ON REUSING FUNCTIONAL TESTS IN MANUFACTURING TESTING

by

Jian Kang

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Sharad C. Seth

Lincoln, Nebraska

December, 2007

UMI Number: 3284262

UMI Microform 3284262

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ON REUSING FUNCTIONAL TESTS IN MANUFACTURING TESTING

Jian Kang, Ph.D.

University of Nebraska, 2007

Adviser: Sharad C. Seth

Functional tests are developed during design verification to ensure the correctness of design. They can be reused in manufacturing testing to alleviate the cost of alternative testing methods and increase product quality. For large industrial circuits, there can be thousands of functional tests, each ranging from thousands to millions of cycles. The reuse thus poses many computational challenges. This dissertation addresses these challenges and the associated diagnosis problem to locate the root cause of failure.

Because of the large number of functional tests, it is impractical to apply all of them to test every fabricated device. The tests that provide good manufacturing defect screening value should be selected. Existing metrics for test selection either suffer from high computational cost, or lead to poor selection quality. An efficient register transfer level coverage metric is proposed which has high selectivity and low cost. As a high level metric, it also allows evaluation of testability issues earlier in the design cycle.

The testability of the selected tests can be further improved by additional observation points. Existing techniques require non-fault-dropping fault simulation which is very expensive. An efficient fault dropping technique is proposed based on the number of locations to which a fault gets propagated. Experimental results demonstrate the effectiveness of the method in achieving close to optimal results in the size of the selected subset with an order of magnitude less time.

The use of functional tests helps screen out defective devices. Failure analysis may be further carried out on defective devices to find the cause of the failure. Fault diagnosis helps failure analysis by reducing the number of candidate locations. Existing diagnosis techniques based on a fault model give erroneous results when a real defect is not accurately modeled. A symbolic approach is proposed which models the fault propagation condition at every gate as a Boolean equation. The solutions of the equations represent the set of all single and multiple causes of the observed behavior, thus give exact diagnosis results.

ACKNOWLDGEMENTS

Most importantly, I am indebted to my dissertation advisor, Professor Sharad C. Seth, for his patience and guidance to me through the years, making my Ph.D. study a precious experience.

Also thanks to Professor Jitender Deogun, Professor Sebastian Elbaum and Professor Sina Balkir for their helpful suggestions for my dissertation.

The staff in the department of Computer Science, Deb Heckens, Shelley Everett, Sally Hawkins, Charles Daniel, Scott Chaffin, deserve thanks for providing numerous assistance.

I would like to thank Vijay Gangaram at Intel for supporting this research, and Yi-Shing Chang, also from Intel, for helpful discussions.

Finally, I devote this dissertation to for my wife, my daughter, and my parents. It is their understanding, support and endless love that make all this happen.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

Design verification and manufacturing testing are two separate stages in testing of very large scale integration (VLSI) circuits to ensure the fabricated device meets the specification. Although functional tests are developed during design verification to ensure that the design is free of design errors, they can be reused during fabrication to help detect manufacturing defects. The reuse, however, poses many computational challenges, particularly for large industrial circuits whose functional tests are often quite large. This dissertation addresses these challenges and the associated diagnosis problem to locate the root cause of failure.

One problem with the reuse of the functional tests for a durable family of microprocessors is the sheer number of functional tests accumulated over the generations of design. It is impractical to apply all of them on the test equipment to test every manufactured device. Instead, it would be desirable to select a small number of those tests whose defect coverage approaches that of the whole set. A prior solution resorts to the less expensive toggle metric, which evaluates the value transition (o to 1 and 1 to 0) of a signal, to save computational effort at the cost of huge coverage loss. To solve this problem, a register transfer level (RTL) coverage metric, termed TRIO (Input/Output TRansition), is proposed, which achieves a better correlation with defect coverage compared to the toggle at low computational cost. Further, as an RTL metric, it can be evaluated at RTL, which further speeds up the computation and helps in evaluating the testability of a design at an earlier stage.

The testability of the selected test can be further improved by assuming that the observability of a circuit's response is not limited to its primary outputs. Modern manufacturing technology allows monitoring of internal nodes by insertion of observation points in the circuit. The use of observation points however entails performance costs hence the number of observation points must be minimized without sacrificing the testability achievable by observing all internal nodes. For industrial designs, the exact methods for observation point selection proposed in the literature, which requires fault simulation in which faults are never dropped, are found to be more than an order of magnitude more complex than the alternative fault simulation which drops a fault after first detection, even for the smaller number of selected tests. Therefore, an efficient fault dropping technique is proposed which is able to achieve close-to-optimal results in the number of selected points, at a computational cost that is an order of magnitude smaller than the exact method.

When a device fails the manufacturing test, understanding of the cause of the failure helps improve the manufacturing process. While it is possible to subject a device to failure-mode analysis (FMA), this hardware technique would be prohibitively expensive if applied to every failed device. Instead, a less expensive software-based fault-diagnosis method is used first to reduce the candidates for FMA to a small number. A symbolic path sensitization method for fault diagnosis is proposed, with a potential to be applied to other testing problems.

1.1 Background

1.1.1 Overall VLSI Testing Process

Since the development of integrated circuits (IC) in 1958, the scale of IC has doubled about every 18 months, following the famous Moore's law [1], reaching device densities of hundreds of million transistors today. Production of such high-density devices is a complicated task that can be described, at a high-level, as a two-stage process (Figure 1.1): design, where the specification is implemented through multiple design levels to a physical layout, and manufacturing, where the physical layout guides the fabrication steps of the device.

At each stage, testing is needed to ensure the design meets the specification. During design, it is necessary to verify that the design is free from design errors. During manufacturing, testing is used to guarantee the device is free from fabrication defects. In this dissertation we focus on the issues involved in reusing design-verification tests as manufacturing tests.

1.1.2 Design Verification

The design of a VLSI device usually begins with the description of the behavior of the system in hardware description language (HDL) such as Verilog or VHDL. With these

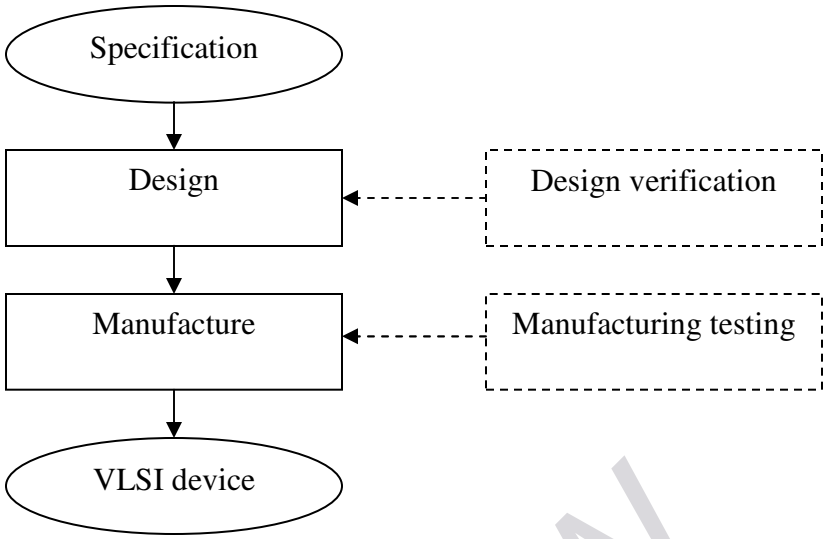

Figure 1.1 Steps in the realization of VLSI devices

languages, design can be described at a high level of abstraction that resembles a piece of software. Electronic design automation (EDA) tools assist the manual effort in progressively translating the high-level design to lower levels, until it is transformed to a physical layout, ready for fabrication. Errors could be introduced in the design description or the process of translation. It is important to correct an error as early in the design cycle as possible, because fixing it at lower levels of design is costlier.

To exercise the design and check for potential errors, both formal techniques [2] and simulation-based methods [3] have been explored. Formal techniques mathematically prove the correctness of design for all combinations of inputs. They are of limited use because they do not scale up well and require too much human intervention for large designs [4]. Simulation-based methods apply tests to exercise the design and compare the

responses with the expected responses. Although they can only find design errors and not prove their absence, they scale up well with design size and hence are widely used.

To help exercise the design and build confidence that the design matches specification, both automated test generation techniques [5, 6] and extensive manual efforts are used. As the tests are intended to verify the functionality of the design with respect to its specification, they are often called *functional tests*.

A well-defined metric is the key to evaluating the goodness of available tests and guiding the effort of generating additional tests. As mentioned, designs described in HDL resemble software programs, therefore code coverage metrics from software testing have been adopted in design verification [4]. Code coverage metric checks for the percentage of a certain code construct that is exercised during the simulation. For example, *line coverage* (or *statement coverage*) checks for execution of lines of code. Other code-based metrics include condition *coverage, path coverage*, etc. These metrics are helpful in analyzing the code that is not exercised; they ignore the observability because full-observability is available during the simulated environment of design verification. However, they do not consider whether the differences caused by a potential error in the exercised code could get observed. Observability-Based Code Coverage Metrics (OCCOM [7]) augments line coverage by propagating the difference caused by some possible error at assignment statement to some monitored point, so the difference could get observed. It is shown that OCCOM better reflects the quality of tests than statement coverage [7].

1.1.3 Manufacturing Testing

Manufacturing of the VLSI devices involves a series of photolithographic steps that are prone to error. Millions of transistors and interconnections are potential victims, and it may require only one of them to go wrong to fail the whole chip. Any small piece of dust or abnormality of geometric shape can result in a defect, so can process variations that affect transistor channel length, threshold voltage, etc. It is not abnormal that a mature process produces only about eighty percent good devices, and zero percent for a newly introduced process [8].

Manufacturing testing is needed to rule out the bad devices before shipping to the customer. At manufacturing stage, it is assumed that the design is free of design errors, so the purpose is to find the defects introduced during manufacturing. Manufacturing testing is challenging, especially under cost and other constraints. First, after a device is manufactured, the application of the stimuli and observation of the response can only happen at primary input and output pins that do not grow in number commensurate with the design size. This problem is especially severe for new design styles like system-on-a-chip (SOC), where a whole board may be integrated into a single chip, hiding many pins which were accessible. Second, new processes often result in an increase in the device scale. With newer processes, the fabricated device is more sensitive to subtle physical effects. Hence, new failure mechanisms like crosstalk [9] and electro-migration [10] must be considered. Third, although the design description has moved to higher levels to enable productivity with the growing design complexity, the development of manufacturing testis is still carried out largely at the gate level or the transistor level. The

computational effort at these lower levels can be very high. For all these reasons, manufacturing testing cannot be perfect for practical designs. Some bad devices will escape the test and get shipped to the customer. The percentage of bad devices in all shipped devices is a measure of the quality of testing, which is called test escape.

As it is not possible to access internal signals in a fabricated device, the use of formal methods such as equivalence checking [11] is limited, and simulation is the major technique in manufacturing testing. Tests are applied using some test equipment to the primary inputs of the device, and the responses of the device are collected and compared with the expected responses. Other type of testing exists. For example, embedding certain logic in the design for the purpose of testing of other parts, a technique called built-in-self-test. These are not the focus of this dissertation.

Fault Modeling

Two testing methods may be used to test a fabricated device. In the first approach, the device under test is regarded as a black box, and tests are applied to verify the functionality of the design without caring about its implementation details. This requires high cost test equipment to deliver the tests and collect responses at the specified functional speed. Alternatively, a white box test approach can be used. Since the structure (gates and interconnections) of the design is already known at the manufacturing stage, tests can be generated for the structure to make sure it is defect-free. This type of testing is called *structural testing* [12]. Some types of defects targeted by structural testing only cause functional problems instead of speed issues, thus less expensive test equipment can be used.

With the knowledge of the design structure, fault models can be defined. A fault model describes the logical behavior of the defect. A defect, such as a small particle falling on the wafer, is a physical reality that is useful to learn, however, mathematically analyzing a physical defect is often difficult. By modeling the logical behavior of the defect, a fault model allows mathematical analysis of the defect. For the example defect of a small particle, it could have caused a short between two wires, which is modeled as a short (also called bridging) fault.

With the introduction of a fault model, structural testing has a clear goal. Based on the fault model we could derive a list of faults that must be covered. Then the goodness of a given test set can be evaluated by the percentage of faults that is covered (called *fault coverage*), and if the coverage is below an acceptable threshold, un-detected faults can be identified and targeted for generating additional tests.

A good fault model should accurately reflect the behavior of the modeled defects and it should be computationally efficient. These criteria were found to be met by the single stuck-at fault model (abbreviated as SSF) [13] which is widely used in testing. The SSF model can be defined by the following rules:

1) The fault is defined on lines, where a line can be either a primary input, primary output, or a gate input or output
2) Only one faulty line exists in the circuit
3) The faulty line's output appears stuck at logic 0 or logic 1 irrespective of its input

According to the model, each line in the circuit could have two stuck-at faults, stuck-at-0 and stuck-at-1. For a netlist with N lines, the total number of SSFs is 2N. This number can be reduced by grouping faults that are functionally equivalent. The SSF model has a relatively small fault size compared with some other fault models, and it has good correlation with real defects. It has been in existence for over fifty years, and continues to be important today. However, the defect types have evolved with the progress of technology some of which cannot be modeled with SSF, therefore additional fault models are employed. For example, to capture defects that produce correct functionality but extra delay, *transition fault* is defined, which assumes that there is a large abnormal delay associated with some gate, so that any path that passes through this gate will have delay problem. Other popular fault models include bridging, path delay, transistor stuck open, transistor stuck short. A transition fault shares similarity with an associated stuck-at fault and can be supported with incremental effort on existing SSF tools. Other models (e.g. path delay fault) are quite different from SSF and have not gained wide acceptance. In the following discussion we assume the SSF model unless explicitly mentioned otherwise.

Fault Simulation

The occurrence of a *detectable fault* changes the functionality at the primary output of the faulty circuit. A detectable fault is a fault that can be detected by some test. We call the fault-free circuit as *good circuit*, and circuit with an injected fault a *bad circuit*. If the good and bad circuits produce different responses for a test, then we say that the injected fault is detected. Simulating the responses of the good circuit and all the bad circuits for a given test set is called *fault simulation*.

In general, Given a set of tests T = {T_1, T_2, …, T_n} and a fault list F = {f_1, f_2, …, f_m}, whose corresponding bad circuits are {Bad_1, Bad_2, …, Bad_m}, fault simulation requires calculating the output response $M_k(i)$ for each circuit k and test i (Table 1-1). By comparing the output responses of the good circuit against each bad circuit, we can determine the fault coverage, and the set of undetected faults in the fault list.

Table 1-1 The computation task for fault simulation

	TEST SEQUENCE					
	T_1	…	T_i	T_{i+1}	…	T_n
Good	$M_0(1)$	…	$M_0(i)$	$M_0(i+1)$	…	$M_0(n)$
Bad_1	$M_1(1)$	…	$M_1(i)$	$M_1(i+1)$	…	$M_1(n)$
…	…	…	…	…	…	…
Bad_k	$M_k(1)$	…	$M_k(i)$	$M_k(i+1)$	…	$M_k(n)$
Bad_{k+1}	$M_{k+1}(1)$	…	$M_{k+1}(i)$	$M_{k+1}(i+1)$	…	$M_{k+1}(n)$
…	…	…	…	…	…	…
Bad_m	$M_m(1)$	…	$M_m(i)$	$M_m(i+1)$	…	$M_m(n)$

The complexity of logic simulation a circuit with G gates is O(G), since in the worst case every gate needs to be evaluated for the new vector once. For single stuck-at faults, since the faults are associated with the inputs and outputs of gates, the number of faults is proportional to the number of gates. The straight-forward way of computing the information in Table 1-1 involves carrying out logic simulation for each (good and bad) circuit, thus the complexity of fault simulation for one test vector is $O(G^2)$. Fault

simulation introduces high computational cost when the design and the tests becomes large, therefore many techniques have been proposed to improve it [12] as summarized below.

First, not all the information in the table has to be computed. If the goal is to determine the set of covered faults, once a fault is detected by some test, it does not need to be simulated further. This technique to speed up fault simulation is called *fault dropping* [12]. Fault dropping however cannot be used in applications that require more information, for example, the observation point selection problem in Chapter 3. When all of the information in Table 1-1 is computed, i.e. no fault is ever dropped, we refer to it as *non-fault-dropping fault simulation*.

Second, there are more efficient ways than simulating the logic of each circuit individually. For example, when the circuits are modeled using logic gates (AND, OR), and simulation of a circuit could use bit-level logic operation, multiple circuits could be simulated simultaneously, taking advantage of the fact that a computer word has multiple bits. This technique is called *parallel fault simulation* [14]. Also, since a fault often causes only a small change in the circuit state, after simulating the good circuit, only the component states that deviate from the good circuit need to be simulated in the technique called *concurrent fault simulation* [15]. Although fault simulation is still much more expensive than logic simulation, these and other techniques have greatly increased its speed.

In our experiment (in Chapter 3), for a sequential industrial circuit with 370K gates and 676K SSF faults, logic simulation of a test with 7K vectors takes 102 seconds, and for a random 5% of the faults, fault-dropping fault simulation takes 2K seconds, and non-fault-dropping fault simulation takes 114K seconds.

Test Generation

When an undetected fault is identified, test generation produces a test to detect this fault. The process of automatically generating a test to detect a fault is called *automatic test pattern generation* (ATPG).

To detect a fault, the generated test needs to give different responses for the good circuit and the bad circuit. This involves excitation of the fault at the faulty location, i.e. producing a difference between the good circuit and the bad circuit, and propagation of the difference to some primary output so the faulty effect could be observed.

For combinational circuits, detection of a single stuck-at only requires one test vector. For example, to detect the stuck-at-0 fault (abbreviated as sa0) on line D in the circuit shown in Figure 1.2, test vector ABC=110 could excite the fault by creating, at line D, logic 1 in the good circuit, and logic 0 in the bad circuit (which is marked as 1/0 in the figure). Also, the difference could propagate to the primary output E and get observed.

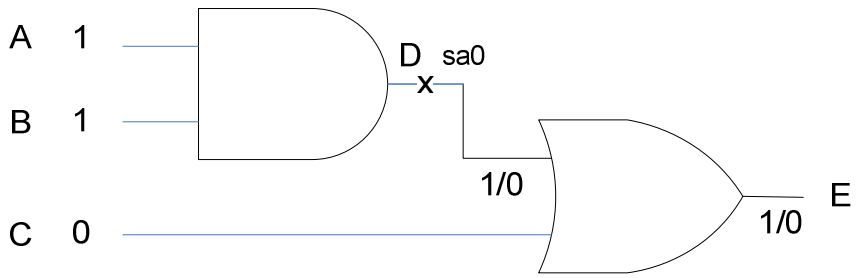

Figure 1.2 Testing the stuck-at-0 fault for the input of an AND gate

In general, for any combinational circuit, the task of ATPG for SSF is then to find an input vector that could detect the given fault. If the combinational circuit has n primary inputs, then the search space would be 2^n input combinations. Test generation for combinational circuits is an NP-complete problem, and it has been a core topic in structural testing [12].

For sequential circuits, a test often requires a sequence of test vectors, and the complexity of test generation is even higher. The flip-flops in a sequential circuit become pseudo primary inputs and pseudo primary outputs for the combinational block, and these pseudo ports cannot be directly controlled or observed. A vector found to test a fault in the combinational block may involve both primary inputs and pseudo primary inputs. While the values on primary inputs can be directly applied, values on pseudo primary inputs are not directly applicable and require justifications in the earlier cycles. Similarly, the difference caused by the fault may have propagated to pseudo primary output, which cannot be directly observed, and require further propagation in later cycles until it reaches some primary output. The existence of pseudo primary inputs and pseudo primary outputs greatly increases the complexity of ATPG. In addition to finding a suitable input