.6	42.6	9.2	9.9	32.5	5.9
	Num. Vectors														
SAF coverage (%)	No TPs	100	92.37	90.31	66.99	75.86	80.50	94.30	100	98.12	66.63	88.71	79.86	80.24	89.94
		1K	98.58	97.26	87.14	94.60	94.47	99.18	100	99.98	75.50	92.76	84.60	92.06	94.52
		10K	98.84	99.71	92.97	99.73	96.38	99.41	100	100	80.04	97.55	98.56	95.38	95.78
	ANN TPI	100	98.82	99.86	94.23	99.69	96.40	99.44		99.99	83.00	97.52	99.57	95.91	95.75
		1K	98.83	99.84	94.65	99.83	96.38	99.44		100	82.95	97.68	99.02	95.66	95.75
		10K	98.84	100	94.35	99.91	96.47	99.44		100	83.03	97.75	99.91	96.07	95.79
	Conventional TPI	100	98.82	99.85	92.00	99.19	96.45	99.06		99.99	63.62	93.28	99.53	94.20	98.48
		1K	98.83	99.82	91.92	99.32	96.43	99.06		100	63.50	93.30	98.96	94.18	98.47
		10K	98.84	100	92.06	99.41	96.47	99.06		100	63.66	93.32	99.85	94.21	98.52
Execution time (min.)	ANN TPI	100	1.14	7.32	30.94	128.72	967.51	1568.82		0.38	131.62	10.79	0.58	72.90	2.00
		1K	1.09	7.17	30.43	124.62	894.43	1488.73		0.11	122.51	9.97	0.55	68.59	1.80
		10K	1.06	8.31	31.65	123.63	930.01	1508.25		0.31	122.19	10.08	0.58	69.67	1.80
	ANN TPI including training time	100	2.08	9.46	34.19	133.12	976.59	1583.21		1.18	137.62	13.04	1.44	77.13	3.65
		1K	2.03	9.31	33.68	129.02	903.51	1503.12		0.91	128.51	12.22	1.41	72.82	3.45
		10K	2.00	10.45	34.90	128.03	939.09	1522.64		1.11	128.19	12.33	1.44	73.90	3.45
	Conventional TPI	100	2.27	23.14	75.50	323.87	7321.71	7199.80		1.25	4004.53	27.12	1.62	204.02	10.06
		1K	2.36	23.31	78.39	323.37	5302.20	7015.34		1.31	482.72	25.90	1.56	195.05	10.02
		10K	2.17	23.77	76.44	329.86	7314.74	7003.95		1.20	3578.09	26.53	1.56	195.57	9.80

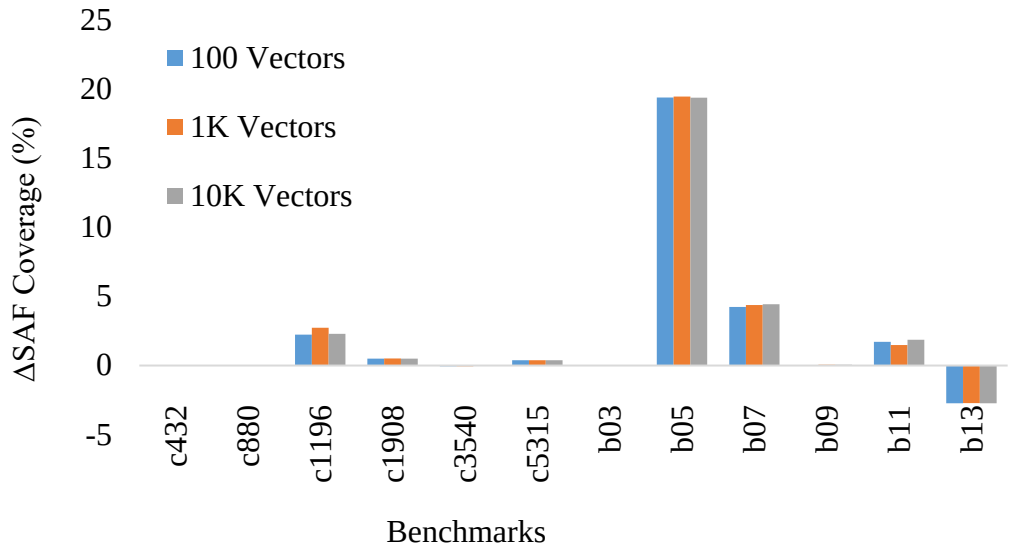


Figure 4.8: ANN and conventional TPI targeting SAF in SAF coverage comparison.

The second observation from Table 4.2 is the time to perform TPI was significantly less for ANN TPI compared to conventional TPI, including when ANN training and data collecting time was proportionally distributed amongst circuits according to circuit size. This is better illustrated in Figure 4.9, which gives the execution time of ANN TPI compared to conventional TPI. This figure clearly shows the ANN TPI performed faster than conventional TPI across all benchmark circuits, with on average ANN TPI being done in one-sixth of the time. This result was understandable given the ANN TPI did not need to re-calculate CC and CO values for every candidate TP, and instead it evaluated the ANN for its input vector, which was a relatively fast process for a small sub-circuit. When training time (approximately 50 minutes) was distributed proportionally amongst all benchmarks according to size (e.g., smaller circuits add a smaller fraction of the 50 minutes and vice versa), the impact on TPI time became negligible for larger circuits and the total TPI time was still decreased across all benchmarks.

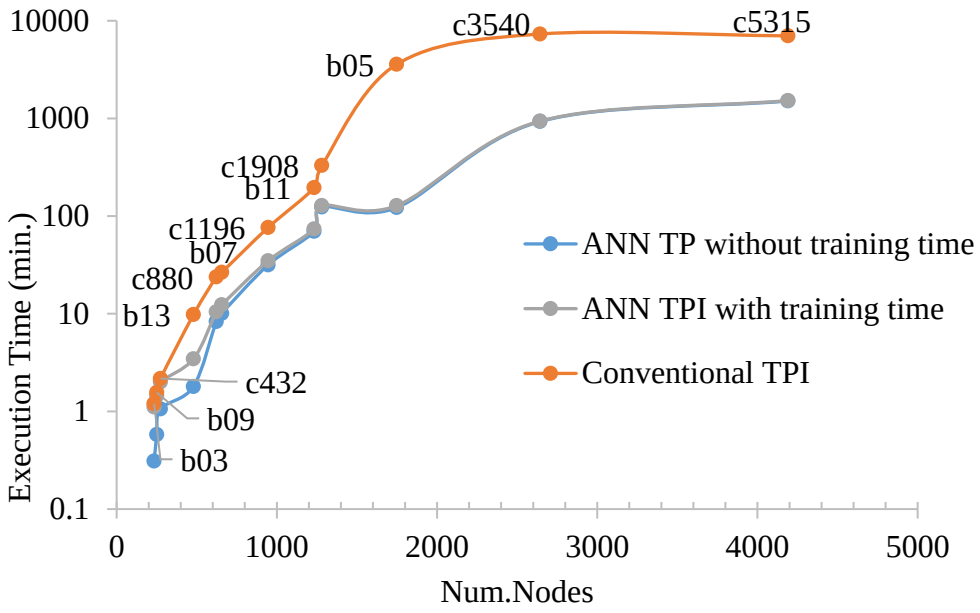


Figure 4.9: ANN and conventional TPI targeting SAF in time comparison.

4.9 Conclusion

This chapter has demonstrated the effectiveness of applying ANNs to TPI for increasing SAF coverage. This chapter has shown that ANNs can effectively predict the impact TPs will have on the SAF coverage for pseudo-random stimulus compared to conventional TPI methods. It has also been demonstrated that TPI algorithms can use ANNs to insert TPs into a circuit at significantly increased speeds, thereby allowing design efforts to be focused elsewhere or allowing additional effort to be spent on increasing TP quality.

Chapter 5

ANNs TPI Targeting Transition Delay Fault Coverage

Delay-causing defects become more common as semiconductor sizes scale [34], and the SAF model can no longer be the sole fault model for evaluating LBIST quality. Usually, control TPs are used for increasing SAF coverage, but they are known to block other faults found in modern technologies [21]. Roy et al. [21] found typically-implemented control TPs (i.e., control-0 test points using AND gates and control-1 test points using OR gates [10]) masked the propagation and excitation of delay faults when active. Two alternatives can mitigate delay masking detriments, but both have drawbacks. First, observe test points can be used exclusively, but exciting RPR faults may require forcing values in circuits and implementing observe test points require expensive latches or output pins. Second, circuits can be tested with TPs both on and off. This is normal industry practice, but disabling TPs mitigates their purpose: ideally, active TPs can test RPR and delay faults simultaneously. Therefore, this study's ANN has the following goal: insert TPs, both control and observe test points, to detect RPR faults without masking delay faults. This chapter aims to address this by comparing ANN TPI targeting TDF against comparable conventional TPI [33] in SAT fault, TDF and execution time.

Much of the content of this chapter appears in work previously published by the author [99].

5.1 Introduction to the Transition Delay Fault Model

With the current generation microprocessors becoming faster and more complex, new challenges are faced in their testing. Research and experiments [100], [101] have shown that for high design quality requirements it is not sufficient to test a design only for SAFs. Hence functional at-speed tests are sometimes used in addition to the conventional scan tests.

The TDF model assumes that delay faults affect only one gate in the circuit. There are two transition faults associated with each gate: a slow-to-rise fault and a slow-to-fall fault. It is assumed that in the fault-free circuit each gate has some nominal delay. Under TDF model, the extra delay caused by a fault is assumed to be large enough to prevent a transition from reaching any primary output at the time of observation.

To detect a transition fault in a combinational circuit, it is necessary to apply two input vectors, $V = < \boldsymbol{v_1}, \boldsymbol{v_2} >$. The first vector, $\boldsymbol{v_1}$, initializes the circuit, while the second vector, $\boldsymbol{v_2}$, activates the fault and propagates its effect to some primary output. During the application of the second vector, the fault behaves as a SAF and vector $\boldsymbol{v_2}$ can be found using SAF test generation tools. For example, for testing a slow-to-rise transition, the first pattern initializes the fault site to 0, and the second pattern is a test for stuck-at-0 at the fault site. A transition fault is considered detected if a transition occurs at the fault site and a sensitized path extends from the fault site to a circuit primary output [102].

The main advantage of the TDF compared to the PDF model is that the number of faults in the circuit is linear in terms of the number of gates. Also, SAF test generation and fault simulation tools can be easily modified for handling transition faults [102].

5.2 Proposed Method

This proposed method used a trained ANN to find a TP's impact on delay fault coverage, and the process was similar to Chapter 4. The label was found through training changes to detected TDF coverage (instead of SAF coverage). This change was a two-step process. First, the training data collection program used the TDF model in lieu of the SAF model. Alternative delay fault models exist, but studies show the TDF model sufficiently represents industrial faults, and implementing the TDF model requires minimal changes to SAF simulators [93]. Second, instead

of probabilistically simulating $2^{I'}$ vectors for each sub-circuit, the training data generation program probabilistically simulated $2^{2 \cdot I'}$ vector pairs because TDFs require two vectors to detect: one to set the initial state and another to launch the circuit transition. In this study, $V \gg 2^{2 \cdot I'}$.

5.3 Input Features

Many input features and feature extraction methods of ANN were similar to Chapter 4 and are briefly redescribed here. First, since ANNs had pre-defined sizes and large ANNs required more training time and data to be useful, the ANN evaluated a TP's quality using a sub-circuit centered around a TP's location. Second, the ANN's input features were the probabilistic calculations of the lines in this sub-circuit. These calculations were COP controllability and observability values [40], which were widely used in many TPI algorithms [33], [93]: these values estimated the probability that a circuit line was logic-1 (and inversely, logic-0) and the probability that a fault on a circuit line was observed, respectively. Third, the gate types in the extracted sub-circuit were represented as one-hot binary strings. During TPI and training data generation, these features were calculated and passed to the ANN in a set order: first controllability values, then observability values, then gate types, all of which were given in a breadth-first order [91].

5.4 Output Label

The output label of the ANN was the change in TDF coverage on a sub-circuit centered around the TP location when the TP was active. Many delay fault models attempted to accurately model delay-causing defects [103], and the TDF model is known to model realistic defects. The use of a sub-circuit allowed training data to be efficiently collected , but this created a potential detriment: other TP quality-calculating heuristics could calculate the effect a TP had on an entire circuit (as was the case of this study's comparison, [93]), and therefore may return more accurate results.

5.5 Training Data Generation

Training an ANN required a set of features attached to known solution labels, or "training data". This training data came from problems that were solved by other means, i.e., problems that were solved by hand or a best-solution that was solved through exhaustive exploration.

The ANN's output label was the "true" TDF coverage on a sub-circuit when inserting a TP: this was obtained through fault simulation. Fault simulation is a CPU-intensive process, which is why most conventional TPI methods use fault coverage estimation techniques [33], [93]. Such estimations can give inaccurate fault coverage results [40], and thus less effective TPs may be chosen for insertion. The ANN attempted to more accurately predict the TDF coverage impact of a TP without performing fault simulation during TPI. Instead, fault simulation was performed during training: the impact of training CPU time was distributed across all uses of the ANN TPI, thus if the ANN was re-used enough times, the impact of training time became negligible.

To obtain the training label (TDF coverage) from a sub-circuit, techniques were needed to reduce training data generation time. Presuming V pseudo-random vector pairs (pairs are required for delay fault simulation) were applied to a sub-circuit with I inputs, a vector pair would be applied more than once, which was likely for significant values of V for smaller sub-circuit sizes (which for this study, $I \leq 6$), and fault simulating a vector pair more than once wasted simulation time. This was exacerbated when sub-circuits were extracted from a circuit, since input vectors to the sub-circuit are not random: paths from circuit inputs to sub-circuit inputs weighed circuit value probabilities and made some vectors more likely to occur than others.

To prevent training vector pairs being applied more than once, the following technique used heuristically-calculated circuit controllability calculations to conditionally simulate all vector pairs. First, for each vector v among the 2^I sub-circuit input vectors, the probability of applying

the vector was calculated using COP circuit controllability information. The probability of the required logic-0 and logic-1 values occurring ("C_i'") was used to calculate this probability:

$$p(vector\ simulated) = p_v = \prod_{\forall i \in I}(1 - (1 - C_i')^N) \qquad 5.1$$

Second, a sub-set of all the 2^{2I} sub-circuit vector pairs was chosen to be simulated using the previously calculated probabilities, i.e., a vector pair was simulated with probability $p_{v_1} \cdot p_{v_2}$. This simulation of vector pairs was the most significant improvement in this ANN compared to that presented in Chapter 4. Third, each of the sub-circuit vector pairs was fault simulated $T + 1$ times: once without any TPs and once for each of the T TPs (in this study, $T = 3$, which represented control-0, control-1, and observe points). Fault effected on output pins were probabilistically observed using "observability" calculations from COP [40]. Lastly, the controllability, observability, and gate types of the sub-circuit were used for training features and the change in fault coverage created by the T TPs were used as training labels for T separate ANNs.

5.6 Training

The training process was the same as Section 4.6. Three separate, smaller ANNs were trained representing the three TP types: control-0, control-1, and observe TPs. ANN training was performed under various hyperparameters to minimize ANN error (in this study, mean squared error).

5.7 Experiment Results

5.7.1 Experiment Setup

The experiment workstation and software were the same as Section 4.8.1.

The conventional TPI method used for comparison was a modified version of [33]. It was an iterative TPI method representative of TPI implemented in industry, although industrial tools will have many additional computation-time optimizations. Conveniently, ANN directly replaced

the “quality” measuring method in [33], which eliminated other sources of fault coverage and CPU time differences. However, the method in [33] did not attempt to insert TPs to increase delay fault coverage. To correct this, the “quality” calculation to judge the detection probability of a stuck-at fault (the controllability of a line, multiplied by the observability of the line) was directly replaced by the calculation proposed in [93], which attempted to model the probability a transition would occur on a line and be observed at a circuit output (the controllability of a line, multiplied by the inverse of this controllability, multiplied by the observability of the line). This created a TPI method which attempted to increase delay fault coverage that used the same information as the proposed ANN. Therefore, the only difference in run-time and fault coverage quality was in judging the “quality” of a given TP.

This study used post-synthesis logic netlists of the ISCAS’85 [96] and ITC’99 [97] benchmarks. A randomly selected subset, given in Table 5.1, was used solely for training, and the table provides the number of sub-circuits (i.e., TP locations) extracted for training -the number of extracted sub-circuits is proportional to the circuit size. It was presumed that circuits were tested in a full-scan environment with scan chains loaded from a 31-bit long PRPG and delay tests were performed using “launch-off-shift”, hence the “inputs” and “outputs” in Table 5.1 and Table 5.2 include flip-flops outputs and inputs, respectively. When TPs were present, control points were enabled by a common “TP enable” pin which was active for half of all vectors applied. Although other methods to selectively enable sub-sets of TPs existed in literature [30], such architectures were not the scope of this study.

Table 5.1: Train Benchmarks for ANN TPI Targeting TDF

Benchmarks	Training Samples	Inputs	Outputs	Gates
c17	2	5	2	13
b02	2	5	5	32
b06	2	11	15	65
b08	8	30	25	204
b10	9	28	23	223
c499	12	41	32	275
c1355	24	41	32	619
b04	32	77	74	803
b12	51	126	127	1197
c2670	49	233	140	1566
c6288	100	32	32	2480
c7552	143	207	108	3827
b15	445	485	519	9371
b14	471	277	299	10343
b20	950	522	512	20716
b21	980	522	512	21061
b22	1443	767	757	30686
b17	1562	1452	1512	33741
b18	5276	3357	3343	117941

Table 5.2: ANN TPI and Conventional TPI Targeting TDF Experimental Results

						Fault Coverage, TDF (%)			Fault Coverage, SAF (%)			TPI Time (s)		
Benchmarks	In.	Out.	Gates	Vec.	TPs	No TPs	Conventional TPI	ANN TPI	No TPs	Conventional TPI	ANN TPI	Conventional TPI	ANN	ANN (inc. training)
b03	34	34	190	216	1	78.31	78.76	78.91	92.14	92.38	92.86	1.00	0.01	0.54
b09	29	29	198	216	1	66.00	67.70	74.31	86.85	89.20	88.56	1.00	0.08	6.08
c432	36	7	203	5832	2	95.49	95.49	95.83	98.71	98.71	99.08	12.00	0.12	9.01
b13	63	63	415	512	6	81.19	81.80	83.24	94.11	94.34	95.93	10.00	0.60	45.07
c880	60	26	469	1331	4	93.35	92.32	93.58	98.29	97.08	98.39	44.00	0.66	49.57
b07	50	57	490	512	4	77.10	77.94	78.42	85.89	87.83	87.68	56.00	0.80	59.94
b11	38	37	801	10000	8	86.52	92.10	87.60	94.15	96.58	95.93	283.00	2.80	210.01
c1908	33	25	938	10000	9	98.85	98.56	98.85	99.66	99.66	99.66	794.00	3.36	252.38
b05	35	70	1032	10000	10	70.08	75.86	71.82	76.23	79.98	78.21	649.00	4.80	360.54
c3540	50	22	1741	10000	12	89.77	92.06	89.85	95.39	95.61	95.61	3746.00	8.00	600.90
c5315	178	123	2608	6859	13	97.50	97.52	97.54	98.97	98.95	99.00	3746.00	13.00	976.46

After many trials of ANN training, the final ANN configuration chosen was 1 hidden neuron layer of 128 neurons, with each hidden neuron using a sigmoid [104] activation function.

When performing TPI and fault simulation, the number of vectors applied (and used for TPI calculation) was individualized for each benchmark and was given in the column labeled “Vec.”

This value was calculated based on projections given in [105]: fault simulation with random vectors was performed on TP-free circuits until 63.2% stuck-at fault coverage was obtained, and this number of random vectors was then used to project the number of random vectors needed to obtain 95% stuck-at fault coverage, and this number or ten thousand was used, whichever was lesser. This represented an industrial environment where either (1) TPs were intended to increase stuck-at fault coverage as much as possible after 95% stuck-at fault coverage was achieved, or (2) 95% stuck-at fault coverage was not obtainable in ten thousand vectors and TPs were inserted to assist in obtaining 95% stuck-at fault coverage.

5.7.2 TPI Experiment Results

This experiment examined the delay fault coverage obtained when TPs were inserted using a conventional TPI method compared to using ANN, and the results of this experiment are given in Table 5.2. These benchmarks were not used for ANN training to prevent an advantage to the ANN. TPs were inserted until either (1) no TP was calculated to increase fault coverage, (2) 99% fault coverage was predicted to be obtained, or (3) the number of inserted TPs was greater than 1% of all nodes. Afterwards, traditional TPI was performed until the same number of TPs was inserted, with the number TPs given under the heading “TPs” in Table 5.2.

Fault coverages are given in Table 5.2, and changes in fault coverage are plotted in Figure 5.1 and Figure 5.2, which shows notable trends. First, inserting TPs using the proposed method never decreased TDF or SAF coverage, while the conventional method decreased (see *c880* and *c1908*). Additionally, the ANN TPI method achieved comparable TDF and SAF coverage compared to the conventional TPI method.

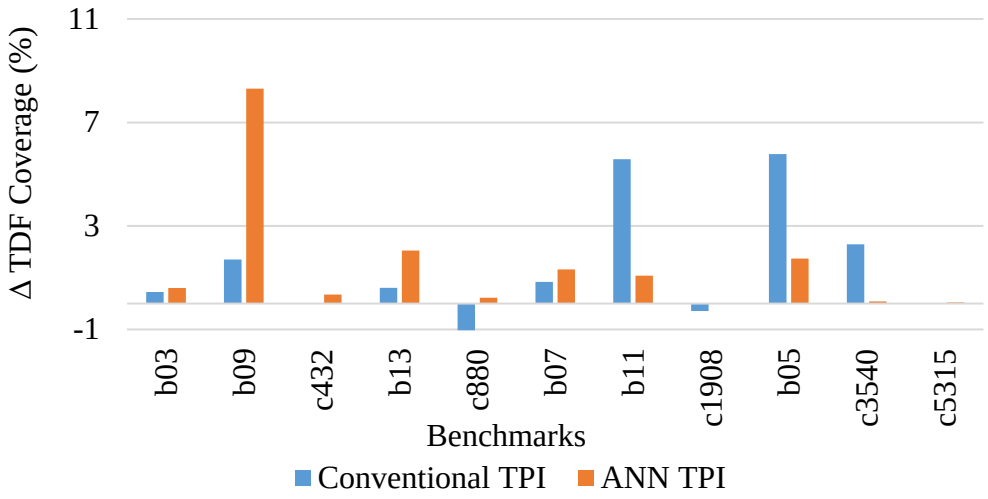


Figure 5.1: ANN and conventional TPI targeting TDF in TDF coverage comparison.

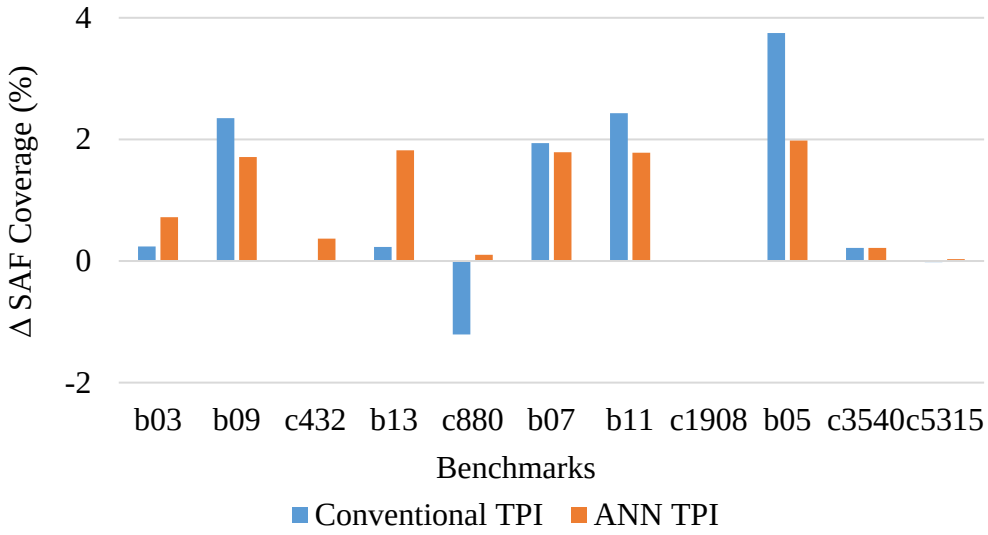


Figure 5.2: ANN and conventional TPI targeting TDF in SAF coverage comparison.

An additional result extracted from the previous experiment was the time required to perform TPI. This is given in Table 5.2 under the heading “TPI Time (s)” for the two TPI methods, which the ANN substantially decreased TPI time, and this remained true when training data generation and ANN training was considered. These results are transposed to Figure 5.3, which plots TPI CPU time relative to the number of logic gates in each circuit. When the training data

generation time (3,544 seconds) and ANN training time (864 seconds, which included exploring multiple ANN configurations) was distributed among benchmarks by circuit size (i.e., more time was added to circuits with more logic gates), the ANN TPI time was still significantly lower for all benchmarks.

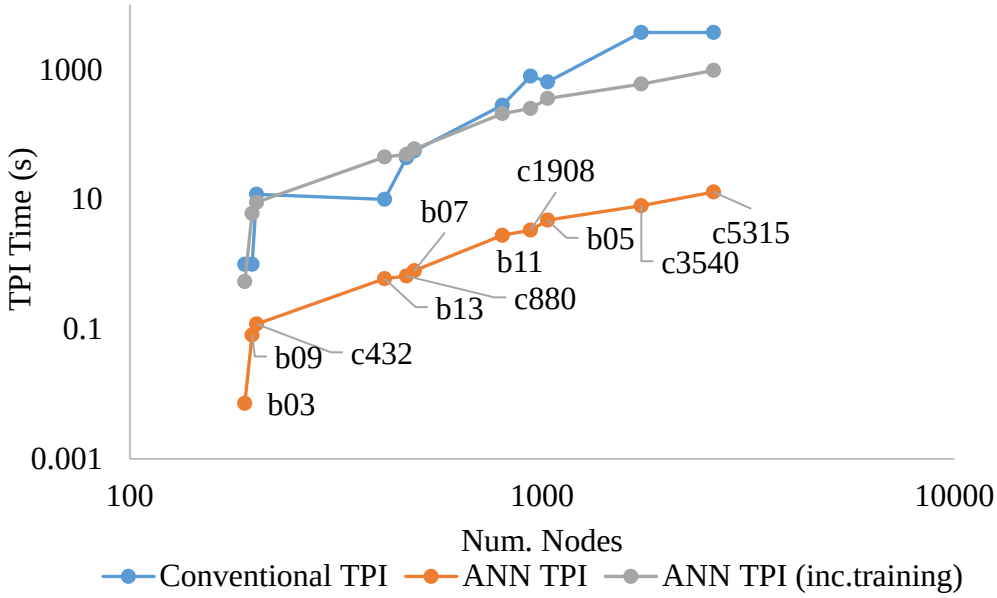


Figure 5.3: ANN and Conventional TPI targeting TDF in time comparison.

5.8 Conclusion

This chapter has demonstrated the effectiveness of applying ANNs to TPI for increasing TDF coverage. This chapter has shown that the ANN TPI obtains comparable TDF coverage and SAF coverage results compared to conventional TPI, and it also obtains results in significantly less time. Given a challenge of TPI (and other EDA problems) is managing and sharing computational resources, reducing TPI time without reducing TP quality is beneficial to circuit designers.

Chapter 6

Developments in ANN TPI

Although the previous Chapters' ANNs were effective in providing better TPs for detecting SAFs and TDFs under pseudo-random stimulus, room for improvement remains. The previous ANNs were trained in a way that may not accurately reflect the nature of the pseudo-random stimulus, thus nuanced changes to the ANN training process and the ANN's use may further increase its utility.

As of this writing, the content of this chapter is currently awaiting publication.

6.1 ANN Input Feature Development

Compared to previous ANN input features, this section adds the number of applied pseudo-random vectors V to the end of the ANN feature string. Depending on the number of vectors to apply, the optimal location of TPs can change. To account for this, this study appends the number of vectors to apply, V, to the ANN feature string.

6.2 ANN Output Label Development

The ANN's output label is the number of additional (or possibly fewer) faults detected in the sub-circuit when inserting a TP into its center. This label contrasts with the label in Section 4.4 and Section 5.4, which was the change in fault coverage in the sub-circuit. This new label more accurately models the impact of "partial" sub-circuits: two TPs on separate sub-circuits can have the same impact on sub-circuit fault coverage (e.g., +10%), but one sub-circuit may have more faults than the other, thus making the sub-circuit with more faults the better choice. If a TP will decrease the number of faults detected (by masking faults), the label is a negative number.

Using the change in the number of faults detected in a sub-circuit, as opposed to the number of faults detected in the entire circuit, poses a challenge: the quality measure may not adequately

represent the TP's impact on the entire circuit. It is possible that many additional faults will be detected in the sub-circuit but fewer faults will be detected in the overall circuit, and vice versa. Section 6.3 explores if this potential detriment has a negative impact on fault coverage.

6.3 Experiment Results

6.3.1 ANN Training Experiments

Hyperparameters greatly affect the quality of an ANN: the ANN topology (convolutional, deep neural networks, fully/sparsely connected, etc.), the number of neurons, the number of hidden neuron layers, activation functions, etc. Additionally, training parameters and effort significantly influence ANN quality, e.g., step size (how much to change dendrite weight and bias magnitudes each iteration) and initial conditions.

Given the vast parameter exploration space, this chapter shows the exploration of two hyperparameters: training data set size and the number of ANN neurons. In these explorations, all other hyperparameters are constant: these constant hyperparameters include neuron arrangements (two hidden layers, with variable amounts of neurons in the first hidden and a single neuron in the second hidden layer, and neurons using sigmoid [104] activation functions) and training parameters (using the Adam optimization algorithm [98] with a 0.01 training step size). Training performs 55,000 iterations for each ANN: for every ANN in this study, ANN error stops decreasing before these iterations are complete.

Training Data Set Size

A known roadblock to previous ANN implementations is the lack of available training data [11]: if not enough data can find a correlation between input features and desired output labels, then creating a useful ANN is impossible. However, providing too much training data increases

training time and can degrade ANN quality through “overfitting [106]”. For these reasons, this chapter explores how much data is required to minimize ANN error.

A random selection of ISCAS’85 [96] and ITC’99 [97] benchmarks serve as training circuits. Table 6.1 lists these benchmarks, their physical qualities (the number of gates, inputs, and outputs), and the number of randomly-extracted training samples per circuit: the number of samples extracted per circuit is proportional to the size of each circuit. Inputs and Outputs include the outputs and inputs of latches (respectively); since this study presumes circuits are tested in a full-scan environment thus latches are fully observable and controllable during test. Note that that some circuits are small and easy to test with pseudo-random stimulus; this is desirable for generating training data, since “poor” TPs must be accurately evaluated as much as “good” TPs.

Table 6.1: Train Benchmarks for ANN TPI

Benchmarks	In	Out	Gates	Training Samples
c17	5	2	13	2
b02	5	5	32	2
b06	11	15	65	2
b08	30	25	204	4
b10	28	23	223	5
c499	41	32	275	6
c1355	41	32	619	12
b04	77	74	803	16
b12	126	127	1197	26
c2670	233	140	1566	25
c6288	32	32	2480	50
c7552	207	108	3827	72
b14_1	277	299	7145	165
b15_1	485	519	13547	311
b21_1	522	512	14932	345
b20_1	522	512	14933	342
b22_1	767	757	22507	516
b17_1	1452	1512	41080	949
b18_1	3357	3343	111802	2520
b19_1	6666	6672	226066	5087

Figure 6.1 shows the impact different training data sizes have on the ANN TPI targeting SAF error. Each horizontal point corresponds to selecting a different number of randomly-selected training samples: 2,097, 10,457 (used for training the ANNs later), 20,910, and 27,181 samples. These amounts correspond to taking fractions of all possible sample locations: 0.1%, 0.15%, 1%, and 1.3%, respectively. The number of neurons in the first ANN layer is always 128 neurons. The plot labeled "training time" shows the time required to train the ANN. The plot labeled "training error" shows the final MSE of the ANN; ideally, this is 0% (i.e., the output label obtained precisely matches the expected label for training samples). The plot labeled "testing error" shows the MSE for 2,244 additional randomly-selected samples that are not used for training.

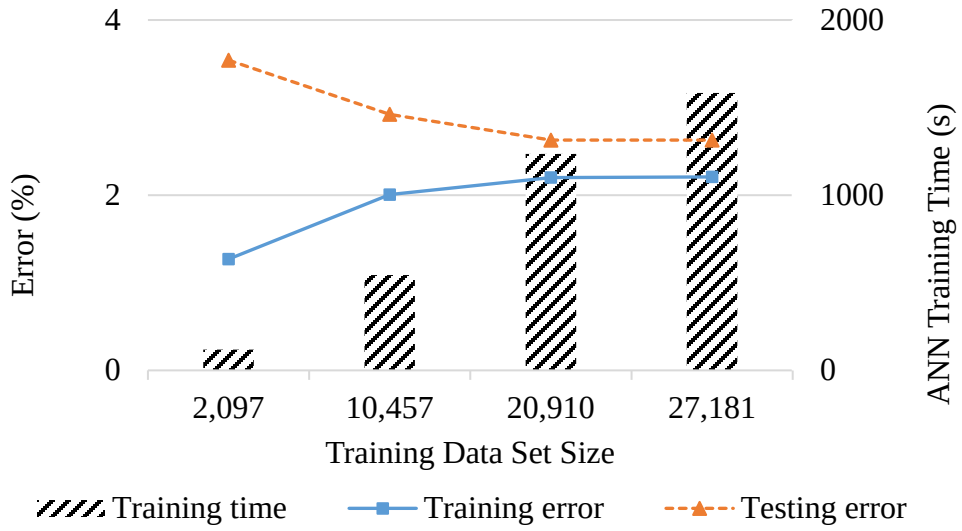


Figure 6.1: Increasing ANN training data typically increases ANN accuracy, but also increases training time.

Figure 6.1 shows three clear trends regarding ANN training time and error. As more data trains the ANN, training requires more time to minimize ANN error. This is because minor changes in ANN weights and biases are more likely to create errors among more training data samples. Likewise, using more training data can increase the error, since satisfying all training samples

becomes more difficult. However, although the error of matching training data increases, the error of matching non-training "testing" data decreases up to an acceptable point in reasonable time.

Although 20,910 training data samples decrease ANN error than 10,457 samples, using this many samples doubles training time with a marginal impact on error. Therefore, future experiments more than use an ANN trained with 10,457 training data samples.

ANN Complexity

Many hyperparameters impact ANN complexity, but this study simplifies complexity to a single variable: the number of neurons present in the first hidden layer. Like the previous experiment, Figure 6.2 gives a plot of ANN training time, training error, and testing error. 10,457 data samples train these ANNs with all other parameters matching the previous experiment.

Figure 6.2 shows, like the previous experiment, more neurons in the first layer translates to more time needed to minimize ANN error, but training error and testing error also decrease. This is because using more neurons can find more accurate correlations between features and desired labels, but more neurons need more time to learn the correlation.

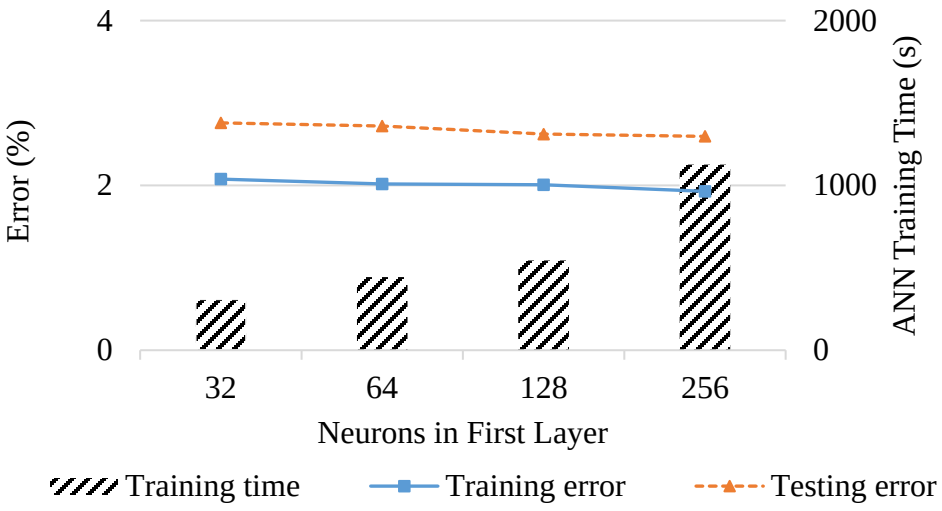


Figure 6.2: Increasing ANN complexity can decrease ANN error, but training time also increases.

6.3.2 ANN TPI Experiment

The experiment setup is same as in Section 4.8.1 and Section 5.7.1. The conventional TPI targeting SAF used for comparison is from [33] and conventional TPI targeting TDF used for comparison is modified from [33] and [93]. These conventional TP-evaluating subroutines use the same information as the proposed ANN (i.e., COP values), so ANN can directly replace the TP-evaluation subroutine of conventional TPI, difference in run-time and fault coverage quality is from these subroutines.

ISCAS'85 [96] and ITC'99 [97] benchmark circuits not used for training are used for performing TPI: this prevents a bias favoring the ANN. Table 6.2 gives details of these benchmarks. Tests (generated from a 64-bit PRPG) generate different numbers of vectors for each benchmark circuit: these tests obtain 95% fault coverage without TPs, apply ten thousand vectors, or take 15 minutes to simulate. This represents an industrial environment where either (1) the circuit does not obtain 95% fault coverage without TPs and TPs are placed to increase fault coverage as much as possible, or (2) ten thousand vectors do not obtain 95% fault coverage and TPs attempt to increase fault coverage to acceptable levels. For this study's original software, fault simulation may not obtain significant fault coverage in reasonable time (e.g., *b14* obtains a minuscule 13% fault coverage in 15 minutes when applying 960 vectors), but this study presumes high-performance industrial programming will remedy this and fault coverage trends seen in this study will apply when simulating more vectors.

To explore the impact of ANN subcircuit sizes, this study trains several ANN models. For larger subcircuits, the neurons in the ANN increase to capture more complex relationships between input features and the output label. The number of neurons in each ANN model is 128 for L=2, 256 for L=3, and 512 for L=4. All other hyperparameters are identical. With all these

configurations, this made a total of 2*3*3=18 neural networks trained: they were trained for each parameter of fault model (stuck-at fault and transition delay fault), test point to analyze (control-0, control-1 and observe), and sub-circuit size to analyze.

Table 6.2: Different sub-circuit size ANN TPI and Conventional TPI Experimental Results

	Benchmark Information					TPI Time(s)			
Benchmarks	In	Out	Gates	Vec.	TPs	Conventional TPI	ANN TPI (L=3)	ANN TPI (L=4)	ANN TPI(L=5)
c432	36	7	203	9984	2	13.89	0.08	0.06	0.18
c880	60	26	469	9984	4	45.45	0.08	0.07	0.8
c1908	33	25	938	9984	9	911.08	0.44	0.36	4.17
c3540	50	22	1741	9984	5	1831.23	0.66	0.53	4.39
c5315	178	123	2608	9984	6	1806.72	1.76	1.39	7.45
b03	34	34	190	9984	1	1.21	0.89	0.73	0.07
b05	35	70	1032	9984	10	727.15	1.20	1.01	5.38
b07	50	57	490	9984	4	56.16	8.76	7.14	0.86
b09	29	29	198	9984	1	2.04	5.55	4.63	0.07
b11	38	39	801	9984	3	119.14	4.37	3.73	1.16
b13	63	63	415	9984	4	9.74	7.60	12.46	0.63
b14	277	299	10343	960	1	5031.64	5.27	4.47	5.69
b15	485	519	9371	1920	1	6472.05	5.71	4.85	5.27
b17	1452	1512	33741	256	1	26607.69	11.57	9.73	15.65
b20	522	512	20716	448	1	14479.94	11.67	9.91	11.52
b21	522	512	21061	448	1	14972.79	19.80	15.87	11.86

Stuck-at Fault Coverage

These experiments find the SAF coverage of the four TPI methods: the conventional TPI targeting SAF [33], the ANN TPI targeting SAF, the conventional TPI targeting TDF ([33], [93]), and the ANN TPI targeting TDF. Figure 6.3 and Figure 6.4 plot the base SAF (i.e., the original circuit with no TPs), and also plot the change in SAF coverage after TPI.

Performing SAF simulation shows several noteworthy trends. First, all ANNs consistently obtain favorable SAF coverages, but the conventional TPI [33] sometimes (i.e., for *b05*) decreases

SAF coverage. Second, as the ANN analyzes larger sub-circuits, the ANN selects higher quality TPs and further increases SAF coverage, presumably because TP evaluation is more accurate. Third, as shown in Figure 6.4, it appears that methods which target SAF (both conventional and ANNs) do not consistently perform better at increasing SAF coverage compared to delay-fault targeting TPI: this result is counterintuitive and motivates future studies.

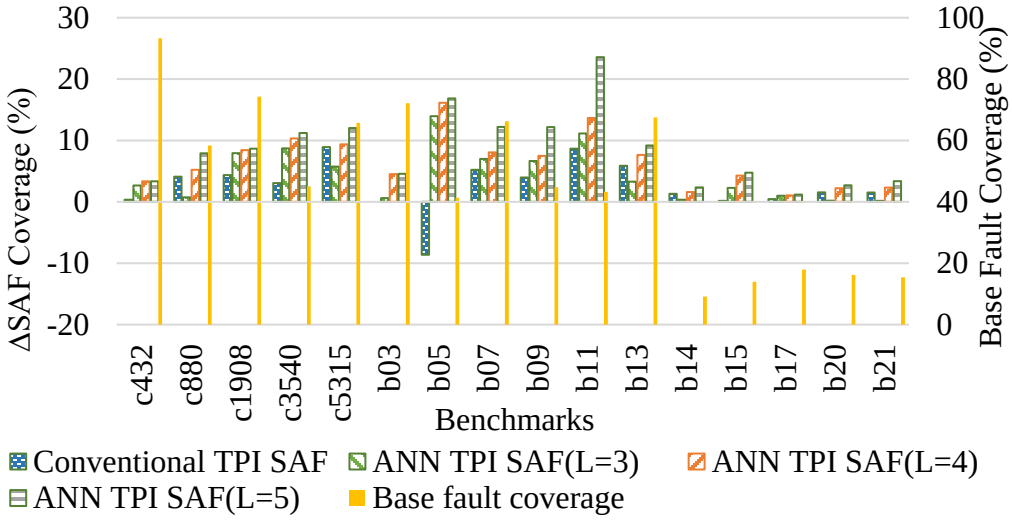


Figure 6.3: Different sub-circuit size ANN TPI and conventional TPI in SAF coverage comparison.

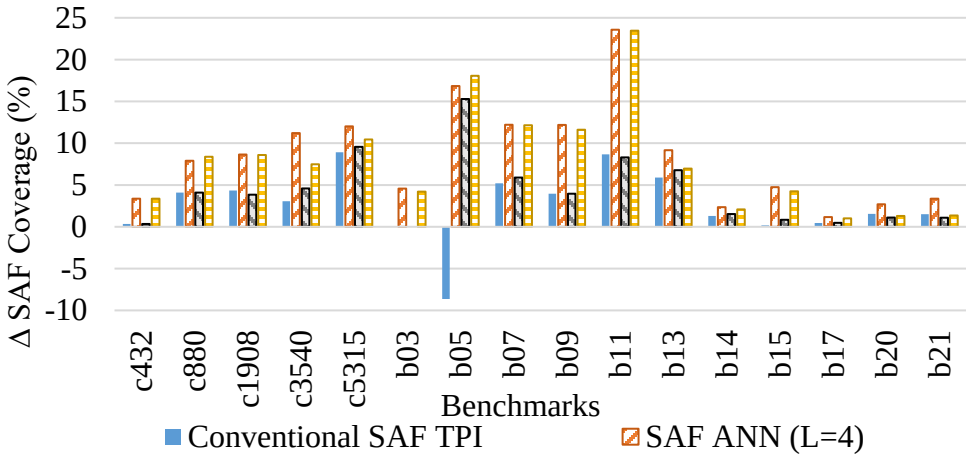


Figure 6.4: SAF targeting conventional and ANN TPI may not increase SAF coverage compared to their TDF targeting counterparts.

Delay Fault Coverage

Like the previous experiment, this experiment simulates delay faults (more specifically, TDFs) with the given number of vectors. 64-bit PRPGS load the scan chains and apply vectors using a

Launch-off-scan method (a.k.a. “skew load”) [107]. Figure 6.5 and Figure 6.6 plot the delay fault coverage in terms of the base delay fault coverage (i.e., the TDF coverage with no TPs) and the change in the fault coverage with TPs inserted with different TPI methods.

The first observation from Figure 6.5 and Figure 6.6 is that all ANN TPI methods consistently obtain favorable delay fault coverage results. On average, an ANN selects TPs that increase fault coverage once for SAF, again for TDF more than using a heuristic. Also like the SAF targeting ANN, the TDF targeting ANN selects higher quality test points when analyzing larger sub-circuits.

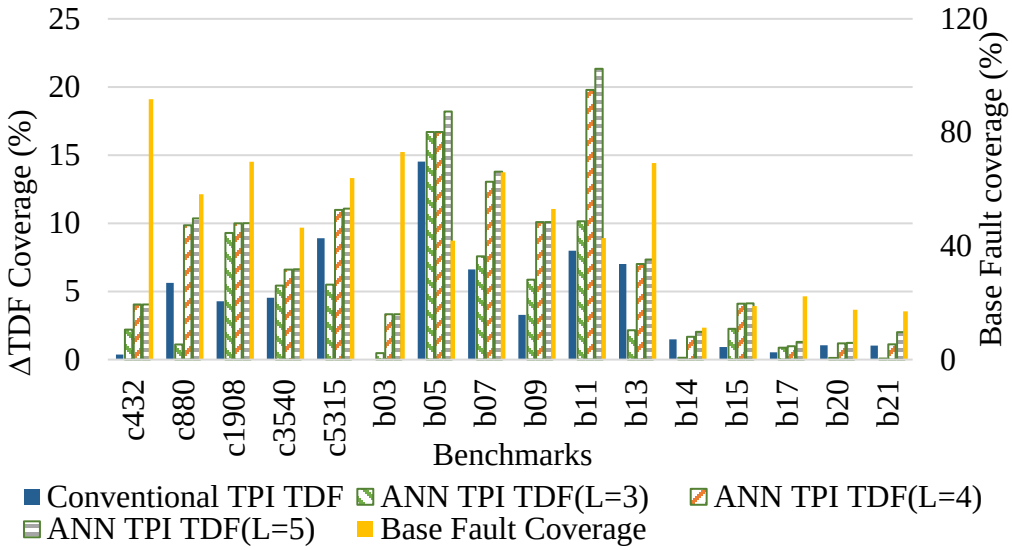


Figure 6.5: Different sub-circuit size ANN TPI and conventional TPI in TDF coverage comparison.

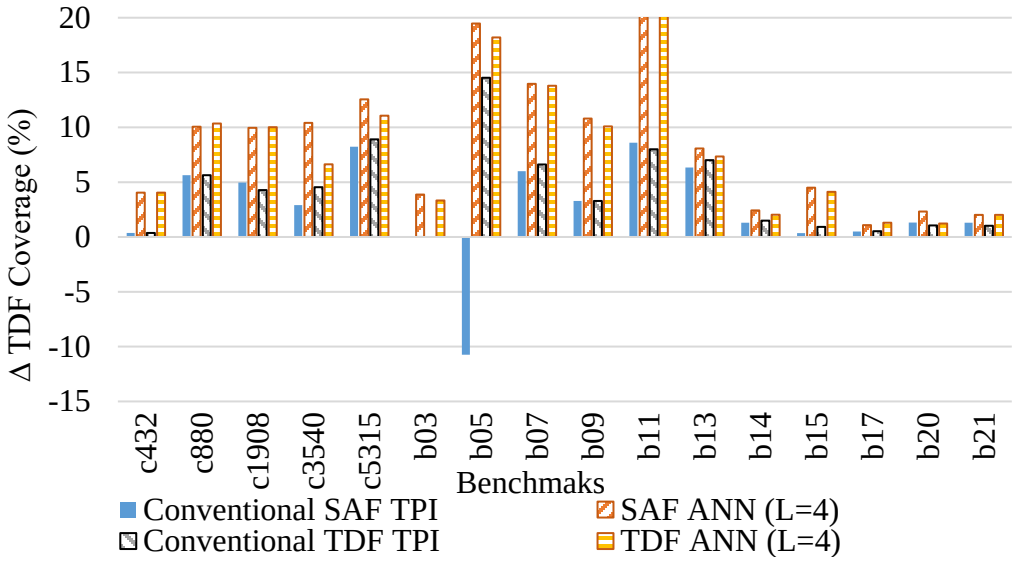


Figure 6.6: TDF targeting conventional and ANN TPI may not increase TDF coverage compared to their SAF targeting counterparts.

TPI Time

An additional result extracted from the previous experiments was the time required to perform TPI. This is given in under the heading "TPI Time (s)" for the stuck-at fault targeting heuristic and stuck-at fault targeting ANNs. Figure 6.7 also plots these results. This figure includes plots for the ANNs that include the training data generation and ANN training time: these plots distribute this time among benchmarks by circuit size (i.e., by adding more time to circuits with more logic gates). For the ANNs with $L = 2$, $L = 3$, and $L = 4$, this training data generation and ANN training time is $231 + 554$, $349 + 2{,}004$, and $458 + 5{,}295$ seconds, respectively.

The TPI time results from Table 6.2 and Figure 6.7 definitively show that performing TPI using ANN requires orders of magnitude less time compared to conventional methods. Additionally, as the previous results showed, fault coverages obtained with these ANNs are comparable or superior to heuristic-derived results, which means the decreased TPI time does not sacrifice TP quality.

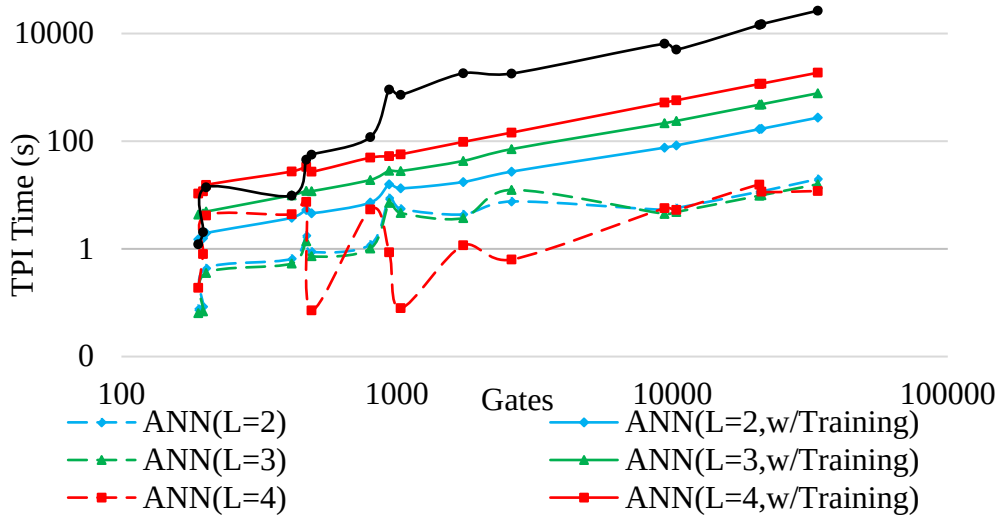


Figure 6.7: Conventional TPI and different sub-circuit ANN TPI in time comparison.

When accounting for ANN training data generation and training time, the time to perform TPI still favors ANN TPI. In industry, the EDA company trains the ANN for only once, and then circuit developers can re-use the ANN numerous times. If developers re-use the tool enough times, the impact of training time becomes negligible. However, when distributing the training time among TPI instances, computational time still favors ANN TPI methods. This would represent a technique of training specifically for a single user, which although is impractical, this still out-performs heuristic TP evaluation.

Chapter 7

Test Power Reduction through Test Point Insertion

7.1 Introduction

In today's foundries, fault coverage is not the sole concern of design for test engineers; increasing circuit density deployed in portable and high-performance microelectronic devices make power consumption a prime concern for VLSI designers. Modern microprocessors are "hot", and their power consumption creates undesired consequences. Circuits become less reliable when large and instantaneous power dissipation causes overheating: circuit failure rates roughly double for every 10 °C increase in temperature [108].

Beyond chip functionality, excessive power during test increases manufacturing costs by requiring more expensive chip packaging or by reducing yield [109]. During die test after wafer etching, bare dies lack packaging with heat removal hardware, thus if bare-die testing fails to carefully control power consumption, the test may destroy the die, thus decreasing yield and increasing production costs. In addition to bare die testing, during wafer test, wafer probes have current limits and high test switching activity increases power instability, this instability changes logic states and causes false failures, thus reducing yield [110].

Studies proposed several solutions for reducing power consumption during test [111], but all have trade-offs. Designers can use oversized power supplies, packaging, and cooling to handle high test current, or designers can test circuits at reduced clock frequencies. However, these solutions have shortcomings: additional test hardware increases cost, and frequency scaling increases test time and decreases defect coverage since reduced clock frequencies will mask dynamic faults.

One widely adopted form of test creates particular power concerns: LBIST. Embedding test hardware on-circuit provides numerous advantages, ranging from easy in-field test and higher quality at-speed test, but achieving high fault coverage for a circuit containing many random pattern resistant faults requires high switching activity during test. Since dynamic power dissipation in CMOS circuits is proportional to switching activity, LBIST may damage a CUT due to excessive heat dissipation or create false failures. Therefore, a balance between high fault coverage and low power consumption is desirable, hopefully without sacrificing one for the other.

TPs can increase circuit quality, but literature studying their impact on power is sparse. Several studies proposed inserting TPs to reduce test power, but these studies presumed unique TP environments that only reduce power during scan: Stefan et al. [112] placed TPs at latch outputs to reduce average power while Sankaralingam et al. [61] reduced peak scan power, but both studies restricted TP placement to latch outputs.

Because power issues may only appear after tape-out, engineers may require *engineering change orders* (ECOs) [113] to remedy test power, but these remedies may impact fault coverage. Like how engineers use metal-mask ECOs to address functional and timing errors found after tape-out [113], DFT engineers may implement TPs using redundant TP hardware (i.e., unused control-0 and control-1 gates) after they find test power issues during silicon bring-up. However, studies must find the TPI procedures that designers must follow to reduce power without affecting test quality under ECO environments.

Given the hazards of excessive power during test, this study used TPs to reduce both average and peak power on existing pseudo-random tests while minimizing impacts on fault coverage. Additionally, this study explored the impact of dividing TPI into multiple phases and finds its impact is substantial.

7.2 Background

7.2.1 Power and Test

Power consumption in CMOS circuits can be static or dynamic. Leakage current drawn from power supplies cause static power, while brief short-circuit current and load capacitance causes dynamic power. Although studies predicted static power may dominate dynamic power in future technologies, dynamic power is still significant in modern circuits [110] and is the subject of this study.

As *system-on-chip* (SoC) designs and deep-submicron geometries proliferate, larger designs, tighter timing constraints, and higher operating frequencies affect power consumption metrics [110]. Average power is the ratio of energy to test time. Instantaneous power is the power consumed at any instant, usually measured as the power consumed after a synchronizing clock. Peak power is the highest instantaneous power and determines the circuit's electrical limits and package requirements. If peak power exceeds the designed limit, the circuit may malfunction. This study addresses both peak power (which may cause false failures) and average power (which may cause overheating and circuit damage).

Historically, test engineers evaluated test strategies through area overhead, fault coverage, test application time, test development effort, etc., but recent high-performance and low-power devices make power management a critical parameter that test engineers cannot ignore. Power consumption during test can be twice as high as functional power for several reasons [114]. First, test efficiency correlates with switching activity: obtaining high fault coverage in few test vectors requires high switching activity to excite and detect more faults. Second, test engineers use parallel SoC testing to reduce test time, which creates parallelization not typical in functional modes. Third, DFT circuitry is often idle during functional modes (which functional designers rely on) but is

active during test. Fourth, consecutive functional input vectors have a significant correlation (thus causing low switching activity), which does not apply to consecutive test vectors [115].

Various studies provided techniques to decrease LBIST power, but they either increased test time [116], [117] or increased the area overhead [118]–[120]. Girard et al. [118] used a PRPG based on cellular automata to decrease switching activity during test. Girard et al. [120] modified scan cells to suppress toggles during scan. Orno et al. [117] determined that an LFSR's seed influences energy consumption and selected seeds using a simulated-annealing algorithm. Andre et al. [119] filtered out non-detecting tests to minimize switching activity. Girard et al. [116] partitioned circuits into separately tested sub-circuits to reduce power at the cost of longer tests.

7.2.2 Power Estimation

Modeling circuit power requires balancing accuracy and computation time. Analog circuit simulators generate accurate supply voltage and current waveforms that compute power consumption accurately (presuming an accurate technology simulation model), but such simulations are computationally intensive. Therefore, most power estimations use logic-level simulation [121]. The energy consumed at a node/gate i per switch (from logic-0 to logic-1 or vice versa) is proportional to $C_i \cdot V_{DD}^2$, where C_i is the output capacitance of the node and V_{DD} is the supply voltage [122]. With a node switching activity of S_i, the average energy consumption is $E \approx C_i \cdot S_i \cdot V_{DD}^2$. The average power during test is the total energy divided by the test time T, or $P_{avg} = E/T$, and the peak power is the highest energy consumed during any clock period. This study further simplified average power as $P_{avg} \approx \sum_{i \in I} S_i$, where I is all nodes in the circuit, this estimation presumes minimal difference in output capacitance for each node and the time when comparing average power is constant. Likewise, this study estimated peak power consumption as the maximum sum of switches in any clock period.

7.2.3 Conventional and Multi-Phase TPI

Most TPI studies presumed all control points are active in a single test phase; although this may reduce switching activity (and power), this can negatively impact fault coverage. Figure 7.1 shows how an active control TP prevents signal transitions (thus reducing dynamic power), but they also prevent fault effects from propagating through them [21]. Although observe points do not create these problems, observe points cannot reduce test power since they do not influence circuit switching activity.

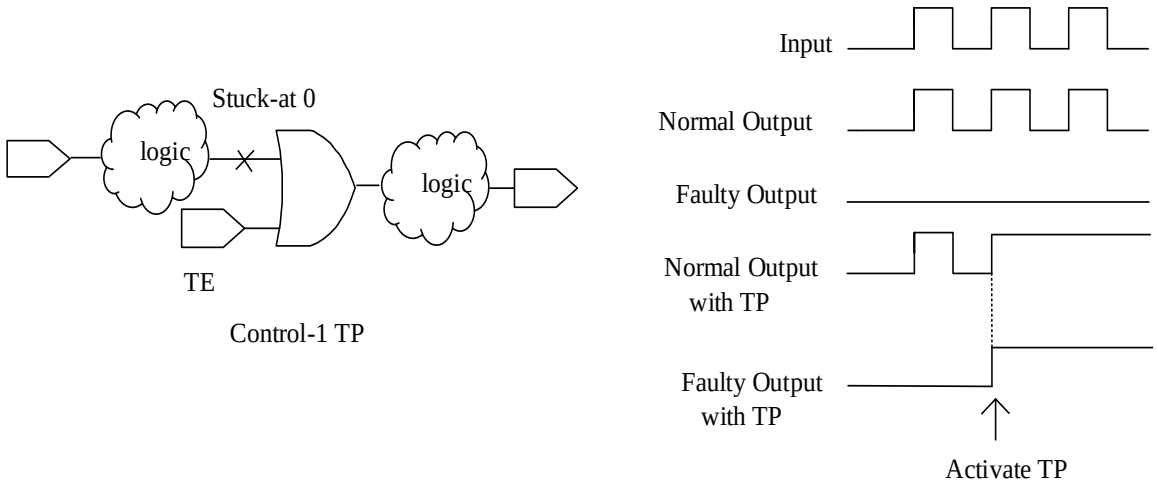


Figure 7.1: Control TPs can reduce switching activity, but also block faults.

To address this shortcoming of control TPs, *multi-phase TPI* (MPTI) [30] utilized a constructive divide and conquer approach. The test session was partitioned into multiple phases, and each phase added to the coverage obtained so far. Within each phase, TPs maximally adding to the fault coverage achieved so far were identified using a probabilistic fault simulation technique. MTPI had several advantages: first, it made easier to predict the impact of multiple control points, since in each phase a new control point was selected in the presence of control points selected so far in the phase, and in this manner, a group of control points operating synergistically was enabled

within each phase; second, power dissipation during test was potentially reduced due to the usage of fixed values of control points.

7.3 TPI for Power Reduction

7.3.1 Environment

This study presumed DFT engineers had "adequate" fault coverage, but test power was too high. This can occur if the first design iteration presents false failures during test (which designers must remedy before the final iteration) or if power analysis predicts high test power. Presuming redundant logic gates are available, designers can implement TPs through metal-mask ECOs, which doesn't apply to other power reducing DFT methods.

7.3.2 Cost Function

Like other TPI studies, this study used the COP algorithm [40] to analyze a circuit's testability, but this study also used COP to quickly estimate a circuit's power. COP generates approximate testability measures by calculating each line's controllability/CC (the probably the line is logic-0/1) and observability/CO (the probability of observing the line's value at a scannable latch or circuit output). For a circuit with n gates, calculating these values required $O(n)$ time.

The first step of evaluating TPs in the proposed TPI strategy was finding TPs decreased fault coverage: this was done using Equation 2.1 in literature [8].

The second step of TPI used COP to predict each line's switching activity. Controllability can estimate this switching activity: if the controllability of a line is 50%, this means the line is logic-0 half the time and logic-1 the other half, which implies it will switch frequently under pseudo-random stimulus. (However, it is possible that consecutive logic-0/1s are highly correlated [123] but this is left to future research.) Otherwise, if the controllability of a line is close to 100% or 0%, this means the line will stay at 1 or 0 and seldom switches under pseudo-random stimulus.

The cost function for estimating stability, ST, was based on all controllability values, where i was the number of lines in the circuit.

$$ST = \sum_0^i |CC_i - 0.5| \qquad 7.1$$

7.3.3 Power-Targeting TPI Algorithm

Like how other iterative TPI algorithms [33] used cost functions to determine the location of TPs, the proposed method used the previous two cost functions to select TPs heuristically. The TPI algorithm inserted TPs iteratively until (1) it predicted no TPs decrease power consumption, (2) the number of TPs inserted reached a pre-designated limit, (3) the algorithm reached the computation time limit, or (4) the algorithm obtains a desired power reduction.

Inserting a single TP was a two step-process. First, the TPI algorithm removed all TPs that are predicted to reduced fault coverage from the candidate TP list. Using the cost function FC in Equation 2.1, the algorithm predicted the impact each TP had on fault coverage and removed the TPs with negative impacts from the candidate list, i.e., the algorithm removed TP t with $\Delta FC_t = FC_t - FC < 0$ from the candidate TP list. To reduce future calculation times, after removing a TP, the algorithm did not reinsert removed TPs into the candidate TP list, nor did the algorithm reinsert them during future TPI phases. Second, the algorithm inserted the TP which decreased power the most. Using the second cost function ST in Equation 7.1, the algorithm calculated the stability before inserting any TP, then for all candidate TPs, the algorithm calculated the stability with the TP, ST_t. The algorithm then inserted the TP with the largest $\mathrm{ST}_t = ST_t - ST$.

7.3.4 Adding Multiple-Phase to the TPI Algorithm

As discussed in Section 7.2.3 MTPI may reduce the undesired fault coverage impacts of control TPs, therefore this study incorporated MPTI techniques [30] for power-targeting TPI. Each phase, the TPI algorithm from Section 7.3.3 inserted one or more TPs. After a phase completed,

fault simulation eliminated faults detected by the phase, and the TPI algorithm of future phases included these detected faults in ΔFC calculations. Future phases presumed previous TPs were inactive and did not impact power or fault coverage calculations. Figure 7.2 illustrates this multi-phased TPI procedure.

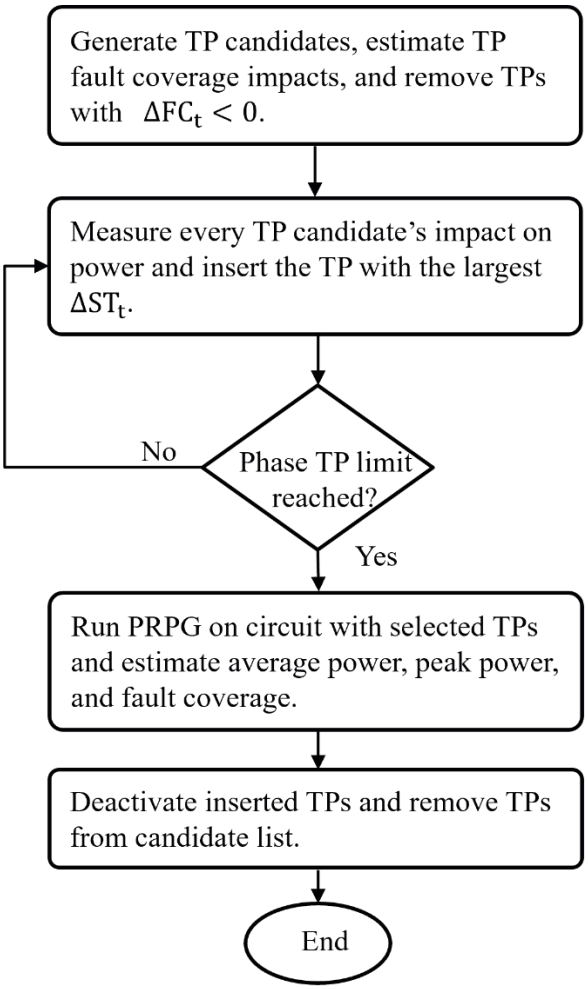


Figure 7.2: Flow chart of multi-phase TPI.

7.4 Experiment Results

7.4.1 Experimental Setup

This study performed all experiments on industry-representative workstations that perform fault simulation, power simulation, and TPI using software written explicitly for these experiments.

These workstations used Intel i7-8700 processors and possessed 8 GB of RAM to execute software written in C++ and compiled using the MSVC++14.15 compiler with maximum optimization parameters.

Experiments used post-synthesis logic netlists of the ISCAS'85 [96] and the ITC'99 [97] benchmarks (see Table 7.1), which represented a wide range of industrial circuits. Experiments tested circuits in a full-scan environment with scan chains loaded from a 64-bit PRPG. Table 7.1 gives the SAF coverage of a circuit with no TPs present under the column FC and the number of vectors that obtained this fault coverage is under the label Vec. This number of vectors was determined by (1) reaching 99% fault coverage or (2) reaching a simulation time limit of 2 hours. When performing MPTI, experiments distributed vectors evenly among each phase. (If a phase could not be evenly divided, the last phase received additional vectors.)

All circuits in Table 7.1 obtained 99% fault efficiency ($\frac{detected\ faults}{all\ faults - redudant\ fualts}$), except for *b14* and *b15*. Since *b14* and *b15*'s fault efficiency was lower than 99%, experiments added observe TPs before power-targeting TPI, which modeled designers adding observe TPs during the LBIST design process. The algorithm from [33] inserted eleven observe TPs into *b14* and sixteen into *b15*, which obtained 99% fault efficiency.

Table 7.1: Power-Targeting TPI Experimental Results

ISCAS'85					ITC'99				
Benchmarks	Gates	TPs	Vec.	SAF (%)	Benchmarks	Gates	TPs	Vec.	SAF (%)
c432	203	2	2816	98.17	b03	190	1	9280	100.00
c499	275	2	1152	98.65	b04	803	8	42240	98.52
c880	469	4	19008	100.00	b05	1032	10	19392	77.21
c1355	619	6	2880	99.51	b07	490	4	895104	99.21
c1908	938	9	12480	99.56	b08	204	2	16320	100.00
c2670	1566	15	2905600	95.84	b09	198	1	51648	100.00
c3540	1741	17	17920	95.83	b10	223	2	8448	100.00
c6288	2480	24	1472	99.30	b11	801	8	56320	95.31
c5315	2608	26	2752	99.00	b12	1197	11	194560	99.12
c7552	3827	38	2461120	97.14	b13	415	4	7040	96.04
					b14	10343	5	200832	91.03
					b15	9371	5	45376	82.88

During TPI, equations from Section 7.3.2 estimated average and peak power, but logic simulation provided final power values through true switching activity. Output capacitance C_i and power supply voltage V_{DD} were technology-dependent, thus this simulation omitted them from the energy calculation presuming (1) all nodes had the same power supply voltage (which is typical) and (2) an average load capacitance fairly represented all nodes. This calculation does not give a true power or energy, but the relative change in switching activity can compare two TPI methods.

7.4.2 Power-targeting TPI vs. Fault-targeting TPI

This first experiment performed TPI twice with a single phase of TPI; one TPI instance maximized fault coverage as the cost function while the other minimized switching activity. This experiment compared the resulting SAF coverage, average power reduction, and peak power reduction of post-TPI circuits compared to the original circuit. Each TPI instance limited the number of TPs to 1% of the total number of nodes.

Figure 7.3 plots the change in SAF coverage of the two TPI methods, which shows interesting trends. First, fault coverage always decreased after TPI. Second, using power-targeting TPI did not appear to have a clearly (un)favorable impact on SAF coverage compared to its traditional counterpart; considering all benchmark circuits, there was no clear benefit to using one method over the other to increase (which never occurred) or minimize the negative impacts on fault coverage. This last trend was counterintuitive given fault-targeting TPI was specifically trying to increase fault coverage.

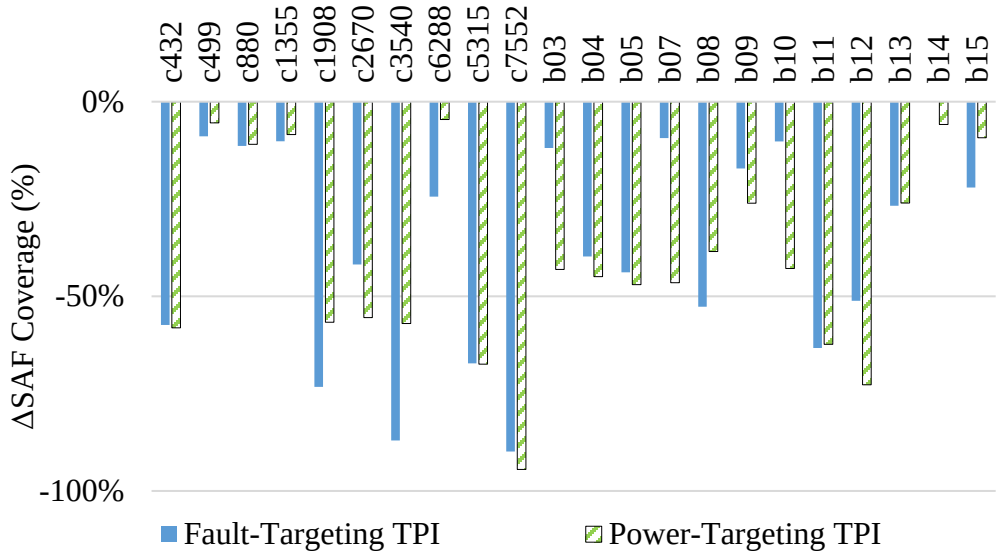


Figure 7.3: Power-targeting TPI and conventional fault-targeting TPI in SAF coverage comparison.

Figure 7.4 plots the change in switching activity (which in this experiment was analogous to dynamic power) obtained by the two TPI methods, which again shows many interesting trends. First, almost all benchmark circuits had a substantial reduction in switching activity with both TPI methods, especially *c7552* which achieved a 69% average power reduction. Second, compared to fault-targeting TPI, the power reduction of the power-targeting TPI outperformed stuck-at fault-targeting TPI for every benchmark. Given SAF trends from Figure 7.3, it appears the consistent

power-reducing benefits of power-targeting TPI did not come at the expense of further reduced fault coverage, although reduced fault coverage by both methods needs must be addressed. This experiment also observed the effect of TPI on peak power in Figure 7.5. Trends on peak power were identical to that of average power.

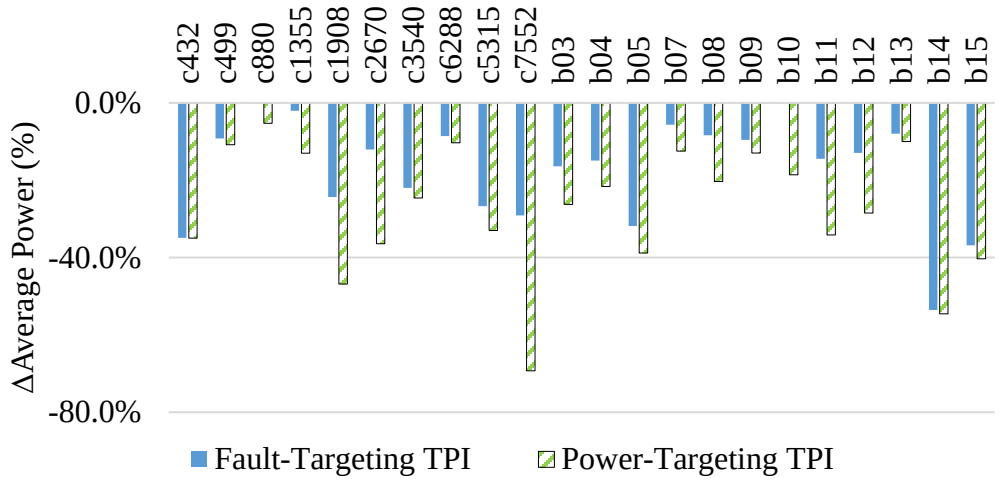


Figure 7.4: Power-targeting TPI and conventional fault-targeting TPI in average power comparison.

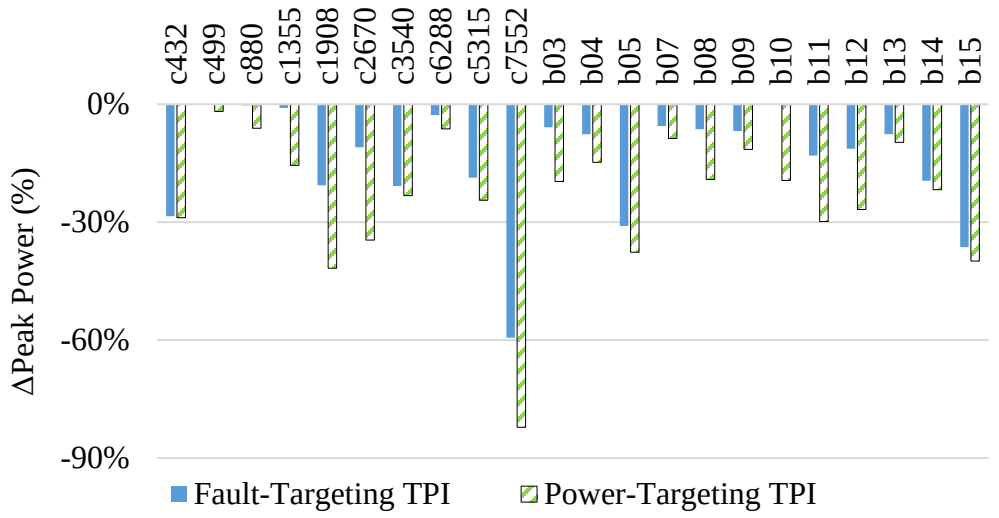


Figure 7.5: Power-targeting TPI and conventional fault-targeting TPI in peak power comparison.

From these results, power-targeting TPI clearly gave consistent improvements to average power and peak power reduction compare to fault-targeting TPI. However, the fault coverage obtained through TPI decreased dramatically, which was not acceptable for post-tape-out TPI. However, using multiple phases of TPI may remedy this.

7.4.3 Multiple Phase of TPI

This second experiment examined how multiple phases of TPI effects SAF coverage, average power, and peak power. This experiment performed TPI using two, three, and four TP phases. These TPI instances used the same limits of the first experiment and inserted the same number of TPs, but when TPI could not divide the total number of TPs evenly among phases, TPI inserted additional TPs into later phases. The first sub-set of experiments (Figure 7.6, Figure 7.7 and Figure 7.8) only observes the utility of power-targeting TPI: whether this utility held when comparing against conventional fault-targeting TPI is shown in the next sub-set of experiments (Figure 7.9, Figure 7.10 and Figure 7.11).

Figure 7.6 shows the SAF coverage of the power-targeting TPI when using multiple phases. Unlike when using a single phase of TPI, the fault coverage of using multiple phases did not always decrease, which confirmed the motivation for using multiple phases of TPI. Second, it appeared using more phases TPI further increased fault coverage, which dramatically showed itself in a few benchmarks (*c2670* and *b15*). In particular, moving from two to three phases of TPI provided the greatest consistent improvement.

Figure 7.7 shows the average power of the power-targeting TPI with multiple phases, and the impact of using more phases of TPs is clear: using more phases negated power reductions. The experiment result trends were similar in peak power, shown in Figure 7.8.

When combining the results of Figure 7.6, Figure 7.7 and Figure 7.8, it's clear that DFT engineers should attempt to address power issues with as few TPI phases as possible: adding more phases minimized undesirable impacts on fault coverage, but additional phases simultaneously negated impact on circuit power. This experiment found a "sweet spot" of three phases, but this experiment left finding whether this value had an analytical backing to future research.

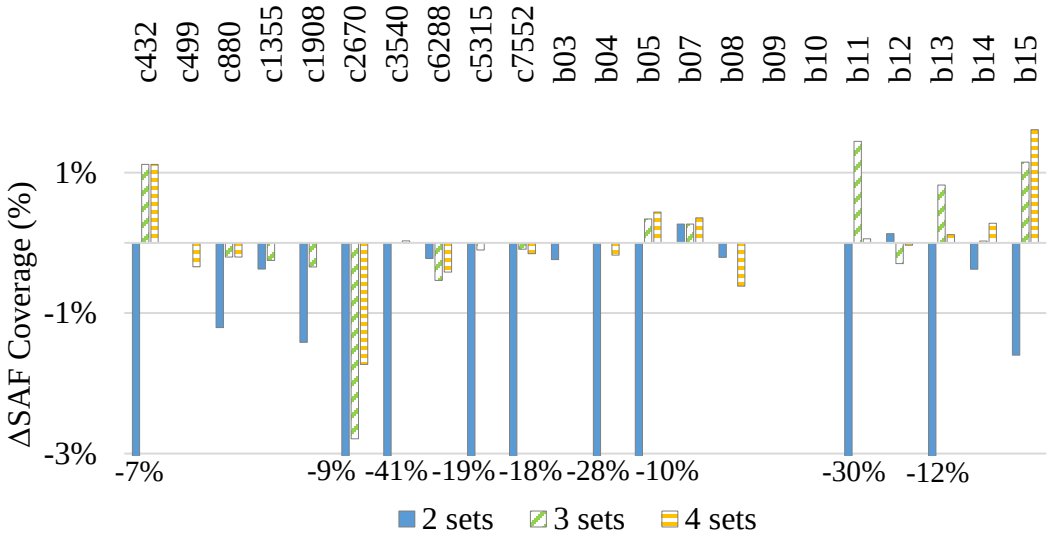


Figure 7.6: The number of phases impacts SAF coverage substantially.

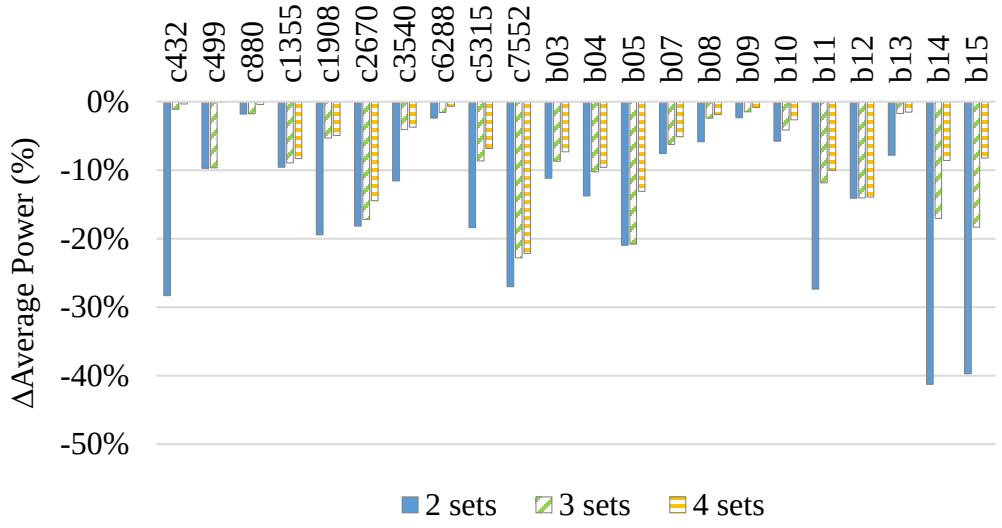


Figure 7.7: More TPI phases, the benefits to average power degrade.

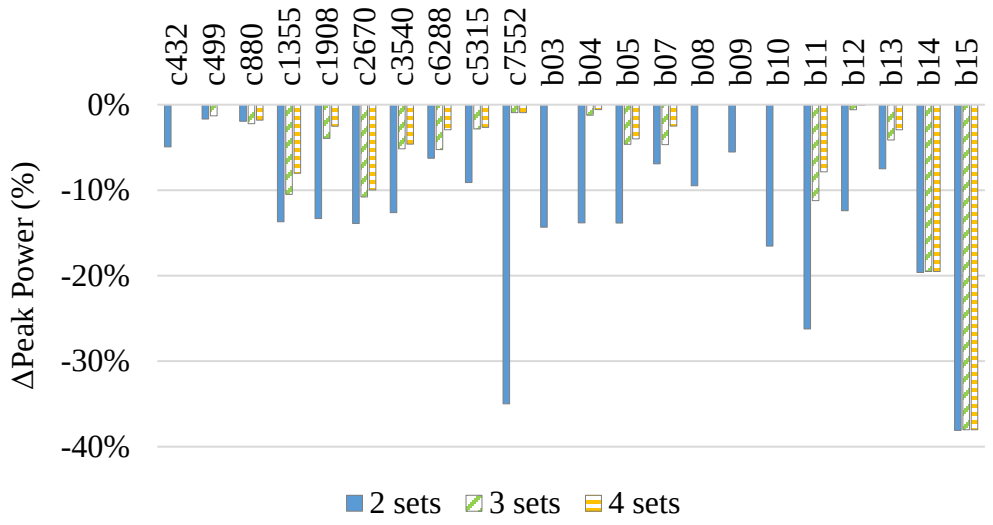


Figure 7.8: More TPI phases, the benefits to peak power degrade.

To compare fault-targeting TPI and power-targeting TPI in multiple phases, Figure 7.9 shows the SAF coverage obtained when using three phases of TPI with both power-targeting TPI and conventional fault-targeting TPI. The results were interesting: although one would expect fault-targeting TPI obtaining higher fault coverages, there was no clear trend in the data indicating this (although extreme results favor fault-targeting TPI).

However, combining the effect on average power (shown in Figure 7.10) with previous results showed a clear benefit to power-targeting TPI: despite power-targeting TPI did not perform significantly worse in terms of fault coverage, it clearly reduced power more consistently. The same trend in peak power is shown in Figure 7.11.

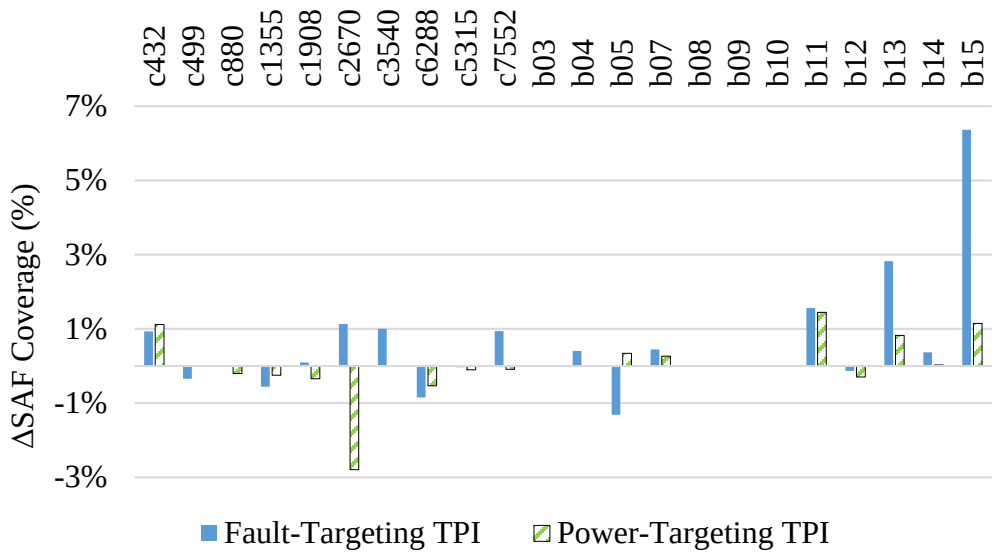


Figure 7.9: The SAF coverage of power-targeting TPI and conventional fault-targeting TPI in three phases.

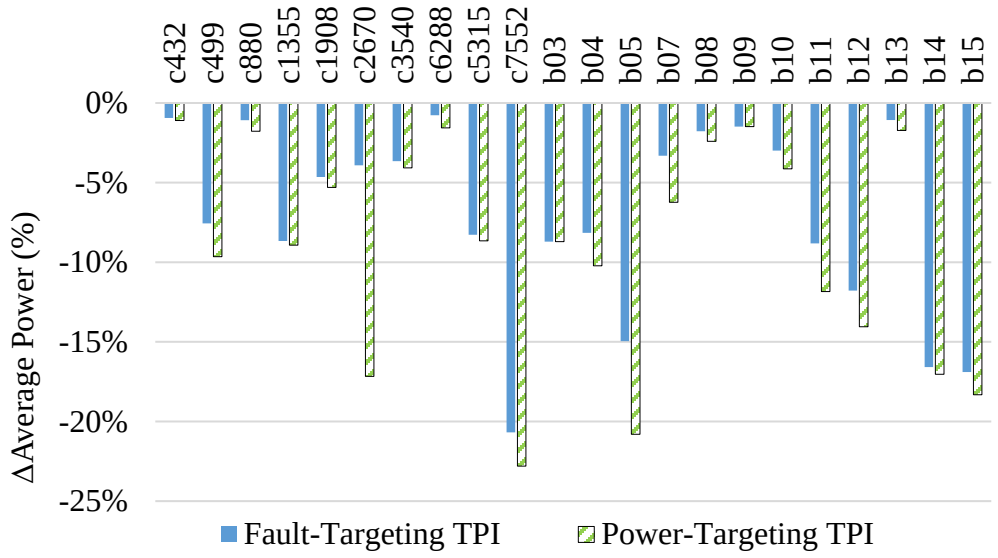


Figure 7.10: The average power of power-targeting TPI and conventional fault-targeting TPI in three phases.

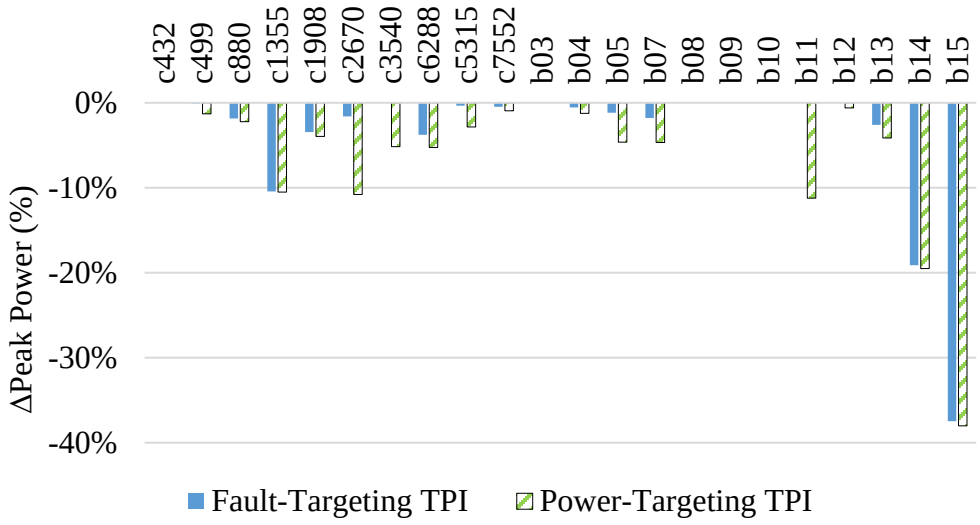


Figure 7.11: The peak power of power-targeting TPI and conventional fault-targeting TPI in three phases.

7.5 Conclusion and Future Directions

This chapter has demonstrated the effectiveness of using TPI for reducing power during test and the need to use multiple phases of TPI. This chapter has shown power-targeting TPI can effectively reduce average and peak power compared to conventional fault-targeting TPI without negatively impacting fault coverage in ECO environments, but to do so, TPI needs multiple phases and designers must resist using too many phases of TPI.

Many future research directions are left unexplored. First, studies show fault and power-estimating cost functions may be inaccurate for many circuits [99], but alternative computation methods like neural networks may achieve fast and accurate power estimations, and thus may obtain better power reductions and higher fault coverages in less computation time. Second, this chapter presumed TPI can only use control TPs, but if power reduces enough, observe TPs can "top off" power-reducing control TPs to further increase fault coverage.

Chapter 8

Conclusion and Future Work

This dissertation presented novel TPI applications that aim for fault coverage increases or power reductions. In order to increase SAT and TDF coverage by inserting TPs, ANNs were used for selecting TPs. These ANNs used circuit information as input features and were trained to predict the impact of TP quality. Once the ANN was trained, it was applied to new circuits without retraining. Compared against conventional TPI, ANN-based TPI obtained better or similar results in less time.

TPI for reducing power was based on a cost function of TP switching activity and MPTI was added to reduce negative impacts of control TP on fault coverage. Compared to fault-targeting TPI, power-targeting TPI can effectively reduce average and peak power without impacting fault coverage.

Several future directions are waiting exploration. First, an ANN can be used for TPI power reduction. Using an ANN for TPI successfully increased fault coverage, and this motivates using an ANN for power reduction. Second, TPI can use the latest machine learning algorithms; ANNs are one type of machine learning, but there are other machine learning algorithms, e.g., reinforced learning [124] and unsupervised learning [125]. As machine learning algorithms become stronger, the latest algorithms will achieve better results faster. Third, ANNs can be used for other DFT problems, like memory test; the advantage of ANNs is their ability to learn, and ANNs can find correlations of features in a circuit and predict the results. For complex DFT problems, ANN can be trained by previous solutions and find answers for new problem.

Bibliography

[1] B. L. Keller and T. J. Snethen, “Built-in Self-test Support in the IBM Engineering Design System,” *IBM J. Res. Dev.*, vol. 34, no. 2.3, pp. 406–415, Mar. 1990.

[2] P. Bardel and W. McAnney, “Self-testing of Multichip Logic Modules,” in *Proc. International Test Conference*, Philadelphia, PA, Nov. 1982, pp. 200–204.

[3] E. B. Eichelberger and E. Lindbloom, “Random-Pattern Coverage Enhancement and Diagnosis for Lssd Logic Self-Test.,” *IBM J. Res. Dev.*, vol. 27, no. 3, pp. 265–272, May 1983.

[4] I. Pomeranz and S. M. Reddy, “3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 12, no. 7, pp. 1050–1058, Jul. 1993.

[5] E. J. Mccluskey, “Built-In Self-Test Techniques,” in *IEEE Design & Test of Computers*, vol. 2, no. 2, pp. 21-28, Apr. 1985.

[6] J. P. Hayes and A. D. Friedman, “Test Point Placement to Simplify Fault Detection,” *IEEE Trans. Comput.*, vol. 100, no. 7, pp. 727–735, Jul. 1974.

[7] A. Briers and K. Totton, “Random Pattern Testability by Fast Fault Simulation,” in *Proceeding IEEE International Test Conference*, Washington, D.C., Sep. 1986, pp. 274–281.

[8] M. J. Geuzebroek, J. T. Van Der Linden, and a. J. Van De Goor, “Test Point Insertion that Facilitates ATPG in Reducing Test Time and Data Volume,” in *Proceedings. International Test Conference*, Baltimore, MD, Dec. 2002, pp. 138–147.

[9] M. J. Geuzebroek, J. T. Van Der Linden, and a. J. Van De Goor, “Test Point Insertion for Compact Test Sets,” in *Proc. International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 292–301.

[10] N. A. Touba and E. J. Mccluskey, “Test Point Insertion Based on Path Tracing,” in *Proceedings 14th VLSI Test Symposium*, Princeton, NJ, Apr. 1996, pp. 2–8.

[11] M. Chern *et al.*, “Improving Scan Chain Diagnostic Accuracy using Multi-stage Artificial Neural Networks,” in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, New York, NY, Jan. 2019, pp. 341–346.

[12] L. R. Gómez and H.-J. Wunderlich, “A Neural-network-based Fault Classifier,” in *2016 IEEE 25th Asian Test Symposium (ATS)*, Hiroshima, Japan, Nov. 2016, pp. 144–149.

[13] M. Youssef, Y. Savaria, and B. Kaminska, “Methodology for Efficiently Inserting and Condensing Test Points,” in *IEEE Proceedings (Computers and Digital Techniques)*, vol. 140, no. 3, pp. 154-160, 1993.

[14] Y. Sun and S. K. Millican, “A Survey: Test Point Insertion in LBIST,” in *IEEE VLSI Test Symposium*, San Diego, CA, Apr. 2020, pp. 1–6.

[15] J. Yang and N. A. Touba, “Test Point Insertion with Control Points Driven by Existing Functional Flip-Flops,” *IEEE Trans. Comput.*, vol. 61, no. 10, pp. 1473–1483, Oct. 2012.

[16] J. R. Fox, “Test-Point Condensation in the Diagnosis of Digital Circuits.,” *Proc. Inst. Electr. Eng.*, vol. 124, no. 2, pp. 89–94, Feb. 1977.

[17] H. Ren, M. Kusko, V. Kravets, and R. Yaari, “Low Cost Test Point Insertion without Using Extra Registers for High Performance Design,” in *International Test Conference*, Austin, TX, Nov. 2009, pp. 1–8.

[18] D. V Bakshi, M. S. Hsiao, and S. K. Shukla, “Techniques for Seed Computation and Testability Enhancement for Logic Built-In Self Test,” 2012.

[19] Y. Fang and A. Albicki, “Efficient Testability Enhancement for Combinational Circuit,” in *VLSI in Computers and Processors*, Austin, TX, Oct. 1995, pp. 168–172.

[20] P. Chan *et al.*, “An Observability Enhancement Approach for Improved Testability and At-speed Test,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 13, no. 8, pp. 1051–1056, Aug. 1994.

[21] S. Roy, B. Stiene, S. K. Millican, and V. D. Agrawal, “Improved Random Pattern Delay Fault Coverage Using Inversion Test Points,” in *IEEE 28th North Atlantic Test Workshop (NATW)*, Burlington, VT, May 2019, pp. 206–211.

[22] K. Juretus and I. Savidis, “Reducing Logic Encryption Overhead Through Gate Level Key Insertion,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, Canada, May 2016, pp. 1714–1717.

[23] H. Vranken, F. S. Sapei, and H. J. Wunderlich, “Impact of Test Point Insertion on Silicon Area and Timing during Layout,” *Proc. - Des. Autom. Test Eur. Conf. Exhib.*, Paris, France, Feb. 2004, pp. 810–815.

[24] J. Yang, B. Nadeau-dostie, N. A. Touba, M. Drive, T. Floor, and S. Jose, “Test Point Insertion Using Functional Flip-Flops to Drive Control Points,” in *International Test Conference*, Austin, TX, Nov. 2009, pp. 1–10.

[25] M. Nakao, S. Kobayashi, K. Hatayama, K. Iijima, and S. Terada, “Low Overhead Test Point Insertion for Scan-based BIST,” in *International Test Conference*, Atlantic City, NJ, Sep. 1999, pp. 348–357.

[26] F. Muradali and R. Janusz, “A Self-Driven Test Structiure for Pseudorandom Testing of Non-Scan Sequential Circuits,” in *Proc. 14th VLSI Test Symposium. IEEE,* Princeton, NJ, Apr. 1996, pp. 17–25.

[27] J. Yang, B. Nadeau-dostie, N. A. Touba, and S. Jose, “Reducing Test Point Area for BIST through Greater Use of Functional Flip-Flops to Drive Control Points,” in *24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Chicago, IL, Oct. 2009, pp. 20–28.

[28] K. Chang, J. R. Jiang, and C. J. Liu, “Reducing Test Point Overhead with Don ’ t-Cares,” in *IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boise, ID, Aug. 2012, pp. 534–537.

[29] N. Z. Basturkmen, S. M. Reddy, and J. Rajski, “Improved Algorithms for Constructive Multi-phase Test Point Insertion for Scan Based BIST,” in *Asia and South Pacific Design Automation Conference*, Bangalore, India, Jan. 2002, pp. 604–611.

[30] N. Tamarapalli and J. Rajski., “Constructive Multi-phase Test Point Insertion for Scan-based BIST,” in *Proceedings International Test Conference. Test and Design Validity*, Washington, DC, Oct. 1996, pp. 649–658.

[31] B. Krishnamurthy and T. Laboratories, “A Dynamic Programming Approach to the Test Point Insertion Problem,” in *Proc. the 24th ACM/IEEE Design Automation Conference*, Miami Beach, FL, Jun. 1987, pp. 695–705.

[32] J. Sziray, “Test Generation and Computational Complexity,” *Proc. IEEE Pac. Rim Int. Symp. Dependable Comput. PRDC,* Pasadena, CA, Dec. 2011, pp. 286–287.

[33] H.-C. Tsai, C.-J. Lin, S. Bhawmik, and K.-T. Cheng, “A Hybrid Algorithm for Test Point Selection for Scan-based BIST,” in *Proceeding the 34th annual conference on Design automation conference*, Anaheim, CA, Jun. 1997, pp. 478–483.

[34] P. Nigh and A. Gattiker, “Test Method Evaluation Experiments and Data,” in *Proceedings International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 454–463.

[35] Y. Ma *et al.*, “High Performance Graph Convolutional Networks with Applications in Testability Analysis,” in *Proceedings of the 56th Annual Design Automation Conference*, Las Vegas, NV, Jun.2019, pp. 1–6.

[36] V. S. Iyngar and D. Brand, “Synthesis of Pseudo-Random Pattern Testable Designs,” in *International Test Conference*, Washington, DC, Aug. 1989, pp. 501–508.

[37] T. Ramakrishnan and L. Kinney, “Extension of the Critical Path Tracing Algorithm,” in *27th ACM/IEEE Design Automation Conference*, Orlando, FL, Jun. 1990, pp. 720–723.

[38] P. Menon, Y. Levendel, and M. Abramovici, “SCRIPT: A Critical Path Tracing Algorithm for Synchronous Sequential Circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, no. 6, pp. 738–747, Jun. 1991.

[39] L. H. Goldstein and E. L. Thigpen, “SCOAP: Sandia Controllability/Observability Analysis Program,” in *17th Design Automation Conference*, Minneapolis, MN, Jun. 1980, pp. 190–196.

[40] F. Brglez, “On Testability Analysis of Combinational Networks,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, Jan. 1984, vol. 1, pp. 705–712.

[41] H. Tsai, S. Member, and K. T. Cheng, “Efficient Test-Point Selection for Scan-Based BIST,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 6, no. 4, pp. 667–676, Dec. 1998.

[42] R. Lisanke, F. Brglez, A. J. Degeus, and D. Gregory, “Testability-Driven Random Test-Pattern Generation,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 6, no. 6, pp. 1082–1087, Nov. 1987.

[43] P. Bist, C. Lin, and T. B. Labs, “Timing-Driven Test Point Insertion for Full-Scan and Partical-Scan BIST,” in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1995, pp. 506–514.

[44] M. Nakao and K. Hatayama, “Accelerated Test Points Selection Method for Scan-Based BIST,” *Proc Sixth Asian Test Symp.*, vol. 2, pp. 359–364, Nov. 1997.

[45] B.H.Seiss, “Test Point Insertion for Scan-Based BIST,” in *Proceedings European Test Conference*,Munich, German, Apr. 1991, pp. 253–262.

[46] Y. Savaria, M. Youssef, B. Kaminska, and M. Koudil, “Automatic Test Point Insertion for Pseudo-random Testing,” in *IEEE International Sympoisum on Circuits and Systems*, Singapore, Jun. 1991, pp. 1960–1963.

[47] M. He, G. K. Contreras, M. Tehranipoor, D. Tran, and L. R. Winemberg, “Test-point Insertion Efficiency Analysis for LBIST Applications,” *Proc. IEEE VLSI Test Symp.*, Las Vegas, NV, Apr. 2016, pp. 1–6.

[48] M. He *et al.*, “Test-Point Insertion Efficiency Analysis for LBIST in High-Assurance Applications,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 25, no. 9, pp. 2602–2615, Jun. 2017.

[49] M. J. Chen and D. Xiang, “Pseudorandom Scan BIST using Improved Test Point Insertion Techniques,” in *International Conference on Solid-State and Integrated Circuits Technology Proceedings (ICSICT)*, Beijing, China, Oct. 2004, pp. 2043–2046.

[50] R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell, “Zero Cost Test Point Insertion Technique to Reduce Test Set Size and Test Generation Time for Structured ASICs,” in *15th Asian Test Symposium,* Fukuoka, Japan, Nov. 2006, pp. 339–348.

[51] C. Acero *et al.*, “Embedded Deterministic Test Points,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 25, no. 10, pp. 2949–2961, Jul. 2017.

[52] E. Moghaddam, N. Mukherjee, J. Rajski, J. Solecki, J. Tyszer, and J. Zawada, “Logic BIST with Capture-Per-Clock Hybrid Test Points,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1028–1041, May 2019.

[53] G. L. Smith and P. Box, “Model for Delay Faults Based upon Paths,” in *Proc. IEEE International Test Conference-(ITC),* Philadelphia, PA, Nov. 1985, pp. 342–351.

[54] I. Pomeranz, S. M. Reddy, and I. City, “An Efficient Non-Enumerative Method to Estimate Path Delay Fault Coverage,” in *IEEE/ACM International Conference on Computer-Aided Design,* Santa Clara, CA, Nov. 1992, pp. 560–567.

[55] I. Pomeranz and S. M. Reddy, “On the Number of Tests to Detect all Path Delay Faults in Combinational Logic Circuits,” *IEEE Trans. Comput.*, vol. 45, no. 1, pp. 50–62, Jan. 1996.

[56] C. J. Lin and S. M. Reddy, “On Delay Fault Testing in Logic Circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 6, no. 5, pp. 694–703, Sep. 1987.

[57] S. Tragoudas and N. Denny, “Testing for Path Delay Faults Using Test Points,” in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.*, Albuquerque, NM, Nov. 1999, pp. 86–94.

[58] I. Pomeranz, S. Member, and S. M. Reddy, “Design-for-Testability for Path Delay Faults in Large Combinational Circuits Using Test Points,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 4, pp. 333–343, Apr. 1998.

[59] P. Uppduri, U. Sparmann, and I. Pomeranz, “On Minimizing the Number of Test Points Needed to Achieve Complete Robust Path Delay Fault Testability,” in *Proc. 14th VLSI Test Symposium. IEEE,* Princeton, NJ, Apr. 1996, pp. 288–295.

[60] U. Sparmann, D. Luxenburger, K. Cheng, and S. M. Reddy, “Fast Identification of Robust Dependent Path Delay Faults,” in *32nd Design Automation Conference*, San Francisco, CA, Jun. 1995, pp. 119–125.

[61] R. Sankaralingam and N. Touba, “Inserting Test Points to Control Peak Power During Scan Testing,” in *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.*, Vancouver, Canada, Nov. 2002, pp. 138–146.

[62] S. Gerstendörfer and H.-J. Wunderlich, “Minimized Power Consumption For Scan-Based Bist,” in *IEEE International Test Conference*, Atlantic City, NJ, Sep. 1999, pp. 203–212.

[63] E. G. Ulrich, V. D. Agrawal, and J. H. Arabian, *Concurrent and Comparative Discrete Event Simulation*. Springer Science & Business Media, 1993.

[64] J. A. Tofte and H. Rahmanian, “An Effort-Minimized Logic BIST Implementation Method,” in *Proc. International Test Conference*, Baltimore, MD, Nov. 2001, pp. 1002–1010.

[65] S. Roy, B. Laboratories, and K. T. Cheng, “Efficient Test Mode Selection & Insertion for RTL-BIST,” in *Proc. International Test Conference,* Atlantic City, NJ, Oct. 2000, pp. 263–272.

[66] H. Vranken, F. Meister, and H. Wunderlich, “Combining Deterministic Logic BIST with Test Point Insertion,” in *Proc. The Seventh IEEE European Test Workshop,* Corfu, Greece, May 2002, pp. 105–110.

[67] N. Gupta, “Artificial neural network,” *Netw. Complex Syst.*, vol. 3, no. 1, pp. 24–28, 2013.

[68] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. E. Mohamed, and H. Arshad, "State-of-the-art in Artificial Neural Network Applications: A Survey," *Heliyon*, vol. 4, no. 11. Elsevier Ltd, p. e00938, Nov. 2018.

[69] W. S. Sarle, "Neural Networks and Statistical Models," 1994.

[70] W. McCulloch and W. Pitts, *Automata Studies*. 1956.

[71] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. Taylor & Francis, 2005.

[72] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

[73] S. Dreyfus, "The Computational Solution of Optimal Control Problems with Time Lag," *Trans. Autom. Control*, vol. 18, no. 4, pp. 383–385, Aug. 1973.

[74] T. S. Huang. Weng, John Juyang Narendra Ahuja, "Learning Recognition and Segmentation using the Cresceptron," *Int. J. Comput. Vis.*, vol. 25, no. 2, pp. 109–143, Nov. 1997.

[75] A. Zell, *Simulation Neuronaler Netze*. 1994.

[76] P. Ramachandran and B. Zoph, "Searching for Activation Functions," *CORR*, 2017.

[77] Z. Huang, H. Chen, C.-J. Hsu, W.-H. Chen, and S. Wu, "Credit Rating Analysis with Support Vector Machines and Neural Networks: a Market Comparative Study," *Decis. Support Syst.*, vol. 37, no. 4, pp. 543–558, Sep. 2004.

[78] J. M. Binner, C. T. Elger, B. Nilsson, and J. A. Tepper, "Predictable Non-linearities in U.S. Inflation," *Econ. Lett.*, vol. 93, no. 3, pp. 323–328, Sep. 2006.

[79] F. A. de Oliveira, C. N. Nobre, and L. E. Zárate, "Applying Artificial Neural Networks to Prediction of Stock Price and Improvement of the Directional Prediction Index – Case

Study of PETR4, Petrobras, Brazil," *Expert Syst. Appl.*, vol. 40, no. 18, pp. 7596–7606, Dec. 2013.

[80] Z. F. Wang, J.-L. Zarader, and S. Argentieri, "Aircraft Fault Diagnosis and Decision System based on Improved Artificial Neural Networks," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Kaohsiung, Taiwan, Jul. 2012, pp. 1123–1128.

[81] X. Lan, C. Qin, Y. Liu, H. Ouyang, and G. Liu, "Intelligent Guidance of Autonomous Mobile Robots based on Adaptive Dynamic Programming," in *34rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Jinzhou, China, Jun. 2019, pp. 695–699.

[82] L. Jin, S. Li, J. Yu, and J. He, "Robot Manipulator Control using Neural Networks: A Survey," *Neurocomputing*, vol. 285, pp. 23–34, Apr. 2018.

[83] A. Alghoul, S. Al Ajrami, G. Al Jarousha, G. Harb, and S. S. Abu-Naser, "Email Classification using Artificial Neural Network," *International Journal of Academic Engineering Research (IJAER)*, vol. 2, no. 11, pp. 8–14, Dec. 2018.

[84] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural Architectures for Named Entity Recognition," *ArXiv Prepr. ArXiv160301360*, May 2016.

[85] K. Wołk and K. Marasek, "Neural-based Machine Translation for Medical Text Domain. Based on European Medicines Agency Leaflet Texts," *Procedia Comput. Sci.*, vol. 64, pp. 2–9, Oct. 2015.

[86] E. Sipos and L.-N. Ivanciu, "Failure Analysis and Prediction using Neural Networks in the Chip Manufacturing Process," in *40th International Spring Seminar on Electronics Technology (ISSE)*, Sofia, Bulgaria, May 2017, pp. 1–5.

[87] D. Guerra, A. Canelas, R. Póvoa, N. Horta, N. Lourenço, and R. Martins, “Artificial Neural Networks as an Alternative for Automatic Analog IC Placement,” in *16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, Lausanne, Switzerland, Jul. 2019, pp. 1–4.

[88] J. P. Janet and H. J. Kulik, “Predicting Electronic Structure Properties of Transition Metal Complexes with Neural Networks,” *Chem. Sci.*, vol. 8, no. 7, pp. 5137–5152, 2017.

[89] S. Roy, S. K. Millican, and V. D. Agrawal, “Unsupervised Learning in Test Generation for Digital Integrated Circuits,” 2021.

[90] S. Roy, S. K. Millican, and V. D. Agrawal, “Machine Intelligence for Efficient Test Pattern Generation,” in *IEEE International Test Conference (ITC)*, Washington, DC, Nov. 2020, pp. 1–5.

[91] Y. Sun and S. Millican, “Test Point Insertion Using Artificial Neural Networks,” in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Miami, FL, Jul. 2019, pp. 253–258.

[92] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, vol. 17. Springer Science & Business Media, 2004.

[93] S. Ghosh, S. Bhunia, A. Raychowdhury, and K. Roy, “A Novel Delay Fault Testing Methodology using Low-overhead Built-in Delay Sensor,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2934–2943, Nov. 2006.

[94] S. Haykin, *Neural Networks and Learning Machines*. Pearson Education India, 2010.

[95] H. Choset and P. Pignon, “Coverage Path Planning: The Boustrophedon Cellular Decomposition BT - Field and Service Robotics,” 1998, pp. 203–209.

[96] F. Brglez and H. Fujiwara, “A Neural Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran,” in *Proc. of International Symposium on Circuits and Systems*, KYOTO, Japan, Jun. 1985, pp. 669.

[97] F. Corno, M. S. Reorda, and G. Squillero, “RT-level ITC’99 Benchmarks and first ATPG Results,” *IEEE Des. Test Comput.*, vol. 17, no. 3, pp. 44–53, Jul. 2000.

[98] D. P. Kingma and J. L. Ba, “ADAM: a Method for Stochastic Optimization,” *ArXiv Prepr. ArXiv14126980*, Dec. 2015.

[99] 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 28th Asian Test Symposium (ATS)*, Kolkata, India, Dec. 2019, pp. 13–15.

[100] O. Bula, J. Moser, J. Trinko, M. Weissman, and F. Woytowich, “Gross Delay Defect Evaluation for a CMOS Logic Design System Product,” *IBM J. Res. Dev.*, vol. 34, no. 2.3, pp. 325–338, Mar. 1990.

[101] P. C. Maxwell, R. C. Aitken, V. Johansen, and I. Chiang, “The Effectiveness of IDDQ, Functional and Scan Tests: How Many Fault Coverages Do We Need?,” in *Proceedings of the IEEE International Test Conference on Discover the New World of Test and Design*, Washington, DC, Sep. 1992, pp. 168–177.

[102] A. Krstic and K.-T. T. Cheng, *Delay Fault Testing for VLSI Circuits*, Springer Science & Business Media, 2012.

[103] J. Mahmod, S. Millican, U. Guin, and V. Agrawal, “Special Session: Delay Fault Testing-present and Future,” in *IEEE 37th VLSI Test Symposium (VTS)*, Monterey, CA, Apr. 2019, pp. 1–10.

[104] A. Menon, K. Mehrotra, C. K. Mohan, and S. Ranka, “Characterization of a Class of Sigmoid Functions with Applications to Neural Networks,” *Neural Netw.*, vol. 9, no. 5, pp. 819–835, Jul. 1996.

[105] T. W. Williams, “Test Length in a Self-testing Environment,” *IEEE Des. Test Comput.*, vol. 2, no. 2, pp. 59–63, Apr. 1985.

[106] G. N. Karystinos and D. A. Pados, “On Overfitting, Generalization, and Randomly Expanded Training Sets,” *IEEE Trans. Neural Netw.*, vol. 11, no. 5, pp. 1050–1057, Sep. 2000.

[107] J. Savir, “Skewed-load Transition Test: Part I, calculus,” in *Proceedings International Test Conference*, Baltimore, MD, Sep. 1992, pp. 705.

[108] C. H. Small, “Shrinking Devices Put the Squeeze on System Packaging,” *EDN*, vol. 39, no. 4, pp. 41–54, 1994.

[109] J. Lee and N. A. Touba, “LFSR-Reseeding Scheme Achieving Low-Power Dissipation During Test,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 396–401, Jan. 2007.

[110] P. Girard, “Survey of Low-Power Testing of VLSI Circuits,” *IEEE Des. Test Comput.*, vol. 19, no. 3, pp. 82–92, Aug. 2002.

[111] J. Monzel *et al.*, “Power Dissipation during Testing: Should We Worry About it?,” in *Panel Session, IEEE VLSI Test Symposium*, Hyderabad, India, Jan. 1997, pp. 456–457.

[112] G. Stefan and W. Hans-Joachim, “Minimized Power Consumption for Scan-based BIST,” *J. Electron. Test.*, vol. 16, no. 3, pp. 203–212, Jun. 2000.

[113] K.-H. Ho, J.-H. R. Jiang, and Y.-W. Chang, "TRECO: Dynamic Technology Remapping for Timing Engineering Change Orders," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 11, pp. 1723–1733, Oct. 2012.

[114] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," in *Digest of Papers Eleventh Annual IEEE VLSI Test Symposium*, Atlantic City, NJ, Apr. 1993, pp. 4–9.

[115] S. Wang and S. K. Gupta, "DS-LFSR : A New BIST TPG for Low Heat Dissipation," in *Proceedings International Test Conference*, Washington, DC, Nov. 1997, pp. 848–857.

[116] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, and M. C. France, "Low Power BIST Design by Hypergraph Partitioning : Methodology and Architectures," in *Proceedings International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 652–661.

[117] F. C. Orno, M. R. Ebaudengo, M. S. O. R. Eorda, G. S. Quillero, M. V Iolante, and P. Torino, "Low Power BIST via Non-Linear Hybrid Cellular Automata," in *Proceedings 18th IEEE VLSI Test Symposium.*, Montreal, Canada, Apr. 2000, pp. 29–34.

[118] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, and M. C. France, "A Test Vector Inhibiting Technique for Low Energy BIST Design," in *Proceedings 17th IEEE VLSI Test Symposium*, Dana Point, CA, Apr. 1999, pp. 407–412.

[119] H. Andre and W. Hans-Joachim, "Low Power Serial Built-In Self-Test," in *Proceedings 3rd European Test Workshop, IEEE*, 1998, pp. 49–53.

[120] P. Girard *et al.*, "Low-Energy BIST Design : Impact of the LFSR TPG Parameters on the Weighted Switching Activity," in *Proceedings the IEEE International Symposium on Circuits and Systems VLSI*, Orlando, FL, May 1999, pp. 110–113.

[121] R. Tjarnstrom, “Power Dissipation Estimate by Switch Level Simulation,” in *Proceedings IEEE International Symposium on Circuits and Systems*, Portland, OR, May 1989, pp. 881–884.

[122] M. A. Cirit, “Estimating Dynamic Power Consumption of CMOS Circuits,” in *Proceedings ICCAD*, Sep. 1987, pp. 534–537.

[123] F. Rashid and V. Agrawal, “Weighted Random and Transition Density Patterns for Scan-BIST,” *IEEE NATW*, Woburn, MA, May 2012.

[124] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[125] N. Grira, M. Crucianu, and N. Boujemaa, “Unsupervised and Semi-supervised Clustering: a Brief Survey,” *Rev. Mach. Learn. Tech. Process. Multimed. Content*, vol. 1, pp. 9–16, Aug. 2004.