Linköping Studies in Science and Technology

Thesis No 725

High-Level Testability Analysis and Enhancement Techniques

by

Erik Larsson

Submitted to the School of Engineering at Linköping University in partial fulfillment of the requirements for the degree of Licentiate of Engineering

Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Linköping 1998

High-Level Testability Analysis and Enhancement Techniques

by

Erik Larsson

September 1998
ISBN 91-7219-305-0
Linköping Studies in Science and Technology
ISSN 0280-7971
Thesis No 725
LiU-Tek-Lic 1998:60

ABSTRACT

Hardware testing is an important issue in the development of digital systems and much research has been devoted to it. Most of this work deals with design for testability at low abstraction levels. In this thesis we deal with the test problems at the high abstraction levels. By considering the testability at an early design stage the test problems can be efficiently solved. We propose behavioral-level testability metrics based on an analysis of the variable range, operation testability and statement reachability. The metrics can be used to detect the hard-to-test parts with a low computational cost. By experiments we show the correlation between our metrics and the fault coverage when partial scan is used. We also show by experiments, where our technique is compared with a commercial gate-level tool, that it is possible to predict the testability on the behavioral-level accurately and efficiently. We have then developed a set of testability improvement transformations applicable direct on the behavioral VHDL specification with no restrictions on the high-level synthesis and a transformation selection strategy based on our testability metrics. We have also developed a controller testability analysis technique which is based on statement reachability and an enhancement technique consisting of branch control, loop termination and register initialization methods. The controller testability analysis and enhancement technique can be used for a register-transfer level design to improve its testability when the design is finally implemented, which is demonstrated by experiments with several benchmarks.

Supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

Department of Computer and Information Science
Linköping University
S-581 83 Linköping
Sweden

High-Level Testability Analysis and Enhancement Techniques

Erik Larsson

ISBN 91-7219-305-0 ISSN 0280-7971
PRINTED IN LINKÖPING, SWEDEN
BY LINKÖPING UNIVERSITY

To Valentina

Abstract

HARDWARE TESTING is an important issue in the development of digital systems and much research has been devoted to it. Most of this work deals with design for testability at low abstraction levels. In this thesis we deal with the test problems at the high abstraction levels. By considering the testability at an early design stage the test problems can be efficiently solved. We propose behavioral-level testability metrics based on an analysis of the variable range, operation testability and statement reachability. The metrics can be used to detect the hard-to-test parts with a low computational cost. By experiments we show the correlation between our metrics and the fault coverage when partial scan is used. We also show by experiments, where our technique is compared with a commercial gate-level tool, that it is possible to predict the testability on the behavioral-level accurately and efficiently. We have then developed a set of testability improvement transformations applicable direct on the behavioral VHDL specification with no restrictions on the high-level synthesis and a transformation selection strategy based on our testability metrics. We have also developed a controller testability analysis technique which is based on statement reachability and an enhancement technique consisting of branch control, loop termination and register initialization methods. The controller testability analysis and enhancement technique can be used for a register-transfer level design to improve its testability when the design is finally implemented, which is demonstrated by experiments with several benchmarks.

Preface

The work presented in this thesis was conducted at the Embedded System Laboratory (ESLAB), Department of Computer and Information Science, Linköping University. It has been a part of the Design For Test (DFT) sub-project which is included in the Advanced Design with ASIC consortium, funded by NUTEK[1]. The main objective for our work within the DFT sub-project has been to develop design-for-testability methodologies and tools to facilitate testing of complex digital systems implemented in VLSI technology.

Our research has mainly been focused on:

- the development of design aid tools to help the designer to perform testability analysis; and
- the development of testability improvement transformations and their selection based on the testability analysis results.

New design tools have been developed to handle the increasing complexity of digital design. However, test tools have not kept pace. In order to ease test problems it is important to consider testing as early as possible in the design process. In our work we have focused on test problems at the register-transfer

1. Swedish National Board for Industrial and Technical Development.

and behavioral levels. An advantage of working on high abstraction levels is that functional properties of the design can explicitly be utilized. Another advantage is that the complexity of a design captured at these levels is reduced. Information regarding testability of an electronic system at the early design stages is also very useful for designers to make appropriate decisions.

Our research has been carried out in close cooperation with the industry, especially the CadLab research center (Gunnar Carlsson) at Ericsson telecom. This cooperation has provided us with a lot of insight into the test problems and a platform to demonstrate the usefulness of our developed techniques.

Acknowledgements

I would like to express my deepest gratitude to a number of people who have helped me towards this thesis. First I would like to thank Professor Zebo Peng who gave me excellent supervision and guidance. I would also like to thank Dr Xinli Gu for the early cooperation presented in Chapter 6, which gave me a good starting point for my own research work.

Colleagues at IDA have given a nice atmosphere to work in. I would especially like to thank former and present members at ESLAB and CADLAB for creating such creative and enjoyable environment not only at work. I would also like to thank the members at the Electronics Systems group at ISY, and especially Mats Larsson, for the help with the Mentor Graphics tool set.

My parents, Knut and Eva, and my brothers, Magnus and Bengt, have always been the greatest support.

Finally, I would like to thank Valentina, my brown eyed girl, for bringing me happiness.

Erik Larsson

Contents

Chapter 1
Introduction

This thesis deals with the problems of hardware testing and we focus on problems at the early stage in the design process. Most previous work has mainly considered test problems at lower abstraction levels. Recently, the increase of complexity of digital designs has led to the development of new modelling techniques at higher and higher abstraction levels. Design tools operating at the high abstraction levels have been developed, but test tools have not kept pace and testing remain a major problem. To reduce this problem, it is important to predict and improve testability as early as possible in the design process.

The objective of hardware testing is to ensure fault-free electronic products and it is carried out after production. The aim is to detect physical faults introduced during or after production, which should be distinguished from hardware verification where the aim is to detect design errors.

In this thesis we define a technique to analyze testability and a transformation technique to improve it for a behavioral VHDL specification. We also propose a technique to analyze the testability for a controller on register-transfer level and a technique to enhance its testability.

1.1 Motivation

Much work in modelling techniques and development of design tools has been performed at low abstraction levels. The increasing complexity of digital designs has introduced new modeling techniques and new design tools at higher abstraction levels. In order to reduce the test generation and application complexity, it is important to predict testability of a design at higher abstraction levels in order to ensure that the final design is testable. At higher abstraction levels the functional properties of the design can explicitly be captured and it can be used to speed up testability analysis. Such information is difficult to extract from a gate-level design.

Several design-for-testability (DFT) techniques have been proposed to improve the testability of a digital system. Most of the techniques are defined for lower abstraction levels. When introducing a DFT technique it may degrade the design. In order to make trade-offs between the testability and design degradation the design is analyzed. Several approaches have been presented for analyzing the testability. However, approaches on higher abstraction levels, register transfer level, usually only consider the data path or the controller. A major problem in hardware testing are the feed-back loops and they must be considered in testability analysis. Another problem is the number of design iterations. A common approach is to analyze the design and then improve one hard-to-test part. The approach is justified in that it keeps the introduced penalty at a minimum. However, for large designs the number of design iterations will be numerous and to reduce them it is important to select more then one hard-to-test part in each iteration.

1.2 Problem Formulation

The aim of our work is to reduce the testing cost, which is usually a large part of the production cost, when developing digital systems.

Our approach is to develop an efficient method to improve the test quality at an early design stage. By test quality we mean fault coverage, test generation time and test application time. The fault coverage is defined for the single stuck-at-fault model.

By efficiency, we mean low computational time, low area overhead and small performance degradation. Early in the design stage refers to the behavioral level and register-transfer level.

The objective is achieved by using a high-level testability enhancement technique. However, the introduction of testability improvement techniques may also degrade the design in terms of extra area and/or extra delay. The aim of a testability analysis is to find a trade-off between testability and design degradation.

1.3 Contributions

The contributions of this thesis are as follows:

- **A behavioral-level testability metrics**. We define a testability metrics that detects hard-to-test parts in a behavioral specification at a low computational cost. An advantage is that the designer gets an early feeling for the test problems and can use this information to improve the testability of the design. The metrics are a combination of variable range, operation testability and statement reachability. We show an application of the testability metrics for partial scan selection and we present an algorithm to calculate the metrics. We perform experiments to show the correlation between our test metrics and the fault coverage and we compare our behavioral-level analysis with a commercial gate-level tool where we show that the hard-to-test parts can be predicted at the behavioral level.

- **Behavioral-level testability transformations.** We propose a set of testability transformations, which include write-insertion, read-insertion, boolean-insertion and reach-insertion, and a transformation selection strategy. The transformations are applicable directly on the behavioral VHDL specification and they do not impose any restrictions on the high-level synthesis process. We propose a selection strategy and by experiments we show the efficiency of our approach. We also present a partitioning scheme based on dependency among variables. By partitioning the variables it is possible to improve the testability for several hard-to-test parts in each design iteration.
- **Controller testability analysis and enhancement techniques**. We propose a testability analysis technique for the controller and a testability enhancement technique for this. The controller has large impact on the testability and by considering it the test problems for the whole design will be reduced. The controller metrics are based on statement reachability and the enhancement technique is based on loop termination, branch control and register initialization. We show by experiments that our enhancement technique improves the testability.

1.4 Thesis Overview

This thesis is organized as follows. In Chapter 2 we provide a background to synthesis for testability and we introduce the basic definitions and testability techniques.

In Chapter 3 we present related work. We present an overview of previous approaches to analyzing the design as well as techniques to improve the testability.

We define our behavioral-level testability metrics in Chapter 4, where we also present an algorithm to calculate the metrics and we show an application of it for partial scan selection. The chapter concludes with experimental results where we show that our metrics detect hard-to-test parts and that we can predict testability on the behavioral level.

In Chapter 5 we propose a design transformation technique and a selection strategy that improves the testability of a behavioral specification. We present experimental results to show that our approach makes the design testable. The traditional design approach is to analyze the testability and then apply a design-for-testability technique on the hardest to test part. For large designs this approach leads to a large number of design iterations. We propose a variable partitioning scheme based on variable dependency.

In Chapter 6 we propose a technique to analyze the testability of the controller and a technique to improve the testability. The analysis is based on statement reachability and the enhancement technique consists of loop breaking, branch control and register initialization. Through experiments we show that our approach improves testability. We conclude the thesis in Chapter 7 where conclusions are made and future issues are addressed.

Chapter 2
Background

Synthesis for testability has a large impact on all stages in the design flow and much research has been devoted to it.

After the introduction in section 2.1, we discuss design representations at different abstraction levels in section 2.2. The high-level synthesis concept is introduced in section 2.3 and finally we discuss design for testability in section 2.4.

2.1 Introduction

The increasing complexity in digital design has enforced new modelling techniques at higher and higher abstraction levels. New techniques and tools have been developed for the transformation of a design representation from one abstraction level to another.

A typical design flow of today is illustrated in Figure 2.1 where three different abstraction levels can be distinguished, behavioral, structural and gate level. The design work starts with a design captured in a behavioral specification. It is transformed by the high-level synthesis process to a structural specification. The logic synthesis process transforms the structural

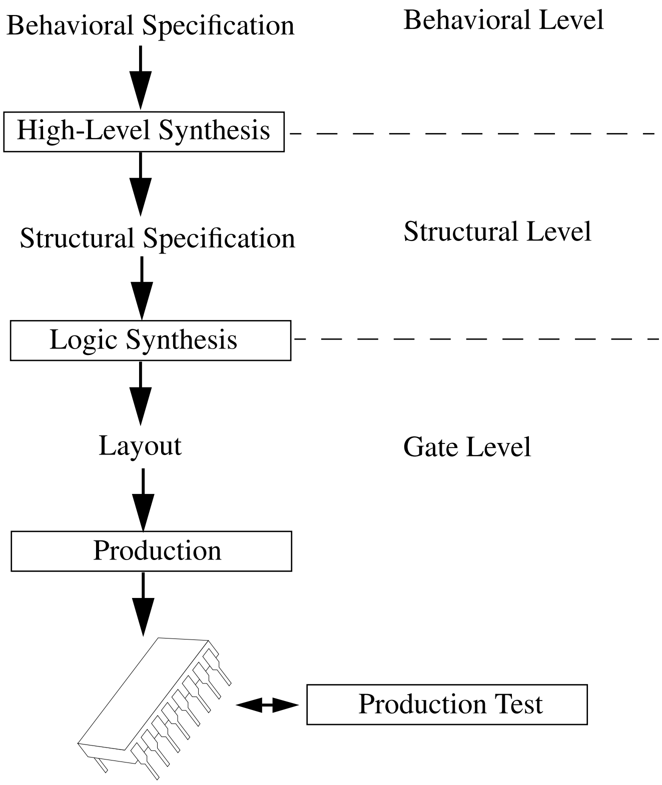


Figure 2.1: The design flow for digital design.

specification to a layout. The layout is then sent for production and after this we perform production tests to detect production errors. A test for all possible errors in the production may require a large test effort. In order to minimize the test effort and maximize the test coverage, we have to consider the test problems during the design process.

2.2 Design Representations

A design can be described at different abstraction levels. At higher abstraction levels fewer implementation specific-properties are found, while at lower abstraction levels more implementation-specific properties are added. Since a model at a high abstraction level contains less implementation-specific details, it is less complex and easier to grasp for a designer than a model at a lower level.

In this section we will cover behavioral, structural and intermediate specifications. Gate-level formats are not covered.

2.2.1 BEHAVIORAL SPECIFICATION

The design work starts with a behavioral specification. The term *behavioral specification* is used to reflect that the specification at this level only captures the behavior of the design. The required resources and implementation structure timing are not specified.

Below we present two behavioral specifications, which are used as inputs to high-level synthesis systems. First, the Synthesia system, which is a commercial available system, and then CAMAD, which is a research system developed at Linköping University.

SYNT is a high-level synthesis system developed at Synthesia [Syn96], which given a behavioral specification in VHDL performs high-level synthesis and generates a register transfer level design.

The behavioral VHDL specification is formed by a subset of VHDL. The main constraint of the Synthesis system is that only sequential statements are accepted in the behavioral specification. This means that only one process can be synthesized, since several processes form concurrent VHDL.

The Camad High-Level synthesis tool is a research tool developed at Linköping University [Pen94]. It accepts as input a behavioral specification specified in VHDL [Ele92] or ADDL, Algorithmic Design Description Language [Fje92], [Pen94]. The latter includes a subset of Pascal with an extension for parallel constructs, COBEGIN and COEND. It allows an arbitrary nesting of parallel processes. The current implementation accepts the following data types: integer, boolean, char, and sub-ranges and one-dimensional arrays of these. Neither dynamic data structure nor recursion is allowed.

2.2.2 STRUCTURAL SPECIFICATION

The structural specification, which is the output from the high-level synthesis process, contains more implementation specific properties than the behavioral specification. From a specification at this level it is possible to derive the number of components and at what time (clock period) a certain operation is performed.

A structural specification using VHDL typically includes component instantiations, the way that the components are connected with each other with signals and a finite state machine. It is usually used as input to a logic synthesis tool.

For example, the subset of VHDL accepted by Mentor Graphics' synthesis tool, Autologic, includes several processes, variables, signals, functions, component declaration, etc. [Me93a], [Me93b]. However, only one wait-statement is accepted for each process.

Another limitation is that the bounds for loops must be known, i.e. no variable loop-statements, which means that all loops can be unrolled.

2.2.3 INTERMEDIATE REPRESENTATION

In high-level synthesis, where a structural specification is generated from a behavioral specification, it is common to first transform the behavioral specification to an intermediate representation.

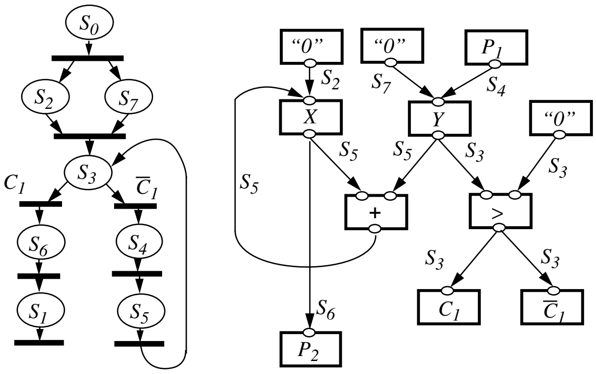
There exist several intermediate specifications, such as the control flow graph, data flow graph and control/data flow graph [Gaj92]. We will here briefly describe a representation called Extended Timed Petri Net, ETPN [Pen94]. The ETPN representation is based on a data flow part that captures the data path and a control flow part that decides the partial ordering of data path operations.

The control flow part is modelled by a Petri net notation and the data path is a directed graph where each vertex (node) has the possibility of multiple inputs and/or outputs, see Figure 2.2. The control flow is modelled as a Petri net, where the Petri net places (S-elements) are the circles while the transitions (T-elements) are the bars in Figure 2.2.

Initially a token is placed at S_0, which is an initial place, Figure 2.2. A transition is enabled if all its input places have at least one token and it may be fired when the transition is enabled and the guard condition is true. Firing an enabled transition removes a token from each of its input places and deposits a token in each of its output places. If no token exists in any of the places, the execution is terminated.

When a place holds a token, its associated arcs in the data path will open for data to flow. For instance when place S_2 holds a token, the edges controlled by S_2 in the data path activate and data is moved.

Some of the intermediate representations are close to behavioral specifications, while others are closer to structural specifications. For instance, data flow graphs and control data flow graphs can be placed in the former class, while representations given in ETPN belong to the latter. With the ETPN it is possible to predict the number of modules needed for the data path and the partial order of operations.



(a) Control part (b) Data path

Figure 2.2: An example of ETPN .

2.3 High-Level Synthesis

High-level synthesis is the transformation of a behavioral specification into a structural implementation [Gaj92]. It mainly consists of highly dependent but usually treated as separated tasks, namely scheduling, allocation and binding of operations to components to fulfill some given design constraint.

Scheduling is basically assigning an operation to a time slot, a control step, which corresponds to a certain clock cycle. If several operations are assigned to the same control step, several functional units are needed. This results in fewer control steps, which results in a faster design but it also leads to a more expensive design [Gaj92].

The allocation task is to select the number and the types of hardware units to be used in a design. Sharing of resources reduces the design size but is only allowed if the units are not used by different operations at the same time. Binding is when the operations are mapped to a certain allocated hardware unit.

High-level synthesis has traditionally been considered as an optimization of a two-dimensional design space defined by area and performance. However, recently the design space has been extended to include power consumption and testability, as well as other criteria.

A popular approach to high-level synthesis is the transformation-based approach which starts with a naive initial solution. The solution is improved by applying transformations until a solution that is close to the optimal solution and which fulfils the given constraints is found.

2.4 Design for Testability

The cost of testing includes costs related to issues such as test pattern generation, fault simulation, generation of fault location information, cost of test equipment and the cost of the test process itself, which is the time required to detect and/or isolate a fault.

The test cost can be reduced by using some design-for-testability (DFT) technique. However, a DFT technique may result in some performance degradation and/or some area overhead. The most important consideration when applying a DFT technique is the selection of places to apply the DFT technique and the trade-off between testability and the performance/area penalty.

The selection of hard-to-test parts includes a trade-off between accuracy in finding the hard-to-test parts and the computational complexity.

2.4.1 Basic Definitions

In this section we will present some basic definitions related to design for testability.

A produced VLSI chip may contain several types of physical defects, such as a broken or missing wire, a wire which is wrongly connected to another wire. Some of the defects are present directly after production, while others may occur after some operation time.

Logical faults are commonly used to model physical defects [Abr90]. The most commonly used fault model is the single stuck-fault (SSF) model, which assumes that the design only contains one fault. It also assumes that when a fault is present, it is either permanently connected to 1 (stuck at 1 fault) or permanently connected to 0 (stuck at 0 fault). A test detects a fault in a circuit if the output of the fault-free circuit is different from the output of the faulty one.

The main advantage of the SSF model is that it represents many different physical defects, and it is technology-independent. Experience has also shown that SSF detects many other faults. Further, using the SSF model the number of faults is low compared with other models [Abr90]. A design with n lines results in $2 * n$ faults.

The *fault coverage* or *test coverage* is used to indicate the quality of tests with a given fault model [Tsu88]. The fault coverage, f, is defined as:

$$f = \frac{n}{N}$$

where n is the number of faults detected by the given test set [Abr90]. N is the total number of faults defined by the given fault model.

2.4.2 TESTABILITY IMPROVEMENT TECHNIQUES

Several techniques are used to improve the testability of a digital circuit. In this section we will present the test point insertion approach, the scan technique, built-in self-test (BIST), and high-level test synthesis.

Test Points

Test point insertion is a simple and straightforward approach to increase the controllability and/or observability of a design. In Figure 2.3(a) a line (wire) between two components is shown.

The ability to set the value of the line (wire) to 0 is enhanced by adding a 0-controllability test point. That is, an extra primary input and an AND-gate are added, see Figure 2.3(b). The 1-controllability, the ability to set a line to 1, is enhanced by adding an extra primary input and an OR-gate, Figure 2.3(c). To increase the observability of a line in a design an extra primary output is added, Figure 2.3(d).

The main advantage of test point insertion is that the technique can be applied to any line in the design. However, the drawback is the large demand for extra primary inputs and outputs. The technique also require extra gates and extra lines which introduce delay.

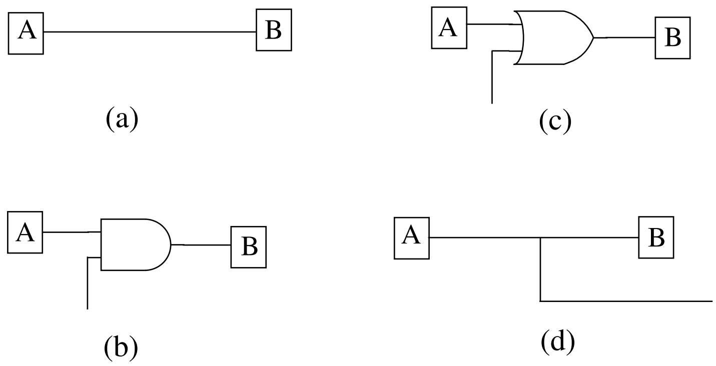


Figure 2.3: Test points for control (AND and OR) and observation enhancement.

Scan Technique

The main problem for test pattern generation is usually due to the sequential parts of the design. The scan technique is a widely used technique which turns a sequential design into a combinational one and makes it easier to generate test patterns. The scan technique enhances controllability and observability by only introducing two extra primary inputs (one for test data input and one for test enable), and one extra primary output used for test data output. In test mode the flip-flops in the design are connected to form a shift register. When the design is in test mode, data is shifted into the design by one of the extra inputs. The circuit then runs for one clock cycle and then data is shifted out on the added primary output.

The basic idea behind the scan technique is illustrated in Figure 2.4. Using the signal "scan selection" the register can be controlled in two modes, normal mode or test mode. In test mode the scan-in is active and the contents of the flip-flops are easily set. The value stored in the flip-flop is also easily observed on the scan-out line.

When all flip-flops are connected to form a scan chain it is called full scan. The overhead introduced by using the scan technique includes routing of new lines, more complex flip-flops than needed, and adding three I/O pins. The overall clock speed may have to be reduced due to additional logic in the flip flops scan path [Abr93]. The test application time may increase since a long scan chain requires many clock cycles to scan in the test vectors. This can be solved by a faster scan clock or by dividing the scan chain into several shorter chains. However, both approaches introduce penalties. The fast scan clock introduces extra area and the division of the scan chain leads to extra primary inputs and primary outputs.

The overhead introduced by using the full scan technique may be too high. Partial scan is a technique where only a subset of the flip-flops in the design are connected in the scan chain. This is done in order to get a trade-off between the testability of the circuit and the overhead introduced by scan design.

Built-In Self-Test

When the scan technique is used, the test vectors are applied from the outside of the unit under test by a tester. The Built-In Self-Test (BIST) technique does not require any external test

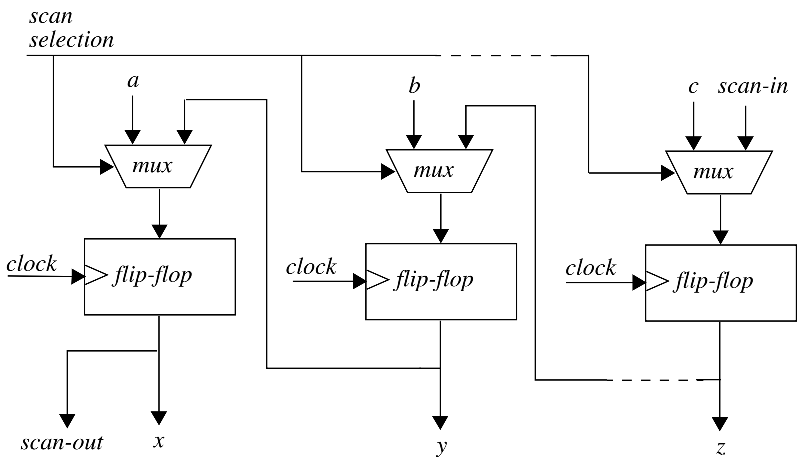


Figure 2.4: The basic idea for scan technique.

equipment. Instead the test pattern generator, response analyser and test controller are placed in the design.

The test pattern generator can be a linear feed-back shift register (LFSR) and the response analyser can be a multiple input signature register (MISR). A built-in logic block observer (BILBO) is a register which can operate both as test pattern generator and signature analyser. However, the disadvantage of using BILBOs is the large area and delay penalty [Wag96].

The BIST can be performed *on-line* or *off-line*. In *on-line* BIST, there is no special test mode. It means that the test occurs during normal mode. Furthermore, the test can occur *concurrently* or *nonconcurrently*. In *concurrent on-line* BIST testing is performed simultaneously with the normal functions, while in *nonconcurrent on-line* BIST the testing is performed when the system is in an idle state.

An advantage of using the BIST technique is that tests are performed at speed. The technique has also a lower test application time compared to the scan technique. For large designs with long scan chains, the scan technique requires much time to scan-in and scan-out test vectors.

Since the BIST technique does not require any special test equipment, it can be used not only for production test, but also for field-level test, to diagnose faults in field-replaceable units.

In order to minimize the overhead the BIST technique uses compaction of test response. This also leads to a loss of information. A disadvantage is that the ability to evaluate the test efficiency is rather limited. Usually BIST using Pseudo-random generated test vectors only produces a signal indicating error or no error [Tsu88].

Test Synthesis

In test synthesis the primary goal is to perform synthesis to achieve testability while keeping area and performance overhead under a given constraint.

In high-level synthesis for testability the scheduling, allocation and binding tasks are performed to achieve a testable design. Due to the size of the design space, efficient testability analysis algorithms must be used. Based on the testability analysis the high-level synthesis can be guided to generate testable designs.

Chapter 3
Related Work

The aim of applying a design-for-testability technique on a design is to improve its testability. However, a DFT technique may lead to some design degradation. In order to maximize the testability and to minimize the design degradation, the testability of a design is analyzed and, based on the analysis result, hard-to-test parts are selected for testability improvement.

Several testability metrics have been developed, as well as techniques to improve the testability. Due to the increase in design complexity, testability approaches have been proposed for different abstraction levels. In this chapter we give an overview of metrics for testability analysis in section 3.1. We describe different techniques to improve the testability in section 3.2. In section 3.3 we summarize the discussion.

3.1 Testability Analysis

In this section we give an overview of previously proposed approaches to measuring the testability of a design. The approaches are grouped according to the abstraction level at which they are used.

3.1.1 GATE LEVEL TESTABILITY ANALYSIS

The early work in testability analysis is usually carried out at the gate level [Gol79], [Gol80], [Gup90], [Par93], [Par95], [Abr91]. A testability metrics should, in general, show that a line close to a primary input is easily controlled and a line close to a primary output is easily observable. Even if such distance-based cost approach is unsophisticated it provides guidance in detecting hard-to-test parts in the circuit. The disadvantage of this approach is that it does not consider the logic.

Rutman developed an analysis method based on three measures, *1(0)-controllability* and *observability* where the logic is considered [Rut72]. The 1(0)-controllability measures the relative difficulty of setting a line l in a circuit C to the logic value 1(0). The observability metrics measure the relative difficulty of propagating an error on a line l to any primary output.

Let $C_0(Z)$ be the 0-controllability for line Z, Figure 3.1. To set the output of the AND gate, i.e. line Z to 0, requires that either X or Y is set to 0. The 0-controllability depends on the controllability of the inputs of the AND-gate. Rutman gives a formula for this:

$$C_0(Z) = min\ \{ C_0(X), C_0(Y)\} + 1 \tag{3.1}$$

where the "+1" is used to account for circuit depth.

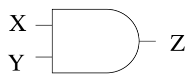


Figure 3.1: A 2-input AND-gate.

For 1-controllability of line Z both X and Y must be 1, the formula is as follows:

$$C_1(Z) = C_1(X) + C_1(Y) + 1 \tag{3.2}$$

To propagate an error signal on X to the output Z we require that $Y=1$ and the formula for observability is:

$$C_2(X) = C_2(Z) + C_1(Y) + 1 \tag{3.3}$$

Using these basic ideas in the formulas above, it is possible by transformation to make them suitable for other types of gates, such as OR, NAND, NOR gates and flip-flops.

The total 0-controllability, 1-controllability and observability values for a circuit is given by S_0, S_1 and S_2. S_0, S_1, and S_2 are calculated using the following formula:

$$S_i = \sum_{l \in L} C_i(l) \qquad (3.4)$$

where i=0,1,2; L is the set of lines in the circuit C.

The total testability of a circuit is then:

$$S = \sum_{i=0}^{2} k_i S_i \qquad (3.5)$$

where k_i are weights assigned to the controllability and observability terms.

A drawback with the formulas for 0-controllability, 1-controllability and observability is that they may lead to problems when the circuit contains reconvergent fanout. For instance, in Figure 3.2, the signals B and C can never be set to the same value, so the correct value of $C_1(X)$ should show that setting X=1 is impossible.

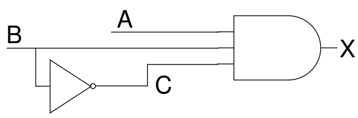


Figure 3.2: Problems in controlling line X.

Based on Rutman's controllability and observability measures Chen and Breuer introduced the sensitivity analysis [Che85]. The basic idea is that an introduction of a DFT technique should affect the global testability and not only the place where the DFT improvement has been applied.

They first measure the total testability of a design using Rutmans' approach and then they calculate the relative improvement for selected points, which is the difference between the testability for the circuit with no DFT compared with applying DFT:

$$\frac{\partial S_i}{\partial S_j(w)} = S_i(S_j(w) = 0) - S_i(S_j(w) = 1) \quad (3.6)$$

where $S_i(S_j(w)=0)$ is the testability with no DFT and $S_i(S_j(w)=1)$ the testability for the circuit with DFT.

Parikh and Abramovici [Par93][Par95] present a method for selecting flip-flops based on sensitivity analysis presented by Chen and Breuer. The sensitivity measure ranks the flip-flops relative to each other based on detectability, which is a measure composed of controllability, sequential depth and enabling cost.

The controllability cost, $C_v(l)$, measures the minimum number of clock cycles required to set line l to the value v. As an example, the 1-controllability of the output of the NAND-gate in Figure 3.3 is the minimum of the 0-controllability values on the inputs.

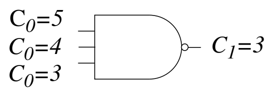


Figure 3.3: Calculating the 1-controllability of a NAND-gate.

The observability cost $O(l)$ is the number of clock cycles required to propagate the value of line l to a primary output, and it consists of two measures, sequential distance and enabling cost. The sequential cost measure, $D(l)$, is the number of flip-flops along the most observable path from l to a primary output. The enabling cost, $E(l)$, is the minimum controllability cost required to enable propagation of a fault effect along a path from the primary input to l. The observability cost is then

$$O(l) = D(l) + E(l) \tag{3.7}$$

The detectability cost for a fault f stuck-at-v at line l is defined as:

$$DET(f) = max\{C_{\bar{v}}l + E(l)\} + D(l) \tag{3.8}$$

The total testability cost for a circuit, called total cost function (*TCF*), is calculated as:

$$TCF = \sum_{f} DET(f) \tag{3.9}$$

The *TCF* value is used as a reference value in the sensitivity analysis, where a change in *TCF* is due to some DFT being used.

An approach similar to sensitivity analysis, called Testability Difference, was proposed by Kim and Kime, who relate the scan flip-flop insertion to an overall improvement of testability [Kim90]. The testability difference is composed of Sequential Controllability Difference, *SCD*, and Sequential Observability Difference, *SOD*.

The flip flops, *FF*, are numbered from 1 to n and the faults, *FLT*, are numbered from 1 to m.

First the *SC* and *SO* for the whole design are calculated when no DFT is applied, i.e. no flip-flops are scanned. After this the *SCD* and *SOD* are calculated for each flip-flop that is scanned. The formulas used are the following:

$$SCD(FF_i) = \sum_{j=1}^{m} \{SC(FLT_j) - SC_i(FLT_j)\} \tag{3.10}$$

$$SOD(FF_i) = \sum_{j=1}^{m} \{SO(FLT_j) - SO_i(FLT_j)\} \tag{3.11}$$

The two measures, SCD and SOD, are then combined to define the testability difference:

$$TD(FF_i) = SCD(FF_i) + SOD(FF_i) \tag{3.12}$$

where $TD(FF_i)$ is the testability difference when FF_i is scanned.

The major drawback with such approaches as sensitivity analysis is the computational complexity. The algorithm must be used several times, first when no DFT is applied and then for each time a DFT is applied. If the designer wants to try n different points to apply DFT, the algorithm is used $n+1$ times. Actually the computational complexity is so high that there is no practical use for the approach [Abr90].

3.1.2 REGISTER-TRANSFER LEVEL TESTABILITY ANALYSIS

On the register-transfer level a common approach to testability analysis has been based on the probabilities of data. Each operation reduces the probability of controlling/observing a line deeply embedded in the design.

Chen and Menon presented a testability analysis based on combinational controllability (CC), sequential controllability (SC), combinational observability (CO) and sequential observability (SO) [Che89]. The controllability metrics are divided into two components, 1 and 0 controllability. Thus, there are six parameters associated with each signal in the circuit.

The combinational controllability measures the probability that a signal has the value 0 or 1. Hence they use two metrics, one for 0, CC_0, and one for 1, CC_1. Using a binary decision diagram the probability of traversing any branch is equal to the combinational controllability of the variable represented by the node from which the branch starts.

The sequential controllability is an estimate of the length of a sequence for setting a signal in a circuit to a specific value. Two values exist and therefore they use two metrics, SC_0 for 0-controllability and SC_1 for 1-controllability. The probability that a change in the input will result as a change in the output is defined as combinational observability, CO. Finally, the sequential observability, SO, is defined as an estimate of the number of time frames required to propagate the effects of a signal change on a line to the primary output.

A similar approach which is also based on probabilities is proposed by Gu [Kuc90], [Gu91], [Gu92], [Gu94],[Gu95b]. For each line in the circuit four metrics exist, combinational controllability (CC), sequential controllability (SC), combinational observability (CO) and sequential observability (SO).

The relationship between the controllability at the output of a functional unit and the controllability at its input depends on the controllability transfer factor (CTF). For the observability there is a relationship between the observability at the inputs of a functional unit and the observability at its output which depends on the observability transfer factor (OTF). CTF reflects the probability of setting a value at a unit's output by randomly exercising its inputs and OTF reflects the probability of observing a unit's input by randomly exciting its other inputs and observing its output.

Both the CTF and the OTF factors are in the range of 0 to 1, where 1 represents the best controllability and observability transfer of a unit.

Another approach based on probabilities is proposed by Flottes *et al.* [Flo97]. The focus is on reconvergence and a transparency metrics for controllability and observability.

The transparency for controllability, T_c, is the ratio of different values that can be set on a functional unit and it is calculated as:

$$T_c = \frac{(C_2 - C_1) \times p}{n} + C_1 \tag{3.13}$$

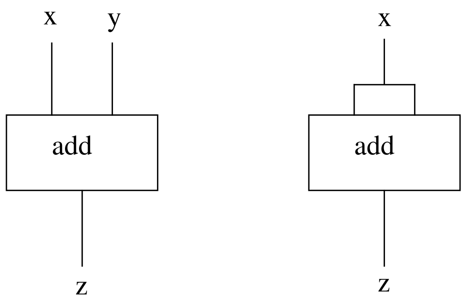


Figure 3.4: Example where p=0 and p=n using the metric proposed by Flottes *et al.*

where n is the bit width of the input ports. p is the number of common bits between the input ports. C_1 and C_2 are the proportion of values that can be obtained on the functional unit's output, given that its input ports are not connected to each other: p=0 (otherwise: p=n). In Figure 3.4 we show an example of z=x+y and z=x+x. In the former case, p=0 since none of the inputs is connected to each other, while p=n when z=x+x since all inputs are connected to each other.

The transparency factor for observability, T_o, is the proportion of pairs of input values that can be distinguished on a functional unit. For instance, for a left-side shift register where n is the number of input bits and with the least significant bit set to 0, the transparency factor is: $T_c = (2^n - 2) / (2^n - 1)$.

The main drawback of the approach is that no loops are accepted in the data path and that the controller is assumed to be testable. A separately testable controller and separately testable data path do not imply that the combination is testable [Dey95].

Lai *et al.* also propose a testability analysis method based on probabilities [Lai97]. The behavioral VHDL is transformed into a Control Data Flow Graph on which scheduling is performed.

After scheduling the testability is measured and the testability is enhanced. The analysis is based on controllability and observability of registers. The controllability metrics are based on entropy, a standard notation from information theory, called randomness [Lai97]. The observability measures the probability that an arbitrary change in the signal's value can be observed at the primary output and is defined as transparency [Lai97].

The randomness, R_c, of a variable c at the output of a component with two inputs a and b is defined as:

$$R_c \approx C_2 \times \frac{R_a + R_b}{2} + C_2 \times \frac{M_a + M_b}{2} + C_3 \tag{3.14}$$

where R_a and R_b are the randomness of variables a and b, and M_a and M_b are the probability distribution of variable a and b assuming a pseudo-random set of input patterns. The coefficients C_1, C_2 and C_3 depend on the bit-length.

The transparency metric for variable c above is:

$$T_c = \frac{T_a + T_b}{2} \tag{3.15}$$

The test generation process is a process of justification and propagating values from a primary input to a primary output. In Figure 3.5 a multiplication of the two variables y and z is shown The propagation of a value from input y to ouptut x depends on the ability to justify a value on z. For instance, if z is always set to 0, no other value than 0 can be propagated on x.

The observability metrics proposed by Rutman depend on the controllability metrics [Rut72]. However, the observability metrics proposed by Gu *et al.* do not for instance consider the justification process when the observability is calculated [Gu95b].

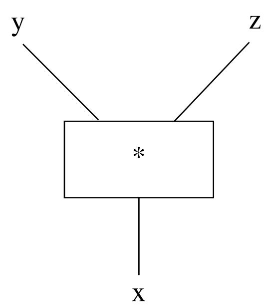


Figure 3.5: Propagation of values from inputs to outputs.

3.1.3 BEHAVIORAL-LEVEL TESTABILITY ANALYSIS

The size of the designs and their complexity are pushing the trend towards high-level designs. A design grows significantly in size when it is synthesized into logic level, i.e. more implementation information is added to the design. This makes it harder to extract information from the design when, for instance we want to find hard-to-test parts.

Until now, most testability analysis methods have focused on the RT level or lower levels. The advantage of working on the RT level and lower levels is that the structure of the design is well-defined.

The major advantages of using high-level testability analysis is the reduction of complexity [Lai97]. However, performing testability analysis before high-level synthesis means that we do not have a structural design where components are allocated, scheduled and assigned.

Chickermane and Patel made a comparative study where high-level test generation was compared with gate-level test generation [Chi92]. Their results show that a high-level DFT tool can accurately predict hard-to-test areas. Furthermore, the high-level DFT can make a more efficient and effective selection of partial scan flip-flops by using high-level circuit information.

On the behavioral level Chen *et al.* introduced a path analysis method to classify, based on controllabilities, the variables into two classes, completely controllable (CC) and non-completely controllable (NCC) [Che92] [Che94]. The classification is based on the ability to control the bits in the variables. If all bits are controllable the variable is classified as CC otherwise as NCC.

The selection process is based on the same assumption as approaches such as the sensitivity analysis [Che93]. An effectiveness value, $EFF(N)$, associated with each NCC variable N, is defined as the relative improvement for applying test point insertion for N:

$$EFF(N) = \frac{\left[\sum_{n_t \in NCC_t} BitSize(n_t) - \sum_{n_m \in NCC_m(N)} BitSize(n_m)\right]}{BitSize(N)}$$

where

- N is an NCC variable,
- NCC_t is the set of original NCC variables,
- $NCC_m(N)$ is the set of NCC variables after variable N has been inserted as a test point,
- $BitSize(N)$ is the number of bits for variable N.

It is not clearly stated, but understood, that the *BitSize* is the number of controllable bits in a variable. However, the variables that are considered in the Effectiveness measure are the variables that are classified as NCC. An Observability measure and classification scheme for the variables is also proposed. However, it is not considered in the selection process.

The few classes and the strict rules for a variable to be classified as a CC variable result in most variables being classified as NNC variables [Le93a]. Another drawback is that no internal loops are allowed in the CDF graph because it might generate an infinite number of paths in the path analysis.

3.2 Testability Improvement

Several techniques have been developed to improve the testability of a digital design. We divide the techniques into three groups, post-synthesis, synthesis and pre-synthesis transformations. Post-synthesis transformations are applicable after the high-level synthesis step, synthesis transformations are applied during the high-level synthesis step, and finally pre-synthesis transformations are applied before the high-level synthesis step.

3.2.1 POST-SYNTHESIS TRANSFORMATIONS

Most DFT techniques have traditionally been post-synthesis techniques. The simplest and most straightforward technique is test point insertion.

As discussed in section 2.4.1, the controllability and/or the observability of a gate-level design can be improved by adding extra I/O pins. The main drawback with test point insertion is the large demand for I/O pins. This problem can be solved in several ways. One way is to add a multiplexer to reduce the number of primary outputs. In Figure 3.6 a multiplexer is added and the need for T_n primary outputs for the T_n test points is reduced to one primary output and one primary input for selection. However, the drawback with the multiplexer is that only one observation point can be observed at a time; hence test time increases [Abr90]. Another approach to reduce the I/O overhead is to use a shift register together with the multiplexer and the demultiplexer [Abr90].

The scan technique discussed earlier uses scan registers with both shift and parallel-load capability. The drawback with the scan technique is that scan flip-flops are larger in terms of silicon area and that additional I/O pins are required. Furthermore, a slower clock may be required because of the extra delay in the scan-path and the test time per pattern increases since each test

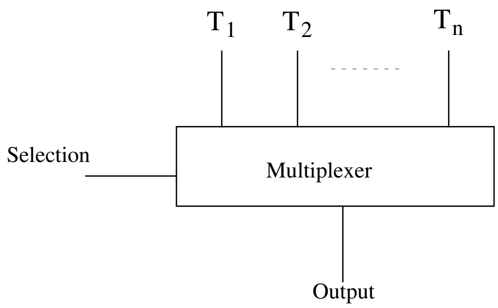


Figure 3.6: Example using a multiplexer to reduce the numper of primary outputs.

pattern has to be shifted in [Abr90]. Several techniques have been proposed to reduce the costs introduced when using scan designs. Norwood and McCluskey propose a technique, called beneficial scan, that combines circuit synthesis and scan chain insertion into one step. Functionality is extracted and used to order the scan chain elements [Nor96].

Initialization is the process of bringing a sequential circuit into a known state at a certain time [Abr90]. This can be achieved by adding a reset line to all flip-flops. Adding a reset to a flip-flop requires less area overhead and delay penalty than scanning the flip-flop [Abr93]. Abramovici *et al.* propose a technique to select which flip-flops should be initialized and at what value.

Ghosh *et al*. propose a technique to multiplex variables which are uncontrollable [Gho95]. The approach is to add multiplexers to the synthesized RTL design to ensure that all modules become testable. Assume we have to propagate the output from an adder through the multiplier in Figure 3.7. If the signal *x* is uncontrollable, we can add a test multiplexer as shown in shadow and multiplex *x* with a test input.

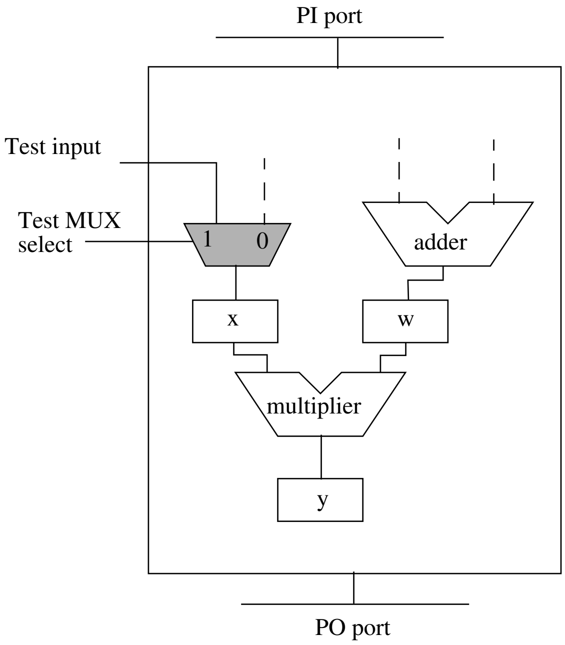


Figure 3.7: Multiplexing with a constant.

3.2.2 SYNTHESIS TRANSFORMATIONS

The high-level synthesis tasks include scheduling, allocation and binding of operations to components and to certain time slots. The tasks must be performed taking testability into account in order to make sure that the synthesized design is testable.

Lee *et al.* proposed a data path scheduling and allocation for testability method based on two heuristics [Lee92], [Le93c]:

- whenever possible, allocate a register to at least one primary input or primary output variable,
- reduce the sequential depth from a controllable register to an observable register.

The scheduling/allocation tasks are performed in separate steps in the approach by Lee *et al*. First scheduling is performed and then allocation is performed. In this way the possibility of testability enhancement is not fully exploited.

An approach where the scheduling and allocation are performed in an integrated way is proposed by Yang and Peng [Yan98a]. The data path allocation approach proposed by Yang and Peng is based on allocation of data path operations using a controllability/observability balance technique. The basic idea is that data path operations with bad controllability and good observability are merged with operations with good controllability but bad observability. The main goal is to generate a data path with good controllability and observability for all nodes and with as few loops as possible [Yan98a].

Yang and Peng use the testability metrics proposed by Gu *et al.* [Gu95b] and for the test synthesis a cost function is proposed:

$$\Delta C = \alpha \times \Delta E + \beta \times \Delta H \qquad (3.16)$$

where α and β are constants defined by the designer. ΔE is the incremental execution time and ΔH is the incremental hardware cost. The hardware cost is computed:

$$H = \sum_i Area(V_i) + \sum_i Len(A_j) \times Wid(A_j) \qquad (3.17)$$

where:

- $Area(V_i)$ is the area cost of the module corresponding to data path node V_i,
- $Len(A_j)$ is the length of the connection represented as data path connection A_j.
- $Wid(A_j)$ is the width of the connection represented as A_j, which is the bit width of the connection multiplied by a given weight factor.

The execution time is equal to the length of the critical path which is detected by analysing the reachability tree of the Petri net model.

Lee *et al.* and Yang and Peng considered the scheduling and allocation part of high-level synthesis while Mujumda *et al.* propose a technique that operates in the binding phase [Muj92]. The technique eliminates as many self-loops as possible during the module and register binding. It is done by modifying the costs associated with the arcs in the network model [Muj92]. If a particular binding tends to increase the number of self-loops, the corresponding arc is given a high cost in order to penalize such binding.

3.2.3 PRE-SYNTHESIS TRANSFORMATIONS

Transformations performed directly in the behavioral specification are classified as pre-synthesis transformations since they are applied before the high-level synthesis process.

A way to reduce the test complexity of a large design is to decompose it to smaller partitions, which are easier to test. A partitioning technique is proposed by Gu *et al.* [Gu95a] where an analysis method is used to select boundary components. These components act as normal registers and/or lines in the normal mode, while they serve as partitioning boundaries in the test mode.

The work is extended by Yang *et al.* [Yan98b] where the testability analysis considers the data path as well as the controller. Further, a quantitative measure is proposed to decide in which cluster to place the boundary components.

Another approach is to use test statement insertion which modifies the behavioral specification to improve its testability [Che94]. The basic idea is to bypass the original statement during test mode. Figure 3.8 illustrates a part of a Control Flow Graph where test statement insertion is applied on the original statement. An extra primary input called test is inserted to distinguish between test mode and normal mode. In test mode n_j is assigned the value of n_{in}, while in normal mode, n_j is assigned the value of n_{org}.

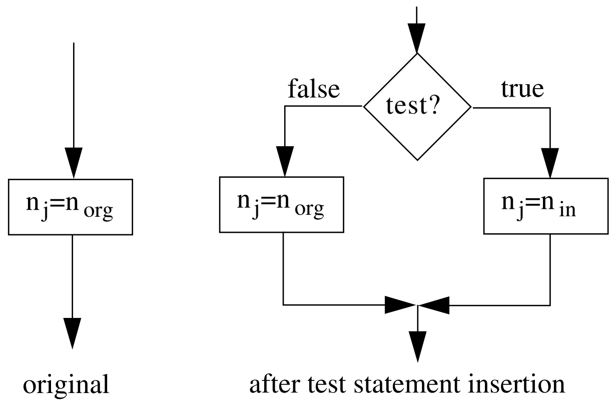


Figure 3.8: Example of test statement insertion.

The problem is to select variable n_{in}. One criterion is that it should be physically close to n_j in order to reduce extra routing. However, such implementation-specific details are not available from the Control Flow Graph.

Varma *et al.* propose a technique that considers the testability of a synthesized design by using knowledge extracted from the behavioral specification [Var93]. A similar technique is proposed by Carletta and Papachristou which makes the behavioral specification testable by modifying it [Car97]. A behavioral-for-test transformation can produce a variety of test behavior. The test behavior, after it is synthesized, may target the testing of all operations of the circuit in parallel. The approach by Carletta and Papachristou uses a transformation which generates a test behavior similar to the design behavior. Such test behavior is easily generated and easily synthesized with the design behavior in a unified way by a synthesis system and the test controller is easily embedded in the system controller [Car97].

An example of adding test behavior proposed by Carletta and Papachristou on a specification is shown in Figure 3.9, where the marked parts are added for the test. In Figure 3.10 the scheduled data flow graph is shown for the design in Figure 3.9,

```
ENTITY example IS
  PORT ( a, b, c, d,
    x, newx : IN BIT_VECTOR(3 downto0)
    test : IN BIT;
    out, newout : OUT BIT_VECTOR(3 downto0)
  );
END example;
ARCHITECTURE behav OF ex IS
  BEGIN
    PROCESS (a,b,c,d,x)
      VARIABLE M1, M2, M3, M4,
        S1, S2, S3 : INTEGER;
    BEGIN
      M1 := a * x;
      S1 := M1 + b;
      newout <= S1;
      IF (test='0') THEN
        M2 := x * x;
      ELSE
        M2 := x * newx;
      END IF;
      M3 := S1 * M2;
      M4 := c * x;
      S2 := M4 + d;
      S3 := s2 + M3;
      out <= S3;
    END PROCESS;
END;
```

Figure 3.9: Example of design-and-test behavior.

and in Figure 3.11 the testable data path for the behavior is found where the inserted elements are in bold. The controller with the embedded test controller is shown in Figure 3.12.

The test input is used to control an extra state in the controller. Due to this the overhead in the controller was high [Car97].

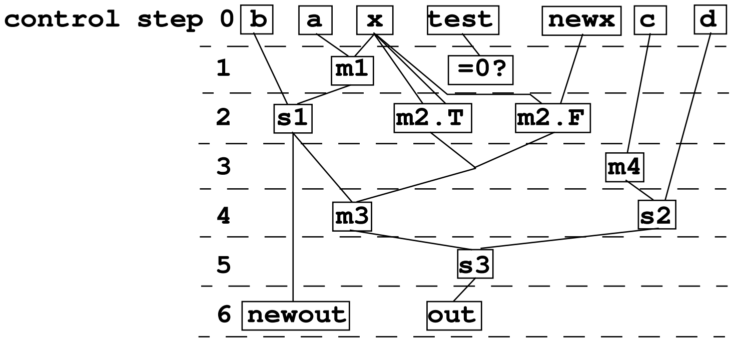


Figure 3.10: Scheduled data flow graph for the design-and-test behavior of Figure 3.9.

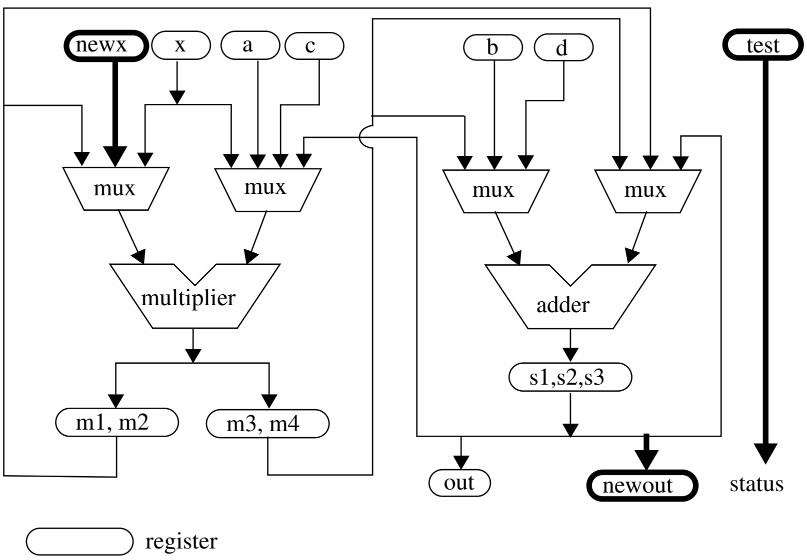


Figure 3.11: The data path for the testable behavior, with inserted elements in bold.

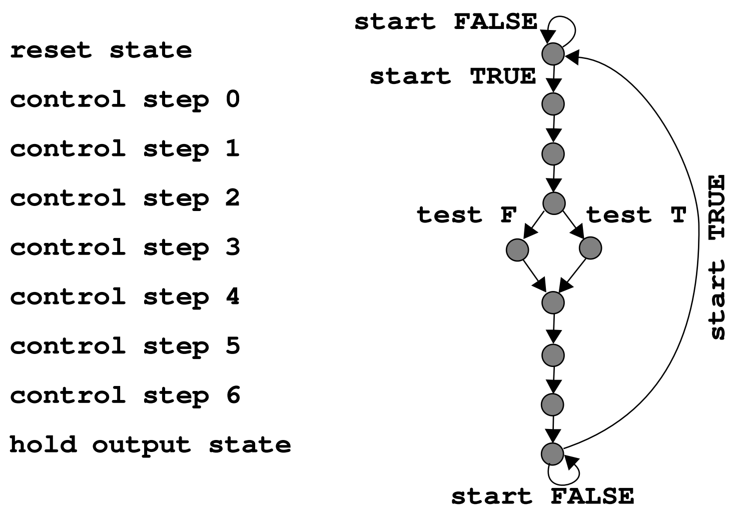
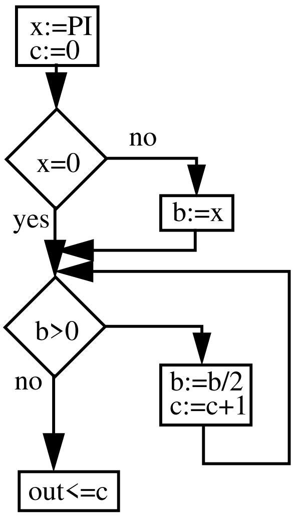


Figure 3.12: Controller with the embedded test controller.

Hsu *et al*. present a testability insertion technique where hard-to-control loops are identified and control points are added at the exit of the hard-to-control loops [Hs96a]. An example is given in Figure 3.13 where the transformations, T1, T2, T3 and T4 are found on the right side.

3.3 Summary

Most research in hardware testing has been focused on lower abstraction levels and several approaches to analyze and improve testability have been proposed. However, due to the increasing complexity of digital designs, modeling techniques at higher abstraction levels have been developed and many design and verification activities take place at these levels. New techniques may be needed for higher abstraction levels. At the behavioral level the functional properties of the design are




```
Type T1: Force False
  x=0 --> x=0 AND C1
Type T2: Force True
  x=0 --> x=0 OR C2
Type T3: Complement
  x=0 --> x=0 XOR C3
Type T4: Load
  IF C4 THEN
    C:= PI
ELSE
  C:=C+1;
```

Figure 3.13: Four types of controllability insertion in the high-level description.

explicitly captured and can be used to speed up testability analysis. This information is difficult to extract from a gate-level design.

A common weakness of the existing testability analysis techniques is the way feed-back loops in the design are handled. Loops are often the cause of problems in test generation and must be considered. However, loops also cause problems for most testability analysis approaches. Flottes *et al.* [Flo97] and Chen *et al.* [Che92] [Che94] assume therefore that no loops exist, and Gu [Gu95b] has only a simple heuristic for loop estimation.

Another problem is that many approaches to RTL only consider the data path. It has been shown that the controller has a large impact on the testability and that a separately testable controller and data path do not mean that the combination is testable.

Most analysis methods are based on the ability to control and observe values on certain lines in the design and the analysis methods aim to guide the designer to find hard-to-test parts. There is usually a trade-off between accuracy in finding the hard-to-test parts and the computational complexity. Approaches such as the sensitivity analysis have such a computational complexity that it is not applicable in practice [Abr90].

Even if all testability analysis approaches provide guidance in detecting hard-to-test parts, no analysis approach has a termination condition. When a hard-to-test part is detected, a DFT-method is applied and the testability is improved. Analysis algorithms based on sensitivity analysis recompute the testability when a DFT is applied and give a relative measure of the improvement but the main question remain; namely, when to terminate the improvement process to guarantee a given fault coverage.

Each testability analysis provides guidance in detecting hard-to-test parts, but few analysis methods provide guidance in the selection of DFT method. By experiments it has been shown that all testability analysis methods detect hard-to-test parts. This has been done by performing experiments but in each approach only one DFT technique has been used. For instance Chen and Breuer use test point insertion [Che93] and Parikh and Abramovici [Par93][Par95] use partial scan. No approach gives any guidance in selecting a DFT technique.

Another common drawback of the proposed testability analysis techniques is that the heuristics for selecting hard-to-test parts only select one part in each design iteration. Therefore after each analysis, only one hard-to-test part is improved and then the design is re-analyzed and its testability is improved again if necessary. This approach is usually justified in that it keeps the overhead introduced by DFT techniques at a minimum and the introduction of a DFT technique usually improves not only the place where it is applied but also the global testability. However, for large designs the number of iterations will also be very large, resulting in a long design time.

In this chapter we have also provided an overview of strategies to improve the testability, classified as post-synthesis, synthesis and pre-synthesis techniques. Some of the traditionally testability techniques, such as test point insertion, BIST and the scan technique, are only mentioned in this chapter. However, they are described in Chapter 2.

The high-level synthesis process can be performed to achieve a testable design. However, one of the problems with synthesis for testability is that the number of loops may be increased due to hardware sharing and the loops in a design are known to cause test problems.

Chapter 4
Testability Analysis

In this chapter a behavioral testability analysis technique is proposed for early prediction of testability by analyzing behavioral VHDL specifications. The technique extracts testability properties by an analysis of variable range, operation testability and statement reachability at a low computational cost. Experimental results show that the behavioral testability analysis technique predicts the hard-to-test parts accurately and efficiently, and can be used to guide the selection of partial scan registers.

After the preliminaries given in section 4.1, the testability metrics are presented in section 4.2. An application of the metrics for partial scan selection is described in section 4.3, and the analysis algorithm is given in section 4.4. Finally, we present experimental results in section 4.5, and conclusions in section 4.6.

4.1 Preliminaries

VHDL is a hardware description language which can be used to model a design at various abstraction levels. At the behavioral level a subset of the VHDL language can be used as input to a high-level synthesis tool. We assume that the behavioral specification is a synthesizable subset of behavioral VHDL [Ele92].

The synthesizable subset that is accepted by our approach includes entity declarations, architecture bodies, package declarations and package bodies with the following properties: an architecture body may contain any number of concurrent statements, scalar and composite types, with the exception of access and file types. Signals can only be of scalar or bit-string type, no recursive calls are allowed and all sequential statements are accepted, with the exception of assertion statements and structural aspects (such as component instantiation or generate statements) which are excluded.

For our testability definitions we assume that Automatic Test Pattern Generation (ATPG) is used and it is random-based and/or deterministic-based and oriented towards the commonly used stuck-at fault model. Many ATPG tools use randomly generated test vectors for finding many easily detected faults and then deterministically generated vectors for harder faults.

4.2 Behavioral Testability Metrics

Our behavioral testability metrics are a combination of *Variable Range*, *Operation Testability* and *Statement Reachability*.

4.2.1 VARIABLE RANGE

If the value range of a variable is limited at a line in the behavioral specification, it reduces the test vector set and makes it harder to test the related hardware.

Definition 4.1: *VR* (l, v) denotes the value range of variable v at line l in the behavioral specification, where $l \in L$. L is the set of lines in the behavioral specification.

For example, if a variable v can have values in the range [-10:10] and [15:20] at line l then *VR* (l, v) = [-10:10, 15:20].

Definition 4.2: *defVR* (v) is the defined value range for a variable v.

Typically *defVR* (v) equals the full range of values defined for a variable v. For example, *defVR* (v) for a 16-bit register declared as a positive integer is [0:2^{16}-1].

Definition 4.3: Let the *relative value range, RVR,* for a variable v at line l, where $l \in L$, be:

$$RVR(l, v) = \frac{|VR(l, v)|}{|defVR(v)|}.$$

The relative value range of a behavioral VHDL example is shown in Figure 4.1, where **`A_IN`** is an input port, **`A_OUT`** is an output port and **`A`** is defined as: **`variable A: integer range 0 to 31`**.

Statements	VR(l,A)	RVR(l,A)
`A:=A_IN;`	0..31	1
`IF A<10 THEN`	0..9	0.31
`  A:=A+5;`	5..14	0.31
`END IF`	–	–
`A_OUT<=A;`	0..31	1

Figure 4.1: An example of Variable Range and Relative Value Range.

4.2.2 OPERATION TESTABILITY

Test vectors applied on the input of a hardware module are used to test the module. If the complete test vector set is available the module is controllable. On the other hand, if some of the test vectors can not be generated, the module is harder to test. The test vector set for a module connected directly to a primary input is complete and uniformly distributed, provided that each bit has a 0.5 probability of being '0' and a 0.5 probability of being '1'. However, the output vector set from the module might not be complete or uniformly distributed, that is, the vectors may occur at different probabilities. The output test vector set which is modified by the first module will be used as an input to the next module. Since the test vector set is no longer complete and uniformly distributed the latter module is harder to test.

We introduce *Operation Testability*, *OpT*, as a metrics that captures the change in distribution of test vectors in the output of an operation assuming all possible test vectors on its input. The optimum case is when the test vectors on an input of an operation are complete and uniformly distributed, the output vectors are also complete and uniformly distributed. This case cannot be satisfied by most operations when they are implemented on hardware.

As an example, in Figure 4.2, the output distribution for a 2-input 3-output adder is shown.

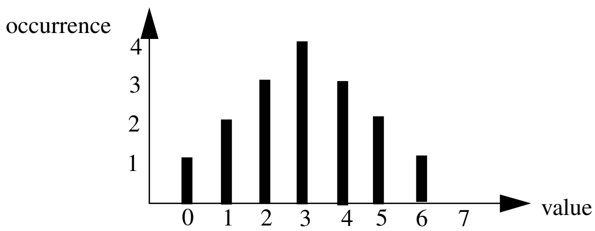


Figure 4.2: Output value (3-bit) distribution for a 2-bit addition.

The difference between a distribution on the output of an operation and an uniform distribution can be measured by the following formula:

$$Q(op) = \sum_{i=1}^{r} \frac{(x_i - n \times p_i)^2}{n \times p_i}$$

where x_i is the number of occurrences of value i, n is the total number of outputs, p_i is the expected probability for value i when each i is assumed to occur with the same frequency, and r is the number of possible values in the output.

In the 2-bit input 3-bit output adder case n=16, p_i=1/8 and r=8, we have:

$$Q(add_{2/3}) = \frac{(1-2)^2}{2} + \frac{(2-2)^2}{2} + \frac{(3-2)^2}{2} + \frac{(4-2)^2}{2} + \frac{(3-2)^2}{2} + \frac{(2-2)^2}{2} + \frac{(1-2)^2}{2} + \frac{(0-2)^2}{2} = 6$$

We are interested in the relative difference in testing different modules. Our metrics aim at showing the relative difference in disturbance of the output due to different operations performed by the modules.

Definition 4.4: The *Operation Testability*, *OpT*, of an operation is defined as:

$$OpT(op) = \frac{1}{Q(op)^{(1/b)}}$$

where b is the word-length, i.e., number of bits of the input operands.

Table 4.1 shows the Q and the *OpT* values for the arithmetic operations at different word-length where a lower value for *OpT* corresponds to an operation which causes a design to be harder to test.

We would also like to use the operation testability metrics to characterize the testability of a line as a whole. We achieve this by the introduction of the following two definitions.

Definition 4.5: The *Test Hardness*, $TH(l)$, at a line l is:

$$TH(l) = \sum_{op \in Op_l} (1 - OpT(op))$$

where Op_l is the set of operations at line l.

Definition 4.6: The *Line Operation Testability*, $LOT(l)$, at line l in the behavioral specification is:

$$LOT(l) = 1 - \frac{TH(l)}{TH_{max}}$$

where TH_{max} is the maximal test hardness at a line in the behavioral specification.

operation		add	sub	mult	div
2-bit	Q	6	6	48	13.5
	OpT	0.41	0.41	0.14	0.27
4-bit	Q	86	86	1408	1196
	OpT	0.33	0.33	0.16	0.17
8-bit	Q	21846	21846	603456	5333181
	OpT	0.29	0.29	0.19	0.14

Table 4.1: Q and OpT values for arithmetic operations.

4.2.3 STATEMENT REACHABILITY

Some statements in the behavioral specification are more difficult to reach then the others due to the control flow. For instance, a statement inside an IF-statement may be hard to reach since the condition has to be fulfilled in order to reach it. Statements that are hard to reach tend to cause test problems.

Definition 4.7: The *true* (*false*) *condition scope*, $cs_t(c_i)$ ($cs_f(c_i)$), of a condition, c_i, is the set of lines in the behavioral specification which will be executed if the condition c_i=*true* (*false*).

The true condition scope, $cs_t(c_1)$, for condition c_1 in Figure 4.3 is the set $\{l_1, l_2\}$ and the false condition scope, $cs_f(c_1)$, for condition c_1 is the set $\{l_3, l_4\}$.

```
(1) IF A<10 THEN   condition c1    true scope c1
(2)   A:=A+1;                          -//-
(3) ELSE                           false scope c1
(4)   A:=A-10;                         -//-
(5) END IF;
```

Figure 4.3: Condition scope.

Definition 4.8: The *Statement Reachability*, *SR* (*l*), is given by:

$$SR(l) = \prod_{c_i \in C} ep_l(c_i)$$

where C is the set of conditions in the specification and the effective probability, $ep_l(c_i)$, for a condition c_i at line l is defined as:

$$ep_l(c_i) = \begin{cases} p_t(c_i) & l \in cs_t(c_i) \\ p_f(c_i) & l \in cs_f(c_i) \\ 1 & otherwise \end{cases}$$

where $p_t(c_i)$ is the probability of condition c_i being true and $p_f(c_i)$ is the probability of c_i being false. The probabilities $p_t(c_i)$ and $p_f(c_i)$ can be obtained by analyzing the value range of the variables involved in the conditions or simulating the specification.

For some conditions the variable range leads to some extreme values for the probability of a condition. For instance for an IF statement where the condition is A=10, this condition will only be true in 1 out of 65536 times if A is a 16-bit integer. In such situations, instead of using the real probability value in calculating the effective probability, we use p_{min} and p_{max} which define the lower bound and the upper bound probability of a condition. Based on experiments we use p_{max} =0.75 and p_{min} =0.25.

4.3 Application of the Behavioral Testability Metrics

The behavioral testability metrics are not targeted at any particular DFT improvement technique. In this section we will show how the behavioral testability metrics can be used when partial scan technique is used.

The variables in the behavioral VHDL can be implemented as registers at the structural level and as flip-flops at the gate level. In high-level synthesis the variables are mapped to registers, and several techniques have been developed to minimize the number of these registers. However, here we conceptually assume that each variable is implemented as a dedicated register. Our objective is to identify already in the behavioral level which variables should be selected as a scan variable and eventually mapped to a scan register.

The flip-flops in the controller, the *state variable,* does not correspond to any variable in the behavioral specification. However, the flip-flops in the state variable are often the cause of problems in test generation and test application and must be considered.

The state variable is indirectly used at every line in the behavioral specification and it can only have one value at each state. If we assume that each line corresponds to one state, then the value range of the state variable at each line is $1/|L|$. We can then use definition 4.3 to get its *RVR* [Lar97].

For partial scan selection we need a metric for variables which reflect all testability features of the variables.

Definition 4.9: The *Testability*, $T(v)$, for a variable v in the behavioral specification is given by

$$T(v) = \sum_{l \in L} \frac{m}{n} \times (\alpha_1 \times RVR(l, v) + \alpha_2 \times LOT(l) + \alpha_3 \times SR(l))$$

where n is the number of times variable v occurs in the behavioral specification and m is 1 if variable v occurs at line l, otherwise m is 0. α_1, α_2 and α_3 are three user-defined constants which are used to reflect the importance of the three metrics for different test strategies.

One feature of our technique is that it allows the scan variable selection procedure to be performed in an iterative manner. After one iteration of analysis and testability insertion we can analyze the testability of the modified design and further improve the testability.

Since any value can be stored and observed in a scanned register we let the value range for a variable v be equals to *defVR* (v) when it is scanned. The *Relative Value Range* for a scanned variable is therefore 1.

Scanning a variable v reduces the *Test Hardness* to:

$$TH(l) = \frac{TH(l) \cdot (i - j)}{i}$$

where i is the number of variables and j the number of variables at line l which are selected to be included in the scan path in this iteration. Note that the state variable affects every line and that we consider it as a variable which appears in each line.

```
THmax=0;
for l=1 to |L| do begin
  for op = 1 to |Opl| do
    TH(l)=TH(l)+(1-get_opt(op));
  if TH(l)>THmax then THmax=TH(l);
  for i=1 to |V| do begin
    VR(vi)=get_variable_range(vi)
    RVR(l,vi)=|VR(l,vi)|/|defVR(vi)|;}
    ni=ni+#times vi is used at line l
    mi,l=1 if vi is used at line l otherwise 0
  end;
  if l in {if, while, for} then calculate SR(l);
  SR(l)=probability for condition scope(l)
end;
for l = 1 to |L| do
  T(vi)=T(vi)+mi,l/ni*(α1*RVR(l,v)+
      α2*(1-TH(l)/THmax+α3*SR(l));
```

Figure 4.4: The testability analysis algorithm.

4.4 Behavioral Testability Analysis Algorithm

The behavioral testability analysis algorithm calculates the testability metrics, value range, operation testability and statement reachability for all lines in the behavioral specification.

The pseudo-code for the algorithm is given in Figure 4.4, where L is the set of lines in the behavioral specification, V is the set of variables, Op_l is the set of operations at a line l.

The algorithm consists mainly of two iterations over the lines in the behavioral specification. After initialization, the first iteration summarize the operation testabilities for all arithmetic operations at a line. For each variable at the line the relative variable range is computed and the statement reachability for the line is calculated. In the second iteration the three metrics are combined into one testability metric.

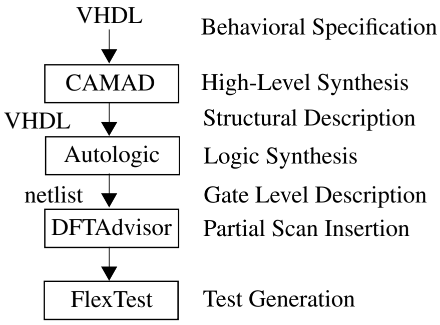


Figure 4.5: The design flow.

4.5 Experimental Results

In this section we present our experimental results. We use the Differential Equation benchmark, Diff, to show the efficiency of using our testability metrics for partial scan selection. We also compare the efficiency of the testability prediction on the behavioral level with those of the gate level, using a set of benchmarks.

We use the CAMAD high-level synthesis tool [Pen94] and Mentor Graphics logic synthesis and test generation tools from release A.4 as the experimental platform, see Figure 4.5. The behavioral VHDL specification is given as input to the CAMAD high-level synthesis tool, which produces a structural VHDL description. Logic synthesis of the structural description is performed by Autologic which generats a netlist [Me93a], [Me93b].

For partial scan insertion we use DFTAdvisor [Me93d] and for test vector generation we use FlexTest [Me93c] with default settings, see Figure 4.5. The benchmarks we use are Diff [Pau89], Sqrt [Tri85], Mag [Tri85], Dct [Kri92] and Tseng [Tse83].

The behavioral specification for the Diff benchmark is given in behavioral VHDL, shown in Figure 4.6. The statement reachability, operation testability and the relative variable range for

```
ENTITY diff IS
  PORT(x_in, y_in, u_in: IN INTEGER;
      x_out, y_out, u_out: OUT INTEGER);
END;

ARCHITECTURE behavior OF diff IS
BEGIN
  PROCESS
    CONST  a = 10000; dx = 1;
    VARIABLE x,y,u,x1,y1,u1 : INTEGER;
  BEGIN                                      Line
    x:=x_in; y:=y_in; u:=u_in;              (1,2,3)
    WHILE x<a LOOP                           (4)
      x1:=x+dx;                              (5)
      u1:=u-(3*x)*(u*dx)-3*y*dx;             (6)
      y1:=y+u*dx;                            (7)
      x:=x1; y:=y1; u:=u1;                   (8,9,10)
    END LOOP;                                (11)
    xout<=x; yout<=y; uout<=u;               (12,13,14)

  END PROCESS;
END behavior;
```

Figure 4.6: The behavioral description of the Diff benchmark.

the Diff benchmark are found in Table 4.1. In Table 4.2 the three metrics are combined into one testability metric. We let α_1=1, α_2=1 and α_3=1, which gives equal importance to the three metrics.

After the analysis of the Diff benchmark we performed experiments to see if the prediction for the testability of the variables was correct. The results are given in Table 4.3. In the table x_1 is ranked as the hardest to test and by scanning it we get the best fault coverage improvement, from 16.49% (no scan) to 97.61%. On the other hand, register y, is considered to be not difficult to test, so scanning it will not improve the design's testability very much; we get only 16.15% fault coverage as the result. We actually get a reduction of the fault coverage when we scan variable y or y_1 compared with no scan. This is due to the fact that variables y and y_1 are easily tested without scan logic and the introduction of scan logic for these variables increases the complexity of the design, which leads to a small decrease in fault coverage.

The comparison between analysis at behavioral and gate level was performed as follows. First we performed a testability analysis on the behavioral VHDL to find hard-to-test variables and rank them. The variables which are hardest to test are scanned. Secondly, we let the Mentor Graphics gate-level DFT-tool select the same number of flip-flops as was selected by our behavioral analysis technique. The results are shown in Table 4.4 where the fault coverage is almost the same for all benchmarks except for the Diff benchmark where the behavioral analysis outperformed the gate-level analysis.

The behavioral testability analysis determined the number of flip-flops to scan. However, the gate-level tool has the advantage of selecting individual flip-flops from any register. The main advantage of the behavioral testability analysis is that it takes much less time to perform the testability analysis task, and therefore speeds up the design process.

Line	Statement Reachability	Operation Testability	Relative Variable Range
1	1	1	x:1
2	1	1	y:1
3	1	1	u:1
4	0.25	1	x:0.15
5	0.25	0.87	x, x1:015
6	0.25	0	y, u, u1:1, x:0.15
7	0.25	0.72	y, y1, u:1
8	0.25	1	x, x1:0.15
9	0.25	1	y, y1:1
10	0.25	1	u, u1:1
11	0.25	1	-
12	1	1	x:0.85
13	1	1	y:1
14	1	1	u:1

Table 4.1: The variable range , operation testability and statement reac hability for the Diff benc hmark.

Line	Testability						
	x	**y**	**u**	**x1**	**y1**	**u1**	**state var**
1	3	-	-	-	-	-	2.07
2	-	3	-	-	-	-	2.07
3	-	-	3	-	-	-	2.07
4	1.4	-	-	-	-	-	2.07
5	1.21	-	-	1.21	-	-	1.13
6	0.40	1.25	1.25	-	-	1.25	0.32
7	-	1.93	1.93	-	1.93	-	1.00
8	1.40	-	-	1.40	-	-	1.32
9	-	2.25	-	-	2.25	-	1.32
10	-	-	2.25	-	-	2.25	1.32
11	-	-	-	-	-	-	1.32
12	2.85	-	-	-	-	-	2.07
13	-	3	-	-	-	-	2.07
14	-	-	3	-	-	-	2.07
T(v)	1.71	2.29	1.91	1.30	2.09	1.75	1.57

Table 4.2: The variable range , operation testability and statement reac hability combined into one test hardness metric for the Diff benc hmark.

Scanned variable	Testability	Fault coverage (%)
x1	1.30	97.61
state register	1.57	97.28
x	1.71	97.69
u1	1.75	91.78
u	1.91	92.33
y1	2.09	15.61
y	2.29	16.15
no scan	not appl.	16.44

Table 4.3: Experimental results for the Diff benchmark.

Design	#flip flops	#scanned flip-flops	Fault coverage(%)		CPU(sec)	
			Behavioral-Level Analysis	Gate-Level Analysis	Behavioral-Level Analysis	Gate-Level Analysis
Diff	100	16	97.3	19.7	2.3	18.8
Sqrt	86	70	99.0	99.0	5.0	19.4
Mag	52	36	95.9	94.9	2.1	5.2
Dct	164	4	98.8	99.1	2.3	7.53
Tseng	213	5	96.3	93.9	2.3	15.1

Table 4.4: Fault coverage comparison of testability analysis at the behavioral and the gate level.

4.6 Conclusions

In this chapter we have proposed a behavioral testability analysis technique for early prediction of testability by analyzing the behavioral VHDL specification. The technique is based on analysis of variable range, operation testability and statement reachability. The testability for the design is predicted at a low computational cost, since the analysis is performed on the behavioral specification which is much less complex then its gate-level implementation.

By experiments we have shown that the behavioral testability analysis technique can predict the hard-to-test parts accurately and efficiently and that the testability analysis results can be used to guide the selection of partial scan registers. The testability metrics also provide an indication of the test features of the final design at a very early design stage. This information can be used by the designer to select an appropriate test strategy and to make an efficient test plan for the final design.

Chapter 5 Testability Improvement Transformations

In this chapter we define a behavioral-level testability improvement transformation technique for modifications applicable directly in the behavioral VHDL specification. The transformations do not impose any restrictions on the high-level synthesis process and we present an application where our behavioral testability metrics are used to guide the testability improvement transformations. Experimental results show the efficiency of our approach.

5.1 Introduction

At the gate-level, test point insertion is a straightforward technique to improve the controllability and/or the observability of a line (wire) in a circuit [Che85]. The technique can also be used on register-transfer level, where a single added test point can be used to control a multiplexer or a register. For instance, Gosh *et al.* use the technique to multiplex variables that are hard to control [Gho95]. On the behavioral level, Varma *et al.*, Carletta and

Papachristou and Chen *et al.* propose a technique which basically is the test point insertion [Var93], [Car97], [Che94]. Carletta have some restrictions on the high-level synthesis and Chen needs knowledge about the physical placement of variables in order to reduce the routing. Hsu *et al.* and Gu *et al.* use the test point insertion technique to improve the controllability of hard-to-test branches [Hs96a], [Hs96b], [Gu97].

A common drawback with the proposed testability analysis techniques is that the heuristics for selecting hard-to-test parts select only one part in each design iteration. Therefore after each analysis, only one hard-to-test part is improved and then the design is re-analyzed and its testability is improved again if necessary. In Section 5.6 we propose a technique to group variables according to how they depend on each other to make it possible to apply several DFT techniques in each design iteration and still keep the DFT overhead as small as possible.

5.2 Testability Improvement Transformations

In this section we will define a set of transformations which are applicable directly on the behavioral VHDL specification in order to improve the testability.

5.2.1 READ-INSERTION

The ability to control the value of a variable in the behavioral specification can be improved by the use of a READ-insertion transformation. The idea is illustrated in Figure 5.1 where the contents of variable *y* is hard to control. In Figure 5.1(b) where test, an extra primary input, is added to determine wheter the design is in normal mode or test mode. In test mode the content of variable *y* is easily controlled. PI is a primary input which can be an existing one or an extra one added only for test purposes.

```
  :                    :
  :                    :
x:=y+5;           IF test THEN
  :                  y:=PI;
  :               END IF;
                  x:=y+5;
                       :

 (a)                  (b)
```

Figure 5.1: Example of READ-insertion.

The penalty introduced by adding an extra primary input only for test is usually too high. Here, we assume that an existing primary input can be used.

5.2.2 WRITE-INSERTION

The WRITE-insertion transformation improves the ability to observe the contents of a variable. The idea is illustrated in Figure 5.2. In Figure 5.2(a) variable x is hard to observe and by applying WRITE-insertion we improve its observability as shown in Figure 5.2(b, c). Here we distinguish two types of WRITE-insertions. The first type writes a value direct on a primary output dedicated for test, Figure 5.2(b). In this case, we do not have to check whether the output is produced during test mode or normal mode. However, if we use WRITE-insertion and use an existing primary output for observing, we have to add an extra pin to check whether we are in test mode or normal mode, which is illustrated in Figure 5.2(c).

Again, the cost of extra primary outputs which will only be used for test is usually unacceptable. Therefore we usually assume the case where an existing primary output is used.

```
                  :                        :
     :            :                        :
     :          x:=y+5;                  x:=y+5;
   x:=y+5;      new_PO:=x;               IF test THEN
     :            :                        PO:=x;
     :                                   END IF;
                                           :
   (a)            (b)                      (c)
```

Figure 5.2: Example of WRITE-insertion.

5.2.3 BOOLEAN-INSERTION

When test point insertion for improving the controllability was introduced, it was defined to be applicable on any wire in a gate-level design by adding an extra AND-gate or an extra OR-gate. For higher level design specifications it would be possible to use the same strategy. However, improving the 1-controllability for one 16 bit register would require 16 OR-gates and 16 extra primary inputs. The introduced over-head penalty, the extra primary inputs, is usually too high. However, for certain structures in the behavioral specification, such as loops and branches, which are known to cause test problems, the test point insertion, as it was originally defined, can be used.

Below we define three types of BOOLEAN-insertions: OR-insertion, AND-insertion and AND/OR-insertion. They are used on the following VHDL constructs: IF, WHILE, FOR and CASE statements.

We use the example in Figure 5.3 to illustrate the use of the BOOLEAN-insertion technique. In Figure 5.3(a) there is a loop construct using the WHILE statement. To improve the ability to terminate the loop we use the AND-insertion, which adds an extra primary input pin called test and an AND-gate, on the WHILE condition, Figure 5.3(b). The result is that by setting the added test pin to false, we can determine when to exit the loop.

```
     :                      :
     :                      :
while z<10 loop        while z<10 and test loop
  z:=z+1;                z:=z+1;
end loop;              end loop
     :                      :
     :                      :

    (a)                    (b)
```

Figure 5.3: Illustration of the use of AND-insertion.

The concept of OR-insertion is similar to the AND-insertion.

The OR-insertion and the AND-insertion require knowledge about which outcome (true or false) of a condition is hard to achieve. For instance in Figure 5.3(a) we have at the WHILE statement a condition *c* (z<10) and in Figure 5.3(b) we use AND-insertion with an extra primary input *test* to form the new condition *c'* which is: *c* and *t*. By using the AND-insertion we increase the ability to force the condition *c'* to false (*c'* will be false if *t* is false) which improves the controllability greatly if it is hard to set condition *c* to false.

In the truth table for the OR-insertion and AND-insertion we note that for OR-insertion the value TRUE (T) has a higher probability of occur than the value FALSE (F) and vice versa when AND-insertion is used. We note that with the test point we can easily control the outcome of the condition. In the OR-insertion case, a TRUE value can easily be obtained, and in the AND-insertion case, a FALSE value can easily be obtained.

By using AND/OR-insertion as shown in Figure 5.4 we achieve full control of the outcome of condition *c*. When in test mode, test is true, and we can set the condition to TRUE or FALSE.

The difference between this insertion and READ-insertion is that the former focuses on the condition while the latter focuses on the content of a specific variable.

```
      :
if test then
  c := PI;
end if;
while c loop
  z:=z+1;
end loop;
      :
```

Figure 5.4: Example of AND/OR-insertion.

5.2.4 REACH-INSERTION

As discussed earlier, some statements in the behavioral VHDL specification are harder to reach compared to other statements due to the control flow. They tend to form hard-to-test parts. The REACH-insertion transformation is used to make the hard-to-reach statements easier to reach when the design is in the test mode.

In Figure 5.5 we illustrate the REACH-Insertion on a small VHDL example and a corresponding control-data-flow graph. By adding an extra primary input pin to determine whether the design is in normal mode or test mode, the hard-to-reach statements can be easily reached in the test mode. In the example, Figure 5.5, the statement *y:=x+4* can, in the test mode, be reached without having to traverse the loop structure first.

5.3 Cost Function for DFT Selection

When a hard-to-test part has been identified, its testability can be improved by applying some DFT technique. Usually, there exist several DFT techniques which can be applied and it is up to the designer to select an appropriate technique for a certain hard-to-test part.

```
    :
    :
if not test then
  while x<10 loop
    x:=x+1;
  end loop;
end if;
y:=x+4
    :
    :
```

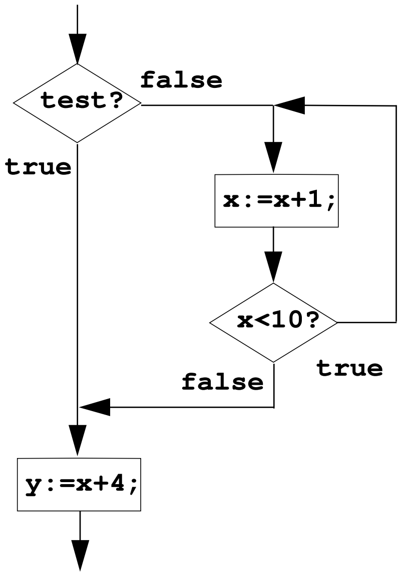


Figure 5.5: Example of REACH-insertion.

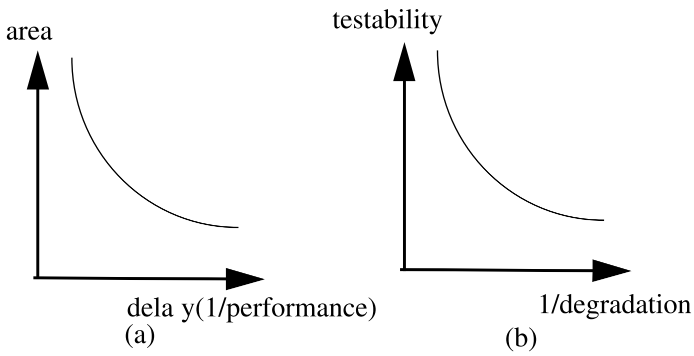


Figure 5.6: Illustration of the design space .

Traditionally the design space has been simplified to a two-dimensional space over area and performance , whic h is illustrated in Figure 5.6(a). If the design is optimized tow ards small delay (improved performance), it usually means that the required area is increased and on the other hand, if the area is minimized, it usually means reduced performance of the design. Today, the design space is multi-dimensional, consisting not only of traditional costs such as area and performance but also costs such as power consumption and testability.

Our aim is to achieve a testable design at a low degradation which is illustrated in Figure 5.6(b) where we have kept the two-dimensional view. We combine all degradation into one component on one axis and the testability on the other. The problem is to find a trade-off between testability and degradation.

An advantage of the general cost function T is that it makes the improvement and degradation into a two-dimensional problem. The degradation using a transformation could be a combination based on penalty introduced during normal mode and test mode. During normal mode the area, power consumption and performance degradation may be changed and during test mode the power consumption, test generation and test application time may be changed due to the transformation:

$$d(t) = \alpha_1 * area(t) + \alpha_2 * power(t) + \alpha_3 * performance(t) + \alpha_4 * test\_power(t) + \alpha_5 * test\_gener(t) + \alpha_6 * test\_appl(t) \quad (5.1)$$

where α_1, α_2, α_3, α_4 and α_5 are user-defined constants.

5.4 Application of the Testability Improvement Transformations

In this section we will show how the testability improvement transformations can be used to improve the testability of a behavioral VHDL specification.

5.4.1 SELECTION BASED ON LOOP BREAKING

As discussed earlier, loops in a design are known to cause test problems and several approaches have been proposed where all loops, except self-loops are broken [Che90], [Chi91], [Lee90]. A design synthesized from a behavioral specification may contain different types of loops. Some of the loops can not be traced in the behavioral VHDL specification. For instance a loop introduced due to hardware sharing is not predictable in the behavioral specification.

However, some loops may be identified straight from the behavioral VHDL specification. For instance, statements such as WHILE and FOR will form control loops and feed-back loops and a statement such as A:=A+B will also form a loop. By an analysis of the behavioral VHDL specification it is possible to identify and break these loops in order to facilitate the test problem.

The loop-breaking approach is simple and straightforward. However, for large designs where the number of loops may be large it might not be feasible to break all loops. In this case a method must be used to select a subset of the loops.

5.4.2 SELECTION BASED ON TESTABILITY METRICS

In Chapter 4 we used the scan technique to show the correlation between our metrics and fault coverage. The scan technique improves both the controllability and the observability by making it possible to store any value directly in the register and to observe any value of the register directly on a primary output. The READ-insertion improves the controllability of a variable since any value can easily be assigned to the variable from a primary input and the WRITE-insertion improves the observability of a variable since any contents of the variable can easily be produced on a primary output.

If we assume that a READ-insertion and a WRITE-insertion on a variable correspond to scanning the corresponding register, we can use the same approach as in Chapter 4 with minor modifications to fit READ and WRITE-insertion.

As in the case of partial scan selection we need metrics for variables which reflect all testability features of the variables. We use the same *Testability*, $T(v)$, metric as defined in Section 4.3.

The main drawback with this approach is that we do not use the BOOLEAN-insertion, which has a low penalty. The testability flow for the approach without BOOLEAN-insertion is shown in Figure 5.7(a) and in Figure 5.7(b) we have extended the testability flow to include BOOLEAN-insertion. Since the penalty for using a BOOLEAN-insertion is low and it is targeted towards a well-known problem, we give high priority to selecting it. In our approach we select BOOLEAN-insertion for a variable if it is among the k hardest variables according to a test metric, where k is defined by the designer. The BOOLEAN-insertion is targeted towards branches detectable in the behavioral VHDL specification and these branches are formed by the statements IF, WHILE, FOR and CASE.

The BOOLEAN-insertion affects the ability to control the condition and for a given condition we have to select which BOOLEAN-insertion to use. The OR-insertion improves the ability to set the condition to true, while using the AND-insertion it is easier to set the condition to false. The AND/OR-insertion makes it easier to set the condition to any value, true or false.

The BOOLEAN-insertions affect the statement reachability and the variable range. To guide the selection of BOOLEAN-insertion, we define a relative improvement metric .

The set of lines affected by a condition c is captured $cs_t(c)$ and $cs_f(c)$, defined by Definition 4.7 in Chapter 4.

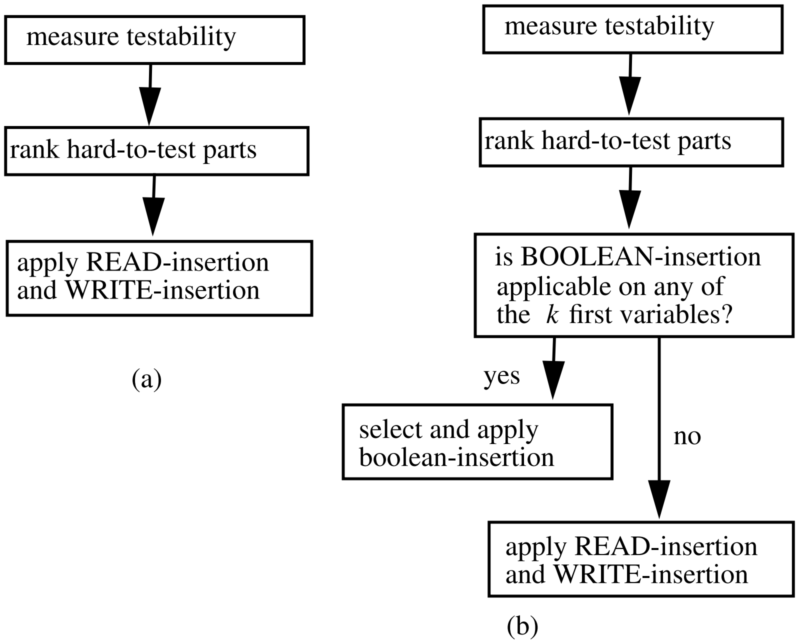


Figure 5.7: Extension of the transformation selection to include BOOLEAN-insertion.

Definition 5.2: Let the Condition Testability, *CT(c),* for a condition *c* be defined as:

$$CT_i(c) = \sum_{l \in cs_t(c) \cup cs_f(c)} \frac{1}{n} \times (RVR(l, v) + SR(l, v))$$

where *n* is the number of times a variable occurs at the lines in the condition scope and *i* is 0 when no insertion is used, 1 for AND-insertion, 2 for OR-insertion and 3 for AND/OR-insertion.

Definition 5.3: Let the Relative Condition Testability, $RTC_i(c)$, for a condition *c* be defined as:

$$RCT_i(c) = CT_i(c) - CT_0(c)$$

where *i* is 1 for AND-insertion, 2 for OR-insertion and 3 for AND/OR-insertion.

```
    :
A:0..31, B:10..20
WHILE A<10 LOOP
  B:=B+A;
  A:0..10, B:10..31
    :
END LOOP;
A:11..31, B:10..31
    :
```

(a)

```
    :
A:0..31, B:10..20
WHILE A<10 OR T LOOP
  B:=B+A;
  A:0..31, B:10..31
    :
END LOOP;
A:11..31, B:10..31
    :
```

(b)

```
    :
A:0..31, B:10..20
WHILE A<10 AND T LOOP
  B:=B+A;
  A:0..10, B:10..30
    :
END LOOP;
A:0..31, B:10..30
    :
```

(c)

```
    :
A:0..31, B:10..20
IF T THEN C:=PI;
WHILE C LOOP
  B:=B+A;
  A:0..31, B:10..31
    :
END LOOP;
A:0..31, B:10..31
    :
```

(d)

Figure 5.8: Illustration of how the variable range changes when BOOLEAN-insertion is used.

In Figure 5.8(a) a sample of VHDL code is shown as well as the variable range for the variables. In Figure 5.8(b) we have used the OR-insertion and we see how the variable range for the variables is changed, in Figure 5.8(c) we use AND-insertion and in Figure 5.8(c) AND/OR-insertion.

In the above approach we used READ-insertion in combination with WRITE-insertion. For each variable we want to improve we apply both a READ-insertion and a WRITE-insertion. These transformations could be used independently on each other. For instance, the READ-insertion could be applied on a variable without using the WRITE-insertion and vice versa.

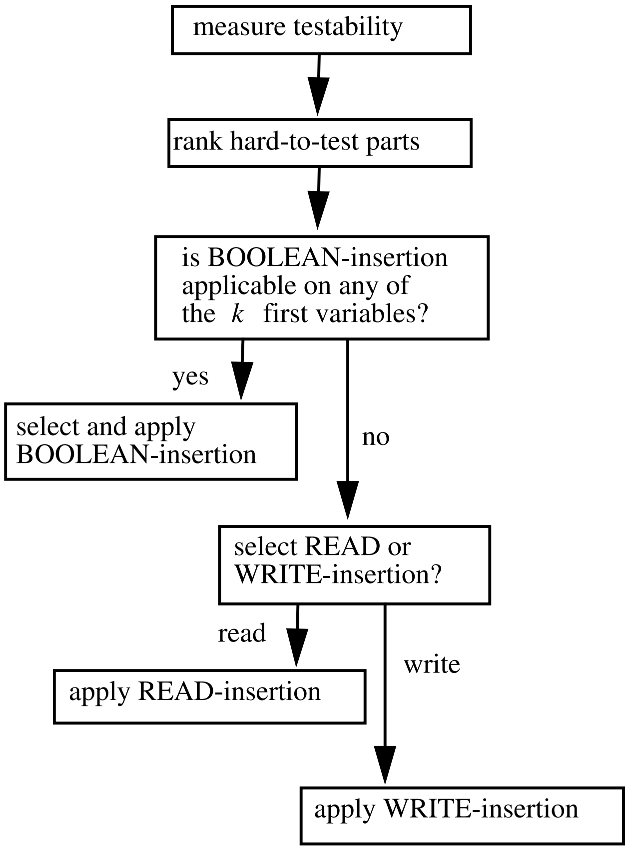


Figure 5.9: Extension of the transformation selection to distinguish between READ-insertion or WRITE-insertion.

We extend the testability flow to distinguish between READ-insertion and WRITE-insertion, Figure 5.9.

Until now we have used the behavioral testability metric as it was defined when partial scan selection was used. We will extend it to make it possible to select either READ-insertion or WRITE-insertion for a variable. We use the testability metric as it was defined for partial scan to rank the variables and we define a strategy to determine whether we should apply a

READ-insertion or a WRITE-insertion on a variable. We note that, the READ-insertion and the WRITE-insertion are applicable on lines with assignments and arithmetic operations.

Definition 5.4: If variable v is the left-hand variable in a statement consisting of an assignment and arithmetic operations, it affects the observability, or otherwise the controllability.

If, for instance, we want to improve the observability of a statement $a:=b+c$, the use of a WRITE-insertion which improves the observability is best placed on the output of the adder, i.e on variable a. On the other hand, to increase the controllability of the adder, a read statement is best placed on either variable b or c.

For the statement $a:=b+c$ Definition 5.4 will let the testability metric for variable a affect the observability for variable a, while for variable b and c the metric will affect the controllability part.

5.5 Experimental Results

In this section we present our experimental results on the testability transformations and we use the experimental platformed described in Section 4.5. We have used the Counter example (Cnt) and the Differential Equation benchmark (Diff) [Pau89]. The behavioral specification for the Diff benchmark is given in behavioral VHDL, in Figure 5.10 and the behavioral specification for the counter is shown in Figure 5.11.

In our first experiment we want to test if partial scan on a register corresponds to applying a READ and a WRITE-insertion on the corresponding variable. The results of the experiments for Diff are shown in Table 5.1. We have the variables/register ordered according to our testability metrics. We see that variable x_1 is ranked as the hardest to test variable and when partial scan is applied on it we get a fault coverage of 97.61%. If instead we apply a READ and WRITE-insertion on variable x_1 (which is

```
ENTITY diff IS
  PORT(x_in, y_in, u_in: IN INTEGER;
      x_out, y_out, u_out: OUT INTEGER);
END;

ARCHITECTURE behavior OF diff IS
BEGIN
  PROCESS
    CONST   a = 10000; dx = 1;
    VARIABLE x,y,u,x1,y1,u1 : INTEGER;
  BEGIN                                     Line
    x:=x_in; y:=y_in; u:=u_in;              (1,2,3)
    WHILE x<a LOOP                          (4)
      x1:=x+dx;                             (5)
      u1:=u-(3*x)*(u*dx)-3*y*dx;            (6)
      y1:=y+u*dx;                           (7)
      x:=x1; y:=y1; u:=u1;                  (8,9,10)
    END LOOP;                               (11)
    xout<=x; yout<=y; uout<=u;              (12,13,14)

  END PROCESS;
END behavior;
```

Figure 5.10: The behavioral specification of the Diff benchmark.

the corresponding variable in the behavioral specification), we get a fault coverage of 96.3%.

When partial scan is used on variable y_1 we only get 15.61% in fault coverage . Applying a READ and WRITE-insertion on the corresponding variable in the behavioral specification gives a fault coverage of 16.8%.

```
ENTITY counter IS
  PORT( x : IN INTEGER;
        y : OUT INTEGER );
END;
ARCHITECTURE behave OF counter IS
BEGIN
  VARIABLE z : INTEGER;
  PROCESS BEGIN                         Line
      z:=x;                             (1)
    WHILE z>0 LOOP                      (2)
      z:=z-1;                           (3)
    END LOOP;                           (4)
    y<=z;                               (5)
  END PROCESS;
END;
```

Figure 5.11: The behavioral specification of the Counter benchmark.

Selected register	Testability	fault coverage(%)	
		scan	READ/WRITE-insertion
x1	1.30	97.6	96.3
x	1.71	97.7	97.3
u1	1.75	91.8	96.1
u	1.91	92.3	94.6
y1	2.09	15.6	17.5
y	2.29	16.2	16.8

Table 5.1: Experiments to show the correlation between our testability metrics, partial scan and READ/WRITE-insertion.

For all registers/variables we see that we have a small difference in fault coverage when we compare partial scan with applying a READ-insertion and a WRITE-insertion. The largest difference is found on register/variable u_1. When it is scanned we achieve a fault coverage of 91.78% but when READ/WRITE-insertion is used the fault coverage is 96.1%. However, even in this case, the difference is only 4.7%. In the next experiment we want to test our strategy for selecting READ-insertion or WRITE-insertion for a variable. The testability analysis of the design is performed and the variables are ranked according to their testability. For each ranked variable we will use our strategy to determine if READ-insertion or WRITE-insertion is appropriate. On lines in the behavioral VHDL specification where READ-insertion and WRITE-insertion can be considered we compare the ratio observe/control. Lines where we have conditions, IF, FOR, WHILE and CASE, are not considered; nor a line directly connected to a primary input or primary output. A high value indicates that a WRITE-insertion is preferable, while a low value indicates that a READ-insertion is to be prefered. In Table 5.2 the testability metric for the Diff benchmark is divided into a controllability part and an observability part and in Table 5.3 the experimental results from the transformation selection is shown.

We have the ranked variables according to our testability metric. For instance, variable x_1 is ranked as the hardest to test variable with testability of 1.30. The control/observe ratio is 0.87 which compared to the c/o ratio for all other variables is high, which indicates that we should use a READ-insertion. By using a READ-insertion on variable x_1 we achieve a fault coverage of 94.5%. We performed experiments where we applied the non-recommended insertion. For variable x_1 we performed experiments where the WRITE-insertion was used and the achieved fault coverage was only 15.7%.

l	Testability											
	x		**y**		**u**		**x1**		**y1**		**u1**	
	c	**o**	**c**	**o**	**c**	**o**	**c**	**o**	**c**	**o**	**c**	**o**
1	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-
5	1.21	-	-	-	-	-	-	1.21	-	-	-	-
6	0.40	-	1.25	-	1.25	-	-	-	-	-	-	1.25
7	-	-	1.93	-	1.93	-	-	-	-	1.93	-	-
8	-	1.40	-	-	-	-	1.40	-	-	-	-	-
9	-	-	-	2.25	-	-	-	-	2.25		-	-
10	-	-	-	-	-	2.25	-	-	-	-	2.25	-
11	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-
Σ	1.61	1.40	3.18	2.25	4.43	2.25	1.40	1.21	2.25	1.93	2.25	1.25
o/c	0.87		0.71		0.51		0.86		0.86		0.56	

Table 5.2: The testability metric divided into a controllability(c) part and an observability(o) part for each line (l) in the Diff benchmark.

In the experiments above we have only used the READ-insertion and the WRITE-insertion; the BOOLEAN-insertion technique was not used. However, since the cost of BOOLEAN-insertion is low and it is targeted towards known test problems it must also be considered.

Variable	Testability	c/o	Selected transform	f.c(%)	Not recommended transform	f.c(%)
x1	1.30	0.87	read	94.5	write	15.7
x	1.71	0.86	read	94.5	write	15.7
u1	1.75	0.55	write	95.2	read	14.1
u	1.91	0.51	write	94.7	read	16.9
y1	2.09	0.86	read	12.7	write	70.7
y	2.29	0.85	read	13.1	write	16.5

Table 5.3: Experiments to show the correlation between our strategy to selecting READ or WRITE-insertion and the fault coverage .

The experiment using the approach when BOOLEAN-insertion, READ-insertion and WRITE-insertion are considered is as follows. The variables are ranked according to their testability using our testability metrics. If BOOLEAN-insertion is applicable on any of the k hardest variables, it is selected for BOOLEAN-insertion. If not, we select READ-insertion or WRITE-insertion for the hardest to test variable. The size of the set k is determined by the designer. Here we let k be 50% of the variables, which for the Diff-benchmark is 3.

Variable x_1 is ranked as the hardest variable to test and the metric c/o indicates that a READ-insertion should be used. By using READ-insertion on variable x_1, we achieve a fault coverage of 94.5%, while when we use a WRITE-insertion we only achieve a fault coverage of 15.7%. On variable y_1 our heuristic indicates that a READ-insertion should be used. However, using the READ-insertion provides a fault coverage of 12.7%, while using a WRITE-insertion we achieve a fault coverage of 70.7%.

In Table 5.4 the variables in the Diff benchmark are ranked according to their testability. We test whether any of the k (=3) first variables is a candidate for BOOLEAN-insertion starting by checking the hardest to test variable. Variable x_1 is not, but x is. We stop the search and select a BOOLEAN-insertion on x.

We also performed experiments on the counter and the results are presented in Table 5.5, where we see that in both cases the BOOLEAN-insertion is used and that a high fault coverage is achieved at a very low area penalty.

Variable	**Testability**	**BOOLEAN-insertion possible?**	**BOOLEAN-insertion**		
			and	**or**	**and/or**
x1	**1.30**	**no**	**-**	**-**	**-**
x	**1.71**	**yes**	**95.7**	**96.8**	**96.8**
u1	**1.75**	**no**	**-**	**-**	**-**
u	**1.91**	**no**	**-**	**-**	**-**
y1	**2.09**	**no**	**-**	**-**	**-**
y	**2.29**	**no**	**-**	**-**	**-**

Table 5.4: Experiments to show the correlation between our strategy to select testability insertion and the fault coverage.

Design	**Transformation**	**Fault coverage (%)**		**Area overhead (mm^2)**	
		No dft	**Dft**	**No dft**	**Dft**
cnt	**1 OR-insertion**	**39.7**	**96.3**	**0.5525**	**0.5539**
diff	**1 OR-insertion**	**13.5**	**96.8**	**7.3596**	**7.3674**

Table 5.5: Experimental results for selection of testability transformation.

5.6 Variable Dependency

Much research has focused on testability analysis where the aim is to determine the hard-to-test parts in a design. When the hard-to-test parts are ranked, only one part is selected for testability improvement. The drawback with this strategy is that the heuristics for selecting hard-to-test parts select only one part in each design iteration. Therefore after each analysis, only one hard-to-test part is improved and then the design is re-analyzed and its testability is improved again if necessary. A different way to keep the DFT overhead small and reduce the number of design iterations is to group the hard-to-test parts according to how they depend on each other. In each iteration, hard-to-test parts from different groups can then be selected and improved without affecting each other. The number of design iterations can be reduced if more than one hard-to-test part is selected and improved in each iteration. The aim of variable dependency analysis is to group the variables based on their dependency.

The testing problem is mainly a problem of justifying values from a primary input to a variable and then propagating values from a variable to a primary output. The controllable paths for a variable v are used to justify a value for variable v and the observable paths are used to propagate the value of the variable to a primary output. Any path from a primary input to a variable can be used for the justification process, and any path from a variable to a primary output can be used for the propagation process. However, we assume that the justification process will mainly use the shortest path from some primary input to the variable and the propagating process the shortest path from the variable to a primary output.

Let $G(V,E)$ be a directed graph where a vertex v in V corresponds to a variable in the behavioral specification (the state variable is not considered).

Definition 5.5: A *start vertex* is a vertex which gets its value directly from a primary input and an *end vertex* is either a variable which is connected to a primary output or a variable which is not used later.

A variable which is not used later comes from statements such as FOR statements where an index variable is used to keep track of the number of loops. Such a variable might not be used later in the design.

Definition 5.6: For a statement of the form $v_i = v_j \; op \; v_k$, where v_i, v_j and v_k are variables and *op* is an operation, there exists an edge (v_j, v_i) in E from vertex v_j to v_i and an edge (v_k, v_i) in E from vertex v_k to v_i.

Definition 5.7: A path, P_i, is a sequence of edges $\{(v_0, v_1), (v_1, v_2),..., (v_{n-1}, v_n)\}$ where v_0 is a start vertex and v_n is an end vertex.

Definition 5.8: The *Shortest Controllable Path*, *SCP* (v), for a variable v, is the shortest path from a start vertex to variable v, and the *Shortest Observable Paths, SOP* (v), is the shortest path from variable v to a primary output.

For instance in Figure 5.12, the shortest controllable path for variable C is the path *PI->A->B->C*, *SCP* (C)={A,B}, and the shortest observable path is the path *C->E->PO*, *SOP* (C)={E}.

The *SCP* and *SOP* information will be used to group variables together in such a way that only one variable will be selected for testability improvement from a group during a design iteration.

An algorithm for grouping variables is given in pseudo-code in Figure 5.13. First the testabilities for all variables are ranged so that T_1 is the hardest to test and T_V is the easiest to test (V is the total number of variables). The hardest variable, T_1, is always selected for DFT improvement. Then the algorithm checks so that variables that use parts of the same shortest observable path and parts of the same shortest controllable path are grouped together. If a variable does not use a previously used

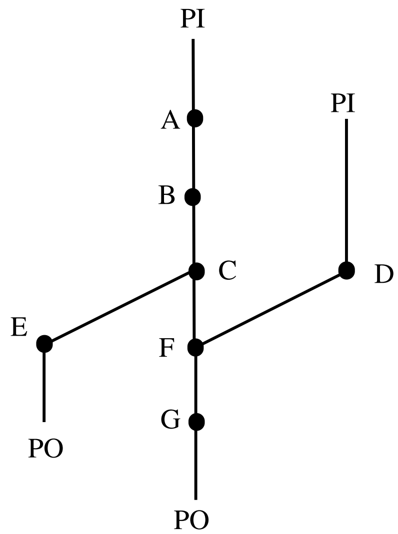


Figure 5.12: Example of Variable Dependency.

```
Sort the testabilities T(v) so that T_1 is the
hardest and T_V is the easiest to test.
Select T_1 for DFT improvement.
for i = 2 to |V|
  Diff = ∅;
  for j = 1 to i − 1
    Diff=Diff+SOP(v_i) ∩SOP(v_j)+SCP(v_i)∩SCP(v_j);
  if Diff = ∅ then
    select T_i for DFT improvement
```

Figure 5.13: Algorithm for grouping variables.

path, it is selected for DFT improvement. The iteration is terminated when all variables are checked.

To illustrate the algorithm we use the example in Figure 5.12. First the variables are sorted and ranked according to their testability. Let us say we have the ranking: *C, F, B, D, G, E, A*. Variable *C* is selected for DFT improvement since it is the hardest-to-test variable. The algorithm checks whether the second hardest variable, variable *F*, can be selected for DFT improvement.

If variable *F* does not use any of the paths used by *C*, it can be selected for DFT improvement. For variable *C* we have $SCP(C)=\{A,B\}$ and $SOP(C)=\{E\}$ and for variable *F* we have $SCP(F)=\{D\}$ and $SOP(F)=\{G\}$. Variable *F* can be selected since the intersections between $SCP(C)$ and $SCP(F)$ and $SOP(C)$ and $SOP(F)$ respectively are empty.

In the next iteration the algorithm checks whether variable B uses any of the paths previously used. In our example we compare the paths used by variable C and F with the paths used by variable B. The algorithm terminates when it has checked all variables.

5.7 Conclusions

In this chapter we have proposed several testability improvement transformation techniques which improve the testability of a behavioral VHDL specification without imposing any restrictions on high-level synthesis. We have also proposed a general technique to select an appropriate DFT technique given a hard-to-test part. We have shown the efficiency of our approach by experiments.

In this chapter we have also proposed a variable grouping scheme, based on the dependency of variables, which can be used to reduce the number of design iterations since several variables can be selected and improved in each iteration.

Chapter 6
Testability Analysis and Enhancement of the Controller

This chapter presents a testability analysis and improvement technique for the controller of an RT level design. It detects hard-to-reach states by analyzing both the data path and the controller of a design. The controller is modified using register initialization, branch control, and loop termination methods to enhance its state reachability. This technique complements the data path scan method and can be used to avoid scanning registers involved in the critical paths. Experimental results show the improvement of fault coverage with a very low area overhead.

6.1 Introduction

As discussed before, many DFT techniques require large area overhead and may degrade the performance of a circuit. Some approaches have been proposed to reduce these drawbacks by using techniques which have low area and performance impact, such as partial scan design [Che94], [Dey93], [Gu94], [Le93b], [Tho94].

Recently Dey *et al.* proposed a DFT technique to improve the controller testability for designs which consists of a controller and a data path [Dey95]. A technique has been developed to identify the control signal conflicts due to control signal correlation imposed by the controller specification. The controller is redesigned in such a way that the identified implications are eliminated by adding extra control vectors.

A synthesis-for-testability approach that uses control points at the conditional branches to improve testability was also proposed by Hsu *et al.* [Hs96a], [Hs96b]. An analysis of the controllability of branch conditions in the control-data flow graph identifies hard-to-control loops. The controllability of the hard-to-control loops is enhanced by inserting control points at the exit conditions of these loops. Test statements are also added if necessary to allow hard-to-control variables to be directly controllable from existing primary inputs.

In this chapter, we propose a general testability analysis and enhancement technique for the controller of a design. It measures the combinational and sequential hardness to reach any state in the controller. The register initialization, branch control and loop termination methods are developed to improve the state reachability of hard-to-reach states.

6.2 Preliminaries

In this section, we first introduce our design representation and testability analysis technique for data path. Our design environment allows designers to specify their designs in behavioral VHDL. The specification is translated into an internal representation, called ETPN [Pen94], which consists of two parts: a data path and a controller. Figure 6.1 presents an example of a behavioral VHDL specification and the corresponding ETPN representation. To simplify the example we assume that the VHDL process is only executed once. The data path is a directed graph with nodes and lines (arcs) where a node represents storage or manipulation of data and a line connecting two nodes represents the flow of data. The controller is modelled as a timed Petri net. The two parts are related through the states (Petri net places) in the controller controlling the data transfers in the data path, and the condition signals in the data path controlling some transition(s) in the controller.

As an example, in Figure 6.1 state S_4 in the controller is used to control the data transfer from input port P_1 to register Y in the data path. When S_4 holds a token, this transfer will take place. Condition nodes C_1 and $\overline{C}_1$ in the data path control the transitions from S_3 to S_6 and the transition from S_3 to S_4 in the controller respectively. State S_0 initially holds a token. The token will be transferred to the consequent state(s) in the next clock cycle. The execution will terminate when all tokens in the controller are consumed. For example, in Figure 6.1, when the token in state S_1 is consumed, the execution will stop.

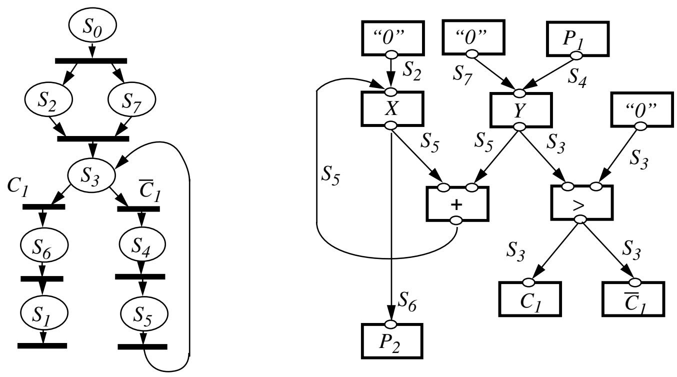
The testability analysis of the data path [Gu94] is defined by the measurements of controllability and observability. The controllability measures the cost of setting up any specific value on a line. The observability, on the other hand, measures the cost of observing any specific value on a line. The controllability and observability measurements reflect respectively two procedures during test generation and test application: the fault sensitization and the fault propagation. Both the controllability and the observability are further defined by two factors: combinational factor and sequential factor. We have, therefore, four measurements for testability of the data path: combinational controllability (CC), sequential controllability (SC), combinational observability (CO) and sequential observability (SO) [Gu95b].

The testability analysis takes into account the structure of a design, the depth from I/O ports and the characteristics of the

```
ENTITY counter IS
  PORT(P1 : IN INTEGER;
        P2 : OUT INTEGER);
END;

ARCHITECTURE behave OF counter IS
BEGIN
  PROCESS( P1 )
    VARIABLE X, Y : INTEGER;
  BEGIN
    X:=0; Y:=0;
    WHILE NOT (Y>0) LOOP
      Y:=P1;
      X:=X+Y;
    END LOOP;
    P2<=X;
  END PROCESS;
END;
```

(a) behavioral VHDL



(b) controller (c) data path

Figure 6.1: A design example in VHDL and ETPN .

components used. It reflects the test generation complexity and test application time for achieving high fault coverage . Improving testability in the data path can be made by transforming some registers with the worst testability analysis measurements to scan registers [Gu94].

6.3 Controller Testability Analysis

The controller testability is measured in terms of the state reachability for each state (Petri net place) of the controller. The state reachability is defined by the difficulty of reaching the state from an initial state. It consists of two measurements: combinational state reachability (CSR_i) and sequential state reachability (SSR_i), for a given state S_i.

Initial State: The initial state, S_0, as illustrated in Figure 6.2, has the best state reachability:

$$CSR_0 = 1 \tag{6.1}$$

$$SSR_0 = 0 \tag{6.2}$$

CSR_0 is assigned to 1 because the probability of reaching this state is 1 and SSR_0 is assigned to 0 because no clock cycles are required to reach this state.

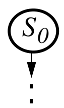


Figure 6.2: Initial state.

Simple Construct: A simple construct consists of one transition with a single input place (S_i) and a single output place (S_j) as illustrated in Figure 6.3. The state reachability will be calculated as:

$$CSR_j = CSR_i \tag{6.3}$$

$$SSR_j = SSR_i + 1 \tag{6.4}$$

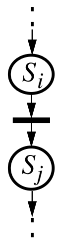


Figure 6.3: Simple construct.

The combinational state reachability for state S_j is the same as that of state S_i. The sequential state reachability of S_j is the state reachability of S_i plus one since one more clock cycle is required to reach state S_j.

OR-Construct : An OR-construct consists of a set of transitions connected to a state such that a state can be reached by any of the transitions in this set. For example, in Figure 6.4, state S_k can be reached either by the transition between state S_i and state S_k or by the transition between state S_j and state S_k. The state reachability is calculated based on the assumption that we can always reach state S_k from a state with the best state reachability. Therefore, we have:

$$CSR_k = \begin{cases} CSR_i & \text{if } \left(CSR_i + \dfrac{SSR_L}{SSR_i} > CSR_j + \dfrac{SSR_L}{SSR_j}\right) \\ CSR_j & \text{otherwise.} \end{cases} \quad (6.5)$$

$$SSR_k = \begin{cases} SSR_i + 1 & \text{if } CSR_k = CSR_i \\ SSR_j + 1 & \text{if } CSR_k = CSR_j \end{cases} \quad (6.6)$$

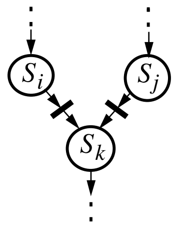


Figure 6.4: OR-construct.

where SSR_L is the largest sequential state reachability in the design, which is an estimation of the longest path from the initial state to the terminating state.

Conditional Construct: In a conditional construct, a state can be reached through a transition only if the condition attached to the transition is true. In Figure 6.5, state S_i can be reached from state S_k only if condition C is true. Otherwise, $\overline{C}$ is true and state S_j will be reached from state S_k. The state reachability is calculated by considering the combinational controllability (CC_c) and the sequential controllability (SC_c) of the condition node in the data path:

$$CSR_i = CSR_k \times CC_c \tag{6.7}$$

$$SSR_i = \mathrm{Max}\{SSR_k, SC_c\} + 1 \tag{6.8}$$

$$CSR_j = CSR_k \times CC_{\bar{c}} \tag{6.9}$$

$$SSR_j = \mathrm{Max}\{SSR_k, SC_{\bar{c}}\} + 1 \tag{6.10}$$

where CC_c is the combinational controllability of the condition attached on the state transition [Gu95b]. The SC_c is the sequential controllability of the condition, i.e., the number of clock cycles required to control the condition [Gu95b]. If the condition is used to control the exit from a loop which has a very large repetition count, we will have a large SC_c which reflects the implication of this loop construct.

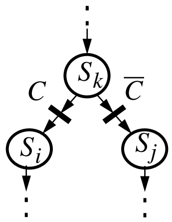


Figure 6.5: Conditional construct.

AND-Construct: An AND-construct consists of a transition such that a state is reachable through the transition when all input states to the transition are reached (hold a token). In Figure 6.6, S_k is reachable only when both state S_i and state S_j are reached. The state reachability of state S_k is calculated by:

$$CSR_k = CSR_i \times CSR_j \tag{6.11}$$

$$SSR_k = \text{Max}\{SSR_i, SSR_j\} + 1 \tag{6.12}$$

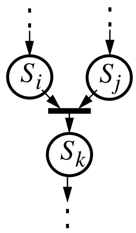


Figure 6.6: AND-construct.

Parallel Construct: In a parallel construct, a set of states will be reached by firing a transition. Figure 6.7 shows a parallel construct. The state reachability of states S_i and S_j in the figure is calculated by the same formula as in the simple construct.

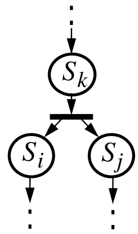


Figure 6.7: Parallel construct.

6.4 State Reachability Analysis Algorithm

The state reachability analysis algorithm calculates the combinational state reachability and the sequential state reachability for all states in a controller. It starts by assigning the state reachability to all initial states and putting these states in a FIFO queue, Q. The breadth-first search strategy is used during the selection of states for calculation. In the next step, one state S is taken out from Q. The construct type of the transition from S to its consequent state(s), for example AND-construct, is checked and the appropriate formulae are used for calculating its CSR and SSR. This procedure is repeated until Q is empty.

A basic sketch of the algorithm is given in Figure 6.8. The calculation of reachability measurements for states included in loops is difficult. Their reachabilities depend not only on some reachabilities already computed but also on reachabilities not yet computed for the states involved in the loop. Our algorithm deals with this problem by first assigning to each state the worst reachability and then updating the reachability only when it is better than the previously assigned value (step 7 in the algorithm in Figure 6.8).

It has to be noted that a loop consists of both conditional and OR-constructs and formulae 6.5-6.10 are used to compute their reachabilities. These computations involve calculation of controllability factors for the conditions controlling the loop execu-

1. Assign all initial states (use formulae 6.1 and 6.2).
2. Put all initial states into queue Q.
3. Assign the rest of the states with the worst CSR and SSR : $CSR := 0$; $SSR := SSR_L$;
4. If Q is empty, then go to 9; else assign the first state in Q to S_{prev}, and remove it from Q.
5. Check the output transition(s) type from S_{prev}:
   a) if it is a simple construct:
   go to 6 (use formulae 6.3 and 6.4).
   b) if it is an AND-construct:
   check if all the other input state(s) have been calculated.
   if "yes", go to 6 (use formulae 6.11 and 6.12).
   if "no", then put S_{prev} to Q and go to 4.
   c) if it is an OR-construct: go to 6 (use formulae 6.5 and 6.6).
   d) if it is a conditional construct or parallel construct:
   go to 6 (use formulae 6.7, 6.8, 6.9 and 6.10 or formulae 6.1 and 6.2).
   e) if it is a terminating transition (leading to an empty state): go to 4.
6. Reach the consequent state(s) S_{cons} and calculate its CSR and SSR by the corresponding formulae.
7. If the newly calculated CSR and SSR are better than stored ones for S_{cons}, replace the stored CSR and SSR by the newly calculated ones and put S_{cons} into Q.
8. Go to 4.
9. End.

Figure 6.8: State Reachability Analysis Algorithm.

tion and thus our reachability calculation takes into account the additional difficulty of controlling the loop exit. The controllability factor calculation for conditions is carried out separately during the data path testability analysis process [Gu95b].

The algorithm produces two reachability measurements for every state S_i, CSR_i and SSR_i. To evaluate the total state reachability we combine these two measurements using the following formula:

$$CSR_i + k \times \frac{SSR_L}{SSR_i} \tag{6.13}$$

where SSR_L is the largest SSR in the design and k is the ratio between CSR and SSR given by designers. This formula is used in selecting the difficult-to-reach states for improvement.

6.5 Controller Testability Enhancements

After analyzing and evaluating the state reachability for all states in the controller, we can identify the hard-to-reach states. Different techniques are then used to make these states easy to reach. In the following, we will discuss several of these techniques.

6.5.1 REGISTER INITIALIZATION

When a register in the data path is hard to initialize due to the hard-to-reach state in the controller, the register initialization/setting technique can be used to improve this situation. Figure 6.9 illustrates the method of enhancing the controllability of setting/initializing register Reg_j through register Reg_i.

This method finds an accessible point in the data path (either a scan register or an input port, such as scan register Reg_i in the figure) which has a short "distance" to the input of the register to be initialized (such as register Reg_j in Figure 6.9) and a short "distance" from the state controlling the accessible point to the state controlling the register in the controller. The distance in the data path is measured by the number of components between the accessible point and the register. The distance in the controller is measured by the number of transitions between

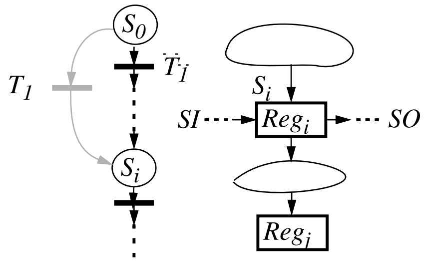


Figure 6.9: Initialize or set Reg_j through Reg_i.

the state controlling the accessible point and the state controlling the register .

In the controller, we improve the state reachability of setting/initializing the register by introducing an extra conditional transition from an initial state to the state controlling the accessible point directly. The condition is controlled by a test signal, T_1. Thus, the transition can be fired when the T_1 signal is true and we can easily set/initialize a register through the closest accessible point to the register. This method has another more important feature, namely that the start execution point of a circuit can be controlled by transferring token(s) from the initial state(s) directly to the state(s) where we want to start the execution and getting the input value(s) from the input port(s) and/or scan register(s). This feature can significantly improve the efficiency of test generation.

6.5.2 BRANCH CONTROL

The state reachability enhancement for a state which is reached through a transition controlled by a condition is required when the controllability of the condition is poor. We assume that the controllability of condition C in Figure 6.9 is poor. To enhance the state reachability of state S_i, we modify condition C to $C \vee T_2$ and $\overline{C}$ to $\overline{C} \wedge \overline{T}_2$, where T_2 is a test signal. When T_2 is true, the transition controlled by the new condition $C \vee T_2$ will be fired, no

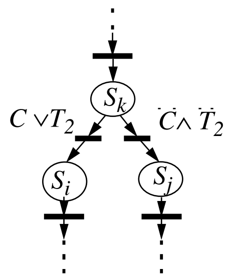


Figure 6.10: Select branch by T_2.

matter what value C has. If we only need to enhance the reachability of state S_i, i.e., state S_k and other previous states are not required during test, we can use the same method as the control enhancement for register setting/initialization to enhance the reachability of state S_i.

6.5.3 LOOP TERMINATION

Feedbacks usually take huge computing time in test generation. The control of feedback termination can not only simplify test generation and shorten test application, but more importantly it can increase the fault coverage by making fault detection easier. Assume a loop running from 10 down to 0. The register holding the loop variable will contain 0 at the end of the loop. It will facilitate testing if we can get other values in the register at the end of the loop. By adding a test point, we make other values possible. In the example we may terminate the loop at any value from 0 to 10. Thus, we will achieve higher fault coverage.

We assume that the controllability of condition C in Figure 6.9 is poor. To enhance the state reachability of state S_i, we modify condition C to $C \vee T_3$ and $\overline{C}$ to $\overline{C} \wedge \overline{T}_3$, where T_3 is a test signal. When T_3 is true, the transition controlled by the new condition $C \vee T_3$ will be fired, no matter what value C has.

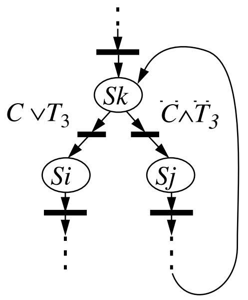


Figure 6.11: Terminate feedback by T_3.

6.6 Experimental Results

We used the Mentor Graphics synthesis and test generation tools as an experimental platform with the default setting used in its test generation process.

We performed experiments on a simple counter, which consists of a controller and a data path, to show the importance of considering the whole design, Table 6.1. When only the data path is considered, we achieve a fault coverage of 98.8% and when only the controller is considered, we achieve a fault coverage of 98.0%. However, we only achieve a fault coverage of 39.7% when the whole design, both the data path and the controller, is considered.

Design	Fault coverage (%)
Only data path	98.8
Only controller	98.0
Data path+Controller	39.7

Table 6.1: Example to illustrate the importance of considering both the controller and the data path.

The benchmarks we used are a counter, a differential equation (diff) [Pau89], and mag [Tri85]. The results are presented in Table 6.2, where the area is measured in mm^2. We applied the loop termination technique to two benchmarks, counter and diff. In the counter benchmark, the fault coverage with no DFT technique is 25.23% and the fault coverage with loop termination technique is increased to 84.67%, an improvement of 235.63%. The area overhead is only 0.25%. In general, the loop termination technique has very low area overhead and is efficient when a design has complicated control loop(s). In benchmark mag, there is no loop. We used the branch control technique instead. The fault coverage increased from 51.17% to 65.23% and the area overhead is 0.5%. When the register initialization method is used, considerable improvement in terms of fault coverage has also been achieved. For example, with diff, the fault coverage is increased from 13.20% to 98.06%, with a overhead of 1.27%.

design	**DFT technique**	**fault coverage (%)**	**area (mm^2)**
counter	no DFT	25.23	0.5525
	loop termination	84.67	0.5539
diff	no DFT	13.20	7.3596
	loop termination	96.33	7.3674
	register initialization	98.06	7.4534
mag	no DFT	51.17	1.6435
	branch control	65.23	1.6528
	register initialization	77.73	1.6989

Table 6.2: Summary of Experimental Results .

6.7 Conclusions

In this chapter, we have presented a method to analyze the testability of a controller. It measures the combinational and sequential hardness to reach each state in the controller. Based on this result, hard-to-reach states are detected and three testability enhancement techniques have then developed to improve the state reachability.

This method has the advantage that it does not suffer from the timing penalty which data path scan technique usually does. It can be used as complement to data path scan in order to achieve better test quality and fewer area and timing penalties.

Experimental results show that this method can effectively improve fault coverage with a very limited area overhead.

Chapter 7
Conclusions and Future Work

The aim of this thesis is to propose techniques to analyze and improve the testability of digital designs at high abstraction levels. Much previous work has, however, focused at lower abstraction levels. The increasing complexity in digital design has motivated new modelling techniques at higher abstraction levels and to reduce the test problems it is important to consider the testability as early as possible.

In this section we will summarize the thesis and point out some directions for future work.

7.1 Conclusions

The modification of a design to make it more testable usually leads to some design degradation in terms of implementation cost or performance. To minimize the design degradation and maximize the testability, it is common to use an analysis technique to find the best trade-off between testability and design degradation.

Chapter 7

In this thesis we propose a technique for analyzing the testability of a behavioral-level specification. Based on the analysis result, we perform testability improvement transformations directly on the specification. Our behavioral-level testability analysis technique is based on variable range, operation testability and statement reachability, and it has a low computational cost. We have shown the correlation between our testability metrics and the fault coverage by experiments, where the results from the testability analysis is used to guide the partial scan selection. By experiments where we compare partial scan selection using our behavioral-level testability metrics with a commercial gate-level tool, we show that the testability can be predicted efficiently and accurately at the behavioral level.

We have also proposed a technique for modifying the behavioral specification to make it more testable and by experiments we have shown the efficiency of the approach. Traditionally only one hard-to-test part has been improved in each design iteration, which is justified since it keeps the degradation at a minimum. However, for large designs the number of design iterations can be numerous. In this thesis we propose a technique for reducing the number of design loops by selecting several hard-to-test parts in each design iteration.

A register-transfer-level design typically consists of a controller and a data path, where the controller controls the flow of data in the data path. Much research has focused on the data path. We propose a testability analysis of the controller and a technique for enhancing its testability. The controller testability analysis is based on statement reachability and the result from the controller testability analysis can be used to guide testability transformations of the controller. The transformations are loop termination, register initialization and branch control and by experiments we have shown the efficiency of our approach.

7.2 Future Work

The following topics are candidates for future work:

- **Termination condition for the testability analyzing/ improving process**. A termination condition decides when to stop the testability analysis and improvement iterations. The normal approach is to analyze the testability and then improve the testability of one-hard-to test part. After that the testability is re-analyzed and a new hard-to-test part is selected. The process will then continue. The question is when to stop this iteration process. Current practice is usually based on trial-and-error. Future work might address a termination condition which can be used in a fully automated approach.
- **Combine the proposed controller metrics with the data path metrics proposed by Gu *et al***. In this thesis we have defined a testability metrics for the controller and Gu *et al.* [Gu95b] have proposed a metric for the data path. The controller metrics depend on the data path metrics and the data path metrics depend on the controller metrics. Future work would be to combine the two metrics in order to achieve a global testability metrics.
- **Selection of DFT-technique**. Much work in defining testability analysis techniques has focused on detecting the hard-to-test parts. By experiment using some known DFT technique the efficiency of the metrics is shown. However, several DFT techniques exist and they have different advantages and disadvantages. Future work will be to define a heuristic which guides the selection of DFT technique.
- **Experiments on larger benchmarks**. The experiments performed in this thesis were all made on rather small benchmarks. It would be desirable to apply the testability technique on larger design examples.

References

[Abr90] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman, Digital Systems Testing and Testable Design, IEEE Press, 1990, ISBN 0-7803-1062-4.

[Abr91] Miron Abramovici, J. J. Kulikowski, and R. K. Roy, The Best Flip-Flops to Scan, *Proceedings of the International Test Conference*, pp. 166-173, Nashville, 1991.

[Abr93] Miron Abramovici, Prashant S. Parikh, Ben Mathew, Daniel G. Saab, and Melvin Breuer, On Selecting Flip-Flops for Partial Reset, *Proceedings of the International Test Conference*, pp. 1008-1012, Baltimore, 1993.

[Car97] Joan E. Carletta, and Christos A. Papachristou, Behavioral Testability Insertion for Datapath/Controller Circuits, *Journal of Electronic Testing: Theory and Applications 11,* pp. 9-28, 1997.

[Che85] Ting-Huai Chen, and Melvin Breuer, Automatic Design for Testability Via Testability Measures, *IEEE Transactions on Computer-Aided Design, vol. CAD-4,* No. 1, January 1985, pp. 3-11, 1985.

[Che89] C. H. Chen, and P . R. Menon, An Approac h to Functional Level T estability Analysis , *Proceedings of the International Test Conference,* pp . 373-380, 1989, Washington.

[Che90] Kw ang-Ting Chen, and V ishw ani D . Agra wal, A P artial Scan Method for Sequential Circuits with F eedbac k, *IEEE Transactions on Computers,* vol. 39., no . 4, pp . 544-548, 1990.

[Che92] Chung-Hsing Chen, BET A: Beha vioral T estability Analyzer and its Application to High-Level Test Generation and Synthesis for T estability , *PhD Dissertation,* Department of Electrical Engineering , University of Illinois at Urbana-Chapaign, 1992.

[Che93] Chung-Hsing Chen, and Daniel G . Saab , A Novel Beha vioral T estability Measure , *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , pp . 1960-1993, vol. 12, no . 12, Dec . 1993.

[Che94] Chung-Hsing Chen, T ana y Karnik, and Daniel G . Saab , Structural and Beha vioral Synthesis for Testability T echniques , *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , pp . 777-785, vol. 13, no . 16, J une 1994.

[Chi91] Vivek Chic kermane , and J anak H. P atel, A F ault Oriented Partial Scan Design Approac h, *Proceedings of the International Conference on Computer-Aided Design* **,** pp . 400-403, Santa Clara, 1991.

[Chi92] Vivek Chic kermane , J aushin Lee , and J anak H. Patel, A Comparative Study of Design for Testability Methods Using High-Level and Gate-Level Descriptions , *Proceedings of the International Conference on Computer-Aided Design* **,** pp . 620-624, Santa Clara, November 1992.

[Dey93] Sujit Dey, Miodrag Potkonjak and Rabindra Roy, Exploiting Hardware Sharing in High Level Synthesis for Partial Scan Optimization, *Proceedings of the International Conference on Computer-Aided Design*, pp. 20-25, Santa Clara, November 1993.

[Dey94] Sujit Dey and Miodrag Potkonjak, Transforming Behavioral Specifications to Facilitate Synthesis of Testable Designs, *Proceedings of the International Test Conference*, pp. 184-193, Washington, October 1994.

[Dey95] Sujit Dey, Vijay Gangaram, and Miodrag Potkonjak, A Controller-Based Design-for-Testability Technique for Controller-Data Path Circuits, *Proceedings of the International Conference on Computer-Aided Design,* pp. 640-645, San Jose, November 1995.

[Ele92] Petru Eles, Krzysztof Kuchcinski, Zebo Peng, and Marius Minea, Compiling VHDL into a High-Level Synthesis Design Representation, *Proceedings of the EURO-DAC,* pp. 604-609, Hamburg 7-10, September 1992.

[Flo97] M.L. Flottes, R. Pires, and B. Rouzeyre, Analyzing Testability from Behavioral to RT level, *Proceedings of the European Design & Test Conference*, pp. 158-165, Paris, March 1997.

[Fje92] Björn Fjellborg, Pipeline Extraction for VLSI Data Path Synthesis, *Ph.D. Dissertation,* Department of Computer and Information Science, Linköping University, 1992.

[Gaj92] Daniel Gajski, Nikil Dutt, Allen Wu, and Steve Lin, High-Level Synthesis, Introduction to Chip and System Design, *Kluwer Academic Publisher*, 1992, ISBN 0-7923-9194-2.

[Gol79] La wrence H. Goldstein SCOAP: Sandia Controllability/Observability Analysis Program, *IEEE Transactions on Circuits and Systems*, Vo. CAS-26, No. 9, pp. 685-693, September 1979.

[Gol80] La wrence H. Goldstein, and Evelyn L. Thigpen SCOAP: Sandia Controllability/Observability Analysis Program, *Proceedings of the Computer-Aided Design,* pp. 190-196, Minneapolis, J une 1980.

[Gho95] Indradeep Ghosh, Anand Raghunathan, and Niraj K. Jha, Design of Hierarc hical T estability of RTL Circuits Obtained by Beha vioral Synthesis, *Proceedings of the International Conference on Computer Design*, Austin, October 1995.

[Gup90] Rajesh Gupta, Rajiv Gupta, and Melvin A. Breuer, The B ALLAST Methodology for Structured P artial Scan Design, *IEEE Transactions on Computers,* vol. 39, no . 4, April 1990, pp . 538-544.

[Gu91] Xinli Gu, Krzysztof Kuc hinski, and Zebo P eng T estability Measure with Reconvergent F anout Analysis and Its Applications, *The Euromicro Journal, Microprocessing and Microprogramming,* vol. 32, nr 1-5, August, 1991, pp . 835-842.

[Gu92] Xinli Gu, Krzysztof Kuc hinski, and Zebo P eng , An Approac h to T estability Analysis and Improvement for VLSI Systems, *The Euromicro Journal, Microprocessing and Microprogramming,* vol. 35, nr . 1-5, September , 1992, pp . 485-492.

[Gu94] Xinli Gu, Krzysztof Kuc hinski, and Zebo Peng, Testability Analysis and Improvement from VHDL Beha vioral Specifications *, Proceedings of EURO-DAC,* pp. 644-649, Grenoble , September 1994.

[Gu95a] Xinli Gu, Krzysztof Kuchinski, and Zebo Peng, An Efficient and Economic Partitioning Approach for Testability, *Proceedings of the International Test Conference,* Washington D. C.*,* October 1995.

[Gu95b] Xinli Gu, RT Level Testability Improvement by Testability Analysis and Transformations, *PhD Dissertation, No. 414*, Department of Computer and Information Science, Linköping University, Sweden.

[Gu97] Xinli Gu, Erik Larsson, Krzysztof Kuchcinski, and Zebo Peng, A Controller Testability Analysis and Control Enhancement Technique, *Proceedings of European Design and Test Conference*, pp. 153-157, Paris, March 1997.

[Hs96a] Frank F. Hsu, Elizabeth M. Rudnick, and Janak Patel, Testability Insertion in Behavioral Descriptions, *Proceedings of International Symposium on System Synthesis,* pp. 139-144, La Jolla, November 1996.

[Hs96b] Frank F. Hsu, Elizabeth M. Rudnick, and Janak Patel, Enhancing High-Level Control-Flow for Improved Testability, *Proceedings of International Conference on Computer-Aided Design*, San Jose, November 1996.

[Kim90] K. Kim, and C. Kime, Partial Scan by Use of Empirical Testability, *Proceedings of International Conference on Computer-Aided Design***,** pp. 314-317, Santa Clara 1990.

[Kim93] Taewhan Kim, Scheduling and Allocation Problems in High-Level Synthesis, *PhD Thesis*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1993.

[Kri92] Ganesh Krishnamoorthy, and John A. Nestor, Data Path Allocation using an Extended Binding Model, *Proceedings of the Design Automation Conference,* pp. 279-284, Anaheim, June 1992.

[Kuc90] Krzysztof Kuchcinski, and Zebo Peng, Testability Analysis in a VLSI High-Level Synthesis System, *The Euromicro Journal, Microprocessing and Microprogramming,* vol. 28, nr 1-5, March 1990, pp. 295-300.

[Lai97] Kowen Lai, Christos A. Papachrisou, and Mikhail Baklashov, High Level Test Synthesis Across the Boundary of Behavioral and Structural Domains, *Proceedings of the International Conference on Computer Design,* pp. 636-641, Austin 1997.

[Lar97] Erik Larsson, and Zebo Peng, Early Prediction of Testability by Analyzing Behavioral VHDL Specifications, *Proceedings of the NORCHIP Conf.,* pp. 259-266, Tallinn, November 1997.

[Lar98] Erik Larsson, and Zebo Peng, Testability Analysis of Behavioral-Level VHDL Specifications, *Proceedings of the European Test Workshop,* pp. 143-144, Sitges, May, 1998.

[Lee90] D. H. Lee, and Sudhakar M. Reddy, On Determining Scan Flip-Flops in Partial-Scan Designs, *Proceedings of the International Conference on Computer-Aided Design,* pp. 322-325, Santa Clara 1990.

[Lee92] Tien-Chien Lee, Wayne H. Wolf, and Niraj K. Jha, Behavioral Synthesis of Easily Testable Data Path Scheduling, *Proceedings of the International Conference on Computer-Aided Design*, pp. 616-619, Santa Clara, November 1992.

[Le93a] Jaushin Lee, and Janak H. Patel, Testability Analysis Based on Structural and Behavioral Information, *Proceedings of the IEEE VLSI Test Symposium,* pp. 139-145, Atlantic City, April 1993.

[Le93b] Tien-Chien Lee, Niraj K. Jha and Wayne H. Wolf, Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and Partial Scan Environments, *Proceedings of the Design Automation Conference*, pp. 292-7, Dallas, June 1993.

[Le93c] Tien-Chien Lee, Behavioral Synthesis of Highly Testable Data Paths in VLSI digital Circuits, *PhD Dissertation,* Department of Electrical Engineering, Princeton Unversity, 1993.

[Me93a] Mentor Graphics, Autologic VHDL Synthesis Guide, *Mentor Graphics,* February, 1993.

[Me93b] Mentor Graphics, Autologic VHDL Optimizer Guide, *Mentor Graphics,* February, 1993.

[Me93c] Mentor Graphics, FlexTest User's and Reference Manual, *Mentor Graphics,* December, 1993.

[Me93d] Mentor Graphics, DFT Advisor User's and Reference Manual, *Mentor Graphics,* December, 1993.

[Muj92] Ashutosh Mujumdar, Kewal Saluja, and Rajiv Jain, Incorporating Testability Considerations in High-Level Synthesis, *International Symposium on Fault-Tolerant Computing 22*, Boston, Massachusetts, July 8-10 1992.

[Nor96] Robert B. Norwood, and Edward J. McCluskey, Synthesis-for-Scan and Scan Chain Ordering, *Proceedings of the 14th IEEE VLSI Test Symposium,* pp. 87-92, Princeton, NJ, April 28 - May 1, New Jersey 1996.

[Par93] Prashant Parikh, and Miron Abramovici, A Cost-Based Approach to Partial Scan, *Proceedings of the Design Automation Conference,* pp. 255-259, Dallas, June 1993.

[Par95] Prashant Parikh, and Miron Abramovici, Testability-Based Partial Scan Analysis, *Journal of Electronic Testing: Theory and Applications,* 7, pp. 61-70, 1995.

[Pau89] Pierre G. Paulin, and John P. Knight, Force-directed scheduling for behavioral synthesis of ASIC's, *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems,* vol. 8, June 1989, pp. 661-679.

[Pen94] Zebo Peng, and Krzysztof Kuchcinski, Automated Transformation of Algorithms into Register-Transfer Level Implementations, *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems,* vol. 13, no 2, February 1994, pp. 150-166.

[Ram94] Champaka Ramachandran, and Fadi J. Kurdahi, Incorporating the Controller Effects During Register Transfer Level Synthesis, *Proceedings of the European Design and Test Conference,* pp. 308-313, Paris 1994

[Rut72] R. A. Rutman, Fault detection test generation for sequential logic by heuristic tree search, *IEEE Computer Group Repository,* pp. 172-187, 1972.

[Syn96] Synthesia, The Synthesia VHDL Design System, *User's Guide,* 1996.

[Tho94] Thomas Thomas, Praveen Vishakantaiah and Jacob A. Abraham, Impact of Behavioral Modifications for Testability, *Proceedings of the 12th IEEE VLSI Test Symposium*, pp. 427-32, New Jersey, April 1994.

[Tri85] Howard Trickey, Compiling Pascal Programs into Silicon, *Ph.D. Disertations,* Deptartment of Computer Science, Stanford University, 1985.

[Tse83] Chia-Jeng Tseng, and Daniel P. Siewiorek, A Procedure for the Automated Synthesis of Bus Style Systems, *Proceedings of the 20th Design Automation Conference*, pp. 490-496, Maimi 1983.

[Tsu88] Frank. F. Tsui, LSI/VLSI Testability Design, *McGraw-Hill Book Company*, 1988, ISBN 0-07-100356-8.

[Var93] Kamal K. Varma, Praveen Vishakantaiah, and Jacob A. Abraham, Generation of Testable Designs from Behavioral Descriptions using High Level Synthesis Tools, *Proceedings of the VLSI Test Symposium,* pp. 124-130, Atlantic City, April 1993.

[Wag96] Kenneth D. Wagner, and Sujit Dey, High-Level Synthesis for Testability: A Survey and Perspective, *Proceedings of the Design Automation Conference,* pp. 131-136, Las Vegas, June 1996.

[Yan98a] Tianrou Yang, and Zebo Peng, An Efficient Algorithm to Integrate Scheduling and Allocation in High-Level Synthesis, *Proceedings of Design Automation and Test in Eutope (DATE)*, pp. 74-81, Paris, February 1998.

[Yan98b] Tianrou Yang, and Zebo Peng, An Improved Register-Transfer Level Functional Partitioning Approach for Testability, *Proceedings of the Euromicro Conference,* 1998, pp. 107-114, Västerås, August 1998.

Avdelning, Institution
Division, department

Department of Computer and Information Science

Institutionen för datavetenskap

Datum
Date

September 1998

Språk
Language

- [ ] Svenska/Swedish
- [x] Engelska/English
- [ ] ____________

Rapporttyp
Report: category

- [x] Licentiatavhandling
- [ ] Examensarbete
- [ ] C-uppsats
- [ ] D-uppsats
- [ ] Övrig rapport
- [ ] ____________

URL för elektronisk version

ISBN 91-7219- 305-0

ISRN LiU-Tek-Lic 1998-60

Serietitel och serienummer
Title of series, numbering

ISSN 0280-7971

Linköping Studies in Science and Technology

Thesis No 725

Titel
Title

High-Level Testability Analysis and Enhancement Techniques

Författare
Author

Erik Larsson

Sammanfattning
Abstract

Hardware testing is an important issue in the development of digital systems and much research has been devoted to it. Most of this work deals with design for testability at low abstraction levels. In this thesis we deal with the test problems at the high abstraction levels. By considering the testability at an early design stage the test problems can be efficiently solved. We propose behavioral-level testability metrics based on an analysis of the variable range, operation testability and statement reachability. The metrics can be used to detect the hard-to-test parts with a low computational cost. By experiments we show the correlation between our metrics and the fault coverage when partial scan is used. We also show by experiments, where our technique is compared with a commercial gate-level tool, that it is possible to predict the testability on the behavioral-level accurately and efficiently. We have then developed a set of testability improvement transformations applicable direct on the behavioral VHDL specification with no restrictions on the high-level synthesis and a transformation selection strategy based on our testability metrics. We have also developed a controller testability analysis technique which is based on statement reachability and an enhancement technique consisting of branch control, loop termination and register initialization methods. The controller testability analysis and enhancement technique can be used for a register-transfer level design to improve its testability when the design is finally implemented, which is demonstrated by experiments with several benchmarks.

Supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

Nyckelord
Keywords

Testability Analysis, Design for Testability, Testability Improvement, Partial Scan, Behavioral Level VHDL

Department of Computer and Information Science
Linköping University

Linköping Studies in Science and Licentiate Theses at the Faculty of Arts and Humanities

FNo 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)
No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.
No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.
No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.
No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.
No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.
No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.
No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.
No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.
No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.
No 104 **Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.
No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.
No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.
No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.
No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.
No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.
No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.
No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.
No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.
No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.
No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.
No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.
No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.
No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.
No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.
No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.
No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.
No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.
No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.
No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.
No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.
No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.
No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.
No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.
No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.
No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.
No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.
No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.
No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.
No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.
No 318 **Lena Srömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.
No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.
No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.
No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.
No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.
No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.
No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.
No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.
No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.
No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.
No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.
No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.
No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agentteoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.

No 437	**Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440	**Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94	**Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94	**Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441	**Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446	**Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450	**Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451	**Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452	**Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455	**Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94	**Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462	**Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463	**Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464	**Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469	**Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473	**Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475	**Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476	**Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. Fallstudier i tre byggföretag, 1995.
No 478	**Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95	**Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482	**Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488	**Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489	**Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497	**Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498	**Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503	**Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95	**Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95	**Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513	**Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517	**Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518	**Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522	**Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538	**Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545	**Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546	**Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96	**Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549	**Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550	**Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557	**Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558	**Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561	**Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563	**Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567	**Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575	**Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576	**Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587	**Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589	**Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591	**Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595	**Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597	**Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.
No 598	**Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599	**Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607	**Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609	**Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4	**Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6	**Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615	**Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623	**Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.

No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.