Optimization of Fault-Insertion Test and Diagnosis of Functional Failures

by

Zhaobo Zhang

Department of Electrical and Computer Engineering
Duke University

Date: ______________________________
Approved:

____
Krishnendu Chakrabarty, Supervisor

____
Hisham Z. Massoud

____
Martin Brooke

____
Patrick Wolf

____
Xinli Gu

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the Graduate School of
Duke University

2011

ABSTRACT

(Engineering—Computer)

Optimization of Fault-Insertion Test and Diagnosis of Functional Failures

by

Zhaobo Zhang

Department of Electrical and Computer Engineering
Duke University

Date: ______________________
Approved:

____
Krishnendu Chakrabarty, Supervisor

____
Hisham Z. Massoud

____
Martin Brooke

____
Patrick Wolf

____
Xinli Gu

An abstract of a dissertation submitted in partial
fulfillment of the requirements for the degree
of Doctor of Philosophy in the Department of
Electrical and Computer Engineering in the Graduate School of
Duke University

2011

Abstract

Advances in semiconductor technology and design automation methods have introduced a new era for electronic products. With design sizes in millions of logic gates and operating frequencies in GHz, defects-per-million rates continue to increase, and defects are manifesting themselves in subtle ways. Traditional test methods are not sufficient to guarantee product quality and diagnostic programs cannot rapidly locate the root cause of failure in large systems. Therefore, there is a need for efficient fault diagnosis methods that can provide quality assurance, accelerate new product release, reduce manufacturing cost, and increase product yield.

This thesis research is focused on fault-insertion test (FIT) and fault diagnosis at the board and system levels. FIT is a promising technique to evaluate system reliability and facilitate fault diagnosis. The error-handling mechanism and system reliability can be assessed in the presence of intentionally inserted faults, and artificial faulty scenarios can be used as references for fault diagnosis. However, FIT needs to be deployed under constraints of silicon area, design effort, availability of equipment, and what is actually possible to test from one design to the next. In this research, physical defect modeling is developed to provide an efficient solution for fault-insertion test. Artificial faults at the pin level are created to represent physical defects inside devices. One pin-level fault is able to mimic the erroneous behaviors caused by multiple internal defects. Therefore, system reliability can be evaluated in a more efficient way.

Fault diagnosis is a major concern in the semiconductor industry. As the density and complexity of systems increase relentlessly and the subtle effects of defects in nanometer technologies become more pronounced, fault diagnosis becomes difficult, time-consuming, and ineffective. Diagnosis of functional failure is especially challenging. Moreover, the cost associated with board-level diagnosis is escalating rapidly. Therefore, this thesis presents a multi-pronged approach to improve the efficiency and accuracy of fault diagnosis, including the construction of a diagnostic framework with FIT and Bayesian inference, the extraction of an effective fault syndrome (error flow), the selection of diagnosis-oriented fault-insertion points, and the application of machine learning for intelligent diagnosis.

First, in the inference-based diagnosis framework, FIT is used to create a large number of faulty samples and derive the probabilities needed for the application of Bayes' theorem; next the probability of a fault candidate being the root cause can be inferred based on the given fault syndromes. Results on a case study using an open-source RISC system-on-chip demonstrate the feasibility and effectiveness of the proposed approach. Second, the concept of error flow is proposed to mimic actual data propagation in a circuit, and thus it reflects the logic functionality and timing behavior of circuits. With this additional information, more fault syndromes are distinguishable. Third, diagnosis-oriented fault-insertion points are defined and selected to create the representative and distinguishable syndromes. Finally, machine learning approaches are used to facilitate the debug and repair process. Without requiring the need to understand the complex functionality of the boards, an intelligent diagnostic system is designed to automatically exploit the diagnostic knowledge available from past cases and make decisions on new cases.

In summary, this research has investigated efficient means to perform fault-insertion test and developed automated and intelligent diagnosis methods targeting functional failures at the board level. For a complex circuit board currently in production, the first-time success rate for diagnosis has been increased from 35.63% to 72.64%. It is expected to contribute to quality assurance, product release acceleration, and manufacturing-cost reduction in the semiconductor industry.

Acknowledgements

Foremost, I would like to express my deep gratitude to my advisor, Prof. Krishnendu Chakrabarty, for his guidance, encouragement and support throughout my Ph.D. pursuit. Without his continuous and extensive support, this thesis would not have been possible. He not only brought me to the VLSI testing realm, but also enlightened me with the proper way of transforming preliminary ideas into practical solutions through independent thinking. His insistence on perfection and high-quality work has brought out the best in me. The knowledge and the way of solving problems I learned from him will benefit all my research life.

I would like to thank to the rest of my thesis committee: Prof. Hisham Z. Massoud, Prof. Martin Brooke, Prof. Patrick Wolf, and Dr. Xinli Gu for their time, advice and encouragement in this work. This work cannot go this far without their guidance and constructive feedback. My special thanks go to Dr. Xinli Gu, who provided immense contributions to this thesis research. He broadened my vision on the practical issues in manufacturing test during three times internship. My memorable experience as an intern at Cisco Systems Inc. is a key contributor to this thesis. I thank my mentor Dr. Zhanglei Wang for sharing his valuable thoughts with me.

I thank my peers, Hongxia Fang, Yang Zhao, Brandon Noia, Ender Yilmaz, Erdem S. Erdogan, Erkan Acar, and Fang Liu, for beneficial discussions and for helping me in my research work.

I thank Dr. Sule Ozev for all her support and guidance in the first year of my graduate study. Her patience and inspirations triggered my initial interests in the testing field.

I thank ECE staff, specifically Samantha Morton, Ellen Currin, Kristen Rogers,

and Autumn Wenner for their instant help when I needed it.

Financial support received from Cisco Systems Inc. and Semiconductor Research Corporation is gratefully acknowledged.

Finally, I express my deepest gratitude and appreciation to my parents, who have always being there with me to share my pains and joys. I gratefully dedicate this dissertation to them for their love, care, and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With continued technology scaling and increased market competition, the performance and complexity of electronic products have increased by orders of magnitude in the past few decades. Advances in semiconductor technology have motivated engineers to achieve higher levels of integration with shorter development cycles. A complex system today consists of several chassis, and each of them contains several printed circuit boards (PCBs). A typical PCB consists of many application-specific integrated circuits (ASICs) and memory devices. Each ASIC consists of hundreds of inputs/outputs (I/Os), millions of logic gates, and several tens of millions of bits of embedded memory. Moreover, the data rates of high-speed I/Os are up to tens of Gbps, and the operating frequencies of microprocessors are several GHz [6]. These advances in chip/system design are placing immense pressure on manufacturing test and fault diagnosis.

In this chapter, we introduce the basic concepts and techniques in manufacturing test and diagnosis. Fault models, testing techniques, and diagnostic methods are first described. The challenges of board-level diagnosis and the motivation of this research are next presented. Finally, an outline of this thesis is provided.

1.1 Overview of Manufacturing Test

Manufacturing test ensures that electronic products have no manufacturing defects before they are shipped to customers. Each fabricated product is subject to manufacturing test. A defect is a physical imperfection caused in manufacturing, e.g.

shorts and opens. It is also referred to as a manufacturing defect or a physical defect. In an electronic system, it is an unintended difference between the implemented hardware and its intended design. To represent a defect at an abstract functional level, fault models are used. Various fault models are used to represent different level of defects. Here we focus on logical fault models used for gate-level circuits [7].

Stuck-at fault model: This is the most common fault model. It assumes that every faulty net is permanently set to either logic '0' or logic '1', corresponding to a stuck-at-0 or stuck-at-1 fault, respectively. A test set based on the stuck-at fault model has been shown to be effective for detecting many defects during manufacturing test. Stuck-at fault test is also referred to as DC test in industry.

Transition fault model: The transition fault model is an example of a delay fault model. Many defects in nanometer technologies cause timing problems. The transition fault model was proposed to model timing defects. It is based on the assumption that a delay fault affects only one gate in the circuit and it causes the combinational delay of the circuit to exceed the clock period. Each gate can be affected by two possible transition faults: slow-to-rise and slow-to-fall. The design tools used for generating tests for stuck-at faults can be easily modified to generate tests for transition faults.

Bridging fault model: A bridging fault represents a short circuit among a group of signals. Two main types of bridging faults are OR-type bridging and AND-type bridging. In OR-type bridging, the logic value of the shorted net is 1-dominant, while the logic value of the shorted net is 0-dominant in AND-type bridging.

Flip fault model: The flip fault is used to model the subtle effects of transient failures that are not easily detected by traditional design-for-testability (DFT) features. It assumes that the logic value of the faulty net is switched to the opposite of the fault-free value. The faulty value is not permanently associated with the faulty net. The correct value appears on the net after some period of time. In order to analyze the subtle effects of failures on modern boards, we consider the flip fault model in our research.

Given a circuit under test (CUT) and a test set, fault coverage is defined as the ratio of the number of faults detected by the test set to the number of all possible faults in the CUT. The fault coverage is a convenient metric for evaluating the effectiveness of a given set of test patterns. Manufacturing test must cover a high proportion of modeled faults to ensure product quality. The fault coverage of stuck-at fault and transition fault is widely used in industry to evaluate test quality. Single stuck-at fault coverage is typically required to be greater than 98%. The coverage requirement for transition faults is less stringent.

Testing strategies vary across the stage of system assembly. In order to better understand the problems and challenges at the board level, let us first go through chip-level testing. At the chip level, scan test is the most common method. In order to shift test patterns in and out, all the flip-flops are stitched to one or more chains, a structure referred to as scan chain. Test patterns are generated by automatic pattern generation (ATPG) tools and scanned into the circuit under test. The response of the circuit is compared with the expected response. The circuit is considered fault-free if the test response matches the expected response.

This testing process can be performed on automatic test equipment (ATE). Modern ATE is a powerful piece of equipment operating at GHz frequency and with high throughput. The testing cost on a high-end ATE can be up to thousands of dollars per pin [7]. Therefore, test data compression techniques have been widely explored in the literature, in order to reduce the volume of test patterns and testing time. To reduce the dependence of the expensive ATE, built-in-self-test

(BIST) techniques have also been used. In BIST, pseudo-random patterns are generated on-chip using a pattern generation circuit, e.g., a linear-feedback shift register. The on-chip pattern generation approach eliminates the need for expensive external testers, and makes high-speed test possible. However, the reliance on pseudorandom patterns to achieve adequate fault coverage leads to a large volume of test data, compared with the use of deterministic patterns generated by ATPG tools.

Chip-level test is a widely-studied and mature topic. The testing goal is straightforward and quantifiable, i.e., to detect a high percentage of defects that are introduced during manufacturing. The test environment (ATE) is well-understood and a large number of pins and ports are available for probing. Sophisticated tools and methodologies are available for inserting effective DFT structures and for automatically generating test patterns to get the desired fault coverage [8]. In this thesis, we focus on the testing and diagnostic problems at higher integration levels. Much less attention has been devoted in the literature to these problems.

Circuit boards consist of previously-tested components. An important objective of board testing is to verify the printed wiring and the contacts between wires and components. A typical manufacturing test line of electronic systems is shown in Fig. 1.1. Testing starts on the left-hand side of the figure. On the right-hand side, completely tested products are placed in inventory or shipped to customers. The testing process is separated into multiple stages to provide better failure isolation and feedback to the manufacturing process.

Process test, such as automated optical inspection (AOI) and automated X-ray inspection (AXI), is first applied to immediately catch process flaws, e.g., solder shorts, unreliable solder joints. In-circuit test is often used to verify the performance of individual components using a bed-of-nails fixture. The bed-of-nails

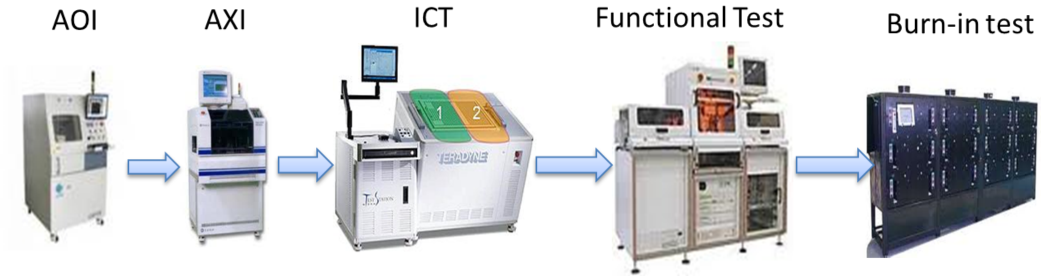


Figure 1.1: A typical manufacturing test line.

fixture is used to gain access to necessary points on the board. This approach is useful to guarantee that a component is in place and receives the correct value, which is important, since a large number of board failures are caused by open/short circuits and wrong components. Functional test, which tests the functional correctness of components and the whole system, is typically run after process test. For products with a high demand for reliability, burn-in test is used to stress boards at high temperature to defect the component that first fails in stressful environments. Sometimes system test is performed as the final test, which is also a type of functional test. Each test technique has its advantages. For example, the solder reliability can be easily checked by AXI. The reversed or inoperative components can be detected by ICT. No single test can cover all possible defects.

With the development of high density assembly on printed circuit boards, the number of access points for in-circuit test keeps decreasing. Another interconnection test method, boundary-scan test, has recently attracted more attention. The boundary-scan architecture is defined in the IEEE 1149.1 and 1149.6 standards [9], [10] to ensure inter-connectivity between components. Simple boards and complex multi-board systems can effectively be tested using the IEEE 1149.6 standard-compliant equipment from the product design phase to mainstream manufacturing. At present, there is an increased focus in the electronics industry on using the concept of remote test and diagnosis in order to provide a mechanism that allows continued support of a product [11].

The boundary-scan test architecture is illustrated in Fig. 1.2. A boundary-scan cell, which includes a multiplexer and latches, is added to each pin on the chip. Boundary-scan cells in a chip can capture data from pins or core logic signals, and force data on to pins. The captured data is serially shifted out and externally compared to the expected results. Then forced test data is serially shifted into the boundary-scan cells. All of this is controlled from a serial data path called the scan path or scan chain. By allowing direct access to nets, boundary-scan eliminates the need for a large number of test vectors typically needed to properly initialize sequential logic. Potential benefits realized from the use of boundary-scan are shorter test times, higher fault coverage, increased diagnostic capability, and lower capital equipment cost.

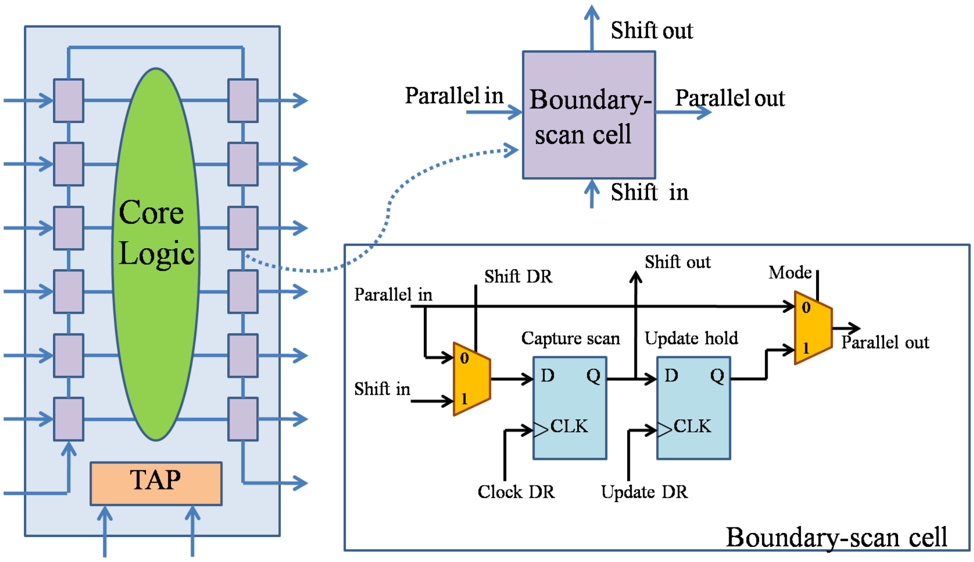


Figure 1.2: The boundary-scan test architecture.

Despite the benefits offered by the above techniques, the individual component testing and interconnection testing are no longer sufficient to guarantee the quality of complex circuit boards [12], [13]. It is often the case that all the chips on a board pass the automated test equipment tests, but the board still fails in functional test. The reason is that the board-level test environment is different from that at the chip level. At the chip manufacturing sites, chips are tested in a standalone mode, but other issues, e.g., signal integrity, must be considered at the board-level test [14]. Functional test plays an important role in capturing the defects that cannot be easily detected in ICT. Its main object is to verify the functional correctness of a board, which is independent of the hardware implementation of the board.

Functional test varies across products. Taking network systems (ethernet switches) as an example, functional test (or diagnostic program) performed on them consists of accessibility tests between the route processor and interface modules, online insertion and removal diagnostic tests, and snake tests through the switch router to ensure connectivity between ports [15]. These functional tests today are indispensable in system test and diagnosis. In practice, functional testing targets only a subset of all the design's functions, usually the critical functions identified by designers. Functional test sequences are often derived from design verification programs, and they are close to the practical scenarios that occur in the field. With the need for higher throughput and more efficient production and testing, electronics contract manufacturers and original equipment manufacturers are demanding sophisticated test platforms for functional testers that are built on open industry standards [16].

1.2 Hardware Errors and Fault-Insertion Test

An error is a wrong output signal produced by a defective system. A typical hardware platform can encounter three types of hardware errors during operation [17, 18]:

(1) Hard error — these are caused by permanent and readily reproducible physical defects. Most defective parts affected by this type of error can be screened by manufacturing testing but some may still be shipped to customers.

(2) Intermittent error — these failures are caused by occasional faults that occur as a result of frequency, voltage or other environmental conditions that are reproducible under the same conditions. This type of fault is usually attributable to corner conditions in design.

(3) Transient error — these are caused by "one time" and non-reproducible faults that are usually caused by high-energy particles (e.g. cosmic radiation, alpha particles) striking latches or memory cells and flipping circuit logic values. Cosmic particles are high-energy particles from outer space that cannot be prevented. Alpha particles originate within chip packaging materials and can be minimized via improvements in the manufacturing process. As semiconductor processes and core voltages keep shrinking, these particle-induced failures are becoming a pressing issue for hardware systems. Thus error detection and handling techniques are crucial to mitigate the consequences of such failures. These errors are also referred to as soft errors or single event upsets (SEU).

Reliable systems employ error-handling techniques to ensure continuous operation in the presence of hardware errors. All current hardware platforms at "system companies" such as Cisco Systems, Inc. employ some level of error handling, from the simplest form of rebooting the box on low-end platforms to more complex hardware redundancy schemes. In addition, such platforms use some level of diagnostic software to diagnose and repair boards in manufacturing and allow customers to quickly identify and repair faulty hardware components within their networks. The need to validate the above types of platform software requires the use of an error stimulus of some sort to ensure that both types of software perform as intended in the presence of faulty hardware. Runtime software must respond to hardware errors in the prescribed manner, and diagnostics must quickly and correctly isolate errors to a minimal set of failing components. This type of error-stimulus testing is called fault-insertion test (FIT).

In order to assess system reliability and facilitate fault diagnosis, FIT intentionally inserts faults to a system and observes the system's behavior in the presence of faults [19]. It provides an indication of how the system responds to real hardware errors. The intentionally inserted faults are used to model the effects of manufacturing defects [20, 13]. FIT is an essential tool to verify the robustness and error-handling capabilities of boards and systems. It is also an accurate and objective method to evaluate the efficiency of diagnostic programs [13].

Fault-insertion test can be either performed in a simulation environment or integrated into hardware with fault insertion features [21]. In a simulation environment, the logic values of nets in a circuit can be forced to faulty values using Verilog programming language interface [22]. Simulation based FIT is easy to implement, but test typically runs for a long time in large systems. Several different hardware implements are in use today. The most common one is external pin-level fault insertion, in which stuck-at faults are inserted to the pins of components by hardwiring [2]. This method is costly and requires physical access to the components, which does not accommodate the high density systems.

In contrast, a pin-level fault insertion technique based on the boundary scan design provides a convenient means of fault insertion, referred to as *fault insertion boundary scan* (FIBS) [2]. FIBS is incorporated into the boundary scan design. Therefore fault insertion can be easily automated without the need for physical access or alteration of the circuit board. The compatibility of the FIBS with boundary scan cells also promises relatively small area overhead.

The implementation of the basic fault-insertion boundary scan cell is illustrated in Fig. 1.3 . It is almost identical to the conventional boundary scan cell, except for the output multiplexer. If the FI enable signal is inactive, the cell performs the function of a normal boundary scan cell. When the FI enable signal is active, the FIBS cell is placed into the fault insertion mode. In this mode, the chip performs normal operations. Faulty values are inserted only at those pins that have their FI enable signal on. The timing penalty caused by the FIBS cell is the delay of one multiplexer. Besides the pin-level fault insertion, faults can be inserted at various locations, e.g., internal critical nets. A simple way is to place a 2-to-1 multiplexer on the net. If fault insertion is enabled, the faulty value is passed out through the multiplexer; otherwise the normal value is selected. Additional faulty signal and control signal are needed.

Fault-insertion test and its variants have attracted considerable attention in the study of reliable system design [23], [24], [25]. In [23], it is used to characterize the effects of transient faults on a high-performance processor pipeline, including the analysis of fault masking and identification of the most vulnerable parts of the processor. In [24], the soft-error resilience results obtained from the statistical fault insertion were verified using proton-beam experiments. In [25], an evaluation tool, DEPEND, is proposed to provide an integrated design and fault insertion

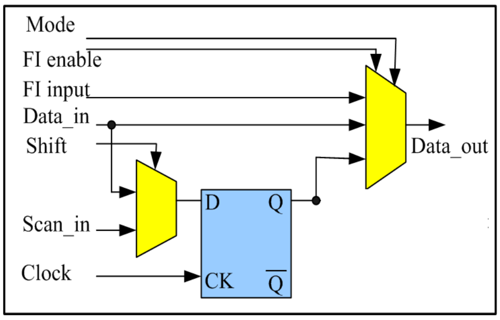
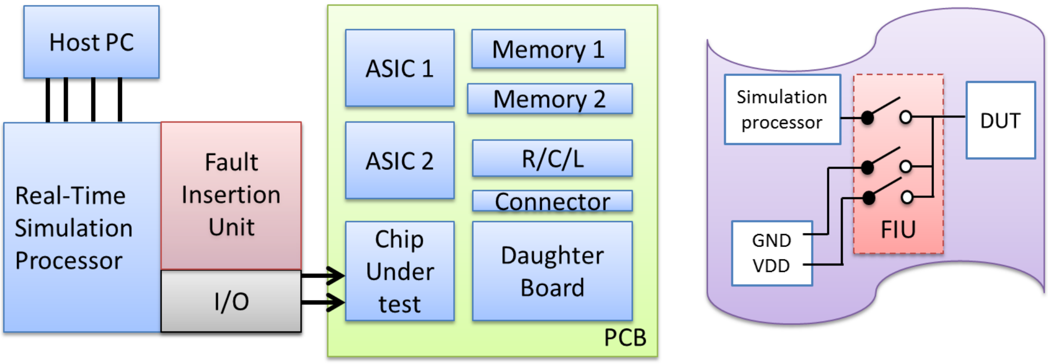


Figure 1.3: Basic fault-insertion boundary scan cell [2].

environment for system dependability analysis.



(a) Test structure (b) Principle

Figure 1.4: Test system with a FIU and the principle of the FIU illustration [3].

National Instruments Corporation has developed fault insertion units (FIUs) to create faults between the device under test (DUT) and the rest of the system to test, characterize, or validate the DUT's behavior under specific failure circumstances [3]. The FIUs are inserted between the I/O interfaces of a test system and the DUT. Therefore, the test system can switch between normal operation and faulty scenarios, such as shorts to power supply, shorts to ground, or opens. In Fig. 1.4, the block diagram of the FIT system and the fault insertion principle are illustrated.

FIT can be also applied for fault diagnosis, especially in the diagnosis of functional failures at system level. As modern systems become more complex, high defect rates have emerged as a serious concern, and defects are manifesting themselves in subtle ways. Defect screening and isolation are increasingly difficult. Traditional scan-test methods are not sufficient to guarantee product quality, and they are not able to rapidly locate the root cause of failures in large systems. In FIT-based diagnosis, artificial faulty scenarios are created by activating fault insertion logic during runtime. The response of a system to a known fault induced by FIT can be “learned”. The system responses under different faulty scenarios are typically saved in a dictionary for future reference. The root cause of a failure can be located by comparing the responses saved in the dictionary to the response of the real faulty system. Various reasoning methods can be used to determine the root cause based on the pre-computed knowledge base, e.g., Bayesian inference, artificial neural networks, etc. [26], [27], [28].

1.3 Board-level Fault Diagnosis and Challenges

Fault diagnosis isolates the root cause of a malfunction system by collecting and analyzing information on system status using measurements, tests, and other information sources. It is important at all stages of the product life cycle but particularly during manufacture and field maintenance.

The goal of fault diagnosis is to determine the fault location. The degree of accuracy to which faults can be located is referred to as diagnostic resolution [29]. Diagnostic success rate is equal to the number of correctly diagnosed cases over the total number of cases under diagnosis. The diagnosis process can be hierarchically carried out as a top-down process (with a system operating in the field) or a bottom-up process (during the fabrication of a system). In the top-down process (system→boards→chips), the first-level diagnosis usually deals with large units such as boards. The faulty board is then tested to locate the faulty component on the board. Accurate location of faults inside a faulty chip is important information for chip manufacturers to improve the fabrication process. In the bottom-up approach (chips→boards→system), a higher level is assembled only from components already tested at a lower level. This is done to minimize the cost of diagnosis and repair, which escalates significantly with the level at which faults are detected.

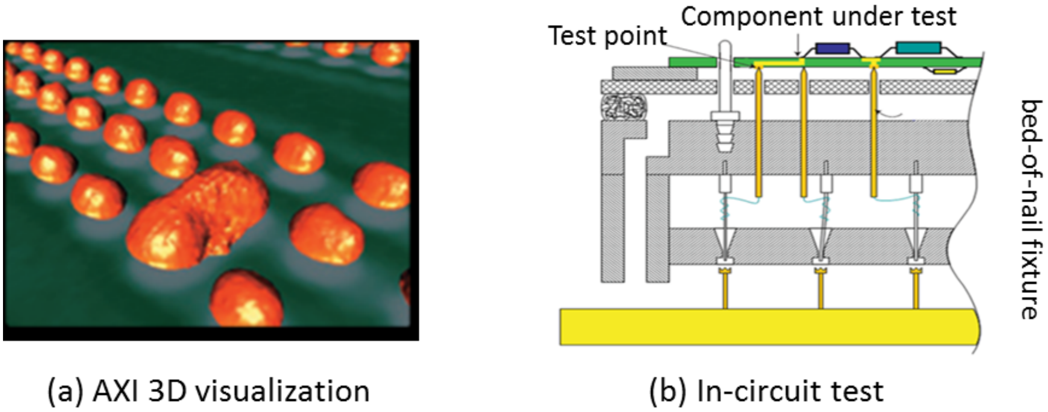
At the chip level, existing electronic design automation tools are able to perform accurate and high-resolution diagnosis. A defect's most probable failure mechanism, logic location and even physical location can be determined, according to scan test patterns [30]. Unfortunately, the board-level diagnosis is much more challenging. There is no effective flow to locate the root cause of a functional failure. Once a failure is detected in the manufacturing line, the failed product is sent to the diagnosis department for repair. Typically technicians run additional functional tests and measurements based on their personal experience. This process is time-consuming, and there is no guarantee on the success of repair. To make matters worse, a board has to be scrapped after a few unsuccessful repair attempts. Current diagnostic software is not able to accurately and rapidly locate the root cause. In addition, the development of the diagnostic software is increasingly challenging, as electronic systems become more complex.

Miscellaneous board-level fault diagnosis methods have been published in the literature. Rule-based, module-based and reasoning-based diagnosis are three widely used diagnosis techniques [31].

Rule-based diagnosis methods take the form "IF syndrome(s), THEN fault(s)" to locate the fault(s). Hundreds or thousands of rules may be required to represent all the relevant knowledge for the system under diagnosis. Rule-based diagnosis involves the extraction of syndromes from the failure, and the firing of rules that match the syndromes. This process is repeated iteratively until the root cause of the failure is found. Rule-based expert systems have been developed for board repair and maintenance [32]. The primary advantage of this flow is its simplicity, but it is difficult to acquire the knowledge needed to construct the rules base.

Model-based diagnosis uses a model to predict faults, and it takes into account observations and information from a real system. The model is an approximate representation of the real system under diagnosis. Models are often constructed in a hierarchical fashion. The initial diagnosis results using a high-level model are passed to the next level of diagnosis with a detailed model. Compared to rule-based techniques, it is easier to represent complex structured knowledge in model-based techniques, thus leading to greater computational efficiency. However, the bottleneck in this approach is the construction of the model for a complex system that is representative of today's designs. In [33], a model-based inference engine was built on the basis of the correct operation of a device, and enhancement to the basic inference engine were presented as well. One advantage of this approach is that the model can be constructed while the product is being developed by the hardware and diagnostic engineers, but the expertise of adjusting the model in cases where the reasoning was initially not correct is usually not available.

Reasoning-based diagnosis involves the storing of experiences of past solutions (known as "cases"), and retrieving a suitable case to solve a new problem. Case retrieval consists of identifying an appropriate set of features for the current problem, using these features to search for similar cases in the database, and eventually deriving the conclusion with a reasoning scheme, e.g., Bayesian inference [34]. In Bayesian inference, the occurrence probability of a fault syndrome given a faulty component is first calculated based on the known cases. Given a new case, the probability of a component being the root cause with the observed syndromes can be derived using Bayes' formula. The component with the highest probability is inferred as the root cause. More details were described in [26]. The effectiveness of a reasoning-based diagnostic system can be continually improved by exploiting the knowledge from previous successful/failed diagnosis attempts. The effectiveness of reasoning-based diagnosis depends on the availability of suitable cases that are generated from historical data or simulation, the extraction of effective features of



(a) AXI 3D visualization (b) In-circuit test

Figure 1.5: Fault identification in AXI and ICT [4].

the cases, and the underlying reasoning scheme.

On the manufacturing line, if a board fails process test (AXI or AOI), or in-circuit test, the diagnosis is relatively easy; see Fig. 1.5. The open or short solder joints are visualized in the image of AXI or AOI. In-circuit test targets an individual component. The root cause is therefore the component under test when a failure occurs. Using boundary-scan test, the failed scan cell can be traced back from the output patterns. Therefore, a challenging task is to locate the root cause of failure when a board fails functional test or system test. The main challenges associated with functional diagnosis are listed below.

First, modern printed circuit boards are very complex, and they typically consist of many ASICs, external memory devices, and hundreds of passive components. It is difficult to understand the functionality of the board and the relationship between root causes and failures under various functional tests. So it is challenging to construct a model to represent the real board, and the traditional model-based diagnosis method cannot be easily used for fault diagnosis on a complex board.

Second, controllability and observability of functional test are limited, and it is very challenging to reproduce the functional failures [35]. Moreover, the board-level test environment is different from that at the component level. At the component manufacturing sites, components are tested in a standalone mode. Thus the components that pass ATE test might still fail in board-level test due to board-level issues, e.g., power supply noise, and signal integrity [14].

Third, state-of-the-art board-level diagnostic software is unable to cope with high complexity and ever-increasing clock frequencies, and the identification of the root cause of failure on a board is a major problem today. Ambiguous or incorrect repair suggestions lead to long debug times and even wrong repair actions. A failed attempt at replacing a fault-free component can significantly increase the repair cost and debug time. A board may have to be scrapped after a few unsuccessful repair attempts.

Finally, due to mass production and the need to ramp-up yield, the time available for diagnostic technicians to debug a given defective board is limited. Yet, thorough diagnosis of boards is a time consuming and tedious process. When the diagnostic software fails to provide clear repair guidance, technicians have to take actions based on their accumulated experience. According to the feedback from manufacturers, the efficiency of a senior technician is notably higher than that of an inexperienced one. However, debug knowledge is difficult to formulate, document, accumulate, and transfer. Experienced technicians are stressed by high workload and they barely have time to train new employees. The high employee turnover rate at manufacturing sites further exacerbates this problem.

It is important to note that the impact of these challenges affect not only on the manufacturing process itself, but they are also key to the entire semiconductor business, both in terms of enabling the timely delivery of future processes and cost effective products, but also in terms of meeting customer expectations for reliability.

1.4 Research Motivation

Printed circuit boards and electronic systems are found in a wide range of applications, e.g., vehicles, medical equipment, data centers, and network communications. With an increase in system complexity, the need for automated effective diagnostic tools has become more acute. This is driven by economic scaling of test, limited board test access, reduced time-to-market, shorter life cycles of products, and environmental requirements. We describe below some of these issues in more detail.

- **Lower manufacturing test cost**

With the development of process technology, the fabrication cost per transistor continues to decrease, but the testing cost per transistor remains the same or even increases slightly. According to the ITRS roadmap, the cost of fabricating a transistor and testing a transistor will be equal in 2012. Today the costs associated with testing and diagnosis is one of the highest contributors to manufacturing cost. The cost of different levels of testing is often estimated using the "rule of ten", as shown in Table 1.1 [1]. The table shows that if it costs $1 to test an ASIC, the cost of locating the same defective ASIC when it is mounted on a board is about $10; when the defective board is plugged into a system, the cost of identifying the faulty board and repairing the system is $100. If the defective system is delivered to end users, the cost of recall and repair will go up to $1000. In 2007, the hardware failure on the Xbox 360, "the red ring of death", cost Microsoft more than $1 Billion. In

Table 1.1: Rule of ten in test economics [1].

Testing	ASIC Testing	Board Testing	System Testing	Out in Field
Cost of Test	1	10	100	1000

2008, Amazon lost $31,000/min within 5 hours of service outage. Therefore, it is necessary to diagnose failures at an earlier assembly stage and provide practical diagnostic tools to lower the diagnosis and repair cost.

- **Reduction of yield loss**

The first-pass rate on current production lines is becoming unsatisfactory. For some complex telecommunication boards, only 60% to 70% boards pass all functional tests the first time. The number of failed boards waiting for diagnosis accumulates rapidly, especially in high-volume production. Circuit boards have to be scrapped, if failures are still detected after a few times of repair. This is a great impediment to profit, since one complex telecommunication board costs up to tens of thousand dollars [36]. Therefore, efficient and accurate diagnosis methods are urgently needed.

- **Accelerate diagnosis process**

In an increasingly competitive marketplace, time-to-market is critical for the success of a company. This is another driver for the use of automated test and diagnostic tools. Locating the root cause of board-level functional failures usually requires an "expert", with engineering skills in both hardware and software. Depending on the complexity of the product, it can take several months (even years) to develop this level of expertise. During initial product ramp, this expertise is usually most needed but often unavailable, so that diagnosis time is often very long [33].

Specifically, a common scenario in industry is "No Trouble Found" (NTF). It means that a component on a board fails board-level functional test, but it passes ATE test when it is returned to the component supplier for warranty replacement or service repair. To solve this problem, we need a diagnostic method that is not only able to locate the faulty component on a malfunctioning board, but it can also narrow down the defective area inside the faulty component. The detailed failure information, e.g., failed test, failed sub-modules, failing cycles, is very important for component suppliers to troubleshoot and debug their manufacturing problems.

- **Accommodate new assembly technology**

Environmental concerns, legislation, and market requirements are leading to increasing use of lead-free assembly technology in the electronics industry [37]. In 2006, the European Union Waste Electrical and Electronic Equipment Directive and Restriction of Hazardous Substances Directive (RoHS) came into effect prohibiting the intentional addition of lead to most consumer electronics produced in the European Union. Quality and inspection of lead-free processes are emerging challenges in manufacturing test. The new assembly methods result in less access to board testing, debug and diagnosis. Changing package geometries are reducing the fault coverage that can be obtained using human visual inspection techniques. Some process imperfection and incorrect passive components may escape AXI, AOI and ICT test. Detection and diagnosis of such faults are new requirements in the functional test stage.

- **Expand industry adoption and use**

Another motivation for this thesis is that current application of automated board-level diagnostic tools is limited. Fault diagnosis has been an active area of recent research and development, but it still faces with challenges related to accuracy, resolution and large data requirements. Over the past three decades, automating fault diagnosis using artificial intelligence (AI) techniques has been a major research topic [38]. There has been much progress, but industry acceptance, particularly in cost-sensitive segments, has not been high. In this thesis research, we provide generic and automated solutions for the board-level fault diagnosis, which provide quantifiable improvement over brute-force, trial-and-error manual repair. We expect that the proposed automated solutions can be easily implemented and deployed in industrial manufacturing lines and debug departments.

1.5 Thesis Outline

Motivated by the needs of board-level manufacturing test and fault diagnosis, we propose solutions to address the difficulties involved with fault-insertion test and functional diagnosis. The thesis contents include high-level fault model to optimize fault-insertion test, effective fault syndromes — error flows, diagnosis-oriented fault insertion point selection, and automated diagnostic system using machine learning algorithms. The remainder of this thesis is organized as follows.

In Chapter 2, physical defect modeling is presented to facilitate fault-insertion test, and hence accelerate system reliability assessment and fault diagnosis. Our goal is to select the most effective set of output pins for fault insertion (FI), such that faulty values on these pins can model most internal physical defects, thus mimicking the behaviors of hardware defects encountered in the customer environment. To achieve this goal, we develop a simulation framework to study the propagation of physical defects from internal nodes to output pins. We use integer linear programming and heuristics to minimize the number of pins for FI and assign appropriate fault types for those selected pins. The fault insertion locations are not limited to the pins at the chip level, but they also can be the outputs of submodules inside chips, since the method can be hierarchically applied to logic blocks within chips. The addition of error-forcing logic within chips and programmable devices permits an additional level of testing and reliability assessment.

In Chapter 3, a diagnosis framework based on FIT and Bayesian inference is presented. Bayesian inference provides a unified and intuitively appealing approach for drawing inferences from observations and *a priori* beliefs. In order to apply

Bayesian inference, we create faulty scenarios using FIT and derive the occurrence probability of a fault syndrome in the presence of a specific fault. Bayes' theorem is then used to calculate the probability of a fault candidate being the root cause. The calculation is performed for each fault candidate. Then all the fault candidates are ranked by the probability. The one with the highest probability is considered to be the root cause of failure. Results on a case study using an open-source RISC system-on-chip (SoC) demonstrate the feasibility and effectiveness of the proposed approach.

In Chapter 4, an alternative fault syndrome, error flow, is proposed to describe functional failures in a more effective way. The time of error occurrence is analyzed as an attribute for functional diagnosis, in order to take the functionality of the circuit into account. Error propagation mimics actual data flow in a circuit, thus it reflects the native behavior of circuits. In contrast to conventional fault syndromes, an error flow takes the time of error occurrence into account. With this additional timing information, more fault syndromes are distinguishable, and thus diagnostic resolution is improved. The advantages of using an error-flow dictionary are demonstrated using the same SoC as in Chapter 3.

In Chapter 5, the diagnosis-oriented fault-insertion point is first defined and a selection scheme based on integer linear programming is proposed. We select the most effective outputs of a module where fault insertion logic should be placed. Faults inserted at the selected points are able to generate fault syndromes that are the most similar to the syndromes produced by the internal defects. This approach also ensures that the ambiguous fault candidates from other modules are maximally removed from the set of suspects.

In Chapter 6, machine learning algorithms are applied in fault diagnosis to avoid the difficulties involved in knowledge acquisition. The use of two machine learning algorithms, artificial neural networks (ANNs) and support vector machines (SVMs), is explored. Fine-grained fault syndromes extracted from failure logs and the corresponding repair actions are used to train the diagnostic system. After training is complete, the diagnostic system is able to suggest the most likely faulty component given a new failed board. The diagnostic accuracy achieved by SVMs-based method is slightly higher than that achieved by ANNs-based method, but the relationship between the critical syndromes and the root causes can be more easily interpreted using the ANNs-based method. An industrial board, which is currently in high-volume production, has been used to validate the effectiveness of this diagnosis approach.

In Chapter 7, we summarize the contributions of the thesis and identify directions for future work.

Chapter 2

Physical Defect Modeling and Optimization of Fault-Insertion Test

Fault-insertion test is a promising method for system reliability assessment and fault isolation. It provides feedback on the fault tolerance of a large system, creates artificial faulty scenarios that can be used as references for fault diagnosis, and leads to a quality diagnostic program. The number of candidate fault insertion locations dramatically increases, as the increase of system density. Optimization of fault insertion location is critical for accelerating the assessment of system reliability and constructing a complete knowledge base for fault diagnosis.

In this chapter, we construct a pin-level fault model that is able to effectively mimic the errors caused by physical defects within the component. A simulation framework and optimization techniques are proposed to select a minimum subset of output pins that can represent as many physical defects as possible. The optimization results provide guidelines on the fault insertion locations and the appropriate fault types for insertion. In addition, three intrinsic characteristics of output pins, including testability number, fan in size, and transition counts, are analyzed. The effectiveness of the proposed model is evaluated in terms of impact on system response and error-detection latency. Experimental results are presented for OpenCore benchmarks.

2.1 Problem Statement and Prior Work

The reliability of products is critical to company success. When unreliable products are released to the market, companies suffer tremendous loss in terms of productivity, reputation, and financial performance. For safety-critical electronic systems, the demand of product reliability is even higher, since the cost of a minute of downtime for a critical system can be in millions of dollars. Fault-insertion test (FIT) is an accurate and objective method to verify the ability of diagnostics to accurately detect and diagnose failures at the board or system level [13]. It improves the speed of releasing a quality diagnostic program before manufacturing and provides feedback on the fault tolerance of large systems.

FIT is resource-limited in terms of silicon area, time, equipment, personnel, and what is actually possible to test from one design to the next. If we consider design hierarchy and all the modules in a hierarchical manner, the number of outputs of the modules is prohibitively large. It is not practical to exhaustively and individually manipulate FI logic at all the potential sites to evaluate system reliability or to do fault isolation. In order to overcome this major limitation, we need to turn on the most effective fault insertion logic to see if the system is reliable under this fault (for reliability assessment), or if the erroneous effects caused by the inserted fault are similar to the failure on the real malfunctioning board (for fault isolation). In addition, at the chip level, usually a small number of chip IOs and internal nodes are equipped with FI logic elements to avoid excessive area and complexity. Hence, the selection of these FIT-enabled IOs or internal nodes becomes critical; these nodes should be able to represent as many effects of physical defects as possible that occur inside the chip.

In FIT, a fault inserted in a system can be specified by at least four parameters, namely fault insertion time, fault location, fault type, and fault duration. Taking all these parameters into account, the fault space is nearly infinite. To cope with the large size of the fault space, several methods have been described in the literature. In [39], a 3-stage random sampling technique was developed; however, a drawback of random selection is the high probability that the injected faults will remain latent, therefore leading to no-response fault insertion experiments. Another selection mechanism based on fault-list generation was reported in [40]. It is an ad hoc technique that depends on the analysis of dataflow and the instruction-set architecture. In [41], a statistical approach was proposed to quantify the number of faults to inject in order to achieve a given confidence, but the explicit locations for fault insertion were not provided.

The key idea in this chapter is to construct a defect model at the pin level of a chip to represent physical defects inside the chip. By utilizing the physical defect model at the pin level, the effects of multiple defects are mimicked by one artificial fault, and hence FIT becomes efficient. This improves the efficiency of system reliability assessment and diagnosis coverage measurement. In Fig. 2.1, the physical defects inside the chip are denoted by circles. These defects manifest themselves at the pins of the chip as errors, and the pin-level artificial faults, denoted by diamonds, are used to mimic the erroneous behaviors at the pins caused by physical defects. By applying FI at the pin level, instead of traditional full module FI, the proposed approach shrinks the large fault space in terms of fault location. This improves the efficiency of system reliability assessment and knowledge-base construction for fault diagnosis. The proposed physical defect modeling not only accelerates fault simulation in system-reliability assessment, but it also provides guidelines on the placement of FI logic elements in an actual hardware implementation.

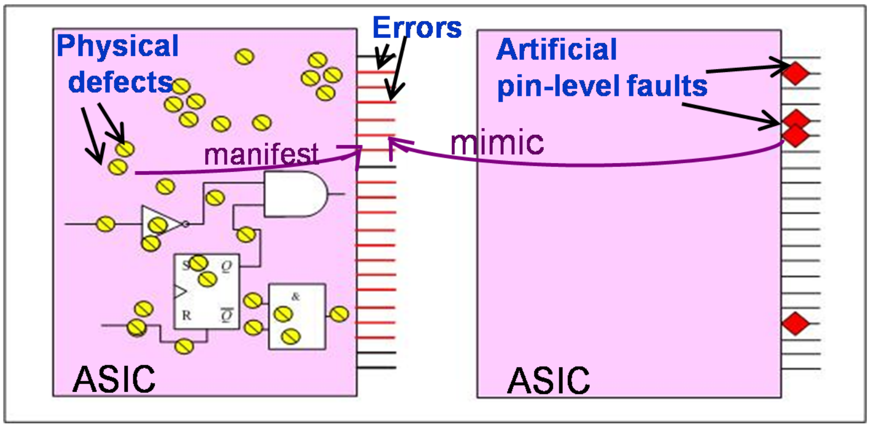


Figure 2.1: Physical defect modeling.

2.2 Model Construction and Optimization of FIT

To construct an effective physical-defect model, an optimization scheme has been developed, whereby the most appropriate fault locations and fault types can be specified for FIT. It is not necessary for the pin-level model to extract the exact response due to every physical defect. We want to extract, represent, and generalize the similarities of the effects caused by physical defects, so that it is possible to use one fault at the pin level to represent many physical defects inside the module, thus reducing FI effort, both for simulation and for hardware implementation. In addition, the pins targeted in this work are not limited to the chip pins, but they also refer to the outputs of modules inside chips. Fault insertion cannot only be performed at the pin level, but also at the logic level. For the sake of similarly, the generalized fault model is referred to as the pin-level fault model.

Physical-defect modeling begins with a logical abstraction of physical defects, and concludes with an optimization step. Four steps involved in the process of model construction are shown in Fig. 2.2. According to the fault simulation results and constrained optimization algorithms, a minimum subset of output pins can be selected and the most appropriate fault types can be assigned to those pins

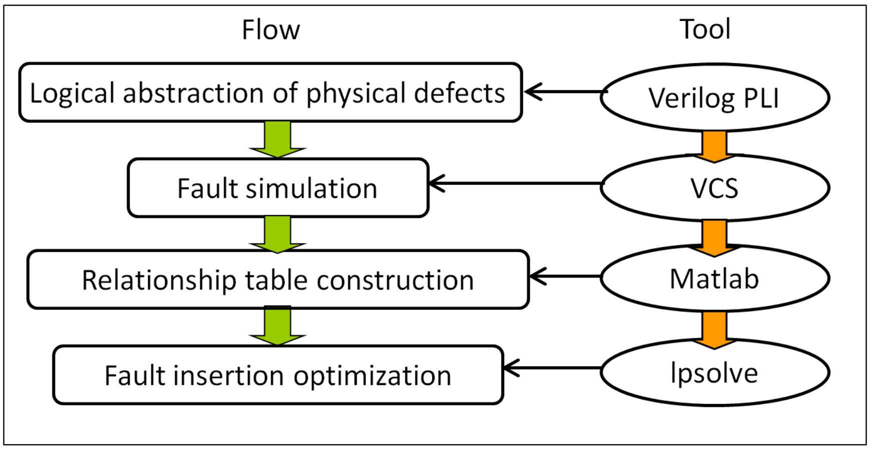


Figure 2.2: The flow of model construction and the tool used in each step.

for FI. We start with the logical abstractions of physical defects and run fault simulation based on these logical abstractions, since FI at the pin level depends on the impact of the physical defects inside the module. Faults are injected using the Verilog programming language interface. Based on our simulation infrastructure, the defect location, type, duration, start and end time all can be specified by the user. Fault simulation runs are carried out using the Synopsys Verilog compiler (VCS). Based on the faulty response of the circuit, a relationship table between injected internal faults and errors at the pin level (pin-faults) can be constructed. After the relationship table is constructed, the optimization problem of pin-fault selection for FIT is formulated as an integer linear programming (ILP) model and solved using a standard solver, such as lpsolve, which can be used for solving pure linear, (mixed) integer/binary, semi-continuous and special ordered sets models [42]. In order to make the method scalable for large designs, a heuristic method is also developed to handle large data sets. The details of each step are presented in the following subsections.

2.2.1 Logical Abstraction of Physical Defects

In our simulation framework, five types of logical abstractions are used to abstract defects at the gate level. Transistor-level defects, e.g., stuck-open, and stuck-on, are not considered in this work in order to limit complexity. The logical abstractions are written in Verilog Procedural Interface (VPI) script. The following is the list of the function and arguments for each abstraction:

1. **$stuck_at (net, start_time, end_time, stuck_value)**: It forces the value of 'net' to 'stuck_value' at 'start_time' and releases forcing at 'end_time'.
2. **$flip (net, start_time)**: It changes the value of the 'net' to the complementary value at 'start_time'. It is a one-time event, and there is no continuous forcing on that net.
3. **$bridging(net, dominant_net, start_time, end_time)**: It forces the value of 'net' to be the same with the value of 'dominant_net' at 'start_time' and releases the forcing at 'end_time'.
4. **$glitch (net, start_time, pulse_width)**: It adds a pulse to 'net' at 'start_time' and the width of the pulse is defined by 'pusle_width'.
5. **$delay (port, rising_delay, falling_delay)**: It applies transition delay on 'port'; the rising-edge delay and falling-edge delay can be specified by 'rising_delay' and 'falling_delay'.

Stuck-at faults are used to abstract and model hard defects (opens or shorts) that occur in manufacturing. Flip faults (bit flips) are typically used as an abstract model of transient defects in simulation. This fault model represents the state-inverting phenomenon, which is a widely used fault model in system reliability evaluation [24, 23]. Bridging faults are used to abstract additional hard defects, such as two signals are shorted together. They are normally restricted to signals that are physically adjacent in the design. Glitch and delay faults can be used to abstract intermittent defects such as crosstalk. Delay faults can also be used to abstract hard defects, such as resistive shorts/opens, and process variations.

The above VPI routines are integrated with a Verilog-based testbench and used to perform fault injection for arbitrary circuits. VPI is the third-generation Verilog programming language interface (PLI) [43]. It uses a C-platform and consists of a set of accesses and routines that can be called from standard C programming language functions. These routines interact with the instantiated simulation modules and users can customize routines to control the nets in the compiled module, such as observing or forcing the logical values of the nets through Verilog tasks.

2.2.2 Fault Simulation

Fault simulation is typically used for the development of manufacturing tests. It determines the coverage for a given set of input stimuli and for a given fault model or a given fault list. With the help of other programs, it can produce a set of vectors with a given fault coverage for manufacturing test [7]. Serial fault simulation is used here to collect the faulty responses caused by the injected internal faults. By injecting internal fault one by one and comparing the output responses of fault circuits with those of fault-free circuits, the mismatches at output pins caused by the internal faults are obtained. To better reflect the effects of physical defects, fault simulation is performed at the gate level and functional patterns are applied.

In fault simulation, only the mismatches between faulty circuits and fault free circuits are stored, in order to save storage space. Specifically, when the output response in fault simulation is different from the fault-free value, we record the current pin identifier, injected internal fault identifier, test pattern identifier, faulty value, and expected value at that pin. According to the recorded mismatch information, we can determine which internal fault can be modeled by a specified pin-level fault and construct the relationship table in the next step.

2.2.3 Pin-Level Fault Model and Relationship Table Construction

To describe the faulty behavior at output pins, three types of pin-level faults are defined: pin-stuck-at-0, pin-stuck-at-1 and pin-flip. They are abstracted from practical scenarios of FIT hardware implementation. Each pin-fault is associated with a time frame in terms of several clock cycles. Within the time frame, the logical value of an output pin is monitored. If the logic value at a pin meet the following constraints, a pin-fault is considered to occur at that pin.

- **Pin-stuck-at-0**

  The logical value at the pin is either "D" or "0" and at least one "D" occurs in that time frame, where a D corresponds to 1/0 (1 in the good circuit, 0 in the faulty circuit)

- **Pin-stuck-at-1**

  The logical value at the pin is either "$\overline{D}$" or "1" and at least one "$\overline{D}$" occurs in that time frame, where a "$\overline{D}$" corresponds to 0/1 (0 in the good circuit, 1 in the faulty circuit).

- **Pin-flip**

  The logical value at the pin is either "$\overline{D}$" or "D", which is always complimentary to the fault free value in that time frame.

In the above fault models, we require that at least one "D" or "$\overline{D}$" occurs within the time frame. This guarantees that the pin has been corrupted in that period. For example, if the logical value of a pin stays "0" during fault simulation, while the golden value of the pin is always "0" in that period, there is no pin-stuck-at-0 fault occurring at that pin. In the remainder of this section, the time frame is called a fault period.

The relationship table (RT) is a two-dimensional matrix, similar to a fault dictionary. It is used to reflect the fact that a pin-fault can represent a defect. Here we assume a pin is able to represent an internal defect, if the defect can manifest itself at that pin. Before providing more details on the RT, we first clarify three types of coverage. The internal defect coverage based on pin-faults is different from the traditional fault coverage measure. To be specific, three types of coverage of interest are listed below:

$$\texttt{Defect coverage} = \frac{\texttt{\#Detectable defects}}{\texttt{\#Total defects}} \tag{2.1}$$

$$\texttt{FI-defect coverage} = \frac{\texttt{\#FI-detectable defects}}{\texttt{\#Detectable defects}} \tag{2.2}$$

$$\texttt{Selection coverage} = \frac{\texttt{\#Selected-FI-detectable defects}}{\texttt{\#FI-detectable defects}} \tag{2.3}$$

Total defects refer to all the injected internal faults. Detectable defects are those internal faults that cause different logical values from golden values at output pins. FI-detectable defects are those internal faults that result in pin-faults occurring at output pins. Moreover, if any pin corrupted by a FI-detectable defect is selected for FI, this FI-Detectable defect is a Selected-FI-detectable defect. Based on these definitions, we aim at defining an appropriate pin-fault model to achieve high FI-defect coverage, and maximizing selection coverage.

The fault insertion parameters, namely, fault type, fault location, and fault insertion time, are controlled by the FI controller. In order to lower area overhead, the controller is typically designed in a hierarchical fashion. A number of pins are grouped and controlled by a group controller, and the actions of group controllers are determined by the global controller, as shown in Fig. 2.3. The global controller receives FI commands from the CPU interface and broadcasts the specified fault insertion parameters to each group controller. Each group controller operates independently. Therefore, when we determine if a pin-level fault can represent the behavior of an internal defect, we consider a group of pin-level faults.

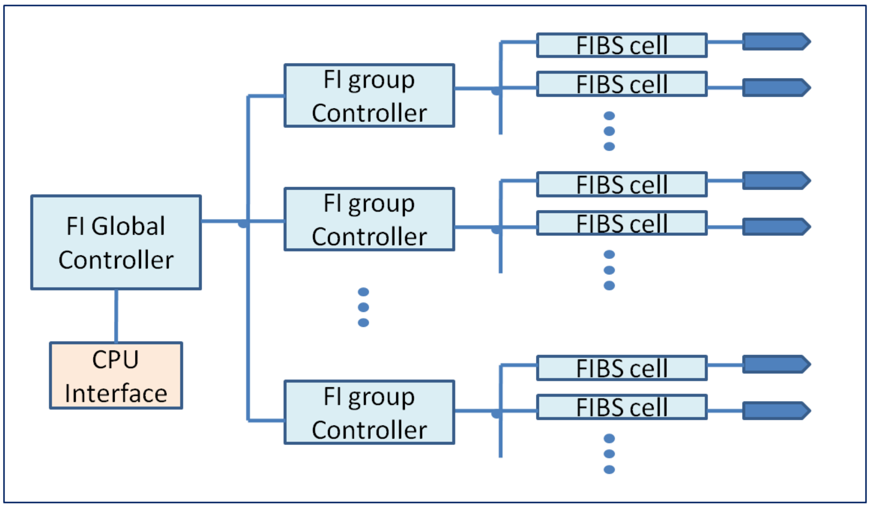


Figure 2.3: Block diagram of fault insertion controller.

Based on our optimization goal and the requirements of optimization methods, two relationship tables are constructed. One is group-fault versus defect RT, reflecting whether a group of pins with a designated fault type can represent a physical defect. An example is shown in Table 2.1. The other is group versus defect RT, which reflects if a group of output pins can represent a defect, without a specified fault type. An example is shown in Table 2.2.

The underlying reason for choosing a group of pins, instead of a pin, for each row in the relationship table is that the adjacent module pins are typically assembled in a group for FI in practical implementation; see Fig. 2.3. For example, a group of pins is controlled by the same FI group controller, and all pins in that group are selected at once.

In Table 2.1, the rows stand for the identifier of a group with a designated fault type, and the columns of the table list the internal defects. The entries in the table are either "1" or "0". A value of "1" implies that the defect on the column can be represented by the group with the associated fault type on that row. Likewise, an entry of "0" implies that the group with the designated fault cannot represent the defect, which therefore means that type of pin-fault does not appear at any output pin in that group during fault simulation. In Table 2.1, there are three groups of output pins and eight defects. If we intend to select a set of groups and corresponding fault types to cover all the defects, we can choose group1 with pin-s-a-0 fault, group1 with pin-flip faults and group 2 with pin-flip fault to cover all the defects. A total of eight defects are covered by the selected rows, denoted in bold in Table 2.1.

Table 2.1: An example of group-fault versus defect RT.

	d1	d2	d3	d4	d5	d6	d7	d8
group1, pin-s-a-0	1	0	1	0	0	0	0	0
group1, pin-s-a-1	0	0	0	1	0	1	1	1
group1, pin-flip	1	1	1	0	0	0	0	0
group2, pin-s-a-0	1	1	0	0	0	0	0	0
group2, pin-s-a-1	0	0	0	0	0	1	0	0
group2, pin-flip	0	0	0	1	1	0	1	0
group3, pin-s-a-0	0	1	0	0	0	0	0	0
group3, pin-s-a-1	0	0	0	0	0	1	0	0
group3, pin-flip	0	0	0	1	0	0	1	0

Table 2.2: An example of group versus defect RT.

	d1	d2	d3	d4	d5	d6	d7	d8
group1	1	1	1	1	0	1	1	1
group2	1	1	0	1	1	1	1	0
group3	0	1	0	1	0	1	1	0

In Table 2.2, the entry is “1” if any pin of the group corresponding to the row is able to represent the defect corresponding to the column. If the entry is “0”, it means that the group listed on that row cannot represent the defect, irrespective of the type of fault inserted on the pins in that group.

2.2.4 Optimization Algorithms for FI Selection

In this section, optimization techniques, based on integer linear programming (ILP) and heuristics, are presented for minimizing the number of output pins for FI and assigning fault types for those selected pins. Therefore the desired coverage level is achieved with the minimum number of pins and the fault-insertion cost.

1) Integer linear programming model

Using fewer FI logic elements is the best way to save FI cost in the hardware implementation of FIT. To minimize the set of groups of pins for FIT, a two-step optimization method is developed: first, minimize the group of pins without consideration of fault types; second, assign the appropriate fault type for each selected group. This selection method can solve the single pin selection problem as well, when the group size is equal to one.

Step 1. Select the group of output pins

In the first step, we only minimize the number of groups to achieve the desired selection coverage without consideration of the fault types of the groups. After the smallest set of groups is found, we use the second step selection to assign fault types for the selected groups. The minimum set of groups can be obtained by solving the following group-selection ILP model.

Group-selection ILP model:

Objective function:

$$\text{Minimize} \sum_{i=1}^{p-m+1} g_i \tag{2.4}$$

Constraints:

$$\sum_{i=1}^{p-m+1} (g_i y_{ik}) \geq d_k, k \in \{1, ..., n\} \tag{2.5}$$

$$\sum_{k=1}^{n} d_k \geq G \tag{2.6}$$

$$\sum_{j=i}^{i+m-1} g_j \leq 1, i \in \{1, ..., p-2m+2\} \tag{2.7}$$

where $G = $ #Selected-FI-Detectable defects; $g_i, d_k \in \{0, 1\}$.

Let the number of output pins in the circuit under test be p, and the group size be m. Therefore, the number of groups is $p - m + 1$. Let the number of injected defects be n. In this model, our objective is to minimize the $\sum_{i=1}^{p-m+1} g_i$, where g_i is an indicator to indicate if group$_i$ is selected. If group$_i$ is selected, $g_i = 1$, otherwise $g_i = 0$. By minimizing the summation in (2.4), we can obtain the smallest set of groups.

Constraints (2.5) and (2.6) ensure that the selected groups achieve the desired selection coverage. In Constraint (2.5), the constant y_{ik} is the entry on the row i and k column in the group versus defect RT. The variable d_k can be either 1 or 0. If d_k =1, it means that defect k is successfully represented by at least one selected group. To satisfy Constraint (2.5), at least for one group, $g_i y_{ik}$ is 1, when

d_k =1. If d_k =0, Constraint (2.5) is always satisfied. The constant G is equal to the number of Selected-FI-Detectable defects. Through Constraints (2.5) and (2.6), we do not specifically restrict which defect should be represented, but the total number of detectable defects by selected groups should be greater than G. This restriction ensures that the desired selection coverage can be achieved. In Constraint (2.7), the summation is less than or equal to one, which implies that the groups consisting of the same pins cannot be simultaneously selected. This constraint is used to guarantee that a pin does not belong to two FI logic elements in a practical implementation.

There are two special cases for this ILP model. One is that 100% selection coverage is required. Under this scenario, the model regresses to a simple coverage problem. For any defect k, d_k =1. Constraint (2.6) is trivially satisfied. There is no change for Constraint (2.7). The other special case is single-pin selection. In this case, group size is equal to one, and Constraint (2.7) is always satisfied.

Although one FI logic element is usually added to a group of pins in a practical FIT implementation, the optimization results for single pin selection can be used to derive a lower bound on the number of FI logic elements needed for group selection with different group size. The lower bound is given by the following formula:

$$\texttt{FI\_group\_m} \geq \lceil \frac{\texttt{\#FI\_single}}{m} \rceil \tag{2.8}$$

where FI_group_m is the minimum number of FI logic elements needed for group selection, the group size is m, and FI_single is the minimum number of FI logic elements needed for single-pin selection.

Step 2. Select fault type for the selected group

In this step, our goal is to assign an appropriate fault type for each selected group. The overhead for different fault-type insertions varies. Stuck-at fault has lower FI cost, since stuck-at fault insertion can be performed by simply soldering a switch between VDD or Gnd and the net under test, and closing the switch. This method is much cheaper, compared with loading a flipped signal to the faulty pin. The costs associated with different fault-type insertion are taken into account by adding weights corresponding to different fault types. Assuming the cost of stuck-at fault insertion is 1 and the cost of flip fault insertion is 2, the fault type selection ILP model is shown below.

Fault-type selection ILP model:

Objective function:

$$\text{Minimize} \sum_{i=1}^{2(p-m+1)} s_i + 2 \sum_{i=2(p-m+1)+1}^{3(p-m+1)} s_i \tag{2.9}$$

Constraints:

$$\sum_{i=1}^{3(p-m+1)} (s_i y'_{ik}) \geq d_k, k \in \{1, ..., n\} \tag{2.10}$$

$$\sum_{k=1}^{n} d_k \geq G \tag{2.11}$$

where G = #Selected-FI-Detectable defects; $s_i, d_k \in \{0, 1\}$.

In the objective function, the variable s_i is an indicator for denoting if the group with the associated fault on the row i is selected, if the group with the associated fault is selected, s_i=1, otherwise s_i=0. The first part of the objective function is the summation of the indicators for pin-s-a-0 and pin-s-a-1 faults. The weights of these stuck-at faults indicators are 1. The second part is the summation of the indicators for pin-flip fault. The weights of these pin-flip fault indicators are 2.

Similarly, Constraints (2.10) and (2.11) ensure that the selected groups with associated fault types can model a sufficient number of defects to achieve desired selection coverage. The constant y'_{ik} is the entry on the row i and column k in the updated group-fault v.s. defect RT. (The group-fault v.s. defect RT is updated by deleting all the unselected groups in step one.) The meanings of parameters p, n and m are the same with those in the first step and the variable d_k has the same meaning as well. The optimization results for the second ILP model provide the final fault-insertion decision, including the groups where the FI logic elements should be inserted and the fault types for each selected group. The overall procedure for the two-step selection technique is summarized below.

ILP Procedure: Select groups and fault types for FI

1. Prepare relationship tables:

   RT1: Group versus Defect RT

   RT2: Group-fault versus Defect RT
2. Construct group-selection ILP model using RT1
3. Select the groups for FI using the first ILP model
4. Update RT2
5. Construct fault-type selection ILP model using updated RT2
6. Select the fault types for FI using the second ILP model

In the proposed ILP model, the goal is to select the minimal subset of output pins for FI and achieve the given defect coverage. The ILP model can be easily adapted to different requirement. For example, we can solve the dual ILP problem by maximizing the defect coverage. This can be achieved by adding a constraint on the number of output pins for FI. We can also take the occurrence frequency of internal faults as another cost factor. This can be achieved by adding a weight for each group selection indicator g_i in the objective function. The weights are inversely proportional to the occurrence frequencies of the internal faults. Therefore, the pin that can represent an internal fault with high occurrence frequency will be selected first.

2) Heuristic algorithm (HA)

The increasing complexity of integrated circuits is resulting in a large number of physical defects in a chip or a module within a chip. In order to model the effect of these defects on chip/module pins, the optimization algorithm should be computationally feasible and able to handle large data sets to meet the requirement of state-of-the-art hardware FIT. To model as many defects as possible and handle large data sets, an efficient and scalable heuristic algorithm is presented next to select the subset of groups. The ILP models presented earlier are useful for deriving lower bounds using LP-relaxation, since linear programming problems can be solved in polynomial time [44].

HA Procedure: Select groups and fault types for FI

```
load group-fault versus defect RT $mat(i, j)$
while (#Selected-FI-detectable defects < C)
{
    find the row $t$ with max $sum(mat(:, t)')$;
    for all columns
    {
        if $mat(t, j)$==1
```

```
        $mat(:,j)$=0;
    }
    $mat(t-m:t+m,:)$=0 % delete adjacent rows
    update #Selected-FI-detectable defects
}
```

In the above procedure, we describe a greedy heuristic algorithm to select the groups associated fault types for FIT. The heuristic algorithm is applied to the group-fault versus defect RT. HA first selects the group-fault that can model the largest number of defects, and then deletes the detected defects from the RT; it repeats this "selection-deletion" process to the updated RT until the required selection coverage is achieved. The computational complexity is related to the selection coverage. It is proportional to the square of the number of groups under selection in the worst case.

2.3 Characteristics of Selected Pins

An optimal set of outputs can be selected to represent the internal physical defects by solving an ILP model. Although the ILP model itself is very simple, the coefficients in the constraints are obtained through exhaustive fault simulation. For example, in order to obtain the coefficient y_{ik} in Constraint (2), we need to run simulation in the presence of every defect and record the circuit responses, and then we know if an error can be observed at a specific output when an internal defect exists. Thus we can determine if a pin-level fault can represent this defect. However, this process is time-consuming, since fault simulation needs to be repeated as many times as the number of defects.

Therefore, it is necessary to analyze the underlying reasons why some output pins can represent more internal defects than the others. If some intrinsic characteristics are the reasons that make these pins more effective to represent internal defects, an alternative based on these characteristics can be used to select the proper fault insertion sites instead of exhaustive fault simulation. This will significantly speed up the system reliability measurement process.

To avoid exhaustive fault simulation, three easily obtained characteristics of outputs are analyzed, namely, controllability number, transitive fan-in, and the transition count. Controllability number and transitive fan-in are circuit topology-related characteristics. The transition count is related to the test sequences and obtainable through one-time logic simulation. Definitions and calculation methods for the three characteristics are described as follows.

- Controllability number

  Controllability for a digital circuit is defined as the difficulty of setting a particular logic signal to zero or one [7]. In testability analysis, the advantage of using controllability number is that it involves circuit topological analysis, and the complexity of its calculation is linear in circuit size. A DFT tool, FastScan, is used here to analyze the 0-controllability and 1-controllability by performing good circuit simulation for all gates in the design. Controllability number (either one or zero) is a measure of the times a gate output is zero or one for a given number of random patterns [30].

- Transitive fan-in

  The number of inputs of an electronic logic gate is often referred as fan-in. The calculation of transitive fan-in involves back-tracing from each output pin/port and finding all nodes and branches in their fan-in cones. The backtracing step is automatically executed by a DFT tool (e.g., Design Compiler). The number of transitive fan-in for each output in a circuit can be easily reported by Design Compiler.

- Transition count

  Transition count is a number that records the times transition occurs on a signal during simulation. Two types of transitions, 0→1 transition and 1→0 transition, are taken into account. If either of the transitions occurs at an output during simulation, the transition count of this output increases by one. Transitions at outputs of a circuit can be directly counted by running logic simulation. This characteristic reflects the state transition during the operation of circuits.

A common attribute of these three characteristics is that they are easy to obtain. Otherwise it is meaningless to analyze these indirect characteristics and fault simulation can be used to obtain more accurate results. In the literature, testability measures (controllability and observability), fan-in, transition count and their combinations have been used as metrics for functional test sequences grading[45] and test set enrichment [46]. For our problem, there is no guarantee that an output with a larger fan-in cone can always represent more internal defects. However, if an output has a large fan-in core, it connects to a large number of nodes and branches that can carry errors. Therefore, there is a high probability that it is affected by internal defects.

Here these three characteristics are used as heuristic measures. If the number of defects observed at an output is highly correlated with any of the characteristics or a combination of them, these characteristics can be used as a guideline for the selection of fault insertion points. Such a guided selection is critical for the assessment of large systems, especially when comprehensive fault simulation is not affordable. The correlation between defect distribution at the outputs and these three characteristics is evaluated using the widely used Pearson product-moment correlation coefficient [47]:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\delta_X \delta_Y} \tag{2.12}$$

where the correlation coefficient $\rho_{X,Y}$ is obtained by dividing the covariance of the two variables (X and Y) by the product of their standard deviations.

2.4 Evaluation of Physical Defect Model

The purpose of fault insertion test in system reliability assessment is to verify the robustness and error-handling capabilities of systems by intentionally inserting faults. According to this goal, an effective and efficient fault model used in fault insertion test should have the following attributes:

- Produce a comprehensive set of failures. Therefore, most of failures occurring in the field can be covered.
- Represent as many physical defects as possible. High defect coverage adds credibility to the results of system reliability assessment.
- Manifest in an observable way. In fault insertion test, a large number of inserted faults are masked. These faults are not useful for the evaluation of error-handling mechanism.
- Manifest with low latency. This is especially important in a simulation environment. Simulation of functional tests in a large system can take several weeks, even months. Detecting an error early during simulation can significantly reduce simulation time.

In order to evaluate the effectiveness of the proposed model in terms of representing physical defects, we compare the system responses to the proposed defect model with system responses to real physical defects. If the proposed model can produce the same erroneous responses to that caused by physical defects, this model is considered to be successful in terms of representing defects.

System responses can be defined from various perspectives. Here the system responses are analyzed using two levels of granularity. First, the program execution flow is used to evaluate the impact of the model on system operation. Taking a CPU as an example, the execution of instructions may fall in an infinite loop if the program counter is faulty, or the execution may halt due to an invalid signal. Without exploring the details of failures, the impact on program execution flow is used for coarse-grained evaluation. Second, to analyze the responses of the proposed model to that of defects in a fine-grained manner, faulty registers in both cases are compared. Assuming that a number of key registers are observable in a design, if we observe that the faulty registers in the presence of the proposed defect model are the same as the faulty ones in the presence of physical defects, the erroneous responses are considered to be the same.

According to the attributes of an effective model, the evaluation of the physical defect model also includes the rate of error occurrence and the error detection-latency. The rate of error occurrence is obtained by dividing the number of FIs that lead to an observable error by the total number of FIs. Error detection-latency is an interval between the cycle of fault insertion and the first cycle when an error is observed. In order to assess the error-handling mechanism in a system, a fault must first be introduced. However, a large number of faults inserted to the system are often masked. Therefore, a defect model with high rate of error occurrence can improve the efficiency of the assessment of error-handling mechanism, and a defect model with low error detection-latency can accelerate system reliability measurement.

2.5 Simulation Results

In this section, simulation results for a USB circuit and an open-source SoC are presented. These two circuits represent modern communication and digital processing systems. For the USB circuit, defect distribution at output pins is first illustrated, and then the tradeoff curves between the number of selected output pins and selection coverage are provided. Next, the selection results related to FI locations and fault types are presented, and three characteristics of outputs are analyzed. Subsequently, experimental results on an open-source SoC OR1200 are provided to verify the effectiveness of the proposed model in terms of the impact on system response, the rate of error occurrence, and the error detection-latency. Comparisons are made among the FI based on the proposed model, FI based on randomly selected faults, and FI based on real defects. All experiments were performed on a pool of state-of-the-art servers running Linux. The Verilog simulator for all simulations is VCS. The ILP model was solved using lpsolve [42]. DFT tools (FastScan and Design compiler) are used to calculate the controllability number and transitive fan-in.

2.5.1 Results for the USB Circuit

In the USB circuit, there are 248 inputs, 116 outputs, and 7110 internal nets at the gate level. Functional tests are from OpenCores [48]. In order to analyze the manifestation of physical defects on the output pins, various logical abstraction of defects, including s-a-0, s-a-1, flip, bridging, glitch, and delay, are inserted to all the internal nodes. One internal fault is inserted at a time. The simulation time for insertion of all the internal faults is nearly 10 hours. The preparation of the relationship table takes 10 minutes. Following this, the ILP solver takes only 1 minute to complete the selection. The heuristic algorithm runs about 2 times faster.

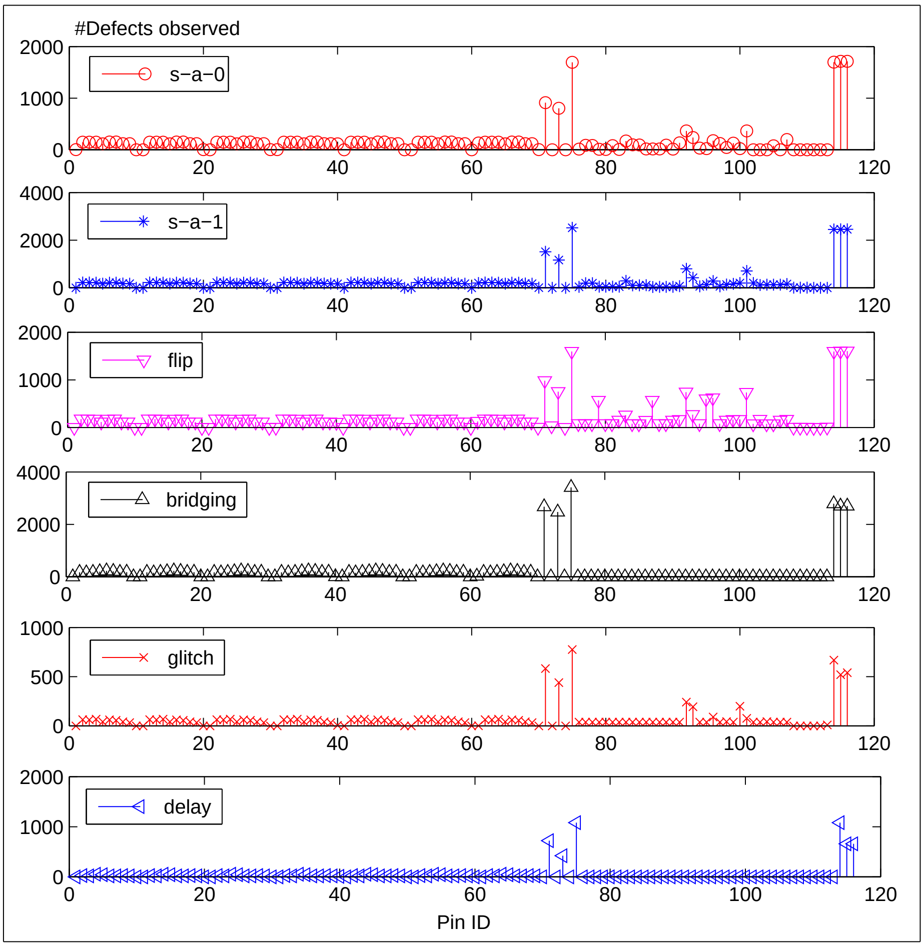


Figure 2.4: USB benchmark: error distribution on output pins.

The distribution of defects observed at output pins is shown in Fig. 2.4. The X-axis stands for the identifier of output pins. The Y-axis stands for the number of internal faults (defects) captured by the corresponding output pin. In Figure 2.4, we can see that the number of defects captured by each output pin is different, which shows that there is significant difference between the output pins in terms of their ability to capture defects. This skewed distribution provides valuable evidence that it is meaningful to select the most effective set of output pins to represent defects; otherwise, if the distribution is uniform, there is no need to select an optimal set of output pins, because all the outputs are equivalent in respect of representing internal defects. Moreover, we find that the distributions for different types of defects are similar.

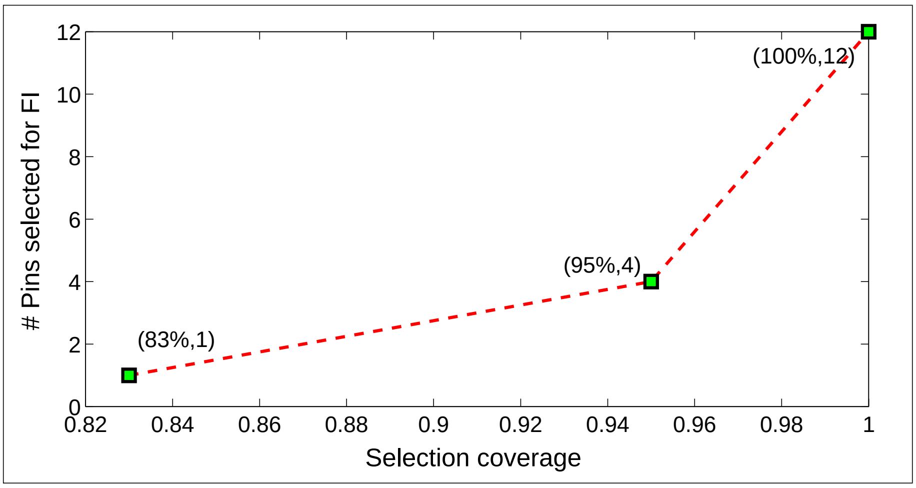


Figure 2.5: Tradeoff between #FIs and selection coverage.

The tradeoff curve between the number of selected output pins and selection coverage is shown in Fig. 2.5, The curve is obtained by setting different selection coverage values in the ILP model and calculating the minimum number of output pins for each case. This trade-off graph provides guidance for the number of insertions are needed to achieve the desired coverage level, and for a fixed number of insertions, what level of selection coverage can be obtained. The total number of defects is 7,471. The type of defects includes s-a-0, s-a-1, flip, bridging, glitch, and delay. In Fig. 2.5, we can see our selection method is very effective. With only one FI logic element, we can obtain 83% selection coverage. That most effective pin is pin$_{115}$, which is the most suitable for defect modeling via FI. In Fig. 2.4, we can see that nearly 1,600 internal stuck-at faults are manifested themselves at pin_{115}. However, there are no more than 200 stuck-at faults showing up at any pin between pin_1 and pin_{70}. Assume that we randomly select one pin from pin_1 to pin_{70} for FI. Compared to this random selection, the proposed selection technique is about 8 times more effective, in terms of the number of defects that can be represented.

The single-pin and group selection results using the ILP-based and HA methods are shown in Table 2.3. The selection coverage is 100%. The fault period is set to 8 clock cycles. In Table 2.3, the selection results are based on the insertion of five types of faults, including stuck-at, flip, bridging, glitch, and delay. A total of 7,471 internal faults are included in the RT. Table 2.3 shows the most proper FI points and the associated fault types for each case. For example, the second column indicates that 12 output pins are selected for single pin selection, namely, pin_{70}, pin_{71}, pin_{73}, pin_{75}, pin_{92}, pin_{95}, pin_{96}, pin_{99}, pin_{108}, pin_{109}, pin_{114}, and pin_{115}. Pin-s-a-0 fault type is assigned at pin_{75} for FI. Often fewer pin-flip faults are suggested, since the cost of pin-flip-insertion is higher. The third column shows the group-selection results obtained using the ILP model for group size m equal to 2. In group selection, group i consists of pin_i and pin_{i+1}, if the group size is 2. The fourth column lists the ILP selection results with group size equal to 3. The fifth column lists the HA selection results. The bottom row shows the total number of pins that are selected for FI. For example, 13 pins are needed to cover all the defects in the HA-based selection results. The number of pins selected by HA is slightly greater than that obtained by ILP model.

In order to analyze the difference in selection results for different types of internal faults, we construct RT with s-a-1, s-a-0, and flip faults, instead of all five types of faults, and repeat the experiments. The number of internal faults in the RT is

Table 2.3: FI selection results for the USB circuit: Part 1.

	ILP			Heuristic
	Single pin	Group in 2	Group in 3	Single pin
Pin-s-a-0	75,115	74,114	73,113	75,92,115
Pin-s-a-1	70,71,73,75 92,95,96,99 108,109	70,72,74 91,95, 98,108	69,73,89, 94,96,107	73,75,101 113
Pin-flip	75,114,115	74,114	73,113	70,92,95,96, 98,99,108, 109,114,115
#FIs	12	8	7	13

Table 2.4: FI selection results for the USB circuit: Part 2.

	ILP			Heuristic
	Single pin	Group in 2	Group in 3	Single pin
Pin-s-a-0	115	114	113	75,115
Pin-s-a-1	70,71,75,95,96, 99,108,109,114	70,74,95, 98,108	69,73,94, 97,107	71,75
Pin-flip	115	114	113	70,92,95,96, 98,99,108, 109,114,115
#FIs	10	6	6	12

5,302. In Table 2.4, selection results are presented in the same way as that in Table 2.3. For the single pin selection, a total of 10 output pins are needed to represent 5,302 internal faults. Compared to Table 2.3, pin_{73} and pin_{92} are excluded, while they are necessary in the representation of bridging, glitch, and delay faults. The selection coverage in this experiment is 100%, therefore the selected 10 pins can represent all the 5,302 internal faults. However, if we consider all the 7,471 faults (5 types), the selection coverage is only 91%. Due to the absence of pin_{73} and pin_{92}, 694 internal faults cannot be represented (covered).

For cross-validation purpose, another functional test is applied to the USB circuit. The goal is to verify that the selection results exhibit a fair degree of pattern independence. Experiments are repeated in the same manner and the selection results are similar. For single-pin selection, pin_{70}, pin_{71}, pin_{75}, pin_{85}, pin_{96}, pin_{99}, pin_{108}, pin_{109}, pin_{114}, pin_{115} are selected for FI. In the ten selected pins, only one pin is different from the previous selection results.

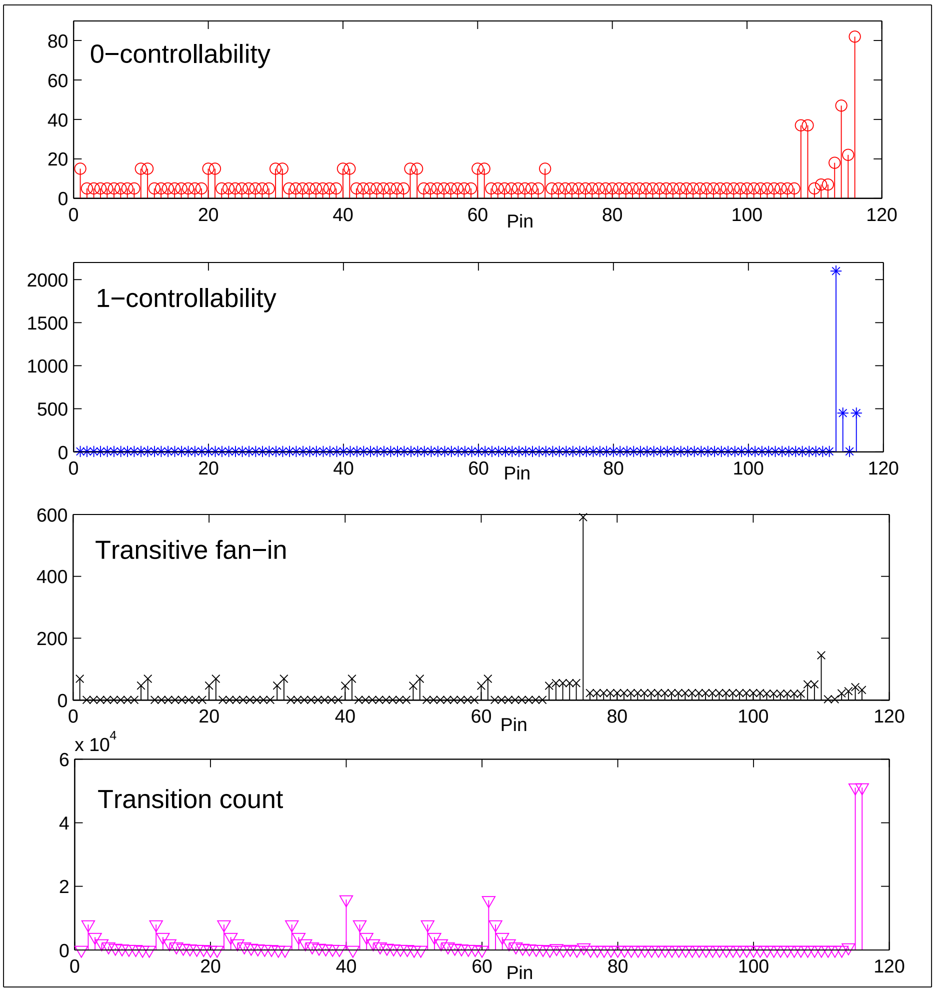


Figure 2.6: The distributions of three related factors in the USB circuit.

In Fig. 2.6, the distributions of three characteristics of output pins are illustrated. From top to bottom, the characteristics are 0-controllability, 1-controllability, transitive fan-in, and transition count. The controllability measure is separated into 0-controllability and 1-controllability. The X-axis stands for the output pin ID; the Y-axis stands for the corresponding characteristics. Comparing Fig. 2.6 to

Fig. 2.4, we can observe some correlations between the number of defects observed at a pin and the three characteristics of the pin. For example, in the subplot of transitive fan-in in Fig. 2.6, pin_{75} has the largest number of transitive fan-in, and coincidentally we find the number of defects observed at pin_{75} is the largest for all the three types of defects in Fig. 2.4. In the subplot of transition count in Fig. 2.6, the last two pins (pin_{115} and pin_{116}) have the largest transition count, and in Fig. 2.4 the number of defects observed at these two pins are fairly large.

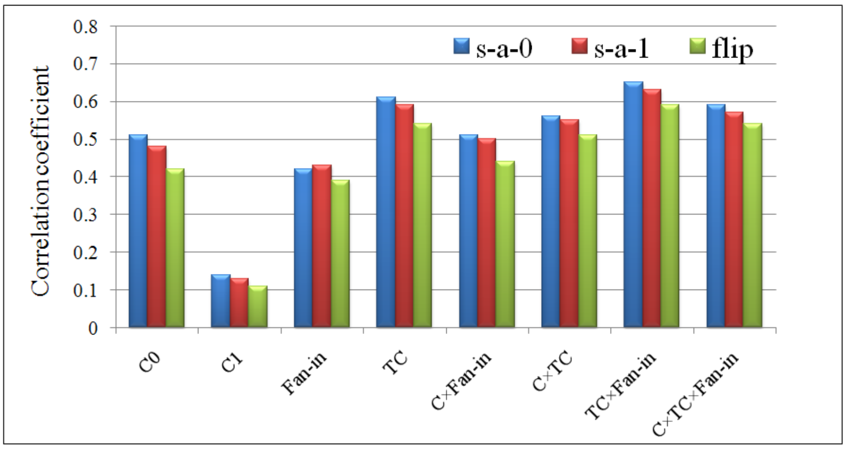


Figure 2.7: Correlation coefficients for the USB circuit.

To quantify the correlations between the number of defects captured at a pin and the characteristics of a pin, the Pearson product-momentum correlation coefficients are calculated and shown in Fig.2.7. In Fig. 2.7, the X-axis stands for the characteristics of a pin, including 0-controllability (C0), 1-controllability (C1), transitive fan-in, transition count (TC), and the combination of these characteristics. The correlation calculation is between these characteristics and three types of defects (s-a-0, s-a-1, and flip). The Y-axis stands for the values of correlation coefficients. The series denoted in blue are the correlation coefficients between s-a-0 defects and all the characteristics and their combinations. The series denoted in red are the correlation coefficients between s-a-1 defects and the characteristics.

Likewise, the series denoted in green are the correlation coefficients between flip defects and the characteristics. We can see that for any given characteristic, the correlation coefficients for different types of defects are nearly the same, since the distributions of three types of defects at output pins have the similar trends; see Fig. 2.4. In all the characteristics and their combinations, the combination of TC×Fan-in is most correlated to the number of defects observed at pins, where the correlation coefficient is greater than 0.6, and the 1-controllability is the least correlated one. If we rank the output pins by the size of fan-in cone, and select the top largest 10 pins. The selection results are pin_1, pin_{11}, pin_{21}, pin_{31}, pin_{41}, pin_{51}, pin_{61}, pin_{73}, pin_{74}, pin_{75}, pin_{110}. The defect coverage by these 10 pins is 82%, assuming that all the three fault types can be inserted at these pins.

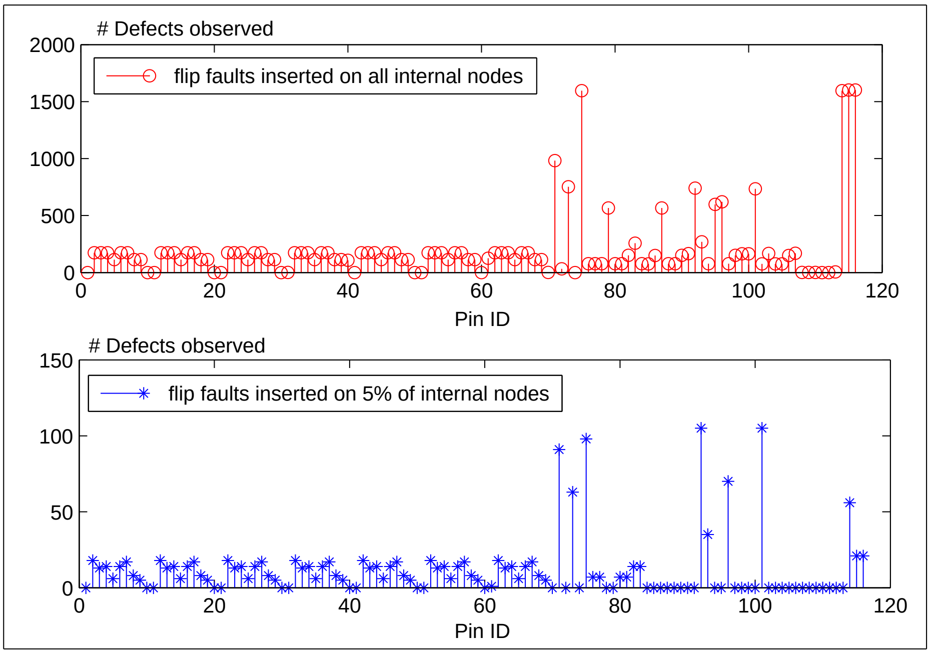


Figure 2.8: Manifestation of flip faults on output pins.

The characteristics of output pins provide guidance on the selection of the most representative pins from another perspective, that of bypassing the fault simulation step. Besides the analysis of characteristics of pins, an alternative for accelerating model construction is to randomly select a small number of internal nodes for fault insertion (FI), instead of injecting faults on all the internal nodes. Taking the flip fault as an example, we compare the distribution of observed defects based on FI on all the internal nodes to FI on randomly-selected 5% of internal nodes; see Fig. 2.8. We find that the output selection results for these two types of insertions are similar. The correlation coefficient between the distribution of flip fault insertion on all internal nodes and on the 5% internal nodes is 0.67. Therefore, when FI on all the internal nodes is not feasible, we can consider inserting faults on a much smaller number of randomly selected internal nodes for model construction.

However, the distribution of observed defects on the outputs varies, when different subsets of internal nodes are selected for FI. We carried out a set of experiments where a total of 30 subsets of internal nodes were selected for flip fault insertion. In each subset, the number of internal nodes is 355, which is 5% of all the internal nodes. The correlation coefficient between the defect distribution with FI on 5% of the nodes and that with FI on all the nodes varies from 0.43 to 0.87. Pin selection results based on different RTs are listed in Table 2.5. Comparisons in pin selection results and defect coverage are made based on three RTs. One RT is constructed by FI on all the internal nodes. The other RTs are constructed by FI on the 5% randomly selected internal nodes. The inserted fault type is flip. For each ILP model, the coverage of the defects recorded in the RT is set to be 100%. Based on this constraint, the pin selection results are shown in the second column in Table 2.5. If the RT is constructed based on all the defects, the coverage of all the defects shown in the fourth column is 100%. When the RT is constructed based on part of defects, the coverage of all the defects drops, even though the selected pins can represent all the defects recorded in the RT. Therefore, when the runtime needed for FI is affordable, it is better to exercise the system with comprehensive FI, in order to construct an accurate physical-defect model.

Table 2.5: FI selection results based on different relationship tables.

	Pin selection results	Coverage of the defects recorded in the RT	Coverage of all the defects
RT with all defects	70,71,75,95 96,99,114 115	100%	100%
RT with 5% defects (random selection 1)	70,71,85 102	100%	85%
RT with 5% defects (random selection 2)	71,73,75 109	100%	73%

2.5.2 Results for SoC OR1200

The OR1200 is a 32-bit RISC with the Harvard microarchitecture, 5 stage integer pipeline, virtual memory support (MMU) and basic DSP capabilities [49]. It consists of a CPU, memory management units, data cache, instruction cache, flash memory, SRAMs, audio/video/ethernect connections, etc. The system block diagram is shown in Fig. 2.9. The CPU block is the critical part in this system. Faults are inserted to the nets inside the CPU and the output pins. A functional test (Cbasic) in use is derived from design verification programs.

The goal of experiments on this system is to compare the system responses in the presence of proposed defect model to that in the presence of the physical defects. Comparisons are made among three fault insertion scenarios, namely, FI on internal nets, FI on random outputs, and FI on selected outputs. The number of fault insertions for these three different sets of locations is the same, which is 1151. For the scenario of FI to internal nets, one flip fault is inserted to one internal net. There are 1151 internal nets inside the CPU. For the scenario of FI to selected

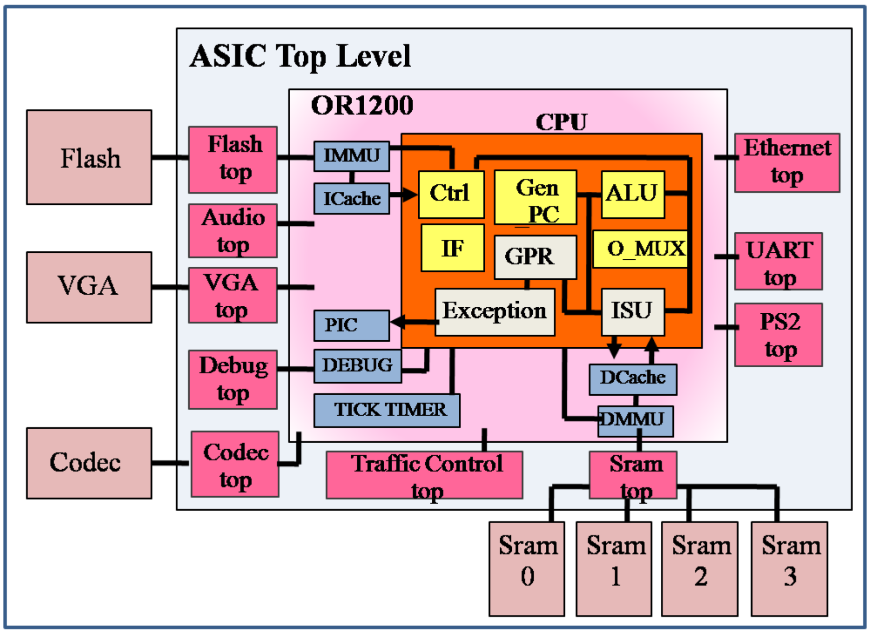


Figure 2.9: Block Diagram of the OpenRISC 1200 SoC.

outputs, more than 20 faults are inserted to each of pins in the selected 71 pins. The total number of fault insertion is 1151. There are 394 output pins in the CPU module. A total of 71 outputs are selected to represent all the FI-detectable defects inside the CPU. The associated fault types include pin-stuck-at-0, pin-stuck-at-1, and pin-flip. Fault period is 8 clock cycles. For the scenario of FI to random outputs, 71 output pins are randomly selected. More than 20 faults are inserted for each pin, and the total number of fault insertions is 1151 as well.

First, we compare the impact on program execution flow for the three FI scenarios. The program execution can be either a non-stop flow or a stalled flow in a failing system. In the operation of this RISC system, a non-stop flow can be caused by a faulty program counter or a stuck exit signal. A stalled flow can be caused by an invalid signal or a faulty freeze signal. In the 1151 FIs to internal nets, 91.2% of the FIs have no impacts on the execution flow, which means these faults inserted on the internal nets are masked; 4.9% of the FIs make the execution stalled and 3.9%

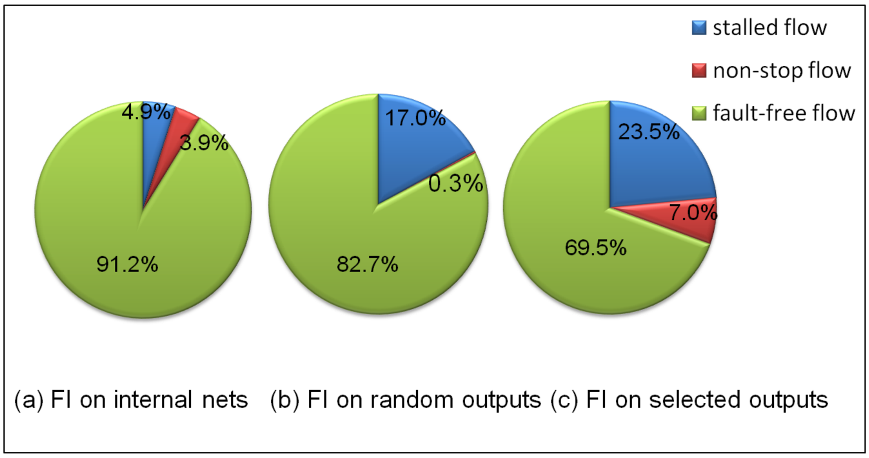


Figure 2.10: Impact on the program execution flow.

of the FIs lead to non-stopped flows. In Fig. 2.10, a pie chart show the portion of three types of execution flows for each FI scenario. Two important observations are made from this pie chart: 1) The rate of error occurrence (non-stop flow and stalled flow) caused by FI at the pin level is higher than that caused by FI on the internal nets; 2) Two types of faulty program execution flows caused by internal defects can be produced by the proposed defect model. Moreover, compared to the scenario of FI on random outputs, FI on the outputs selected using proposed method provide more faulty flows.

Second, we evaluate the impact on the logic values of registers for the three FI scenarios. Assume that 36 key registers (general purpose registers, control and status registers) are observable in this design. In this comparison, we record the faulty register ID in each fault insertion and accumulate the counts. In Fig. 2.11, the X-axis stands for the register ID from 1 to 36; the Y-axis stands for the number of FIs that lead the register faulty. From top to bottom, the results for FI on selected pins, FI on random pins, and FI on internal nets are listed, respectively. We can see that out of the 1151 FIs for the three scenarios, register_1 to register_{15} are faulty more than 400 times when faults are inserted at selected pins, while they

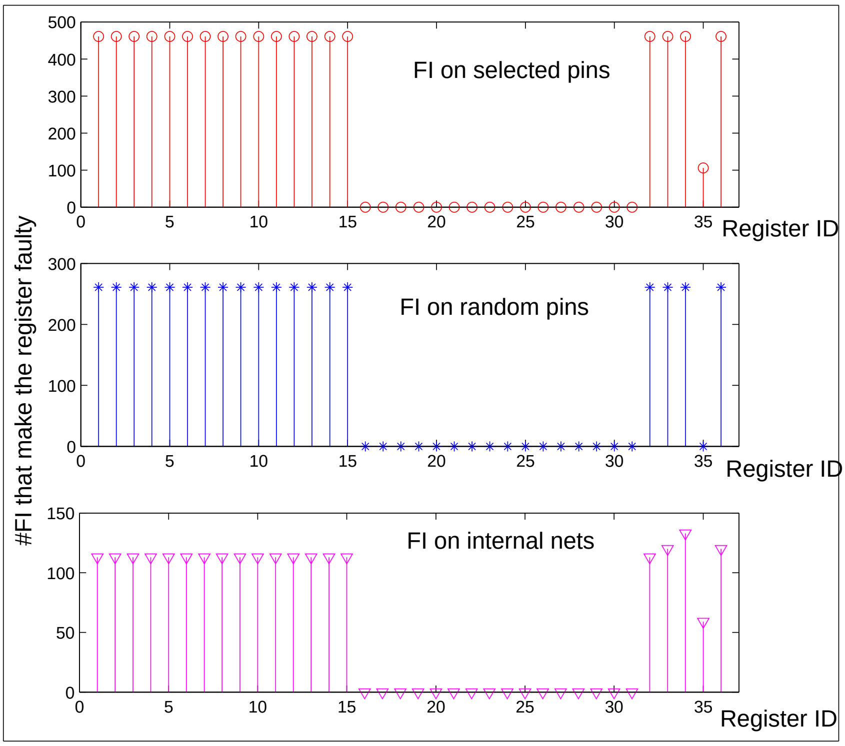


Figure 2.11: Impact on the correctness of the registers.

are faulty less than 150 times for the scenario of FI on internal nets. The rate of error occurrence for FI on selected pins is nearly 3 times higher than that for FI on random pins. Moreover, all the registers that get faulty values in the presence of internal defects are corrupted by the proposed model. Register$_{35}$ is missing for the scenario of FI on random pins, i.e. faulty values can be observed on register$_{35}$ in the presence of internal defects, but they cannot be observed by inserting faults on the randomly-selected outputs. From this point of view, effects of defects are more accurately mimicked by the proposed defect model.

Third, we compare the error detection latency for the three FI scenarios. The error detection latency is the interval between the cycle of fault insertion and the first cycle when an error is observed. The observation points are the 36 registers described above. If an error is observed earlier, the simulation can be terminated.

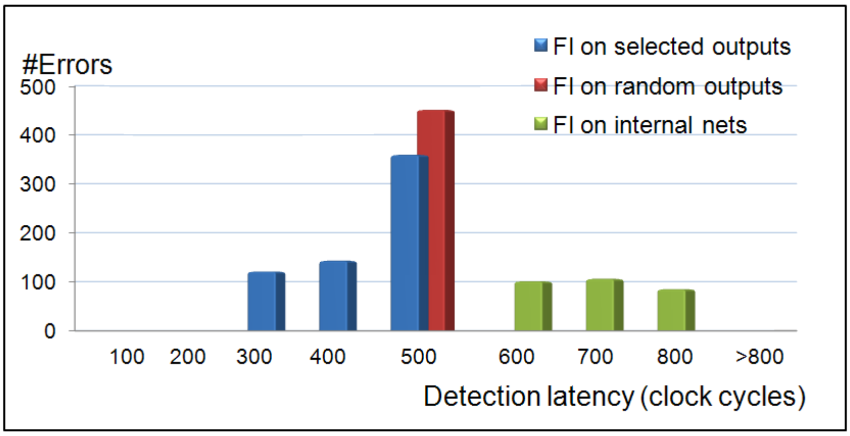


Figure 2.12: Impact on error detection latency.

This is very important in the simulation of large systems, since functional test in a large system can run for several weeks [33]. In Fig. 2.12, an error distribution is shown in terms of error detection latency. The X-axis stands for the error detection latency specified in clock cycles. The Y-axis stands for the number of errors that can be detected in a certain period. Three sets of data, namely, FI on selected outputs, FI on random outputs, and FI on internal nets, are denoted in blue, red and green, respectively. For example, the first column (from left to right) in Fig. 2.12 indicates that 121 errors caused by FI to selected outputs are detected within 300 cycles. From the distribution, we can see that the error detection latency caused by FI on the pin level is lower than that caused by internal defects. This is expected, since it takes time for errors to propagate from internal nets to output pins.

From the system simulation results for the SoC OR1200, we find that the same impact on a system caused by real physical defects can be produced using the proposed physical defect model. In a coarse-grained manner, the same types of faulty program execution flows can be reproduced. In a fine-grained manner, the registers corrupted by the physical defects can be corrupted by the proposed defect model. In addition, errors caused by the pin-level faults can be detected earlier than those caused by internal defects. Moreover, the faults inserted at the outputs selected by our optimization method are more effective than the faults inserted at the randomly selected outputs in terms of the above three metrics.

2.6 Summary

In this chapter, a simulation framework and optimization techniques have been presented to guide the selection of several important parameters in hardware FIT, such as the number of FI logic elements, FI-defect coverage, fault period, and the group size of pins. The selection method utilizes ILP-based optimization and a greedy heuristic algorithm to determine the optimal FI locations and fault types to mimic the effects of the defects within the module. From an analysis of the intrinsic characteristics of outputs, the pins with large transitive fan-in and high transition count are found to capture a large number of internal defects. Fault-insertion logic elements are suggested to be first placed on these outputs. The effectiveness of the proposed physical defect model is verified in terms of impact on system response, the rate of error occurrence, and the error detection-latency. The efficiency of FIT-based system reliability test can therefore be improved using the proposed effective physical defect model and optimization method.

Chapter 3

Diagnosis Framework using FIT and Bayesian Inference

Fault diagnosis is necessary to identify the root cause of a malfunction circuit by collecting and analyzing the failure information. To locate faults in small circuits, we can do look-up using a fault dictionary and find the fault based on the error pattern, often referred to as the fault syndrome. Typically, a number of faults and the corresponding fault syndromes are recorded in a fault dictionary, which can be constructed by fault simulation. However, increasing integration densities and high operating speeds are leading to a large number of faults and syndromes, especially at the board and system level. Moreover, the syndrome of a fault may not always be the same, due to the subtle manifestation of manufacturing defects. Functional failures sometimes are not irreproducible. Therefore, a probabilistic approach is needed to model and overcome the uncertainties involved in functional diagnosis.

In this chapter, we develop a diagnosis framework based on fault-insertion test and Bayesian inference, which allows us to identify faulty devices or faulty modules within a device on a failing board with high confidence. Bayesian inference is a powerful probabilistic method for pattern analysis, classification, and decision making under uncertainty. Fault-insertion test is used to create sample faulty scenarios, and then calculate conditional probabilities. Results on a case study using an open-source RISC system-on-chip highlight the effectiveness of the proposed framework in terms of fault-localization accuracy and correctness of diagnosis.

3.1 Prior Work

Diagnosis problems at board and system level test and diagnosis are more challenging than that at chip level, as discussed in Chapter 1. Most diagnosis strategies published in the literature focus on failures that occur in scan (structural) test [50], [51], [52], but the diagnosis of functional failures has received much less attention. There is an increased focus on inference-based diagnostic techniques targeting functional failures. A number of inference-based diagnostic techniques have been proposed, e.g., model-based inference and Bayesian inference.

In [33], a model-based inference engine was built on the basis of the correct operation of a device, and enhancement to the basic inference engine were presented as well. For model-based inference, it is difficult to adjust the model, if the inference was initially incorrect. Today it is very challenging to build an accurate model to represent a complex electronic system. In [34], an automated diagnosis system named MonteJade was developed based on a combination of model-based and probabilistic approaches.

In [53], a Bayesian-based incremental approach to functional diagnosis was presented. It is used to identify candidate failing components based on the pass/fail information of the executed tests. It suggests the next test to be run, to identify the faulty component by analyzing the achieved results. However, in today's manufacturing testing line, it is often the case that the failed board has to be repaired first, and then the next test can be run. In addition, sufficient diagnostic programs/tests, which can be used to further locate the root cause, are often unavailable.

In [54], a diagnostic framework using Bayesian statistics was proposed for parametric fault diagnosis in analog circuits. A sensitivity guided test input selection scheme was used to determine the measurement attributes that are most likely to distinguish among the faults. One assumption was that the distribution of the faulty value of a parameter is Gaussian. Thus the probabilities used in these calculations are neither realistic nor derived from real sample space.

In this work, we take the advantage of fault-insertion test to create faulty samples. Therefore, the occurrence probability of a syndrome given a specific fault can be calculated based on the actual database (failure logs). The sample size can be determined based on the needs of diagnostic accuracy. Fine-grained syndromes are recorded during FIT, instead of the traditional pass/fail results. According to the adaptive feature of Bayesian inference, the probability of a fault candidate being the root cause of failure can be updated when new syndromes are available. There is no constraint on the number of syndromes. Therefore, the proposed diagnostic framework can handle those scenarios where syndromes are incomplete.

The remainder of this chapter is organized as follows. Section 3.2 provides the basics of Bayesian inference and its application to board-level diagnosis. Section 3.3 presents the development of the diagnosis framework and potential directions for improving diagnostic resolution. Section 3.4 presents the design of experiments and experimental results for Open RISC 1200 system-on-chip (SoC). Section 3.5 concludes this chapter.

3.2 Bayesian Inference and Its Application to Fault Diagnosis

Bayesian theory provides a unified and intuitively appealing approach for drawing inferences from observations and *a priori* beliefs. It builds on Bayes' formula, shown in (3.1), in which F_1 through F_q are a set of mutually exclusive and jointly exhaustive events. The parameter R_k is an event that depends on the occurrence of F_1,...,F_q. $P(F_1)$ through $P(F_q)$ are probabilities (*prior*s) of the occurrence of each event F_1 through F_q. *Prior*s are *a priori* beliefs obtained before observation on event R_k is made. The conditional probability $P(R_k|F_i)$ is the probability of occurrence of event R_k given that event F_i occurs. The occurrence probability of event F_i given that event R_k occurs is denoted by $P(F_i|R_k)$. This probability is defined as *posterior*, which depends on the *a priori* belief of the occurrence probability of event F_i and the observation on event R_k.

$$P(F_i|R_k) = \frac{P(R_k|F_i)P(F_i)}{P(R_k)} = \frac{P(R_k|F_i)P(F_i)}{\sum_{j=1}^{q} P(R_k|F_j)P(F_j)} \tag{3.1}$$

Bayesian inference has been successfully used in power-circuit and analog-circuit diagnosis [54, 55]. In this chapter, we show how to apply Bayesian inference to board-level diagnosis. Suppose that we want to locate the faulty ASIC in a malfunctioning board. To apply Bayesian inference to this practical problem, let F_1 through F_q in (3.1) be a set of faults at the pin level of ASICs. These faults are candidates for the root cause of board failure. Let R_k be the kth observation point. Specifically, observation points are observable registers (e.g. CSRs) on the board. The artificial faults and the observation points are denoted on a real board shown in Fig. 3.1. Note that *prior* $P(F_i)$ is the *a priori* occurrence probability of fault F_i; $P(R_k|F_i)$ is the probability of an error occurring on the kth register given that fault F_i occurs. $P(F_i|R_k)$ is the occurrence probability of F_i given an error in the kth register. This is the *posterior* based on the observation on the kth register.

To understand this inference process better, suppose that there are two possible faults (F_1, F_2) on a failing board and one register (R_1) is observable. The problem is to determine the more likely fault resulting in the board failure (out of F_1 and

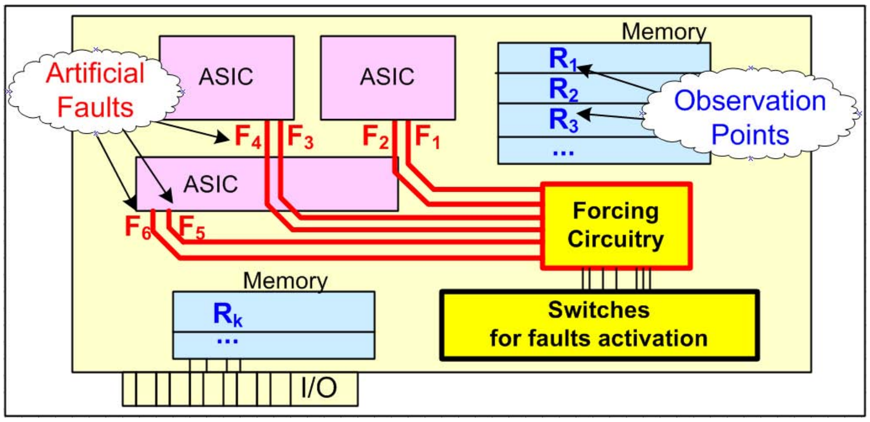


Figure 3.1: FIT for Bayesian-Inference Application.

F_2). Assume that when fault F_1 occurs, the occurrence probability of error on register R_1 is 0.8; when fault F_2 occurs, the occurrence probability of error on register R_1 is 0.4. Note that we find an error on register R_1 on the actual failing board. In this scenario:

$P(F_1) = P(F_2) = 0.5$

$P(R_1|F_1) = 0.8,\ P(R_1|F_2) = 0.4$

$P(F_1|R_1) = \frac{P(R_1|F_1)P(F_1)}{P(R_1|F_1)P(F_1)+P(R_1|F_2)P(F_2)} = 0.67$

$P(F_2|R_1) = \frac{P(R_1|F_2)P(F_2)}{P(R_1|F_1)P(F_1)+P(R_1|F_2)P(F_2)} = 0.33.$

Therefore, the probability that F_1 occurs is 0.67 and the probability that F_2 occurs is 0.33. Based on the given conditional probabilities and the observations from the bad board, we infer that fault F_1 is the more likely suspect fault causing board failure.

In Bayesian inference-based diagnosis, we initialize the *prior* for each fault in an equiprobable manner, i.e., $1/q$. Parameter q is the number of all the possible faults. There is only one observable register in the above example. When multiple registers are observable, the *posterior* is updated iteratively by observation on each register. The calculated *posterior*, based on the observation of register R_k, is used as the *prior* of the next calculation based on the observation of register R_{k+1}. Thus, there is no requirement on the number of observation points (observable registers). Decisions can be made based on existing observations and when new observation is available, the inference results can be updated. This adaptive attribute is one of the main advantages of Bayesian inference. Another advantage of using Bayesian inference in diagnosis is efficiency, since it eliminates the debug time for devices that are determined to be fault-free by the inference engine. Moreover, an inference-based engine such as this allows for an automated diagnostic process. Assuming that the Bayesian framework is accurate, diagnosis results can be directly gathered by sending the fault syndrome and database to the inference engine. The fault syndrome refers to the observations on registers from bad boards, and database includes all the conditional probabilities needed in the *posterior* calculation, which are learned from fault-insertion test.

3.3 Diagnosis Framework

The Bayesian inference based diagnostic framework consists of a learning step and the diagnosis step. In the learning step, pin-level faults are intentionally inserted to a fault-free (reference) board. The inserted faults and corresponding behaviors of the board are recorded. Conditional probabilities ($P(R_k|F_i)$) are computed and saved in a database. In the second step, according to the conditional probabilities in the database and bad-board syndrome, the probability of each artificial fault occurring or being the root cause is inferred using Bayes' formula and the fault with the highest occurrence probability is deemed to be the most-likely candidate. Therefore, the ASIC where the most suspect fault resides is considered to be the faulty ASIC leading to board failure. This framework can be hierarchically used to diagnose faulty components inside ASICs as well.

Table 3.1: An example of fault syndromes.

	F_1	F_2	F_3	F_4
R_1	1	0	1	1
R_2	1	1	1	1
R_3	0	1	0	1
R_4	0	0	0	1
R_5	1	1	1	0

3.3.1 Probability Preparation using FIT

The purpose of the learning step is to study board responses to the artificial faults. In (3.1), we can see that the conditional probability $P(R_k|F_i)$ is the key to *posterior* calculation. The computation of this conditional probability is the main task in the learning step. More details of the conditional probability calculation are presented below.

In FIT, an artificial fault can be inserted on the board by turning on fault-insertion logic. During fault simulation, logic values of observable registers are recorded, and a set of comparison results of register values is called the fault syndrome. The logical values of registers in fault simulation are compared with golden values. For each register, the comparison result is either a match or a mismatch. Specifically, if there is a match, we record a "0"; if there is a mismatch, we record a "1". Thus, the fault syndrome can be denoted by a binary vector. The length of this vector is equal to the number of observable registers l. Therefore, the number of all possible comparison outcomes is 2^l. A fault syndrome table is shown in Table 3.1. There are 5 observable registers on the board and four pin-level faults are inserted. For example, when pin-level fault F_1 is inserted, the fault syndrome is (1,1,0,0,1), which means that detectable errors appear in register R_1, R_2 and R_5 in the presence of pin-level fault F_1.

Thus far, fault syndromes in the presence of different pin-level faults have been

Table 3.2: Database of conditional probabilities.

	F_1	F_2	F_3	F_4
R_1	90/100	2/100	99/100	71/100
R_2	82/100	68/100	21/100	71/100
R_3	0/100	92/100	10/100	65/100
R_4	4/100	0/100	7/100	100/100
R_5	86/100	46/100	51/100	30/100

obtained, but we cannot conclude that the probability of error occurring on register R_1 is 1 when pin-level fault F_1 occurs. The reason for this is that the effect of defects on modern boards is subtle and error responses are not always reproducible. To get an accurate error-occurrence probability, fault simulation needs to be run multiple times and an average is used to compute the conditional probability. Therefore, we repeat fault simulation multiple times, and accumulate the count of errors occurring on each register. The relative frequency of error is equal to the count of error occurrence divided by the total number of simulations. When the simulation count is large enough, our premise is that the relative frequency of an error converges to the corresponding conditional probability. Thus, the entry on the kth row and ith column in Table 3.2 is the conditional probability $P(R_k|F_i)$, assuming that the number of simulations is large enough. In the first step, by recording the fault and corresponding syndrome, the database of conditional probability is obtained. This database is reusable for the diagnosis of different failing boards.

3.3.2 Root Cause Inference

In diagnosis step, the probability of each fault being the root cause is inferred using Bayes' formula. According to the fault syndrome on the bad board, if an error occurs on register R_k, the possibility of fault F_i to be root cause can be directly calculated using (3.1). The conditional probability $P(R_k|F_i)$ can be computed from the database constructed in the first step. If no error occurs on register R_k, the probability that fault F_i is the root cause is calculated using (3.2):

$$P(F_i|\overline{R_k}) = \frac{P(\overline{R_k}|F_i)P(F_i)}{\sum_{j=1}^{m} P(\overline{R_k}|F_j)P(F_j)} \tag{3.2}$$

where $P(\overline{R_k}|F_i) = 1 - P(R_k|F_i)$. The probability $P(F_i|\overline{R_k})$ denotes the occurrence probability of F_i when the observation from register R_k does not contain an error. The computation is similar for the case of error occurrence.

Taking the advantage of the adaptive attribute of Bayesian theorem, we can update the *posterior* when new observations are made. Therefore, if the number of observable registers on the board is q, the *posterior* of each fault will be updated q times. The *posterior* in the final update is considered to be the probability of the fault being the root cause. In the first round, the *prior* is initialized in an equiprobable manner. Afterwards, the *prior* is updated by the *posterior* calculated in the previous round.

Fault syndromes for a failing board might also not be reproducible due to the subtle manifestations of transient faults. To describe the fault syndrome of the failing board more realistically and get accurate diagnosis, simulation on the failing board is performed multiple times as well. The final *posterior* is the average of all the *posterior* computed from each fault syndrome.

Due to the lack of observability and limited sample size, some faults cannot be distinguished from one another, resulting in an ambiguity group. In order to retain diagnosis accuracy, we not only consider the fault with the highest *posterior* as the root cause, but also several faults with high *posterior* values. A discussion on how to improve diagnosis accuracy and shrink the ambiguity group size is presented in Section 3.4.

3.4 Simulation Results

To evaluate the proposed framework, experiments are performed on the OpenRISC 1200 SoC [49]. The CPU block is the critical part in this system. Therefore, artificial faults are inserted at the output pins of six functional modules inside the CPU, namely ALU, Ctrl, Insn_fectch, Operand_MUX and Gen_PC, as shown in Fig. 3.2. These six modules are the suspect modules causing system malfunction. We assume that registers in six memory units are observable, namely IMMU, DMMU, DC_top, IC_top, register files and special purpose register (SPR). Four functional tests, *Dhry*, *Basic*, *CBasic* and *Multi*, are pre-loaded in flash for diagnosis. *Dhry* is a synthetic benchmark workload, *Basic* and *Cbasic* include most basic instructions, and the test *Multi* mainly consists of multiply instructions [49].

In our experiments, 23 pins from six functional modules are selected to insert faults. Some of the pins are randomly selected bits of data buses and other pins are flag signals. More details of the pin-level fault selection technique are presented in [56]. The fault syndrome is a combination of observations from 17 registers. Fault simulation with pin-level faults is used for constructing the database of error-occurrence probabilities for a specified pin-level fault. A malfunctioning system is created by inserting bit-flip faults within one of the 6 functional modules. Fault simulation is repeated 100 times for each fault, and the fault insertion instant is randomly selected during test execution.

Experimental results for the OpenRISC1200 are presented. All the experiments were performed on a pool of state-of-the-art servers running Linux. The Verilog simulator for all simulations is VCS (Y-2006.06-SP1-16). Four functional tests were applied to the OpenRISC system. A single run of the tests takes 0.5 min, 1 min, 4 min, and 4.5 min, respectively. According to the experiment design above, fault

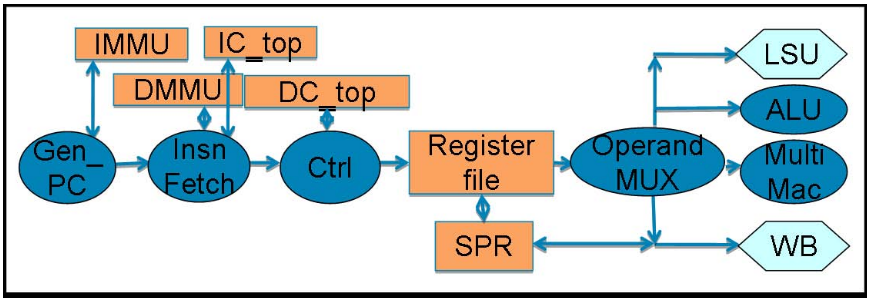


Figure 3.2: Block diagram of the CPU in OpenRISC 1200.

simulation is repeated 100 times for each fault. The total simulation time is nearly 600 hours. Bayesian inference is implemented using Matlab and it takes only a few seconds to obtain the inference results.

Due to the irreproducible nature of transient fault and practical limits on the number of simulations, faults at the pins of an actual faulty module may not always have the highest occurrence probability, which necessitates the inclusion of faults with smaller occurrence probability in an ambiguity group. In our experiments, if the *posterior* value of a fault is greater than 5% and this fault ranks in the list of faults with the top five *posterior* values, we place this fault in the ambiguity fault group. The threshold for selection depends on the number of fault candidates, but a characterization of the optimal threshold is left for future work.

Table 3.3 shows functional modules and the corresponding pin-level faults. Those pin-level faults are used for locating faulty modules inside the system. For example, F_{19}, F_{20}, F_{21} are faults inserted at ALU output pins for learning. According to the diagnosis results, if F_{19} has the highest occurrence probability, our conclusion is that the ALU module is the most likely faulty module causing system failure. Faults at the pins of real faulty module are called target faults. Diagnosis is considered to be correct, when at least one target fault falls in the ambiguity fault group.

Table 3.3: Functional modules and the corresponding pin-level faults.

Functional modules	Corresponding pin-level faults
ALU	F_{19}, F_{20}, F_{21}
Ctrl	F_9, F_{10}, F_{11}, F_{12}
IF	F_5, F_6, F_7, F_8
O_MUX	F_{13}, F_{14}, F_{15}, F_{16}, F_{17}, F_{18}
Multi	F_{22}, F_{23}
Gen_PC	F_1, F_2, F_3, F_4

Table 3.4: Diagnosis results using functional test *Basic*.

Faulty module	Ambiguity fault group	Rank	correctness
ALU	$\mathbf{F_{21}}$,F_6,$\mathbf{F_{20}}$,F_4,F_3	3	Yes
Ctrl	F_4,F_3	7	No
IF	F_{20},F_{17} ,F_{13},F_{18},$\mathbf{F_5}$	1	Yes
O_MUX	$F_{\mathbf{17}}$,$\mathbf{F_{18}}$,F_4,F_3	3	Yes
Multi	$\mathbf{F_{26}}$,F_{17},F_{14},F_{13},F_{11}	5	Yes
Gen_PC	$\mathbf{F_1}$,$\mathbf{F_4}$,$\mathbf{F_3}$	1	Yes

Table 3.4 shows diagnosis results for six different failing systems using the first functional test *Basic*. Each row is related to one failing case. The first column in Table V lists the actual faulty module in the failing system. The second column is the ambiguity fault group. Faults are placed in the ambiguity group if they meet the criteria defined above. Target faults are highlighted in bold fault. The third column shows the highest rank of target faults out of 23 fault candidates. The fourth column indicates the correctness of diagnosis. From Table 3.3, we can see 5 failing cases out of 6 can be correctly diagnosed based on only one functional test. If the faulty module is Ctrl in the failing system, we cannot correctly locate it.

Likewise, diagnosis results for six failing systems using the second functional test *Dhry* are shown in Table 3.5. In Table 3.5, the system with the faulty IF module cannot be correctly diagnosed, but the faulty Ctrl module can be distinguished by *Dhry*. We get blank results for faulty multiplier case, because the internal defects inserted in the multiplier do not cause errors on any of the observable registers for

Table 3.5: Diagnosis results using functional test *Dhry*.

Faulty module	Ambiguity fault group	Rank	correctness
ALU	F_{15},$\mathbf{F_{21}}$,F_8,F_2,F_3	3	Yes
Ctrl	$\mathbf{F_{11}}$,F_2,F_3	3	Yes
IF	F_{21} ,F_{15},F_{20},F_2	9	No
O_MUX	$\mathbf{F_{15}}$,F_{21},F_2,F_3,F_5	5	Yes
Multi	—	—	—
Gen_PC	$\mathbf{F_2}$,$\mathbf{F_3}$	1	Yes

the *Dhry* test. In other words, *Dhry* cannot sensitize the faults occurring inside the multiplier module.

In Table 3.6, two functional tests are used to improve diagnostic resolution. Here diagnostic resolution refers to the size of the ambiguity fault group. Four correctly diagnosed cases using the *Basic* test are selected for this experiment. First, the test *Basic* is applied to the system and ambiguity fault groups obtained under this test are shown in the second column in Table 3.6, which are the same as the results shown in Table 3.4. Subsequently, the *Dhry* test is applied to distinguish/rank faults in the ambiguity fault group obtained based on the previous test. In the second round inference, the candidates are only those faults in the previous ambiguity fault group. If the *posterior* value of a candidate is greater than 20% and it ranks in top-3 highest *posterior*, it is placed in the second-round ambiguity group, shown in the third column. The correctness is shown in the fourth column in Table 3.6. The average of ambiguity group size shrinks from 4.25 to 2.75 without loss of diagnosis correctness.

Fig. 3.3 shows the improvement of diagnosis accuracy for the Ctrl module when multiple functional tests are used. In Fig. 3.3, the y-axis is the rank of a pin-level fault of the Ctrl module, and the x-axis refers to the functional tests applied for diagnosis. To diagnose a system with faulty Ctrl module, we start by using the *Basic* test. The rank of fault F_{11} is 7. Next we apply the *Dhry* test based on the

Table 3.6: Improved diagnosis using the *Basic* and *Dhry* tests.

Faulty module	Ambiguity group in *Basic*	Ambiguity group in *Dhry*	Correctness
ALU	$\mathbf{F_{21}}$,F_6,$\mathbf{F_{20}}$,F_4,F_3	F_4,$\mathbf{F_{20}}$,F_3	Yes
IF	F_{20},F_{17} ,F_{13},F_{18},$\mathbf{F_5}$	F_{20},$\mathbf{F_5}$	Yes
O_MUX	F_{17},$\mathbf{F_{18}}$,F_4,F_3	$\mathbf{F_{18}}$,F_4,F_3	Yes
Gen_PC	$\mathbf{F_1}$,$\mathbf{F_4}$,$\mathbf{F_3}$	$\mathbf{F_1}$,$\mathbf{F_4}$,$\mathbf{F_3}$	Yes

Table 3.7: Multiple fault diagnosis using test *Basic*.

Faulty module	Ambiguity fault group	Inferred suspect module
ALU & IF	F_{18} $\mathbf{F_5}$,F_4,F_3	IF, O_MUX,Gen_PC
O_MUX & Gen_PC	$\mathbf{F_2}$,$\mathbf{F_3}$,$\mathbf{F_4}$	Gen_PC
IF & Gen_PC	$\mathbf{F_1}$,$\mathbf{F_2}$,$\mathbf{F_4}$,$\mathbf{F_3}$	Gen_PC
ALU & O_MUX	$\mathbf{F_{20}}$,$\mathbf{F_{17}}$,F_4,$\mathbf{F_{18}}$,F_3	ALU,O_MUX,Gen_PC

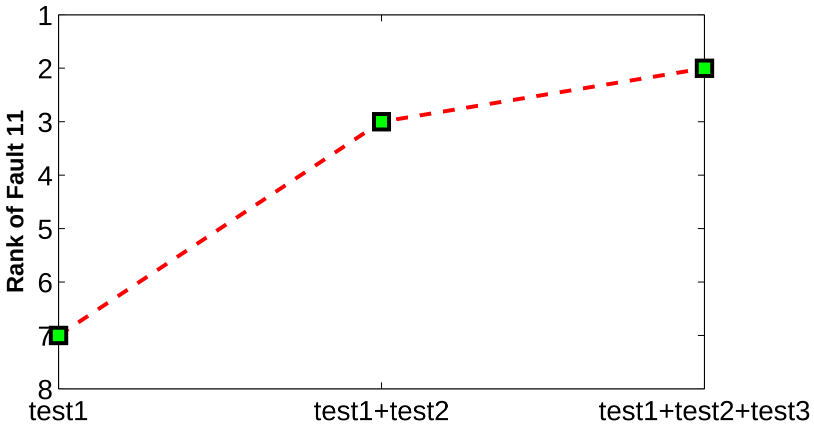


Figure 3.3: Improvement of diagnosis accuracy using more tests for the Ctrl module.

first test, and fault F_{11} is now rank third. When the third test *Multi* is also used, the rank becomes 2. This shows that when multiple tests are applied in diagnosis, diagnosis accuracy can be improved using Bayesian inference.

We next study the diagnosis of multiple faults using the proposed framework. Two faults inside different functional modules are inserted per trial to mimic the situation that multiple faults exist in a malfunction system. The database of conditional probability is the same as before. The diagnosis results are shown in Table 3.7. They are not as accurate as before, since the conditional probabilities used in the computation are obtained from single fault simulation. Multiple fault diagnosis is difficult and computationally expensive, as the volume of fault hypotheses grows exponentially with the number of faults in the system. A promising solution is to partition the overall system into subsystems, within which there is likely to be a single fault. This partitioning enables the application of single-fault diagnosis, which has only linear complexity, to the subsystems without the need to handle the exponential hypothesis explosion.

3.5 Summary

We have presented a board-level Bayesian inference-based fault diagnosis strategy. This approach offers two key advantages. First, it does not depend on the DFT features, which eliminates the requirement of building standard boundary-scan cells for all devices on a board. Second, by learning the behavior of board/system under different faulty scenarios, the inference engine can automatically perform diagnosis without manual effort. Experiments have been performed on an open-source RISC SoC to illustrate the use of Bayesian inference in fault diagnosis. Our results show that Bayesian inference-based diagnosis is a promising strategy to tackle the challenges involved in quality assurance of complex boards. By being adaptive, automated and self-learning, the approach is expected to lead to a new breakthrough in board/system diagnosis.

Chapter 4

An Effective Description of Functional Failure: Error Flow

Fault-insertion test has been optimized in previous chapters using high-level generalized defect models. However, inserting the most representative faults alone is not sufficient for accurate fault diagnosis. Another critical issue is to derive an accurate description of the failure. The information extracted from the failures significantly affects the diagnostic results. Traditionally, test pass/fail is the most common syndrome, which is too vague to locate the root cause. Some fine-grained syndromes, such as faulty registers, are in use today, but the diagnostic resolution achieved by them is still limited. In contrast, we propose an alternative fault syndrome, error flow, to facilitate the functional diagnosis. Error flow mimics actual data propagation in a circuit, thus it reflects the native (functional) mode of circuit operation. Taking the functionality of the circuit into account, the functional failure analysis and fault identification become easier.

In order to use error flows to identify the root cause, error flow is first learned from a good circuit by intentionally inserting faults. Then the root cause of a failing circuit is determined by comparing the similarity between the pre-learned error flows and the error flow observed from the failing circuit. The similarity of two error flows is evaluated based on the length of the longest common subsequence. Results for an open-source RISC SoC and an industrial communication circuit highlight the effectiveness of the proposed method.

4.1 Prior Work

In this chapter, we analyze error propagation in a failing circuit (chip or board) and its application in fault diagnosis. Error propagation reflects the intrinsic characteristics of a circuit in terms of its functionality. For example, in typical five-stage pipeline processor microarchitecture, errors caused by the program counter generator propagate in a different manner, compared to the errors caused by a defective ALU. Therefore, when we describe a failure, we not only record the location where an error is observed, but also record when the error is observed. The goal here is to locate the faulty component on a malfunctioning board, and hierarchically use this error flow based method to narrow down the defective areas in the failed component. This is extremely important for component suppliers to address the NTF problem and provide qualified components to system companies.

The concept of error propagation has been published in the literature [5], [57]. In [5], error propagation path was used to isolate failing functional blocks in modern microprocessors. However, the advantage of error propagation was not fully exploited. The key idea in [5] is to record the faults that can be detected at each observation point through fault simulation on a good processor. The observation points here refer to those locations where logic values can be observed, e.g., scannable registers. In order to diagnose a failing processor, once an error is observed at an observation point, all the faults that can be detected at this observation point are considered to be fault candidates. This approach still considers errors at each observation point separately, rather than matching a faulty block with a specific error propagation path.

In [57], a backtracing method for identifying the root cause of functional failures in the UltraSPARC family of processors was described. In this method, the faulty block can be located through four steps, namely logic cone extraction, cone input constraint generation, path sensitization, and extraction of the components to analyze for next cycle. The diagnosis results depend on an in-depth knowledge of the architecture and analysis of circuit topology.

The remainder of this chapter is organized as follows. Section 3.2 describes the definition of error flow and error-flow dictionary, and the advantages of an error-flow dictionary over a traditional fault dictionary. Section 3.3 describes the fault diagnosis framework using an error-flow dictionary. It consists of error-flow dictionary construction and root-cause diagnosis. Section 3.4 discusses the difficulties in the extraction of error flow in real chips/boards and provides two approaches in use today that can be adopted for error flow extraction. Finally, diagnosis results for a RISC SoC OR1200 and an industrial communication circuit are provided.

4.2 Dictionary-based Diagnosis

Before we describe the definition and advantages of error flow and error-flow dictionary, let's first go through the traditional dictionary-based diagnosis. A fault dictionary is commonly used for fault diagnosis [50], [58], [59]. In its simplest form, a dictionary includes fault entries and their corresponding syndromes. It is often created through fault simulation before diagnosis. During fault simulation, the erroneous behavior of the circuit under test is recorded, which is referred as a *fault syndrome*. In a traditional fault dictionary, a fault syndrome can be the pass/fail results of a given set of tests referred as a *coarse-grained fault syndrome*, or the match/mismatch results of all the outputs and internal observable points referred as a *fine-grained fault syndrome*. To locate the root cause of a failure, the actual behavior observed from the failing circuit is compared with each fault syndrome saved in the fault dictionary. If this look-up process succeeds in finding a match, the dictionary entry indicates the corresponding fault that leads to the failure on the circuit. A dictionary based on coarse-grained fault syndromes has smaller size, but it includes less failure information, comparing to the dictionary based on find-grained fault syndromes.

Table 4.1: An example of dictionary-based diagnosis.

(a) Fault Dictionary

	F_1	F_2	F_3	F_4	F_5	F_6
Ob_1	1	1	1	1	1	0
Ob_2	1	0	1	1	1	0
Ob_3	0	1	1	1	1	0
Ob_4	1	1	0	0	1	1

(b) Actual Failing Cases

	C_1	C_2	C_3
Ob_1	1	1	1
Ob_2	1	1	0
Ob_3	0	1	1
Ob_4	1	0	0

An example of diagnosis using a fine-grained fault dictionary is shown in Table 4.1. Assume that in a malfunctioning microprocessor, six modules (GenPC, IF, Ctrl, OMUX, ALU, Multi,) inside CPU are fault candidates. The status of the microprocessor can be observed from four observation points. The two observation points used are data buses (outputs of the ALU and IF). The other two are general purpose registers. The six fault candidates are denoted by F_1, F_2, F_3, F_4, F_5 and F_6. Four observation points are denoted by Ob_1, Ob_2, Ob_3 and Ob_4 in the fault dictionary; see part(a). A fault syndrome here is a binary vector, whose length is equal to the number of observation points on the circuit. An entry "1" means that the logic value at the observation point matches the golden value, and entry "0" implies that a mismatch occurs at the observation point. For example, the fault syndrome for F_1 is "1101"", which means that errors are observed at Ob_1, Ob_2 and Ob_4, when the first module is faulty. In Table 4.1 part(b), fault syndromes of three actual failing cases (C_1, C_2, C_3) are listed. For the failing case C_1, we can see that it has the same syndrome with F_1, therefore fault F_1 is inferred to be the root cause of failing case C_1. For the failing case C_2, it has the same fault syndrome with F_3 and F_4, and we find that the fault syndromes of F_3 and F_4 are the same. In this case, these two faults are undistinguishable, and often they are placed in a group, referred as an *ambiguity group*. For the failing case C_3, its fault syndrome is different from any of the syndromes in the dictionary. Therefore, we cannot diagnose the failing case C_3 based on the existing dictionary.

Although a traditional fault dictionary can be used for locating the root cause of failures in small circuits, they are not applicable to the diagnosis of functional failures in large systems due to the following reasons. First, in large systems, the number of potential faults is extremely large. It is impossible to create a fault dictionary using all possible faults and enumerating them. Second, a functional test typically includes hundreds of thousands to millions of cycles. It is impractical to run all the available functional tests to generate a traditional coarse-grained dictionary, and only the pass/fail information of a test is insufficient to accurately locate the root cause. Third, fault syndromes are often significantly overlapped, even in a fine-grained dictionary, so that it is difficult to achieve high fault-localization accuracy with a traditional fault dictionary. Therefore, an alternative error-flow dictionary is proposed to tackle these challenges. This dictionary reflects the functionality of a circuit and offers significant potential for accurate diagnosis.

4.3 Error Flow and Error-Flow Dictionary

An error flow is a sequence of integers, which records the order in which errors are observed at all the observation points. The exact time instant of error occurrence is not included in the error flow. An element in the sequence is the identifier of the observation point. Therefore, the error flow sequence consists of a series of distinct integers. The length of the sequence is not fixed, since a fault may not manifest itself at all the observation points. The maximum length of the error flow sequence is equal to the number of observation points. As for an error-flow dictionary, it consists of error flows and the root cause that leads to the corresponding error flow.

An illustration of an error flow is shown in Fig. 4.1. Assume that six modules and six observation points in a CPU are taken into account. Fig. 4.1 shows the propagation of two errors. One is caused by faulty module 2 (Instruction Fetch), which starts from observation point 1, and then goes to observation point 2 and observation point 4. The other is caused by fault module 5 (ALU). The error flows are shown at the bottom of the figure.

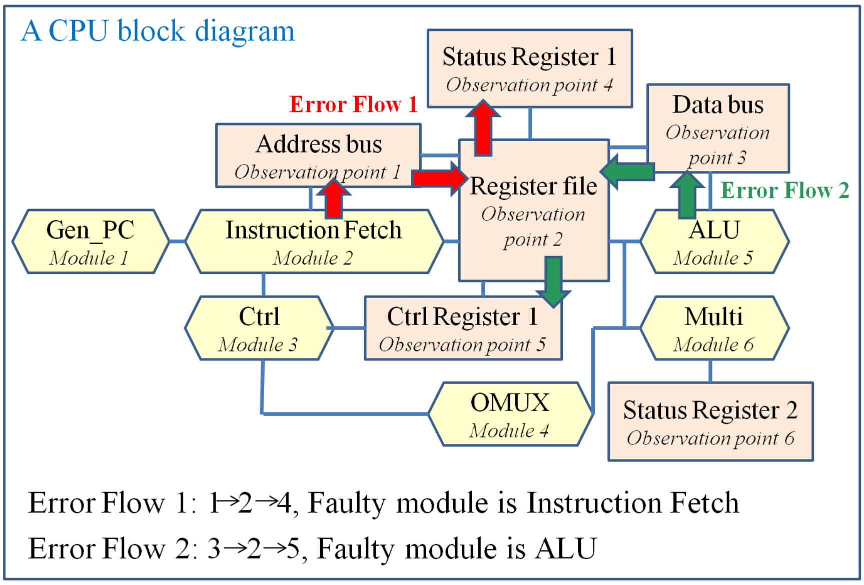


Figure 4.1: Illustration of error flows in a CPU block diagram.

It can be readily seen that error flow records the error propagation caused by a fault. It can be computed by recording the error occurrence location and the first clock cycle when the error is observed, referred as the *first failing cycle*. For a given fault, we can obtain the error observation locations and the first failing cycles for each location by comparing the logic values at observation points with golden values. The error flow is derived by ordering the observation points in terms of the

Table 4.2: An example of an error-flow dictionary.

Fault Candidate	Error Flows
Ctrl	1→ 3→ 2→ 4
ALU	3→ 4 → 1 3→ 2 → 4 → 1
Gen_PC	3→ 5 → 1 3→ 5 → 2

first failing cycle. The fault here can be a faulty ASIC on a board, a faulty module in an ASIC, or a faulty sub-module in a high-level module. Therefore, this is a generic method that is applicable at both chip and board levels. The diagnostic granularity depends on the faults used in the error-flow dictionary.

An example of an error-flow dictionary is shown in Table 4.2. Three fault candidates are shown in the first column, and the corresponding error flows are shown in the second column. Fault candidates, Ctrl, ALU, and Gen_PC, are functional units in a CPU. From the table, we can see that one faulty functional unit may have more than one error flow (e.g., ALU and Gen_PC). The reason is that many possible faults within a functional unit can lead to an observable error. Faults occurring at different locations or clock cycles may cause different error flows.

Note that error flow not only records where errors are observed, but it also records the order of error occurrence on the observation points. The order of error occurrence reflects the dataflow in a circuit, which is related to the functionality of the circuit. Therefore, diagnostic accuracy is expected to be higher with this function-related information. The advantages of using an error-flow dictionary compared to a traditional fine-grained fault dictionary can be seen in Table 4.3. In the traditional dictionary, the errors caused by fault F_1 are observed at observation points Ob_1 and Ob_2, which is the same as for fault F_2. Based on the traditional fault dictionary, faults F_1 and F_2 are undistinguishable, since they have the same fault syndrome. However, if we know that the errors can be observed in different orders at these two observation points, we can distinguish fault F_1 from F_2 using an error-flow dictionary shown in Table 4.3(b). Comparison of diagnosis results using two dictionaries for real life circuits is presented in Section V.

Table 4.3: Comparison of a traditional dictionary with an error-flow dictionary.

(a) Traditional Dictionary

Fault candidate	Ob_1	Ob_2	Ob_3
F_1	1	1	0
F_2	1	1	0

(b) Error Flow Dictionary

Fault candidate	Error Flow
F_1	1→2
F_2	2→1

4.4 Application of Error Flow in Fault Diagnosis

Diagnosis is performed by comparing the error flow observed on a malfunctioning board to the pre-learned error flows. In order to diagnose the root cause of functional failures, we first compute an error-flow dictionary using fault simulation. Next the dictionary entries are compared to the error flow observed on the failing circuit. The most similar error flow is selected from the dictionary, and its corresponding fault is considered to be the root cause of the failure. The similarity of two error flows is evaluated based on the longest common subsequence, a widely used metric in string-matching algorithm. The diagnosis process is divided into two steps: error-flow dictionary construction and root-cause diagnosis.

4.4.1 Error-Flow Dictionary Construction

The error-flow dictionary is computed in a hardware description language (HDL) simulation environment. The computation can be started when the RTL model of a design is available such that the dictionary will be ready before tapeout and manufacturing. Therefore, the diagnosis process can be immediately started when a failure occurs on a chip/board.

Based on the integration level under diagnosis, the fault candidates in the dictionary can be ASICs on a board, functional units in an ASIC, or sub-modules in a functional unit. Observation points typically are critical registers (e.g., CSRs). Let us assume for the sake for discussion that the diagnosis goal is to locate the faulty functional unit in a failing ASIC. In order to construct the error-flow dictionary, we first run logic simulation on a good ASIC and save golden logic values for all the observable registers. Second, we run fault simulation by inserting faults at the ports of a functional unit. The ports for fault insertion are randomly selected from bits of buses and control/status signals. Third, we compare logic values at these observation points in fault simulation with golden values in logic simulation, and save the first failing cycle for each observation point. Finally, we order the observation points by the first failing cycle, and then derive an error flow for this faulty functional unit. An error dictionary is constructed by repeating this process for all the functional units on the ASIC. The construction procedure is shown in Fig. 4.2. At the board level, diagnosis granularity is the components/ASICs on the board. Similarly, we insert faults at the ASIC pin level, observe critical registers within ASICs and observable memory components on the board, record the first failing cycle, and compute the error flow.

In this work, flip faults are inserted at the port/pin level to create a faulty unit. A flip fault is used to model the subtle effects of transient fault that are not easy detected by traditional DFT features [23]. At the chip-level, defects modeled by stuck-at faults are easy to screen using ATPG patterns. At the board level, the open/shorts at the interconnection among components can be screened using boundary scan test. Permanent failures caused by stuck-at faults do not pose the

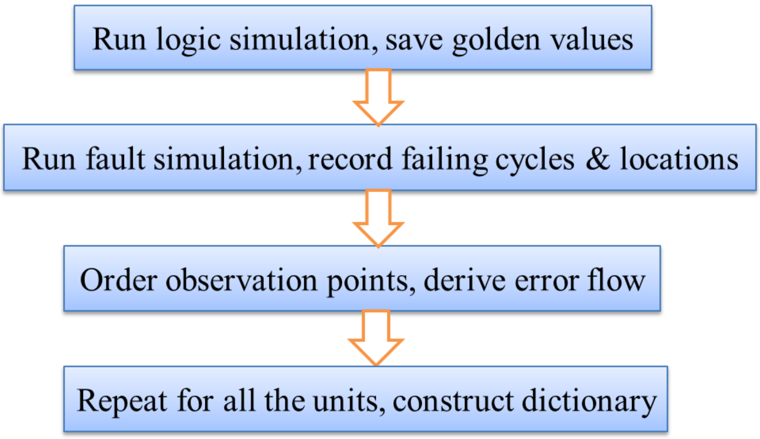


Figure 4.2: The procedure for error-flow dictionary construction.

main challenges in fault diagnosis [6], [9]. Therefore, we use flip faults to generate intermittent/transient failures occurring in modern high-speed circuits.

For a flip fault, we assume that the logic value of the faulty wire or state element is switched to the opposite of the fault-free value. The faulty value is not permanently associated with the faulty net or state element. The value on the net will be updated in the next clock cycle based on the logic. A fault insertion environment has been designed using Verilog Procedural Interface (VPI), which provides an application program interface to Verilog HDL. Faults can be inserted by invoking a VPI routine in Verilog code [56]. One flip fault at one output is inserted at a time in fault simulation. To make a module faulty, faults are inserted at the outputs of the module instead of internal nodes, since the number of internal nodes is often extremely large. Faults at module outputs are used to effectively represent internal defects. Details on the selection of the most representative outputs for fault insertion are described in [56].

The error-flow dictionary records erroneous behaviors of a circuit in the presence of different faults. This prior knowledge is learned through fault insertion experiments on a good circuit, and it is subsequently used in the fault diagnosis of a malfunctioning circuit. In practice, if a board under diagnosis is very complex, it is infeasible to run board-level simulation for collecting the information for the construction of the dictionary. An alternative method is based on the repair history of malfunctioning boards from the manufacturer's side. In the computation of a traditional fault dictionary, for each fault entry, only the location of error occurrence needs to be recorded. Compared with the traditional dictionary, the first failing cycle of error occurrence for each fault entry also needs to be stored in the construction of an error-flow dictionary. The storage required is therefore doubled, but this is not a limitation in a simulation environment.

4.4.2 Root-Cause Diagnosis

In the root-cause diagnosis step, we want to select an error flow from the dictionary that is the most similar to the error flow observed in the failing circuit, and then the fault corresponding to this error flow is identified as the root cause of the failure. The problem of comparing the similarity of two error flows can be viewed as a sequence-matching problem. The sequence-matching problem has been extensively studied in the literature [60], [61], [62]. We choose a widely used metric, namely the longest common subsequence (LCS), to evaluate the similarity of two sequences.

There are various algorithms to quickly compute the LCS [61], [62]. In our problem, the error flow is a series of distinct integers, and the length of the flow is equal to the limited number of observation points in a circuit. Therefore, the computation of LCS for error flow sequences is relatively easy. We can simply calculate the common subsequence starting with every observation point, and then choose the longest common subsequence as the LCS of the flow. For example, suppose two error flows to be compared are $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ and $2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5$. There are no common subsequences starting with 1, 4 and 5. The common subsequence starting with 2 is $2 \rightarrow 3 \rightarrow 4$; the common subsequence starting with 3 is $3 \rightarrow 4$. Therefore, the LCS of these two error flow is $2 \rightarrow 3 \rightarrow 4$, and the length of the LCS is 3. The computation effort is proportional to the square of the number of observation points. Two error flows being compared do not need to have the same length.

If more than one flow in the dictionary has the same degree of similarity with the given error flow, i.e., they have the same length of LCS with the given flow, we compare the start position of the LCS. The one that has an earlier LCS start position is considered to be more similar to the given flow. If two candidate flows have the same length of the LCS and the same start position, we conclude that the corresponding faults of these flows are equally likely to be the root cause. These faults are undistinguishable using our method and placed in an ambiguity group. The diagnosis accuracy is evaluated by the ambiguity group size and the correct diagnosis outcome.

In an error flow, the first few observation points are more important than the later observation points. Error observed at the later observation points may be caused by the earlier errors, rather than the root cause of the failure. Therefore, the later observation points are usually not very useful and tend to increase the number of suspect faults [5]. In the error flow comparison, we first select the flows that have the same first observation point with the error flow in the failing circuit, and then we calculate the similarity of two flows based on the above method. The complete diagnosis flow is shown in the Fig. 4.3.

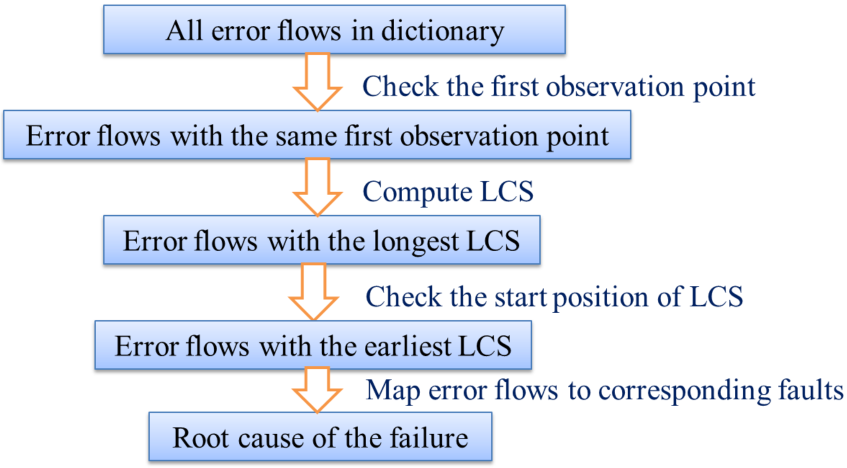


Figure 4.3: Diagnosis flow using an error-flow dictionary.

4.5 Error Flow Extraction in Real Chips/Boards

In a simulation environment, it is easy to save logic values at all the observation points in a large window, even the complete time frame of test execution can be considered. However, in practice, it is impossible to save values at all the observation points for a long period due to limited memory space. Therefore, while the error-flow dictionary can be easily constructed in a simulation environment, the error flow from an actual failing chip/board may be difficult to obtain. Also, it may not be complete or accurate. Two approaches in use today can be adopted to observe error flow in a manufactured chip/board, by recording failing cycles and failing observation points. One is an automatic test equipment (ATE)-based approach and the other is a system platform-based approach [5].

The ATE-based approach is often used for collecting failure information at the chip level. In order to obtain the first failing cycle, a snapshot of the internal observable cells (scan cells) is taken hundreds of cycles before the functional failure, and then the captured logic values are compared with golden values. If the captured values match the golden values, the first failing cycle most likely occurred between

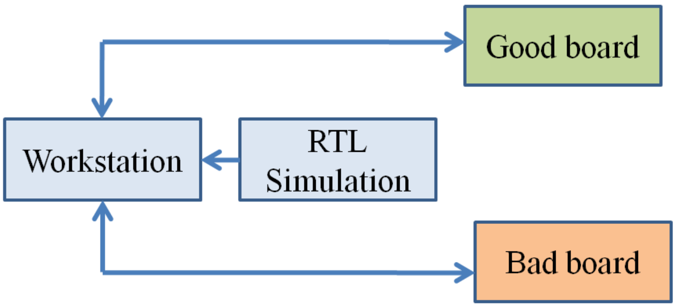


Figure 4.4: System platform architecture for extraction of error flows [5].

the current snap-shot cycle and the observed functional failure cycle; otherwise, the first failing cycle most likely occurred before the current snapshot cycle. Additional snap-shots of the scan cells from the failing circuit are taken and compared with the corresponding golden values. The search is continued in a binary fashion to obtain the first failing cycle [5]. Once the first failing cycle is determined, a few more snap-shots are needed to obtain the order of error occurrence at each observation point.

The system platform-based approach is applicable to both chip and board level diagnosis. An architecture of a system platform is illustrated in Fig. 4.4 [5]. In such a platform, the good board can be used as a reference. To collect the error flow information on the bad board, the same functional tests can be loaded into memory cells of both boards. A workstation can be used for controlling operations and communicating with both boards. Logic values at observation points obtained on the bad board are then compared with the RTL simulation data. An error flow can be derived by ordering the observation points in terms of the first failing cycle.

In addition, we can also use a trace buffer, which is typically integrated in a processor-based system to capture a snapshot of an executing system. Although the trace buffer is a limited resource, it provides a piece of historical record of application-code execution, timing, and data accesses. With this program history, it is easy to trace back through the program to examine what happened before the point of failure. This information is very useful for collecting failure information and extracting error flows.

4.6 Simulation Results

In this section, simulation results for a RISC SoC OR1200 and a communication controller from Cisco Systems Inc. are presented. All experiments are performed in the VCS simulation environment on a pool of state-of-the-art servers running Linux. Various failing cases are created for diagnosis, and results are verified using different functional tests. Diagnosis results highlight the effectiveness of error-flow dictionary-based method in terms of fault localization accuracy and correctness of diagnosis.

4.6.1 Results for OR1200

The CPU is the critical unit in the OR1200 system. So fault candidates in the error-flow dictionary are five functional units inside the CPU, namely ALU, Ctrl, IF, O_MUX and Gen_PC. A total of 17 observation points are selected from six memory units (IMMU, DMMU, DC top, IC top, GPRs and SPRs). Three functional tests, Basic, Dhry and CBasic, derived from design verification programs are used for diagnosis. We create different failing cases caused by these five units and examine the diagnosis results obtained by proposed error flow dictionary.

1) Diagnosis results using one functional test

Fault simulation is performed at the RT level. Flip faults are inserted at the outputs of the modules. Considering the function of the outputs, fault insertion locations include all the control and status signals and part of bits of data and address buses. The clock cycles for fault insertion are randomly selected. Selection of the diagnosis-efficient fault insertion points is another interesting problem [56], [20]. Here we focus on the diagnosis flow and the usage of an error-flow dictionary. One flip fault is inserted in one simulation run. For each faulty functional unit, fault simulation is performed 10 times, so that 10 error flows are derived for each faulty functional unit. There are 50 entries in the error-flow dictionary. A segment of error flow dictionary is shown in Table 4.4. The first three entries of each faulty unit are listed. For the IF unit, all the 10 entries are the same. Although error flows vary across fault insertion locations and insertion cycles, error flows of the same faulty unit often start with the same observation point and share a long common subsequence. The first observation point in each error flow is denoted in bold in Table 4.4, which is very important for distinguishing faults.

Artificial failing cases are created by inserting flip faults to the internal nets of the modules. Fault insertion locations include all the internal nets. Fault insertion cycles are randomly selected in the execution of test. Most of the inserted internal faults are masked (nearly 90%) [23]. A total of 48 artificial failing cases that produce observable errors are used for diagnosis. In Table 4.5, error flows of five artificial failing cases are listed as an example. The root cause of the failure is shown in the first column, and the corresponding error flow is shown in the second column. Diagnosis results of the 48 cases are shown in Table 4.6.

In Table 4.6, the diagnosis results are reported in terms of the number of failing cases under diagnosis, the number of cases that have been correctly diagnosed, correct rate and average ambiguity group size. For example, there are 18 failing cases with faulty Gen_PC under diagnosis and 17 cases are correctly diagnosed . The correct diagnosis rate is 94% and the average size of ambiguity group is 1.29. Specifically, one case is misdiagnosed in the 18 cases. In this case, Ctrl

Table 4.4: Error-flow dictionary for the OR1200 circuit.

Faulty Unit	Error Flow
Gen_PC	**8** → 9 → 2 → 3 → 7 → 16 → 17 → 6 → 1 → 5 → 12 → 13 → 14 → 10 → 11 → 4; **13** → 12 → 5 → 8 → 9 → 2 → 3 → 6 → 7 → 16 → 17 → 1; **8** → 9 → 10 → 16 → 2 → 3 → 6 → 7 → 17 → 11 → 14 → 12 → 13 → 1 → 5; ...
IF	**2** → 7 → 12 → 13 → 14 → 3 → 6; ...
Ctrl	**8** → 9 → 2 → 3 → 6 → 16 → 17 → 7 → 1 → 5 → 12 → 13 → 14 → 10 → 11 → 4; **8** → 9 → 2 → 3 → 6 → 7 → 16 → 17 → 1 → 5 → 12 → 13 → 14 → 10 → 11 → 4; **8** → 9 → 2 → 3 → 6 → 7 → 16 → 17 → 1 → 5 → 12 → 13 → 14 → 10 → 11; ...
O_MUX	**1** → 2 → 3 → 7 → 12; **6** → 1 → 2 → 11 → 13 → 7 → 3 → 12; **1** → 2 → 3 → 7; ...
ALU	**1** → 2 → 3 → 7 → 12; **1** → 3 → 7 → 12 → 2; **1** → 3 → 7 → 2 → 6 → 5 → 8 → 16 → 17 → 10 → 11 → 14 → 12 → 13 → 4; ...

is incorrectly identified as the root cause. In all the correctly diagnosed cases, for 12 cases, Gen_PC is identified as the exclusive faulty unit; for 4 cases, both Gen_PC and Ctrl are identified as the root cause; for 1 case, Gen_PC and ALU are identified as the root cause. Therefore, the average size of ambiguity group for the 17 correctly diagnosed cases is 1.29. Diagnosis results for other failing cases with different faulty units are listed in the same manner.

Diagnosis results using another functional test Dhry are listed in Table 4.7. For the 65 cases under diagnosis, 57 cases are correctly diagnosed. The correct diagnosis rate is up to 88%, and the average size of ambiguity group is less than 2. This means that the root cause can be accurately located using our method. The diagnosis correctness and fault-localization accuracy using proposed error-flow dictionary is compared to that obtained using traditional fault dictionary.

The comparison of diagnosis results for five failing cases using two different methods is shown in Table 4.8. Functional test Basic is used for diagnosis. Both the error-flow dictionary and the traditional dictionary are constructed based on the 50 simulation runs. Therefore, no additional efforts are included for the construction of error-flow dictionary with regard to simulation. For the Gen_PC case, the two methods provide the same results. However, for the IF and O_MUX cases, the root cause is exclusively and correctly identified by the proposed method. For the Ctrl case, it is correctly diagnosed by the proposed method, while it is misdiagnosed by the inference based method. Therefore, the error-flow dictionary based diagnosis method can provide higher diagnosis accuracy and fault-localization accuracy.

2) Results for the diagnosis of multiple faults

For multiple fault diagnosis, we create a failing system with two faulty modules Gen_PC and Ctrl. Faults within the two modules are inserted randomly. A total of 22 failing cases are created for diagnosis. Results are shown in Table 4.9. For 9 cases, diagnosis results indicate that Gen_PC and Ctrl as the root cause of the

Table 4.5: Error flows observed from five failing cases.

Faulty Unit	Error Flow
Gen_PC	**8** → 9 → 10 → 16 → 2 → 6 → 7 → 3 → 17 → 1 → 11 → 14 → 12 → 13 → 5 → 4
IF	**2** → 3 → 6
Ctrl	**8** → 9 → 16 → 17 → 2 → 3 → 6 → 7 → 1 → 5 → 10 → 11 → 14 → 12 → 13 → 4
O_MUX	**6** → 1 → 7 → 3 → 12 → 2
ALU	**1** → 2 → 3 → 7 → 12 → 6

Table 4.6: Diagnosis results using functional test Basic.

Faulty Unit	No. of cases under diagnosis	No. of correct diagnosis	Correct rate	Average size of ambiguity group
Gen_PC	18	17	94%	1.29
IF	5	5	100%	1
Ctrl	5	4	80%	2
O_MUX	10	7	70%	1.14
ALU	10	10	100%	1

Table 4.7: Diagnosis results using functional test Dhry.

Faulty Unit	No. of cases under diagnosis	No. of correct diagnosis	Correct rate	Average size of ambiguity group
Gen_PC	21	20	95%	1.3
IF	2	2	100%	1
Ctrl	8	6	75%	1.83
O_MUX	31	26	83%	1.15
ALU	3	3	100%	1

failure. The conclusions are correct and consistent with the actual failing cases. For 11 cases, diagnosis results are partially correct, including either Gen_PC or Ctrl. For 2 cases, IF is misdiagnosed as being the root cause.

3) Error flow under different functional tests

Three functional tests, namely Basic, Dhry, Cbasic, are used in the next set of experiments. These three tests are applied separately in the presence of the same

Table 4.8: Comparison of diagnosis results using error-flow dictionary and traditional dictionary.

Root cause	Error-flow dictionary results	Traditional dictionary results
Gen_PC	Gen_PC	Gen_PC
IF	IF	IF, O_MUX, ALU
Ctrl	Ctrl	Gen_PC
O_MUX	O_MUX	Gen_PC, O_MUX
ALU	ALU, O_MUX	Ctrl, ALU, IF

Table 4.9: Multiple faults diagnosis results.

Root cause	Diagnosis Results	No. of cases
Gen_PC & Ctrl	Gen_PC & Ctrl	9
	Gen_PC	5
	Ctrl	4
	Gen_PC & Ctrl & IF	1
	Gen_PC & ALU	1
	IF	2

Table 4.10: Error flow observed in three different tests.

Faulty Unit	Test	Error Flow
ALU	Cbasic	**1** → 3 → 7 → 2 → 6 → 5 → 12
	Dhry	**1** → 3 → 7 → 2 → 5
	Basic	**1** → 2 → 7 → 12 → 6
IF	Cbasic	**2** → 3 → 6 → 7 → 1
	Dhry	**2** → 3 → 6 → 7
	Basic	**2** → 3 → 6

faulty functional unit. Faults are inserted at the same port of a functional unit for the three test sequences. From the simulation results, we find that errors caused by the same faulty unit in different functional tests propagate along a similar path. Error flows observed in three different tests are shown in Table 4.10. This observation confirms that the error flow reflects intrinsic characteristics of a circuit. Test sequences appear to have less effect on the error flow.

4) Diagnosis results using an incomplete error flow

It is sometimes necessary to extract only an incomplete error flow from the actual failing circuit due to limited memory. Diagnosis results for a failing case using incomplete error flows are shown in Table 4.11. The root cause of the failing case is Gen_PC. Diagnosis results based on the complete error flow are first listed and the root cause is accurately located, and then the diagnosis results using three incomplete error flows are listed. When we compare an incomplete error flow to the error flows in the dictionary, if there is no error flow in a dictionary that has the same first observation point with the given error flow, we calculate the LCS of each error flow in the dictionary. The most similar error flow is the one with the longest length of LCS and the earliest start position. From the results, we can see that fault-localization accuracy is lowered using incomplete error flows. In order to improve the fault-localization accuracy, more snapshots of the observation points need be taken to obtain an accurate error flow.

Table 4.11: Diagnosis results using incomplete error flow.

	Error flow caused by faulty Gen_PC	Diagnosis results
Complete Error flow	8 → 9 → 10 → 16 → 2 → 6 → 7 → 3 → 17 → 1 → 11 → 14 → 12 → 13 → 5 → 4	Gen_PC
Incomplete Error flow 1	8 → 9 → 10 → 16 → 2 → 6 → 7 → 3 → 17 → 1 → 12 → 13 → 5 → 4	Gen_PC, Ctrl
Incomplete Error flow 2	9 → 10 → 16 → 2 → 7 → 3 → 17 → 1 → 12 → 13 → 5 → 4	Gen_PC, Ctrl
Incomplete Error flow 3	8 → 16 → 7 → 3 → 17 → 1 → 12 → 13 → 5 → 4	Gen_PC

4.6.2 Results for a Communication Controller

In the results for OR1200, we described the construction of the error-flow dictionary and generation of artificial failing cases for diagnosis. Diagnosis results for the cases with single fault, multiple faults, and incomplete error flows were reported in details. Diagnosis results using traditional fault dictionary were also presented for comparison. In this subsection, we take a communication controller as an example for locating faults within an ASIC, and then extend the error flow concept to the board level. The communication controller used in this experiment is an industrial port ASIC. It performs traffic forwarding, quality of service, route processing, etc.

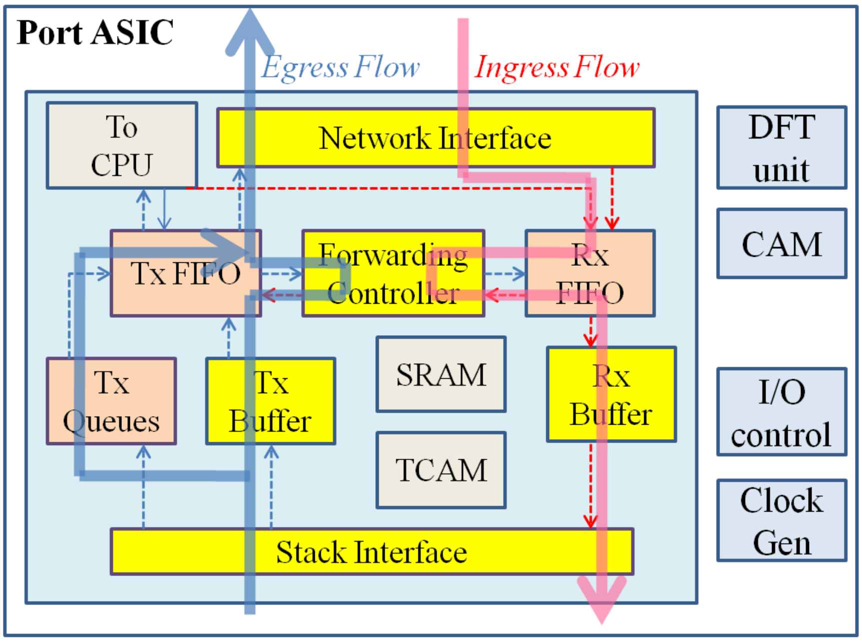
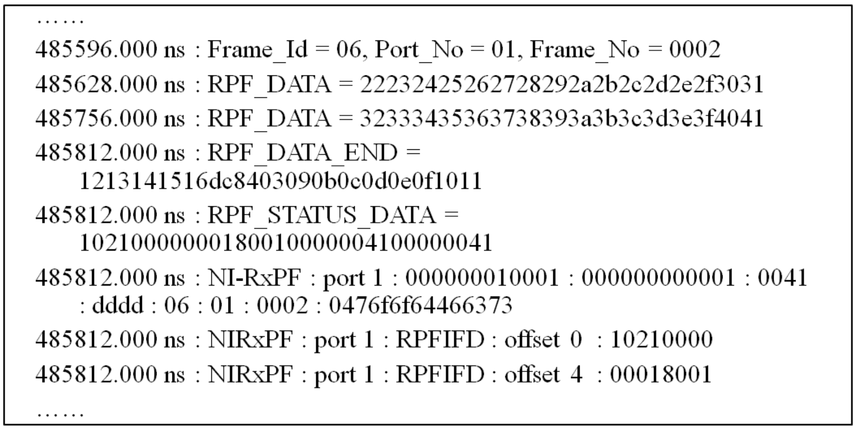


Figure 4.5: Block diagram of a port ASIC and data flows.

on an ethernet switch. The block diagram of the ASIC is shown in Fig. 4.5. The communication core mainly consists of transceiver data path, receiver data path, network interface, stack interface, and forwarding controller. The peripherals consist of high speed I/O controller, clock generator, design for test (DFT) features and content-addressable memory (CAM). Data flows (egress flow and ingress flow) are shown in different colors in Fig. 4.5.

Traffic test is run on this ASIC. For example, a packet is sent from network interface to the stack interface and expects to receive it within a certain time period. If the packet does not reach the interface within the expected time period, or the received packet is corrupted, an error is registered. A total of 11 observation points are selected to monitor the traffic, including control and status registers and data buses (RPF_DATA, RPF_STATUS_DATA, NIRxPF, etc.). A segment of the log file that records the values of these observation points is shown in Fig. 4.6.



```
……
485596.000 ns : Frame_Id = 06, Port_No = 01, Frame_No = 0002
485628.000 ns : RPF_DATA = 2223242526272829 2a2b2c2d2e2f3031
485756.000 ns : RPF_DATA = 32333435363738393a3b3c3d3e3f4041
485812.000 ns : RPF_DATA_END =
    1213141516dc8403090b0c0d0e0f1011
485812.000 ns : RPF_STATUS_DATA =
    10210000000180010000004100000041
485812.000 ns : NI-RxPF : port 1 : 000000010001 : 000000000001 : 0041
    : dddd : 06 : 01 : 0002 : 0476f6f64466373
485812.000 ns : NIRxPF : port 1 : RPFIFD : offset 0  : 10210000
485812.000 ns : NIRxPF : port 1 : RPFIFD : offset 4  : 00018001
……
```

Figure 4.6: A segment of the log file that tracks the observation points.

Error flows are derived by comparing the log file of a faulty circuit to that of a fault-free circuit as mentioned before. Five modules are taken as fault candidates. They are RxBuffer, TxBuffer, StackIF, NetworkIF, and ForwardingCtrl. Likewise, faults are inserted at the outputs of modules, and one fault is inserted in one simulation run. Ten simulation runs are performed for each faulty module. A total of 50 error flows are recorded in the dictionary. In order to create artificial failing cases for diagnosis, faults are randomly inserted to the internal nets/registers of the five modules. The number of cases under diagnosis is 34.

Diagnosis results for all the failing cases are shown in Table 4.12. In Table 4.12, the diagnosis results are described in the same manner as the OR1200 design. Four failing cases with faulty RxBuffer out of five are correctly diagnosed. All failing cases with faulty TxBuffer and faulty NetworkIF are correctly diagnosed. A total of 28 cases are correctly diagnosed out of the 34 cases. The overall correct diagnosis rate is 82.4%. For most of the correctly diagnosed cases, the root cause can be exclusively and accurately located. The average size of ambiguity group is 1.25. In contrast, using traditional fault dictionary, the overall correct diagnosis rate for the 34 cases is 70.3%. A total of 24 cases are correctly diagnosed. The average size of an ambiguity group is 2.8.

Table 4.12: Diagnosis results for the communication controller.

Faulty Unit	No. of cases under diagnosis	No. of correct diagnosis	Correct rate	Avg. size of ambiguity group
RxBuffer	5	4	80%	1.25
TxBuffer	6	6	100%	1
StackIF	8	6	75%	1.33
NetworkIF	5	5	100%	1.2
FwdCtrl	10	7	70%	1.43

In order to compare the diagnosis results of error-flow dictionary and traditional fault dictionary, five cases are randomly selected from the 34 cases, whose error flows are shown in Table 4.13. The comparison results are listed in Table 4.14. For this communication ASIC, the advantage of error-flow dictionary-based diagnosis method is more obvious. From the error flows in Table 4.13, we can that almost all the observation points appear in an error flow, which is true for the remainder 29 failing cases as well. Therefore, if we use a traditionary fault dictionary, the fault syndromes for different faulty modules are nearly identical. This can lead to a large ambiguity group and low localization accuracy. This conclusion is supported by the diagnosis results shown in Table 4.14. For four cases (RxBuffer, StackIF, NetworkIF, ForwardingCtrl), the traditional method indicates that three modules are equally likely to be the root cause. Since there are only five fault candidates under diagnosis, the fault localization accuracy is very low. In contrast, the root causes of three cases (RxBuffer, TxBuffer, NetworkIF) are correctly and exclusively located using the proposed error-flow dictionary based method. For the other two cases, two suspect modules are in the final diagnosis results.

Considering the application of error flow based diagnosis on a printed circuit board, let us take a high-speed ethernet switch as an example. Fig. 4.7 shows a

Table 4.13: Error flow of actual failing circuit.

Faulty Unit	Error Flow
RxBuffer	**10** → 8 → 5 → 1 → 2 → 3 → 7 → 11 → 9 → 6
TxBuffer	**5** → 3 → 7 → 2 → 1 → 10 → 8
StackIF	**10** → 8 → 11 → 9 → 3 → 7 → 5 → 1 → 2 → 6
NetworkIF	**6** → 2 → 3 → 7 → 1 → 10 → 8 → 5 → 11 → 9
ForwardingCtrl	**2** → 1 → 11 → 9 → 3 → 7 → 5 → 6 → 10 → 8

Table 4.14: Comparison of diagnosis results using error-flow dictionary and traditional fault dictionary.

Root cause	Error-flow dictionary results	Traditional dictionary results
RxBuffer	RxBuffer	NetworkIF, StackIF, TxBuffer
TxBuffer	TxBuffer	TxBuffer
StackIF	StackIF, RxBuffer	NetworkIF, StackIF, TxBuffer
NetworkIF	NetworkIF	NetworkIF, StackIF, TxBuffer
FwdCtrl	FwdCtrl, TxBuffer	NetworkIF, StackIF, TxBuffer

block diagram of an ethernet switch using two port ASICs [63]. On this board, CPU controls the switch-to-switch communication and synchronization, and updates the routing caches attached to each port ASIC. Switch fabric provides inter-connect data path and control path. It performs local switching within a switch. Functions of the port ASIC include traffic forwarding, route processing, etc. Using StachWisePlus technology, a stacking architecture optimized for Gigabit ethernet, multiple boards can be stacked to a unified system [63].

In Fig. 4.7, two data flows are illustrated. The left one is a local data flow. The packet is forwarded from the source port PHY to the port ASIC, the switch Fabric, the port ASIC, and finally to the destination PHY. The right data flow shows the propagation of a packet to a remote destination (e.g., a port on another board). To locate the faulty components on this kind of communication systems,

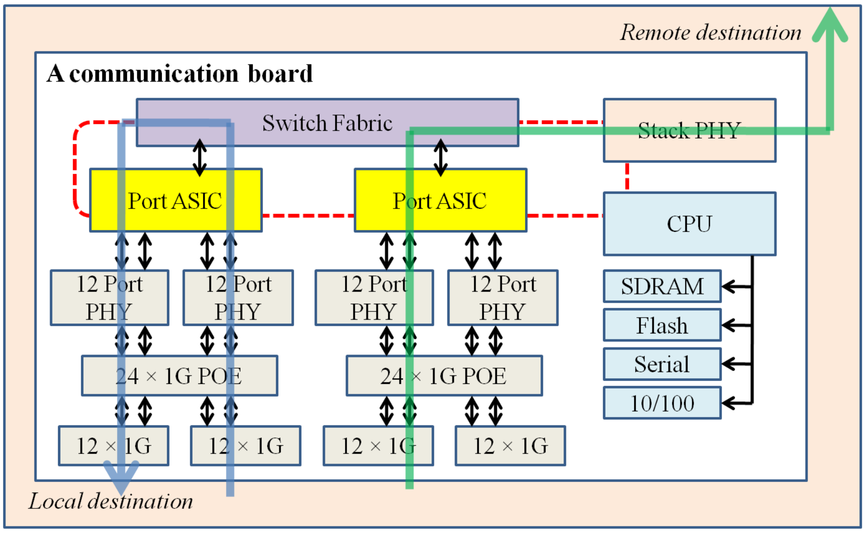


Figure 4.7: Block diagram of an ethernet switch and the traffic on it.

error flow based diagnosis provides an effective solution. The diagnosis results for the port ASIC have been described above. At the board level, traffic is often monitored by error/drop counters, interrupt status registers, CRC registers, etc. These are viewed as observation points on a board. Tracking the order of error occurrence at these observation points significantly facilitates the localization of the root cause on a malfunctioning board. For example, if errors are observed at a port of Fabric and error counters in the port ASIC, and the error flow is from the port ASIC to Fabric, the port ASIC is more likely to be defective. Furthermore, a local traffic test can be run on the port ASIC to narrow down the faulty module inside. Analysis of error flows on the switch-like systems is ongoing. Improvements and issues will be described in the future.

4.7 Summary

We have presented an error-flow dictionary-based fault diagnosis method, which is an automated and generic approach to diagnose functional failures at chip/board level. Compared to traditional dictionary-based diagnosis methods, our proposed method takes the order of error occurrence into account for fault diagnosis. The error flow here represents the data flow of a circuit, and it reflects the intrinsic characteristics of the circuit in terms of functionality. Therefore, it provides an effective solution to the diagnosis of functional failures. Moveover, the root cause of a failure can be narrowed down by hierarchically using the error-flow based diagnosis method, which is a significant benefit for board- or system-level diagnosis.

Another advantage of this approach is that the error flow dictionary can be computed while the product is being developed. This can potentially decrease the diagnosis time when a failure occurs on a prototype or in the field. Moreover, the root-cause diagnosis is based on the similarity of two error flows, which is evaluated using the length of the longest common subsequence. Simulation results on an open-source RISC SoC and an industrial communication circuit demonstrate high fault localization accuracy and diagnostic correctness of the proposed method.

A potential limitation of our method is that the diagnostic accuracy may be affected due to the incompleteness of the error flow observed from the actual failing circuit. This can be addressed by increasing the number of snapshots of observation points on the failing circuit to obtain a relatively complete error flow. By reflecting the functionality of the circuit using error flow, this proposed method is expected to address the challenges involved in the diagnosis of NTF functional failures in complex chips/boards.

Chapter 5

Diagnosis-Oriented Fault-Insertion Point Selection

In Chapter 2, we constructed a pin-level fault model to generalize the behavior of internal defects. The pin-level faults we selected are those that can cover the most internal defects. Thus inserting the most representative faults is able to accelerate FIT in system reliability assessment. Besides the reliability assessment, another important application of FIT is fault diagnosis. Fault-insertion test has been used to create artificial faulty scenarios in industry. These artificial faulty scenarios are used as references. Diagnosis process is eased by comparing the real failure on the manufacturing line or in the field with the artificial ones. However, it is time-consuming and impractical to insert faults everywhere and record the fault syndromes. It is necessary to insert the faults that are the most diagnosis-efficient.

In this chapter, we present an optimization method to select the diagnosis-oriented fault-insertion points. Faults inserted at these points are able to produce similar syndromes to those caused by the internal defects, in the meantime syndromes from different components or modules are different from each other. In this way, the inserted faults can mimic the erroneous behavior of real defects, and the different syndromes produced by these faults pave the way for fault diagnosis. The optimization model is based on integer linear programming. Error flows proposed in the previous chapter are used as fault syndromes. Results on diagnostic accuracy for the OR1200 system highlight the effectiveness of the proposed method compared to a baseline random fault-insertion scheme.

5.1 Problem Statement

Diagnosis of functional failures is challenging and time-consuming due to the following reasons [6]. First, functional test typically includes hundreds of thousands to millions of cycles. Thus the execution time of a functional test is overly long, and a large volume of data dumped during simulation need to be analyzed for fault diagnosis. Second, replication of functional failures is challenging. Intermittent/transient failures are difficult to reproduce as the effect of defects becomes subtle in high-speed circuits. Third, controllability and observability for functional test are very limited. Many devices are only accessible via control and status registers. Moreover, engineering expertise is often required in functional diagnosis and it takes a long time to develop, especially in the initial product-ramp phase.

Due to these challenges, it is difficult to diagnose the functional failures only based on the fault syndromes on a failing system. An alternative method, fault-insertion test described in Chapter 1, has attracted considerable attention in the study of fault diagnosis [26], [20], [13]. Artificial faults are used to mimic the effects of manufacturing defects and in-field intermittent/transient errors occurring on a failing system [56]. For the diagnosis purpose, a set of fault syndromes with known failure reasons is produced by FIT. To locate the root cause of the failure on a failing system, we can simply compare the syndrome on the failing system to the syndromes produced by FIT. The intentionally inserted fault that produces the most similar syndrome to that observed on the failing system is considered to be the root cause of the failure. The difficulties involved in analyzing and diagnosing the functional failures are bypassed by this cause-effect method.

In practice, FIT is a required test step at system companies [13], [24], [25]. In this step, system hardware integrated with system software can be tested as a whole in the area of hardware-error handling, and the artificial fault insertion provides an indication of the system response to real hardware errors. FIT is a promising technique in fault diagnosis, but it is difficult to decide the fault-insertion points because of the prohibitively large number of potential fault sites in a system. In addition, the artificial syndromes produced by FIT may be different from the syndrome on the real failing system. Therefore, this work aims at selecting the most diagnosis-efficient locations for fault insertion. Through inserting faults to these locations, the most similar syndrome can be generated, and thus the fault-localization accuracy can be improved.

The remainder of this chapter is organized as follows. Section 5.2 provides the definition of the diagnosis-oriented fault-insertion point and introduces similarity score as a metric for selection of the diagnosis-oriented FI points. Section 5.3 presents the selection flow and the optimization model using integer linear programming. Section 5.4 describes the simulation results for the OR1200 circuit.

5.2 Diagnosis-oriented Fault-Insertion Points

Our goal is to select the fault-insertion points that meet the following criteria:

- Faults inserted at those points produce very similar syndromes to that caused by the internal defects.
- Syndromes of one faulty component/module are different to that of other faulty components/modules.

The first criterion guarantees that the artificial faults inserted at the outputs of a module can accurately mimic the erroneous behaviors caused by the real defects occurring inside that module. The second criterion increases the differences of fault syndromes of the current module and the syndromes of other modules. This type of fault-insertion points is referred to as diagnosis-oriented fault-insertion point.

The selection of diagnosis-oriented fault-insertion points is independent of the diagnosis method. Diagnosis accuracy is improved by inserting faults at the most appropriate outputs, which are selected by the proposed method. In this work, we take error flows as fault syndromes. The error-flow based diagnosis presented in Chapter 4 is used to verify the effectiveness of the proposed method.

5.3 FI Point Selection and Optimization Model

In this section, the method of the diagnosis-oriented fault insertion point selection is described. There are five steps in the selection flow; see Fig. 5.1. In general, there are two sets of fault syndromes. The first set is created by pin-level faults, and the second set is created by internal defects. We analyze the similarity of the syndromes between the two sets, and select the pin-level faults that can best represent the internal defects and be distinguished from the pin-level faults from other components.

5.3.1 Selection Flow

1) Collect distinct error flows caused by pin-level faults.

Bit-flip faults are inserted to each output of each module. One fault is inserted per trial. Therefore, the simulation times is equal to the summation of the number of outputs of all the modules. Fault insertion time instances are randomly selected in the test execution. Observation points are key registers in the system (e.g., control and status registers). Error flow is derived by ordering the error occurrence at all the observation points. If no errors are observed at any observation points

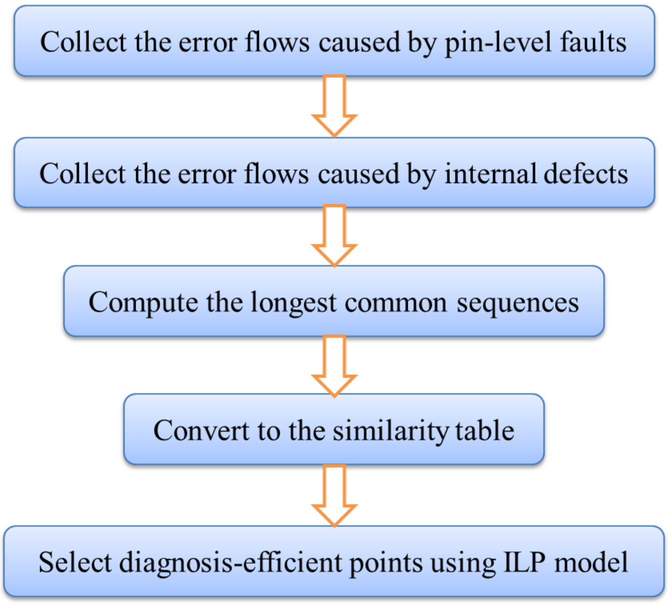


Figure 5.1: Flow of the selection of diagnosis-efficient FI points.

in the presence of a fault, or the error flow caused by a fault is the same to an existing error flow from the same module, the output where the fault is inserted is removed from the list of candidates for the diagnosis-efficient outputs. Finally, the distinct fault syndromes and corresponding faults for each module are obtained.

2) Collect error flows caused by internal defects.

Bit-flip faults are inserted to a number of randomly selected internal nodes and flip-flops of each module. In a simulation environment, the internal bit-flip faults are used as the internal defects caused in manufacturing. Likewise, error flows are derived by ordering the error occurrence at all the observation points. Fault syndromes and corresponding internal defects are stored.

3) Calculate the longest common sequence between error flows.

In the same manner as in the previous chapter, the length of the longest common sequence (LCS) is used as a metric to evaluate the similarity between two error flows. The length of the LCS is saved in a table, where the columns stand for the pin-level faults, and the rows stand for the internal defects. An entry in the table is the length of LCS between the error flow caused by the artificial fault in the corresponding column and the error flow caused by the defect in the corresponding row.

4) Convert the LCS length table to a similarity table. In a similarity table, the columns stand for the pin-level faults, and the rows stand for the internal defects, which are the same with that in the LCS length table. To convert the LCS length table into the similarity table, we rank all the entries in the same row. The largest entry gets the highest rank, and thus gets the highest similarity score, vice versa. For example, the rank of the shortest length is 1. The similarity score is an exponential functional of the rank with base 2; see (5.1). Ranking is performed separately for each row in the table.

$$\texttt{Similarity Score} = 2^{\texttt{Rank}} \tag{5.1}$$

The reason of assigning the similarity score in such a manner is because of the following attribute:

$$2^n > \sum_{i=0}^{n-1} 2^i \tag{5.2}$$

The inequality shown in (5.2) implies that the highest similarity score is greater than the summation of all the remainder similarity scores in a row. We assume that the same similarity score is counted only once. The advantage of using such a scoring mechanism is that it is very easy to compare the highest score in two sets. Assume that there are two sets (A and B), where elements are powers of 2, and we want to determine if the highest score in A is greater than B. We can simply compare the summation of all the scores in A with the summation of all the scores in B. From (5.2), we know that if the summation of all the scores in A is no less than the summation of all the scores in B, the highest score in A must be greater than or equal to the highest score in B. This attribute greatly facilitates the construction of the optimization model.

5) Maximize the number of correct diagnosis cases obtained using the ILP model.

Diagnosis is correct if a fault inserted at one of the outputs of the real faulty module is indicated as the root cause of the failure, i.e., a fault inserted at one of the outputs of the real faulty module gets the highest similarity score. In order to maximize the number of correct diagnosis, an ILP model is developed. By solving the ILP model, the most diagnosis-efficient outputs (fault insertion points) are determined. An output is considered to be a diagnosis-efficient output if the syndrome of a fault inserted at this output is the more similar to that caused by its internal defect, compared to the syndromes caused by faults inserted at the outputs of the other modules.

5.3.2 ILP Model for Optimization

The problem of maximizing the number of correct diagnosis by selecting the most diagnosis-efficient outputs can be addressed using the following ILP model.
Objective function:

$$\texttt{Maximize} \sum_{i=1}^{D} g_i \tag{5.3}$$

Subject to:

$$\sum_{j \epsilon T_{i_1}} (c_{i,j} \times s_j) - \sum_{j \epsilon T_{i_2}} (c_{i,j} \times s_j) > (g_i - 1) \times M \tag{5.4}$$

Where $g_i, s_j \in \{0, 1\}$.

In this model, g_i, s_j are variables; $c_{i,j}$ is the corresponding entry in the similarity table; and M is a large constant, which is much greater than the highest similarity score in the table. Our objective is to maximize the summation of g_i, where g_i is an indicator to show if a case is correctly diagnosed. If the failing case i is correctly diagnosed, g_i=1; otherwise g_i=0. The total number of failing cases is D. The variable s_j is an indicator to indicate if the fault j is selected. If fault j is selected for fault insertion, s_j=1, otherwise, s_j=0.

In Constraint (5.4), T_{i_1} is a set of artificial faults that are inserted at the outputs of the actual faulty module, which leads to failing case i. T_{i_2} is a set of all the artificial faults except those in set T_{i_1}. The first term on the left-hand side of (5.4) is the summation of the similarity score of all the artificial faults inserted at the outputs of the real faulty module. The second term is the summation of the similarity score of all the artificial faults from the other modules. According to the attribute of our scoring mechanism denoted in (5.2), if the highest similarity score from set T_{i_1} is greater than that in set T_{i_2}, the left-hand side is greater than 0, and thus 1 can be assigned to g_i. The failing case i is correctly diagnosed. If not, the failing case i cannot be accurately diagnosed. Indicator $g_i = 0$ and the right-hand side of (5.4) is a large negative number. The constraint is trivially satisfied. By maximizing the summation of g_i, the most diagnosis-efficient outputs are determined.

In practice, the number of faults under selection can be large and thus the rank of a fault can be a large number as well. Subsequently, 2^{rank} is an extremely large number. However, the selection goal is to ensure that the fault with the highest score is one of the faults inserted at the outputs of the real faulty module. Therefore, it is only necessary to consider the faults that generate the most similar syndromes compared to the syndromes of defects, i.e., we only need to focus on the large entries. We can simply assign a 0 score for the smaller entries that are ranked far behind.

Another practical problem is that more than one fault may have the same similarity score, since some faults may have the same length of LCS. In order to ensure the correctness of the ILP model, the same similarity score must only be added once in the summation in Constraint (5.4). We add an intermediate variable to address this problem. Assume for failing case i there are N faults that have the same similarity score R, and the set of these faults is named as A. We introduce an intermediate variable $b_{i,R}$. It is a 0/1 variable. If any of the faults in set A is selected, $b_{i,R}$=1; otherwise, $b_{i,R}$=0. Thus by substituting $R \times b_{i,R}$ for $\sum_{j \epsilon A}(c_{i,j} \times s_j)$ in the summation, we can guarantee that the same similarity score is only added once. The value of $b_{i,R}$ is constrained by (5.5).

$$\sum_{j \epsilon A} s_j \leq N \times b_{i,R} \leq N \times \sum_{j \epsilon A} s_j \tag{5.5}$$

A simple example is presented in this subsection to illustrate the proposed selection method. Assume that three modules are fault candidates in a circuit. Faults F_1, F_2, F_3 are inserted at the outputs of module$_1$. Faults F_4, F_5 are inserted at the outputs of module$_2$. Faults F_6, F_7 are inserted at the outputs of module$_3$. Defects d_1 and d_2 are inside the module$_1$. Defects d_3 and d_4 are inside module$_2$. Defect d_5 is inside module$_3$. The LCS length table between the syndromes of the five defects and seven artificial faults is shown in Table 5.1.

In order to simplify the ILP model and include fewer variables in the model, non-zero similarity scores are only assigned to the top three largest entries in a row in the LCS length table, i.e., the scores of the entries that are ranked 1st, 2nd, 3rd are respectively 8, 4, 2, and the scores of the remainder entries are 0. The

Table 5.1: A LCS length table.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
d_1	2	3	6	4	7	0	2
d_2	8	5	3	1	4	4	2
d_3	4	2	5	7	6	3	4
d_4	0	4	7	5	7	3	2
d_5	3	4	6	2	4	5	8

similarity score table computed based on Table 5.1 is shown in Table 5.2.

Table 5.2: An similarity table.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
d_1	0	4	0	2	8	0	0
d_2	8	4	0	0	2	2	0
d_3	0	0	2	8	4	0	0
d_4	0	2	8	4	8	0	0
d_5	0	0	4	0	0	2	8

The ILP model for this example is shown below. There are five cases under diagnosis. The objective function is to maximize the number of cases that are correctly diagnosed. Constraints (5.7), (5.8), (5.10), (5.11), (5.12) determine if the failing case caused by defect d_1, d_2, d_3, d_4, d_5 can be correctly diagnosed, respectively. Constraint (5.9) determines the value of the intermediate variable $b_{2,2}$. This constraint is added since two faults get the same similarity score of 2 for the second failing case.

Objective function:

$$\texttt{Maximize:} \sum_{k=1}^{5} g_i \tag{5.6}$$

Subject to:

$$4s_2 - (2s_4 + 8s_5) > 100(g_1 - 1) \tag{5.7}$$

$$(8s_1 + 4s_2) - (2b_{2,2}) > 100(g_2 - 1) \tag{5.8}$$

$$s_5 + s_6 \leq 2b_{2,2} \leq 2(s_5 + s_6) \tag{5.9}$$

$$(8s_4 + 4s_5) - 2s_3 > 100(g_3 - 1) \tag{5.10}$$

$$(4s_4 + 8s_5) - (2s_2 + 8s_3) > 100(g_4 - 1) \tag{5.11}$$

$$(2s_6 + 8s_7) - 4s_3 > 100(g_5 - 1) \tag{5.12}$$

This ILP model can be solved by a linear programming solver. The public domain linear programming solver lp_solve is used for this purpose [42]. The maximum value of the objective function is 5. From the selection results, s_2, s_4, s_7 are equal to 1; g_1 through g_5 are equal to 1. All the other variables are 0. Therefore, we know that if faults F_2, F_4, F_7 are inserted for diagnosis, all the five failing cases can be correctly diagnosed.

5.4 Simulation Results

In order to verify the effectiveness of the proposed selection scheme, experiments are performed on OR1200 as before. Fault candidates are five functional units inside the CPU, namely ALU, Ctrl, IF, O_MUX and Gen_PC. A total of 17 observation points are selected from six memory units (IMMU, DMMU, DC top, IC top, GPRs and SPRs). Functional test, Basic, derived from design verification programs is used in the experiments. Failing cases are created by inserting faults to the internal nodes of the five functional units. In this section, selection results are first presented. Next, the diagnosis results using the diagnosis-efficient outputs are presented and compared to the results using randomly selected outputs.

The diagnosis granularity in this experiment is a functional unit. The root cause of a failing system is one of the five functional units. Diagnosis is implemented by inserting faults to the outputs of the five modules and comparing the syndromes of artificial faults to the syndromes of the real failing case. The goal is to select the outputs of the five units that can produce the most similar syndrome to that shown on the real failing system, and thus provide high diagnosis accuracy. For the five functional units (ALU, Ctrl, IF, OMUX, GenPC), the numbers of outputs are 36, 239, 69, 96, and 42, respectively.

First, we insert a flip fault to each output of each module. One fault is inserted in one trial. The time instance of fault insertion is randomly selected during test execution. If a fault inserted to an output is masked, i.e., no errors are observed in the simulation, the output is removed from the list of candidates. In addition, if the faults at two outputs produce the same syndrome, only one of the outputs is left for selection. Simulation results show that the number of distinct fault syndromes from the faulty ALU unit is 13, which means that faults inserted at 13 outputs lead to 13 distinct syndromes, and the 13 outputs of the ALU unit are the candidates for the diagnosis-efficient outputs. Likewise, the number of distinct syndromes from the Ctrl, OMUX, IF, and GenPC unit is respectively 47, 23, 45, and 36.

In order to select the diagnosis-efficient outputs for fault insertion, we create a number of failing cases by inserting faults to the internal nodes of the functional units, which is used to mimic the failing scenarios caused by manufacturing defects. For each functional unit, 30 internal nodes are randomly selected for fault insertion. For the ALU unit, 2 distinct syndromes are produced. For the other four units (Ctrl, OMUX, IF, and GenPC), the number of distinct syndromes is respectively 4, 2, 4, and 4. A number of defects are masked. They cannot manifest themselves at any observation points. Fortunately, an exhaustive collection of the syndromes of defects is not necessary in the proposed selection scheme. Selection of diagnosis-efficient outputs is based on the syndromes of existing failing cases. Syndromes from a different set of failing cases are used to verify the diagnosis efficiency of the selected outputs.

Based on the existing syndromes, a 16×164 matrix is first computed as the LCS length table. The rows stand for the faults at the outputs. The columns stand for the defects (internal faults). The entries are the length of LCS between the syndromes of the faults in the corresponding rows and the syndromes of the defects in the corresponding columns. To convert the LCS length table into the similarity table, we rank all the entries in the same row. The smallest entry gets the highest similarity score. The ranking is performed separately for each row in the table. In order to simplify the ILP model and include fewer variables in the model, non-zero similarity scores are only assigned to the three smallest entries, i.e., the scores of the entries that are ranked 1st, 2nd, 3rd are respectively 8, 4, 2, and the scores of the remainder entries are 0. A segment of the similarity table for OR1200 is shown in Table 5.3.

In Table 5.3, d_1 through d_{16} are 16 manufacturing defects leading to system failures. Specifically, d_1 and d_2 are the defects inside the ALU unit; d_3 through d_6 are the defects inside the Ctrl unit; d_7 and d_8 are the defects inside the IF unit; d_9 through d_{12} are the defects inside the OMUX unit; and d_{13} through d_{16} are the defects inside the GenPC unit. In the columns, F_1 through F_{164} are 164 faults inserted at the outputs of the five units. Specifically, F_1 through F_{13} are faults inserted at 13 outputs of the ALU unit; F_{14} through F_{60} are faults inserted at 47 outputs of the Ctrl unit; F_{61} through F_{83} are faults inserted at 23 outputs of the IF unit; F_{84} through F_{128} are faults inserted at 45 outputs of the OMUX unit; and

Table 5.3: A segment of the similarity table for OR1200.

	F_1	...	F_{13}	F_{14}	...	F_{60}	F_{61}	...	F_{83}	F_{84}	...	F_{128}	F_{129}	...	F_{164}
d_1	0	...	2	0	...	0	0	...	0	0	...	0	0	...	0
d_2	0	...	8	0	...	0	0	...	0	0	...	0	0	...	0
d_3	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_4	0	...	0	8	...	0	0	...	2	0	...	0	0	...	0
d_5	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_6	0	...	4	0	...	0	0	...	0	0	...	0	0	...	0
d_7	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_8	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_9	0	...	4	0	...	0	4	...	0	0	...	0	0	...	0
d_{10}	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_{11}	0	...	2	0	...	0	0	...	0	0	...	0	0	...	0
d_{12}	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_{13}	0	...	0	0	...	0	0	...	0	0	...	0	0	...	0
d_{14}	2	...	2	2	...	2	2	...	8	2	...	2	0	...	2
d_{15}	0	...	0	8	...	0	0	...	4	4	...	0	0	...	0
d_{16}	0	...	0	8	...	0	0	...	0	2	...	0	0	...	0

F_{129} through F_{164} are faults inserted at 36 outputs of the GenPC unit.

According to the principle of the error-flow based diagnosis, the fault that produces the most similar syndrome to that observed on the failing system is considered to be the root cause. Assume that the diagnosis granularity is a functional unit. If the fault inserted at the output of one functional unit gets the highest similarity score, the corresponding functional unit is determined as the root cause. Taking the failing case caused by defect d_1 as an example, we can correctly diagnose this case, if one of the fault from F_1 to F_{13} gets the highest similarity score among all the fault candidates. Therefore, in order to achieve the highest correct diagnosis rate, the goal of the selection is that the faults at the output of the actual faulty unit get the highest score among all the selected faults for each failing case. In each row of Table 5.3, the similarity scores of the faults at the output of the actual faulty unit are denoted in grey. The goal of selection is to guarantee that the highest score in each row is from the grey area. The ILP model for this optimization problem is shown as follows.

Objective function:

$$\texttt{Maximize:} \sum_{k=1}^{16} g_i \tag{5.13}$$

Subject to:

$$2s_{11} + 4s_4 - (2b_{1,2} + 4s_{126} + 8s_{31}) > 100(g_1 - 1) \tag{5.14}$$

$$s_{32} + s_{87} + s_{88} + s_{89} + s_{92} \leq 5b_{1,2} \leq 5s_{32} + 5_{s87} + 5_{s88} + 5s_{89} + 5s_{92} \tag{5.15}$$

$$4s_4 + 8s_{11} - (2b_{2,2} + 4s_{126}) > 100(g_2 - 1) \tag{5.16}$$

$$s_{31} + s_{73} + s_{88} + s_{92} \leq 4b_{2,2} \leq 4s_{31} + 4s_{73} + 4s_{88} + 4s_{92} \tag{5.17}$$

...

$$0 - (2b_{161,2} + 4s_{66} - 8s_{14}) > 100(g_{161} - 1) \tag{5.18}$$

There are 110 constraints and 227 variables in this model. It is solved by lp_solve on a 32-bit windows machine [42]. The CPU time is 107.314 seconds. A total of 14 faults are selected from the 164 faults and 13 failing cases can be accurately diagnosed by the set of selected faults without any ambiguous fault candidates. Besides the 13 accurately diagnosed cases, the root causes of 2 cases are correctly diagnosed as well, but the actual faulty unit is not exclusively located. The correctness rate is 93.75%. The selection results are listed in Table 5.4. The highest score in each row is denoted in bold. We can see that for most of the rows, the highest score is located in the grey area, i.e., the failing cases represented in the rows are accurately diagnosed. Specifically, the root causes of failing cases caused

Table 5.4: Similarity table between the selected faults and the defects.

	F_4	F_{15}	F_{18}	F_{32}	F_{34}	F_{54}	F_{62}	F_{69}	F_{85}	F_{89}	F_{109}	F_{127}	F_{142}	F_{161}
d_1	**4**	0	0	2	0	0	0	0	0	0	2	0	0	0
d_2	**4**	0	0	0	0	0	0	0	0	0	0	0	0	0
d_3	0	0	0	0	0	**4**	0	0	0	0	0	0	0	0
d_4	0	**2**	2	0	0	0	0	0	0	0	0	0	0	0
d_5	0	0	**8**	0	0	0	0	0	0	0	0	0	0	0
d_6	2	0	0	0	**4**	0	0	0	0	2	0	0	0	0
d_7	0	0	0	0	0	4	**8**	0	0	0	0	0	0	0
d_8	2	0	0	0	0	0	0	**4**	0	0	0	0	0	0
d_9	2	0	0	0	0	0	0	0	**8**	0	0	0	0	0
d_{10}	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0
d_{11}	4	0	0	2	4	0	0	2	0	**8**	0	0	0	0
d_{12}	0	0	0	0	0	0	0	0	0	0	0	**2**	0	0
$\mathbf{d_{13}}$	0	0	**8**	0	0	0	0	0	0	0	0	0	0	0
d_{14}	**4**	**4**	0	2	2	2	0	0	0	2	**4**	0	2	**4**
d_{15}	0	2	2	0	0	2	0	0	2	0	0	0	2	4
d_{16}	0	0	0	0	0	0	0	0	0	0	**2**	0	**2**	0

by defect d_{14} and d_{16} are not exclusively located. The failing case caused by defect d_{13} denoted in boldface is misdiagnosed.

Compared to the optimized selection using an ILP model, 14 faults at the outputs are randomly selected from the 164 faults. Diagnostic results show that 9 out of 16 defects are correctly diagnosed. In order to make a fair comparison, we repeat the random selection 30 times. The distribution of the number of correctly diagnosed cases of the 30 trials is illustrated in Fig. 5.2. The X-axis stands for the number of defects that are correctly diagnosed. The Y-axis stands for the number of trials. For example, 14 defects are correctly diagnosed in one trial, and 11 defects are correctly diagnosed in two trials. The average of the number of correctly diagnosed defects is 8.3. In contrast, the number of correctly diagnosed defects is 15 using the proposed diagnosis-oriented selection method.

Finally, for cross-verification of the effectiveness of the selected outputs, a new set of failing cases are created for diagnosis. The total number of cases is 437. The

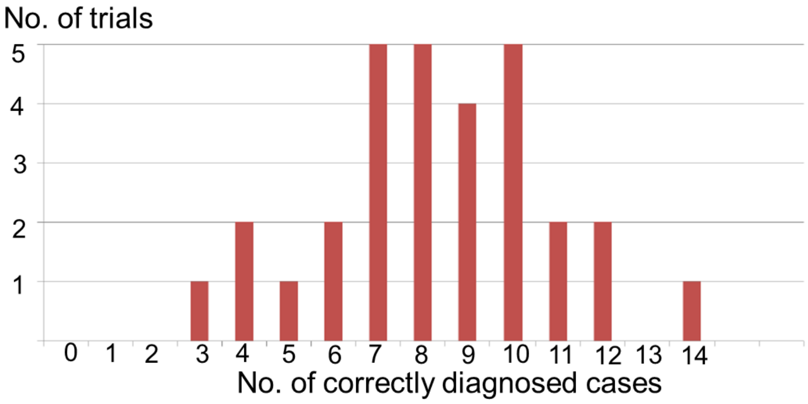


Figure 5.2: The distribution of the number of correctly diagnosed cases.

number of failing cases caused by faulty ALU, Ctrl, IF, OMUX, and GenPC unit are 30, 74, 21, 100, and 212, respectively. Likewise, these failing cases are created by randomly inserting faults to the internal nodes of the functional units. A total of 351 out of 437 cases are correctly diagnosed using the 14 selected diagnosis-efficient outputs. The correctness rate is 80.3%. The correctness rate drops because new syndromes are included in the new set of failing cases. In comparison, 14 faults are randomly selected like before, and the random selection is performed 30 times. The average of the number of correctly diagnosed cases using randomly selected outputs is 255. The correctness rate is 58%. The correctness rate obtained by using the diagnosis-efficient outputs is more than 20% higher than that obtained by randomly selected outputs. This highlights the effectiveness of the proposed diagnosis-oriented fault insertion point selection method.

5.5 Summary

We have presented a diagnosis-oriented fault-insertion point selection method, in order to maximize the the number of correctly diagnosed cases by inserting faults at the diagnosis-efficient outputs. The optimization problem has been described using an ILP model. The construction of the ILP model is based on pre-learning knowledge, which involves the fault syndromes with known root causes and the syndromes produced by FIT. The optimal set of fault insertion points is determined using the ILP model. The syndromes produced by the faults inserted at the selected outputs of a module are the most similar to the syndromes caused by defects inside the module, while they are different from the syndromes produced by faults inserted at the outputs of other faulty modules. This guarantees high diagnosis accuracy. Results on a case study using an open-source RISC system-on-chip highlight the effectiveness of the proposed method compared to a random selected scheme. The optimal solution of the fault insertion points provides guidelines for FIT based functional diagnosis.

Chapter 6

Machine Learning for Board-Level Fault Diagnosis

Diagnosis of functional failures at the board level is critical for improving product yield and reducing manufacturing cost. State-of-the-art board-level diagnostic software is unable to cope with high complexity and ever-increasing clock frequencies, and the identification of the root cause of failure on a board is a major problem today. Ambiguous or incorrect repair suggestions lead to long debug times and even wrong repair actions, which significantly increases the repair cost and adversely impacts yield.

In this chapter, an intelligent diagnosis system is developed, which is able to automatically learn debug knowledge from empirical data and identify the most likely faulty component given a new failed board. Thus it eliminates the difficulties involved in the traditional knowledge acquisition. Fine-grained fault syndromes extracted from failure logs and the corresponding repair actions are used to train the system. The usage of two machine learning algorithms (artificial neural networks and support vector machines) in fault diagnosis is explored. From the theoretical perspective, support vector machines provide globally optimal solutions, unlike artificial neural networks that usually converge to locally optimal solutions. However, an advantage of artificial neural networks is that the relationship between fault candidates and fault syndromes can be easily interpreted from these models. An industrial board, which is currently in production, is used to validate the diagnosis approach in terms of diagnostic accuracy, resolution, and quantifiable improvement over current diagnostic software.

6.1 Problem Statement and Prior Work

Field data and experience reports from repair technicians highlight many problems with diagnostic software currently in use, especially for functional tests that involve actual data in a real application. The diagnostic resolution offered by today's tools is limited to ASICs on the board. No repair guidance is provided for memory devices or passive components on the board. Diagnostic resolution is also poor in practice—multiple repair candidates are often listed and these candidates are not prioritized. Technicians are forced to run debug programs repeatedly and carry out physical probing in many places to identify the root cause of failures, a practice that significantly increases the debug and repair time. Based on past repair records, we have found the debug time for the functional test considered here to be as high as several weeks. The correctness of diagnosis, i.e., the probability of the actual failing component included in the list of suspects, is unacceptably low, and the root cause is seldom exclusively pinpointed.

In order to overcome the difficulties described above and provide accurate diagnostic results, we propose an intelligent diagnosis method based on machine learning algorithms. Machine learning, a branch of artificial intelligence, is focused on automatic learning from empirical data and making intelligent decisions. Artificial neural networks (ANNs), which are inspired by the structure/functional aspects of biological neural networks, are used to model complex relationships between inputs and outputs, or to capture the statistical relationship between observed variables. ANNs have been applied to integrated circuits testing and debug over the past two decades [64], [65], [66], [33], [64], [67].

Most of existing ANN-based diagnosis methods in the literature target either small combinational circuits or analog circuits. In [66], a three-layer feed-forward network is trained with back propagation and is designed to target multiple faults in small combinational circuits. The test data are derived from a fault truth table, which is constructed by inserting random single stuck-at faults. The effectiveness of the method is validated only with small circuits that consists fewer than ten logic gates. In [67], the diagnosis of telephone exchange line cards using ANNs at British Telecom is described. A three-layer feed-forward network, with 77 input neurons and 8 output neurons, was constructed to solve the diagnosis problem in structural test. The inputs are measurements from in-circuit test (ICT). The outputs are fault candidates, which are 8 categories of the faulty components (e.g., resistors, relays).

The problems targeted above are different from the functional diagnosis problem addressed in this thesis. In functional diagnosis, fault syndromes are extracted from a failure log, e.g., mismatched interface, error counter, etc. Therefore, fault syndromes are obtained from the results of functional test sequences, instead of measurements from ICT. Moreover, the diagnostic goal is to accurately locate the faulty component on the board rather than the category of the component. For example, hundreds of resistors can be soldered on a board. We obtain very limited diagnostic information, if we only determine that a resistor caused the failure, without pointing out the specific resistor.

In [33], an ANNs-based method was proposed as a possible solution for correcting the inaccurate diagnostic function derived from model-based diagnosis. The inputs represent the pass/fail results of tests. Assume that the number of tests is N. The number of input neurons is N, and the number of neurons on hidden layer is $N(N-1)/2$. The outputs represent the fault candidates. Due to the large number of hidden neurons, the entire syndrome space cannot be used to train the ANNs [33]. Moreover, diagnostic accuracy depends on the selection of syndromes. No results were provided in [33] regarding the training time, the number of tests in use, and the improvement in the diagnostic accuracy on actual test cases.

In contrast to existing applications of general-purpose ANNs in fault diagnosis, we propose a carefully crafted architecture of ANNs. It can be rapidly trained and thus it can handle large datesets with thousands of inputs and outputs. It is initialized with the occurrence probability of fault syndromes, which significantly improves the diagnostic accuracy. It can easily interpret the relationship between the fault syndromes and repair action. In prior work, the effectiveness of ANNs is always associated with the problem of how to interpret the data derived from the ANNs [68]. Moreover, with the fine-grained fault syndromes from one failed test, the proposed architecture can locate the most likely faulty component. This facilitates diagnosis for a particular test and accelerates the repair flow, especially for the tests for which faults are difficult to debug (e.g., traffic test for a board used in telecom applications, as explained later).

Besides ANNs, we explore the application of another machine learning algorithm, support vector machines (SVMs) [69]. The SVM algorithm is based on the statistical learning theory and the Vapnik-Chervonenkis (VC) dimension introduced in [69]. A Support Vector Machine (SVM) performs classification by constructing an optimal hyperplane that separates the data into two categories. Compared to ANNs, this new supervised learning method has a number of theoretical and computational merits. A significant advantage of SVMs is that the solution to an SVM is globally optimal and unique, while ANNs suffer from multiple local minima. We analyze the performance of SVMs with real manufacturing data. The best design of SVMs is determined by extensive simulation results.

The remainder of this chapter is organized as follows. Section 6.2 describes the proposed intelligent diagnosis method based on ANNs. The basic concepts of

ANNs is first presented, and then the proposed architecture of ANNs and the experimental results on an industrial board are presented in details. The training data and test cases used in the experiments are taken from the manufacturing line of a board currently in production. Diagnostic accuracy is compared to that achieved by traditional ANNs, Bayesian statistics, and commercial diagnostic software. The high accuracy achieved by the proposed ANNs highlights the effectiveness of the proposed method. Section 6.3 first describes the principle of support vector machines, and then describes how to apply SVMs to the board-level fault diagnosis problem. Simulation results of the SVMs-based diagnostic system are presented in this section as well. Section 6.4 compares the SVMs-based method with the ANNs-based method from theoretical and experimental perspectives. Section 6.5 summarizes this chapter.

6.2 Intelligent Diagnosis using Artificial Neural Networks (ANNs)

One of the advantages of using an ANN-based method is that it avoids the time overhead associated with knowledge acquisition and rule development required for expert systems [67]. Without the need to understand the complex functionality of the boards, ANNs automatically obtain relevant diagnostic knowledge during the training process, which reduces dependence on human knowledge. Generally speaking, a set of fault syndromes and corresponding repair actions are given, which are from the repair history in the manufacturing database. According to the given information, ANNs implicitly generate the connections between fault syndromes and the repair actions. After training is complete, the ANNs are able to derive the most likely correct repair action for a new set of syndromes.

6.2.1 Introduction to ANNs

Artificial neural networks (ANNs) are widely used for pattern classification and related problems [70]. ANNs consist of neurons and weighted connections between neurons. Neurons are arranged in layers, and weighted connections link the neurons in different layers. A value is associated with each connection, referred to as *weight*, corresponding to the synaptic strength of neuron connections. The behavior of an ANN depends on both the weights and the input-output function, referred to as *transfer function*. This function typically falls into one of three categories, namely, linear, step, and sigmoid. A simple ANN and the computation in an artificial neuron is shown in Fig. 6.1. Two basic network architectures are feed-forward and recurrent. In the feed-forward architecture, there is no feedback between layers as the network shown in Fig. 6.1. In the recurrent architecture, there is feedback between layers, thus these networks can remember prior inputs. Before use, ANNs must learn from training examples. In supervised learning, we are given a set of example pairs, and the aim is to find an appropriate function that matches the examples. Back propagation algorithm is widely used for training ANNs, which minimizes the the difference between actual outputs of ANNs and the desired values using gradient descent [70]. When the difference is less than the pre-defined threshold, referred to as *performance goal*, the training is complete. The design and implementation of ANNs are supported by Matlab neural network toolbox [71]. A multi-layer feed-forward neural network can therefore be easily implemented in Matlab.

In this work, ANNs are adopted to address the challenges involved in functional diagnosis at the board level. The intention is to develop an automated diagnosis tool that can learn from historical repair data, involve less human effort, and provide accurate diagnostic guidance. The automated learning characteristic

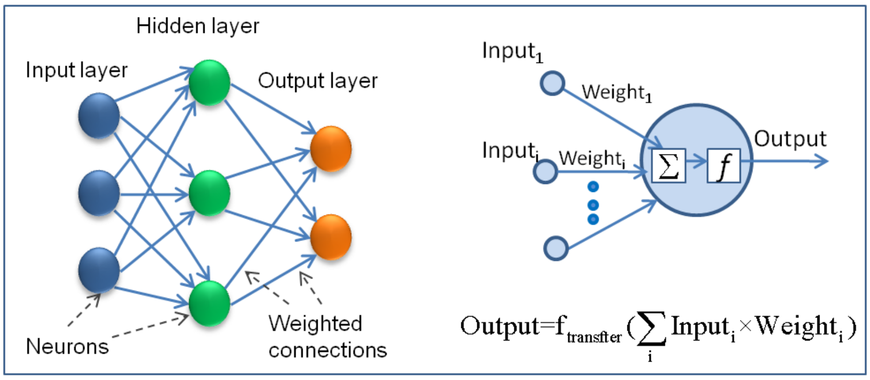


Figure 6.1: A two-layer neural network and the computation in a neuron.

of ANNs fits this problem well. The proposed ANNs intelligently construct the connections between fault syndromes and repair actions, without the need for a complete understanding of the complex functionality of a board.

6.2.2 Proposed Architecture of ANNs for Fault Diagnosis

Traditionally, a three-layer feed-forward network is used to deal with the fault diagnosis problem in small scale circuits, and the weights of the network are randomly initialized [66], [64], [65], [33], [67], [27]. In board-level functional diagnosis, thousands of syndromes (inputs of the ANNs) and hundreds of repair actions (outputs of the ANNs) must be extracted from the failure logs. If we use traditional ANNs to solve this practical problem, the training time is prohibitively long and the diagnostic accuracy is low, as highlighted in the next sub-section. Although the training of ANNs can be viewed as a one-time cost, a structure of an ANN that can be rapidly trained benefits the diagnosis flow. This is because ANNs need to be updated when new training cases are available. In addition, traditional ANNs typically take the pass/fail results of multiple tests as the inputs. These coarse-grained syndromes cannot effectively locate the faulty component for a failed functional test. In practice, it is often the case that one test sequence or a suite has to be cleared before we can run the next-level test suite. Our target is to diagnose the root cause of failure based on the failure log of one test suite (referred to here as a test, without loss of generality).

In contrast to existing applications of ANNs to fault diagnosis, we propose a group of single-layer networks to diagnose the root cause of board-level functional failures. The group size is equal to the number of known faulty components that have been replaced in the past (based on the available repair history). The initialization of weights is based on the occurrence probabilities of fault syndromes, instead of random initialization. For each single-layer network, the input neurons represent fault syndromes, and the single output neuron represents a component. Details on the extraction of fault syndromes and replaced components are presented in the next subsection. The input value is either 1 or 0. A "1" implies that the fault syndrome appears in the log file; otherwise, it is a "0". The desired (ideal) value at the output neuron is either "0" or "1". A "1" means that the component represented by this network is the root cause of failure, and it should be replaced to repair the malfunctioning board; a "0" means that the corresponding component is not the root cause. The actual value at the output is a fraction between 0 and 1, which can be viewed as the probability of the component being the root cause. A value closer to 1 implies that the component represented by the network is more likely to be the root cause, and vice versa. The proposed architecture is generic in the sense that it can be used for the functional diagnosis of various types of systems. For a different system, we only need to prepare a new set of training data for the ANNs, which can be easily achieved by updating the scripts used for syndrome extraction. An example of the architecture of ANNs with 500 syndromes and 100 actions is depicted in Fig. 6.2. The training of the neural networks and diagnosis flow are described below.

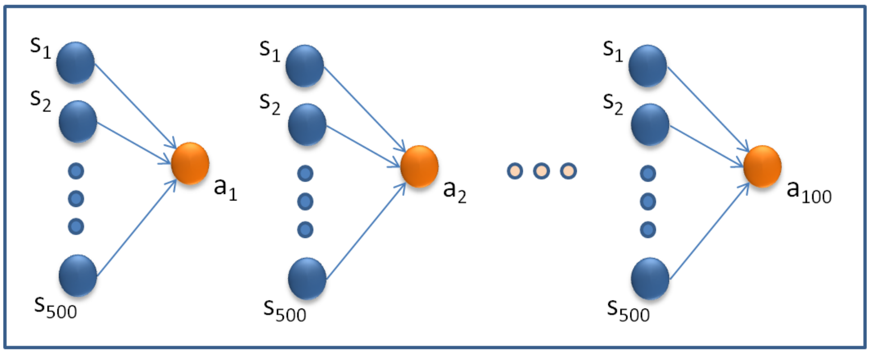


Figure 6.2: An illustration of the proposed ANN architecture.

The proposed diagnosis flow includes four steps that are described in Fig. 6.3. Generally speaking, ANNs first learn from the historical cases, and then diagnose a new case based on the accumulated knowledge.

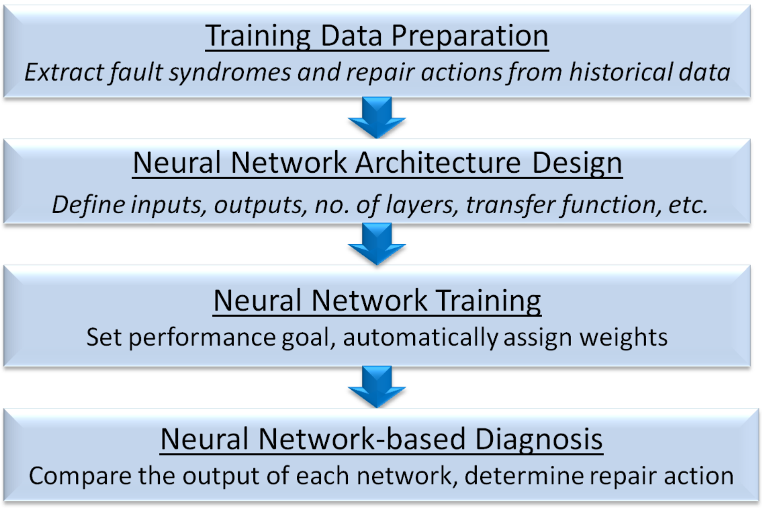
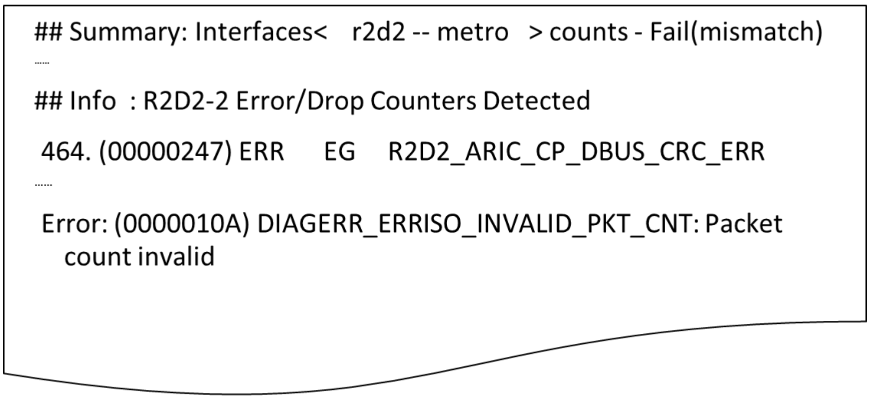


Figure 6.3: The diagnosis flow using neural networks.

In Step 1, we prepare the data (inputs and outputs) for the training of ANNs. This information is derived from boards that have been repaired successfully. The fault syndromes in the log files are extracted as the inputs, and the corresponding repair action is taken as the output. The extraction of fault syndromes is critical for the learning step. The fault syndromes should provide a complete description of the failure, and the extracted syndromes for different actions should have sufficient diversity such that we can eliminate ambiguity in the eventual repair recommendations. Fault syndromes vary across the products and tests. Taking network systems as an example, traffic tests are performed through route processors, interface modules, and port ASICs to ensure correct functionality and connectivity. Critical fault syndromes in a failed traffic test often consist of error counters, drop counters, interrupt bits, mismatch interfaces, ECC/CRC errors, etc. To extract these fault syndromes, we parse the log files by keywords and extract the syndromes using Perl scripts. For example, a segment of the log file for a failed traffic test is shown in Fig. 6.4. The fault syndromes extracted are r2d2-metro (mismatched interface), R2D2_ARIC_CP_DBUS_CRC_ERR (error counter) and 0000010A (error ID). Each of these elements is considered to be one syndrome. According to the syntax of different log files, changes may be needed in the parsing scripts. The repair action is often directly recorded in the database, e.g., “replacement is component U100”. The extracted syndromes and actions are used as inputs and outputs for the training of ANNs.



```
## Summary: Interfaces<   r2d2 -- metro   > counts - Fail(mismatch)
......
## Info  : R2D2-2 Error/Drop Counters Detected
 464. (00000247) ERR     EG     R2D2_ARIC_CP_DBUS_CRC_ERR
......
 Error: (0000010A) DIAGERR_ERRISO_INVALID_PKT_CNT: Packet
    count invalid
```

Figure 6.4: A segment of the log file for a failed functional test.

In Step 2, we define the numbers of inputs, outputs, and layers for the ANNs as part of architecture specification. These definitions are related to training efficiency and diagnostic accuracy. Typically, the pass/fail information of a test is defined as the inputs and the fault candidates are defined as the outputs [33] [66] [67]. In order to locate the root cause of a failed test, we take the fine-grained fault syndromes as the inputs of ANNs, which are extracted from failure logs in Step 1. The number of inputs is equal to the number of extracted syndromes. We attempt to extract as much error information as possible from failure logs. The output of an ANN represents the component that can potentially be the root cause of failure. The proposed ANN architecture consists of a group of single-layer networks and each network has one output neuron.

In Step 3, we train the ANNs based on the back-propagation algorithm [72]. Once the architecture of ANNs is defined, the network is ready for training. The output of a neural network is a function of the summation of each input times the corresponding weight. The process of training a neural network involves finding the function and tuning the weights of the network to optimize the network performance. The *performance goal* is the maximum allowable difference between the actual output and the desired output. If the difference between the actual output and the desired value is less than the performance goal, training is complete, otherwise weights are adjusted by the difference in values and the actual output value is re-calculated. The details of the back-propagation algorithm are not shown here [73]. In the training phase, all the training data (successfully repaired cases) are fed to each network. The calculation of weights takes into account all the past cases based on the available repair history. Taking Fig. 6.2 as an example, if 500 syndromes are extracted from failure logs, the input to each network is a binary vector of 500 bits. Assume there are a total of 1000 cases for training. The input to each network is a 500×1000 matrix, and the output is a 1×1000 matrix. Each input vector has an expected value (0 or 1). Based on pre-defined transfer function and weights, we iteratively calculate the outputs and weights until the performance goal is achieved. The training of feed-forward neural networks is supported by the Matlab neural network toolbox. After setting the input data, output data, transfer function, the number of layers of ANNs, and performance goal, the weights can be automated adjusted using Matlab. More details can be found in [71].

In Step 4 (diagnosis step), we examine the output of each network and determine the most likely faulty component. After the training is complete, appropriate weights have been assigned to all the ANNs. Therefore, the outputs of ANNs can be easily calculated on the basis of new inputs. In our architecture, each single-layer network stands for a component on a board. The output value of a network is between 0 and 1, which can be viewed as the probability of the corresponding component being the root cause. If the output value is closer to 1, the component is more likely to be the root cause. For example, suppose that the output value of network N is 0.995, which is the highest among all the networks. Then the component represented by network N is identified to be the root cause of failure.

6.2.3 Advantages of Proposed ANNs-based Method

In this sub-section, the advantages of proposed architecture are presented in terms of the following three aspects.

- Simple structure

In prior work on the use of ANNs for fault diagnosis, input neurons represent fault syndromes and output neurons represent the faulty components. All the data are fed into one large neural network. The training process therefore takes very long time, and the computation requires a large amount of memory. Moreover, the calculation of weights in a large network may not converge, i.e., it may not be possible to find the weights that meet the performance goal with the back propagation algorithm, because the minimum gradient is reached [73].

In our work, we decompose this large neural network into several small neural networks. Each smaller neural network has only one output neuron, and each network represents one faulty component. No changes are made to the input neurons. This architecture is scalable to large datasets. The training time and memory needed for training are significantly reduced without any loss of diagnostic accuracy. In the literature, two-layer neural networks are often employed for classification and pattern-recognition problems [74]. In our work, the key goal is to analyze the relationship between fault syndromes and repair actions. A one-layer network directly connects input syndromes to the output action, and the weight on each connection reflects the contribution of the syndrome towards the decision to take this action. Therefore, one-layer network without biases fit this problem well. Our experiments on a large number of production boards have shown that there is no improvement in diagnostic accuracy if we use a more complex, two-layer neural networks. When biases are included in the network, there is even a drop in the success rate for diagnosis and repair. Commonly used transfer functions in neural networks include the linear, sigmoid, and step functions [32]. The linear function was found to be the most effective for our problem.

- Weight initialization

Weight initialization is very important in the training of ANNs because the correctness of diagnosis and diagnosis accuracy depend on the choice of initial weights. It is difficult to determine the best set of initial weights that provides the highest diagnostic accuracy. Weights are randomly initialized in [33] [66] [67]. In contrast, we initialize the weights based on the occurrence possibilities of fault syndromes. The correct diagnosis rate obtained using the weights initialized by the occurrence possibilities is nearly 10% higher than the rate obtained using randomly initialized weights for the same training data and test cases. This initialization provides a good starting point for the calculation of the weights.

- Automatic extraction of if-then rules

The benefit of using ANNs for fault diagnosis is accompanied with the difficulty of interpreting and processing the data obtained from the network [68]. The actions suggested by ANNs alone do not allow technicians to understand the underlying failure reasons. Moreover, data entry errors may exist in the database and that might lead to incorrect training of the ANNs. Such problems cannot be easily identified if the faulty component identifier is the only information provided by the ANNs. Using the proposed architecture, if-then rules can be automatically extracted in a simple way. The extraction of the rules is based on the weights of ANNs. The fault syndrome (input neuron) with the largest weight of a trained ANN is placed in the if-part, and the corresponding action (output neuron) is placed in the then-part. For example, suppose that the fault syndrome with the largest weight in an ANN is errors occurring at Counter_1, and the corresponding action of this ANN is the replacement of Component_A. Then the if-then rule is that if errors occur at Counter_1, then replace Component_A. This linguistic interpretation helps technicians to understand the connections between syndromes and actions. These automatically extracted rules provide considerable insights in the diagnosis of functional failures, especially in the initial product-ramp phase. Note that if some meaningless rules are generated, we need to scrutinize the training data. The generation of such rules can conceivably be caused by incorrect history records, e.g., a wrong record of the component name.

6.2.4 Simulation Results using ANNs-based Diagnostic System

Experiments were performed on an industrial network switching board that is currently in high-volume production. A total of 1023 successfully repaired boards are selected from the contract manufacturer's database. These boards all failed one particular type of functional test. We select this test for our experiments, since data packages go through all the ASICs on the board in this test; therefore, fault isolation is extremely challenging in this case. The diagnostic accuracy of current diagnostic software for this test is unacceptably low. Failure logs were downloaded from the database and parsed by Perl scripts. The ANNs were implemented using the Matlab2007a toolkit. Experiments and the computation were run on a state-of-the-art Linux server.

In order to validate the effectiveness of the proposed diagnosis method, the 1023 cases (or boards) are separated into two groups: 811 cases for training and 212 cases for diagnosis. The training cases are between the first time interval of production, while the cases for diagnosis are from the second time interval. First, we parse the failure logs of the 811 boards. Fault syndromes are classified using 8 attributes, referred to as dimensions, namely, error ID, mismatched interface, component with error/drop counter, error counter, drop counter, component with interrupt, interrupt bit, and error message code. There are a number of elements in each dimension. For example, there are a total of 13 different error IDs, e.g., 000001A1, 00000116, etc. A total of 1056 syndromes and 98 replaced components are extracted from the 811 training cases. These components include ASICs, external memory devices, daughter boards, resistors, capacitors, inductors, and connectors.

Based on the 811 training cases, a total of 98 single-layer neural networks are built. Each of them has 1056 input neurons and one output neurons. The overall training time for the 98 networks is 4.4 minutes. The diagnosis step for the 212 cases takes only 5 seconds. In the diagnostic results, all the candidate components are listed with the corresponding probability of being the root cause. The probability is equal to the output value of the corresponding network. If the component with the highest probability in this list is the actual faulty component (root cause), the case is considered to be successfully diagnosed. The success rate is shown in Fig. 6.5. For the 1st-time success rate, a case is counted as a successful case only when the actual faulty component gets the highest probability. For the 2nd-time success rate, a case is counted as a successful case, if the actual faulty component gets either the 1st or the 2nd highest probability. A similar definition is used for the 3rd-time success rate. In Fig. 6.5, we can see that 66.98% of the boards can be correctly diagnosed and repaired without ambiguity. If three attempts are allowed in the repair process, 75.98% of the boards can be successfully repaired.

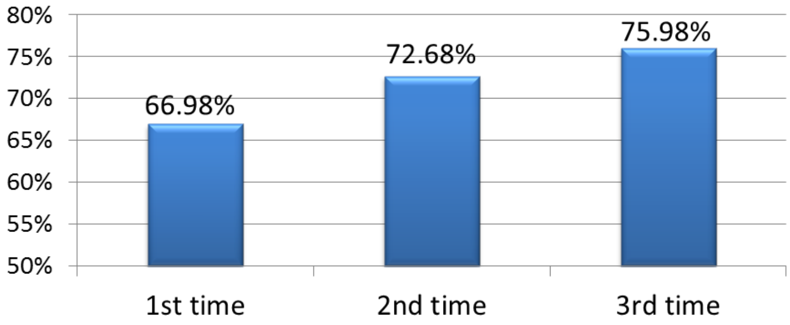


Figure 6.5: Success rate of proposed ANNs for the 212 test cases.

We compare the diagnosis success rate obtained using the proposed ANNs to that obtained using traditional ANNs and Bayesian inference proposed in Chapter 3, with the same training cases and test cases. For traditional ANNs, we consider a 3-layer network with 1056 input neurons and 98 output neurons [67], [27]. The number of neurons in hidden layer is 20. We adjust the number of neurons in the hidden layer to observe the variation of the 1st attempt success rate. A total of six different numbers are used, respectively, 5, 10, 20, 50, 100, 500. When the number of neurons in the hidden layer is equal to 20, we obtain the highest 1st-attempt success rate (58.96%). When the number is equal to 500, we obtain the the lowest 1st attempt success rate (11.79%). In the comparison shown in Table 6.1, we list the highest 1st-attempt success rate achieved by the traditional ANNs. However, it is still lower than the success rate achieved by the proposed ANNs. The difference of 8% in the 1st-attempt success rates and the difference of 15% in the success within three attempts is especially significant for high-volume production.

Table 6.1: Comparison of success rate obtained by ANNs and Bayesian inference

	1st attempt	within 2 attempts	within 3 attempts
Proposed ANNs	66.98%	72.68%	75.98%
Traditional ANNs (20 hidden neurons)	58.96%	60.85%	61.32%
Bayesian inference	53.77%	57.08%	61.32%

In the Bayesian-based method, first, we calculate the prior probability based on the 811 training cases, which is the occurrence probability of a syndrome given a repair action. For example, suppose that a particular repair action has been taken for 10 times, and a given syndrome showed up 8 times in these 10 cases. Then the prior probability is 0.8. All other occurrence probabilities of syndromes, given repair actions, can be calculated in the same manner. After obtaining the prior probabilities, we can calculate the posterior probability based on Bayes' theorem. We take all the fault syndromes into account in the calculation of the posterior probability. For each action, the times of calculation of the posterior probability is equal to the number of fault syndromes. Therefore, the final probability of an action being a correct action is obtained using the information reflected by all the fault syndromes. More details are described in [26]. According to the method described above, we calculate the prior probabilities based on 811 cases and diagnose the same 212 test cases. The success rates within 3 attempts are listed in Table 6.1. We can see that the proposed ANNs-based method has an obvious advantage over Bayesian inference in terms of success rate. The computation times of the proposed ANNs, traditional ANNs, and Bayesian inference are 4.35 minutes, 45.23 minutes, and 2.48 minutes, respectively. Experiments are implemented using Matlab2007a on a state-of-the-art Linux server.

For this challenging functional test, current diagnostic software used in the production line considers any component that exhibits error as a fault candidate, and no suggestions are provided regarding which component is more likely to be the root cause. Compared to the 1st-time success rate of the current diagnostic software, the 1st-attempt success rate for the proposed method is about two times higher, and significantly higher than even the 3rd-time success rate of the diagnostic software that is currently deployed. Based on the repair suggestions provided by current diagnostic software, the debug time for this particular functional test is as high as several weeks, which is clearly not feasible in practice. Debug efficiency is expected to be improved with accurate repair suggestions provided by the proposed method.

6.3 Intelligent Diagnosis using Support Vector Machines (SVMs)

6.3.1 Introduction to SVMs

A support vector machine (SVM) is a supervised machine learning algorithm proposed by Vapnik in 1995 [69]. It has a number of theoretical and computational merits, for example, simple geometrical interpretation of the margin, uniqueness of the solution, statistical robustness of the loss function, modularity of the kernel function, and overfitting control through the choice of a single regularization parameter. It has been applied to various fields as a powerful pattern classification tool, e.g., fault diagnosis, image classification, face recognition, etc. [75], [76]. A brief introduction to SVMs is presented below.

The goal of SVMs is to define the optimal separating hyperplane (OSH) to separate two classes. The vectors from the same class fall on the same side of an optimal separating hyperplane, and the distance from the closest vectors to an optimal separating hyperplane is the maximum among all the separating hyperplanes. An important and unique feature of this approach is that the solution is only based on those vectors that are the closest to the OSH. These vectors are called support vectors. Therefore, the classification accuracy does not depend on the size of training data set. In Fig. 6.6, a set of two-dimensional vectors is separated by an optimal separating hyperplane. The left side of the OSH is the class labeled by $y = +1$. The other class on the right side is labeled by $y = -1$. After the OSH is determined, new vectors can be classified using the SVM. For example, given a new vector (illustrated by the square with a question mark in Fig. 6.6), SVMs will place it into the class of $y = -1$, since it falls on the right side of the OSH.

Specifically, the OSH is calculated in the following way. let $(\boldsymbol{x}_i, y_i)$, i=1,2, ..., n be a set of training examples, and $\boldsymbol{x}_i \in \boldsymbol{R}^d$, $y_i \in \{-1, +1\}$. The vector $\boldsymbol{x}_i$ is considered as input, and d is the dimension of the input vectors. Each input vector belongs to one of the two classes. One is labeled by $y = 1$; and the other is labeled by $y = -1$. If the set can be linearly separated, there must be a hyperplane satisfying (1):

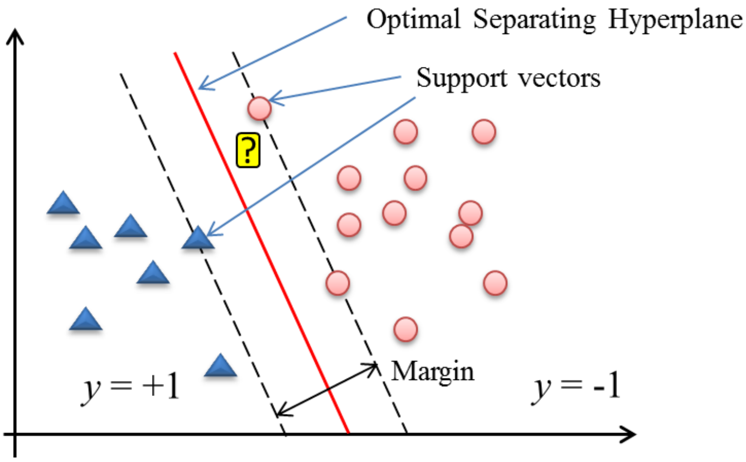


Figure 6.6: Geometric illustration of the optimal separating hyperplane.

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) > 0, \quad i = 1, ..., N \tag{6.1}$$

Where $\boldsymbol{w}$ is a d-dimensional vector, b is a scalar, and $\boldsymbol{w} \cdot \boldsymbol{x}_i$ is the inner product of $\boldsymbol{w}$ and $\boldsymbol{x}_i$. In order to calculate the OSH, the distance from the closest vector to the hyperplane needs to be maximized. According to [69], the distance from the closest vector to the hyperplane is $1/\|\boldsymbol{w}\|$, and $2/\|\boldsymbol{w}\|$ is defined as margin; see Fig.6.6. The margin is a measure of the generalization ability. The larger the margin is, the better the generalization of the classifier is expected to be [69]. In the literature, the linear SVMs problem is often formulated as follows.

$$\text{Minimize: } \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{6.2}$$

Subject to

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1, \quad i = 1, ..., N \tag{6.3}$$

This quadratic optimization problem can be solved using Lagrange multipliers. Let $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_N)$ be the N non-negative Lagrange multipliers associated with each input vector $\boldsymbol{x}_i$. Detailed calculations can be found in [69]. Here we simply assume that the solution of the Lagrangian problem is $\boldsymbol{\alpha}^0 = (\alpha_1^0, ..., \alpha_N^0)$. Then the solution of the original optimization problem is:

$$\boldsymbol{w}^0 = \sum_{i=1}^{N} \alpha_i^0 y_i \boldsymbol{x}_i \tag{6.4}$$

$$b^0 = 1 - \boldsymbol{w}^0 \cdot \boldsymbol{x}_i \tag{6.5}$$

The decision function of the OSH can be written as

$$f(\boldsymbol{x}) = \operatorname{sgn}(\sum_{i=1}^{N} \alpha_i^0 y_i \boldsymbol{x}_i \cdot \boldsymbol{x} + b^0) \tag{6.6}$$

To classify a new vector, if $f(\boldsymbol{x}) > 0$, the vector belongs to the class labeled by $y = +1$. If $f(\boldsymbol{x}) < 0$, the vector belongs to the class labeled by $y = -1$.

When data are not linearly separable, we can either introduce slack variables in linear SVMs or use nonlinear SVMs. For the slack variables based solution, the optimization problem becomes:

$$\text{Minimize: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i \tag{6.7}$$

Subject to

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i = 1, ..., N \tag{6.8}$$

Where the slack variable $\xi_i \geq 0$, and C is penalty parameter. The purpose of the slack variables is to allow misclassifications, which have their corresponding $\xi_i > 1$. The value of parameter C is adjustable. A larger C means a higher penalty to misclassifications. The choice of C should be on the balance between overfitting and underfitting. The effects of penalty parameter C on the diagnostic accuracy are presented in Section VI.

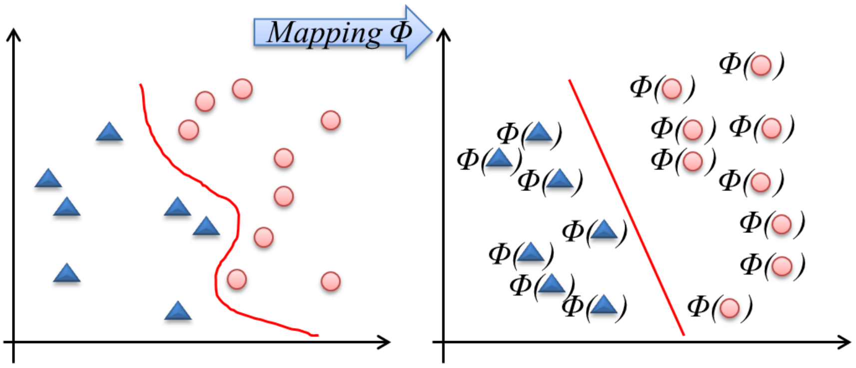


Figure 6.7: Illustration of the nonlinear SVMs.

In nonlinear SVMs, the input vector is mapped to a high-dimensional feature space through some nonlinear mapping, and the OSH is constructed in the feature space. This idea is illustrated in Fig. 6.7. An important advantage of the SVM is that it is not necessary to explicitly implement the transformation and to determine the separating hyperplane in the possibly very-high dimensional feature space. Instead a kernel representation can be used, where the solution is written as a weighted sum of the values of certain kernel function evaluated at the support vectors. The choice of kernel is crucial for the success of SVMs. Polynomial kernel, Gaussian kernel, and exponential kernel are three widely used kernel functions [75]. In the simulation results in this thesis, we compare the diagnostic accuracy of SVMs based on all three kernel functions.

Originally SVMs were designed for linear binary classification problems. In practice, classification problems are not limited by two classes. In the board-level fault diagnosis, the number of fault candidates (classes) is typically a few hundreds. In [77], a modified design of SVMs was proposed in order to incorporate multi-class learning. Besides this, an alternative is to combine several binary SVMs. "One against one" provides pairwise comparisons between classes. "One against all" compares a given class with all the other classes. According to a comparative

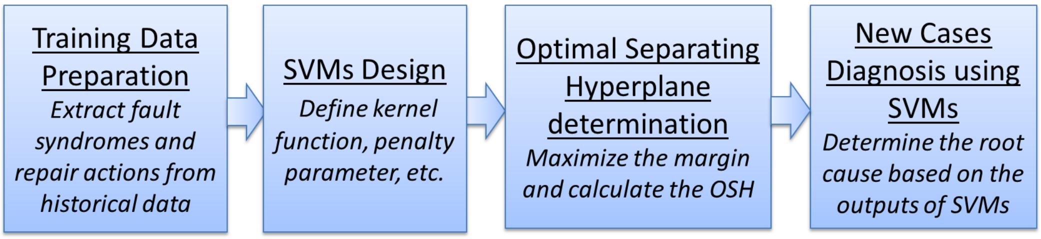


Figure 6.8: The diagnosis flow using SVMs.

study in [78], the accuracies of these methods are almost the same. Therefore, we choose the “one against all” in our problem, which has the lowest computation complexity.

6.3.2 Application of SVMs in Functional Diagnosis

The SVMs-based diagnosis flow consists of the four steps described in Fig. 6.8. Generally speaking, a set of training data (fault syndromes and corresponding repair actions) are first prepared, which are obtained from the repair history in the manufacturing database. Then SVMs determine the OSH based on the training data. After the OSH is determined, the SVMs-based diagnostic system is ready to diagnose new cases.

The diagnostic flow using SVMs is very similar to the flow in ANNs-based diagnosis. In the preparation of training data, the fault syndromes in the log files are extracted as the inputs, and the corresponding repair action is taken as the output. The multi-class SVMs are designed in the one-against-all manner, i.e., if there are M replaced components, there will be M SVMs. Each SVM represents one component. All the training cases are used to determine the OSH of each SVM. A pair of data (fault syndromes and the corresponding replaced component) is extracted from each case. Fault syndromes are formatted to a binary vector, which is used as the input of a SVM. The length of the vector is equal to the number of fault syndromes. Entry “1” means the fault syndrome appears, otherwise the entry is “0”. Output is either “-1” or “+1”. If the replaced component of the current case happens to be the component represented by the SVM, the output is “+1”, otherwise it is “-1”.

The choice of kernel function and penalty parameter affects the performance of SVMs. However, there are no generic rules to select the best kernel and other parameters of SVMs for a specific problem [79],[80]. In [79], an evolutionary algorithm was presented for the selection of kernel functions. The performance of kernel functions was evaluated using only two data sets. The computational complexity depends on the number of generations in the evolution. There is no guarantee that the selected kernel function can result in the optimum solution of a classification problem. In this work, we determine the best design of SVMs in a heuristic way. According to extensive experimental results, we find that the SVMs with a linear kernel and a relatively small penalty parameter provide the highest accuracy for board-level fault diagnosis. The determination of the OSH can be considered as the training of SVMs. The OSH is determined by solving the quadratic optimization problem described in Section III. According to the given inputs and outputs, we can calculate the parameters, $\boldsymbol{w}$ and b using Matlab. An open-source SVMs toolbox is provided in [81].

After the training is completed, the parameters $\boldsymbol{w}$ and b are determined. Given a new input vector, we can calculate the output of the SVM using the decision function (Eq. (6.6)). In the diagnosis step, we rank the output of all the SVMs, and select the component represented by the SVM with the largest output as the root cause. For example, assume there are three fault candidates (A, B, C). The decision functions of the OSH of three SVMs are $f_A(\boldsymbol{x}), f_B(\boldsymbol{x}), f_C(\boldsymbol{x})$, respectively. Given a new input vector $\boldsymbol{x}_0$, $f_A(\boldsymbol{x}_0) = 0.1, f_B(\boldsymbol{x}_0) = 0.8, f_C(\boldsymbol{x}_0) = -0.2$. Then the most likely root cause of the failure is component B. This is different from the original definition of the decision function in Eq. (6.6). In the proposed multi-class SVMs-based diagnostic system, the larger the output of the SVM is, the more likely it is that the component represented by the SVM is the root cause.

6.3.3 Simulation Results using SVMs-based Method

Experiments were performed on the same industrial network switching board as described in Section 6.2.4. The SVMs-based learning system is built to learn the repair knowledge from the existing successful cases, which is not easily documented and grasped by technicians. The SVMs algorithm was implemented using the Matlab2007a toolbox. In the following, diagnostic results using the SVMs-based learning system are first presented. Diagnostic results vary across different design of the SVMs, e.g., kernel function, penalty parameter, etc. Extensive simulation results show that the SVMs-based diagnostic system with a linear kernel and a small penalty parameter provides the best results. Next, the SVMs-based diagnostic system and the ANNs-based diagnostic system are compared using the same training and testing set.

As before, 811 cases are used for learning and the remaining 212 cases are used to evaluate the diagnostic accuracy of the learning system. A total of 1056 fault syndromes are extracted from the failure logs of the 811 learning cases. The number of faulty components that appear in the learning set is 98. Based on the above information, 98 one-against-all SVMs are constructed. Each SVM represent one component that appears in the learning cases.

We first apply different types of kernel functions in the SVMs-based learning system. Polynomial kernel (degree=2), Gaussian kernel, and Exponential kernel are used in the experiments. The diagnostic results are listed in Table 6.2. The

Table 6.2: Diagnosis results using different kernel functions.

	Polynomial	Gaussian	Exponential
Empirical Success Rate (SR)	94.7%	94.2%	93.96%
Test set SR (one attempt)	72.64%	61.79%	57.08%
Test set SR (two attempts)	78.3%	72.64%	67.45%
Test set SR (three attempts)	81.13%	75.47%	70.28%
Training time (minutes)	1.75	40.82	158.98

empirical success rate (SR) is the ratio of the number of correctly diagnosed cases to the number of all cases in the training set. It reflects how good the learning system fits the empirical data. The test set SR is the ratio of the number of correctly diagnosed cases to the number of all the cases in the testing set. In practice, when the first attempt of repair fails, subsequent attempts are typically allowed. Therefore, the success rates within three attempts are listed in Table 6.2. We can see that polynomial kernel function provides the highest empirical SR and the highest SRs on test sets. The first attempt SR on the test set of SVMs with polynomial kernel is up to 72.64%. In addition, the training time is much less than for the other two kernels.

From Table 6.2, we can see that the polynomial kernel function fits the fault diagnosis problem the best. We conducted further experiments using the polynomial kernel to determine other parameters in the SVMs. Diagnosis results for different degrees of polynomial kernel are shown in Table 6.3. As we increase the degree, the computational complexity and training time increase, but the SR on test sets drops slightly. The reason is that high-degree polynomial kernels lead to overfitting. Next, we focus on the polynomial kernel with degree equal to 1 (linear) and 2 (quadratic). We randomly select 811 cases from the all the available cases, and evaluate the remainder 212 cases using linear-kernel and quadratic-kernel SVMs. The experiment is repeated 100 times. The distribution of the first attempt SR on

Table 6.3: Diagnosis results of polynomial kernel with different degrees.

Polynomial kernel	degree=1	degree=2	degree=3	degree=5
Empirical SR	92.23%	94.7%	94.57%	94.45%
Test set SR (one attempt)	72.64%	72.64%	72.17%	68.87%
Test set SR (two attempts)	77.83%	78.3%	76.89%	76.89%
Test set SR (three attempts)	81.13%	81.13%	81.13%	80.19%
Training time (minutes)	1.43	1.75	2.41	3.57

test sets of the 100 trials is illustrated in Fig. 6.9. In the 100 trials, the highest SR obtained using the linear kernel is 79.24%, the lowest SR is 66.51%, and the average is 73.33%; the highest SR obtained using quadratic kernel is 77.36%, the lowest SR is 65.09%, and the average is 72.76%. According to the above analysis and simulation results, the SVMs based diagnostic system with linear kernel provides the highest success rate and fits our fault diagnosis problem the best.

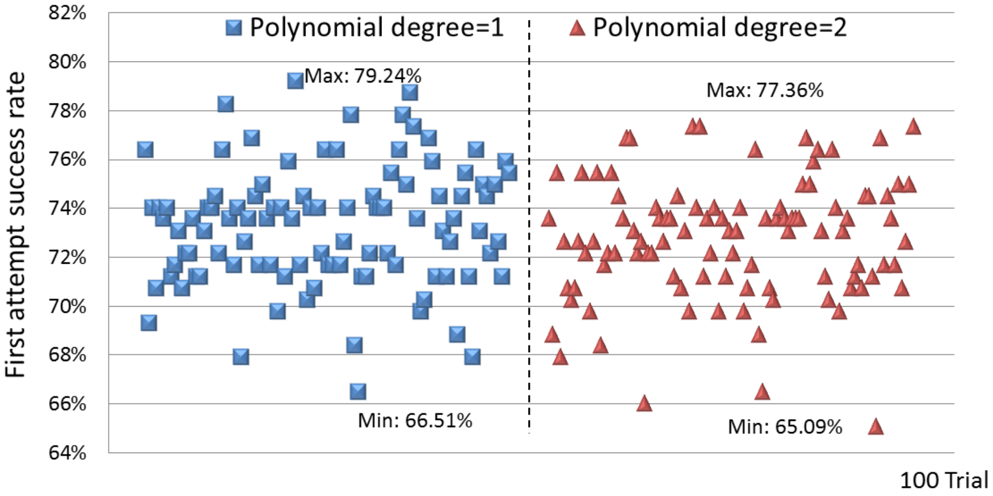


Figure 6.9: The distribution of the first-attempt SR in 100 trials.

Another interesting parameter in the design of the SVM model is the penalty parameter C; a larger C corresponds to the assignment of a higher penalty to misclassification. The empirical success rate can be improved by setting a larger penalty parameter, but this may lead to overfitting, and the SR on test sets might

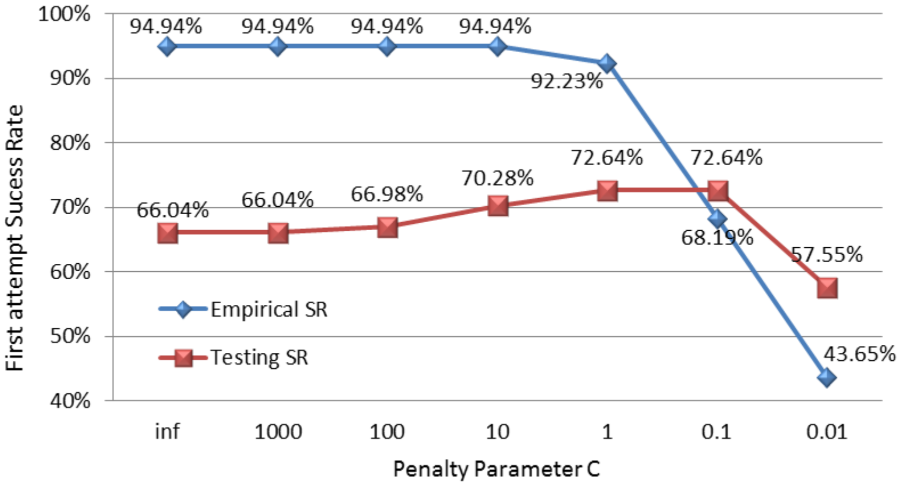


Figure 6.10: The effects of penalty parameter on the success rate.

actually drop. The effects of penalty parameter on the empirical SR and SR on test sets are illustrated in Fig. 6.10. The SVMs-based diagnostic system with linear kernel is used in this experiment. The penalty parameter decreases from infinity to 0.001. When C is less than 1, the empirical SR significantly drops as the decrease of C. The SR on test sets starts dropping after C is less than 0.01. According to Fig. 6.10, the value around 1 is a good choice for the penalty parameter for this problem. With three attempts, the diagnostic success rate of the diagnostic software currently used in the production line is lower than 50%. The success rate is improved more than 30% using the proposed diagnostic system.

6.4 Comparison of two algorithms

6.4.1 Diagnostic Results Comparison

Based on the same 811 training cases and 212 testing cases, results for the two diagnostic systems are listed in Table 6.4. Although the empirical SR of the SVMs-based system is lower than that of the ANNs-based system, the SRs on test sets of

SVMs-based system are higher than that of the ANNs-based system. This supports the notion that SVMs have better generalization performance (i.e., success rate on test sets) than ANNs. In terms of training time, the SVMs-based system is about three times faster than the ANNs-based system.

Table 6.4: The comparison of two diagnostic systems.

	ANNs	SVMs (linear kernel)
Empirical SR	95.19%	92.23%
Test set SR (one attempt)	66.98%	72.64%
Test set SR (two attempts)	72.68%	77.83%
Test set SR (three attempts)	75.98%	81.13%
Training time (minutes)	4.4	1.43

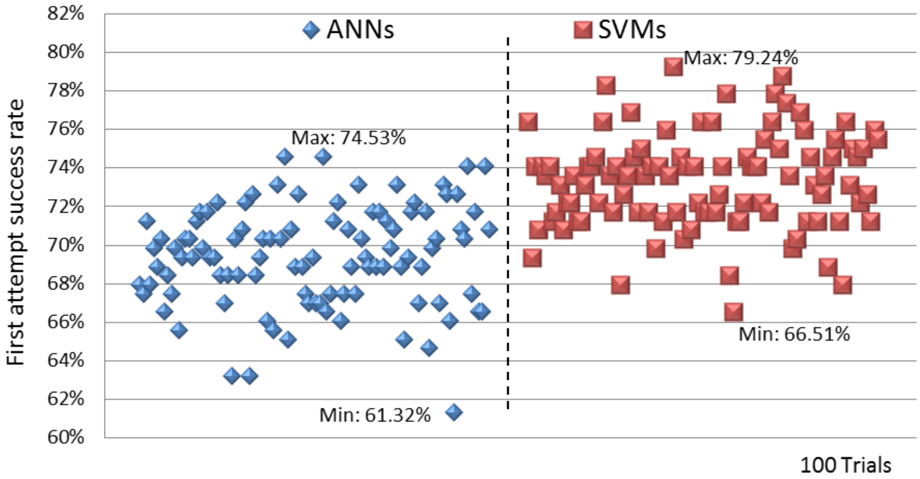


Figure 6.11: The distribution of the first attempt SR of 100 trials.

The diagnostic accuracy is related to the number of training cases and the types of training cases. For some of the training cases, it is easy for the diagnostic system to derive relationships between syndromes and actions, but some other training cases might introduce conflicts among existing relationships. Therefore, from all the 1023 available cases, we randomly select 811 cases for training and use the remaining 212 cases for diagnosis. A total of 100 trials are generated. The first attempt success rate of 100 trials obtained with two learning systems are illustrated

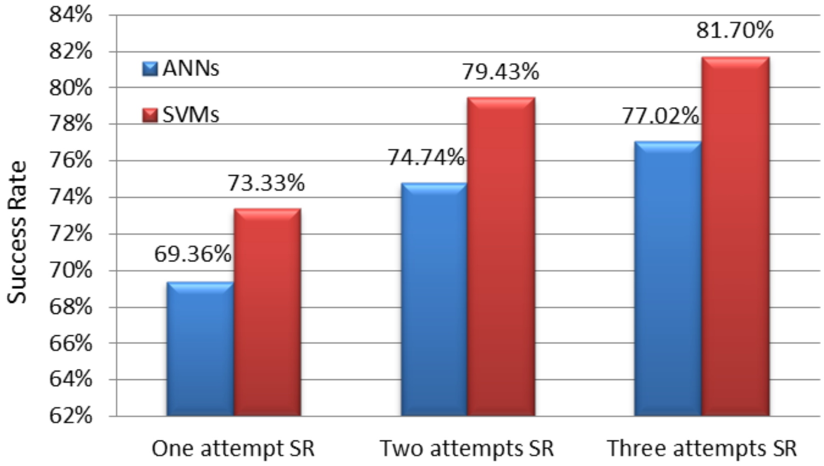


Figure 6.12: The average success rate over 100 trials.

in Fig. 6.11. We can see that the first attempt SR of SVMs-based system is higher than that of ANNs-based system in almost all trials. Specifically, there is only one trial where the diagnostic accuracy of SVMs is lower than that of ANNs. The average of success rate of the 100 trials is shown in Fig. 6.12.

In addition, we analyze the effect of training set size on the success rate of two diagnostic systems. The test set consists of 212 cases and the size of training set is decreased from 800 to 400. The variation of SR with decrease in the training set size is illustrated in Fig. 6.13. The success rate is the average of 10 trials. We can see that the first-attempt SR of the SVMs-based system does not drop as much as that of the ANNs-based system. This is another advantage of the SVMs-based diagnostic system. The diagnosis success rate is not significantly affected by the size of the training set. This feature benefits the diagnosis process, especially at the initial manufacturing phase when a sufficient number of learning cases are not available. The theoretical explanation for this finding is that the optimal separating hyperplane of a SVM is determined by the support vectors falling on the edge, rather than all the vectors (training cases).

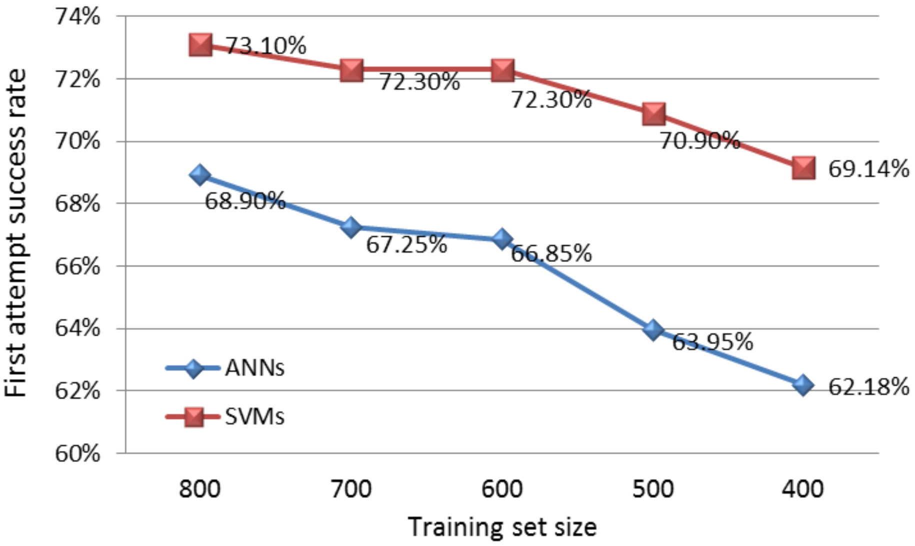


Figure 6.13: The variation of SR with the decrease of the training set size.

In terms of diagnostic accuracy and the training time, the SVMs-based method is superior to the ANNs-based method. However, the relationship between faults and syndromes can be easily derived from the straightforward ANNs-based method as discussed in Section 6.2. The critical syndromes caused by a faulty component can be determined based on the weights of ANNs. This is very helpful for diagnosticians in understanding failure mechanisms, especially at the initial production phase.

6.4.2 Theoretical Comparison

The development of ANNs followed a heuristic path, with applications and extensive experimentation preceding theory. In contrast, the development of SVMs first involves sound statical learning theory, then implementation and experiments. The fundamental difference between the two approaches is that ANNs minimize empirical risk (misclassifications in the training set), but SVMs minimize the risk of misclassifying cases of the test set. Therefore, SVMs are less prone to overfitting and often outperform ANNs in practice.

ANNs use the back-propagation algorithm to search for the minimum of the error function (mean squared error) in the weight space. The combination of weights, which minimizes the error function, is considered to be a solution of the learning problem. However, the solution obtained using the back-propagation algorithm can only converge to local minimum. In contrast, the solution to an SVM is global and unique, because SVMs are formulated as a convex quadratic optimization problem.

In addition, the computational complexity of SVMs does not depend on the dimensionality of the input space. SVMs do not attempt to control model complexity by keeping the dimension of the input vectors small. In order to linearly separate the cases, SVMs sometimes transfer the input space to a higher-dimensional feature space. The model complexity of SVMs is automatically determined in the quadratic programming procedure by the selection of the support vectors. Another advantage of SVMs is its geometric interpretation. The determination of the OSH can be viewed as the maximization of the margin between support vectors from different classes.

6.5 Summary

We have presented an intelligent diagnosis method based on machine learning algorithms to overcome the difficulties involved in the debug and repair of board-level functional failures. A simple but effective neural network architecture has been developed such that the network can be quickly trained and high diagnostic accuracy can be achieved. The ANNs are trained to learn the fundamental connections between fault syndromes and repair actions, which are not easy to infer from visual inspection of log files and large datasets. In order to achieve high diagnostic accuracy and resolution, the architecture of ANNs has been carefully designed.

Besides ANNs, we explored another machine learning algorithm, namely SVMs, which is able to provide the optimal separating hyperplane in classification. The selection of the parameters in the SVM model follows a heuristic manner. Through extensive simulation results, we found that the SVMs with a linear kernel and a small penalty parameter provide the highest diagnostic accuracy. Using the same training and testing sets, the diagnostic accuracy of SVMs is nearly 5% higher than that of ANNs, and the training time is about three times faster. This diagnostic system is expected to be implemented in a commercial manufacturing line in the near future to accelerate the debug process.

Chapter 7

Conclusions and Future Work

Manufacturing test and fault diagnosis significantly affect product quality, time-to-market, and the competitive strength of a company. The ever-increasing integration density and clock frequency make it more difficult to ensure product quality at the board and system level. Existing board-level testing and diagnostic strategies are insufficient to meet the requirements of the user community for high-reliability products. Functional test has been increasingly used, but the diagnosis of functional failure in large systems is very challenging. The high manufacturing cost caused by low product yield and failed repair attempts motivates the need for an efficient and effective diagnostic system. In this work, we have conducted multi-pronged research to address pressing issues in board-level functional diagnosis.

The research reported in this thesis presents the optimization of fault-insertion test and intelligent diagnostic methods targeting functional failures. The goal of optimization of fault-insertion test is to select the most effective fault insertion points for system reliability assessment and fault diagnosis. The pin-level fault model has been proposed to generalize the erroneous behavior of internal defects, and thus accelerate reliability assessment. Diagnosis-oriented fault-insertion points are defined and selected to create the representative and distinguishable syndromes, in order to facilitate fault diagnosis. Considering the time of error occurrence, an effective description of functional failure, error flow, is proposed to improve diagnostic resolution.

The goal of our research on intelligent diagnostic methods has been to reduce the dependence on the time-consuming and ineffective human effort during the debug-knowledge acquisition process. Machine learning algorithms and statistical inference have been used for diagnosis. Quantifiable improvement over current diagnostic software has been achieved. The proposed solution is not limited to a particular telecommunication board. It is generic and applicable to various products. Although the goal of this work was to solve the problems at the board level, it is also scalable to large electronic systems.

7.1 Thesis Contributions

This thesis aims at solving practical problems and tackling fundamental issues in testing and diagnosis faced at electronics system companies. Integer linear programming, statistical inference, and machine learning have been successfully applied to test optimization and fault diagnosis.Our results show that the optimal solutions to FIT and automated intelligent diagnosis greatly improve test efficiency and diagnostic accuracy. Product yield improvement and repair cost reduction are expected through the implementation of the proposed solutions in the manufacturing line and for debug.

Chapter 2 proposed a pin-level fault model to represent the erroneous behavior of physical defects, and presented an optimization algorithm to determine optimal fault locations and fault types for fault-insert test. Through analyzing the effects of physical defects to system operation and the effects of pin-level faults, we found that the same failure caused by internal physical defects can be successfully mimicked by inserting the pin-level faults, which verifies the model's correctness. This high-level fault model and optimized fault insertion locations eliminate the difficulties associated with exercising the prohibitively large number of physical defects in FIT. Therefore, FIT-based system reliability assessment is accelerated, and we have high confidence in the assessment results, as the percentage of physical defects covered during FIT is known.

Chapter 3 presented a diagnostic framework based on FIT and Bayesian theorem. Bayesian theorem infers the occurrence probability of an event based on the occurrence of other evidence or observations. This fits the fault diagnosis problem well, where we want to infer the probability of a fault candidate being the root cause based on the observed fault syndromes. One difficulty associated with Bayesian-based diagnosis is to collect the occurrence probability of a fault syndrome given a fault. This conditional probability is critical in the Bayesian inference. Previously, it was approximated based on diagnosticians' experience or a pre-assumed distribution function. In this work, we take the advantage of fault-insertion test to create real faulty samples for the collection of conditional probabilities. The conditional probabilities derived from real faulty samples make Bayesian inference based diagnosis more accurate. In addition, we found that diagnostic accuracy can be improved by using multiple tests.

Chapter 4 presented an effective way to describe functional failure using the concept of error flow. Functional failure is difficult to diagnose without the information of circuit functionality. However, it is impossible for diagnosticians to grasp the complex functionality of each product. Based on the analysis of the functional failure, we found that the error occurrence time is another important factor, besides the error occurrence location. Therefore, the locations where errors are observed are ordered in terms of error occurrence time. This sequence is then referred to as error flow. Error flow reflects data propagation, which is related to the functional characteristics of a circuit. Functional characteristics are extracted and considered in fault diagnosis by using error flow. Functional failure analysis and fault identification thus become easier.

Chapter 5 presented a diagnosis-oriented fault-insertion point selection method.

We defined the diagnostic-efficient fault-insertion points, where faults should be inserted, in order to create the representative and distinguishable fault syndromes. The syndromes produced by the faults inserted at the selected points are the most similar to the syndromes caused by internal defects, while they are different from the syndromes of other faulty components. Higher diagnosis accuracy is achieved compared to the randomly selected fault-insertion points. The proposed concept of diagnosis-oriented fault-insertion points provides valuable guidelines for FIT-based diagnosis.

Chapter 6 presented an automated and intelligent diagnosis method based on machine learning to address pressing issues in board-level functional diagnosis. Artificial neural networks and support vector machines have been used to automatically learn the diagnostic knowledge from repair history and identify the root cause of a newly failed board. Both diagnostic systems are scalable and can be rapidly trained. The relationship between faulty components and the syndromes can be easily interpreted from the ANNs based diagnostic system, but the diagnostic accuracy achieved by SVMs-based system is slightly higher. The machine learning algorithms are able to extract critical syndromes from large datasets. Therefore, the proposed automated method eliminates human involvement in fault syndrome collection and classification. Based on the diagnostic results, fault candidates are ordered based on their probabilities of being the root cause. Quantifiable improvement over current diagnostic software is achieved in terms of diagnostic accuracy and resolution. The SVMs-based diagnostic system is currently being implemented on the production line. Rapid diagnosis and high success rate are expected for real systems.

7.2 Future Work

Board and system level test and diagnosis is attracting increasing attention, as we move towards more complex high-level integration. Test optimization and diagnosis automation are pressing needs in industry. The proposed intelligent diagnosis method in this thesis is a very promising solution for diagnosis automation. It can be advanced along the following directions.

- **Fault syndrome pre-processing**

Fault syndromes used to train the diagnostic system play an important role in fault diagnosis. It is difficult to determine the critical syndromes among thousands of syndromes, when we analyze a failing case. In this work, we included all the syndromes when we trained the ANNs-based and the SVMs-based system, since one advantage of the proposed diagnostic system is a simple and effective architecture. Thus all the syndromes can be handled at the same time. However, we found that some syndromes are highly correlated to others. An improvement in the diagnosis results can potentially be achieved if we remove the highly correlated syndromes, or perform diagnosis with a subset of syndromes in multiple steps. It will be interesting to carry out further analysis on the correlation between syndromes and the use of subsets of syndromes.

- **SVMs design optimization**

Accurate diagnostic results have demonstrated the superior performance of SVMs-based diagnostic system. However, the proposed SVMs-based diagnostic system is designed in a heuristic manner. There is no guarantee that the selected kernel function and the penalty parameter are the best possible solution. From previous analysis, we know that these parameters greatly affect diagnostic accuracy. Therefore, diagnostic accuracy can conceivably be improved by optimizing the parameters in the SVMs. In the literature, a number of solutions were proposed to optimize the parameters in SVMs [82], [83], [84], [85]. One method is to utilize the leave-one-out evaluation. One case is taken out from the training test and used to test the SVMs each time. The misclassification rate of leave-one-out test is considered to be a measure of the model generalization capability [84]. Another solution is based on the use of Bayes' theorem to select an optimal penalty parameter and optimal kernel parameters [85]. SVMs are integrated with Bayesian evidence framework, within which the penalty parameter is optimized by maximizing the posterior probability of the model. The posterior probability is calculated via the Bayesian theorem. It is a direction to explore the optimization methods in SVM design.

- **Hybrid solution based on a combination of rule-based and case-based diagnosis**

Fault diagnosis solutions are likely to be incomplete in the sense that some fault cases do not fit the diagnostic system. Even if we can tune the diagnostic system to fit all the training cases with extremely high computational complexity, the trained system can be an overfit for new test cases. A potential solution is to combine case-based diagnosis with the rule-based diagnosis. For those cases that cannot be trained or correctly diagnosed, we can define special rules to describe the fault syndromes and corresponding repair actions. In this way, case-based diagnostic system can handle most cases in an accurate and efficient manner, and special cases are covered by the pre-defined rules. Moreover, derivative rules can be easily obtained in a symmetric system. For example, in a symmetric network switching system, there are 24 ports and one data path from each port. If one faulty scenario occurring on one port is observed, similar scenarios on the other ports can be possibly derived.

Bibliography

[1] Y.-T. Lin. Economic design for manufacturing system test and field maintenance. In *Southeastern Symposium on System Theory*, pages 40–44, 2005.

[2] S. Chau. Fault injection boundary scan design for verification of fault tolerant systems. In *IEEE International Test Conference*, pages 677–682, 1994.

[3] National Instruments. *Using Fault Insertion Units (FIUs) for Electronic Testing.* ftp://ftp.ni.com/pub/devzone/pdf/tut_10340.pdf, 2010.

[4] K. P. Parker. A new probing technique for high-speed/high-density printed circuit boards. In *IEEE International Test Conference*, 2005.

[5] M.E. Amyeen, S. Venkataraman, and M.W. Mak. Microprocessor system failures debug and fault isolation methodology. In *IEEE International Test Conference*, 2009, Paper 15.1.

[6] T. Vo, Z. Wang, T. Eaton, P. Ghosh, L. Young, W. Wang, H. Jun, D. Singletary, and X. Gu. Design for board and system level structural test and diangosis. In *IEEE International Test Conference*, pages 1–10, 2006.

[7] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing.* Kluwer Academic Publishers, Norwell, MA, 2000.

[8] G. D. Robinson. Board and system test with SoC DFT. In *IEEE International Test Conference.* Panel 1.3, 2005.

[9] IEEE Standard 1149.1, Test Access Port and Boundary-Scan Architecture. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=938734.

[10] IEEE Standard 1149.6, Boundary-Scan Tesing of Advanced Digital Networks. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1196298.

[11] I. Reis, P. Collins, and M. V. Houcke. On-line Boundary-Scan testing in service of extended products. In *IEEE International Test Conference.* Paper 2.4, 2009.

[12] K. P. Parker. Defect coverage of Boundary-Scan test: What does it mean when a Boundary-Scan test passes? In *IEEE International Test Conference*, 2003, Paper 49.1.

[13] B. Eklow, A. Hosseini, K. Chi, S. Pullela, T. Vo, and H. Chau. Simulation based system level fault insertion using co-verification tools. In *IEEE International Test Conference*, pages 704–710, 2004.

[14] Z. Conroy, G. Richmond, X. Gu, and B. Eklow. A practical perspective on reducing ASIC NTFs. In *IEEE International Test Conference*, pages 343–349, 2005.

[15] Hardware Troubleshooting for Catalyst 8540/8510 MSRs. http://www.cisco.com/application/pdf/paws/21334/85TsgId.pdf.

[16] National Instruments. Strategies for lowering the cost of manufacturing test. http://zone.ni.com/devzone/cda/tut/p/id/2908, 2006.

[17] C. Constantinescu. Impact of deep submicron technology on dependability of VLSI circuits. In *International Conference on Dependable Systems and Networks*, pages 205–209, 2002.

[18] N. Lopez-Benitez and K.S. Trivedi. Multiprocessor performability analysis. *IEEE Transactions on Reliability*, volume 42, pages 579-587, 1993.

[19] J. Arlat, A. Costes, Y. Crouzet, J. Laprie, and D. Powell. Fault injection and dependability evaluation of fault-tolerant systems. In *IEEE Transactions on Computers*, volume 42, pages 913–923, 1993.

[20] B. Huang, M. Rodriguez, M. Li, and C. Smidts. On the development of fault injection profiles. In *Reliability and Maintainability Symposium*, pages 226–231, 2007.

[21] S. Chau. Fault injection scan design for enhanced VLSI design verification. In *IEEE VLSI Test Symposium*, pages 109–111, 1993.

[22] S. Sutherland. *The Verilog PLI Handbook*. Springer, New York, NY, 2002.

[23] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *IEEE International Test Conference*, pages 61–70, 2004.

[24] P. Kudva, J. Kellington, P. Sanda, R. McBeth, J. Schumann, and R. Kalla. Fault injection verification of IBM POWER6 soft error resilience. In *Architectural Support for Gigascale Integration (ASGI) Workshop*. San Diego, CA, 2007.

[25] K. K. Goswami, R. K. Iyer, and L. Young. DEPEND: A simulation-based environment for system level dependability analysis. *IEEE Transactions on Computers*, volume 46, pages 60-74, 1997.

[26] Z. Zhang, Z. Wang, X. Gu, and K. Chakrabarty. Board-level fault diagnosis using Bayesian inference. In *IEEE VLSI Test Symposium*, pages 244–249, 2010.

[27] J.H. Lim, H. C. Lui, A.H. Tan, and H.H. Teh. INSIDE: a connectionist case-based diagnostic expert system that learns incrementally. In *IEEE International Joint Confference on Neural Networks*, volume 2, pages 1693–1698, 1991.

[28] Z. Zhang, Z. Wang, X. Gu, and K. Chakrabarty. Board-level fault diagnosis using an error-flow dictionary. In *IEEE International Test Conference*, 2010, Paper 16.3.

[29] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Press, Piscataway, NJ, 1990.

[30] Mentor Graphics, Tessent Diagnosis, http://www.mentor.com/products/silicon-yield/products/diagnosis.

[31] W.G. Fenton, T.M. McGinnity, and L.P. Maguire. Fault diagnosis of electronic systems using intelligent techniques: a review. In *IEEE Trans. on System, Man, and Cybernetics*, volume 31, pages 269–281, 2001.

[32] G. F. Luger and W. A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Addison-Wesley, Reading, MA, 1998.

[33] C. O'Farrill, M. Moakil-Chbany, and B. Eklow. Optimized reasoning-based diagnosis for non-random, board-level, production defects. In *IEEE International Test Conference*, pages 173–179, 2005.

[34] L. Barford, V. Kanevsky, and L. Kamas. Bayesian fault diagnosis in large-scale measurement systems. In *IEEE Instrumentation and Measurement Technology Conference*, volume 2, pages 1234–1239, 2004.

[35] H. Fang, Z. Wang, X. Gu, and K. Chakrabarty. Mimicking of functional state space with structural tests for the diagnosis of board-level functional failures. In *IEEE Asian Test Symposium*, 2010.

[36] Cisco Line Cards. http://www.cisco.com/en/US/products/hw/modules/ps2710/prod_module_series_home.html.

[37] International Electronics Manufacturing Initiative Organazition. *iNEMI Technolog Roadmaps.* http://thor.inemi.org/webdownload/roadmapping/2009_Roadmap, 2009.

[38] B. Fenton, M. McGinnity, and L. Maguire. Fault diagnosis of electronic systems using artificial intelligence. In *IEEE Instrumentation & Measurement Magazine*, volume 5, pages 16–20, 2002.

[39] C. Constantinescu. Inferring coverage probabilities by optimum 3-stage sampling. In *Reliability and Maintainability Symposium*, pages 22–25, 1996.

[40] D. T. Smith, B. W. Johnson, and J. A. Profeta. System dependability evaluation via a fault list generation algorithm. *IEEE Transactions on Computers*, volume 45, pages 974-979, 1996.

[41] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *IEEE Design Automation and Test in Europe Conference*, 2009.

[42] Introduction to lpsolve. http://sourceforge.net/projects/lpsolve.

[43] IEEE standard, Verilog Hardware Description Language. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00954909.

[44] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, Belmont, MA, 1997.

[45] H. Fang, K. Chakrabarty, A. Jas, S. Patil, and C. Trimurti. RT-level deviation-based grading of functional test sequences. In *IEEE VLSI Test Symposium*, pages 264–269, 2009.

[46] Z. Wang, K. Chakrabarty, and M. Goessel. Test set enrichment using a probabilistic fault model and the theory of output deviations. In *IEEE Design Automation and Test in Europe Conference*, pages 1275–1280, 2006.

[47] G. Casella and R.L. Berger. *Statistical Inference.* Duxbury Press, Boston, MA, 2001.

[48] USB 1.1 Functional IP core. http://opencores.com/project,usbhostslave.

[49] OR1200 Architecture. http://www.opencores.org/project,or1k.

[50] R. Guo, Y. Huang, and W. Cheng. Fault dictionary based scan chain failure diagnosis. In *IEEE Asian Test Symposium*, pages 45–52, 2007.

[51] Y. Huang, R. Guo, W. Cheng, and J. Li. Survery of scan chain diagnosis. In *IEEE Transactions on Design and Test of Computers*, volume 25, pages 240–248, 2008.

[52] S. Chun, Y. Kim, T. Kim, and S. Kang. An efficient scan chain diagnosis method using a new symbolic simulation. In *IEEE VLSI Test Symposium*, pages 73–78, 2008.

[53] L. Amati, C. Bolchini, L. Frigerio, F. Salice, B. Eklow, E. Brambilla, F. Franzoso, and M. Martin. An incremental approach to functional diagnosis. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI systems*, pages 392–400, 2009.

[54] F. Liu, P. K. Nikolov, and Sule Ozev. Parametric fault diagnosis for analog circuits using a Bayesian framework. In *IEEE VLSI Test Symposium*, pages 1–6, 2006.

[55] B. Ye, Z. Luo, W. Zhang, and C. Piao. Fault diagnosis for power circuits based on SVM within the Bayesian framework. In *World Congress on Intelliggent Control and Automation*, pages 5125–5129, 2008.

[56] Z. Zhang, Z. Wang, X. Gu, and K. Chakrabarty. Physical defect modeling for fault insertion in system reliability test. In *IEEE International Test Conference*, 2009, Paper 12.4.

[57] O. Caty, P. Dahlgren, and I. Bayraktaroglu. Microprocessor silicon debug based on filure propagation tracing. In *IEEE International Test Conference*, 2005, Paper 12.2.

[58] I. Pomeranz and S.M. Reddy. On dictionary-based fault location in digital logic circuits. *IEEE Trans. Computers*, 1997, vol. 46, pp. 48-59.

[59] G. Wedge. Using boundary scan with a fault dictionary to test and diagnose clusters of non-scan logic. In *AUTOTESTCON*, pages 400–404, 1996.

[60] D.A. Adjeroh, M.C. Lee, and I. King. A distance measure for video sequence similarity matching. In *International workshop on Multi-Media Database Managemet Systems*, pages 72–79, 1998.

[61] N. Yazdani and Z.M. Ozsoyoglu. Sequence matching of images. In *International Conference on Scientific and Statistical Database Systems*, pages 53–62, 1996.

[62] M. Lu and H. Lin. Parallel algorithms for the longest common subsequence problem. In *IEEE Trans. on Parallel and Distributed Systems*, volume 5, pages 835–848, 1994.

[63] Introduction to Cisco Catalyst 3750-E Series Switches. http://www.cisco.com/en/US/products/ps7077/.

[64] E. Liau and D. Schmitt-Landsiedel. Computational intelligence based testing for semiconductor measurement systems. In *IEEE International Test Conference*, pages 906–915, 2005.

[65] S. Ellouz, P. Gamand, C. Kelma, B. Vandewiele, and B. Allard. Combining internal probing with artificial neural networks for optimal RFIC testing. In *IEEE International Test Conference*, pages 1–9, 2006.

[66] A. A. Al-Jumah and T. Arslan. Artificial neural network based multiple fault diagnosis in digital circuits. In *IEEE International Symposium on Circuits and Systems*, volume 2, pages 304–307, 1998.

[67] K. A. E. Totton and P. R. Limb. Experience in using neural networks for electronic diagnosis. In *International Conference of Artifical Neural Networks*, pages 115–118, 1991.

[68] H. Ishibuchi and M. Nii. Generating fuzzy if-then rules from trained neural networks: linguistic analysis of neural networks. In *IEEE International Conference on Neural Networks*, volume 2, pages 1133–1138, 1996.

[69] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[70] S. F. Zornetzer, J. L. Davis, C. Lau, and T. McKenna. *An Introduction to Neural and Electronic Networks.* Academic Press, San Diego, CA, 1990.

[71] Neural network toolbox. http://www.mathworks.com/products/neuralnet/.

[72] S. Haykin. *Neural Networks and Learning Machines.* Prentice Hall, New Jersey, 2008.

[73] Y. Li, Y. Fu, H. Li, and S. Zhang. The improved training algorithm of back propagation neural network with self-adaptive learning rate. In *IEEE International Conference on Computational Intelligence and Natural Computing*, volume 1, pages 73–76, 2009.

[74] S.Y. Hsu, T. Masters, M. Olson, M. Tenorio, and T. Grogan. Comparative analysis of five neural network models. *Remote Sensing Reviews*, 1992, vol. 6, pp. 319-329.

[75] O. Chapelle, P. Haffner, and V. Vapnik. Support vector machines for histogram-based image classification. In *IEEE Transactions on Neural Networks*, volume 10, pages 1055–1064, 1999.

[76] G. Gao, Y. Zhu, G. Duan, and Y. Zhang. Intelligent fault identification based on wavelet packet energy analysis and svm. In *International conference on Control, Automation, Robotics and Vision*, pages 1–5, 2006.

[77] J. Weston and C. Watkins. Multiclass support vector machines. In *Univ. London, U.K., Tech. Rep. CSD-TR-98-04*, 1998.

[78] C. Hsu and C. Lin. A comparison of methods for multiclass support vector machines. In *IEEE Transactions on Neural Networks*, volume 13, pages 415–425, 2002.

[79] K. Thadani, A. Ashutosh, V.K. Jayaraman, and V. Sundararajan. Evolutionary selection of kernels in support vector machines. In *International Conference on Advanced Computing and Communications*, pages 19–24, 2006.

[80] C. Yang, W. Lee, and S. Lee. Learning of kernel functions in support vector machines. In *International Joint Conference on Neural Networks*, pages 1150–1155, 2006.

[81] A. Rakotomamonjy and S. Canu. *SVM and Kernel Methods Matlab Toolbox.* http://asi.insa-rouen.fr/enseignants/ arakotom/toolbox/index.html, 2008.

[82] G.C. Cawley. Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs. In *International Joint Conference on Neural Networks*, pages 1661–1668, 2006.

[83] Y. Zhao and K.C. Keong. Fast leave-one-out evaluation and improvement on inference for LS-SVMs. In *International Conference on Pattern Recognition*, volume 3, pages 494–497, 2004.

[84] S. Chen, C.J. Harris, and X. Hong. Construction of RBF classifiers with tunable units using orthogonal forward selection based on leave-one-out misclassification rate. In *International Joint Conference on Neural Networks*, pages 3358–3362, 2006.

[85] T.V. Gestel, J.A.K. Suykens, G. Lanckriet, A. Lambrechts, B.D. Moor, and J. Vandewalle. A Bayesian framework for least squares support vector machine classifers, Gaussian processes and kernel Fisher discriminant analysis. In *Neural Computation*, volume 14, pages 1115–1148, 2002.

Biography

Zhaobo Zhang

zz18@ee.duke.edu

PERSONAL DATA

Date of birth: October 25, 1986.
Place of birth: Shenyang, Liaoning, China.

EDUCATION

Doctor of Philosophy, Duke University, NC, USA, expected 2011.
Master of Science, Duke University, NC, USA, 2009.
Bachelor of Science, Tsinghua University, Beijing, China, 2007.

PUBLICATIONS

- **Conference Papers**

*1. Z. Zhang, X. Kavousianos, Y. Tsiatouhas and K. Chakrabarty, "A BIST solution for testing and repair of multi-mode power switches", accepted for publication in *Proc. IEEE International On-line Test Symposium*, 2011.

2. Z. Zhang, K. Chakrabarty, Z. Wang, Z. Wang and X. Gu, "Smart Diagnosis: Efficient Board-level Diagnosis and Repair using Artificial Neural Networks", *Proc. IEEE International Test Conference*, Paper 4.1, 2011.

*3. Z. Zhang, X. Kavousianos, Y. Luo, Y. Tsiatouhas and K. Chakrabarty, "Signature Analysis for testing, diagnosis, and repair of multi-mode power switches", *Proc. IEEE European Test Symposium*, pp. 13-18, 2011.

*4. Z. Zhang, X. Kavousianos, K. Chakrabarty and Y. Tsiatouhas, "A Robust and Reconfigurable Multi-Mode Power Gating Architecture", *Proc. International Conference on VLSI Design*, pp.280-285, 2011.

5. Z. Zhang, Z. Wang, X. Gu and K. Chakrabarty, "Board-level Fault Diagnosis using an Error-Flow Dictionary", *Proc. IEEE International Test Conference*, Paper 16.3, 2010.

6. Z. Zhang, Z. Wang, X. Gu and K. Chakrabarty, "Optimization and Selection of Diagnosis-Oriented Fault-Insertion Points for System Test", *Proc. IEEE Asian Test Symposium*, pp. 429-432, 2010.

7. Z. Zhang, Z. Wang, X. Gu and K. Chakrabarty, "Board-Level Fault Diagnosis using Bayesian Inference", *Proc. IEEE VLSI Test Symposium*, pp. 244-249, 2010.

8. Z. Zhang, Z. Wang, X. Gu and K. Chakrabarty, “Physical Defect Modeling for Fault Insertion in System Reliability Test”, *Proc. IEEE International Test Conference*, Paper 12.4, 2009

*9. Z. Zhang and S. Ozev, “Parametric Fault Diagnosis for Analog Circuit Based on Neural Networks”, *Proc. IEEE North Atlantic Test Workshop*, pp. 1-6, Boston, MA, May, 2008.

- **Journal Paper**

  1. Z. Zhang, Z. Wang, X. Gu and K. Chakrabarty, “Physical-defect Modeling and Optimization for Fault-insertion Test”, accepted for publication in *IEEE Transactions on VLSI Systems*, 2011.

- **Submitted Paper**

  1. Z. Zhang, X. Gu and K. Chakrabarty, “Support Vector Machines based Diagnostic System for Board-level Functional Diagnosis”, submitted to *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011.

PATENTS

- Filed U.S. patent, Z. Wang, X. Gu, Z. Wang, Z. Zhang, H. Zong, “Adaptive Manufacturing Line Diagnosis”, 2011.
- Filed U.S. patent, K. Chakrabarty, X. Kavousianos, Z. Zhang, “Power Switch Design and Method for Reducing Leakage Power In Low-Power Integrated Circuits”, 2010.

PROFESSIONAL ACTIVITIES

- IEEE student member
- Served as a reviewer for the IEEE Transactions on VLSI Systems, IEEE International Test Conference and IEEE VLSI Test Symposium.

WORK EXPERIENCE

- Intern, Network Division, Huawei Technologies Co. Ltd., Santa Clara, CA, Mar. 2011 - Present.
- Intern, Embedded Test, Cisco Systems, San Jose, CA, Feb. 2009 - Aug. 2009 and Feb. 2010 - June 2010.
- Intern, IBM T. J. Watson Research Center, Yorktown Heights, NY, May 2008 - Aug. 2008.

---

*Not related to Ph.D. thesis work.