Linköping Studies in Science and Technology

Dissertation No. 660

An Integrated System-Level Design for Testability Methodology

by

Erik Larsson

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2000

ISBN 91-7219-890-7
ISSN 0345-7524

To Eva and Knut

Abstract

HARDWARE TESTING is commonly used to check whether faults exist in a digital system. Much research has been devoted to the development of advanced hardware testing techniques and methods to support design for testability (DFT). However, most existing DFT methods deal only with testability issues at low abstraction levels, while new modelling and design techniques have been developed for design at high abstraction levels due to the increasing complexity of digital systems.

The main objective of this thesis is to address test problems faced by the designer at the system level. Considering the testability issues at early design stages can reduce the test problems at lower abstraction levels and lead to the reduction of the total test cost. The objective is achieved by developing several new methods to help the designers to analyze the testability and improve it as well as to perform test scheduling and test access mechanism design.

The developed methods have been integrated into a systematic methodology for the testing of system-on-chip. The methodology consists of several efficient techniques to support test scheduling, test access mechanism design, test set selection, test parallelization and test resource placement. An optimization strategy has also been developed which minimizes test application time and test access mechanism cost, while considering constraints on tests, power consumption and test resources.

Several novel approaches to analyzing the testability of a system at behavioral level and register-transfer level have also been developed. Based on the analysis results, difficult-to-test parts of a design are identified and modified by transformations to improve testability of the whole system.

Extensive experiments, based on benchmark examples and industrial designs, have been carried out to demonstrate the usefulness and efficiency of the proposed methodology and techniques. The experimental results show clearly the advantages of considering testability in the early design stages at the system level.

Acknowledgements

IT HAS BEEN an amazingly good time working with this thesis. Many people have contributed in different ways. I am grateful for this and I would like to acknowledge the support.

I was lucky to get the opportunity to join the Embedded System Laboratory (ESLAB). My supervisor Professor Zebo Peng has got a talent for creating a good working atmosphere. For my work, he gave me valuable guidelines and hints combined with much freedom. An important combination for me.

The present and former members of ESLAB and CADLAB have created a creative and enjoyable environment to be part of. It is a joy to be among such persons. Colleagues at IDA have also given a nice atmosphere to work in and I would especially like to mention the effort made by the department to support the graduate students.

I would like to thank Dr. Xinli Gu for the early cooperation presented in Chapter 9 and several members at the Electronics Systems group, ISY, who helped me with the Mentor Graphics tool set.

The research, funded by NUTEK[1], has been carried out in close cooperation with the industry, especially with Gunnar Carlsson at CadLab Research Center, Ericsson. The cooperation and Gunnar's humble hints have provided me with many insights and a platform to demonstrate the developed techniques.

I am also happy to have the friends I have. And finally, I would like to mention my parents, Knut and Eva, and my brothers, Magnus and Bengt, who always have been the greatest support.

Erik Larsson
Linköping, November 2000

1. Swedish National Board for Industrial and Technical Development.

Contents

Preliminaries

PART I

Chapter 1
Introduction

THIS THESIS DEALS with the problems of hardware testing and focuses on problems at the early stage in the design process. Most previous work in hardware testing has mainly considered test problems at lower abstraction levels. However, the increasing 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 and design for testability tools have not kept pace and testing of complex hardware structure remains a major problem.

The main aim of hardware testing is to detect physical faults introduced during or after production. It should be distinguished from hardware verification where the aim is to detect design errors. In hardware testing a set of test vectors are applied to the system and their responses are compared with expected responses. Due to the increasing complexity of digital systems, large systems are often partitioned to allow concurrent testing of different partitions.

In this thesis an integrated framework for testing system-on-chip (SOC) including a set of algorithms is proposed. The objec-

tives are to minimize the total test application time and the test access mechanism while considering several issues. Constraints among tests and limitation on test power consumption, tester bandwidth and tester memory are considered. Further, the approach considers also the placement of test resources, test set selection and test parallelization for each block in the system.

It is also important to predict and improve testability as early as possible in the design process. In this thesis a technique to analyze testability and a transformation technique to improve it for a behavioral VHDL specification are defined. A technique to analyze the testability for a controller on register-transfer level and a technique to enhance its testability are also proposed.

The rest of this chapter is organized as follows. The motivation for the thesis is given in Section 1.1 followed by the problem formulation in Section 1.2. The contributions of the thesis are presented in Section 1.3 and finally an overview of the thesis is given in Section 1.4.

1.1 Motivation

The objective of hardware testing is to ensure fault-free electronic products and it is carried out after production and/or certain period of operation. Much work in modelling techniques and development of design tools has been performed at low abstraction levels such as the gate level. The increasing complexity of digital designs has led to the need for and the development of new modeling techniques and new design tools at higher abstraction levels. The prediction and enhancement of testability and the integration of testability at an early design stage are therefore becoming very important.

1.1.1 TEST SCHEDULING AND TEST ACCESS MECHANISM DESIGN

An effect of the increasing complexity of digital systems is increasing test application time. In order to minimize it, it is important to consider testability of a design at higher abstraction levels where the objective is to ensure that the final design is testable at a low cost.

Minimization of test application time is especially important for core-based designs. The core-based design approach is developed to handle the increasing design complexity. Cores which are developed by different design teams or purchased from different vendors, known as intellectual properties (IP) cores, are integrated usually into a single chip.

A test schedule for such a system determines the order of the tests and in order to minimize the total test time, several tests are to be scheduled concurrently. However, there may exist several types of constraint which reduces the ability for simultaneously execution of tests. Several test scheduling techniques have been proposed. However, most consider only a few issues. In order to give the designer an early overall feeling for the test problems and to allow the designer to efficiently explore the design space, it is important to consider many issues affecting the application time. Furthermore, an access mechanism for transporting test data in the system has to be designed at a minimal cost.

1.1.2 TESTABILITY ANALYSIS AND ENHANCEMENT

In order to reduce the test generation and application complexity, it is important to consider and to predict testability of a design at higher abstraction levels in order to ensure that the final design is testable at a low cost. At higher abstraction levels the functional properties of the design can be explicitly captured and it can be used to speed up testability analysis. Such information is difficult to extract from a gate-level design.

An introduction of a design-for-testability (DFT) technique in a system improves the testability but it may also introduce some degradation. It is therefore important to analyze the testability and find a trade-off between testability and design degradation. Several testability analysis approaches have been proposed. However, most are defined for low abstraction levels and those defined for higher abstraction levels, register-transfer level, usually only consider either the data path or the control part of the design.

Therefore a testability analysis technique considering the whole design at high abstraction level is needed. Furthermore, due to the fact that the feed-back loop structure is a major problem in hardware testing, the testability analysis approach must be capable of handling such structures. In order to make the testability analysis technique useful for the designer, the computational cost of the analysis technique must be reasonable.

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 such as core-based systems. This thesis fulfils the objectives by considering:

- *Test scheduling*, which is an ordering of the tests.
- *Test access mechanism design*, the design of an infrastructure to transport test data in the system.
- *Testability analysis*, where the hard-to-test parts of the system are detected.
- *Testability improvement* where the detected hard-to-test parts are modified to be easier to test.

Our main goal is to develop efficient methods 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 stages at register-transfer level and above.

The objective of reducing test application time is to be achieved by efficient test scheduling and the objective of reducing test generation time and improving fault coverage by high-level testability enhancement technique. Since, the introduction of testability improvement techniques may also degrade the design in terms of extra area and/or extra delay, the developed testability analysis technique should be able to find a good trade-off between testability and design degradation.

1.3 Contributions

The main contributions of this thesis are as follows:

- A framework for the testing of system-on-chip (SOC), which includes a set of design algorithms to deal with test scheduling, test access mechanism design, test sets selection, test parallelization, and test resource placement. The approach minimizes the test application time and the test access mechanism cost while considering constraints on tests, power consumption and test resources.
- A testability analysis technique to detect hard-to-test parts and a set of testability enhancement transformations to improve the testability and a selection strategy.

The rest of this section describes the contributions in more detail.

1.3.1 A FRAMEWORK FOR THE TESTING OF SYSTEM-ON-CHIP

In this thesis, a combined test scheduling and test access mechanism design approach is introduced. The approach minimizes the test application time while several factors are considered; these factors are: conflicts among tests, power limitations, test

resource placement, test parallelization and the minimization of the test access mechanism. Conflicts among tests include, for instance, sharing of test resources. These issues are of importance in the development of core-based system.

Experiments have been performed where the efficiency of the test scheduling technique has been shown. Its low computational cost allows it to be used for industrial designs. The test scheduling in combination with test access mechanism design has been investigated and an optimization technique is proposed. Furthermore, a technique for the placement of test resources is proposed.

Experiments have been performed to show the efficiency of the proposed approach. Regarding the test scheduling the proposed technique shows better results when comparing with other techniques in respect to test time and computational cost. The detailed experimental results could be found in [Lar99b], [Lar00a], [Lar00b], [Lar00c], [Lar00d] and [Lar00e].

1.3.2 TESTABILITY ANALYSIS AND ENHANCEMENT

A testability analysis technique that detects hard-to-test parts at a high abstraction level design representation of a system has been developed. The analysis is based on a qualitative metrics. The 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. Another advantage of early considerations of testability is that functional properties are easier to be found in a high-level design representation compared to a gate-level design.

Our testability metric is 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 correlation between our test metrics and the fault coverage. We compare our behavioral level analy-

sis with a commercial gate-level tool and show that the hard-to-test parts can be predicted accurately at the behavioral level.

We have focused on testability analysis and enhancement for the controller part of a digital design. The controller usually has a large impact on the testability of the whole design 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.

We propose a set of behavioral level 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. The work is reported in [Gu97], [Lar97], [Lar98a], [Lar98b], [Lar99a].

1.4 Thesis Overview

This thesis is divided into four parts:

- **Preliminaries**. A general background to hardware testing is described where the focus is on synthesis for testability as well as the basic terminology of testability techniques.
- **Test Scheduling and Test Access Mechanism Design**. In Part II, the background to the testing of system-on-chip (SOC) is given as well as an overview of related work. Followed by introducing the test scheduling and test access

mechanism design algorithms. An integrated framework including a set of design algorithms for testing of system-on-chip. The aim of the test scheduling is to order the tests in the system to minimize the test application time while considering several important constraints. The test access mechanism algorithm minimizes the size of the infrastructure used for transportation of test data. An integrated approach is defined where test scheduling, test access mechanism design, test parallelization and test set selection are combined. Part II concludes with several experiments on benchmarks as well as on industrial designs.

- **Testability Analysis and Testability Improvement Transformations**. Part III opens with an overview of previous approaches to analyzing the design as well as techniques to improve the testability. The behavioral level testability metrics are given in Chapter 7, including 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 8 we propose a design transformation technique and a selection strategy that improves the testability of a behavioral specification. Experimental results are presented to show that the approach makes the design testable. In Chapter 9 a technique to analyze the testability of the controller and a technique to improve the testability are proposed. 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.
- **Conclusions and Future Work**. In Part IV, the thesis concludes with conclusions and a discussion on future work.

Chapter 2
Background

TESTABILITY HAS A LARGE impact on all stages in the design flow and much research has been devoted to it. This chapter gives the background and an introduction to modelling techniques and basic definitions and techniques used for design for test (DFT) ability.

After the introduction in Section 2.1, design representations are discussed in Section 2.2. In Section 2.3 high-level synthesis is discussed and the chapter concludes with a discussion on DFT, Section 2.4.

2.1 Introduction

The development of microelectronic technology has lead to the implementation of system-on-chip (SOC), where a complete system, consisting of several application specific integrated circuits (ASIC), microprocessors, memories and other intellectual properties (IP) blocks, is implemented on a single chip.

Designing such systems usually starts with a system specification where the system's functionality is captured, see Figure 2.1. The specification is partitioned and synthesised

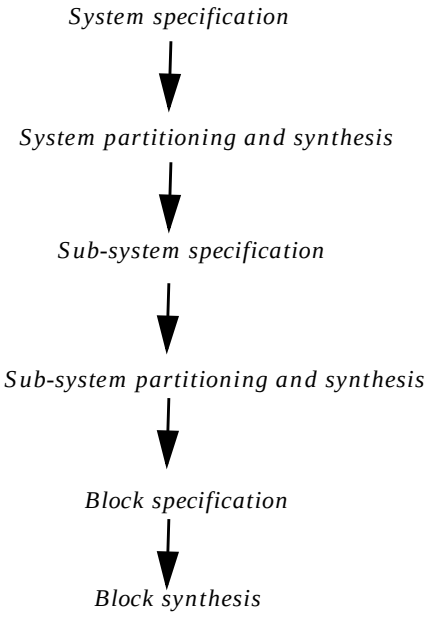


Figure 2.1: High-level design for digital systems.

(implementation specific details are added) into sub-system specifications, see Figure 2.2 for an example. The sub-systems may be further partitioned into blocks and then a design flow as in Figure 2.3 may be applied on each block.

In order to reduce the design time complete sub-systems or blocks may be reused. When sub-systems or blocks are reused some steps in the design flow in Figure 2.3 may not be needed. For instance, assuming that the microprocessor in Figure 2.2 will be given as a structural specification due to the reuse of the previously designed microprocessor, then the high-level synthesis step is not performed.

Modelling techniques at higher abstraction levels have been developed due to the increasing complexity of digital designs. In the design flow illustrated in Figure 2.3 three different abstraction levels are distinguished, behavioral, structural and gate

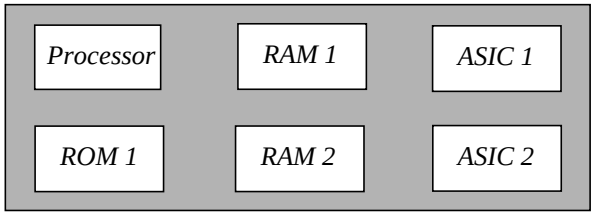


Figure 2.2: An example of a system partitioned into sub-systems.

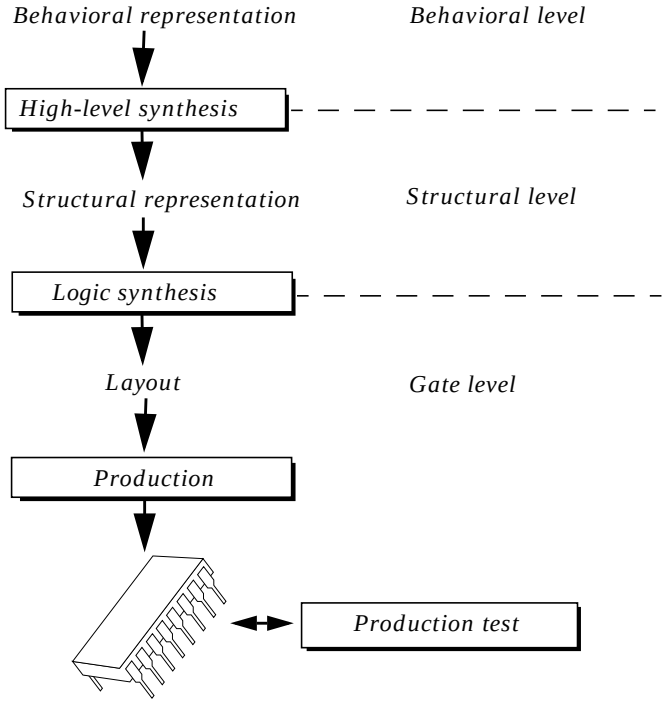


Figure 2.3: The synthesis flow for basic blocks.

level. The design work can start with a sub-system or block captured in a behavioral specification which is transformed to a structural specification by the high-level synthesis process. The logic synthesis process transforms the structural specification to a layout which is sent for production.

In order to decrease the development time it is also common to reuse previously designed parts which are incorporated as subparts in the final system. These pre-designed parts, called cores, may be incorporated at any abstraction level. For instance if a processor is incorporated, it is usually delivered as a gate-level specification by the core provider.

When the design is completed, the system is manufactured and then production tests are performed to detect production errors. Testing of the system may also be performed during the operation and maintenance of it. Hardware testing may also be used to detect design errors. However, a test for all possible errors 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

During the design process, a system or a part of it 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 fewer 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 representations. System-level modelling techniques as proposed by Cortes *et al.* [Cor00] and gate-level formats are not covered.

2.2.1 BEHAVIORAL REPRESENTATION

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

As an example, the CAMAD high-level synthesis tool, a research system developed by our research group, accepts as input a behavioral specification in VHDL [Ele92] or ADDL, Algorithmic Design Description Language [Fje92], [Pen94]. The latter was constructed especially for the CAMAD system. It is close to a subset of Pascal, with a few extensions [Fje92]. Some restrictions have been introduced in ADDL compared to full Pascal, motivated since it is to be used for hardware synthesis. Dynamic structures, files and recursion are not included in ADDL.

The extensions to Pascal are the use of *ports*, *modules* and *parallel statements*. A *port* is a connection to the external environment and a *module* is syntactically close to a procedure. However, a *module* is seen as a primitive operation mapped to a supposed hardware module. Parallel statements, enclosed by *cobegin* and *coend*, specify that the enclosed statements may execute in parallel, and synchronised at the *coend*.

2.2.2 STRUCTURAL REPRESENTATION

The structural representation, which is usually generated as the output of the high-level synthesis process, contains more implementation specific properties than the behavioral representation. From a representation 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 representation captured in VHDL typically includes component instantiations, the way that the components are connected with each other with signals and a finite state

machine describing the controller. 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 representation is generated from a behavioral representation, it is common to first transform the behavioral representation to an intermediate representation to allow efficient design space exploration of different design alternatives.

There exist several intermediate representations, 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 operations 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 by a directed graph where each vertex (node) has the possibility of multiple inputs and/or outputs, see Figure 2.4. In the figure, Petri net places (S-elements) are the circles while the transitions (T-elements) are the bars in Figure 2.4.

Initially a token is placed at S_0, which is an initial place, see Figure 2.4. 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

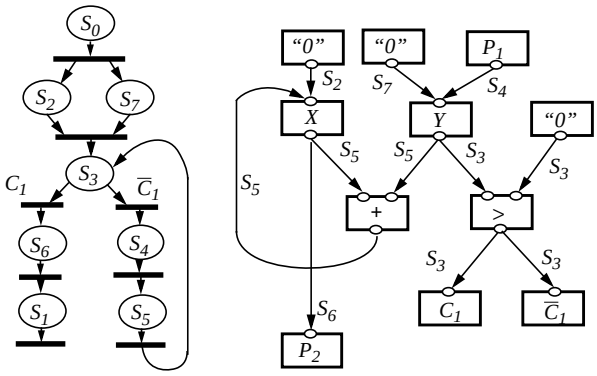


Figure 2.4: An example of ETPN.

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

2.3 High-Level Synthesis

High-level synthesis is the transformation of a behavioral representation into a structural implementation [Gaj92]. It consists mainly 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 assignment of operations to a time slots, or control step, which corresponds to a certain clock cycle. If several operations are assigned to the same control steps, several functional units are needed. This results in fewer control steps, which results in a faster design, but also leads to more expensive circuits [Gaj92].

The allocation task is to select the number and types of hardware units to be used in a design. Sharing of hardware resources reduces the design size but it is only allowed if the units are not used by different operations at the same time. Binding deals with the operations mapping to a certain module library components.

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 [Gru00] and testability, as well as other criteria such as timing constraints [Hal98].

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 that fulfils the given constraints is found.

2.4 Testing and Design for Testability

In this section testing and design for testability (DFT) are introduced. These are important for the testing of SOCs and, further, for SOCs the volume of test data (test vectors and test response) is increasing leading to high total test application time. Therefore, it is important to consider the transportation of test data and the scheduling of tests. The test application time depends on the bandwidth of the test access mechanism and how efficient the tests are ordered (scheduled).

A test access mechanism is used for the transportation of test vectors and test responses. Test vectors have to be transported from the test sources (test generators) to the blocks under test and the test responses have to be transported from the blocks under test to the test sink (test response evaluators). The size of the access mechanism depends on the placement of test resources and the bandwidth.

An efficient test schedule orders the tests in such order that the test application time is minimized.

Faults and fault models are discussed in Section 2.4.1 followed by a discussion of test generation in Section 2.4.2. Techniques for improving the testability such as test point insertion, scan, built-in self-test and test synthesis are described in Section 2.4.3.

2.4.1 FAULTS AND FAULT MODELS

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 test process itself, which is the time required to detect and/or isolate a fault.

The test cost can be reduced by using some 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 computational complexity.

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, at a point, 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 physical defects. 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:

$$f = \frac{n}{N} \tag{2.1}$$

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 TEST GENERATION

A system is tested by applying a set of test pattern (vectors/stimuli) on its primary inputs and then compare the test response on its primary outputs with know good vectors. An illustration in Figure 2.5 shows a *test control unit* which controls the *test pattern generator* and the *test response evaluator.*

Traditionally the test patterns are supplied from an external tester. However, due to the increasing capacity of the integrated circuit technology, a complete system consisting of several complex blocks can be integrated on a single chip. One of the advantages of this integration is that the performance can increase

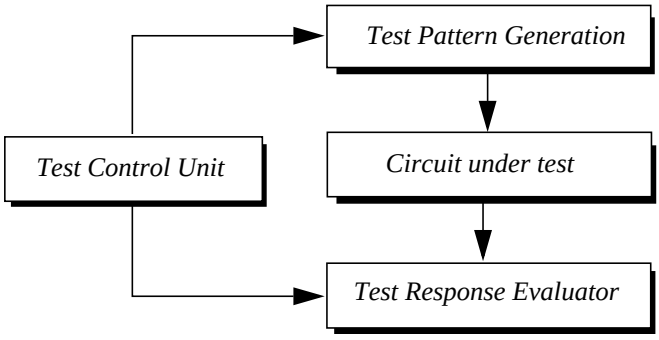


Figure 2.5: General view of a circuit under test.

mainly because there is no chip-to-chip connection which used to be a major performance bottle-neck. Due to the increasing performance of systems and the limitation of bandwidth when using external testers, there is a trend in moving the main functions of the external tester onto the chip. This would mean that all blocks in Figure 2.5 are placed on chip.

Furthermore, for large systems, it is not feasible to have only one test pattern generator and one test response evaluator as in Figure 2.5. An example of a system with several test pattern generators and test response evaluator is given in Figure 2.6.

The test generators are often of different types with their own advantages and disadvantages. For instance, TPG_1 and TPG_2 can be of different types in order to fit respectively circuit-under-test. One approach to minimizing test application time while keeping test quality high (fault coverage) is to allow a flexibility where each circuit under test is to be tested by several test sets from different test generators.

2.4.3 TESTABILITY IMPROVEMENT TECHNIQUES

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

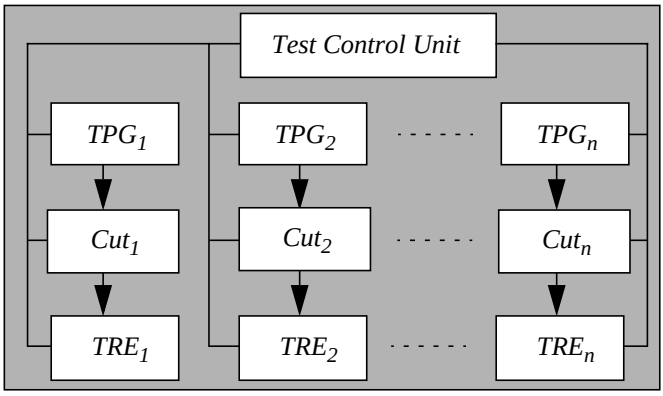


Figure 2.6: General view of a circuit under test.

Test Point Insertion

Test point insertion is a simple and straightforward approach to increasing the controllability and/or observability of a design. In Figure 2.7(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 2.7(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 2.7(c). To increase the observability of the line an extra primary output is added, Figure 2.7(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 requires extra gates and extra lines which introduce additional delay.

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 that turns a sequential circuit into a

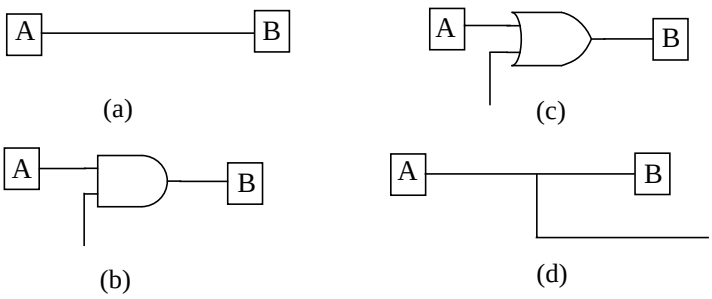


Figure 2.7: Test points for control and observation enhancement.

purely combinational one for which it is 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 the test mode the flip-flops in the design are connected to form a shift register. When the design is in the test mode, data is shifted into the design by one of the extra inputs. The circuit then runs for one clock cycle and the data captured at the flip-flops are shifted out on the added primary output.

The basic idea behind the scan technique is illustrated in Figure 2.8. Using the signal *scan selection* the register can be controlled in two modes, the normal mode or the test mode. In the 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 one or more scan chains it is called full scan. In such cases all flip-flops are *scan controllable* and *scan observable*, which turns them into *pseudo-primary inputs* and *pseudo-primary outputs*, respectively [Ste00]. The advantage is that combinational logic and the register cells in the scan chain can be completely tested. Full scan converts the problem of testing a sequential circuit into that of testing a combinational circuit.

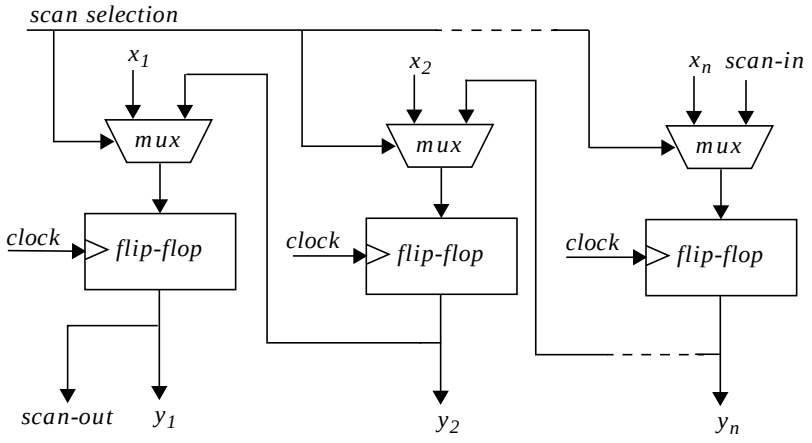


Figure 2.8: The basic idea for scan technique.

The testing of a combinational circuit is easier than the testing of a sequential one mainly since in the latter case test patterns must be applied at different states and changing from one state may require several intermediate steps. Furthermore, if a global reset is not available, an initialization sequence or state identification process is required making the problem even harder.

The overhead introduced by using the scan technique includes routing of new lines, more complex flip-flops, and three additional I/O pins. The overall clock speed may have to be reduced due to the additional logic in the flip-flops [Abr93]. The test application time may increase since a long scan chain requires many clock cycles to scan in the test vectors and scan out the test response. This can be solved by a faster scan clock or by dividing the scan chain into several shorter chains, which is called parallelization. However, these two solutions entail certain penalties. The fast scan clock needs 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 have a good trade-off between the testability of the circuit and the overhead induced by scan design.

Built-In Self-Test

When the scan technique is used, the test vectors are typically applied from the outside of the chip under test by a tester, see Figure 2.9. However, the Built-In Self-Test (BIST) technique does not require any external test equipment. Instead the test pattern generator, response analyser and test controller are integrated into the design. This may be achieved as shown in Figure 2.9 by integrating the test resources into the system which allow tests to be performed at any time since the test resources are built into the system. Another advantage of BIST is that the technique does not suffer from the bandwidth limitations which exist for external testers.

In order to further minimize test application time, the scan chains may be replaced and all registers are turned into test generations and/or test analysers. In such an approach, a new test may be applied in each clock cycle, *test-per-clock*. Compare with the scan approach where each test vector has to be scanned in, *test-per-scan*. The test pattern generator can be implemented as a *linear feed-back shift register* (LFSR) and the response analyser as a *multiple input signature register* (MISR). A built-in logic block observer (BILBO) is a register which can operate both as a test pattern generator and a signature analyser. However, the disadvantage of using BILBOs is the large area and delay penalty [Wag96].

An advantage of using the BIST technique is that tests are performed at speed. The technique also has a lower test application time compared to the scan technique.

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

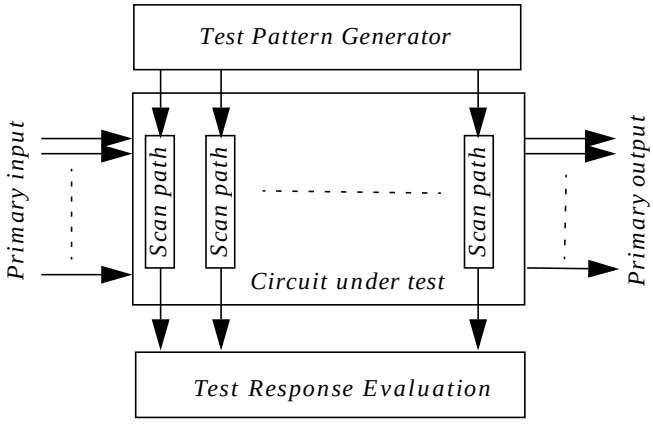


Figure 2.9: Testers for Scan Paths.

In order to minimize overhead, the BIST technique usually 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

The above DFT approaches mean usually that additional test-related hardware is added to an existing design. In test synthesis the primary goal is to perform the synthesis task in such way that the produced output achieves good testability while keeping area and performance overhead under a given constraint. The high-level synthesis tasks, scheduling, allocation and binding, are performed to achieve a testable design. However, due to the increasing complexity of digital designs, the size of the design space increases. Therefore, it is important to define efficient testability analysis algorithms which are used to guide the test synthesis. Based on the results from testability analysis the high-level synthesis can be guided to generate testable designs.

Test Scheduling and Test Access Mechanism Design

PART II

Chapter 3
Introduction and Related Work

THE SYSTEM-ON-CHIP TECHNIQUE makes it possible to integrate a complex system on a single chip. The technique introduces new possibilities but also challenges, where one major challenge is the testing of such complex system. This chapter gives an overview of research and techniques for system-on-chip testing.

3.1 Introduction

The development of microelectronic technology has lead to the implementation of system-on-chip (SOC), where a complete system is integrated on a single chip. Such a system is usually made more testable by the introduction of some design for testability (DFT) mechanisms.

Several DFT techniques such as test point insertion, scan and different types of built-in self-test (BIST) have been used for SOC testing. For complex SOC design several test techniques may have to be used at the same time since they all have their

respective advantages and disadvantages. Furthermore, when IP-blocks are used, they may already contain a test mechanism which is different from the rest of the design and it has to be incorporated in the overall test strategy of the whole system.

There are many similarities in testing PCBs (printed circuit board) and SOCs. The major difference is however twofold. For PCB, testing of each individual component can often be carried out before mounting on the board and the components can be accessed for test via probing. Neither of these is possible when testing SOCs. This means that testing the completed system, in the context of SOC, becomes even more crucial and difficult.

One main problem of testing SOCs is the long test application time due to the complex design and the need for large amount of test patterns. In order to keep test application time to a minimum, it is desirable to apply as many tests as possible concurrently. However, there are a number of factors that constrain concurrent application of several tests, which include:

- *Power consumption*,
- *Test set selection*,
- *Test resource limitations*,
- *Test resource floor-planning*,
- *Test access mechanism*, and
- *Conflicts among tests*.

In the rest of this chapter, we will analyze the implication of these factors.

3.1.1 POWER CONSUMPTION

The power consumption during test is usually higher than during the normal operation mode of a circuit due to the increased number of switches per node which is desirable in order to detect as many faults as possible in the minimum of time [Her98]. However, the high power consumption may damage the system, because it generates extensive heat.

The power dissipation in a CMOS circuit consists of a static and a dynamic part. The static power dissipation is derived from leakage current or other current drawn continuously from the power supply, and the dynamic power dissipation is due to switching transient current and charging and discharging of load capacitances [Wes92].

The static power dissipation and the dissipation due to switching transient current are negligible compared to the dissipation due to loading and unloading of capacitances, which is given by [Wes92]:

$$P_{dyn} = \frac{1}{2} \times V^2 \times C \times f \times a \tag{3.1}$$

where V is the voltage, C is the capacitance, f is the clock frequency and a is the switching activity.

All parameters but the switching activity in formula (3.1) can be estimated using a design library. The switching activity depends on the input data and there are two main approaches to estimating it, based on simulation or probability. During testing the input to the design consists of the test vectors and it is possible to make use of the test vectors generated by an ATPG tool to estimate the switch activity for a circuit under test. An approach where the test vectors are ordered based on Hamming distance has been proposed by Girard *et al.* [Gir98].

Zorian and Chou *et al.* use an additive model for estimating the power consumption [Zor93] [Cho97]. The power dissipation for a test session s_j is defined as:

$$P(s_j) = \sum_{t_i \in s_j} P(t_i) \tag{3.2}$$

where t_i is a test scheduled in test session s_j.

The power dissipation is usually considered to originate from gates. However, power may dissipate not only from blocks but also from large buses. For instance, for a wire of length 10 mm the capacitance will be about 7 pF [Eri00]. In calculation of power consumption, the average capacitance should be used,

which is close to half of the worst-case capacitance [Eri00]. Assume a system running at 100 Mhz where the average switch activity (frequency) is 25 MHz for random input data. At 2 volts the power consumption is calculated by using formula 3.1:

$$P = \frac{1}{2} \times C \times V^2 \times f \times \alpha = \frac{1}{2} \times 3.5 \times 10^{-12} \times 2^2 \times 25 \times 10^6 = 0.175mW$$

In a realistic example the width of the data bus from the memory is 512 bits which results in a power dissipation of 90 mW (512×0.175=89.6).

3.1.2 TEST RESOURCES

The test control unit controls the test resources which are either generators (sources) or analysers (sinks). The test stimuli (vectors/patterns) is created or stored at a test source and the test response is evaluated at a test sink. The test stimuli set is basically generated using the following four approaches namely:

- *exhaustive,*
- *random,*
- *pseudo-random,* and
- *deterministic.*

The basic ideas behind them and their advantages and disadvantages are outlined below.

Exhaustive-based test generation

An exhaustive test set includes all possible patterns. This is easily implemented using a counter. The area-overhead and design complexity is low and it is feasible to place such a generator on-chip. However, the approach is often not feasible since the number of possible patterns is too high: for a *n*-bit input design 2^n patterns are generated which results in extremely long test application time.

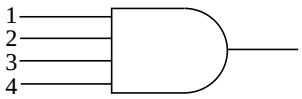


Figure 3.1: A 4-input AND-gate.

Random-based test generation

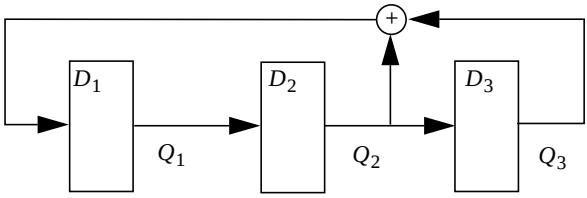
Another approach is to use the random-based techniques. The draw-back with randomly generated test patterns is that some patterns are hard to achieve. For instance, generating a test pattern that creates a one on the output of an AND gate is only achieved when all inputs are one; the probability is $1/2^n$. For a 4-bit AND-gate the probability is only 0.0625 ($1/2^4$), Figure 3.1. This means that a large set of test vectors has to be generated in order to achieve high fault coverage, which leads to long test application time.

Pseudo-random-based test generation

A pseudo-random test pattern set can be achieved using a *linear feedback shift register* (LFSR). An advantage is their reasonable design complexity and low area overhead which allow on-chip implementation. An example of an LFSR is shown in Figure 3.2 where one module-2 adder and three flip-flops are used. The sequence can be tuned by defining the feedback function to suit the block under test.

Deterministic test generation

A deterministic test vector set is created using an *automatic test pattern generator* (ATPG) tool where the structure of the circuit under test is analysed and based on this analysis, test vectors are created. The size of the test vector set is relatively small compared to other techniques, which reduces test application time. However, the generated test vector set has to be applied to



S_0	0	1	1
S_1	0	0	1
S_2	1	0	0
S_3	0	1	0
S_4	1	0	1
S_5	1	1	0
S_6	1	1	1
S_7	0	1	1

Figure 3.2: Example of 3-stage linear feedback shift register based on x^3+x+1 and generated sequence where S_0 is the initial state.

the circuit using an external tester since it is inefficient to store the test vector set in a memory on the chip. The external testers have the following limitations [Het99]:

- Scan usually operates at a maximum frequency of 50 MHz,
- Tester memory is usually very limited, and
- It can support a maximum of 8 scan chains, resulting in long test application time for large designs.

A graph with the fault coverage as a function of the number of test patterns is shown in Figure 3.3. Initially the fault coverage increases rapidly due to that faults easy to detect are detected. However, in the end few faults are detected due to the fact that the remaining faults, the random-resistant faults, are hard for an LSFT to detect. This curve applies in general to all test generation techniques. However, the faults that are hard to detect may be different for different techniques. Therefore, approaches where several test sets are generated for a block with different

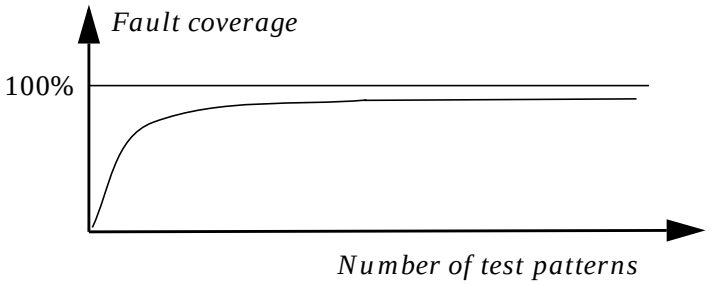


Figure 3.3: Fault coverage function of test patterns.

test resources (different techniques) in order to detect all faults in a minimum of test application time have been developed. For example, Jervan *et al.* propose a Hybrid BIST [Jer00].

3.1.3 TEST CONFLICTS

Tests may not be scheduled concurrently due to several types of conflicts. For instance, assume that the core in a wrapper is tested by two tests where one uses the external test source and test sink while the other uses the on-chip test source and test sink. These two test can not be scheduled concurrently since they both target the same logic.

3.2 Test Access Mechanism Design

A test infrastructure consists of two parts. One part for the transportation of test data and another part which controls the transportation.

In a fully BISTed system where each block in the system has its own dedicated test resources, no test data is needed to be transported. Only an infrastructure controlling the tests is required. Zorian proposes a technique for such systems [Zor93]. Håkegård's approach can also be used to synthesize a test controller for this purpose [Håk98].

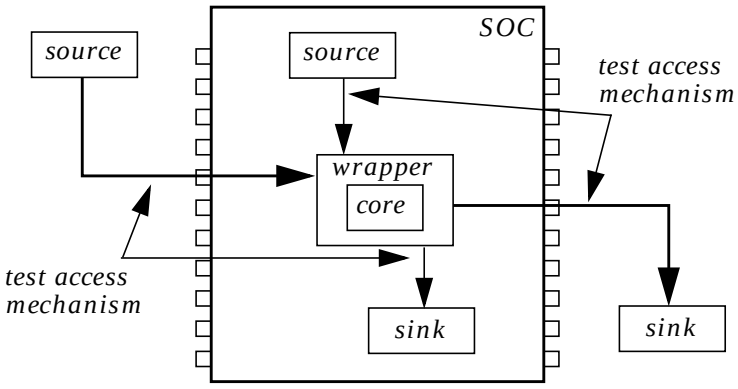


Figure 3.4: Test sources and sinks.

The test data transportation mechanism transports test data to and from the cores in the system (Figure 3.4). Due to the increasing complexity of systems, the amount of test data to be transported is becoming substantial. Research has focused on test infrastructure optimization in order to minimize the total test application time.

The test application time for *multiplexed*, *daisychain* and *distributed* scan chain architectures are investigated by Aertes *et al.* [Aer98].

In a *multiplexed architecture*, see Figure 3.5, all cores are assigned to all available scan bandwidth, *i.e.* all cores are connected to all scan inputs and all scan outputs of the system. At any moment, only one core can use the outputs due to multiplexing. The result is that the cores have to be tested in sequence.

For the discussion on multiplexed, daisychain and distributed architecture the following is assumed to be given for each core i in the system:

f_i: the number of scannable flip-flops,

p_i: the number of test patterns, and

N: the scan bandwidth for the system, maximal number of scan-chains.

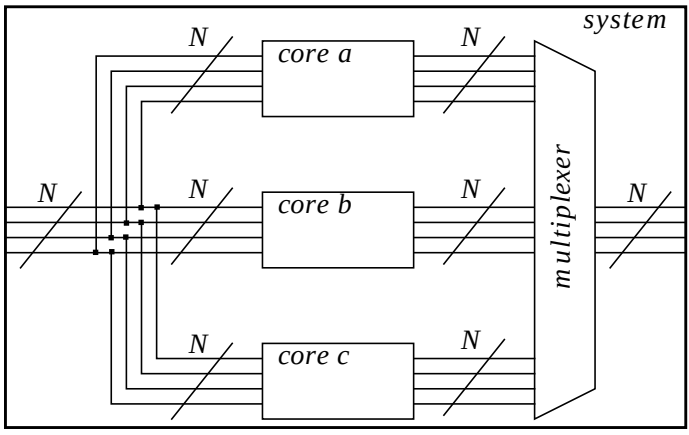


Figure 3.5: Example of the multiplexer architecture.

In scan-based systems it is common to use a pipelined approach where the test response from one pattern is scanned out, the next pattern is scanned in simultaneously. The test application time t_i for a core i is given by:

$$t_i = \left\lceil \frac{f_i}{n_i} \right\rceil \cdot (p_i + 1) + p_i \qquad (3.3)$$

In the muliplexed architecture n_i=N. The term +1 in Equation (3.3) is added due to the fact that the pipelining can not be used for scanning out the last pattern.

The pipelining approach can be used when several cores are tested in a sequence. While the first pattern is scanned in for a core, the test response from last pattern can be scanned out for the previous core under test. The test application time using the multiplexed architecture is given by:

$$T = \sum_{i \in C} \left(p_i \cdot \left\lceil \frac{f_i}{N} \right\rceil + p_i \right) + max_{i \in C} \left\lceil \frac{f_i}{N} \right\rceil \qquad (3.4)$$

where the maximum results from filling the largest core.

In the *daisychain architecture*, Figure 3.6, a bypass structure is added to shorten the access path for individual cores. The

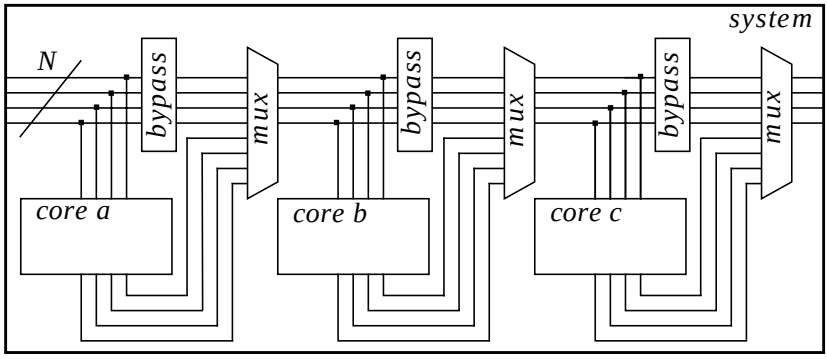


Figure 3.6: Example of the daisychain architecture.

bypass register and 2-to-1 multiplexer allow flexible access to individual cores which can be accessed using the internal scan chain of the cores and/or by using the bypass structure.

The bypass offers an optional way to access cores and a bypass selection strategy is proposed by Aertes *et al.* [Aer98], where all cores are tested simultaneously by rearranging the test vectors. The approach starts by not using any of the bypass structures and all cores are tested simultaneously. At the time when the test of a core is completed, its bypass is used for the rest of the tests. Due to the delay of the bypass registers, this approach is more efficient compared to testing all cores in sequence.

Assume the system in Figure 3.6 where p_a=10, p_b=20, p_c=30 and f_a=f_b=f_c=10. When the cores are tested in a sequence the test time of the system is 720 (10·(10+1+1)+(20·(10+1+1)+(30·(10+1+1)). Note that the terms +1+1 are due to the bypass registers. However, using the approach proposed by Aertes *et al.*, the test time for the system is reduced to 630 (10·30+10·(20+1)+10·(10+1+1)).

The test application using this scheme is given by:

$$T = \sum_{i=1}^{|C|} \left((p_i - p_{i-1}) \cdot \left(i - 1 + \sum_{j=i}^{|C|} \left\lceil \frac{f_j}{N} \right\rceil \right) \right) + p_{|C|} \qquad (3.5)$$

where p_0=-1.

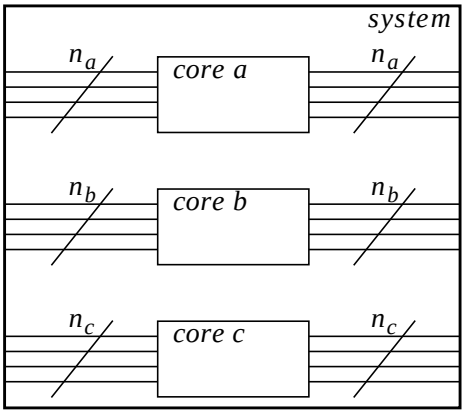


Figure 3.7: Example of the distribution architecture.

Note that the indices in Equation (3.5) are rearranged into a non-decreasing number of patterns.

In the distributed architecture each core is given a number of scan chains, Figure 3.7. The problem is to assign scan chains to each core *i* in order to minimize the test time, *i.e.* assign values to n_i where $0<n_i \leq N$.

The test application time for a core *i* in the distribution architect is given by Equation (3.3) and the total test time for the system is given by:

$$T = max_{i \in C}(t_i) \tag{3.6}$$

An algorithm is proposed to assign the bandwidth n_i for each core *i*, Figure 3.8., where the goal is to find a distribution of scan chains such that the test time of the system is minimized while all cores are accessed, expressed as:

$$min_{\bar{n} \in N^{|C|}}(max_{i \in C}(t_i)), \sum_{i \in C} n_i \leq N \wedge \forall i \in C\{n_i > 0\} \tag{3.7}$$

The algorithm presented in Figure 3.8 works as follows. Each core is assigned to one scan-chain which is required to test the system. In each iteration of the loop, the core with the highest test time is selected and another scan chain is distributed to the

```
forall i∈C
  n_i=1
  t_i=⌈f_i/n_i⌉·(p_i+1)+p_i
sort elements of C according to test time
L=N-|C|
while L≠0
  determine i* for which t_i*=max_i∈C(t_i)
  let n_i*=n_i*+1 and update t_i* accordingly
  let L=L-1
n_i gives the number of scan chains for core i
max_i∈C(t_i) gives the test time
```

Figure 3.8: Algorithm for scan chain distribution.

core which reduces its test time. The iterations are terminated when no more scan chains can be distributed.

Given an SOC and the maximum total test bus width, the distribution of test bus width to the cores in the system is investigated by Chakrabarty [Ch00a].

3.3 Test Isolation and Test Access

For SOC testing, a test access mechanism or a test infrastructure is usually added to the chip in order to facilitate test access and test isolation. Its purpose is to feed the SOC with test data. Furthermore, its design is important due to the fact that it may influence the possibility of executing test concurrently in order to minimize test application time. A test access mechanism is also needed for testing printed circuit boards (PCB).

For PCB designs the Boundary-scan test (IEEE 1149.1) standard has been defined and for SOC designs Boundary-scan (IEEE 1149.1), TestShell and P1500 may be applicable. In this section the Boundary-scan is described briefly and an overview of the TestShell approach and the P1500 proposal is given.

3.3.1 THE BOUNDARY-SCAN STANDARDS

The main objective of PCB testing is to ensure a proper mounting of components and correct interconnections between components. One way to achieve this objective is to add shift registers next to each input/output (I/O) pin of the component to ease test access.

The IEEE 1149.1 standard for the Standard Test Access Port and Boundary-scan Architecture deals primarily with the use of an on-board test bus and the protocol associated with it. It includes elements for controlling the bus, I/O ports for connecting the chip with the bus and some on-chip control logic to interface the test bus with the DFT hardware of the chip [Abr90]. In addition, the IEEE 1149.1 standard requires Boundary-scan registers on the chip.

A general form of a chip with support for 1149.1 is shown in Figure 3.9 with the basic hardware elements: *test access port* (TAP), *TAP controller*, *instruction register* (IR), and a group of *test data registers* (TDRs) [Ble93].

The TAP provides access to many of the test support functions built into a component and it consists of four inputs of which one is optional and a single output: the *test clock input* (TCK) which allows the Boundary-scan part of the component to operate synchronously and independently of the built-in system clock; the *test mode select input* (TMS) is interpreted by the TAP Controller to control the test operations; *the test data input* (TDI) feeds the instruction register or the test data registers serially with data depending on the TAP controller; *the test reset input* (TRST) is an optional input which is used to force the controller logic to the reset state independently of TCK and TMS signals; and the *test data output* (TDO). Depending on the state of the TAP controller, the contents of either the instruction register or a data register is serially shifted out on TDO.

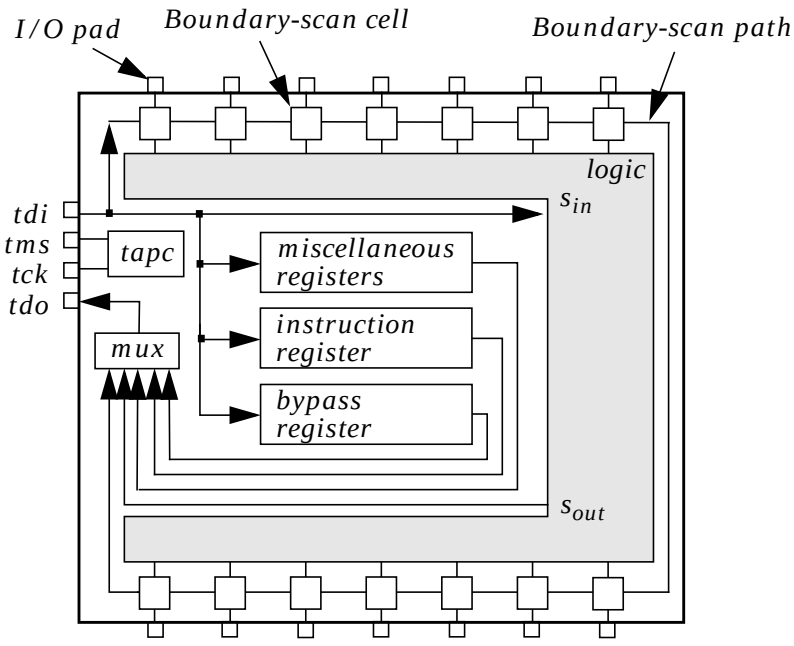


Figure 3.9: An example of chip architecture for IEEE 1149.1.

The TAP controller, named *tapc* in Figure 3.9, is a synchronous finite-state machine which generates clock and control signals for the instruction register and the test data registers.

The test instructions can be shifted into the instruction register and a set of mandatory and optional instructions are defined by the IEEE 1149.1 standard. Furthermore design-specific instructions may be added when the component is designed.

The Boundary-scan Architecture contains at a minimum two test data registers: the Bypass Register and the Boundary-scan Register. The advantage of the mandatory bypass register, implemented as a single stage shift-register, is to shorten the serial path for shifting test data from the component's TDI to its TDO [Ble93]. The Boundary-scan register of a component consists of series of Boundary-scan cells arranged to form a scan path around the core, see Figure 3.9. [Ble93].

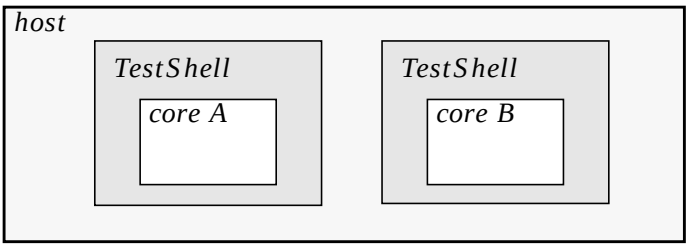


Figure 3.10: Three hierachy layers: core, TestShell and host.

3.3.2 THE TESTSHELL AND P1500 APPROACH

The TestShell is an approach to reducing the test access and test isolation problem for system-on-chip designs proposed by Marinissen *et al.* [Mar98]. Since a component to be used in a PCB is tested before mounting, while in SOC a core is to be tested after the complete chip is manufactured, a test access and test isolation method for SOC, in addition to support the test applicable by Boundary-scan, must efficiently solve the problem of testing the core themselves. It would be possible to perform component testing using Boundary-scan and the technique can be transferred to SOC. However, due to the serial access used in Boundary-scan it would lead to an excessively long test time for systems with numerous cores.

The TestShell approach consists of three layers of hierarchy, see Figure 3.10, namely:

- the *core* or the *IP module*,
- the *TestShell*, and
- the *host*.

The *core* or the *IP module* is the object to be tested and it is designed to include some DFT mechanism. No particular DFT technique is assumed by the TestShell. The *host* is the environment where the core is embedded. It can be a complete IC, or a

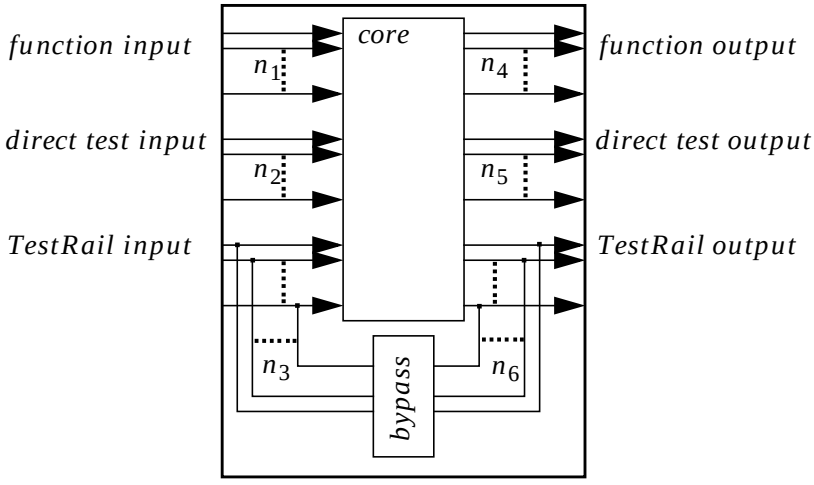


Figure 3.11: Host-TestShell interface.

design module which will be an IP module itself. Finally, the *TestShell* is the interface between the core and the host and it contains three types of input/output terminals, see Figure 3.11:

- *Function* input/output corresponds one-to-one to the normal inputs and outputs of the core.
- *TestRail* input/outputs are the test access mechanism for the TestShell with variable width and an optional bypass.
- *Direct* test input/outputs are used for signals which can not be provided through the TestRail due to their non-synchronous or non-digital nature.

The conceptual view of a TestCell is illustrated in Figure 3.12 and it has four mandatory modes:

- *Function* mode, where the TestShell is transparent and the core is in normal mode, *i.e.* not tested. It is achieved by setting the multiplexers m_1=0 and m_2=0.
- *IP Test* mode, where the core within a TestShell is tested. In this case the multiplexers should be set as: m_1=1 and m_2=0

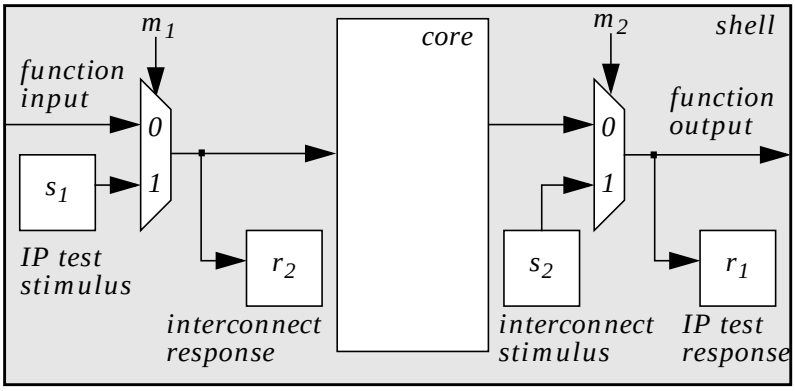


Figure 3.12: Conceptual view of the Test Cell.

where test stimulus comes from s_1 and test response is captured in r_1.

- *Interconnect Test* mode, where the interconnections between cores are tested. The multiplexers are set to m_1=0 and m_2=1 where r_2 captures the response from a function input and s_2 holds the test stimulus for a function output.
- *Bypass* mode, where test data is transported through the core regardless if the core has transparent modes. It may be used when several cores are connected serially into one TestRail, to shorten an access path to the core-under-test, see Bypass using Boundary-scan in Section 3.3.1. It is not shown in Figure 3.12. The bypass is implemented as a clocked register.

Figure 3.13 illustrates the TestShell approach where a Test Cell is attached to each functional core terminal (primary input and primary output).

TestRail

Every TestShell has a TestRail which is the test data transport mechanism used to transport test patterns and responses for synchronous digital tests.

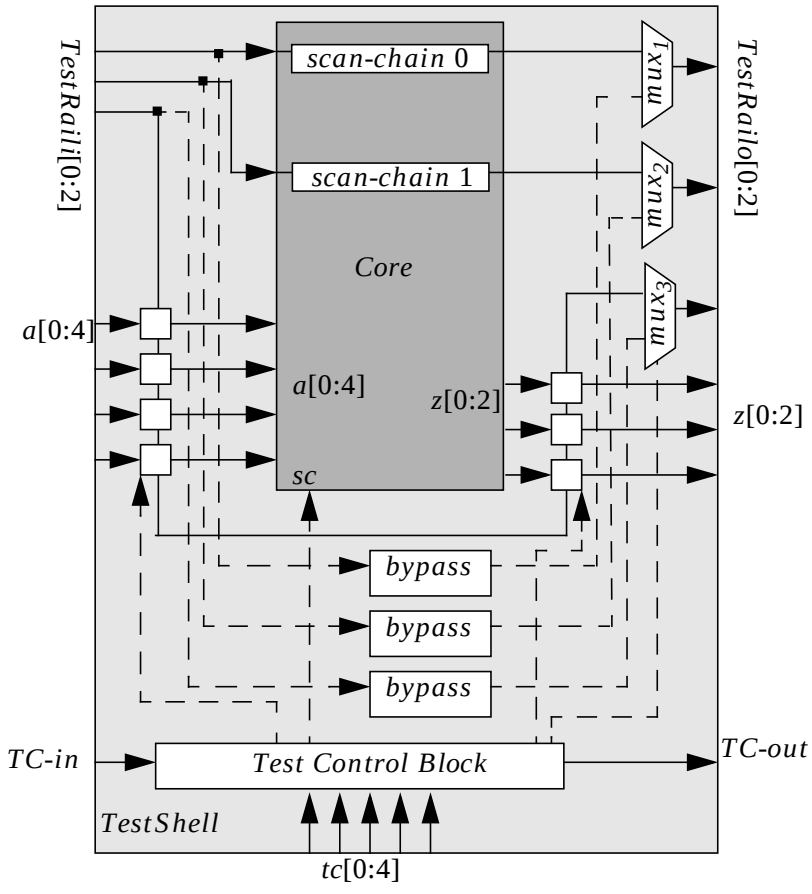


Figure 3.13: The TestShell approach.

The width n ($n \geq 0$) of the TestRail is a trade-off between the following parameters:

- *Host pins* available for test form an important limiting factor with respect to the maximal TestRail width.
- *Test time* is dependent on the test data bandwidth.
- *Silicon area* required for wiring the TestRail increases with the width of the TestRail.

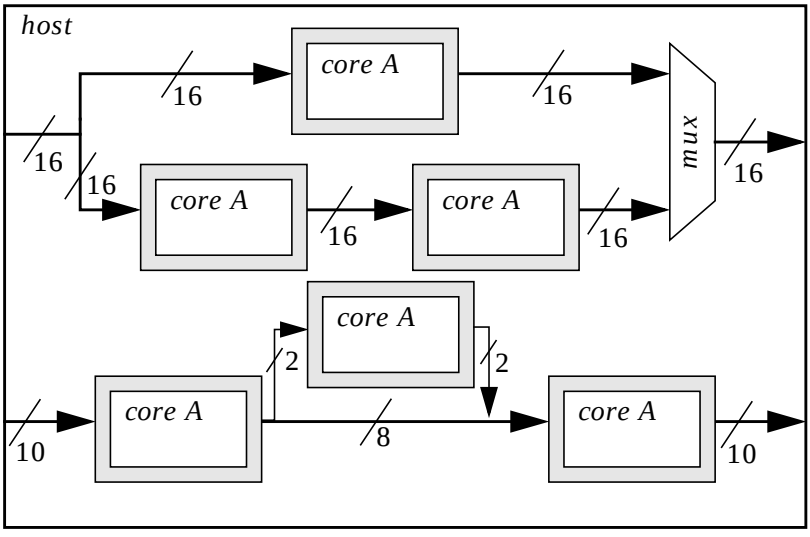


Figure 3.14: Example of possible host-level TestRail connections.

The TestRail is designed to allow flexibility; see Figure 3.14, where an example is shown to illustrate some possible connections. Within the TestShell the connections may vary. The three basic forms of connection are as follows, see Figure 3.15:

- *Parallel* connection means that the TestRail is a one-to-one connected to the terminal of the core.
- *Serial* connection means that a single TestRail wire is connected to multiple IP terminals forming a shift register, similar to Boundary-scan (Section 3.3.1).
- *Compressed* connection refers to decompression hardware at core inputs or compression hardware at core outputs.

It is also possible to use a combination of the above types of connections. The type of connection selected for a particular core depends mainly on the width of the available TestRail.

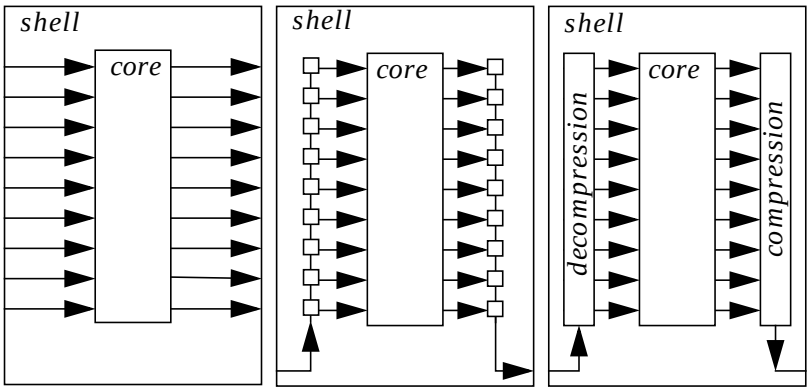


Figure 3.15: The core-level TestRail connections.

A standardized Test Control Mechanism to control the operation of the TestShell means that instructions are loaded in the Test Control Block, see Figure 3.13.

A similar approach to the TestShell is the P1500 proposal (see Figure 3.16) [P1500]. The P1500 consists of a Core Test Wrapper and a Core Test Language. The wrapper uses *Wrapper Boundary Cells* with functionality similar to the Test Cell in TestShell and the Boundary-scan Cell in the Boundary-scan approach. Instructions are loaded to the *Wrapper Instruction Register* (WIR) which is similar to the Test Control Mechanism in TestShell and the instruction register in Boundary-scan.

The differences between the TestShell wrapper and the P1500 approach is that the former allow a bypass of the test access mechanism (TAM) width while the P1500 only has a single-bit bypass, the *single-bit TAM plug* (STP). The P1500 wrapper connects to one mandatory one-bit wide TAM and zero or more scalable-width TAM, *multi-bit TAM plug* (MTP). The P1500 allows different widths of the *multi-bit TAM plugs* (MTP) input and output.

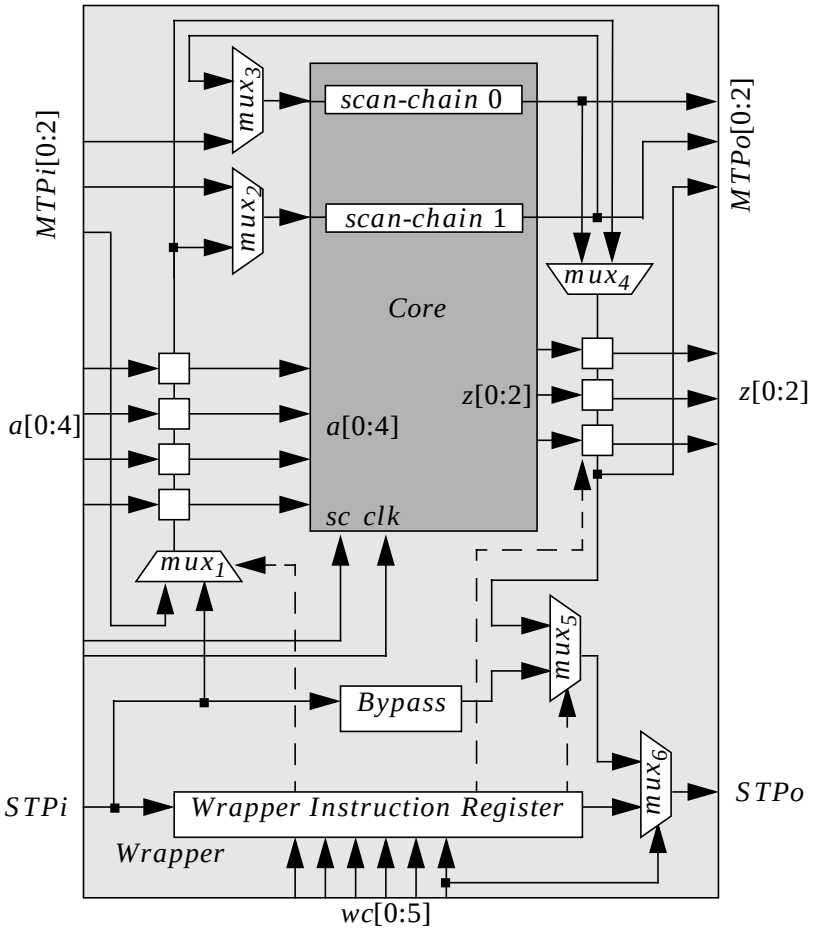


Figure 3.16: The P1500 approach.

Another wrapper approach, called TestCollar, is proposed by Varma and Bhatia [Var98]. The approach is similar to the TestShell; however, the approach does not have the bypass feature which reduces flexibility allowing only one core to be served at a time which will affect the total test application time.

Recently an approach combining P1500 and the TestShell approach has been proposed by Marinissen *et al.* [Mar00]. A

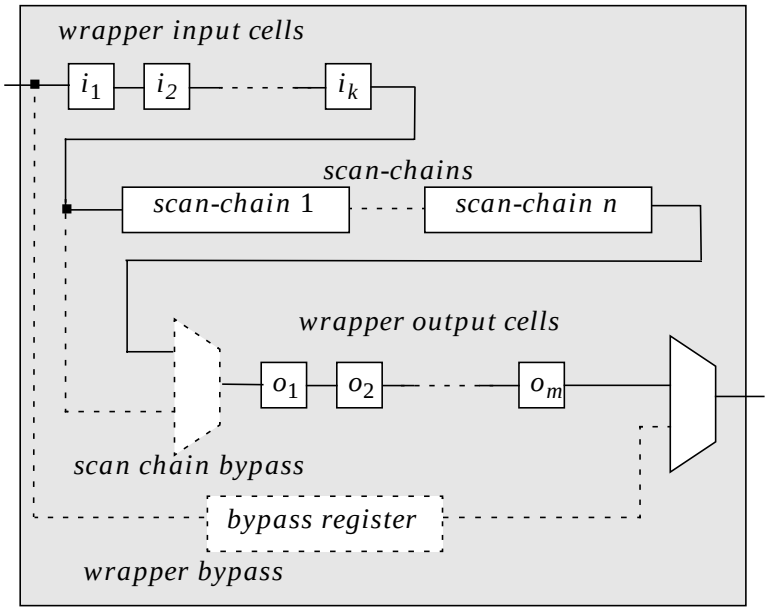


Figure 3.17: Proposed bypass structures where optional items are dashed [Mar00].

major advantage is the flexible bypass introduced, see Figure 3.17. Two types of bypass styles are defined, wrapper bypass and scan-chain bypass. The wrapper bypass is the same as used in the TestShell while the scan-chain bypass is a flexible structure which can be inserted at any place between a terminal input and a terminal output. The advantage is that it allows a non-clocked bypass structure which can be used to bypass the complete core.

The design of the core test wrapper and the test data infrastructure may affect the test application time. For instance, consider an example of a design as illustrated in Figure 3.18 where two blocks with one scan chain are to be tested. In Figure 3.18(a) a wire is added from the test generator to the scan-in of the scan-chain at core 1 and from the scan-out to the scan in of the scan-

chain at core 2. Finally, the scan output of the scan-chain at core 2 is connected to the test response evaluator.

The test time for a scan-based block is linear to $f_i \times p_i$, where f_i is the number of flip-flops and p_i is the number of test patterns in scan-chain i.

Assume the following: f_1=50, p_1=100 and f_2=25, p_2=50 for the example in Figure 3.18. By testing the blocks in sequence, block1 followed by block2, the total test time is:

$T_a=(f_1+f_2)\times p+(f_1+f_2) \times p_2+1=11251;$

where +1 refers to the fact that the last test pattern is shifted out while no new pattern is shifted in.

The approach is very inefficient and by arranging the test patterns in such way that both scan-chains 1 and 2 are seen as a single scan-chain the test application time can be reduced to:

$T_b=(f_1+f_2)\times max(p_1,p_2)+1=7501.$

A bypass structure such as the TestShell can be introduced to further minimize the test application time. By testing the blocks in sequence, *block 1* followed by *block 2*, using the bypass structure the total test time is:

$T_a=(f_1+1)\times p_1+(f_2+1)\times p_2+1=6401;$

where the +1 refers to the clocked structure used in TestShell.

Aertes *et al*. defined formulas for the bypass in the TestShell approach, see Section 3.2. Using their approach the test application time in the example in Figure 3.18 would be:

$T_d=(f_1+f_2)\times p_2+(f_1+1)\times(p_1-p_2)+1=6301;$

as given in Equation (3.5). The approach by Aertes *et al*. has thus the lowest test application time. The approach is based on a bypass structure implemented as a clocked register which affects the total test time.

However, assuming a non-clocked bypass structure or not considering the effect of a clocked bypass results in a test application time as:

$T_e=f_1\times p_1+f_2\times p_2+1=6251.$

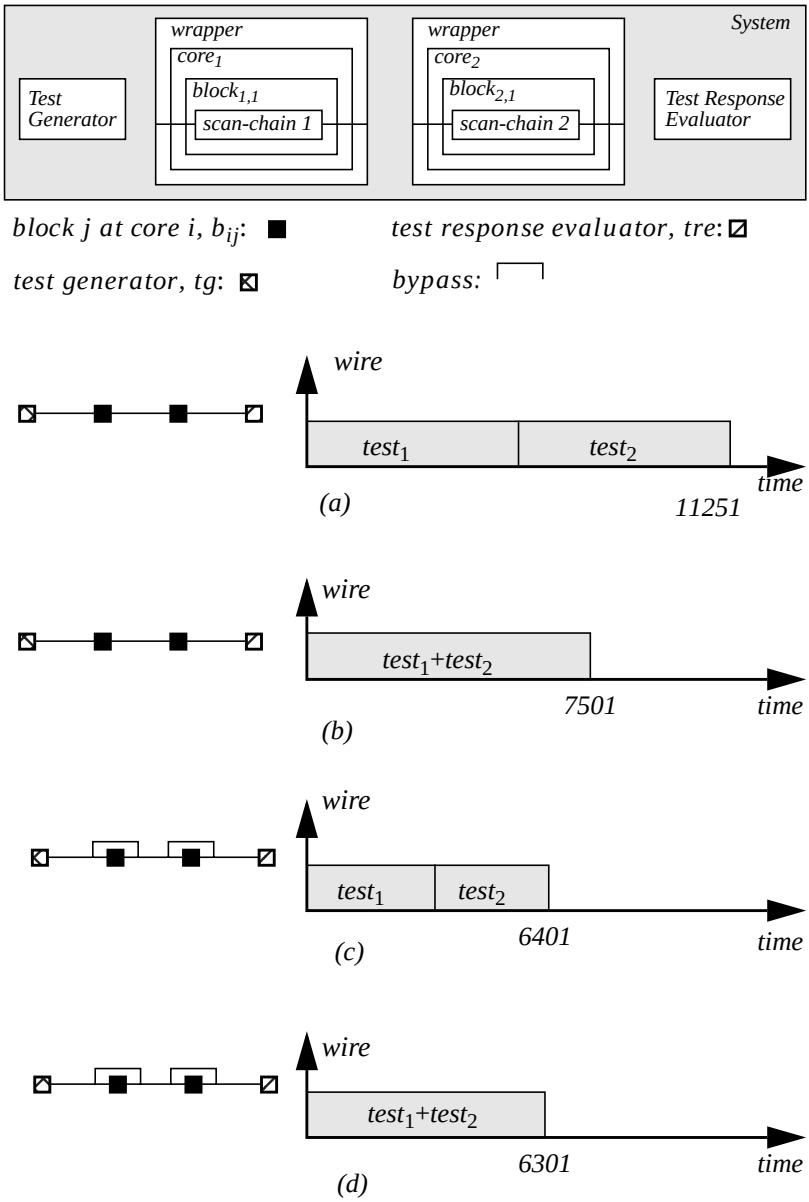


Figure 3.18: Test bus usage example.

The example above shows the importance of considering the test bus and the effect of using a clocked bypass structure.

The above approaches effectively reduce the test isolation and test access problems in printed-circuit-boards and system-on-chip. However, due to the complexity in such systems the amount of test data to be transported increases and efficient methods are required. A serial access mechanism is no longer sufficient due to the test application time it requires.

Boundary-scan was developed assuming that chips are tested to be good before mounting. By this assumption, the amount of test data to be transported is basically for interconnection test. However, for SOC where the cores are to be tested after mounting this assumption is no longer valid.

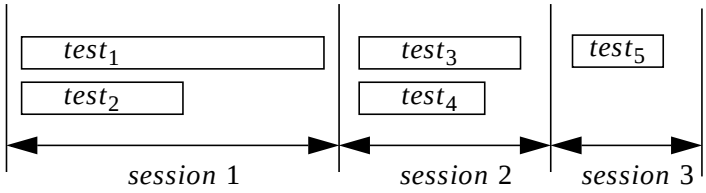
In general the number of host pins available determines the maximal bandwidth for external testers only. However, for test sources placed on-chip the number of connections are more or less unlimited [Mar98].

3.4 Test Scheduling

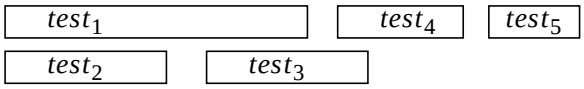
The test application time can be minimized by ordering tests in an efficient manner. Three basic scheduling strategies can be distinguished, namely [Cra88]:

- *Nonpartitioned testing*,
- *Partitioned testing with run to completion*, and
- *Partitioned testing*.

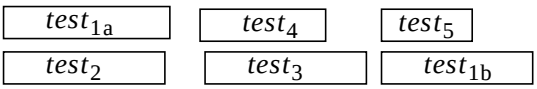
The three scheduling strategies are illustrated in Figure 3.19. In *nonpartitioned testing* no new tests are allowed to start until all tests in a session are completed. In *partitioned testing with run to completion* a test may be scheduled to start as soon as possible. For instance, $test_3$ is started before $test_1$ is completed. Finally, in *partitioned testing* the tests may be interrupted at



(a) Nonpartitioned testing



(b) Partitioned testing with run to completion



(c) Partitioned testing

Figure 3.19: Scheduling approaches.

any time. The requirement is that all tests must be completed by the end of testing. In Figure 3.19(c) $test_1$ is interrupted and runs as two segments with indexes *a* and *b*.

A test scheduling approach is proposed by Garg *et al.*, where the test time is minimized while the constraints among the tests are considered [Gar91]. A system and its tests can be modelled using a *resource graph*, see Figure 3.20, where the tests in the system are on the top level and the resources are on the bottom level. An edge between nodes at different levels indicates that a test t_i tests a resource r_j or a resource r_j is needed to perform test t_i. This means that the resource graph captures information on resource conflicts. For instance, in Figure 3.20 both test t_1 and test t_3 use resource r_1 which means that test t_1 and test t_3 can not be scheduled simultaneously.

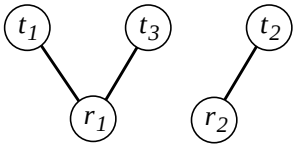


Figure 3.20: A resource graph.

Given a resource graph, a *test compatibility graph* (*TCG*) (Figure 3.22) can be obtained, where the nodes define the different test plans and the edges specify that two tests are compatible. From the test compatibility graph in Figure 3.22 it can be determined that test t_1 and t_2, for example, can be executed concurrently.

The problem of finding the minimal number of test groups such that tests within a group can be executed concurrently can be formulated as a clique partitioning problem [Gar91]. Finding the minimal clique cover on a TCG is an *non-deterministic polynomial* (*NP)* complete problem, which justifies the use of heuristics [Gar91].

Given a TCG, Garg *et al.* construct a binary tree called *time zone tree* (*TZT*) [Gar91]. Each node in the TZT represents a time zone and its constraints, *i.e.* tests associated with the zone. An illustrative example of the approach proposed by Garg *et al.* is presented in Figure 3.21. The example is based on the test compatibility graph shown in Figure 3.22 which is obtained from the resource graph illustrated in Figure 3.20.

Initially the root $R = < \varnothing, \Sigma\, l(t_i) >$ is unconstrained ($\varnothing$) and of length $7T$ ($\Sigma\, l(t_i)=4T+2T+T$). When a test t_k is assigned to R, two branches are created with two nodes, the first with the constraint t_k and length $l(t_k)$, and the second with no constraint ($\varnothing$) and length $\Sigma\, l(t_i) - l(t_k)$.

For the first test, the test with the maximum length is selected. If several such tests exist, favour is given to the test with the highest compatibility. For all other tests, the selection

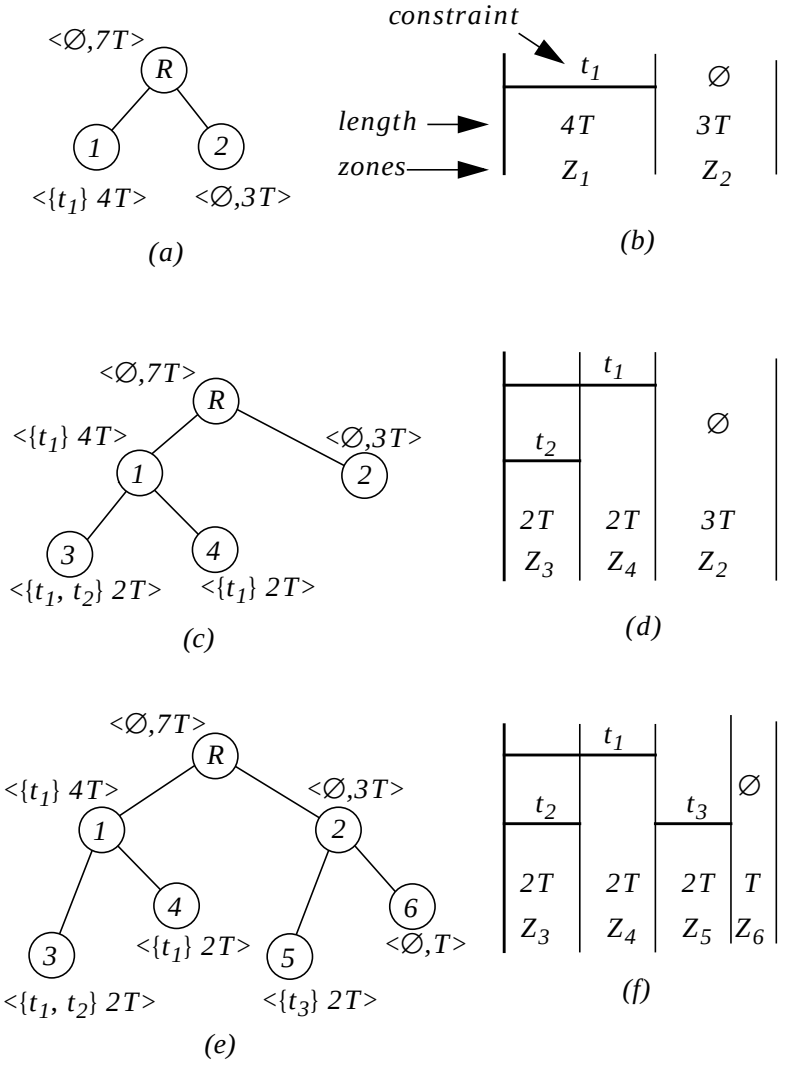


Figure 3.21: The test scheduling approach proposed by Garg *et al.* [Gar91].

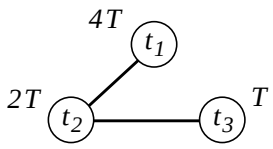


Figure 3.22: A test compatibility graph.

is based on the cost function $CF(t_i)$, where the selected test t_i has the least value according to the cost function:

$$CF(t_i) = \sum_{j=1}^{|T|} (l(t_j) - Opp(t_j/t_i)) \tag{3.8}$$

where:

$$Opp(t_j/t_i) = \begin{cases} l(Z_k), \text{ if } t_j \text{ is compatible with } t_i \\ l(Z_k), \text{ if } t_j \text{ is not compatible with } t_i \text{ and } l(Z_k) > l(Z_k) \\ 0, \text{ otherwise.} \end{cases}$$

In the example, given in Figure 3.21, t_1 is selected first and when appended to the tree, two branches (or zones) Z_1 and Z_2 are created, see Figure 3.21(a), (b). Next when t_2 is assigned to zone Z_1, node 3 is appended to the tree with constraints and length as shown in Figure 3.21 (c). Node 4 is also created at this time, denoting that Z_4 is of length $2T$ and constrained by t_1 only. Finally, test t_3 is assigned, resulting in the TZT shown in Figure 3.21(e) and the corresponding diagram is in Figure 3.21(f). The scheduling diagram is directly derived by an inspection of the leafs of the TZT from left to right. And the worst case computational cost of the approach is of the order $O(n^3)$ [Gar91].

Chakrabarty proposes a test scheduling algorithm where test time is minimized while test constraints are considered. First Chakrabarty shows that the test scheduling problem is equivalent to open-shop scheduling [Gon76]. Then a test scheduling algorithm is proposed, see Figure 3.23 [Ch00b]. In the approach tests are scheduled as soon as possible. If a conflict among two tests occurs, the test with the shortest test time is scheduled

```
Procedure SHORTEST_TASK_FIRST({t_i})
begin
for i:= 1 to m do /* there are m tasks */
  start_time_i := 0;
while flag = 1 do begin
  flag = 0;
  for i:= 1 to m do
    for j := i + 1 to m do
      if x_ij=1 then
        /* x_ij=1 if i and j are conflicting */
        if OVERLAP(i,j) then begin
          if start_time_i+l_i>start_time_j+l_j then
            start_time_i+l_i:=start_time_j+l_j
          else
            start_time_i+l_i:=start_time_j+l_j;
          flag := 1;
        end;
      end;
end;
```

Figure 3.23: The shortest-task-first procedure [Ch00b].

first. The algorithm in Figure 3.23 has a worst case execution time of $O(n^3)$ for n tests.

Other test scheduling approaches where test time is minimized while considering test conflicts are proposed by Kime and Saluja [Kim82], Craig *et al.* [Cra88] and Jone *et al.* [Jon89].

An approach where the test application time is minimized while constraints on power consumption are considered is proposed by Zorian [Zor93]. The tests in the system are partitioned in such a way that the tests in a partition can be executed concurrently and the power dissipation within each partition is below the maximal allowed power dissipation. The partitioning is guided by the placement of the blocks in the system. Tests at blocks which are physically close to each others are placed in the same partition. This approach to partitioning minimizes the

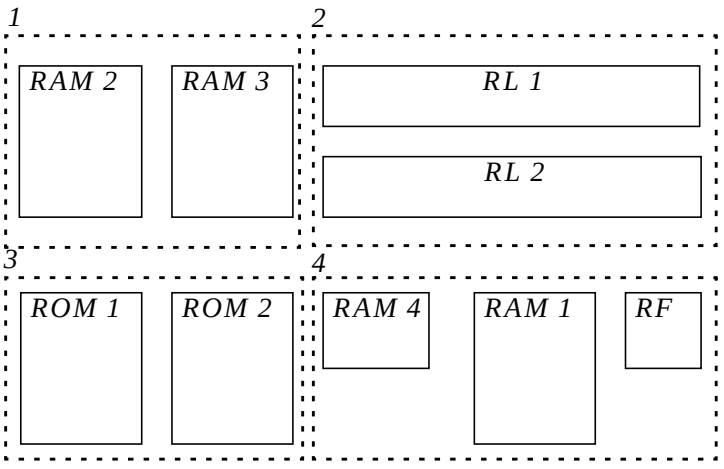


Figure 3.24: ASIC Z floor-plan and test partitioning.

amount of control lines added for controlling the tests of the system since the same control line is used for a complete partition.

The system ASIC Z is used to illustrate the approach by Zorian, see Figure 3.24, where the design is partitioned into four partitions, marked with numbers 1 to 4. Table 3.1 gives the design data for this example and the test schedule for ASIC Z is shown in Figure 3.25.

Another approach to test scheduling, where test application time is minimized while constraints among tests and test power consumption are considered, is proposed by Chou *et al.* [Cho97]. This approach works on a TCG with added power constraints and test length information constructed from a resource graph (Figure 3.26).

In order to minimize the complexity of the test controller, the tests are assigned to test sessions and no new tests are started until all tests in a session are completed.

The power dissipation for a test session s_j is given by:

$$P(s_j) = \sum_{t_i \in s_j} P(t_i) \tag{3.9}$$

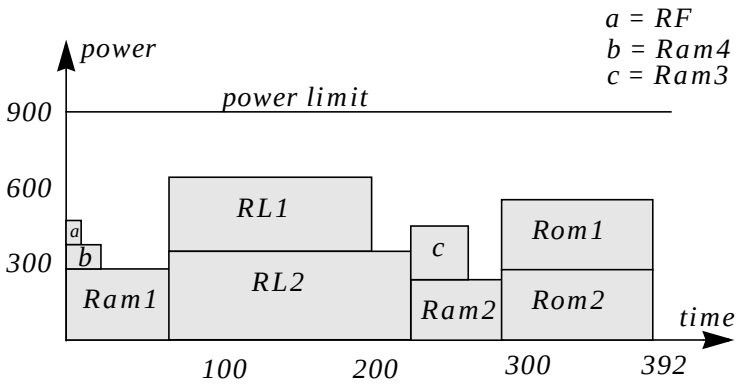


Figure 3.25: ASIC Z test schedule using the approach proposed by Zorian [Zor93].

The power constraint is defined as:

$$P(s_j) \leq P_{max} \qquad \forall j \tag{3.10}$$

From the TCG a *power compatible set* (PCS) is derived where the tests in each set (clique) are time compatible with each other and satisfy the power constraints. For instance PCS=$\{t_4,t_3,t_1\}$ in such a set, as illustrated in Figure 3.26.

Block	Test Time	Idle Power	Test Power
RL1	134	0	295
RL2	160	0	352
RF	10	19	95
RAM1	69	20	282
RAM2	61	17	241
RAM3	38	11	213
RAM4	23	7	96
ROM1	102	23	279
ROM2	102	23	279

Table 3.1: ASIC Z characteristics.

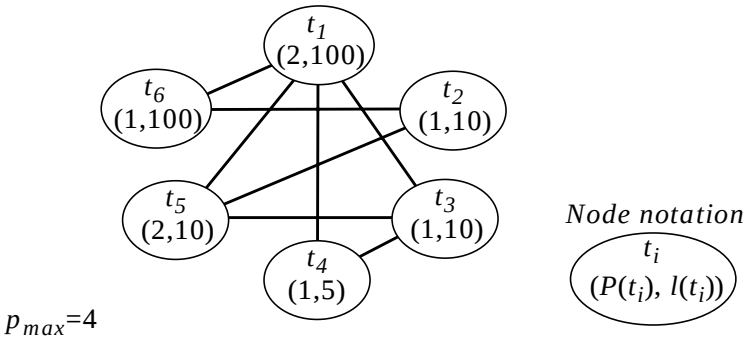


Figure 3.26: TCG with added power constraint and test length for each test.

A *power compatible list* (PCL) H is a PCS such that the elements in H are arranged in descending order of length. For instance, the PCL for PCS=$\{t_4, t_3, t_1\}$ is $H=\{t_1, t_3, t_4\}$ since $l(t_1)\geq l(t_1)\geq l(t_1)$.

A *derived* PCL (DPCL) is an ordered subset of a PCL or DPCL such that the test length of the first element is strictly less than the test length of the first element in the original PCL. For instance the DPCLs of the PCL $H=\{t_1, t_3, t_4\}$ are $H'=\{t_3, t_4\}$ and $H''=\{t_4\}$.

A *reduced* DPCL (RDPCL) set is the set of all DPCLs derivable from all possible PCLs such that each DPCL appears only once. Furthermore, if DPCL $h_1=(t_1, t_2,...,t_m)$ and DPCL $h_2=(t_{i1}, t_{i2},..., t_{ik})$ such that $t_{ij} \in h_1$, j=1, 2,..., k and $l(h_1)=l(h_2)$, then h_2 is removed from the TDPCL set.

Given a TCG, as shown in Figure 3.26, the steps in the approach by Chou *et al.* are as follows.

1. All possible cliques are identified: $G_1=\{t_1, t_3, t_5\}$, $G_2=\{t_1, t_3, t_4\}$, $G_3=\{t_1, t_6\}$, $G_4=\{t_2, t_5\}$, $G_5=\{t_2, t_6\}$.
2. All possible PCLs are: (t_1, t_3), (t_1, t_5), (t_3, t_5) obtained from G_1, (t_1, t_3, t_4) from G_2, (t_1, t_6) from G_3, (t_2, t_5) from G_4 and finally (t_2, t_6) from G_5.

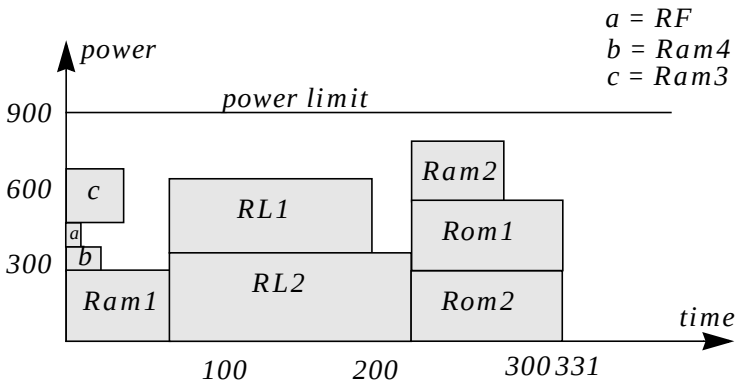


Figure 3.27: ASIC Z schedule using the approach proposed by Chou *et al.* [Cho97].

3. The reduced DPCLs are: (t_1, t_5), (t_5), (t_3, t_5), (t_1, t_3, t_4), (t_3, t_4), (t_4), (t_1, t_6), (t_2, t_5), (t_2, t_6).
4. Using a minimum cover table, see Table 3.2, to find an optimum schedule over the compatible tests, the test schedule is: (t_3, t_4), (t_2, t_5), (t_1, t_6) with a total test time of 120.

The test schedule achieved on the ASIC Z system by the approach proposed by Chou *et al.* is shown in Figure 3.27. The total test application time is 331; the approach proposed by Zorian needs 392 time units, see Figure 3.25.

The identification of all cliques in the TCG graph is an NP-complete problem and therefore a greedy approach such as proposed by Muresan *et al.* is justified where test time is minimized while test constraints and power consumption are considered [Mur00].

A basic assumption in the approaches by Chou *et al.* [Cho97] and by Zorian [Zor93] is that no new tests are started until all tests in a test session are all completed. Due to this assumption the test controller is minimized. However, this assumption is not valid in the approach proposed by Muresan *et al.* [Mur00].

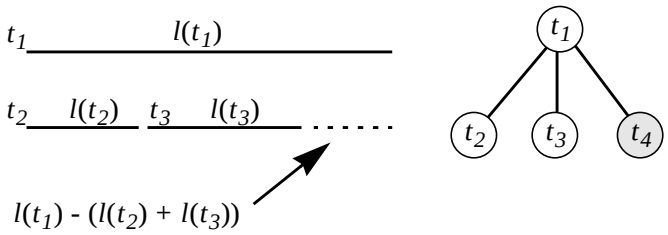


Figure 3.28: Merging example by Muresan *et al*. [Mur00].

Muresan *et al.* define an extension called the *expanded compatibility tree* (ECT) of the compatibility tree introduced by Jone *et al.*, where the number of children are generalized. For instance, assume tests t_1, t_2, t_3 and t_4, where t_2, t_3 and t_4 are compatible with t_1, see Figure 3.28. However, t_2, t_3 and t_4 are not compatible with each other. Assume that the test length $l(t_2)+l(t_3)<l(t_1)$ and t_4 is to be scheduled. If $l(t_4)\leq l(t_1)-(l(t_2)+l(t_3)$ then t_4 can be inserted in the ECT.

Neither the approach by Chou *et al.* [Cho97] nor that by Muresan *et al.* [Mur00] consider the routing of control lines which was considered by Zorian [Zor93] by partitioning the tests due to their physical placement in the system.

RDPCL	t_1	t_2	t_3	t_4	t_5	t_6	Cost
(t_1, t_3, t_4)	x		x	x			100
(t_1, t_5)	x				x		100
(t_1, t_6)	x					x	100
(t_2, t_6)		x				x	100
(t_3, t_5)			x		x		10
(t_2, t_5)		x			x		10
(t_3, t_4)			x	x			10
(t_5)					x		10
(t_4)				x			5

Table 3.2: Covering table.

An approach to handling the test complexity is proposed by Håkegård where a hierarchical approach for synthesising test controllers is defined [Håk98].

3.5 Test Set Selection

A test set is used to test a system or a part of it. A complex system may be tested by several test sets in order to achieve a sufficient fault coverage and the selection of the test sets for the cores in the system affects the total test application time. For instance, assume a system consisting of 4 cores as in Figure 3.29, where each core is tested by a BIST and an external tester and the external tester can only test one core a the time. For each core it is possible to determine several test sets with sufficient fault coverage where the test sets differ in test time ratio between BIST test time and external test time. In Figure 3.29 two solutions for testing the cores are shown where in Figure 3.29(a) the total test time is much higher than that in Figure 3.29(b) due to the use of different of test sets.

Sugihara *et al.* propose a technique for test set selection, where each core is tested by a test set consisting of two parts, one based on BIST and another based on external testing [Sug98].

For each core i a set of test sets is defined, $v_i \in V_i$. Each test set v_i consists of a BIST part and a part using an external tester. $BC(v_i)$ is the number of BIST clock cycles for test set v_i, and $ETC(v_i)$ is the number of clock cycles using the external tester.

The total time used for external testing T_{ET} is given by:

$$T_{ET} = \sum_{i=0}^{n-1} \frac{ETC(v_i)}{FT} \tag{3.11}$$

where FT is the frequency of the external tester.

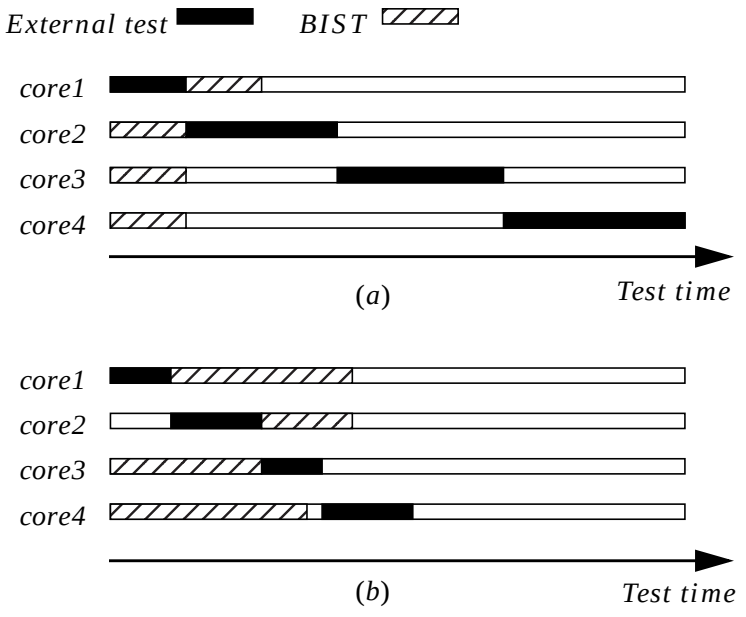


Figure 3.29: Example of test time.

The total time used for external testing T_{ET} is given by:

$$T_{v_i} = \frac{BC(v_i)}{F} + \frac{ETC(v_i)}{FT} \tag{3.12}$$

where F is the system frequency used at BIST.

The total test application time, T, for the system is given by:

$$T = max\left\{T_{ET}, max_{i=0}^{n-1}\{T_{v_i}\}\right\} \tag{3.13}$$

The main problem is to determine the test set v_i for each core i. Chakrabarty proposes a *mixed-integer linear programming* model for the test set selection problem where the BIST structure may be shared among cores [Cha99].

Chapter 4
Test Scheduling and Test Access Mechanism Design

A test schedule determines the order of the tests for a system under test while a test infrastructure transports and controls test data in the system. In this chapter, an approach is proposed to integrate test scheduling, test access mechanism design, test set selection, test resource floor-planning and test parallelization. The approach considers test conflicts and limitations on test power, tester bandwidth and tester memory.

4.1 Introduction

A traditional DFT flow starts with a system specification, see Figure 4.1. An architecture is created based on the characteristics of the system specification. It is made testable by the introduction of DFT mechanism into the design. Test sets are then selected and the test resources are placed in the system. Finally, the tests are scheduled and a test access mechanism is designed for the transportation of test data.

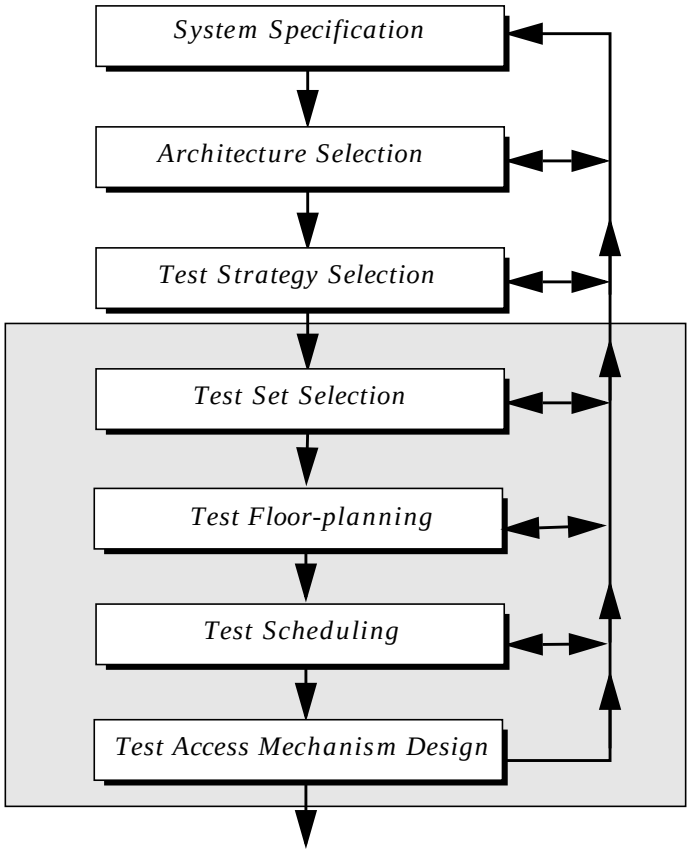


Figure 4.1: System test design flow.

If the system being designed, at any step in the DFT flow, shows unacceptable results, some design steps have to be repeated, which is illustrated with arrows going backwards in Figure 4.1. These design iterations could be repeated numerous times and it is therefore important that the algorithms used in the design space exploration process in the different design steps have a low computational cost.

On the other hand, before the final design is generated, a more extensive optimization can be allowed to take longer time, *i.e.* to have a higher computational cost.

The rest of this chapter describes techniques for test set selection, test scheduling, test floor-planning, test parallelization and test bus design. These design steps are traditionally considered as four separate distinct steps. However, they are highly interdependent on each other and it is important to consider them in a combined manner in order to produce an efficient solution.

4.2 System Modelling

A SOC example is illustrated in Figure 4.2, where each core is placed in a wrapper in order to achieve efficient test isolation. Each core consists of at least one block with an added DFT technique and in this example, all blocks are tested using the scan technique. The *test access port* (tap) is the connection to an external tester and the on-chip test resources, *test generator* 1, *test generator* 2, *response evaluator* 1 and *response evaluator* 2 are integrated into the system to support BIST.

The system in Figure 4.2 can be viewed as in Figure 4.3 and modelled as a *system with test*, ST=(C, R_{source}, R_{sink}, $p_{\max}$, T, *source*, *sink*, *core*, *block*, *constraint*, *memory*, *bandwidth*) where:
$C=\{c_1, c_2,..., c_n\}$ is a finite set of cores where each core consists of a finite set of blocks, $c_i=\{b_{i,1}, b_{i,2},..., b_{i,m}\}$. Each core consists of at least one block and each block $b_{i,j} \in B$ is given by:

$p_{idle}(b_{i,j})$: idle power,
$par_{min}(b_{i,j})$: minimal parallelization degree, and
$par_{max}(b_{i,j})$: maximal parallelization degree;

$R_{source}=\{r_1, r_2,..., r_p\}$ is a finite set of test sources;
$R_{sink}=\{r_1, r_2,..., r_q\}$ is a finite set of test sinks;
$p_{\max}$: maximal allowed power at any time;
$T=\{BT_{1,1,1}, BT_{1,1,2},..., BT_{n,m,k}\}$ is a finite set of *block tests* (BT) where $BT_{i,j,k}=\{t_1, t_2, ..., t_l\}$ be the k:th set of test sets, where each of the k sets are sufficient for efficient test of block $b_{i,j}$ at core c_i. Efficient in respect to the test set is determined by the designer. Each test t_l is a set of test vectors for test of a block produced by

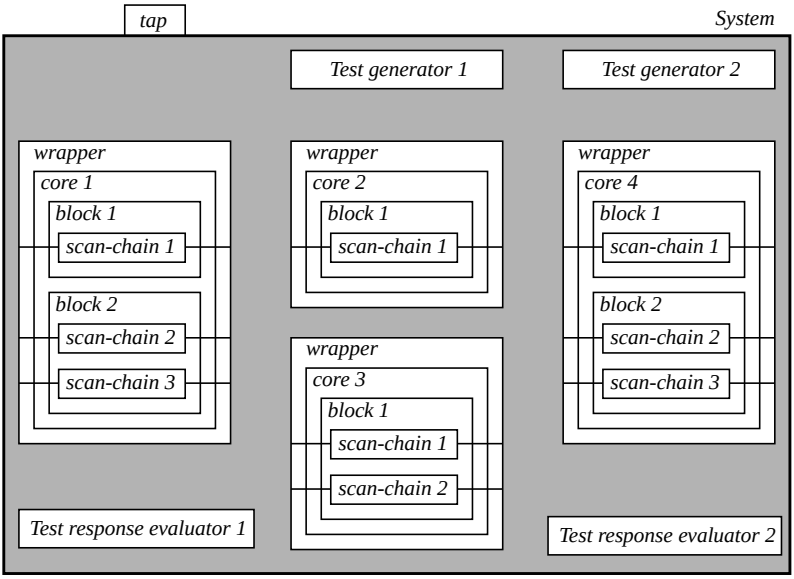


Figure 4.2: An illustrative example.

one test generator and analyzed by one response evaluator. The test resources are defined as pairs, a source with its corresponding sink.

Each test $t_i \in T$ is given by:

$t_{test}(t_i)$: test time at parallelization degree 1, $par(t_i) = 1$,
$p_{test}(t_i)$: test power parallelization degree 1, $par(t_i) = 1$,
$t_{memory}(t_i)$: memory required for test pattern storage.

$source$: $T \rightarrow R_{source}$ defines the test source for a test;
$sink$: $T \rightarrow R_{sink}$ defines the test sink for a test;
$core$: $B \rightarrow C$ gives the core where a block is placed;
$block$: $T \rightarrow B$ gives the block where a test is placed;
$constraint$: $T \rightarrow B$ defines the set of blocks required for a test;
$memory(r_i)$: memory available at test source $r_i \in R_{source}$;
$bandwidth(r_i)$: bandwidth capability at test source $r_i \in R_{source}$.

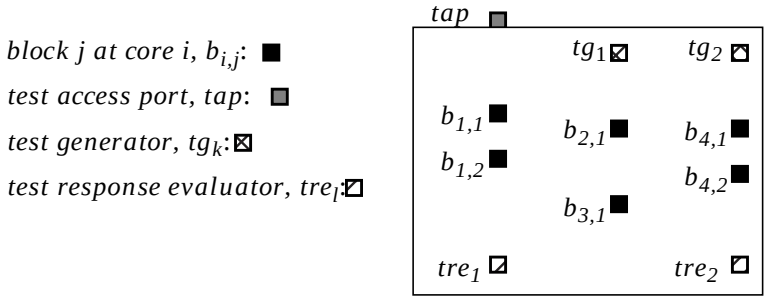


Figure 4.3: A graph representation of the system illustrated in Figure 4.2.

The system is tested by applying a set of tests where each test set get its test vectors from a test source determined by the function *source* and the test response is evaluated at a test sink given by the function *sink*.

The test time, t_{test}, test power consumption, p_{test}, and memory requirement, t_{memory}, are given for each of the tests in the system. The maximal and minimal degree of parallelization for a block is given by par_{max} and par_{min}. For instance, assume that $par_{max}(b_{3,1})=2$ and $par_{min}(b_{3,1})=1$ for block 1 at core 3 in Figure 4.2. Selecting parallelization degree to be 1 means that all scannable flip-flops are connected in a single scan-chain. On the other hand if it is selected to be 2, there will be two scan-chains at the block which reduces the test time.

4.3 Test Scheduling

In the approach proposed by Zorian [Zor93], see Section 3.4, a test schedule is created where test application time is minimized under power consumption constraint. Furthermore, new tests are only allowed to start when all tests in a session are completed and tests are grouped based on their physical placement.

In the approach by Chou *et al.* [Cho97] and by Muresan *et al.* [Mur00] focus is on test time minimization and the routing of test control lines are not considered. Therefore grouping based on physical placement, which minimizes the routing of test control lines, is not considered. Furthermore, regarding the approach proposed by Muresan *et al.* [Mur00] the minimization of the test controller is not considered and tests are allowed to start even if all tests in a session are not completed which minimizes test application time further.

For the approach described in this thesis it is optional if tests should be allowed to start even if not all tests at the moment are completed. A reason for not allowing tests to start when other tests are running is that it minimizes the complexity of the test controller. However, the test application time increases in such an approach. Furthermore, the approach described in this thesis does not consider grouping of tests based on the physical placement of corresponding blocks. Such grouping is motivated since it reduces the amount of extra control lines. All tests within a group placed physically close can be controlled by the same control line. However, the routing of the test data transportation mechanism is considered in this thesis.

The basic difference between the test scheduling technique proposed in this thesis and the approaches introduced by Zorian [Zor93], Chou *et al.* [Cho97] and Muresan *et al.* [Mur00] is illustrated for a small system with four tests, see Figure 4.4. In the approach proposed by Zorian [Zor93] and Chou *et al.* [Cho97] $test_3$ and $test_4$ would not be allowed to be scheduled as in Figure 4.4 due to the fact that new tests are only allowed to start when all tests in a session are completed. In the approach proposed by Muresan *et al.* [Mur00] $test_3$ can be scheduled as in Figure 4.4 if it is completed no later than $test_1$ is completed. This means that $test_4$ can not be scheduled as in Figure 4.4. For the approach proposed in this thesis, if tests are allowed to start even if all tests are not completed, it is possible to schedule $test_4$ as in Figure 4.4. In this way, more flexibility is achieved and it is

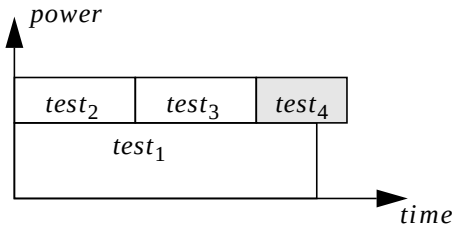


Figure 4.4: Example of test scheduling.

possible to explore the trade-off between test application time and test controller complexity

The tests for an SOC have to be scheduled where the start time, end time and bus for each test has to be determined.

Let a schedule S be an ordered set of tests such that:

$$\{S(t_i) < S(t_j) \,|\, t_{start}(t_i) \leq t_{start}(t_j), i < j, \forall t_i \in S, \forall t_j \in S\} \qquad (4.1)$$

where $S(t_i)$ gives the position of test t_i in S and $S(t_i)<S(t_j)$ means that $S(t_i)$ is placed before $S(t_j)$.

For each test, t_i, the start time and the bus (if the test access mechanism is to be considered) have to be determined before inserted in the schedule, S. The function $t_{start}(t_i)$ gives the time when test t_i is scheduled to start and the function $t_{end}(t_i)$ gives the time when test t_i ends:

$$t_{end}(t_i) = t_{start}(t_i) + t_{test}(t_i) \qquad (4.2)$$

The Boolean function *scheduled*(t_i, t_1, t_2) is true if a test $t_i \in S$ and is scheduled between t_1 and t_2; that is:

$$\{t_i \in S \wedge \neg(t_{end}(t_i) < t_1 \vee t_{start}(t_i) > t_2)\} \qquad (4.3)$$

The Boolean function *scheduled*(r_i, t_1, t_2) is true if a source r_i is used by a test scheduled between t_1 and t_2; that is:

$$\{r_i = source(t_j) \wedge \forall t_j \in S \wedge \neg(t_{end}(t_j) < t_1 \vee t_{start}(t_j) > t_2)\} \qquad (4.4)$$

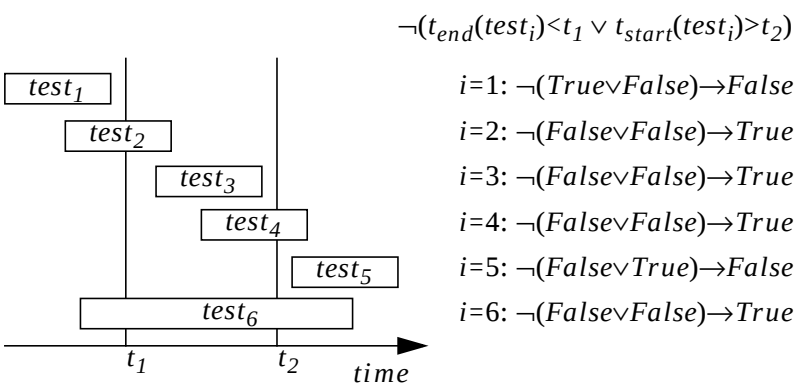


Figure 4.5: The function *scheduled*.

The Boolean function $scheduled(r_i, t_1, t_2)$ is true if a sink r_i is used by a test scheduled between t_1 and t_2; that is:

$$\{r_i = sink(t_j) \wedge \forall t_j \in S \wedge \neg(t_{end}(t_j) < t_1 \vee t_{start}(t_j) > t_2)\} \quad (4.5)$$

Similarly, the Boolean function $scheduled(constraint(t_i), t_1, t_2)$ is true if all blocks in the set of constraints are not scheduled between t_1 and t_2; that is:

$$\{block(t_j) \in constraint(t_i) \wedge \neg(t_{end}(t_j) < t_1 \vee t_{start}(t_j) > t_2)\} \quad (4.6)$$

The brief example in Figure 4.5 is used to illustrate the *scheduled* function. Six tests are scheduled and for each of them a computation is performed to determine whether it is scheduled between t_1 and t_2 or not.

4.3.1 TEST CONFLICTS

Tests may not be scheduled concurrently due to test conflicts. For SOC designs several different types of conflicts may occur. However, it is important to define a general approach to capture these conflicts and allowing flexible design of the system.

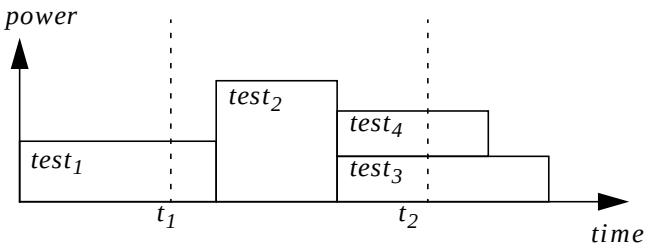


Figure 4.6: Scheduled power.

In this thesis we define conflicts as the set of blocks required for a test to be scheduled. A test of a block may only be scheduled if all its required blocks are available.

In this way we are able to capture many different types of test conflicts. For instance, if a set of blocks share a dedicated clock. Such constraints are easily captured with the proposed approach if these blocks can not be tested concurrently.

Test resource conflicts are handled separately.

4.3.2 POWER DISSIPATION

The power consumed at the test mode can be much higher than during normal mode due to the increased switching activity [Her98]. An additive model used by Zorian [Zor93], Chou *et al.* [Cho97] and Muresan *et al.* [Mur00] for power consumption is used in our approach. The function $p_{sch}(t_1, t_2)$ denotes the peak scheduled power between t_1 and t_2:

$$max\left\{\sum_{\forall t_i \;\; scheduled(t_i, t)} p_{test}(t_i) - p_{idle}(block(t_i)) + \sum_{\forall b_{i,j} \in B} p_{idle}(b_{i,j}) \middle| t_1 \le t \le t_2 \right\} \quad (4.7)$$

where $scheduled(t_i,t)$ is true if a test t_i is scheduled at time t.

We assume that only one test may be applied on each block concurrently. For instance, applying the function $p_{sch}(t_1, t_2)$ on the schedule in Figure 4.6, with t_1 and t_2 as indicated in the figure, returns $p_{test}(test_2) + p_{idle}(block(test_1)) + p_{idle}(block(test_3)) + p_{idle}(block(test_4))$.

4.4 Test Floor-planning

An efficient placement of test resources in the system minimizes the routing of the test access mechanism. Zorian proposes a test scheduling technique where the routing of test control lines is minimized [Zor93], see Section 3.4. In the approach, the cores are assumed to be tested with one dedicated BIST structure per core which means that no access mechanism is required for transporting test data. However, in the general case it is not feasible to assume that all cores can always be tested with only one dedicated BIST structure. Furthermore, a block may be tested by several test sets produced and analyzed using different test resources. Test resources may also be shared among several blocks at different cores. Therefore is it important to consider the placement of test resource since it affects the routing of the test access mechanism.

4.5 Test Set

In our approach, each test set is associated with a test source and a test sink. Furthermore, the test power consumption, test memory requirement and test application time are assumed to be given.

A strategy for test set selection is proposed by Sugihara *et al.* [Sug98], described in Section 3.5, where each core is tested by two test sets: one set using an external tester and one set produced at a BIST structure. In this thesis an approach is proposed where the number of test sets needed for a block is generalized to be arbitrary.

In order to evaluate the most efficient test schedule, it is desirable to try different test sets for the system where the test sets show different characteristics. We assume that a block test is a set of tests that completely tests the block.

For instance, assume that the block tests $BT_{i,j,1}=\{t_1, t_2, t_3\}$, $BT_{i,j,2}=\{t_4, t_5, t_6\}$, and $BT_{i,j,3}=\{t_7, t_8\}$ are given for a block $b_{i,j}$ at

core c_i. Each of the block tests is sufficient to test block $b_{i,j}$. Therefore only one of these sets must be selected. However, all the tests within the selected block test must be scheduled. Note that it is not useful to select tests from different block tests for the testing of a particular block. For instance, applying only tests t_1, t_4 and t_7 is sufficient.

4.6 Test Access Mechanism

The test access mechanism is the infrastructure used to transport test stimuli and test response in the system. Test vectors are transported from test sources to the core under test and test response is transported from the core under test to the test sink.

4.6.1 TEST ACCESS MECHANISM DESIGN

For a given system with test *ST*, defined above, the problem of test access mechanism design is twofold:

- A test access mechanism has to be added for transportation of test vectors from the test sources to the cores and for transportation of test response from the cores to the test sinks.
- A test schedule on the access mechanism, which in principle is to determine for all tests which wire to be used.

A system can be modelled as a directed graph, $G=(V,A)$, where a vertex v_i in V corresponds to a member of the set of cores, C, the set of test sources, R_{source}, and the set of test sinks, R_{sink}, *i.e.* $v_i \in C \vee R_{source} \vee R_{sink}$.

An arc, $a_k \in A$, between two vertices v_i and v_j indicates the existence of a test access mechanism (a wire) where it is possible to transport test data from v_i to v_j.

Initially no test access mechanism exists in the system, *i.e.* $A=\emptyset$. However, if the functional infrastructure or part of it may be used for test purpose, it is specified in A.

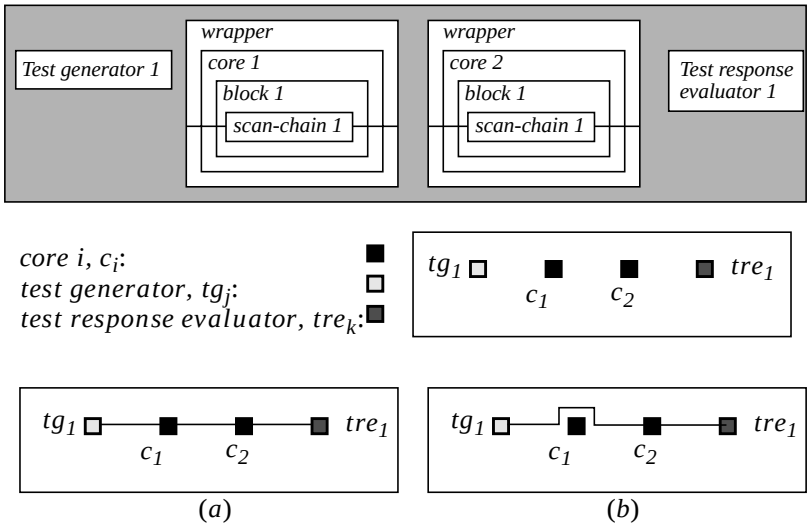


Figure 4.7: Design of the test access mechanism.

The problem of connecting vertices at a minimal cost in length is similar to the *travelling salesperson problem* (*TSP*) which is known to be an NP-complete problem and justifies the use of heuristics [Ree93].

When adding a test access mechanism between test generator tg_1 and core c_2 and test response evaluator tre_1, as illustrated in Figure 4.7, and the test data must pass through a core several routing options are possible:

1. through the logic of the core c_1, see Figure 4.7(a), using the transparent mode of the core, for instance,
2. through an optional bypass structure of core c_1, see Figure 4.7(a), and
3. around the core c_1, see Figure 4.7(b), where the access mechanism is not connected to the core.

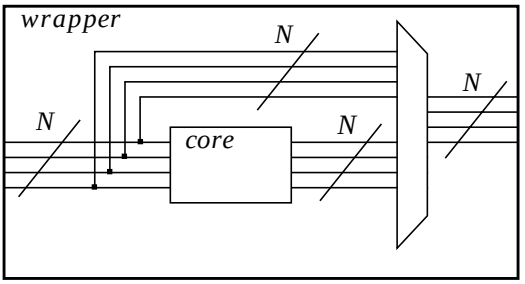


Figure 4.8: Bypass with no delay.

The advantage of alternatives 1 and 2 above is that the test access mechanism can be reused when testing a block at core c_1. However, there might be an additional delay due to such a structure which does not occur at alternative 3. In this thesis it is assumed that either a non-clocked bypass is used, see Figure 4.8, or that the effect of a clocked bypass is ignored. A non-clocked by-pass can be achieved using a wrapper approach as proposed by Marinissen *et al.* [Mar00].

4.6.2 TEST PARALLELIZATION

By test parallelization we mean the division of a scan-chain into several scan-chains of shorter length, which will lead to a shorter test application time. The test application time for a scan-based design is mainly determined by the number of test vectors, the length of the scan-chain and the clock frequencies.

We assume that the degree of *parallelization* is linear with respect to test time and test power consumption. The test time $t'_{test}(t_i)$ for a test t_i is given by:

$$t'_{test}(t_i) = \left\lceil \frac{t_{test}(t_i)}{par(block(t_i))} \right\rceil \qquad (4.8)$$

where $t_{test}(t_i)$ is the test time for a test t_i at no parallelization degree, *i.e.* par=1, $par(block(t_i))$ is the degree of parallelization.

It should be noted that the parallelization at a block can not be different for different test sets. For instance, the original scan-chain can not be divided into n chains at one moment and to m chains at another moment where $m \neq n$.

Let $parallelization(b_{i,j})$ be the parallelization degree for tests at block $b_{i,j}$. The parallelization degree can be defined by the designer or determined by an algorithm. In our approach, the integrated test algorithm will determine the $parallelization(b_{i,j})$ for all blocks, $b_{i,j} \in B$.

Aertes *et al.* assume, as in this thesis, that the test time may be divided into equal portions [Aer98]. A division of the scan-chain into several sub-chains reduces the test application time since all sub-chains may be loaded with different test vectors concurrently. Furthermore, due to the shorter length of the scan-chains, less shift in and out is required for each test vector. However, the switching activity may increase resulting in higher test power consumption. In this thesis it is assumed that the term $t_{test} \times p_{test}$ is constant and that the test power $p'_{test}(t_i)$ for a test t_i is given by:

$$p'_{test}(t_i) = p_{test}(t_i) \times par(block(t_i)) \qquad (4.9)$$

where $p_{test}(t_i)$ is the test power for a test t_i at no parallelization degree, $par=1$.

Definition 4.10: A test bus w_i is a path of edges $\{(v_0,v_1), (v_1,v_2),...,(v_{n-1},v_n)\}$ where $v_0 \in R_{source}$ and $v_n \in R_{sink}$.

The physical placement of all blocks, $b_{i,j} \in B$, all test sources $v_k \in R_{source}$ and all test sinks $v_l \in R_{sink}$ is given by x and y coordinates, the *x-placement* and the *y-placement* respectively.

Definition 4.11: $\Delta y_{i,j}$ is defined as: $\{y(v_i) - y(v_j) | v_i, v_j \in V\}$ and $\Delta x_{i,j}$ is defined as: $\{x(v_i) - x(v_j) | v_i, v_j \in V\}$ where $x(v_i)$ and $y(v_i)$ are the physical placements defined by *x-placement* and the *y-placement* respectively for a vertex v_i.

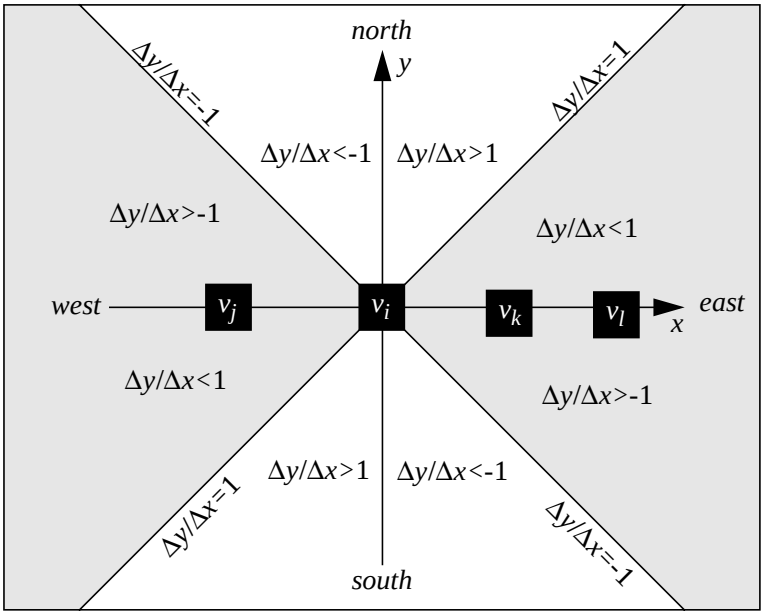


Figure 4.9: North, east, south and west orientation.

The distance between vertex v_i and vertex v_j is given by:

$$dist(v_i, v_j) = (\sqrt{(\Delta y_{i,j})^2 + (\Delta x_{i,j})^2}, v_i \in V, v_j \in V) \qquad (4.12)$$

Since the problem of finding the shortest path in a graph is an NP-complete problem [Gar79], it is practical to develop efficient heuristics for solving this problem. In our approach a heuristic makes use of the information of the neighbourhood of the cores and test resources. The information on the nearest core in four direction, *north*, *east*, *south* and *west*, are stored for each vertex. For instance v_j is the west core of v_i in Figure 4.9 which means that it is the vertex where $\Delta y_{i,j}/\Delta x_{i,j}<1$ or $\Delta y_{i,j}/\Delta x_{i,j}>-1$ and $x(v_j)>x(v_i)$ with the minimal distance between v_i and v_j, $dist(v_i,v_j)$ for all v_i, $v_j \in V$ where $i \neq j$.

The function $south(v_i)$ of vertex v_i gives the closest vertex south of v_i and it is defined as:

$$\left\{ v_j \middle| \left(\frac{\Delta y_{i,j}}{\Delta x_{i,j}} > 1 \vee \frac{\Delta y_{i,j}}{\Delta x_{i,j}} < -1 \right), y(v_j) < y(v_i), i \neq j, \forall v_i \forall v_j min\{dist(v_i, v_j)\} \right\} \quad (4.13)$$

The function $north(v_i)$ of vertex v_i is the closest vertex north of v_i and it is defined as:

$$\left\{ v_j \middle| \left(\frac{\Delta y_{i,j}}{\Delta x_{i,j}} > 1 \vee \frac{\Delta y_{i,j}}{\Delta x_{i,j}} < -1 \right), y(v_j) > y(v_i), i \neq j, \forall v_i \forall v_j min\{dist(v_i, v_j)\} \right\} \quad (4.14)$$

The function $west(c_i)$ of vertex v_i is the closest vertex west of v_i and it is defined as:

$$\left\{ v_j \middle| \left(-1 < \frac{\Delta y_{i,j}}{\Delta x_{i,j}} < 1 \right), x(v_j) < x(v_i), i \neq j, \forall v_i \forall v_j min\{dist(v_i, v_j)\} \right\} \quad (4.15)$$

The function $east(c_i)$ of vertex v_i is the closest vertex east of v_i and it is defined as:

$$\left\{ v_j \middle| \left(-1 < \frac{\Delta y_{i,j}}{\Delta x_{i,j}} < 1 \right), x(v_j) > x(v_i), i \neq j, \forall v_i \forall v_j min\{dist(v_i, v_j)\} \right\} \quad (4.16)$$

The operation $insert(v_i, v_j)$ inserts a directed arc from vertex v_i to vertex v_j *if and only if*:

$$\{south(v_i, v_j) \vee north(v_i, v_j) \vee west(v_i, v_j) \vee east(v_i, v_j)\} \quad (4.17)$$

The function $closest(v_i, v_j)$ gives the closest vertex in of v_j

$$\{v_k | min\{dist(v_k, v_j)\}, v_k = east(v_i) \vee v_k = south(v_i) \vee v_k = west(v_i) \vee v_k = north(v_i)\} \quad (4.18)$$

Among the four candidate vertex given by $east(v_i)$, $south(v_i)$, $west(v_i)$ and $north(v_i)$ the one closest to v_j is selected in Equation 4.18.

The operation $add(v_i, v_j)$ recursivly adds arcs from v_i to v_j, following the closest neighbourhood nodes:

$$\begin{aligned} &if(i \neq j)\{\dot{v}_j = closest(v_i, v_j); \\ &\quad insert(v_i, \dot{v}_j); \\ &\quad add(\dot{v}_j, v_j)\} \end{aligned} \tag{4.19}$$

The Boolean function $scheduled(w_i, t_1, t_2)$ is true when a path w_i is used by tests scheduled between t_1 and t_2:

$$\{w_i \in bus(t_j) \wedge \forall t_j \in S \wedge \neg(t_{end}(t_j) < t_1 \vee t_{start}(t_j) > t_2)\} \tag{4.20}$$

where $bus(t_j)$ is the set of buses allocated for test t_j.

4.6.3 TEST SOURCE LIMITATIONS

A test generator may use a memory for storing the test patterns. In particular, external test generators use such a memory with a limited size which introduces constraints on test scheduling [Het99].

The function $memory_{alloc}(r_i, t_1, t_2)$ provides the peak allocated memory between t_1 to t_2:

$$max\left\{\sum_{t_j \in S} \left|t_{memory}(t_j)\right| \middle| scheduled(t_j, t) \wedge r_i = source(t_j) \middle| t_1 \leq t \leq t_2\right\} \tag{4.21}$$

A test source has usually a limited bandwidth. For instance, an external tester may only support a limited number of scan chains at a time or there could be a limit in the available pins for test. This information is captured in the attribute bandwidth for each test source.

The function $bandwidth_{alloc}(r_i, t_1, t_2)$ gives the maximal number of buses allocated between t_1 and t_2:

$$max\left\{\sum_{t_j \in S} \left|bus(t_j)\right| \middle| scheduled(t_j, t) \wedge r_i = source(t_j) \middle| t_1 \leq t \leq t_2\right\} \tag{4.22}$$

where $bus(t_j)$ is the set of buses allocated for test t_j.

The functions memory and bandwidth provide the maximal memory respectively bandwidth used by the scheduled tests during a time interval, compare with scheduled power in Figure 4.6.

4.7 The System Test Algorithm

The system test algorithm described in this section integrates the issues discussed above. It selects test sets for the blocks in the design, schedule the tests, floor-plan the test resources needed and designs a test access mechanism while considering test conflicts, power consumption and test resource limitations.

The tests are initially sorted according to a key k, which can be based on *power*(p), *test time*(t) or *power*×*test time*(pt).

If k=p; $f_p(t_i) = p_{test}(t_i)$ (4.23)

if k=t; $f_t(t_i) = t_{test}(t_i)$ (4.24)

if k=pt; $f_{pt}(t_i) = t_{test}(t_i) \times p_{test}(t_i)$ (4.25)

Let P be the ordered set where the tests are ordered based on the key k, that is:

$$(P(t_i) < P(t_j) \,|\, f_k(t_i) \leq f_k(t_j), t_i \in T, t_j \in T) \quad (4.26)$$

where $P(t_i) < P(t_j)$indicates that test t_i is sorted before test t_j.

If new tests are allowed to be scheduled even if all tests in a session are completed, see point 2 in Section 4.3, the function *nexttime*(t_{old}) provides the next time where it is possible to schedule:

$$\{t_i \,|\, min(t_{end}(t_i)), t_{old} < t_{end}(t_i), \forall t_i \in S\} \quad (4.27)$$

If new tests are not allowed to be scheduled until all tests in a session are all completed, see point 2 in Section 4.3, the function *nexttime*(t_{old}) provides the next point when it is possible to schedule a new test:

$$\{t_i \,|\, max(t_{end}(t_i)), t_{old} < t_{end}(t_i), \forall t_i \in S\} \quad (4.28)$$

4.7.1 THE ALGORITHM

In this section the issues discussed above are combined in an algorithm. The input to the algorithm is a system with tests and the output of the algorithm is a test schedule and a test access mechanism. The algorithm is divided into three parts namely:

- the *system test algorithm*,
- the *test resource floor-planning algorithm*, and
- the *test access mechanism design algorithm*.

We will explain the steps in the algorithm in details. In the following text, the line starting with a number corresponds to the step in the algorithm with the same number.

The System Test Algorithm

The system test algorithm is illustrated in Figure 4.10. First all tests in the system are sorted based on a key *k* and placed in a list *P*. Initially no tests are scheduled and the time is set to zero, *i.e.*:

1: *Sort all tests in P based on time, power or time×power;*
2: $S=\emptyset$;
3: $t=0$;

The algorithm then iterates until for all blocks in the system there exists a block test where all tests within the particular block test are scheduled:

4: *until* $\forall b_{p,q} \exists BT_{p,q,r} \forall t_s \in BT_{p,q,r} \wedge t_s \in S$ *do begin*

A second iteration is performed over all tests in the system where in each iteration a test *cur* is checked and its characteristics are computed, such as test source, core and test sink:

5: *for all cur in P do begin*
6: $b_{i,j} = block(cur)$;
7: $v_a = source(cur)$;
8: $v_b = c_i$;
9: $v_c = sink(cur)$;

1: Sort all tests in P based on time, power or time×power;
2: $S=\varnothing$;
3: $t=0$;
4: until $\forall b_{p,q} \exists BT_{p,q,r} \forall t_s \in BT_{p,q,r} \wedge t_s \in S$ *do begin*
5: for all cur in P do begin
6: $b_{i,j} = block(cur)$;
7: $v_a = source(cur)$;
8: $v_b = c_i$;
9: $v_c = sink(cur)$;
10: if $(parallelization(b_{i,j})=\infty)$ *then begin*
11: $par = min\{par_{max}(b_{i,j}),$
$\lfloor (p_{max}-p_{sch}(t, t_{end})) / p_{test}(cur) \rfloor,$
$bandwidth(v_a, t, t_{end})-bandwidth_{alloc}(v_a, t, t_{end})\}$;
12: end else
13: $par = parallelization(b_{i,j})$;
14: $t_{end} = t + \lceil t_{test}(cur)/par \rceil$;
15: $p_{test}'(cur) = p_{test}(cur) \times par$;
16: if $(\neg \exists t_f (t_f \in BT_{i,j,k} \wedge t_f \in S \wedge cur \notin BT_{i,j,k}) \wedge$
$(par \geq par_{min}(b_{i,j})) \wedge$
$\neg scheduled(cur, t, t_{end}) \wedge$
$\neg scheduled(v_a, t, t_{end}) \wedge$
$\neg scheduled(v_c, t, t_{end}) \wedge$
$\neg scheduled(constraint(cur), t, t_{end}) \wedge$
$memory(v_a) > t_{memory}(cur) + memory_{alloc}(v_a, t, t_{end}))$
then begin
17: if $(parallelization(b_{i,j}=\infty)$ *then*
18: $parallelization(b_{i,j}=par$;
19: call floor-planning procedure;
20: call test access mechanism procedure;
21: $t_{start}(cur)=t$;
22: $t_{end}(cur)=t_{end}$;
23: $S=S \cup \{cur\}$;
24: $P=P-\{cur\}$;
25: end;
26: end;
27: $t=nexttime(t)$;
28:end;

Figure 4.10: The system test algorithm.

The parallelization affects the test time and the test power consumption of the test and it depends on:

- parallelization at the block,
- available power, and
- available bandwidth of test resources.

It must be the same for all tests at a specific block. For instance at a scan-based block the scan chain can not be divided in n chains at one moment and m chains at another moment ($n \neq m$), see Section 4.6.2.

It is desirable to maximize the parallelization at a block since it minimizes the test time at the block and also because it uses the resources at a minimum of time which allow other tests to use these resources. If the parallelization for a block is not determined it is computed as the minimum among the maximal parallelization at a block, the available power and the available bandwidth during t to t_{end}:

10: *if* ($parallelization(b_{i,j})=\infty$) *then begin*
11: $par=min\{par_{max}(b_{i,j}),$
$\lfloor (p_{max}-p_{sch}(t, t_{end})) / p_{test}(cur) \rfloor,$
$(bandwidth(v_a, t, t_{end})-bandwidth_{alloc}(v_a, t, t_{end}))\}$;
12: *end else*
13: $par=parallelization(b_{i,j})$;

When the degree of parallelization is given, the test time and the test power consumption is computed for the test *cur*:

14: $t_{end}=t+\lceil t_{test}(cur)/par \rceil$;
15: $p_{test}'(cur)=p_{test}(cur)\times par$;

A check of test *cur* is performed to determine whether it is possible to schedule it at time t which include:

- $\neg\exists t_f\ (t_f \in BT_{i,j,k} \wedge t_f \in S \wedge cur \notin BT_{i,j,k})$ checks that there does not exist any scheduled test where the test belong to $BT_{i,j,k}$ and the test *cur* does not belong to $BT_{i,j,k}$. It is not allowed to schedule a test at a block when a test belonging to another block test is scheduled on this block.

- $par \geq par_{min}(b_{i,j})$ checks whether the computed parallelization is higher than the minimal allowed parallelization at the block.
- $\neg scheduled(cur, t, t_{end})$ checks whether test cur is not scheduled during t to t_{end},
- $\neg scheduled(v_a, t, t_{end})$ checks whether the test source used by test cur is not scheduled during t to $t_{end,}$
- $\neg scheduled(v_c, t, t_{end})$ checks whether the test sink used by test cur is not scheduled during t to $t_{end,}$
- $\neg scheduled(constraint(cur), t, t_{end})$ checks whether the blocks required by test cur are not scheduled during t to t_{end}, and
- $memory(v_a) > t_{memory}(cur) + memory_{alloc}(v_a, t, t_{end})$ checks the available memory at the test source needed by test cur during t to t_{end}.

16: *if* $(\neg\exists t_f (t_f \in BT_{i,j,k} \wedge t_f \in S \wedge cur \notin BT_{i,j,k}) \wedge$
$(par \geq par_{min}(b_{i,j})) \wedge$
$\neg scheduled(cur, t, t_{end}) \wedge$
$\neg scheduled(v_a, t, t_{end}) \wedge$
$\neg scheduled(v_c, t, t_{end}) \wedge$
$\neg scheduled(constraint(cur), t, t_{end}) \wedge$
$memory(v_a) > t_{memory}(cur) + memory_{alloc}(v_a, t, t_{end}))$
then begin

If the parallelization is not stored for the block, it is set to par:

17: *if* $(parallelization(b_{i,j}) = \infty)$ *then*
18: $parallelization(b_{i,j}) = par$;

The *test resource floor-planning* procedure, see Figure 4.11, and the *test access mechanism procedure* are called, see Figure 4.14.

19: *call floor-planning procedure*;
20: *call test access mechanism procedure;*

The attributes for the test is set, the test is inserted in S and removed from P:

21: $t_{start}(cur) = t$;
22: $t_{end}(cur) = t_{end}$;
23: $S = S \cup \{cur\}$;
24: $P = P - \{cur\}$;
25: *end*;
26: *end*;

After each iteration over the tests in P a new time is calculated:

27: *t=nexttime(t);*
28: *end;*

The Test Resource Floor-planning Algorithm

The system test algorithm makes use of another algorithm which performs the test resource floor-planning, illustrated in Figure 4.11. The algorithm floor-plan test resources and it determines whereas a test resource should be moved. If a test resource is selected to be moved, the new placement is determined by the algorithm. The input to the algorithm is a system with test, ST, where test resources may be placed or not.

Assume a test *cur* to be scheduled. If the test resources have not been floor-planned in the system, they are placed at the core which contain the block tested by test *cur*:

1: *if* $\neg$*scheduled*(v_a, 0, t_{end}) *then begin*
2: $y(v_a)=y(cur)$;
3: $x(v_a)=x(cur)$;
4: *end;*
5: *if* $\neg$*scheduled*(v_c, 0, t_{end}) *then begin*
6: $y(v_c)=y(cur)$;
7: $x(v_c)=x(cur)$;
8: *end;*

Then the algorithm determines whether the test resources should be moved or not. A loop over the required test resources starts. In this thesis we assume up to two resources, one test source and one test sink, to be used per test. In the loop, the length of connecting the test source v_a with the block where test *cur* is to be performed v_b and the test sink v_c is computed.

9: *for all required test resources begin*
10: $new=dist(v_a, v_b)+dist(v_b, v_c)$;

The notation $\exists^{!u}$ is used by Barwise and Etchemendy [Bar93] to denote that there exist exactly u objects. In the approach by Barwise and Etchemendy u is known in advance. However, here it is assumed to be returned which means that u is the number of

1: if $\neg scheduled(v_a, 0, t_{end})$ *then begin*
2: $y(v_a)=y(cur)$;
3: $x(v_a)=x(cur)$;
4: end;
5: if $\neg scheduled(v_c, 0, t_{end})$ *then begin*
6: $y(v_c)=y(cur)$;
7: $x(v_c)=x(cur)$;
8: end;
9: for all required test resources begin
10: $new=dist(v_a, v_b)+dist(v_b, v_c)$;
11: $\exists^{!u} w_u \neg scheduled(w_u, t, t_{end}) \wedge v_a \in w_u \wedge v_b \in w_u \wedge v_c \in w_u \wedge w_u \in A$
12: $\exists^{!v'} v_a \in w_v \wedge v_c \in w_v \wedge w_v \in A$
13: $v=v'-u$;
14: $extend=0$;
15: for all min (par, v) w_l *such* $\neg scheduled(w_l, t, t_{end}) \wedge v_m \in w_l \wedge v_n \in w_l \wedge v_o \in w_l \wedge w_l \in A$ *do begin*
16: $extend=extend+min\{new, dist(v_m, v_a)+dist(v_a, v_m)+dist(v_n, v_b)+dist(v_b, v_n)+dist(v_o, v_c)+dist(v_c, v_o)\}$
17: end;
18: if $(par>v)$ *then*
19: $extend=extend+new\times(par-u)$;
20: if $dist(v_a,v_b)>0 \wedge dist(v_b,v_c)>0$ *then*
21: $move=v\times min\{dist(v_a, v_b), dist(v_b, v_c)\}$
22: else
23: $move=v\times max\{dist(v_a, v_b), dist(v_b, v_c)\}$
24: if $(move<min\{extend, new\times par\})$ *then begin*
25: $\Delta_{x,y}=min\{dist(v_a,v_b),dist(v_c,v_b)|dist(v_a,v_b)>0\ dist(v_c,v_b)>0\}$
26: for $g=1$ *to* $v+u$
27: $add(v_x, v_b)$;
28: $y(v_x)=y(v_b)$;
29: $x(v_x)=x(v_b)$;
30: end;
31:end;

Figure 4.11: The test resource floor-planning algorithm.

connections not scheduled from t to t_{end} and connecting v_a, v_b and v_c. Further, v' is the number of wires connecting the test source and the test sink and v is the number of not scheduled wires connecting the test source and the test sink.

11: $\exists^{!u} w_u \neg scheduled(w_u, t, t_{end}) \wedge v_a \in w_u \wedge v_b \in w_u \wedge v_c \in w_u \wedge w_u \in A$
12: $\exists^{!v'} v_a \in w_v \wedge v_c \in w_v \wedge w_v \in A$
13: $v = v' - u$;

The variable *extend* is initialized to zero and an iteration over the minimum between *par* and *v* is performed where *par* is the parallelization. The total length of extending the existing wires is calculated.

14: $extend = 0$;
15: *for all* $min(par, v)$ w_l *such* $\neg scheduled(w_l, t, t_{end})$, $v_m \in w_l$, $v_n \in w_l$, $v_o \in w_l$, $w_l \in A$ *do begin*
16: $extend = extend + min\{new, dist(v_m, v_a) + dist(v_a, v_m) + dist(v_n, v_b) + dist(v_b, v_n) + dist(v_o, v_c) + dist(v_c, v_o)\}$
17: *end*;

If *par* is greater than *v*, the available wires are not sufficient, *i.e.* new wires have to be added with a length of *new*.

18: *if* $(par > v)$ *then*
19: $extend = extend + new \times (par - u)$;

The cost of moving a test resource is given as *v* multiplied by the shortest distance between the core, v_b and v_a and v_c respectively:

20: *if* $dist(v_a, v_b) > 0 \wedge dist(v_b, v_c) > 0$ *then*
21: $move = v \times min\{dist(v_a, v_b), dist(v_b, v_c)\}$
22: *else*
23: $move = v \times max\{dist(v_a, v_b), dist(v_b, v_c)\}$

If *move* is less than the minimum between *extend*, cost of extending existing wires, and $new \times par$, cost of adding completely new wires, a test resource is moved. If a test resource is to be moved, the shortest distance between v_b to v_a and between v_b to v_c is selected and $v+u$ wires are added. Finally the test resource is moved to the core where test *cur* is to be applied.

```
24:     if (move<min{extend, new×par}) then begin
25:         Δx,y=min{dist(va,vb),dist(vc,vb)|dist(va,vb)>0 dist(vc,vb)>0}
26:         for g= 1 to v+u
27:             add(vx, vb);
28:         y(vx)=y(vb);
29:         x(vx)=x(vb);
30:     end;
31: end;
```

The test floor-plan algorithm is illustrated using the part of a system shown in Figure 4.12. A test at vertex v_b is to be scheduled using the test generator at vertex v_a and the test response evaluator at vertex v_c. For simplicity, we assume that the distance between all vertices is 1 unit length. Three wires exist in this part of the system, w_1, w_2 and w_3. At the moment wire w_2 is occupied by test data transportation and can not be used. Assume that par = 3, *i.e.* this test requires three wires connecting v_a,v_b and v_c for test data transportation.

The algorithm first checks whether the test resources are placed in the system. In this example they are placed. A loop over the required number of test resources starts and the length of a new wire connecting v_a with v_b and v_b with v_c (connecting v_a, v_3, v_7, v_b, v_8, v_9, v_6, v_c) is computed to 7 length units and it is stored in variable *new*.

In this example there is one wire, w_2, connecting v_a, v_b and v_c which can be used at this moment, *i.e.* u=1 and there are 3 wires connecting v_a and v_c, *i.e.* v'=3, and v=2.

An iteration over $min(par, v)$, in this example $min(3,2)$=2, wires where a wire is checked as to whether it can be used in each iteration. First, wire w_3 is found since its cost is lowest (=0) since it connects v_a, v_b and v_c, *i.e.* variable *extend*=0. In the second iteration, *extend* becomes 4. The algorithm does not find any wire connecting v_a, v_b and v_c (note w_2 is not available and w_3 was selected in the previous iteration). The selected cost is the minimum of adding a new wire at a cost of 7 length units and the cost of extending wire w_1 at 4 length units (connecting v_2 to v_6 to

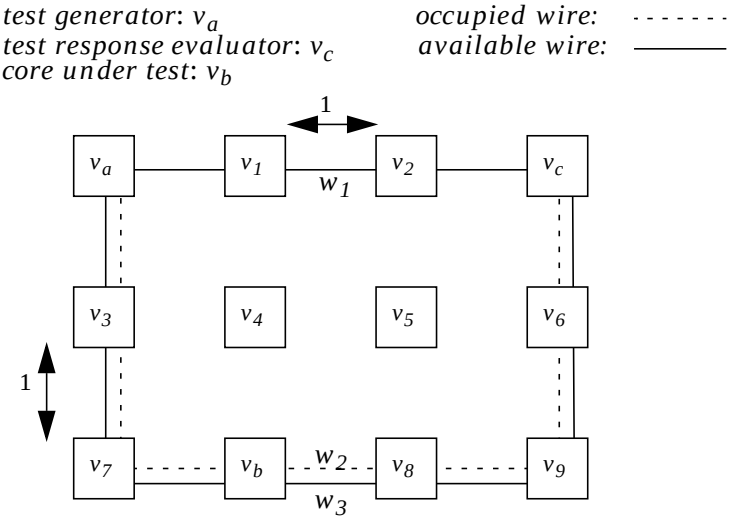


Figure 4.12: Example to illustrate the test resource floor-planning algorithm.

v_b and from v_b to v_6 and to v_3). Such extension is required in order to concurrently transport test stimuli and test response on a wire. This is further explained in Figure 4.16.

The loop terminates and the cost of selecting two wires has been computed. However, a third wire is required and since no wire is available, the cost of adding a new wire with cost 7 has to be added, *i.e. extend*=11 (4+7).

The cost of moving a test resource is given by the number of wires connecting the test source and the test sink multiplied by the distance to the core under test. For instance, if v_a is to be moved to v_b the wires at v_a has to be routed to v_b. In this example there are three wires (v+u) connecting v_a and v_c. The cost of moving the test resource closest to v_b is evaluated. In this example $move=(v+u)\times min\{dist(v_a,v_b),dist(v_c,v_b)\}=(2+1)\times min\{3,4\}=9$.

Since *move* is less than *extend*, the test resource closest to v_b is moved. In this example it is v_a and all wires at v_a are extended to v_b (where v_a will be placed), see Figure 4.13 for the placement

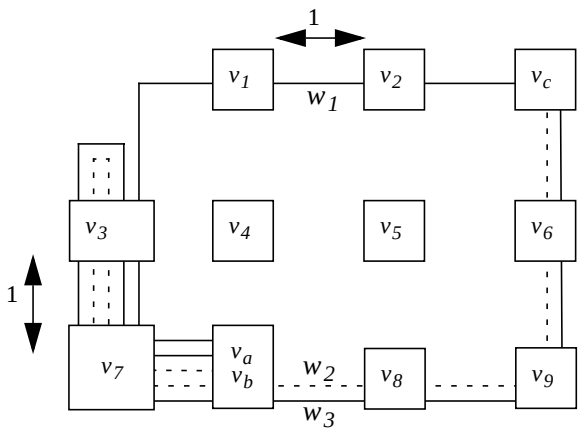


Figure 4.13: Example used to illustrate the test resource floor-planning algorithm.

and routing. The first of the two iterations over the test resources are completed. A second iteration follows which is performed in similar way.

The Test Access Mechanism Design Algorithm

The system test algorithm makes use of an algorithm for the test access mechanism design, illustrated in Figure 4.14. The basic idea of the algorithm is to minimize the routing of connecting a block with required test resources. The algorithm iterates over the required number of wires and in each iteration it checks for the minimal wire connection. The most efficient is to use an existing wire which connects the test resource and the core under test. If no such wire exists, the minimum between adding a new wire and extending an existing wire is explored.

In detail the algorithm works as follows. An iteration is performed over the number of required wires determined by *par*.

1: *for g*= 1 *to par begin*
2: *extend*=∞;

```
1:   for g=1 to par begin
2:      extend=∞;
3:      if ∃w_l{v_a∈w_l∧v_b∈w_l∧v_c∈w_l∧
           ¬scheduled(w_l, t, t_end)}
4:      else
5:      begin
6:         new=dist(v_a, v_b)+dist(v_b, v_c);
7:         if ∃w_l∀v_m∀v_n∀v_o min{dist(v_m, v_a)+
              dist(v_a, v_m)+dist(v_n, v_b)+dist(v_b, v_n)+
              dist(v_o,v_c)+ dist(v_c, v_o)}∧
              ¬scheduled(w_l, t, t_end) ∧
              v_m∈w_l ∧ v_n∈w_l ∧v_o∈w_l ∧w_l∈A
           then
8:            extend=dist(v_m, v_a)+dist(v_a, v_m)+
                 dist(v_n, v_b)+ dist(v_b, v_n)+
                 dist(v_o, v_c)+dist(v_c, v_o);
9:         if (new<extend)
10:           w_l=add(v_a, v_b)+add(v_b, v_c);
11:        else begin
12:           Δ_a,b=0; Δ_c,d=0; Δ_e,f=0;
13:           while Δ_a,b≠∞ ∧ Δ_c,d≠∞ ∧ Δ_e,f≠∞ begin
14:              if Δ_a,b≠∞ then
15:                 Δ_a,b = min{dist(v_m, v_a)| v_m∈w_l }
16:              if Δ_c,d≠∞ then
17:                 Δ_c,d = min{dist(v_n, v_b)| v_n∈w_l}
18:              if Δ_e,f≠∞ then
19:                 Δ_e,f = min{dist(v_o, v_c)| v_o∈w_l}
20:              Δ_x,y=min(Δ_ab, Δ_cd, Δ_ef);
21:              w_l=w_l ∪ add(v_x, v_y)+add(v_x, v_y);
22:              Δ_x,y=∞;
23:           end;
24:        end;
25:     end;
26:     t_bus(cur)=t_bus(cur)∪w_l;
27:  end;
```

Figure 4.14: Test access mechanism design algorithm.

If there is a wire not being used by any test during t to t_{end} and connecting the test source, the core under test and the test sink, it is selected, step 3. Otherwise, the distance of adding a new wire is calculated, step 6. Note, that the test resource floor-planning also calculates this distance but since the test resources could be moved a new calculation has to be performed.

Then a check is made to find a wire not being used by a test during t to t_{end} which is as short as possible. Its length is computed and stored in *extend*.

3: *if* $\exists w_l\{v_a \in w_l \wedge v_b \in w_l \wedge v_c \in w_l \wedge$
$\neg scheduled(w_l, t, t_{end})\}$
4: *else*
5: *begin*
6: $new = dist(v_a, v_b) + dist(v_b, v_c)$;
7: *if* $\exists w_l \forall v_m \forall v_n \forall v_o min\{dist(v_m, v_a) +$
$dist(v_a, v_m) + dist(v_n, v_b) + dist(v_b, v_n) +$
$dist(v_o, v_c) + dist(v_c, v_o)\} \wedge$
$\neg scheduled(w_l, t, t_{end}),$
$v_m \in w_l, v_n \in w_l, v_o \in w_l, w_l \in A$
then
8: $extend = dist(v_m, v_a) + dist(v_a, v_m) +$
$dist(v_n, v_b) + dist(v_b, v_n) +$
$dist(v_o, v_c) + dist(v_c, v_o)$;

If *new* is less than *extend*, the cost of adding a new wire is lower than extending a wire, a new wire is added; otherwise an existing wire is extended.

9: *if* $(new < extend)$
10: $w_l = add(v_a, v_b) + add(v_b, v_c)$;
11: *else begin*

For the wire extension an iteration is performed while $\Delta_{a,b} \neq \infty \wedge \Delta_{c,d} \neq \infty \wedge \Delta_{e,f} \neq \infty$. In each iteration the $\Delta_{a,b}$, $\Delta_{c,d}$ and $\Delta_{e,f}$ is re-calculated if they are not set to ∞ and the minimum among them is selected and the wire is extended.

12: $\Delta_{a,b}=0$; $\Delta_{c,d}=0$; $\Delta_{e,f}=0$;
13: *while* $\Delta_{a,b} \neq \infty \wedge \Delta_{c,d} \neq \infty \wedge \Delta_{e,f} \neq \infty$ *begin*
14: *if* $\Delta_{a,b} \neq \infty$ *then*
15: $\Delta_{a,b} = min\{dist(v_m, v_a) | v_m \in w_l\}$
16: *if* $\Delta_{c,d} \neq \infty$ *then*
17: $\Delta_{c,d} = min\{dist(v_n, v_b) | v_n \in w_l\}$
18: *if* $\Delta_{e,f} \neq \infty$ *then*
19: $\Delta_{e,f} = min\{dist(v_o, v_c) | v_o \in w_l\}$
20: $\Delta_{x,y}=min(\Delta_{ab}, \Delta_{cd}, \Delta_{ef})$;
21: $w_l=w_l \cup add(v_x, v_y)+add(v_x, v_y)$;
22: $\Delta_{x,y}=\infty$;
23: *end;*
24: *end;*
25: *end;*

Finally, the wire is added to be used by the test *cur*:

26: $t_{bus}(cur)=t_{bus}(cur) \cup w_l$;
27: *end*;

A part of a system is illustrated in Figure 4.15 which will be used to explain the test access mechanism design algorithm. In this part of the system there are nine vertices (cores or test resources) and three wires, w_1, w_2 and w_3. The distance between two neighbouring vertices is 1 length unit. A test at a block at core v_b is to be scheduled and it requires test source v_a and test sink v_c. Assume that par=2 for this test which means that two connections (wires) are required for test data transportation. Wire w_1 and wire w_2 are connecting v_a, v_b and v_c. However, w_1 is being used at the moment and can therefore not be used.

In the first of the two iterations (par=2), the algorithm detects that wire w_2 connects v_a, v_b and v_c and it is available for test data transportation at the moment. Wire w_2 is selected. In the second iteration, the algorithm detects that there is no available wire connecting v_a, v_b and v_c. Therefore the algorithm determines whether a new wire should be added connecting v_a, v_b and v_c or whether an existing wire should be extended to include v_a, v_b and v_c. The length of adding a new wire is calculated as being 3 length units. For the extension of a wire, the wire requiring the

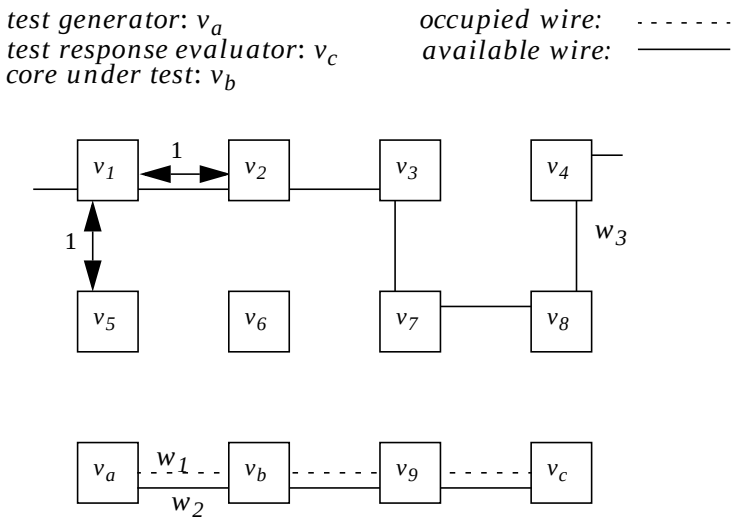


Figure 4.15: Test access mechanism design.

minimal extension is selected. In this case it is wire w_3 and extending it requires 4 length units for connecting v_a to v_1, 4 length units for connecting v_2 to v_b and 2 length units for connecting v_8 to v_c. In total 10 length units are required. The brief example in Figure 4.16 motivates the calculation of extending a wire. If wire w_1 is to be extended to include the scan-chain at $block_{2,1}$ at $core_2$ a wire has to be added from $core_1$ to $core_2$ and a wire from $core_2$ to $core_1$ in order to make it possible to concurrently transport test stimuli to $core_2$ and test response from $core_2$.

In the example in Figure 4.15, a new wire is added since adding a new wire is less expensive compared to extending wire w_3.

4.7.2 COMPUTATIONAL COMPLEXITY

The main iterations for the algorithms are given in Figure 4.17 where the test resource placement and the test access mechanism design parts are excluded. This makes it possible to compare the technique with other approaches.

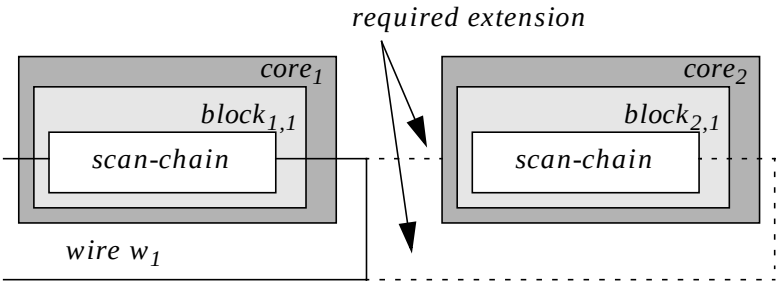


Figure 4.16: Extending a wire.

1:sort tests according to a key and put them in a list P.
2:while P is not empty
3: for i = 1 to |P|
4: if item$_i$ = ok then remove item$_i$ from P

Figure 4.17: Computational complexity analysis of the test scheduling technique.

Sorting can be performed at using a sorting algorithm at $O(|P|\times log\ |P|)$ [Aho87] and the worst case is occur when only one test is scheduled in the loop at point 3 in Figure 4.17 and it is given by:

$$\sum_{i=0}^{|P-1|}(P-i) = P+(P-1)+(P-2)+\ldots+(P-(P-2))+(P-(P-1)) = \\ = \frac{|P|^2}{2}+\frac{|P|}{2} \qquad (4.29)$$

The total complexity is $|P|\times log(P) + |P|^2/2 + |P|/2$ which is of $O(n^2)$. For instance, the approach proposed by Garg *et al.* [Gar91] and Chakrabarty *et al.* have a worst case complexity of $O(n^3)$ [Cha99].

4.8 Simulated Annealing

Simulated annealing is an optimization technique proposed by Kirkpatrick *et al*. [Kir83] where a hill-climbing mechanism is used to avoid getting stuck in a local optimum.

4.8.1 THE SIMULATED ANNEALING ALGORITHM

The basic idea behind Simulated annealing is that a combinatorial optimization procedure corresponds to the annealing process in physics where a material is heated up to its melting point and then its minimal energy state is found by slowly lowering the temperature [Kir83].

The Simulated annealing algorithm is outlined in Figure 4.18. An initial solution is first created. A minor modification of it creates a neighbouring solution and the cost of the new solution is evaluated. If the new solution is better than the previous, the new solution is kept. A worse solution can be accepted at a certain probability, which is controlled by a parameter referred to as temperature.

The temperature is decreased during the optimization process, and the probability of accepting a worse solution decreases with the reduction of the temperature value. When the temperature value is approximately zero, the optimization terminates.

The initial temperature *TI*, the temperature length *TL* and alpha α ($0 < \alpha < 1$) have to be determined.

The advantages with Simulated annealing is the relative ease in implementing it. However, it suffers from long computational time and it requires complicated tuning of the annealing parameters *TI*, *TL* and α [Gaj92]. Furthermore, there is no clear rules for parameter selection. In this thesis, the parameter selection is based on experimental results.

We use the Simulated annealing algorithm for test scheduling and for combined test scheduling and test access mechanism. In

```
1:  Construct initial solution, x^now;
2:  Initial Temperature: T:=TI;
3:  while stop criteria not met do begin
4:      for i = 1 to TL do begin
5:          Generate randomly a neighboring solution x'∈N(x^now);
6:          Compute change of cost function ΔC:=C(x')-C(x^now);
7:          if ΔC≤0 then x^now=x'
8:              else begin
9:                  Generate q := random(0, 1);
10:                 if q<e^(-ΔC/T) then x^now=x'
11:             end;
12:     end;
13:     Set new temperature T:=α×T;
14: end;
15: Return solution that corresponds to the minimum cost function;
```

Figure 4.18: Simulated annealing.

this thesis we use our heuristic to create the initial solution in both cases. For the test scheduling case we create a neighbouring solution by randomly selecting a test from an existing schedule and schedule it as soon as possible but not at the same place as it was in the original schedule.

For the test scheduling and test infrastructure design a modification is defined as a random insertion or deletion of a test bus wire connecting a test source and the cores on the path to a test sink. It is obvious that if the functional bus is allowed to be used, it may not be deleted.

4.8.2 COST FUNCTION

The cost function of a test schedule, S, and the added test access mechanism, A, is given by:

$$C(S, A) = \alpha_1 \times T(S) + \alpha_2 \times L(A) \qquad (4.30)$$

where:

$T(S)$ is the test application time for a sequence of tests, S,
$L(A)$ is the total length of the test access mechanism,

α_1, α_2 are two designer-specified constants used to determine the importance of the test application time versus the test bus area.

The test application time, $T(S)$, for a schedule, S, is defined by:

$$T(S) = \{t_{end}(t_i) \mid \forall t_i(max\{t_{end}(t_i)\}), t_i \in S\} \quad (4.31)$$

The length, $L(A)$, of the test access mechanism, A, is given by:

$$\sum_{w_j \in A} \sum_{j=0}^{|w_i - 1|} dist(v_j, v_{j+1}), v_j, v_{j+1} \in w_i \quad (4.32)$$

4.9 Tabu Search

Tabu search is an artificial intelligence inspired technique where the intelligence is kept in a memory. An initial solution is transformed, by successive moves, to an optimal solution [Glo86].

A *short term memory* with a predefined length is used to remember the given number of recent moves. These moves are not allowed to be repeated, *i.e.*, they are tabus. An *aspiration criterion*, when a tabu may be overridden, is defined to indicate when a tabu-status move can be performed. One example of such aspiration criteria is that the move will lead to a solution which has the best so far. In our algorithm this aspiration criteria is used. A *long term memory* is also used to keep track of certain characteristics of the successful moves. It capture information about which parts of the solution space that has been investigated. This information can be used to guide the search to a *diversification*, encouraging jumping out of local optimum, or a *intensification* coveraging towards a certain area to reach the optimum (either local or global).

In each iteration a neighbourhood is examined where the best move is selected to form the next solution. For each core and test resource in the system it is specified weather it may be moved or

```
1:  Construct initial solution, x^now;
2:  for each iteration
3:     for each solution x_k ∈ N(x^now) do begin
4:        Compute change of cost function ΔC:=C(x_k)-C(x^now);
5:           for all ΔC<0, in increasing order of ΔC do begin
6:                 if not tabu(t_k) or tabu_aspirant(t_k) then
7:                    x^now= x_k;
8:                    update tabu and history list;
9:                    goto 2;
10:             end;
11:       end;
12:    Return solution that corresponds to the minimum cost function;
```

Figure 4.19: Tabu search algorithm.

not and a set of pre-defined places where a core or a test resource may be placed.

The Tabu search algorithm is illustrated in Figure 4.19 and the cost function is the same as described in Section 4.8.

The initial solution is created by our heuristic and a move is performed by changing placement between two test resources.

The Tabu search and the Simulated annealing implementations are also combined where the Tabu search optimizes the floor planning while the Simulated annealing minimizes the test application time and the test access mechanism design for each given floor-plan.

4.10 Conclusions

In this chapter an integrated test framework for SOC has been defined. Several issues which are important for the designer when developing an efficient test solution are described, generalized and combined into the framework. Algorithms combining these issues have been defined where test application time and test access mechanism are minimized while performing:

- test sets selection for each block,
- test parallelization,
- test resource floor-planning,
- test access mechanism design, and
- test scheduling

under the following constraints:

- general test conflicts,
- power consumption,
- test source memory, and
- test source bandwidth.

For further optimization of the test scheduling and/or test access mechanism Simulated annealing is used and for further optimization of the test resource floor-planning Tabu search is used. The heuristics are used for creating the initial solutions and during the design space exploration process.

Chapter 5
Experimental Results

IN THIS CHAPTER the experimental results are presented. After the introduction in Section 5.1, the results from the experiments on test scheduling are reported in Section 5.2 and the experimental results on test access mechanism design are presented in Section 5.3. The results from the experiments on test scheduling and test access mechanism design are presented in Section 5.4 and in Section 5.5 the results from the experiments on test parallelization are presented. Finally, in Section 5.6 the experiments on test resource placement are presented and the chapter is summarized in Section 5.7.

5.1 Introduction

We have made several experiments using benchmarks and industrial designs. We have compared our approach to other proposed approaches and we compared our approach with the optimal solution. In cases where no optimal solution is known, we use extensive optimization using our Simulated annealing (SA) implementation, see Section 4.8, and/or our Tabu search (TS) implementation, see Section 4.9.

All experiments, where a computational cost is stated, are performed on a Sun Ultra Sparc 10 with a 450 MHz processor and 256 Mbyte RAM.

5.2 Test Scheduling

The test scheduling approach proposed in this thesis is compared to several previously proposed approaches. In Section 5.2.1 and Section 5.2.2 we perform experiments where the test application time is minimized considering test conflicts. In Sections 5.2.3 to 5.2.9 we report the results from experiments where test application time is minimized while considering test conflicts and test power consumption.

5.2.1 EXPERIMENT ON DESIGN KIME

The design Kime, described in Appendix A.1, has been used by Kime and Saluja [Kim82], Craig *et al.* [Cra88], Jone *et al.* [Jon89] and Garg *et al.* [Gar91]. The design contains test conflicts and the test application time for the optimal solution is 318 time units. Since no power consumption is given for the tests, we only performed the experiment using our approach with an initial sorting of the tests based on time. The solution from our approach is shown in Figure 5.1 and it was produced within one second. All approaches but the one proposed by Kime and Saluja find the optimal solution, see Table 5.1.

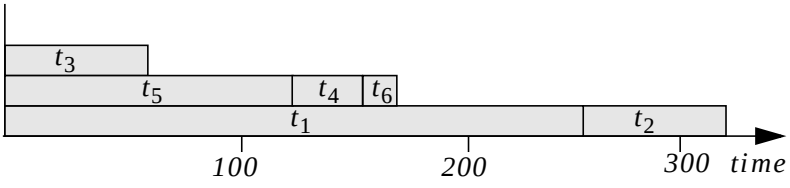


Figure 5.1: Test schedule using our heuristic on design Kime.

Approach	Test time	Difference to optimum
Optimal	318	-
Kime and Saluja	349	8.9%
Craig *et al.*	318	0%
Jone *et al.*	318	0%
Garg *et al.*	318	0%
Our heuristic (time sort)	318	0%

Table 5.1: Experimental results on design Kime.

5.2.2 EXPERIMENT ON SYSTEM S

The System S defined by Chakrabarty [Ch00a] consists of six ISCAS 85 benchmarks where each of them is assumed to be a core, see Appendix A.2. Each core is tested by two test sets and the test sets can not be applied to the same core concurrently. Furthermore, the external tester can only be used by one test at a time. A BIST pattern takes one clock cycle to apply and an external test pattern is applied at a speed ten times slower and all cores have their own BIST resource [Ch00a].

No power consumption is given for the tests and therefore we only use our scheduling approach with an initial sorting of the tests based on time. The test schedule achieved by our approach is shown in Figure 5.2.

All results are presented in Table 5.2 where the test application time of the optimal solution is 1152810. The test application time of the solution produced by the approach proposed by Chakrabarty is 1204630 which is 4.5% worse than the optimal solution. Our approach finds the optimal solution within a second.

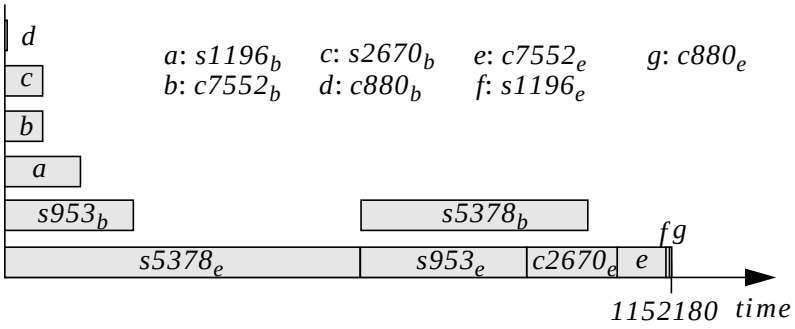


Figure 5.2: Test schedule using our heuristic on System S.

Approach	Test time	Difference to optimum
Optimal	1152810	-
Chakrabarty	1204630	4.5%
Our heuristic (time sort)	1152810	0%

Table 5.2: Experimental results on System S.

5.2.3 EXPERIMENT ON DESIGN MURESAN

The design by Muresan *et al.* contains test conflicts and power constraints, see Appendix A.3 [Mur00]. The total test application time using the approach by Muresan *et al.* is 29 time units, see the test schedule in Figure 5.3. The results from our SA optimization is shown in Figure 5.4 and it was running with an initial temperature (TI) of 400, a temperature length (TL) of 400 and α=0.97. The experimental results using our approach with initial sorting of tests based on power, time and power×time are shown in Figure 5.5.

All experimental results are presented in Table 5.3. Our heuristics show better results in all cases compared to the solution by Muresan *et al.* Further, our approach with initial sorting based on *power×time* results in a solution only 4% from the solution produced by SA. All solutions using our approach were produced within a second while the SA optimization required 90 seconds.

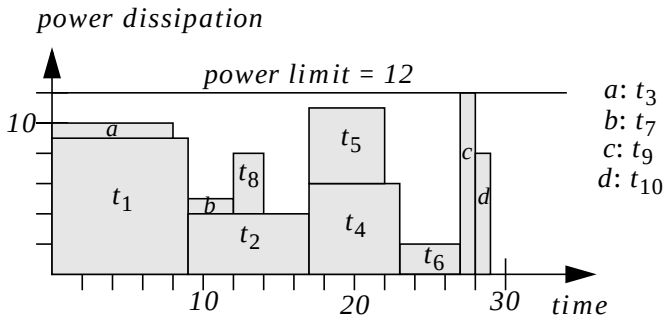


Figure 5.3: Test scheduling solution produced by Muresan *et al.* on design Muresan.

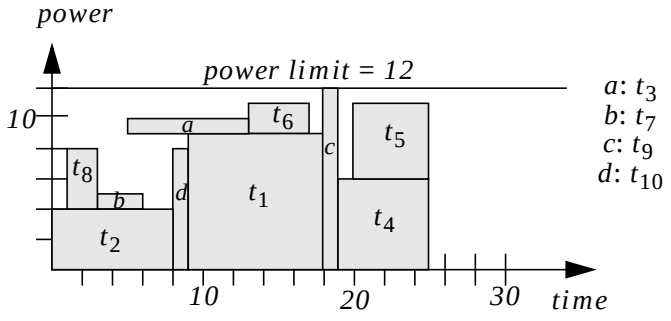


Figure 5.4: Test scheduling solution produced by our Simulated annealing implementation on design Muresan.

Approach	Test time	Difference to SA	Cpu time (sec.)
Our Simulated annealing	25	-	90
Muresan *et al.*	29	16%	-
Our heuristic, (power sort)	28	12%	1
Our heuristic, (time sort)	28	12%	1
Our heuristic, (power×time sort)	26	4%	1

Table 5.3: Experimental results on design Muresan.

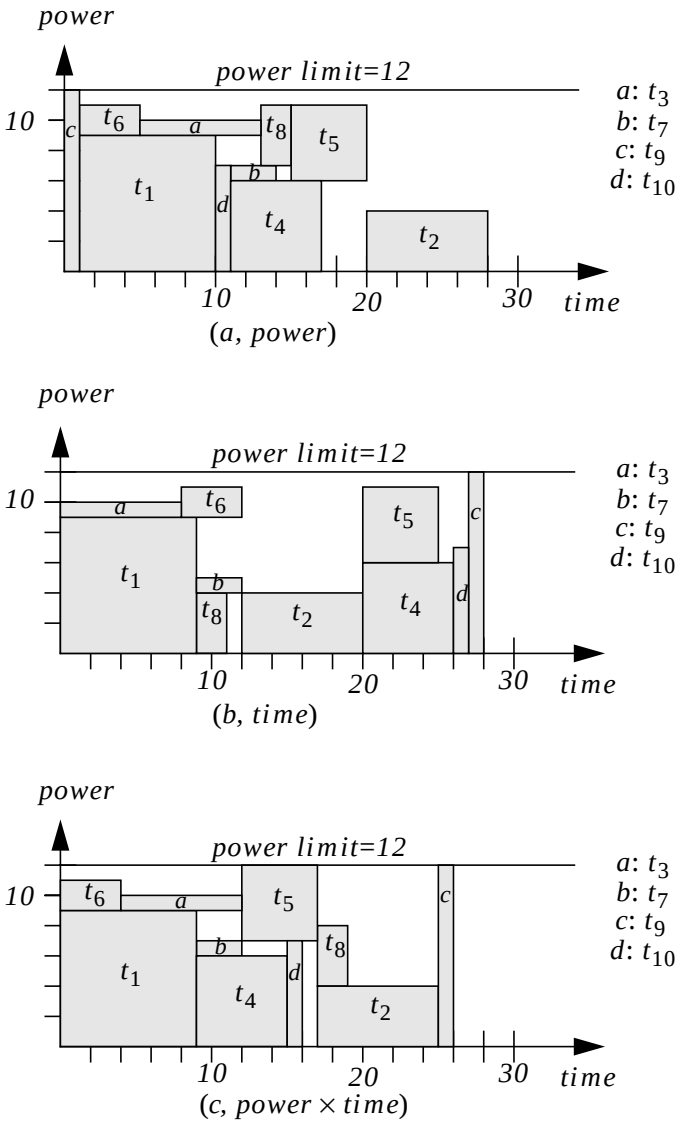


Figure 5.5: Test schedule using our heuristic with initial sorting based on power (a), time (b) and power×time (c) on design Muresan.

5.2.4 EXPERIMENT ONE ON ASIC Z

We have compared our test scheduling technique with the approaches proposed by Zorian [Zor93] and Chou *et al*. [Cho97] using the ASIC Z design, see Appendix A.4. The assumptions for the experiment are the same as Chou *et al.* [Cho97], namely:

- maximal power dissipation is limited to 900 mW,
- all tests can be applied concurrently,
- the power consumption for idle blocks are excluded, and
- no new tests are allowed to start until all tests in the previous session are completed.

The test schedules achieved by the approach proposed by Zorian and Chou *et al.* are presented in Figure 3.25 and in Figure 3.27 respectively. The test schedules achieved by our approach, with an initial sorting based on power, time and power×time, are the same and shown in Figure 5.6.

All experimental results are presented in Table 5.4. Using our approach the total test application time is 300 in all cases of initial sorting. The approach proposed by Zorian results in a solution with four test sessions and a test application time of 392. The approach proposed by Chou *et al.* results in a solution with three test sessions and a test time of 331. The optimal solution has a test application time of 300, see Table 5.5. The approach proposed by Zorian is 30.7% from the optimal solution and the approach proposed by Chou *et al.* is 10.3% from optimum. Our approach finds the optimal solution within a second.

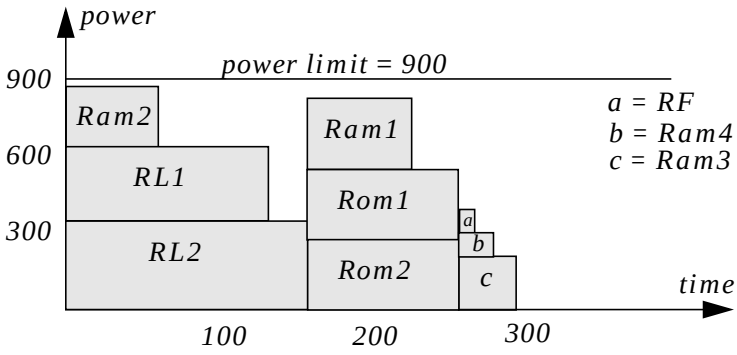


Figure 5.6: Test schedule achieved using our heuristic on ASIC Z.

Test session	Zorian		Chou *et al.*		Our heuristic (power, time, power×time)	
	Time	**Blocks**	**Time**	**Blocks**	**Time**	**Blocks**
1	69	Ram1, Ram4, RF	69	Ram1, Ram3, Ram4, RF	160	RL2, RL1, Ram2
2	160	RL1, RL2	160	RL1, RL2	102	Ram1, Rom1, Rom2
3	61	Ram2, Ram3	102	Rom1, Rom2, Ram2	38	Ram3, Ram4, RF
4	102	Rom1, Rom2				
Total time:	**392**		**331**		**300**	

Table 5.4: A comparison of different test scheduling approaches on ASIC Z.

Approach	Test time	Difference to optimum
Optimum	300	-
Zorian	392	30.7%
Chou *et al.*	331	10.3%
Our heuristic (time sort)	300	0%
Our heuristic (power sort)	300	0%
Our heuristic (power×time sort)	300	0%

Table 5.5: Experimental results on ASIC Z.

5.2.5 EXPERIMENT TWO ON ASIC Z

We have performed experiments on the ASIC Z design, see Appendix A.4, with the following assumptions:

- maximal power dissipation is limited to 900 mW,
- all tests can be applied concurrently,
- idle power is not considered, and
- new tests are allowed to start even if all tests are not completed.

The difference compared to the experiments in Section 5.2.4 is that tests are allowed to start even if all tests at the moment are not completed. The test schedules using our approach with the initial sorting of tests based on power, time and power×time are shown in Figure 5.7. The experimental results are presented in Table 5.6 and all cases result in a test application time of 262. The SA was running with initial temperature (TI) of 400, temperature length (TL) of 400 and α=0.97 and it found a solution at a cost of 262, see Table 5.6. Our test scheduling technique finds a solution at the same cost as the SA within a second while the SA optimization required 74 seconds to complete.

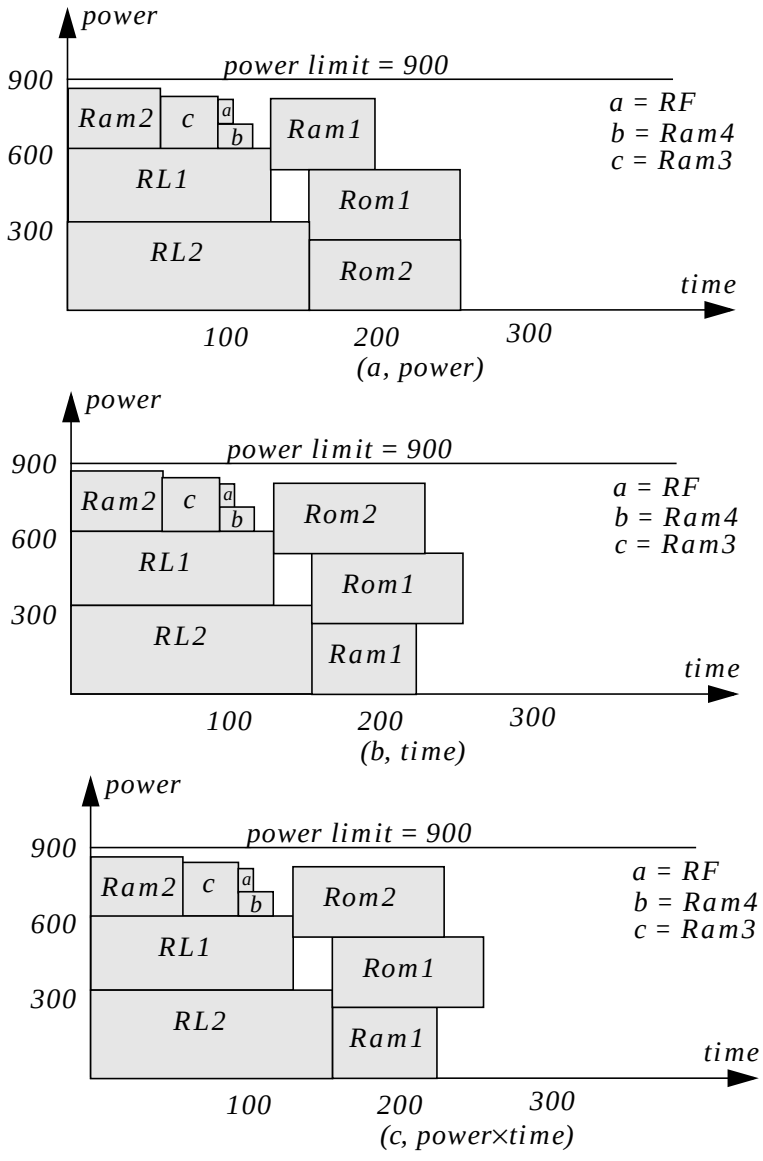


Figure 5.7: Test schedule achieved using our heuristic on ASIC Z using initial sorting based on power(a), time(b) and power×time(c).

Approach	Idle power considered	Test time	Difference to Simulated annealing	Cpu time (sec.)
Our Simulated annealing	no	262	-	74
Our heuristic (power sort)	no	262	0%	1
Our heuristic (time sort)	no	262	0%	1
Our heuristic (power×time sort)	no	262	0%	1

Table 5.6: Experimental results on ASIC Z.

5.2.6 EXPERIMENT THREE ON ASIC Z

We have also performed experiments on ASIC Z, see Appendix A.4, with the following assumptions:

- maximal power dissipation limited to 900 mW,
- all tests can be applied concurrently,
- idle power is considered,
- new tests are allowed to start even if all tests are not completed.

The test schedules achieved using our approach with the initial sorting of the tests based on power, time and power×time are shown in Figure 5.8. The SA was running with initial temperature (TI) of 400, temperature length (TL) of 400 and α=0.99 resulting in a test schedule shown in Figure 5.9.

The experimental results are presented in Table 5.7. Our approach with an initial sorting based on power results in a solution of 300 time units which is 9.5% from the solution produced by the SA optimization. Our approach using an initial sorting of the tests based on time and power×time results in a test time of 290 which is 5.8% from the SA solution. All solutions using our approach were produced within a second while the SA required 223 seconds.

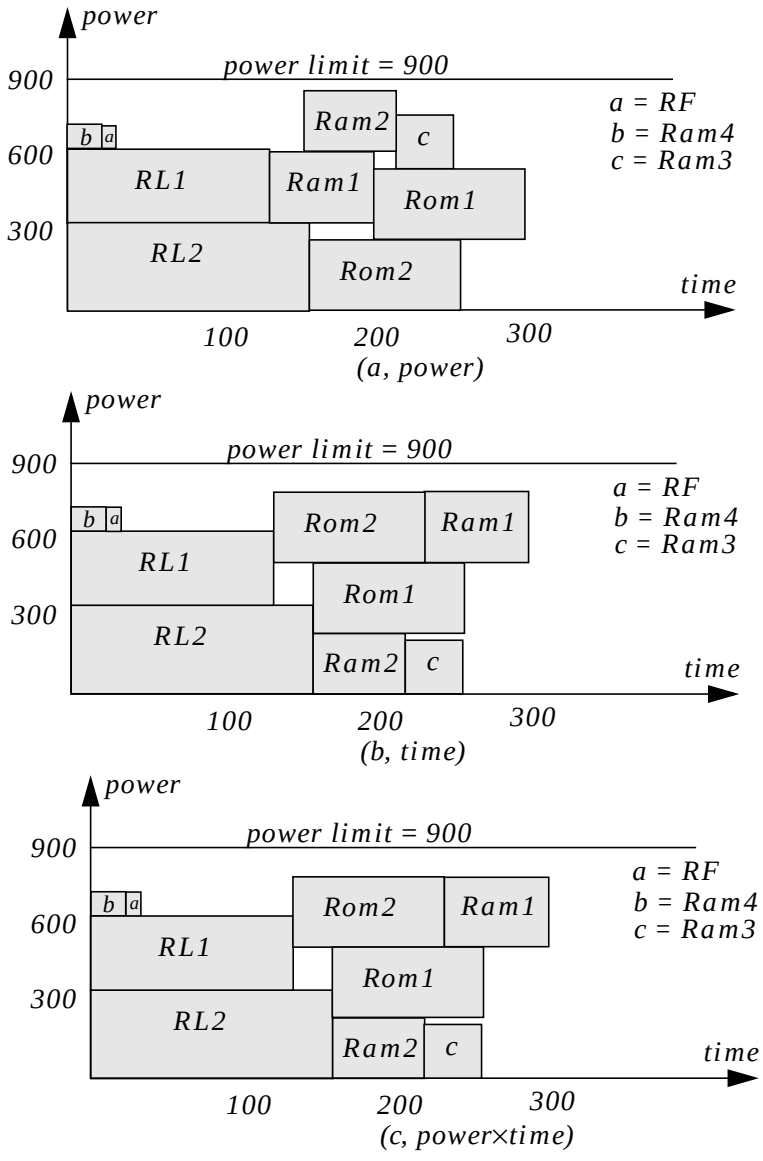


Figure 5.8: Test schedule achieved using our heuristic on ASIC Z using initial sorting based on power (a), time (b) and power×time (c).

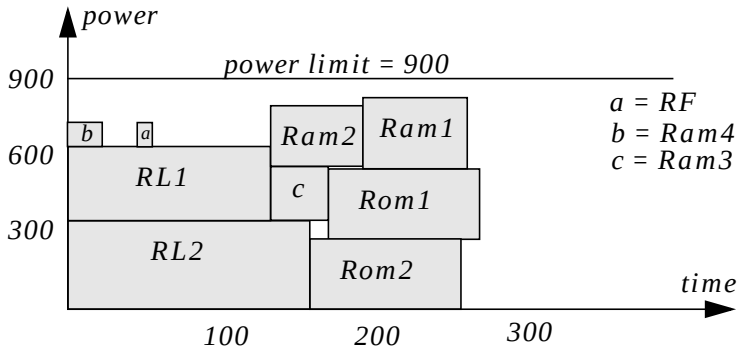


Figure 5.9: Test schedule achieved using our Simulated annealing implementation on ASIC Z.

Approach	Idle power considered	Test time	Difference to SA	Cpu time (sec.)
Our Simulated annealing	yes	274	-	223
Our heuristic (power sort)	yes	300	9.5%	1
Our heuristic (time sort)	yes	290	5.8%	1
Our heuristic (power×time sort)	yes	290	5.8%	1

Table 5.7: Results on ASIC Z.

5.2.7 EXPERIMENT ON EXTENDED ASIC Z

Extended ASIC Z, see Appendix A.5, is an extension of ASIC Z, see Appendix A.4, where each block is tested by three tests. There are two test sets for the block testing and one test set for interconnection test.

We allow tests to start even if all tests at the moment are not completed and idle power is not considered.

The results from the experiments are presented in Table 5.8. Our approach with an initial sorting of the tests based on power achieve a solution at 313. Our approach with an initial solution based on time and power×time results in solutions with a test application time of 287. The SA optimization with initial temperature on 400, temperature length of 400 and α=0.97 finds a solution at a cost of 264. The SA was running for 132 seconds while all solutions using our approach required less than 1 second to complete.

Approach	Test time	Difference to SA	Cpu time (sec.)
Our Simulated annealing	264	-	132
Our heuristic (power sort)	313	18.5%	1
Our heuristic (time sort)	287	8.7%	1
Our heuristic (power×time sort)	287	8.7%	1

Table 5.8: . Results on Extended ASIC Z.

5.2.8 EXPERIMENTS ON SYSTEM L

The System L is an industrial design, see Appendix A.6, where no data is available for test D, G and F. They are therefore excluded from the experiments.

The 15 tests are scheduled by a designer as shown in Figure 5.10 with a test application time of 1592 time units. Our approach with an initial sorting based on power is shown in Figure 5.11 and the test application time is 1077.

All experimental results are presented in Table 5.9. Our approach finds the optimal solution in all cases of initial sorting, which is 32% better than the solution produced by the designer. The time required to produce the solutions using our approach was for each case less than one second.

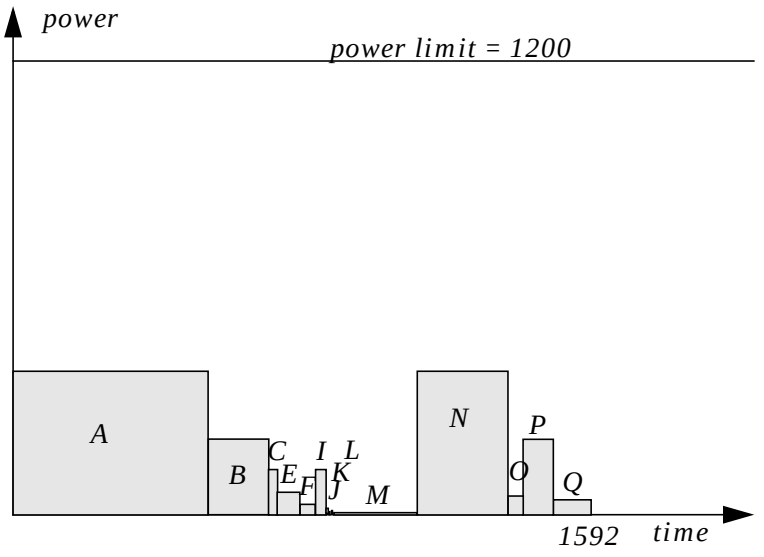


Figure 5.10: Designer's test schedule on System L.

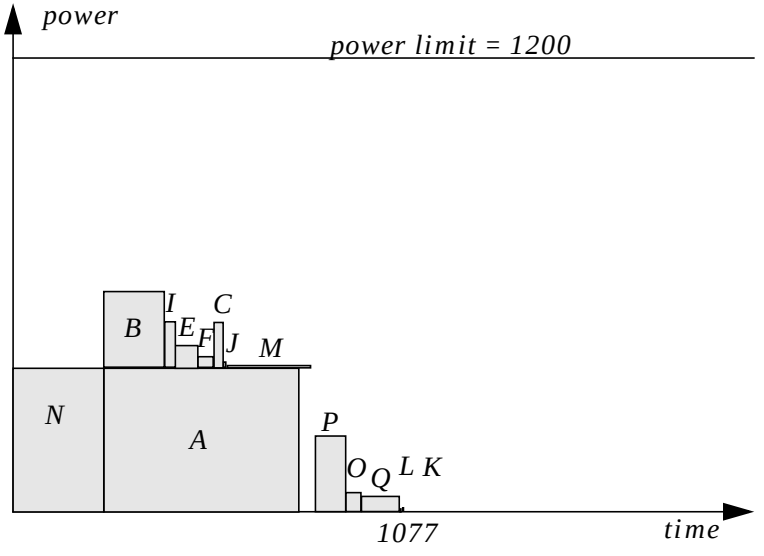


Figure 5.11: Test schedule achieved using our heuristic with sorting based on power on System L.

Approach	Test time	Difference to optimum
Optimal	1077	-
Designer	1592	32.3%
Our heuristic (power sort)	1077	0%
Our heuristic (time sort)	1077	0%
Our heuristic (power×time sort)	1077	0%

Table 5.9: Experimental results on System L.

5.2.9 EXPERIMENTS ON ERICSSON DESIGN

The results from the experiments on the Ericsson design, an industrial design described in Appendix A.7, are presented in Table 5.10. Extensive optimization using SA finds a solution with a test application time of 30899 where we used an initial temperature (TI) of 200, temperature length (TL) of 200 and α=0.95.

Our approach with an initial sorting based on power finds a solution at a test time of 37336 which is 20% from the solution produced by our SA implementation. Our approaches with initial sorting based on time and power×time find a solution at 34762 which is 12.5% from the solution found by SA. The SA optimization was running for 3260 seconds. The best SA solution was found after 465 seconds as illustrated in Figure 5.12. Our approaches find their solutions at 3 seconds.

Approach	Test time	Difference to SA	Cpu time (sec.)
Our Simulated annealing	30899	-	3260
Our heuristic (power sort)	37336	20.1%	1
Our heuristic (time sort)	34762	12.5%	1
Our heuristic (power×time sort)	34762	12.5%	1

Table 5.10: Experimental results on design Ericsson.

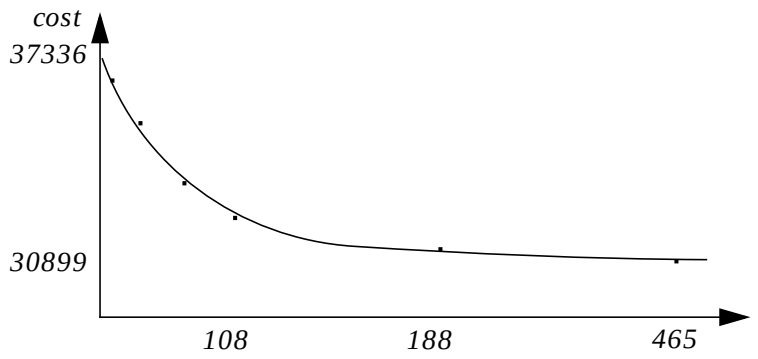


Figure 5.12: Simulated annealing optimization on Ericsson.

5.3 Test Access Mechanism Design

The experimental results on the test access mechanism design algorithm are presented in this section. For all cores and test resources in the design we assume a single point placement given by its *x*-coordinates and *y*-coordinates.

5.3.1 EXPERIMENT ON SYSTEM S

The System S consists of six ISCAS 85 benchmarks where each of them is assumed to be a core, see Appendix A.2. Using our test access mechanism design algorithm with the test access port TAP_{in} placed at (0,20) and TAP_{out} at (40,20) the length of the test access mechanism is 120 length units and the routing is shown in Figure 5.13. This is the optimal solution and it was produced within a second.

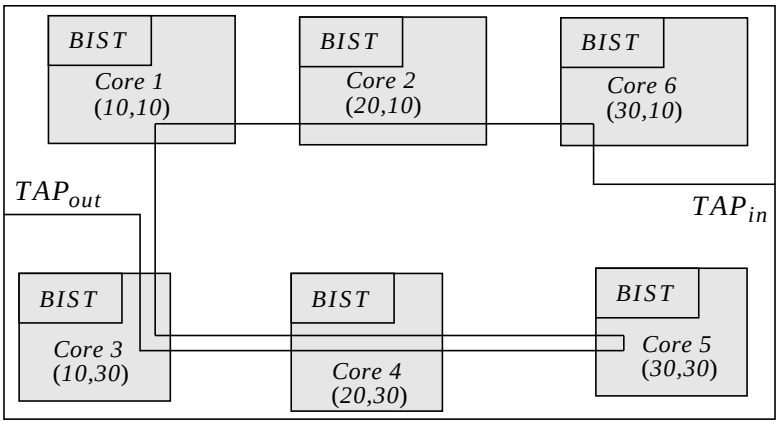


Figure 5.13: Test access design using our heuristic on System S.

5.4 Test Scheduling and Test Access Mechanism Design

In this section we report the results on the experiments where we minimize the test application time and the test access mechanism. In Section 5.4.1 the results from the experiments where test time and test access mechanism are minimized while test conflicts are considered on System S are presented. In Sections 5.4.2, 5.4.3, 5.4.4 and 5.4.5 the test application time and test access mechanism are minimized while considering test conflicts and test power consumption.

For all cores and test resources in the design we assume a single point placement given by its *x*-coordinates and *y*-coordinates.

5.4.1 EXPERIMENT ON SYSTEM S

In System S, see Appendix A.2, only one test may use the external tester at a time. However, in this experiment we assume that several tests can use the external tester concurrently. In the experiment, we minimize test application time and test access mechanism while considering test conflicts.

Since no power limitations are given for the tests in the system, we only performed experiments using our approach with an initial sorting of the tests based on time. The test schedule and test bus assignment is shown in Figure 5.14 and the routing of the test access mechanism is shown in Figure 5.15. Our solution requires 5 test buses for the transportation of test data from the external tester to the cores and from the cores to the external tester. The BIST tests do not require any test access mechanism since each core has its dedicated test resources. The test application time for the solution is 996194 and the length of the test access mechanism is 320 and it was computed within a second.

The test schedule and the test bus assignment after optimization using SA are shown in Figure 5.16 and the routing of the test access mechanism is shown in Figure 5.17. The experimental results are also presented in Table 5.11 where the test application time is the same using our approach and SA. However, the SA improves the test access mechanism from 320 to 160 which is an improvement of 100%. The SA required 1004 seconds (TI=TL=100, α=0.99).

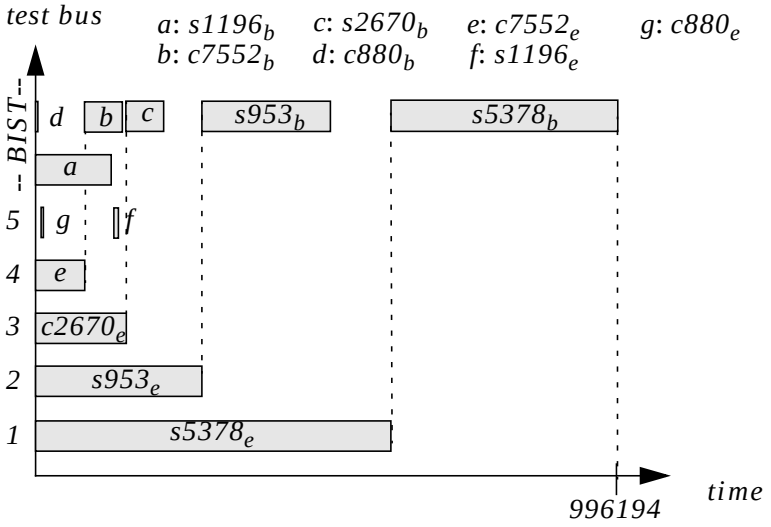


Figure 5.14: Our test schedule on System S.

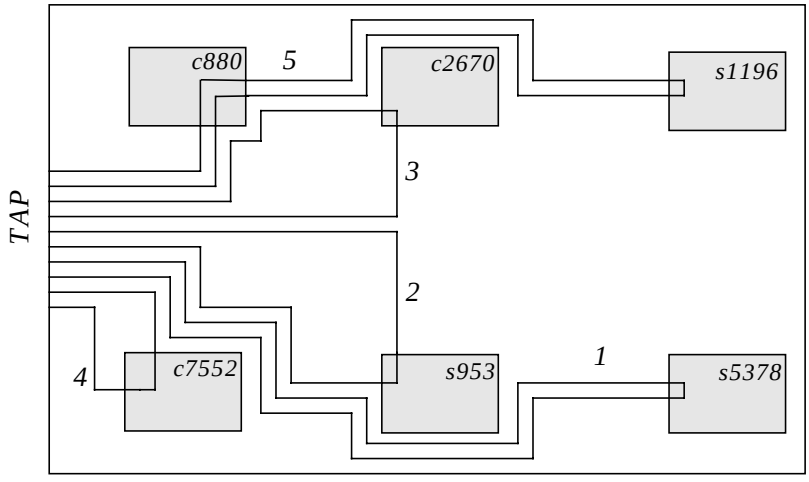


Figure 5.15: Test access mechanism design using our heuristic on System S.

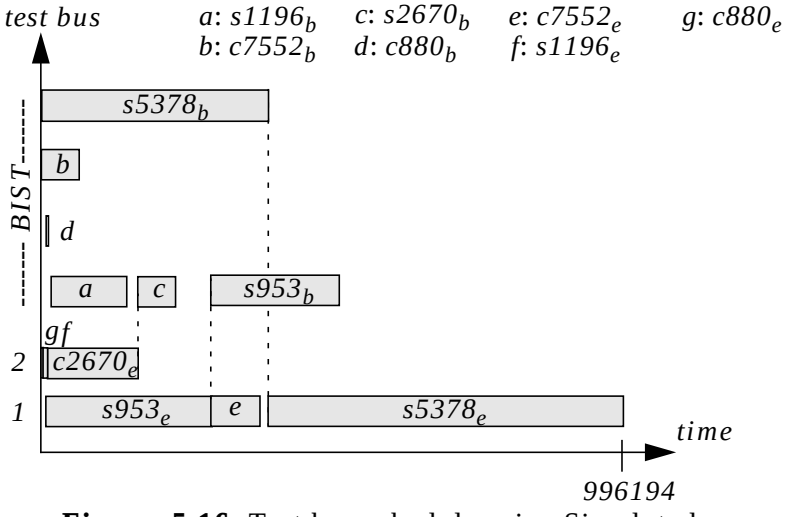


Figure 5.16: Test bus schedule using Simulated annealing on System S.

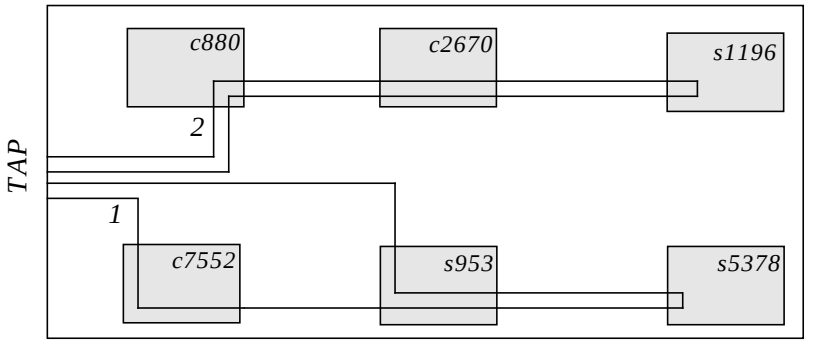


Figure 5.17: Test access mechanism design using Simulated annealing on System S.

Approach	Test time	Test access mechanism	Cpu time (sec.)
Our Simulated annealing	996194	160	1004
Our heuristic (time)	996194	320	1
Difference to SA	0%	100%	-

Table 5.11: Results on System S.

5.4.2 EXPERIMENT ONE ON ASIC Z

In the ASIC Z design each block has its own dedicated BIST structure, see Appendix A.4, which means that each block has its own test source and test sink. Therefore, there is no need for any mechanism for test data transportation. However, in this experiment we assume that no BIST structure exists in the design and all tests are applied using an external tester. Further, we assume that the external tester can support several tests concurrently.

We assume the maximal allowed power dissipation is 900 mW. In this experiment we do not consider idle power and we allow new tests to start even if all tests are not completed.

The solutions using our approach with an initial sorting of the tests based on power is shown in Figure 5.18 and in Figure 5.19.

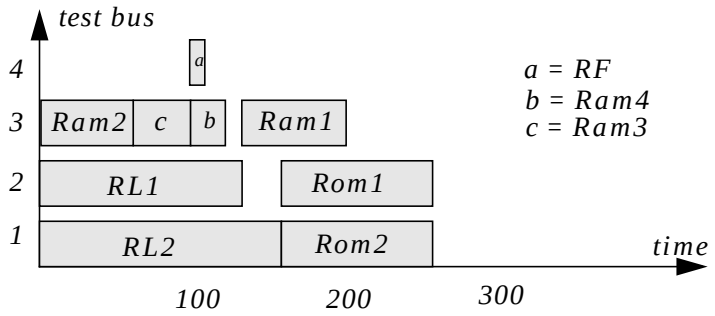


Figure 5.18: Test bus schedule achieved using our heuristic on ASIC Z using initial sorting based on power.

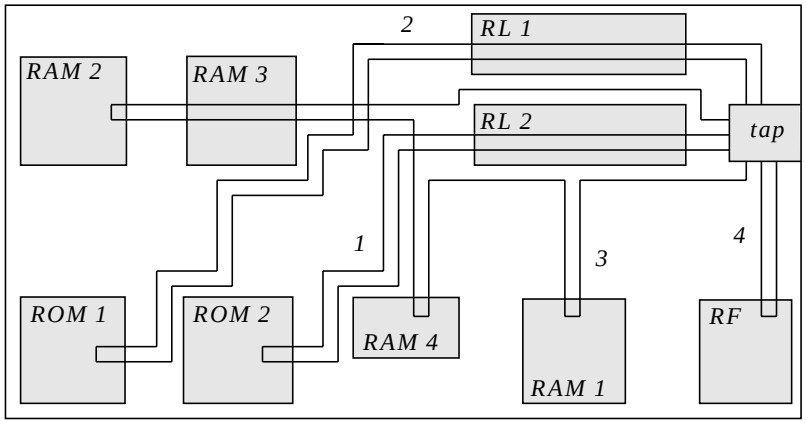


Figure 5.19: Test access mechanism design using our heuristic with initial sorting based on power.

Four test buses are required for the solution and the tests are scheduled on the test buses as shown in Figure 5.18. The test access mechanism is routed as shown in Figure 5.19. The test application time is 262 and the length of the test access mechanism is 360.

Our approach using an initial sorting of the tests based on time and power×time results in the same solution and it was produced within one second. The test bus schedule is shown in

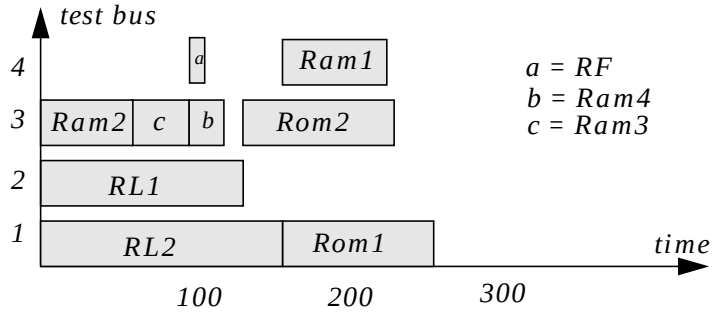


Figure 5.20: Test bus schedule achieved using our heuristic on ASIC Z using initial sorting based on time and power×time.

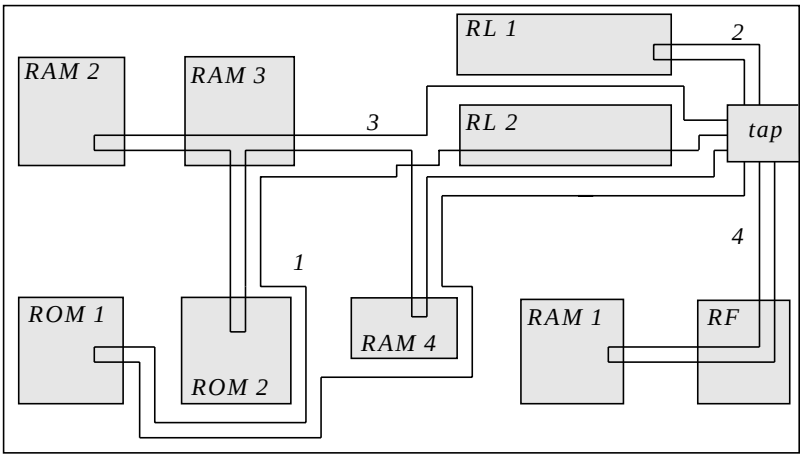


Figure 5.21: Test access mechanism design using our heuristic with initial sorting based on time and power×time sort.

Figure 5.20 and the routing of the test buses are shown in Figure 5.21. The test application time for the solution is 262 and the length of the test buses is 300.

For the SA we used initial temperature(TI)=500, temperature length(TL)=500 and α=0.99. The SA was running for 865 sec-

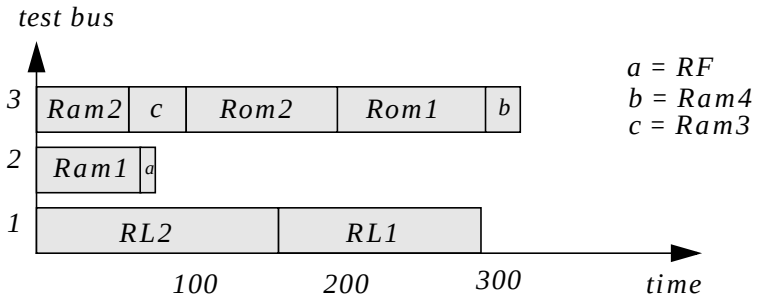


Figure 5.22: Test bus schedule achieved using Simulated annealing.

onds and the solution was produced with a test application time of 326 and a test access mechanism at a cost of 180. The test bus schedule is shown in Figure 5.22 and the design of the test access mechanism is shown in Figure 5.23.

All experimental results are presented in Table 5.12. All the initial solutions provided a test application time of 262 which is better than the solution produced by SA. However, the bus solution produced by SA is better than the solution produced by our approach with initial sorting of the tests based on time and power×time.

This experiment shows the problem and the importance of combining the two costs for test time and test access mechanism to a single value. In the experiment above we have used one (1) as the balancing factor which means that the cost is computed as: test time + 1×test bus length.

We have compared the effect of different cost balancing factors between test application time and test access mechanism and we run the SA with an initial temperatur(TI)=400, temperature length(TL)=200 and α=0.95. The optimal test application time is 262 and the optimal test access bus is 120. However, these are not possible to achieve at the same time. The test application time with an optimal bus (120) would require all tests to be scheduled in sequence resulting in a test time of 699.

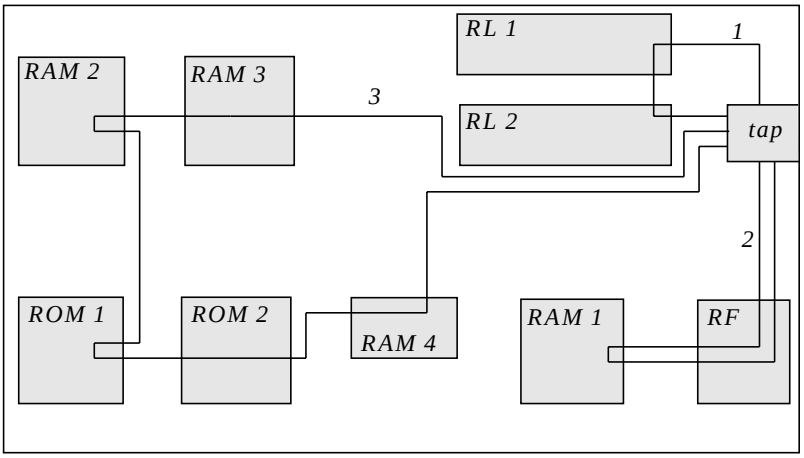


Figure 5.23: Test access mechanism design using Simulated annealing.

The results from the experiments are collected in Table 5.13. The cost improvement increases with the cost balancing factor. For instance, with a cost factor of one (1) the improvement using SA is 11% but when the cost factor is four (4) the improvement is 52%. However, it should be noted that the test application time increases. For instance, the solution produced by SA with a cost factor of 4 has a test application time 35% higher than the initial solution.

Approach	Test time	Test bus	Total cost	Difference to SA	Cpu time (sec.)
Our Simulated annealing	326	180	506	-	865
Our heuristic (power)	262	360	622	22.9%	1
Our heuristic (time)	262	300	562	11.1%	1
Our heuristic (power×time)	262	300	562	11.1%	1

Table 5.12: Results on ASIC Z.

Approach	Test time	Test bus	Factor	Total cost
Our heuristic (power×time)	262	300	1	562
Our SA	326	180	1	506
Difference to SA	-20%	67%		11%
Our heuristic (power×time)	262	300	2	862
Our SA	326	180	2	686
Difference to SA	-20%	67%		26%
Our heuristic (power×time)	262	300	3	1162
Our SA	405	160	3	885
Difference to SA	-35%	88%		31%
Our heuristic (power×time)	262	300	4	1462
Our SA	405	140	4	965
Difference to SA	-35%	114%		52%

Table 5.13: Comparing the balance factor between test time and test access mechanism cost.

5.4.3 EXPERIMENT TWO ON ASIC Z

The experiment performed in this section use the ASIC Z, see Appendix A.4, and use the same assumptions as in Section 5.4.2. The difference is that idle power is considered in this experiment.

The solutions using our approach with an initial sorting of the tests based on power are shown in Figure 5.24 and in Figure 5.25. Four test buses are required for the solution and the tests are scheduled on them as shown in Figure 5.24. The test access mechanism is routed as shown in Figure 5.25. The solution results in a test application time of 300 and a length of the test access mechanism is 360 and the solution was computed within one second.

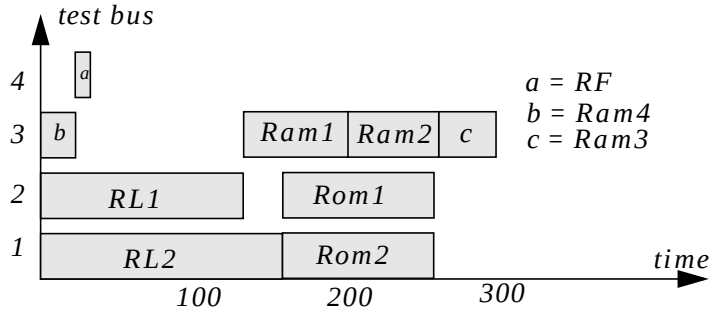


Figure 5.24: Test bus schedule achieved using our heuristic on ASIC Z using initial sorting based on power.

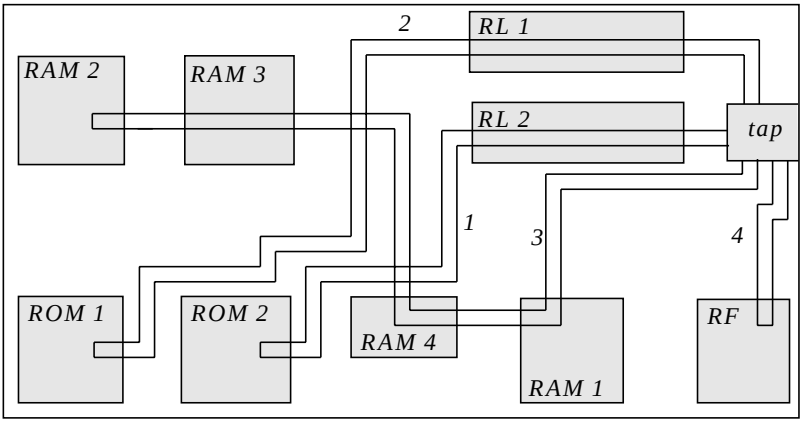


Figure 5.25: Test access mechanism design using our heuristic with initial sorting based on power.

For our approach with an initial sorting of the tests based on time and power×time, the solutions are the same, see Figure 5.26 and Figure 5.27. Four test buses are required for the solution and the tests are scheduled on them as in Figure 5.26. The test access mechanism is routed as in Figure 5.27. The total test application time is 290 and the length of the test access mechanism is 360.

The SA produces a solution with a test bus schedule as shown in Figure 5.28 where the test access mechanism is routed as shown in Figure 5.29. The test application time for the solution

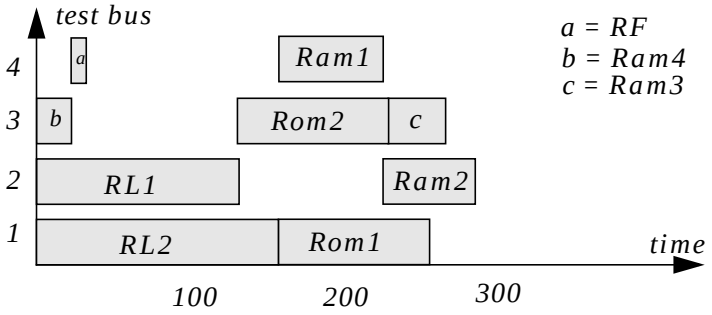


Figure 5.26: Test bus schedule using our heuristic on ASIC Z using sorting based on time and on power×time.

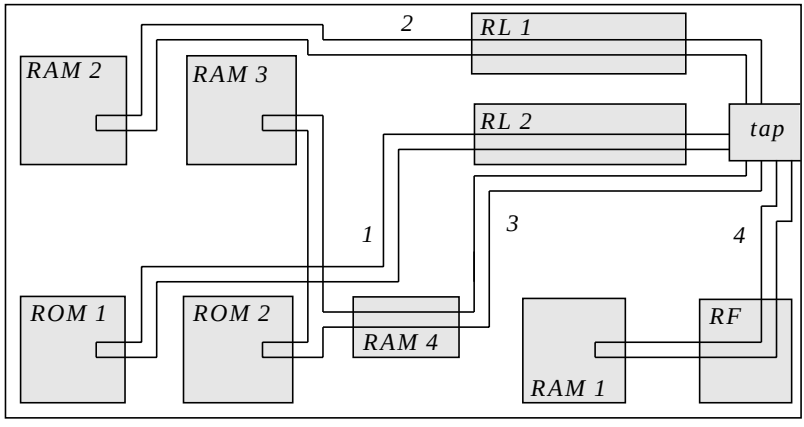


Figure 5.27: Test access mechanism design using our heuristic with sorting based on time and on power×time.

is 334 and the cost of the test access mechanism is 180. SA was running for 855 seconds with an initial temperature of 300, a temperature length of 300 and α=0.97.

All results from this experiment are collected and presented in Table 5.14. For instance, our approach with sorting based on power is 23.6% from the solution produced by SA. The test access mechanism in the solution produced by SA is 80% better than our solution. However, the test application time in the SA solution is 10% worse.

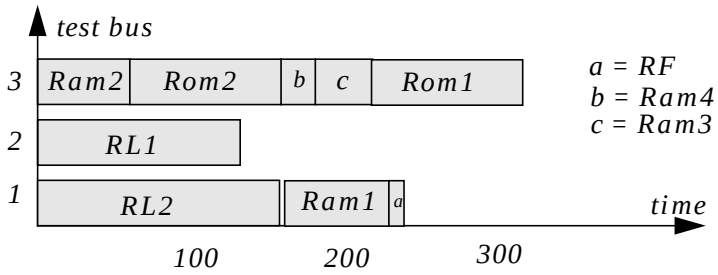


Figure 5.28: Test bus schedule achieved from SA.

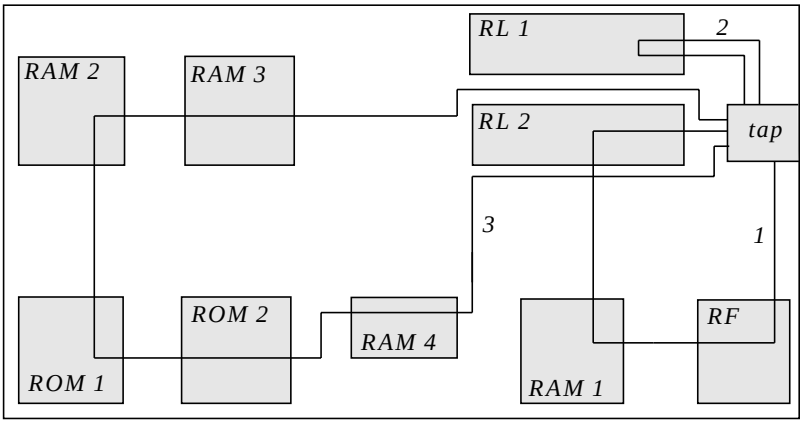


Figure 5.29: Test access mechanism design produced by Simulated annealing.

Approach	Test time	Diff. to SA	Test bus	Diff. to SA	Total cost	Diff. to SA	Cpu (s)
Our SA	334	-	180	-	514	-	855
Our heuristic (power)	300	-10.2%	360	100%	660	28.4%	1
Our heuristic (time)	290	-13.2%	360	100%	650	20.9%	1
Our heuristic (power×time)	290	-13.2%	360	100%	650	20.9%	1

Table 5.14: Experimental results on ASIC Z.

5.4.4 EXPERIMENTS ON EXTENDED ASIC Z

The Extended ASIC Z, see Section A.5, is an extension of ASIC Z, see Section A.4. The design consists of 9 cores which all are tested by three tests, where two tests are for the logic at each core and one test checks the interconnections.

The experimental results are presented in Table 5.15. Our approach with an initial sorting of the tests based on power produces a solution with a test application time of 313 and a cost of the test access mechanism of 720. The total cost is 1033. The solutions with the tests initially sorted based on time and power×time give solutions with test application time of 287 and a cost of the test access mechanism of 660. The solution produced by SA optimization has a test application time of 270 and a cost of the test access mechanism of 560 where TI=TL=200 and α=0.97.

The difference in total cost using our approach compared to SA is 14.1% for the approach with time and power×time sorting. In more detail, the test access mechanism is minimized from 660 to 560. However, the test application time is higher for the solution produced with SA.

The SA optimization was running for one hour and the optimal solution was found after 4549 seconds while the computational cost using our approach was less than a second.

Approach	Test time	Diff. to SA	Test bus	Diff. to SA	Total cost	Diff. to SA	Cpu (s)
Our SA	270	-	560	-	830	-	4549
Our heuristic (power)	313	15.9%	720	28.6%	1033	24.5%	1
Our heuristic (time)	287	6.3%	660	17.9%	947	14.1%	1
Our heuristic (power×time)	287	6.3%	660	17.9%	947	14.1%	1

Table 5.15: Results on Extended ASIC Z.

5.4.5 EXPERIMENTS ON ERICSSON

The Ericsson design, described in Appendix A.7, consists of 170 tests. We use the design as it is described and tests are allowed to start even if all tests in a session are not completed.

Our approach with an initial sorting of the tests based on power results in a solution with a test application time of 37336 and a test access mechanism cost of 8245. This solution took 81 seconds to produce. Using our approach with tests sorted according to time gives a solution with a test application time of 34762 and a test access mechanism cost of 9350 and 79 seconds was required to produce the solution. Our approach with tests initially sorted based on power×time gives a solution with a test application time of 34762 and a test access mechanism cost of 8520 and the solution was produced after 62 seconds. The SA optimization produced a solution with a test application time of 33082 and a bus cost of 6910. The optimization was aborted after 15 hours.

All experimental results are presented in Table 5.16 and the computational costs are in Table 5.17. The total cost (test time and size of test access mechanism) are in all cases reduced when our approach is compared to the solution produced by SA. For instance, our approach with sorting based on power×time is in respect to test application time reduced from 34762 to 33082, which is an improvement of 5.1%. Furthermore, the test access mechanism is reduced from 8520 to 6910, which is a reduction of 23.3%. However, the time required for the SA was 15 hours, while our approach produced solutions within two minutes.

Approach	Test time	Diff. to SA	Test bus	Diff. to SA	Total cost	Diff. to SA
Our SA	33082	-	6910	-	46902	-
Our heuristic (power)	37336	11.4%	8245	19.3%	53826	14.8%
Our heuristic (time)	34762	5.1%	9350	35.3%	53462	14.0%
Our heuristic (power×time)	34762	5.1%	8520	23.3%	51802	10.4%

Table 5.16: Results on Ericsson.

Approach	Computational cost
Our heuristic (power)	81 seconds
Our heuristic (time)	79 seconds
Our heuristic (power×time)	62 seconds
Our Simulated annealing	15 hours

Table 5.17: Computational cost for the experiments on Ericsson.

5.5 Test Parallelization

The System L is an industrial design, see Appendix A.6, where no data is available for test D, G and F. They are therefore excluded from the experiments.

The 15 tests are scheduled by a designer as shown in Figure 5.10 with a test application time of 1592 time units. Our approach with an initial sorting based on power is in Figure 5.11 and the test application time is 1077.

Our approach with initial sorting based on power allowing test parallelization is shown in Figure 5.30. The test application

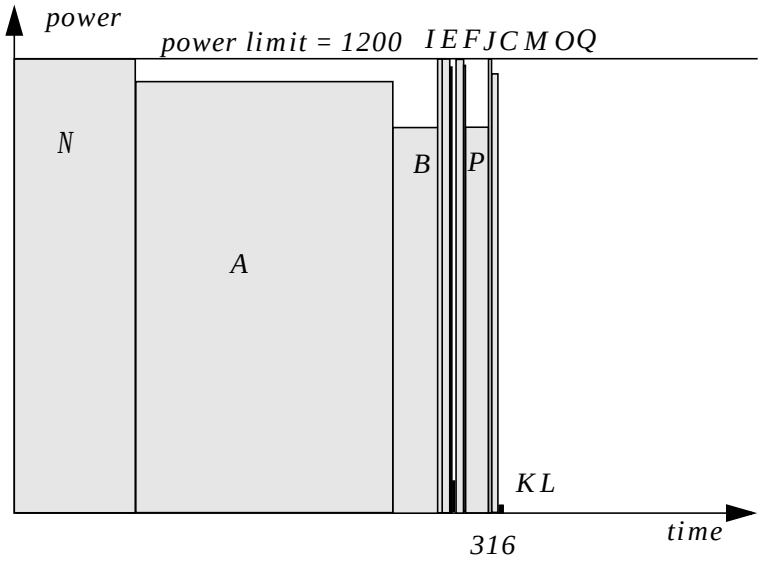


Figure 5.30: Test schedule using our heuristic with initial sorting based on power and allowing test parallelization on System L.

time for it is 316. Our approach with initial sorting based on time and power×time also results in a test application time of 316. The solution was computed within a second. The SA produced a solution of 316 with initial temperature (TI)=200, temperature length(TL)= 200 and α=0.95. The SA optimization was running for 38 seconds.

All results from the experiments are presented in Table 5.18. The designer's solution from the test schedule, see Section 5.2.8, was 1592. In Section 5.2.8 we reported a test schedule achieved with our approach at 1077. These results did not allow test parallelization. When test parallelization is allowed the test application time can be reduced to 316 which is more than 400% better than the designer's solution.

Approach	Test time	Difference to SA	Cpu time (sec.)
Our SA	316	-	38
Designer	1592	403.8%	-
Our heuristic (power sort)	1077	240.8%	1
Our heuristic (time sort)	1077	240.8%	1
Our heuristic (power×time sort)	1077	240.8%	1
Our heuristic with parallelization (power sort)	316	0%	1
Our heuristic with parallelization (time sort)	316	0%	1
Our heuristic with parallelization (power×time sort)	316	0%	1

Table 5.18: Experimental results on System L.

5.6 Test Resource Placement

We have made experiments where the placement of test resources are optimized in order to minimize test time and the cost of the test access mechanism.

5.6.1 EXPERIMENT ON ASIC Z

In this experiment we try to find the optimal placement for the test access port. We use the ASIC Z design where each block has its own dedicated BIST structure, see Appendix A.4. However, in this experiment we assume that all tests are applied using an external tester. We assume that several tests may use the tester at the same time and idle power was not considered.

In the experiment, we let our TS optimization implementation search for the best placement and the result is point *m*. We then made experiments on all of the points, *a* to *t*, in Figure 5.31 and the results are in Table 5.19 where point *m* is the one with lowest cost. This verifies that TS finds the best solution.

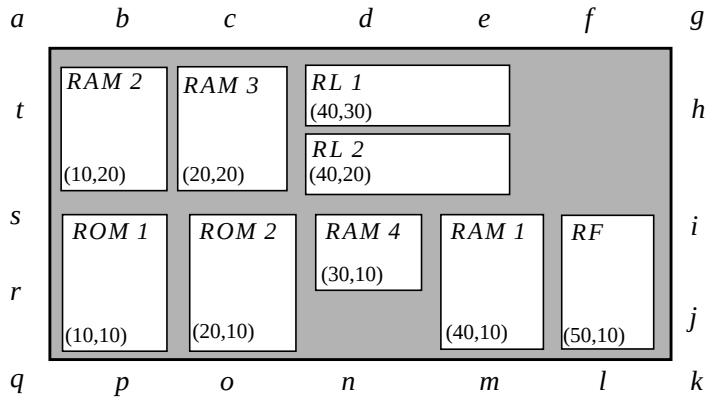


Figure 5.31: TAP placement in ASIC Z.

5.6.2 EXPERIMENT ON EXTENDED ASIC Z

In this experiment we try to find the best placement of on-chip test resources which minimizes test application time and the cost of the test access mechanism. We have used our test resource placement algorithm on Extended ASIC Z, see Appendix A.5, and we do not consider idle power and we allow tests to start as soon as possible.

In Extended ASIC Z the RAM memories share one BIST structure and the ROM memories share one. The other blocks have their own dedicated BIST structure. The solution using our test resource placement algorithm results in a test schedule at a test application time of 313 and a cost of the test access mechanism of 720. The total cost is 1033, see Table 5.20.

We use a TS optimization to search for a better placement of the on-chip test resources and TS finds a placement where the test application time is 313 and the cost of the test access mechanism is 620 resulting in a total cost of 933. The difference compared to our algorithm is 10.7% and the TS optimization took 220 seconds.

We also performed experiments with TS where each individual move is further optimized using SA. This optimization

Point	Placement		Test time	Test bus	Total cost
	x	y			
a	0	40	262	540	802
b	10	40	262	460	722
c	20	40	262	400	662
d	30	40	262	400	662
e	40	40	262	420	682
f	50	40	262	480	742
g	60	40	262	560	822
h	60	30	262	480	742
i	60	20	262	440	702
j	60	10	262	440	702
k	60	0	262	460	722
l	50	0	262	340	602
m	40	0	262	300	562
n	30	0	262	320	582
o	20	0	262	340	602
p	10	0	262	420	682
q	0	0	262	500	762
r	0	10	262	420	682
s	0	20	262	400	662
t	0	30	262	440	702

Table 5.19: Experimental results on ASIC Z.

results in a solution where the test application time is 293 and the cost of the test access mechanism is 420. The total cost is 713. Our initial solution is 44.9% from this solution and the TS without SA is 30.9% from it. This solution was found after 5606 seconds or 91779 iterations and the optimization took almost 3 hours. For the SA we used initial temperature of 5, temperature length of 10 and α=0.70.

Approach	Test time	Test bus	Total cost	Difference to TS + SA	Cpu time (sec.)
Our placement approach(power sort)	313	720	1033	44.9%	1
Our TS+ our heuristic(power)	313	620	933	30.9%	220
TS+SA (power)	293	420	713	-	5606

Table 5.20: Experimental results on Extended ASIC Z.

For each of the placements above we evaluated the test application time and the cost of test access mechanism using more extensive SA optimization. We used an initial temperature of 200, a temperature length of 200 and α=0.97 running for 75 minutes. The experimental results are presented in Table 5.21. Our initial solution is 24.5% from the solution by SA. The TS placement is 10.2% from the SA solution and finally the TS and the SA placement is 2.9% from SA solution.

Approach	Test time	Test bus	Total cost	Difference to SA
Our placement heuristic (power sort)	313	720	1033	24.5%
Our Simulated annealing	270	560	830	-
Our Tabu search+ our heuristic (power)	313	620	933	10.2%
Our Simulated annealing	278	560	838	-
Our TS+our SA+ our heuristic(power)	293	420	713	2.9%
Our Simulated annealing	313	380	693	-

Table 5.21: Experimental results on Extended ASIC Z.

5.7 Summary

We have in this chapter by experiment demonstrated the usefulness of our approaches. The experiments have been performed using several benchmark examples and industrial designs.

We have compared our approach with other techniques on test scheduling on design Kime and System S and for both examples our approach finds the optimal solution. We have, on design Muresan, ASIC Z, Extended ASIC Z, System L and Ericsson, performed experiments where test application time is minimized while test conflicts and test power consumption are considered. Our approach finds optimal or near optimal solutions at a very low computational cost.

We performed experiments using our approach to design the test access mechanism. In this experiment we used System S and our approach finds the optimal solution. We have also performed experiments on test scheduling and test access mechanism on System S, ASIC Z, Extended ASIC Z and Ericsson.

We have performed experiments on System L where we combine test scheduling and test parallelization. The results show that by combining test scheduling and test parallelization the test application time can be minimized compared to considering these tasks separately.

Finally, we performed experiments where test resource placement is considered in order to achieve minimal test time and cost for the test access mechanism.

Testability Analysis and Enhancement Technique

PART III

Chapter 6
Introduction and 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 terms of additional delay and increased silicon area. In order to maximize the testability and to minimize design degradation, the testability of a design must be carefully analyzed and, based on the analysis result, hard-to-test parts are selected for testability improvement.

Several testability metrics have been developed and reported in the literature, as well as techniques to improve testability. Due to the increase in design complexity, testability analysis and enhancement approaches have been proposed for different abstraction levels. We provide an overview of metrics for testability analysis in section 6.1. We describe different techniques to improve testability in section 6.2. In section 6.3 we summarize the discussion.

6.1 Testability Analysis

In this section we present 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.

6.1.1 GATE LEVEL TESTABILITY ANALYSIS

The early work in testability analysis is usually carried out at the gate level [Gol79], [Gol80], [Gup90], [Par93], [Par95], [Abr91]. Several testability metrics at this level are based on the concepts of controllability and observability. An example of such a metric is to attach a testability value to each line (wire) of a design in such a way that a line close to a primary input is easily controlled and a line close to a primary output is easily observable. Even though such a distance-based metric is unsophisticated, it provides fairly good 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-controllability*, *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 shown in Figure 6.1. 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

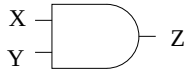


Figure 6.1: A 2-input AND-gate.

depends on the controllability of the inputs of the AND-gate. Rutman gives a formula for this:

$$C_0(Z) = min\{C_0(X), C_0(Y)\} + 1 \tag{6.1}$$

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

For 1-controllability of line Z both X and Y must be 1 and the formula becomes:

$$C_1(Z) = C_1(X) + C_1(Y) + 1 \tag{6.2}$$

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

$$C_2(X) = C_2(Z) + C_1(Y) + 1 \tag{6.3}$$

where $C_2(X)$ indicates the observability at line X.

Using the basic ideas in the formulas above, it is possible to develop formulas 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:

$$S_i = \sum_{l \in L} C_i(l) \tag{6.4}$$

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

The total testability of a circuit is then defined as:

$$S = \sum_{i=0}^{2} k_i \times S_i \tag{6.5}$$

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 6.2, the signals B and C can never be set to the same

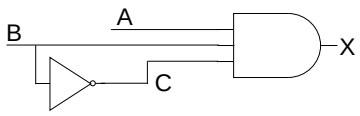


Figure 6.2: Problems in controlling line X.

value, so the correct value of $C_1(X)$ should show that setting X=1 is impossible.

Based on Rutman's controllability and observability measures Chen and Breuer introduced the concept of 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 calculate the relative improvement for selected points, which is the difference between the testability for the circuit with no DFT compared with applying DFT:

$$\frac{\partial S_i}{\partial S_j(w)} = S_i(S_j(w)=0) - S_i(S_j(w)=1) \qquad (6.6)$$

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 6.3 is the minimum of the 0-controllability values on the inputs.

The observability cost $O(l)$ is the number of clock cycles required to propagate the value of line l to a primary output, and

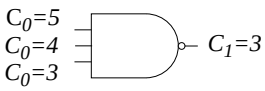


Figure 6.3: Calculating the 1-controllability of a NAND-gate.

it consists of two measures, sequential distance and enabling cost. The sequential cost measure, $D(l)$, indicates 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

$$O(l) = D(l) + E(l) \tag{6.7}$$

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

$$DET(f) = max\{C_{\bar{v}}(l), E(l)\} + D(l) \tag{6.8}$$

The total testability cost for a circuit, *i.e.* the total cost function (*TCF*), is calculated as:

$$TCF = \sum_{f \in F} DET(f) \tag{6.9}$$

where F is the set of target faults.

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, 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:

$$SCD(FF_i) = \sum_{j=1}^{m} \{SC(FLT_j) - SC_i(FLT_j)\} \quad (6.10)$$

$$SOD(FF_i) = \sum_{j=1}^{m} \{SO(FLT_j) - SO_i(FLT_j)\} \quad (6.11)$$

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

$$TD(FF_i) = SCD(FF_i) + SOD(FF_i) \quad (6.12)$$

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

The major drawback with such approaches as sensitivity analysis is 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].

6.1.2 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 logic operation in a design usually reduces the probability of controlling/observing a line embedded in the design.

Chen and Menon presented a testability analysis technique based on combinational controllability (*CC*), sequential controllability (*SC*), combinational observability (*CO*) and sequential observability (*SO*) [Che89]. The controllability metrics are further divided into two components, 1 and 0 controllability. Thus, there are six parameters associated with each line 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 in 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 *et al.* [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 is defined by the controllability transfer factor (CTF). For 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.

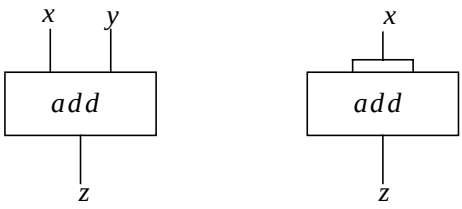


Figure 6.4: Example where p=0 and p=n using the metric proposed by Flottes *et al.*

Another approach based on probabilities is proposed by Flottes *et al.* [Flo97]. The focus there 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:

$$T_c = \frac{(C_2 - C_1) \times p}{n} + C_1 \qquad (6.13)$$

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 6.4 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 another, 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 above approach is that no loops are accepted in the data path and that the controller is assumed to be testable. However, a separately testable controller and sepa-

rately testable data path do not imply that the combination is testable [Dey95].

Lai *et al.* propose also 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, known as 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:

$$R_c \approx C_1 \times \frac{R_a + R_b}{2} + C_2 \times \frac{M_a + M_b}{2} + C_3 \tag{6.14}$$

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:

$$T_c = \frac{T_a + T_b}{2} \tag{6.15}$$

The test generation process is a process of justification and propagating values from a primary input to a primary output. In Figure 6.5 a multiplication of the two variables y and z is shown. The propagation of a value from input y to output 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 met-

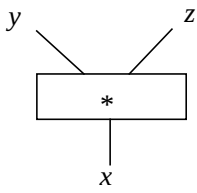


Figure 6.5: Propagation of values from inputs to outputs.

rics proposed by Gu *et al.* do not consider the justification process when the observability is calculated [Gu95b].

6.1.3 BEHAVIORAL-LEVEL TESTABILITY ANALYSIS

The size of digital systems and their complexity are pushing the trend towards design at higher abstraction levels. A design grows significantly in complexity when it is synthesized into the logic level because more implementation information is added to the design. This can make it harder to extract vital information from it when, for instance, we want to find hard-to-test parts.

Until recently, 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. Therefore, more accurate information about its testability can be extracted provided that the complexity is not too high.

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 the physical components are allocated.

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 design 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) ones [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:

$$EFF(N) = \frac{\left[\sum_{n_t \in NCC_t} BitSize(n_t) - \sum_{n_m \in NCC_m(N)} BitSize(n_m)\right]}{BitSize(N)} \qquad (6.16)$$

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 selected as a test point, and
- $BitSize(N)$ 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 are 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.

An approach where a *value range propagation* technique is used for testability analysis is proposed by Seshadri and Hsiao [Ses00]. A *static single assignment* (SSA) representation is used to calculate controllability and observability metrics from a behavioral VHDL specification. An example of a behavioral VHDL specification is shown in Figure 6.6. and its SSA in Figure 6.7.

The SSA rules are:

1. Each definition of a variable, which occurs at places where the variable receives a new value, is assigned a unique name.
2. A Φ–function is used to combine results at points in a program. For instance, $x=\Phi(p,q,...)$ where x gets the value p if the control flows into the basic block is via the first path, q if the second path is selected, and so on.
3. Each use of a variable makes use of exactly one name generated from the rules above. Use of a variable occurs when it is needed in the definition of itself or another variable.

For instance the variables in Figure 6.7 are x, y and l where x has 5 defines, 5 uses, and 2 joins, which result in seven SSA components, x_0 to x_6.

The notation to describe an m weighted value:

$$\{W_i[L_i;U_i;S_i], \ldots\} \tag{6.17}$$

where W_i is the probability of the corresponding range; L_i and U_i gives the lower and the upper bounds of the given range; S_i is the size of steps taken when going from the lower to the upper bound; and $i = 1,2,3...m$. For instance, x_0 is 1.0[0:0:0] at initialization, see Figure 6.7 at block b_0. To extract information at the join, block b_1 in Figure 6.7, at the for-loop controlled by x, x_1 is allocated. The SSA for x_1 is calculated by merging values from x_0 and x_6. However, x_6 is not computed yet and it may require sev-

```
library ieee;
use ieee.std_logic_1164.all;
package my_data_types is
  subtype data is integer range 15 downto -15;
end my_data_types;

use work.my_data_types.all;
entity example is
  port( clk: in bit;
        z: out data );
end example;

architecture simple of example is
begin
  process
    variable x: integer range 0 to 15;
    variable y: data;
    variable l: boolean;
    begin
      y := 0; z <= 0; l := false;
      for x in 0 to 9 loop
        wait until ( clk'event) and (clk='1');
        l := true;
        if ( x <= 1 ) then
          y := x - 1;
        else
          y := x + 1;
        end if;
        z <= y;
      end loop;
      l := false;
  end process;
end simple;
```

Figure 6.6: A behavioral VHDL description.

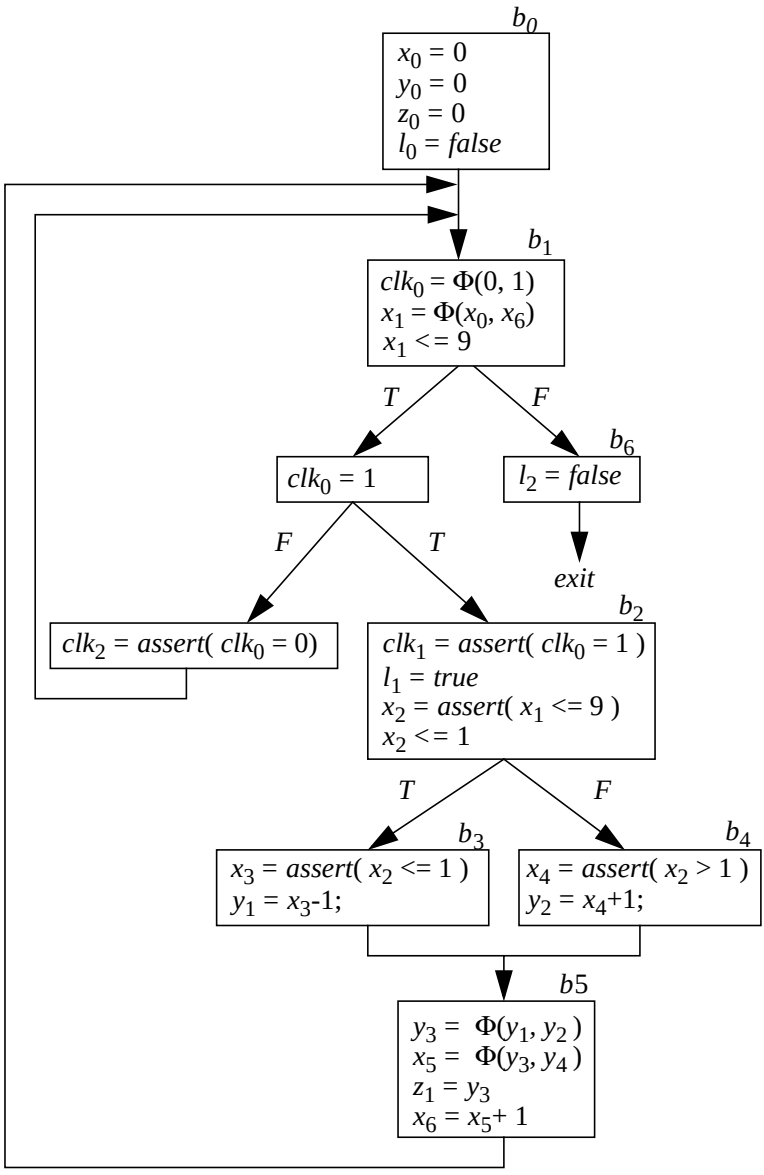


Figure 6.7: The SSA representation of the example in Figure 6.6.

eral iterations through the SSA flow graph to determine the value for x_1. The SSA variables and weighted value ranges for the given example are presented in Table 6.1.

In the approach by Seshadri and Hsiao testability is the combination of controllability and observability for the variables in the behavioral specification.

X		Y		Z	
SSA var	**WVR**	**SSA var**	**WVR**	**SSA var**	**WVR**
x_0	1.0[0:0:0]	y_0	1.0[0:0:0]	z_0	1.0[F:F0]
x_1	1.0[0:10:1]	y_1	1.0[-1:0:1]	z_1	1.0[T:T:0]
x_2	1.0[0:9:1]	y_2	1.0[3:10:1]	z_2	1.0[F:F:0]
x_3	1.0[0:1:1]	y_3	{0.2[-1:0:1],		
x_4	1.0[2:9:1]		0.8[3:10:1]}		
x_5	1.0[0:9:1]				
x_6	1.0[0:10:1]				

Table 6.1: SSA variables and weighted vaule ranges.

Controllability

Given a value range $W[L{:}U{:}S]$ of v_j and a value a in the range, the controllability value C_a of v_j is the difficulty in assigning a to v_j, C_a is computed as:

$$C_a = 1/P_a \tag{6.18}$$

where P_a is the probability of v_j assuming value a.

For instance, assume a 5-bit natural variable, each of the 32 values may occur at the same probability, *i.e.* $C_0{=}C_1{=}...{=}C_{31}{=}$ 1/(1/32)=32.

1. Assignment inside a loop structure

For branch structures with f paths and the probability from path k being P_k the resulting value ranges can be computed as:

$$\left\{P_k\left\{W_i^k\left\lfloor L_i^k;U_i^k;S_i^k\right\rceil\right\}\right\} \tag{6.19}$$

where k = 1, 2, 3,..., f and i=1,2,3...n.

The controllability value for each v_j at the kth fanout is:

$$C_i^k = \frac{1/(P^k \times W_i) + 1/((P^k \times W_i) \times (P_{loop})^{((U_i - L_i)/S_i)})}{2} \tag{6.20}$$

The density of v_j, denoted D for the kth path, corresponds to a value range as the number of unique values v_j can take in the given range to the total number of values it can take.

$$D_{v_j}^k = \sum_{i=0}^{n} \frac{ST_i + 1}{T} \tag{6.21}$$

where ST_i=((U_i - L_i)/S_i) and T is the total number of values a variable can take. For example if V is an 5-bit natural, T=32.

The controllability for the component SSA variable over its complete set of value ranges at path k is given as:

$$C_{v_j}^k = \sum_{i=0}^{n} \frac{\left(C_i^k \times ((U_i - L_i)/S_i + 1)/T\right)}{D_{v_j}^k} \tag{6.22}$$

The controllability for the entire component for *all* f paths is computed by:

$$C_{v_j} = \sum_{k=0}^{f} \frac{\left(C_{v_j}^k \times D_{v_j}^k\right)}{\sum_{k=1}^{f} D_{v_j}^k} \tag{6.23}$$

For simple assignments inside a loop, *i.e.* no Φ-function, let k=1 and P^k=1 since only one path exists. For instance consider

$y_2 = x_4 + 1$ in b_4 in Figure 6.7 where $P_{loop} = 0.91$, T=31 (-15 to 15) and y_2={1.0[3:10:1} the density D and the controllability C is given by:

$$D = \frac{ST+1}{T} = \frac{\frac{U-L}{S}+1}{T} = \frac{\frac{10-3}{1}+1}{31} = 0.258$$

$$C = \frac{1/W + 1/(1 \times (P_{loop})^{ST})}{2} = \frac{1/1.0 + 1/(1 \times 0.91^{7})}{2} = 1.468$$

2. Assignment Outside a Loop Structure

If an assignment occurs before the exit of a loop structure, set ST=0 and use formulas 6.20. Otherwise the controllability value for each v_j at the kth fanout is:

$$C_i^k = \frac{1}{P^k \times W_i \times (1 - P_{loop})^{iter}} \tag{6.24}$$

and the density is calculated using formula 6.21 with ST=0.

For simple assignments where only one path exists, k=1 and P^k=1. And the controllability and density for variables are calculated using formulas 6.21, 6.22, and 6.23.

3. Loop Structure

The controllability and the density for a variable at a loop are computed using Equation 6.20 and 6.21 where $STi=ST_{x1}$, k=1 and P^k=1.

Observability metrics

The observability O of a variable V measures the ease of propagating a value to a primary output.

Initially, the observability values for all non-output variables are set to infinity and to zero for all primary outputs. The following rules are then applied to calculate the observability:

1. Fan-outs of a branch structure with *f* fan-outs:

$$O_V = \frac{1}{max(P_k)} \tag{6.25}$$

   where k=1, 2,..., f.
2. Inside a loop structure with a probability P_{loop} of staying in the loop:

$$O_V = \frac{1}{P_{loop}} \tag{6.26}$$

3. In a basic block with q in-edges each with the probability P_q:

$$O_V = \frac{1}{max(P_q)} \tag{6.27}$$

If one in-edge to a block is from a loop structure of *iter* iterations, then the value P_q corresponding to this in-edge is $(1\text{-}P_{loop})^{iter}$. The observability values for each of its n component SSA variables are calculated and the final observability is:

$$O_V = min\left(O_V^i\right) \tag{6.28}$$

where i=1, 2,..., n.

Testability

The testability metrics T_V of a variable V is

$$T_V = 0.8 \times \frac{\sum_{i=0}^{j} C_{v_i}}{\sum_{k=0}^{j} D_{v_k}} + 0.2 \times Ov \tag{6.29}$$

where v_i ($0 \le i \le j$) and v_k ($0 \le k \le j$) are both SSS component variables not used in a Φ-function assignment. For v_k the set should also contribute to at least one unique value to D_V.

The controllability C_y is computed as:

$$Cy = C_{y_0} + C_{y_3} = 1.0 + 2.521 = 3.521$$

where y_1 and y_2 used in the Φ-function of y_3 and therefore excluded.

The density D_y is given as below since the value range of y_0 is a subset of the value range for y_3:

$$Dy = D_{y_3} = 0.323$$

Then the testability T_y is computed as:

$$T_y = 0.8 \times \frac{C_y}{D_y} + 0.2 \times O_y = 0.8 \times \frac{3.521}{0.323} + 0.2 \times 1.099 = 8.94$$

(The testability T_y is computed as 8.72 in the paper which is achieved when the observability is excluded [Ses00]).

The variables are ranked in decreasing order of their controllability to density ratio with the hardest to control variable first. For the variables shown in Figure 6.6 the ranking is *l*, *y*, *x* (2.87^{10}, 10.90, 4.05). The testability metrics *l*, *y*, *x* (∞, 8.72, ∞).

For testability improvement partial scan is assumed except when a variable is difficult to control but easy to observe.

If a variable is difficult to control but easy to observe, it is a candidate for overloading or test point insertion. For all other variables (hard to observe, or hard to control and observe) partial scan is used.

The major drawback of the approach is that the number of loops has to be known in advance, *i.e.* loops must be able to be unrolled to a sequence of statements, which limits the usability of the approach.

Furthermore, the control state register does not correspond to any variable in a behavioral VHDL specification and there is no testability metric for it.

Another drawback is that for testability differences between different operations are not covered. For instance, for adders and multipliers it is usually assumed that a multiplier is harder to test compared to an adder. However, in the approach by Seshadri and Hsiao [Ses00] this is not the case if simple assignments (ST=0) are assumed with the same input value range. The testability for both is given by:

$$T = 0.8 \times \frac{1/W}{1/T} + 1 \qquad (6.30)$$

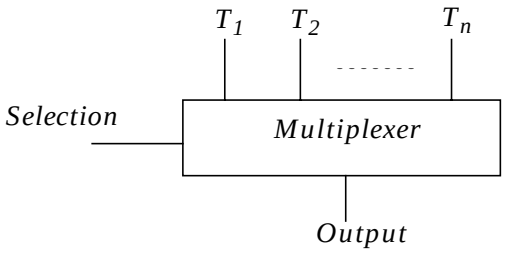


Figure 6.8: Example using a multiplexer to reduce the number of primary outputs.

6.2 Testability Improvement

Several techniques have been developed to improve the testability of a digital circuits. 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.

6.2.1 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 is to add a multiplexer to reduce the number of primary outputs. In Figure 6.8 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 obser-

vation 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 a multiplexer and a 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 pattern has to be shifted in [Abr90]. Several techniques have been proposed to reduce the costs introduced when using scan designs. For example, 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 a multiplier as shown in Figure 6.9. If the signal *x* is uncontrollable, we can add a test multiplexer as shown in shadow and multiplex *x* with a test input. In this way, we can easily control the contents of register *x*.

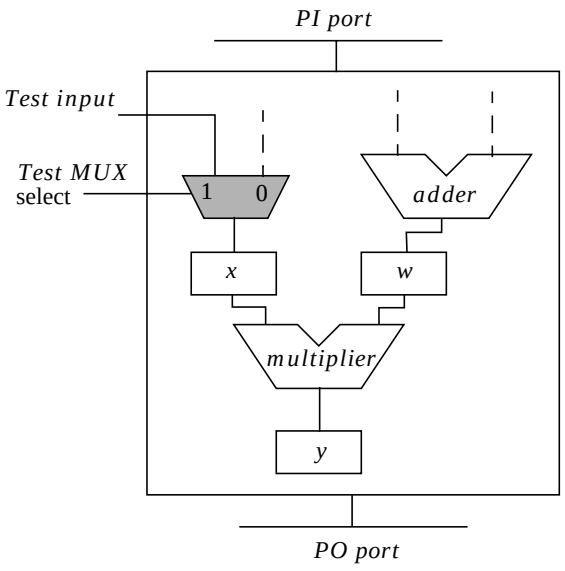


Figure 6.9: Multiplexing with a constant.

6.2.2 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, and
- 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 and then allocation are 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 [Yan98]. 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 [Yan98].

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

$$\Delta C = \alpha \times \Delta E + \beta \times \Delta H \tag{6.31}$$

where α and β are two coefficients defined by the designer. ΔE is the incremental execution time and ΔH is the incremental hardware cost, which compares the difference between execution time and cost of the design after transformation and before transformation. The hardware cost for the data path is computed as:

$$H = \sum_i Area(V_i) + \sum_i Len(A_j) \times Wid(A_j) \tag{6.32}$$

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 used to represent the control flow of the given design.

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 by Mujumda *et al.* eliminates as many self-loops as possible during 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.

6.2.3 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.* [Yan98] where the testability analysis considers the data path as well as the controller. Further, a quantitative measure is proposed to determine 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 6.10 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 dis-

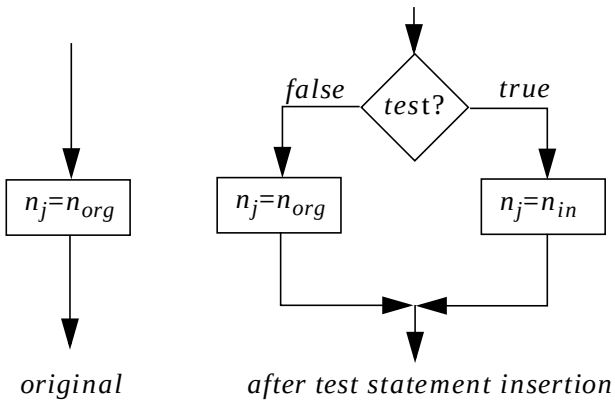


Figure 6.10: An example of test statement insertion.

tinguish between the test mode and the normal mode. In the test mode n_j is assigned the value of n_{in}, while in the normal mode, n_j is assigned the value of n_{org}.

The problem here 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 behaviors. 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 6.11, where the marked parts are added for the test. In Figure 6.12 the scheduled data flow graph is shown for the design in Figure 6.11, and in Figure 6.13 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 6.14.

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

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 [Hsu96a]. An example is given in Figure 6.15 where the transformations, T1, T2, T3 and T4 are found on the right-hand side.

6.3 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. At the behavioral level the functional properties of the design are 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

```
ENTITY example IS
  PORT (
    a, b, c, d : IN BIT_VECTOR(3 downto0);
    x : IN BIT_VECTOR(3 downto0);
    newx : IN BIT_VECTOR(3 downto0);
    test : IN BIT;
    out : OUT BIT_VECTOR(3 downto0)
    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 : INTEGER;
        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 6.11: An example of design-and-test behavior.

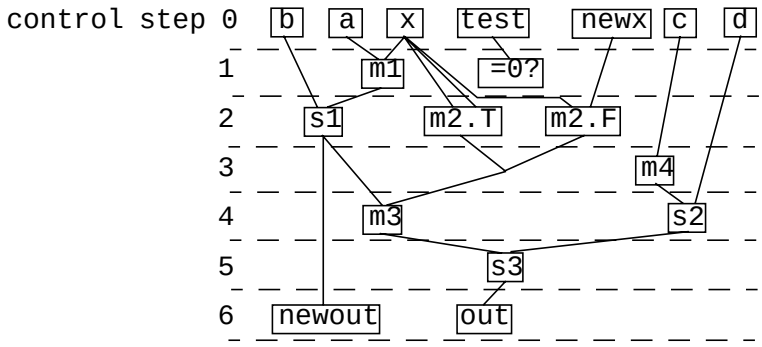


Figure 6.12: Scheduled data flow graph for the design-and-test behavior of Figure 6.11.

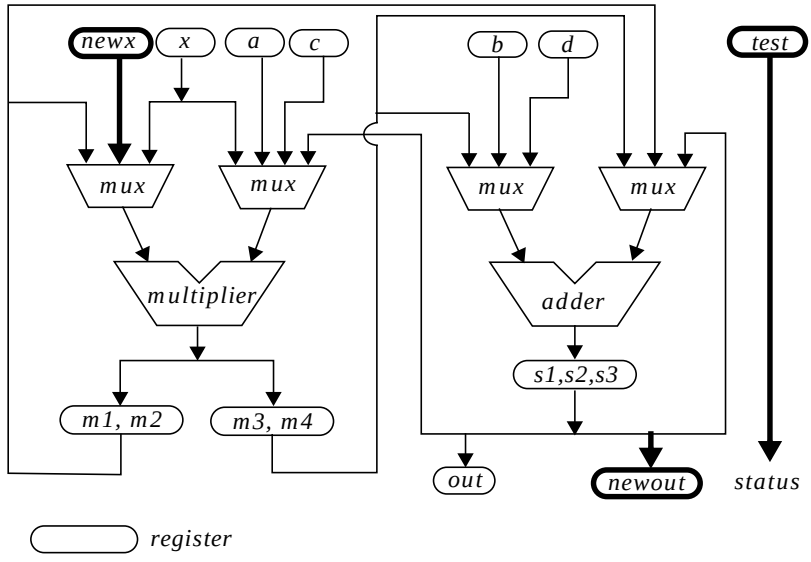


Figure 6.13: The data path for the testable behavior, with inserted elements in bold.

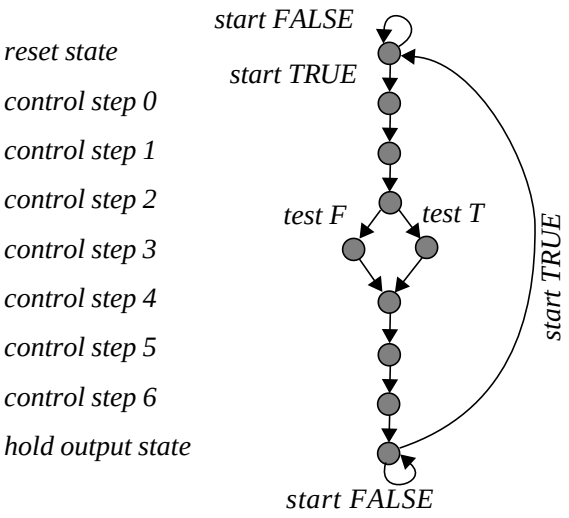
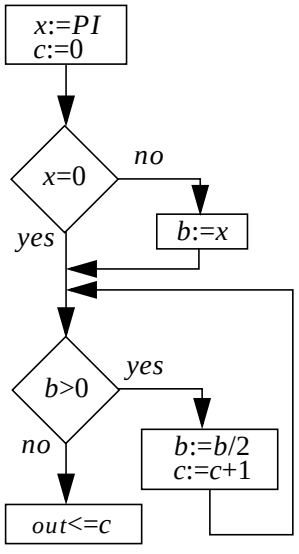


Figure 6.14: Controller with the embedded test controller.




```
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 6.15: Four types of controllability insertion in the high-level description.

Gu [Gu95b] has only a simple heuristic for loop estimation. Seshadri and Hsiao allow bounded loops where the number of iterations is known in advance [Ses00].

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 analyzing the ability to control and observe values on certain lines in the design and these methods aim to guide the designer to find hard-to-control and/or har-to-observe parts. There is usually a trade-off between accuracy in finding the hard-to-test parts and computational complexity. Several approaches such as sensitivity analysis have such a computational complexity that they are not applicable in practice for industrial designs [Abr90].

Even if all testability analysis method approaches provide guidance in detecting hard-to-test parts, no analysis approach has a good 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 provide 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. In experiments it has been shown that the existing testability analysis methods detect hard-to-test parts. This has been done mostly by performing experiments but in each approach only one DFT technique has been used. For instance Chen and Breuer use test point insertion [Che93], while Parikh and Abramovici [Par93][Par95] use partial scan. One exception here is the work by Seshadri and Hsiao which

proposes a technique for the selection of test points or scan for a register based on the testability of the variable [Ses00].

Another common drawback of the existing testability analysis techniques is that the heuristics for selecting hard-to-test parts only select one basic unit 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, since the introduction of a DFT technique usually improves not only the place where it is applied but also the global testability especially the testability of units in the neighbourhood of the target part. 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.

Considerable amount of work has been done at low abstraction levels defining testability analysis techniques and enhancement methods. However, due to increasing design complexity, new modelling techniques have been developed for higher abstraction levels. It is important, for these new abstraction levels, to develop techniques for analyzing testability and enhancing it. Furthermore, it is also important to develop techniques which can be integrated in the test synthesis process.

Chapter 7
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 7.1, the testability metrics are presented in section 7.2. An application of the metrics for partial scan selection is described in section 7.3, and the analysis algorithm is presented in section 7.4. Finally, we present experimental results in section 7.5, and conclusions in section 7.6.

7.1 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. Our assumption is based on the fact that many ATPG tools use randomly generated test vectors for finding many easily detected faults and then deterministically generated vectors for harder faults.

7.2 Behavioral Testability Metrics

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

7.2.1 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 7.1: $VR(l, v)$ 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 7.2: $defVR(v)$ 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 $[0:2^{16}-1]$.

The notation $|S|$ represents the number of elements for a set S. For instance if S={a,b,c} then $|S|$=3. For a variable v at a line l, the value range $VR(l, v)$ = [-10:10, 15:20] and we let $|VR(l,v)|$ = 21+5=26, *i.e.* the number of different possible values in the range.

Definition 7.3: 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 7.1, 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 7.1: An example of Variable Range and Relative Value Range.

7.2.2 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 metric 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 7.2, the output distribution for a 2-input 3-output adder is shown.

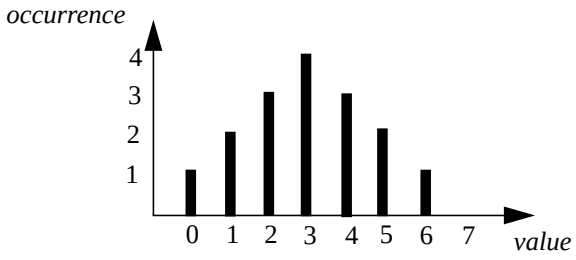


Figure 7.2: Output value (3-bit) distribution for a 2-bit addition.

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

$$Q(op) = \sum_{i=1}^{r} \frac{(x_i - n \times p_i)^2}{n \times p_i} \tag{7.1}$$

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 7.4: 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 7.1 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.

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 7.1: Q and OpT values for arithmetic operations.

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 7.5: 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 7.6: 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.

7.2.3 STATEMENT REACHABILITY

Some statements in a behavioral specification are more difficult to reach then 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.

```
(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 7.3: Condition scope.

Definition 7.7: 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 7.3 is the set $\{l_1, l_2\}$ and the false condition scope, $cs_f(c_1)$, for condition c_1 is the set $\{l_3, l_4\}$.

Definition 7.8: 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 set p_{max}=0.75 and p_{min}=0.25.

7.3 Application of the Behavioral Testability Metrics

The behavioral testability metrics defined above are not targeted at any particular DFT improvement technique. In this section we will show how the behavioral testability metrics can be used when the 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 scan variables and eventually mapped to scan registers.

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 7.3 to get its *RVR* [Lar97].

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

Definition 7.9: 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 coefficients 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 equal 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:

$$TH(l) = \frac{TH(l) \cdot (i - j)}{i} \qquad (7.2)$$

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.

7.4 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 7.4, where L is the set of lines in the behavioral specification, V is the set of variables, and 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 summarizes the operation testabilities for all arithmetic operations at a line. For each variable at the line the relative

```
TH_max=0;
for l=1 to |L| do begin
  for op = 1 to |Op_l| do begin
    TH(l)=TH(l)+(1-get_opt(op));
  end;
  if TH(l)>TH_max then TH_max=TH(l);
  for i=1 to |V| do begin
    VR(v_i)=get_variable_range(v_i);
    RVR(l,v_i)=|VR(l,v_i)|/|defVR(v_i)|;
    n_i=n_i+#times v_i is used at line l;
    m_i,l=1 if v_i 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 begin
  T(v_i)=T(v_i)+m_i,l/n_i*(α1*RVR(l,v)+
      α2*(1-TH(l)/TH_max+α3*SR(l));
end;
```

Figure 7.4: The testability analysis algorithm.

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.

7.5 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 7.5. 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 generates 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 7.5. 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 7.6. The statement reachability, line operation testability and the relative variable range for the Diff benchmark are found in Table 7.2. The test hardness, $TH(l)$ at line l is given by all operations used at that line. For instance at line 7, $TH(7)=(1-OpT(adder))+(1-OpT(multiplier))= (1-0.29)+(1-0.19)=1.52$. Then the Line Operation Testability at line 7 is $LOT(7)=1-TH(7)/TH_{max}=1-1.52/5.47=0.72$ where TH_{max} comes from line 5 which has the highest test hardness.

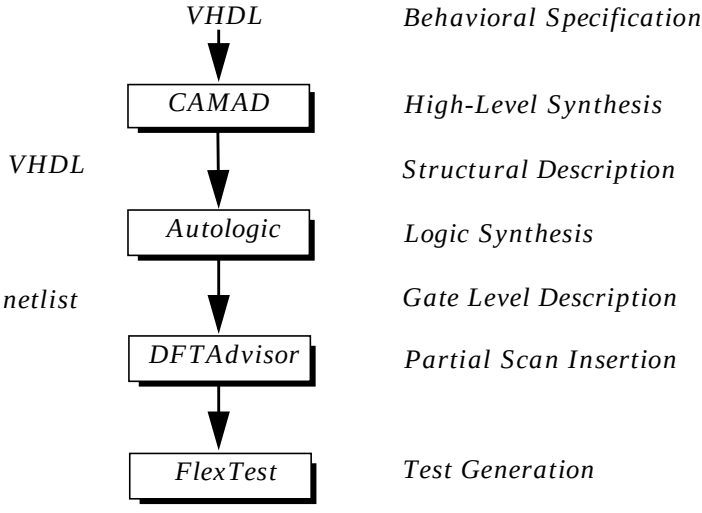


Figure 7.5: The design flow.

```
ENTITY diff IS
  PORT( x_in : IN INTEGER RANGE 0 TO 255;
        y_in : IN INTEGER RANGE 0 TO 255;
        u_in : IN INTEGER RANGE 0 TO 255;
        x_out : OUT INTEGER RANGE 0 TO 255;
        y_out : OUT INTEGER RANGE 0 TO 255;
        u_out : OUT INTEGER RANGE 0 TO 255);
END;

ARCHITECTURE behavior OF diff IS
BEGIN
  PROCESS
    CONST
      a = 38;
      dx = 1;
    VARIABLE
      x,y,u : INTEGER RANGE 0 TO 255;
      x1,y1,u1 : INTEGER RANGE 0 TO 255;
  BEGIN                                          Line
    x:=x_in;                                      (1)
    y:=y_in;                                      (2)
    u:=u_in;                                      (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;                                      (8)
      y:=y1;                                      (9)
      u:=u1;                                     (10)
    END LOOP;                                    (11)
    xout<=x;                                     (12)
    yout<=y;                                     (13)
    uout<=u;                                     (14)

  END PROCESS;
END behavior;
```

Figure 7.6: The behavioral description of the Diff benchmark.

In Table 7.3 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. For instance, the testability, $T(x)$, for variable x at is then 0.25+1+0.15=1.4.

After the analysis of the Diff benchmark we performed experiments to see whether the prediction for the testability of the variables was correct. The results are given in Table 7.4. 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 not considered to be difficult to test, so scanning it will not improve the design's testability very much; we get only 16.15% fault coverage as a 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 7.5 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, as shown in Table 7.5.

Line	Statement Reachability	Line 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:0.15 x1:0.15
6	0.25	0	y:1 u:1 u1:1 x:0.15
7	0.25	0.72	y:1 y1:1 u:1
8	0.25	1	x:0.15 x1:0.15
9	0.25	1	y:1 y1:1
10	0.25	1	u:1 u1:1
11	0.25	1	
12	1	1	x:0.85
13	1	1	y:1
14	1	1	u:1

Table 7.2: The variable range, operation testability and statement reachability for the Diff benchmark.

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 7.3: The variable range, operation testability and statement reachability combined into one test hardness metric for the Diff benchmark.

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 7.4: 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 7.5: Fault coverage comparison of testability analysis at the behavioral and the gate level.

7.6 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.

In 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 8
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.

8.1 Basic Transformations

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

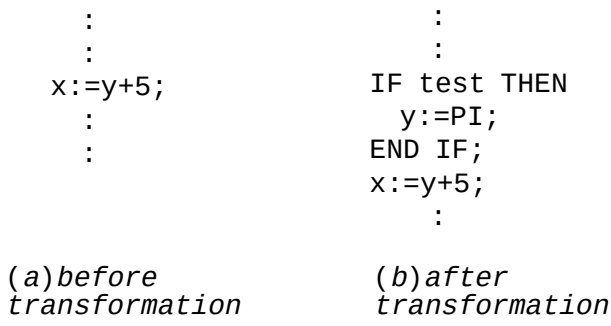


Figure 8.1: Example of READ-insertion.

8.1.1 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 8.1 where the content of variable *y* is hard to control. In Figure 8.1(b) *test*, an extra primary input, is added to determine whether 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. 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.

8.1.2 WRITE-INSERTION

The WRITE-insertion transformation improves the ability to observe the contents of a variable. The idea is illustrated in Figure 8.2. In Figure 8.2(a) variable *x* is hard to observe and by applying WRITE-insertion we improve its observability as shown in Figure 8.2(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 8.2(b). In this case, we do

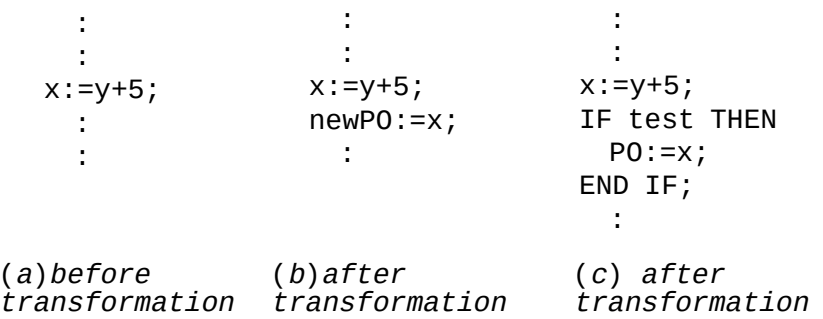


Figure 8.2: Example of WRITE-insertion.

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 observation, we have to add an extra pin to check whether we are in test mode or normal mode, which is illustrated in Figure 8.2(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.

8.1.3 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 major 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

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

(*a*) *before transformation*

(*b*) *after transformation*

Figure 8.3: Illustration of the use of AND-insertion.

on the following VHDL constructs: IF, WHILE, FOR and CASE statements.

We use the example in Figure 8.3 to illustrate the use of the BOOLEAN-insertion technique. In Figure 8.3(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 8.3(b). The result is that by setting the added test pin to false, we can determine when to exit the loop. 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 8.3(a) we have at the WHILE statement a condition *c* (z<10) and in Figure 8.3(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 occurance than the value FALSE (F) and vice versa when AND-insertion is used. We note that with the test

```
    :
    :
while c loop
  z:=z+1;
end loop;
    :
    :
```

(*a*)*before transformation*

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

(*b*)*after transformation*

Figure 8.4: Example of AND/OR-insertion.

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 8.4 (b) 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.

8.1.4 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 8.5 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 state-

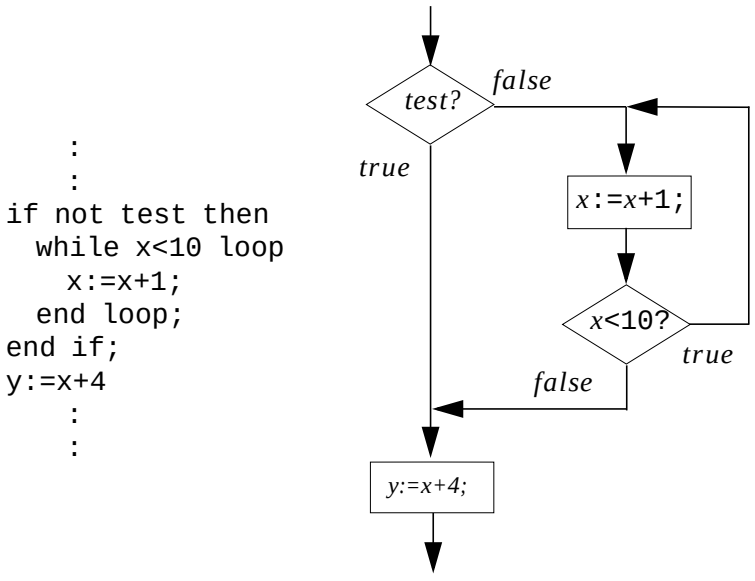


Figure 8.5: Example of REACH-insertion.

ments can be easily reached in the test mode. In the example, Figure 8.5, the statement *y:=x+*4 can, in the test mode, be reached without having to traverse the loop structure first.

8.2 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.

Traditionally the design space has been simplified to a two-dimensional space over area and performance, which is illustrated in Figure 8.6(a). If the design is optimized towards small delay (improved performance), it usually means that the

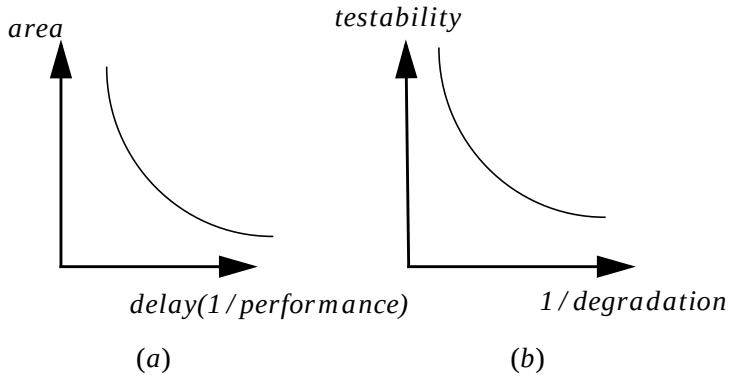


Figure 8.6: Illustration of the design space.

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 8.6(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 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.

8.3 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.

8.3.1 SELECTION BASED ON LOOP BREAKING

As discussed earlier, loops in a design are known to cause major 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 reduce 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.

8.3.2 SELECTION BASED ON TESTABILITY METRICS

In Chapter 7 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 7 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* metrics, $T(v)$, as defined in Section 7.3.

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 8.7(a) and in Figure 8.7(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.

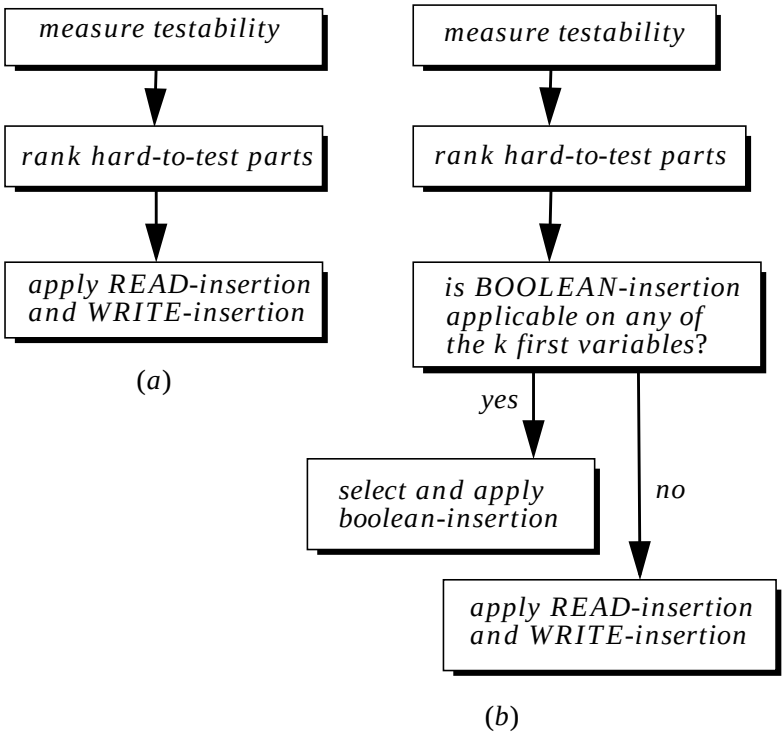


Figure 8.7: Extension of the transformation selection to include BOOLEAN-insertion.

Note, that the testability selection is based on selecting variables and not specific lines in the behavioral VHDL specification. Furthermore, there is no guidance on which lines in the specification a transformation should be applied. The reason is explained with an example. Assume a READ-insertion on a variable which after synthesis corresponds register *r*, see Figure 8.8. The added *test* controls the added multiplexer and can be incooperated in the controller. However, we assume that the test control is added as an extra primary input and doing so makes it possible to set test at any time when executing. The result is that the selection of where (which line) in the behavio-

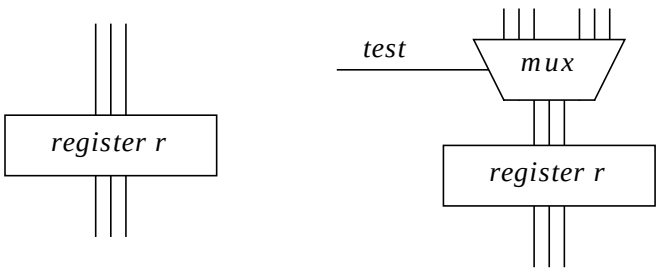


Figure 8.8: The behavioral testability transformations at register-transfer level.

ral specification the transformation is inserted is not important. What is important is the selection of which variables to modify.

The set of lines affected by a condition *c* is captured $cs_t(c)$ and $cs_f(c)$, defined by Definition 7.7 in Chapter 7.

Definition 8.1: 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 *v* 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.

The CT is not calculated for variables not occurring, *i.e.* $n>0$ in the definition above.

Definition 8.2: 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.

In Figure 8.9(a) a sample of VHDL code is shown as well as the variable range for the variables. In Figure 8.9(b) we have

```
    :
A:0..31, B:10..20
WHILE A<10 LOOP
  B:=B+A;
  A:0..9, B:10..29
    :
END LOOP;
A:10..31, B:10..29
    :
```

(*a*)*original*

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

(*b*)*OR-insertion*

```
    :
A:0..31, B:10..20
WHILE A<10 AND T LOOP
  B:=B+A;
  A:0..9, B:10..29
    :
END LOOP;
A:0..31, B:10..30
    :
```

(*c*)*AND-insertion*

```
    :
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*)*AND/OR-insertion*

Figure 8.9: Illustration of how the variable range changes when BOOLEAN-insertion is used.

used the OR-insertion and we see how the variable range for the variables is changed, in Figure 8.9(c) we use AND-insertion and in Figure 8.9(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

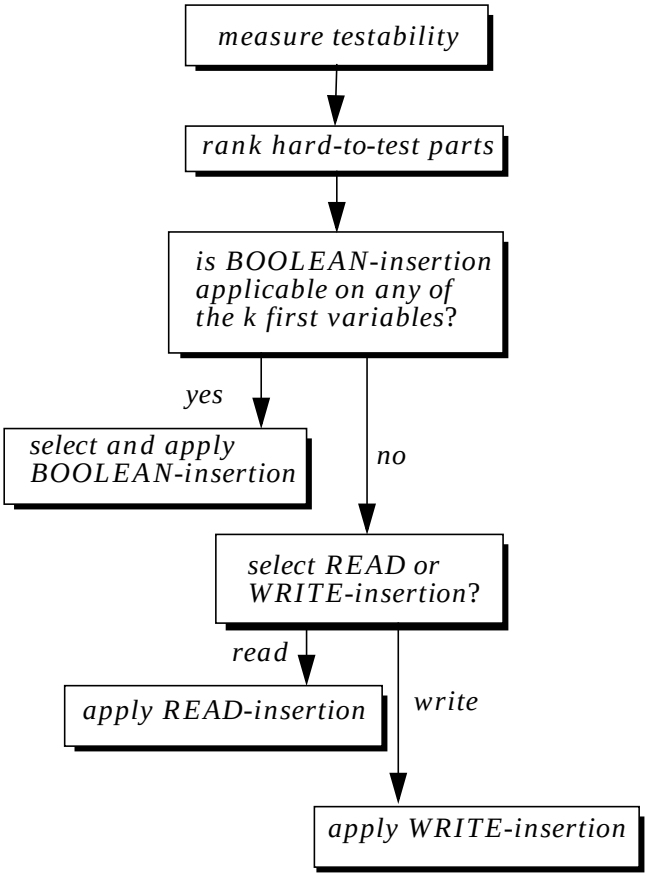


Figure 8.10: Extension of the transformation selection to distinguish between READ-insertion or WRITE-insertion.

a variable without using the WRITE-insertion and vice versa. We extend the testability flow to distinguish between READ-insertion and WRITE-insertion, Figure 8.10.

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 first use the testability met-

ric as it was defined for partial scan to rank the variables and we define then a strategy to determine whether we should apply a READ-insertion or a WRITE-insertion. We note that, the READ-insertion and the WRITE-insertion are applicable on lines with assignments and arithmetic operations.

Definition 8.3: 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 8.3 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.

8.4 Experimental Results

In this section we present our experimental results on the testability transformations and we use the experimental platform described in Section 7.5. 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 8.11 and the behavioral specification for the Counter is shown in Figure 8.12.

In our first experiment we want to test whether 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 8.1. 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

```
ENTITY diff IS
  PORT( x_in : IN INTEGER RANGE 0 TO 255;
        y_in : IN INTEGER RANGE 0 TO 255;
        u_in : IN INTEGER RANGE 0 TO 255;
        x_out : OUT INTEGER RANGE 0 TO 255;
        y_out : OUT INTEGER RANGE 0 TO 255;
        u_out : OUT INTEGER RANGE 0 TO 255);
END;

ARCHITECTURE behavior OF diff IS
BEGIN
  PROCESS
    CONST
      a = 38;
      dx = 1;
    VARIABLE
      x, y, u : INTEGER RANGE 0 TO 255;
      x1, y1, u1 : INTEGER RANGE 0 TO 255;
  BEGIN                                          Line
    x:=x_in;                                     (1)
    y:=y_in;                                     (2)
    u:=u_in;                                     (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;                                     (8)
      y:=y1;                                     (9)
      u:=u1;                                     (10)
    END LOOP;                                    (11)
    xout<=x;                                     (12)
    yout<=y;                                     (13)
    uout<=u;                                     (14)

  END PROCESS;
END behavior;
```

Figure 8.11: The behavioral specification of the Diff benchmark.

```
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 8.12: 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 8.1: Experiments to show the correlation between our testability metrics, partial scan and READ/WRITE-insertion.

variable x_1 (which is 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 provides a fault coverage of 16.8%.

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 c/o. 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 preferred. In Table 8.2 the testability metric for the Diff benchmark is divided into a controllability part and an observability part and in Table 8.3 the experimental results from the transformation selection are 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 c/o ratio is 0.87 which, compared to the c/o ratio for all other variables, is high. This indicates that we should use a READ-insertion. By using a READ-insertion on variable x_1 we achieve a fault coverage of

Line	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
c/o	0.87		0.71		0.51		0.86		0.86		0.56	

Table 8.2: The testability metric divided into a controllability(c) part and an observability(o) part for each line (l) in the Diff benchmark.

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%.

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	0.56	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	0.71	read	13.1	write	16.5

Table 8.3: Experiments to show the correlation between our strategy to selecting READ or WRITE-insertion and the fault coverage.

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.

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 c/o ratio 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%. However, on variable y_1 our heuristic indicates that a READ-insertion should be used. But using the READ-insertion provides a fault coverage of 12.7%, while using a WRITE-insertion we achieve a fault coverage of 70.7%. This indicates that our heuristic makes a correct transformation selection regarding variable x_1 but a mistake for variable y_1. However, since candidates for transformation are selected based on their ranked test difficulties, meaning that a hard-to-test variable is a better candidate than a variable ranked at lower test hardness. In the example, this means that x_1 will be selected since it is the hardest-to-test variable and variable $y_1$which is ranked low is in the first place is not a good candidate for transformation selection.

In Table 8.4 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 8.5, 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.

8.5 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

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 8.4: 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 8.5: Experimental results for selection of testability transformation.

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 8.4: 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 8.5: 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 8.6: A path, P_i, is a sequence of edges $\{(v_o, v_1), (v_1, v_2),..., (v_{n-1}, v_n)\}$ where v_0 is a start vertex and v_n is an end vertex.

Definition 8.7: 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 8.13, 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*}.

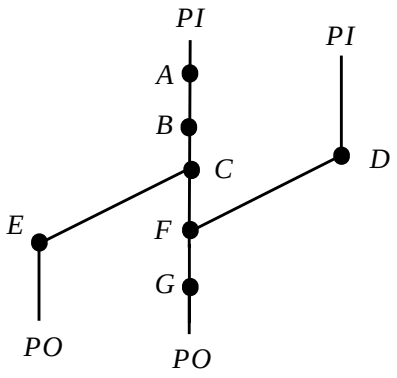


Figure 8.13: Example of Variable Dependency.

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

To illustrate the algorithm we use the example in Figure 8.13. 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

```
Sort the testabilities T(v) so that
T1 is the hardest and TV is the
easiest to test.
Select T1 for DFT improvement.
for i = 2 to |V|
  Diff = ∅;
  for j = 1 to i - 1
    Diff=Diff+SOP(vi)∩SOP(vj)+
      SCP(vi)∩SCP(vj);
  if Diff = ∅ then
    select Ti for DFT improvement
```

Figure 8.14: Algorithm for grouping variables.

$SCP(F)=\{D\}$ 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.

8.6 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 9
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.

9.1 Introduction

As discussed before, many DFT techniques require large area overhead and may degrade the performance of a circuit. Several 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.* [Hsu96a], [Hsu96b]. 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.

9.2 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 9.1 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.

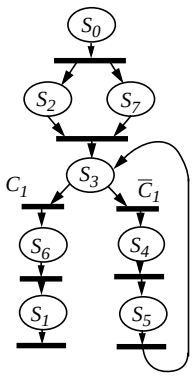
As an example, in Figure 9.1 state S_4 in the controller is used to control the data transfer from input port P_1 to register Y in the data path. When S_4 holds a token [Pet81], this transfer will take place. Condition nodes C_1 and $\overline{C}_1$ in the data path control the transitions from S_3 to 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 9.1, 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 as discussed in Section 6.1.2. The testability analysis takes into account the structure of a design, the depth from I/O ports and the characteristics of the 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].

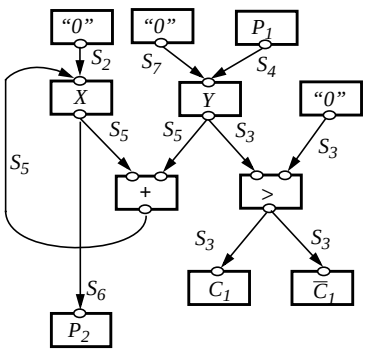
```
ENTITY counter IS
  PORT( P1 : IN INTEGER;
        P2 : OUT INTEGER);
END;

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 9.1: A design example in VHDL and ETPN.

9.3 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 9.2, has the best state reachability:

$$CSR_0 = 1 \tag{9.1}$$

$$SSR_0 = 0 \tag{9.2}$$

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 9.2: 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 9.3. The state reachability will be calculated as:

$$CSR_j = CSR_i \tag{9.3}$$

$$SSR_j = SSR_i + 1 \tag{9.4}$$

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.

Figure 9.3: Simple construct.

Here we assume that each state will hold for one clock cycle time. For advanced treatment of clock cycle time refer to [Pen94].

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 9.4, 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:

$$CSR_k = \begin{cases} CSR_i & if\left(CSR_i + \dfrac{SSR_L}{SSR_i} > CSR_j + \dfrac{SSR_L}{SSR_j}\right) \\ CSR_j & otherwise. \end{cases} \tag{9.5}$$

$$SSR_k = \begin{cases} SSR_i + 1 & if\ CSR_k = CSR_i \\ SSR_j + 1 & if\ CSR_k = CSR_j \end{cases} \tag{9.6}$$

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.

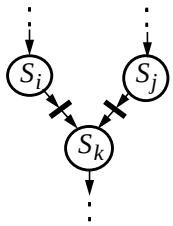


Figure 9.4: OR-construct.

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 9.5, 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:

$$CSR_i = CSR_k \times CC_c \tag{9.7}$$

$$SSR_i = max\{SSR_k, SC_c\} + 1 \tag{9.8}$$

$$CSR_j = CSR_k \times CC_{\bar{c}} \tag{9.9}$$

$$SSR_j = max\{SSR_k, SC_{\bar{c}}\} + 1 \tag{9.10}$$

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.

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

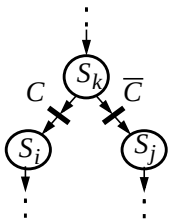


Figure 9.5: Conditional construct.

Figure 9.6, 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:

$$CSR_k = CSR_i \times CSR_j \tag{9.11}$$

$$SSR_k = \text{Max}\{SSR_i, SSR_j\} + 1 \tag{9.12}$$

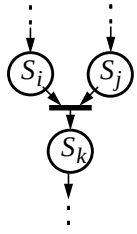


Figure 9.6: AND-construct.

Parallel Construct: In a parallel construct, a set of states will be reached by firing a transition. Figure 9.7 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.

9.4 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

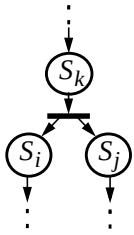


Figure 9.7: Parallel construct.

FIFO queue, *Q*. A 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 formulas 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 9.8. 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 9.8).

It must be noted that a loop consists of both conditional and OR-constructs and formulas 9.5-9.10 are used to compute their reachabilities. These computations involve calculation of controllability factors for the conditions controlling the loop execution 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].

1. *Assign all initial states (use formulas 9.1 and 9.2).*
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 formulas 9.3 and 9.4).
   b) if it is an AND-construct:
   check if all the other input state(s) have been calculated.
   if "yes", go to 6 (use formulas 9.11 and 9.12).
   if "no", then put S_{prev} to Q and go to 4.
   c) if it is an OR-construct: go to 6 (use formulas 9.5 and 9.6).
   d) if it is a conditional construct or parallel construct:
   go to 6 (use formulas 9.7, 9.8, 9.9 and 9.10 or formulas 9.1 and 9.2).
   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 formulas.*
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 9.8: State reachability analysis algorithm.

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:

$$CSR_i + k \times \frac{SSR_L}{SSR_i} \tag{9.13}$$

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.

9.5 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.

9.5.1 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 9.9 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 9.9) 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

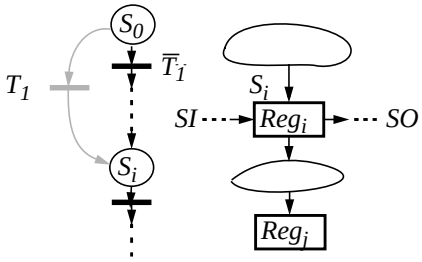


Figure 9.9: Initialize or set Reg_j through Reg_i.

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.

9.5.2 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 9.9 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 matter what value C has. If we only need to enhance the reach-

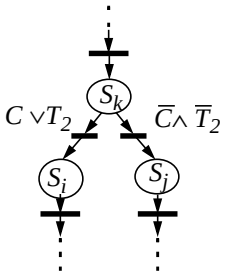


Figure 9.10: Select branch by T_2.

ability 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.

9.5.3 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 9.9 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.

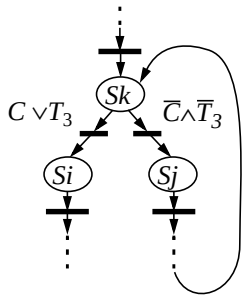


Figure 9.11: Terminate feedback by T_3.

9.6 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 first performed experiments on a simple counter, which consists of a controller and a data path, to show the importance of considering the whole design. The results are presented in Table 9.1. 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 9.1: Example to illustrate the importance of considering both the controller and the data path.

The second group of experiments was performed on several benchmarks, a Counter, a differential equation (Diff) [Pau89], and Mag [Tri85]. The results are presented in Table 9.2, 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 the 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 9.2: Summary of experimental results.

9.7 Summary

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 state reachability.

The proposed technique 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 smaller area and timing penalties.

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

Conclusions and Future Work

PART IV

Chapter 10
Conclusions

10.1 Thesis Summery

The aim of the work presented in this thesis is to develop useful methods to give the designer an early feeling for the test problems and a guidance in the search for an efficient test solution. The methods are developed mainly at the system level since we believe that it is important for the designer to have an overall perspective of the system and its test problems as early as possible. Further, by considering the test problems at high abstraction level, the test problems can be reduced at production and operation and maintenance.

This thesis consists of two major contributions. The first one is the development of the test scheduling and test access mechanism design technique. Our proposed technique minimizes the test application time and the test access mechanism cost while considering several issues and constraints. The second contribution deals with testability analysis of the behavioral level VHDL specification and several testability improvement transformations. Furthermore, a testability analysis of the controller at a

register-transfer level design and transformations to enhance its testability have also been developed.

10.1.1 TEST SCHEDULING AND TEST ACCESS MECHANISM

The testing of SOC is sometimes compared to the testing of PCB. However, when developing a PCB the components are assumed to be tested before mounting. This is not the case with SOC. This means that much more testing is required for a SOC compared to a PCB and more test vectors and test response have to be transported in SOC designs than in PCB designs.

A SOC consists of several cores where each core may consist of several blocks. A sequential testing of such a system leads to an unacceptably long test time. Several tests must be applied concurrent. However, concurrent testing can lead to high test power consumption which can damage the system. Furthermore, several constraints limit concurrent testing.

In this thesis, a methodology for the testing of SOC has been developed. The methodology consists of several integrated and efficient algorithms for test scheduling, test access mechanism design, test parallelization, test set selection and test resource placement. Furthermore, the test resources may have limited memory and bandwidth which are also considered in our approach. Optimization techniques where the test application time and the test access mechanism are minimized, while considering test conflicts, test power consumption and test resources, have been developed.

The methodology considers both test scheduling and test parallelization which is an advantage since it can reduce the test application time.

We have performed several experiments on academic benchmarks and on industrial designs and we have compared our approach with several other approaches. We have demonstrated that the proposed technique is useful for large industrial designs.

10.1.2 TESTABILITY ANALYSIS AND ENHANCEMENT TECHNIQUES

The modification of a design to make it more testable usually leads to some design degradation in terms of implementation cost and/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.

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 iterations by selecting several hard-to-test parts in each design step.

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 in design for testability 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.

10.2 Conclusions

In this thesis we have developed methods which help and guide the designer in the search for an efficient test solution. The methods are developed mainly at the system-level which is important in order to get an early feeling for the test problems of the whole system.

The high complexity of SOC, which requires extensive testing and high amount of test data to be transported, has led to the need for a systematic test methodology. We have developed such a methodology and shown the efficiency of our approach by performing extensive experiments on academic benchmarks and industrial designs.

We have also developed a behavioral level testability analysis technique and shown that it detects the hard-to-test parts. We have defined a set of testability improvement transformations and a selection strategy for them. We have also developed a technique to analyze the testability of the controller and testability enhancement transformations of the controller.

The main conclusions we have drawn from this research is as follows:

- By considering testability at the early design stages, efficient test solutions can be developed, which leads to the reduction of the total test cost.
- It is important to integrate test scheduling and test access mechanism design in order to generate optimal solutions for SOC testing.

- The complexity of testability analysis at the behavioral and RT levels is much smaller than that at the gate level. A carefully designed testability analysis algorithm at the high levels of abstraction can produce the same quality of analysis results, which can be used to guide testability enhancement transformations.
- It is important to consider the testability of the controller together with the data path in order to generate highly testable designs at the RT level.

Chapter 11
Future Work

In this chapter we discuss possible future work. In Section 11.1 general estimation techniques are discussed. Possible future work regarding test scheduling and test access mechanism are given in Section 11.2, and some open issues for testability analysis and enhancement techniques in Section 11.3.

11.1 Estimation of Test Parameters

The techniques and algorithms proposed in this thesis require some characteristics data about the system such as test time and test power consumption for each test. It is assumed that this data is given and fixed, which is the case for certain applications. An estimation technique for these parameters would be desirable. For instance, knowing the test vector set it would be possible to define an estimation on the switch activity resulting in an estimate of the test power consumption.

Another problem to consider is test parallelization and its effect on test time and test power consumption where the main question would be estimations on test time and test power consumption at different degrees of parallelization.

11.2 Test Scheduling and Test Access Mechanism

Several important issues for test scheduling and test access mechanism design have been described in this thesis. An important continuation of the work would be to grade these factors on their importance.

In the algorithm on test parallelization the degree of parallelization for each block was selected. For the finial optimization using Simulated annealing no changing of degree in test parallelization was allowed. It would be interesting to explore the possibility of selecting different degrees of parallelization and different test sets during the optimization process.

There is usually a limitation of the testers memory which has been covered by our approach. However, no experiments were performed on testers memory. It would be interesting to perform such experiments. Furthermore, the test bandwidth which has its origin at several places in the system is included in the algorithms but no experiments have been performed.

Even if the heuristics presented in this thesis gives good results, in each design step only one factor is minimized at a time. It would be interesting to develop an efficient heuristic considering a cost function dealing with several factors simultaneously which still has low computational cost.

The test access mechanism is becoming more and more important. The amount of test data to be transported in a microelectronic system tend to increase especially in SOC where the cores have to be tested extensively. Due to the increasing performance of the systems where the timing is becoming critical, it is also likely that delay faults must be considered for more and more parts in the system. The result is that even more test data is to be transported.

A wrapper around a core efficiently eases test access and isolates the core under test. However, the wrappers currently available only allow a limited bandwidth. Flexible test access and

bypass structures for wrappers are important issues to be considered in the future.

11.3 Testability Analysis and Testability Enhancements

Much research has focused on testability analysis where gate-level designs or RT-level designs have been analyzed and then their testability improved. Some work has been defined for behavioral level specifications such as the one presented in this thesis. For SOC testability analysis there is very little work done.

Regarding testability enhancement techniques, most work is done at gate-level while at behavioral level there is also very little work done.

Extensions of the testability analysis

The behavioral level testability analysis defined in this thesis is limited to a subset in VHDL. For instance, only one process is allowed. It would be desirable to extend it to include the whole VHDL.

Furthermore, a testability analysis of a complete SOC is needed. Such approach should consider SOC characteristics which has not been included in previous testability analysis techniques, such as multiple clock domains. Furthermore, as design complexity increase, design parts are reused and/or complete intellectual property (IP) blocks are incooperated in the design. These parts can be described at different abstraction levels. Analyzing the testability for a design which consists of parts described at different abstraction levels has not been done.

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. The 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.

Furthermore, as test application time for systems increase, there is a need of developing DFT techniques based on test-per-clock instead of test-per-scan. As these techniques are developed, new selection strategies must also follow. From a system-level perspective, it is not obvious that test-per-clock may always be the optimum for all cores or blocks in the system. Strategies

combining test-per-scan and test-per-clock considering the systems total test time effect is important.

Experiments on larger benchmarks

The experiments using the testability analysis and testability improvement technique were performed on rather small benchmarks. It would be desirable to apply the testability technique on larger design examples.

Appendix

PART V

Appendix A

BENCHMARKS AND INDUSTRIAL DESIGNS used to illustrate approaches in the thesis are described in this appendix. The benchmark examples are a design presented by Kime and Saluja [Kim82], System S defined by Chakrabaraty [Cha99] and a design presented by Muresan *et al*. [Mur00] The industrial designs are ASIC Z presented by Zorian [Zor93] with added data by Chou *et al*. [Cho97], an extended version of ASIC Z, the System L and the Ericsson design [Eri00].

A.1 Design Kime

The test compatibility graph of a design with six tests is taken from Kime and Saluja [Kim82], see Figure A.1. Test t_1 and t_6 may be scheduled concurrently since an arc exists between node t_1 and node t_6. On the other hand, test t_1 and t_2 may not be scheduled concurrently since no arc exists between the node t_1 and node t_2. Each node has its test time attached to it. For instance, test t_1 requires 255 time units.

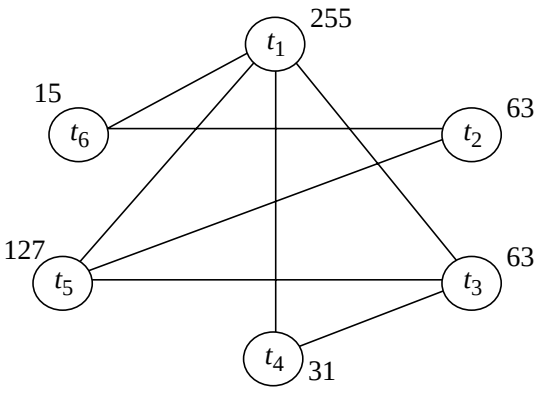


Figure A.1: Test compatibility graph.

A.2 System S

System S is defined by Chakrabaraty [Cha99] and it consists of six cores where each core is an ISCAS benchmark (core), see Figure A.2. Data for the system is given in Table A.1 where i is the core and for each core i the number of external test cycles, e_i, and number of BIST cycles, b_i are specified.

Each core is tested by two test sets, one BIST test set and one deterministic test set. The deterministic test vector set is applied using an external tester and the test bus. Only one core at the time can use the test bus and the external tester. The BIST patterns take one clock cycle to apply while the external tester is ten times slower.

We have added placement for the cores in the system, see Table A.1.

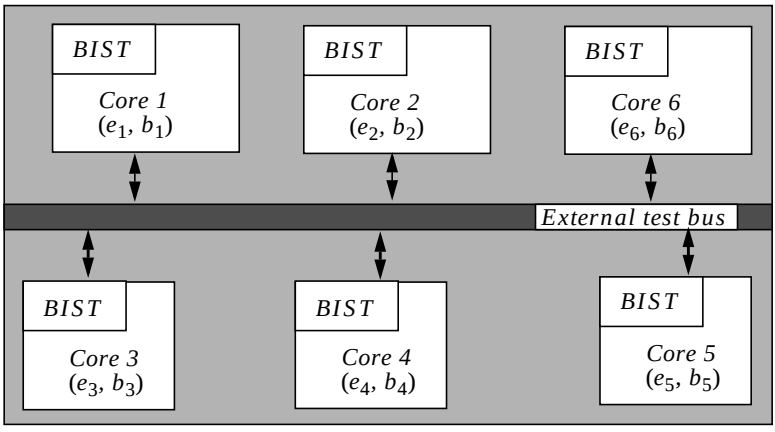


Figure A.2: System S.

Circuit	Core *i*	Number of external test cycles, e_i.	Number of BIST cycles, b_i.	Placement	
				x	y
c880	1	377	4096	10	10
c2670	2	15958	64000	20	10
c7552	3	8448	64000	10	30
s953	4	28959	217140	20	30
s5378	5	60698	389210	30	30
s1196	6	778	135200	30	10

Table A.1: Test data for the cores in System S.

A.3 Design Muresan

Muresan *et al.* present a design with the design data presented in Table A.2 [Mur00]. For instance, test t_2 requires 8 time units and 4 power units and it is test compatible with the following tests: $\{t_1, t_3, t_7, t_9\}$. For instance, it means that test t_2 can be scheduled at the same time as test t_3.

The power limit for the design is 12 power units.

Test	Test time	Test power	Test Compatibility
t_1	9	9	t_2, t_3, t_5, t_6, t_8, t_9
t_2	8	4	t_1, t_3, t_7, t_8
t_3	8	1	t_1, t_2, t_4, t_7, t_9, t_{10}
t_4	6	6	t_3, t_5, t_7, t_8
t_5	5	5	t_1, t_4, t_9, t_{10}
t_6	4	2	t_1, t_7, t_8, t_9
t_7	3	1	t_2, t_3, t_4, t_6, t_8, t_9
t_8	2	4	t_1, t_2, t_4, t_6, t_7, t_9, t_{10}
t_9	1	12	t_1, t_3, t_5, t_6, t_7, t_8, t_{10}
t_{10}	1	7	t_3, t_5, t_8, t_9

Table A.2: Design data for design Muresan.

A.4 ASIC Z

The ASIC Z design presented by Zorian [Zor93] with the estimations on test length made by Chou *et al.* is in Figure A.3 and Table A.3. The power consumption for each block when it is in idle mode and for each test when it is in test mode is given by Zorian. The test lengths for each test is computed by Chou *et al.* with an assumption of linear dependency between test length and block size, see Table A.3 [Cho97].

The design originally consists of 10 cores. However, no data is available for one block therefore it is excluded from the design. The maximal allowed power dissipation of the system is 900 mW. All blocks have their own dedicated BIST which means that all tests can be scheduled concurrently.

We have added the placement, see Table A.3, where each block is given an x-placement and a y-placement.

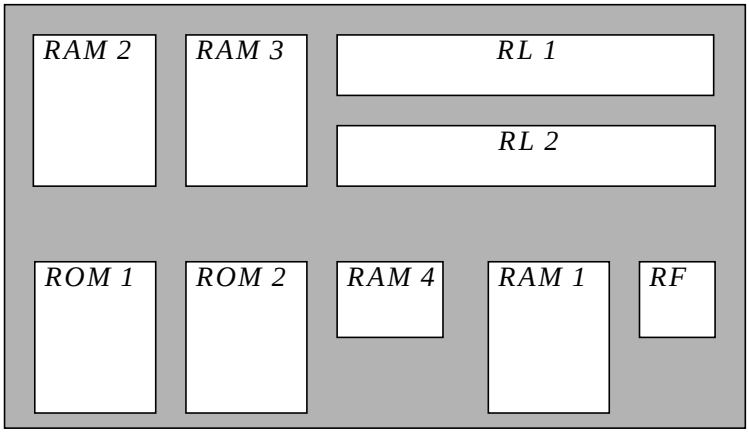


Figure A.3: ASIC Z floor-plan.

Block	Size	Test Time	Idle Power	Test Power	Placement	
					x	y
RL1	13400 gates	134	0	295	40	30
RL2	16000 gates	160	0	352	40	20
RF	64×17 bits	10	19	95	50	10
RAM1	768×9 bits	69	20	282	40	10
RAM2	768×8 bits	61	17	241	10	20
RAM3	768×5 bits	38	11	213	20	20
RAM4	768×3 bits	23	7	96	30	10
ROM1	1024×10 bits	102	23	279	10	10
ROM2	1024×10 bits	102	23	279	20	10

Table A.3: ASIC Z characteristics.

A.5 Extended ASIC Z

The Extended ASIC Z design is an extended version of ASIC Z, see Appendix A.4. For each core three tests are defined:

- an interconnection test,
- an BIST test, and
- an external test.

In total there are 27 tests spread over the 9 cores. The maximal power consumption and placement is assumed to be the same as for ASIC Z, Appendix A.4.

The characteristics for Extended ASIC Z are in Table A.4. For instance, a BIST test at RL1 require test generator TG_{rl1} and test analyser TA_{rl1}. The test takes 67 time units and consumes 295 mW and when it is applied no other tests at RL1 can be performed.

The interconnection tests are performed between two cores. For instance core RL1 performs an interconnection test with RL2 which requires 10 time units and 10 mW. When this test is applied it is assumed that no other test can be performed at RL1 and RL2 (specified under block constraint in Table A.4).

In this design the BIST resources are shared and each BIST resources can be used by one test at a time. For instance when RAM1 is tested using TG_{ram} and TA_{ram} no other tests can be performed using these test resources.

The external tests are connected through TAP and several tests can be applied concurrently using the external tester.

For Extended ASIC Z all tests at a core are at one block which means that the BIST and the external test may not be scheduled concurrently.

Core	Test time	Test power	Test source	Test sink	Block constraint
	67	295	TAP	TAP	RL1
RL1	67	295	TG_{rl1}	TA_{rl1}	RL1
	10	10	TAP	TAP	RL1, RL2
	80	352	TAP	TAP	RL2
RL2	80	352	TG_{rl2}	TA_{rl2}	RL2
	10	10	TAP	TAP	RL2, RAM3
	5	95	TAP	TAP	RF
RF	5	95	TG_{rf}	TA_{rf}	RF
	10	10	TAP	TAP	RF,RL1
	35	282	TAP	TAP	RAM1
RAM1	35	282	TG_{ram}	TA_{ram}	RAM1
	10	10	TAP	TAP	RAM1,RF
	31	241	TAP	TAP	RAM2
RAM2	31	241	TG_{ram}	TA_{ram}	RAM2
	10	10	TAP	TAP	RAM2, ROM1
	19	213	TAP	TAP	RAM3
RAM3	19	213	TG_{ram}	TA_{ram}	RAM3
	10	10	TAP	TAP	RAM3, RAM2
	12	96	TAP	TAP	RAM4
RAM4	12	96	TG_{ram}	TA_{ram}	RAM4
	10	10	TAP	TAP	RAM4, RAM1
	51	279	TAP	TAP	ROM1
ROM1	51	279	TG_{rom}	TA_{rom}	ROM1
	10	10	TAP	TAP	ROM1, ROM2
	51	279	TAP	TAP	ROM2
ROM2	51	279	TG_{rom}	TA_{rom}	ROM2
	10	10	TAP	TAP	ROM2, RAM4

Table A.4: Extended ASIC Z characteristics.

A.6 System L

System L is an industrial design consisting of 14 cores named A through N, see Table A.5. It is tested by 17 tests distributed over the system as block-level tests and top-level tests. The block-level tests and the top-level tests can not be executed simultaneously. Furthermore, all block-level using the test bus can not be executed concurrently. The top-level tests are using the functional pins which makes concurrent scheduling among them impossible.

All tests are using external test resources and the total power limit for the system is 1200 mW.

A.7 Ericsson design

The Ericsson design, see Figure A.4, consists of 8 digital signal processor (DSP) cores: a block for DSP control (DSPIOC); 2 memory banks, a common program memory (CPM) and common data memory (CDM); a control unit for each memory bank, common data memory controller (CDMC) and common program memory controller (CPMC); and five other blocks, RX1C, RX0C, DMAIOC, CKReg and TXC. In total there are 18 cores.

Each of the DSP cores in the Ericsson design in Figure A.4 consists of four banks of local data memory (LDM), one bank of local program memory and two banks of other memory (LZM) and five logic blocks, see Figure A.5. The memory banks of the CPM block and the CDM block in Figure A.4 are shown in Figure A.6 respectively in Figure A.7.

The characteristics for each of the blocks in the design are in Table A.6 where the test time, test power and test resource is specified for each block in the system. The idle power is zero for all blocks. The DSPs are numbered by n in range 0 to 7 which results in total 170 ($17 \times 7 + 51$) tests.

Test	Block	Test	Test time	Idle power	Test power	Test port
Block-level tests	A	Test A	515	1	379	scan
	B	Test B	160	1	205	test-bus
	C	Test C	110	1	23	test-bus
	D	Test D	Tested as part of other top-level test			
	E	Test E	61	1	57	test-bus
	F	Test F	38	1	27	test-bus
	G	Test G	Tested as part of other top-level test			
	H	Test H	Tested as part of other top-level test			
	I	Test I	29	1	120	test-bus
	J	Test J	6	1	13	test-bus
	K	Test K	3	1	9	test-bus
	L	Test L	3	1	9	test-bus
	M	Test M	218	1	5	test-bus
Top-level tests	A	Test N	232	1	379	functional pins
	N	Test O	41	1	50	functional pins
	B	Test P	72	1	205	functional pins
	D	Test Q	104	1	39	functional pins

Table A.5: System L characteristics.

The maximal allowed power consumption is limited to 5125 mW. For each logic block two test sets are applied. One using an external tester and one on-chip tester. These tests can not be applied at the same time since they test the same logic. All logic blocks within a DSP core share one test source and test sink for the on-chip test. The connection to the external tester is named TAP and several tests may use the external tester concurrently.

All memory blocks of the same type have their own test resources. For instance, the blocks within the CPM have one test

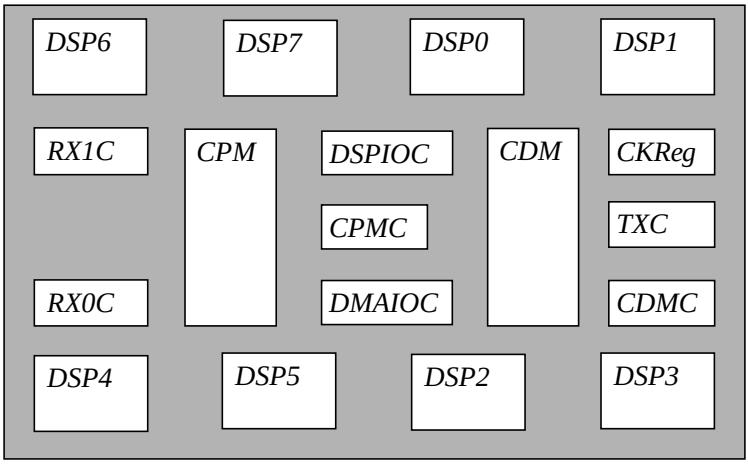


Figure A.4: The Ericsson design.

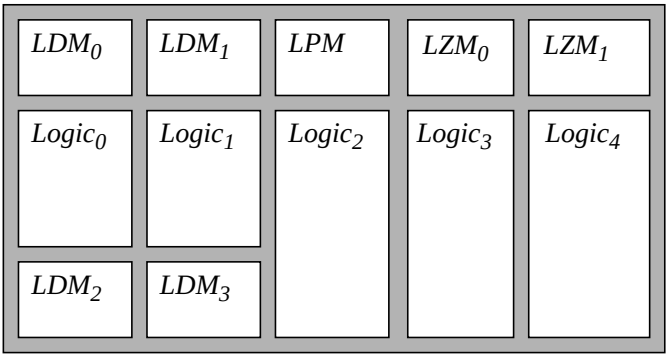


Figure A.5: The blocks within each DSP_n.

generator and one test response analyser. The placement of all blocks are in Table A.7.

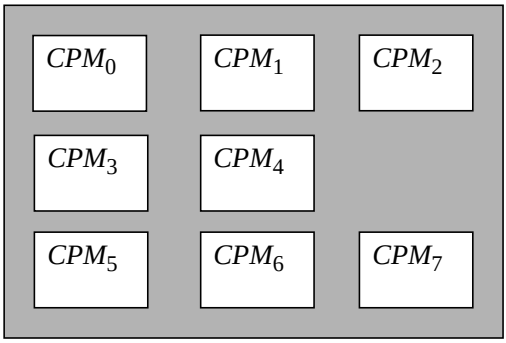


Figure A.6: The blocks within CPM.

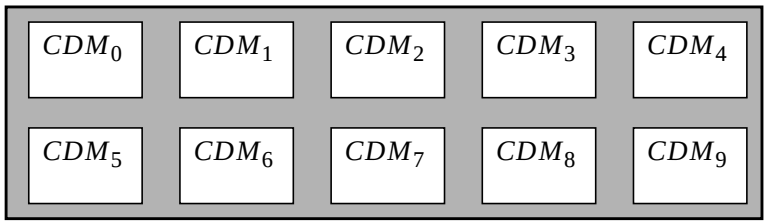


Figure A.7: The common data memory bank.

Block		Test	Test time	Test power	Test source	Test sink
RX0C		1	970	375	TAP	TAP
		2	970	375	TG_0	TRA_0
RX1C		3	970	375	TAP	TAP
		4	970	375	TG_0	TRA_0
DSPIOC		5	1592	710	TAP	TAP
		6	1592	710	TG_0	TRA_0
CPMC		7	480	172	TAP	TAP
		8	480	172	TG_0	TRA_0
DMAIOC		9	3325	207	TAP	TAP
		10	3325	207	TG_0	TRA_0
CKReg		11	505	118	TAP	TAP
		12	505	118	TG_0	TRA_0
CDMC		13	224	86	TAP	TAP
		14	224	86	TG_0	TRA_0
TXC		15	364	140	TAP	TAP
		16	364	140	TG_0	TRA_0
CPM_i		17+i	239	80	TG_1	TRA_1
CDM_j		25+j	369	64	TG_1	TRA_1
DSP_n	LPM	17×n+35	46	16	$TG_{n,0}$	$TRA_{n,0}$
	LDM_l	17×n+l+36	92	8	$TG_{n,0}$	$TRA_{n,0}$
	LZM_m	17×n+m+40	23	2	$TG_{n,0}$	$TRA_{n,0}$
	$Logic_0$	17×n+42	4435	152	TAP	TAP
		17×n+43	4435	152	$TG_{n,1}$	$TRA_{n,1}$
	$Logic_1$	17×n+44	4435	152	TAP	TAP
		17×n+45	4435	152	$TG_{n,1}$	$TRA_{n,1}$
	$Logic_2$	17×n+46	7009	230	TAP	TAP
		17×n+47	7009	230	$TG_{n,1}$	$TRA_{n,1}$
	$Logic_3$	17×n+48	7224	250	TAP	TAP
		17×n+49	7224	250	$TG_{n,1}$	$TRA_{n,1}$
	$Logic_4$	17×n+50	7796	270	TAP	TAP
		17×n+51	7796	270	$TG_{n,1}$	$TRA_{n,1}$

Table A.6: Design characteristics Ericsson.

Block	X	Y	Block	X	Y
TG6	0	0	TG0	80	0
TG6L	10	0	TG0L	90	0
DSP6LDM1	20	0	DSP0LDM1	100	0
DSP6LDM2	30	0	DSP0LDM2	110	0
DSP6LDM3	0	10	DSP0LDM3	80	10
DSP6LDM4	10	10	DSP0LDM4	90	10
DSP6LPM	20	10	DSP0LPM	100	10
DSP6LZM1	30	10	DSP0LZM1	110	10
DSP6LZM2	0	20	DSP0LZM2	80	20
DSP6L1	10	20	DSP0L1	90	20
DSP6L2	20	20	DSP0L2	100	20
DSP6L3	30	20	DSP0L3	110	20
DSP6L4	0	30	DSP0L4	80	30
DSP6L5	10	30	DSP0L5	90	30
SA6	20	30	SA0	100	30
SA6L	30	30	SA0L	110	30
TG7	40	0	TG1	120	0
TG7L	50	0	TG1L	130	0
DSP7LDM1	60	0	DSP1LDM1	140	0
DSP7LDM2	70	0	DSP1LDM2	150	0
DSP7LDM3	40	10	DSP1LDM3	120	10
DSP7LDM4	50	10	DSP1LDM4	130	10
DS7LPM	60	10	DSP1LPM	140	10
DSP7LZM1	70	10	DSP1LZM1	150	10
DSP7LZM2	40	20	DSP1LZM2	120	20
DSP7L1	50	20	DSP1L1	130	20
DSP7L2	60	20	DSP1L2	140	20
DSP7L3	70	20	DSP1L3	150	20
DSP7L4	40	30	DSP1L4	120	30
DSP7L5	50	30	DSP1L5	130	30
SA7	60	30	SA1	140	30
SA7L	70	30	SA1L	150	30

Table A.7: Placement characteristics Ericsson.

Block	X	Y	Block	X	Y
TG4	0	60	TG2	80	60
TG4L	10	60	TG2L	90	60
DSP4LDM1	20	60	DSP2LDM1	100	60
DSP4LDM2	30	60	DSP2LDM2	110	60
DSP4LDM3	0	70	DSP2LDM3	80	70
DSP4LDM4	10	70	DSP2LDM4	90	70
DSP4LPM	20	70	DSP2LPM	100	70
DSP4LZM1	30	70	DSP2LZM1	110	70
DSP4LZM2	0	80	DSP2LZM2	80	80
DSP4L1	10	80	DSP2L1	90	80
DSP4L2	20	80	DSP2L2	100	80
DSP4L3	30	80	DSP2L3	110	80
DSP4L4	0	90	DSP2L4	80	90
DSP4L5	10	90	DSP2L5	90	90
SA4	20	90	SA2	100	90
SA4L	30	90	SA2L	110	90
TG5	40	60	TG3	120	60
TG5L	50	60	TG3L	130	60
DSP5LDM1	60	60	DSP3LDM1	140	60
DSP5LDM2	70	60	DSP3LDM2	150	60
DSP5LDM3	40	70	DSP3LDM3	120	70
DSP5LDM4	50	70	DSP3LDM4	130	70
DS5LPM	60	70	DSP3LPM	140	70
DSP5LZM1	70	70	DSP3LZM1	150	70
DSP5LZM2	40	80	DSP3LZM2	120	80
DSP5L1	50	80	DSP3L1	130	80
DSP5L2	60	80	DSP3L2	140	80
DSP5L3	70	80	DSP3L3	150	80
DSP5L4	40	90	DSP3L4	120	90
DSP5L5	50	90	DSP3L5	130	90
SA5	60	90	SA3	140	90
SA5L	70	90	SA3L	150	90

Table A.7: Placement characteristics Ericsson.

Block	X	Y	Block	X	Y
TG8b	0	40	CDM4	0	50
TG9b	10	40	CDM5	10	50
TG10	20	40	CDM6	20	50
CPM0	30	40	CDM7	30	50
CPM1	40	40	CDM8	40	50
CPM2	50	40	RX0C	50	50
CPM3	60	40	RX1C	60	50
CPM4	70	40	CPMC	70	50
CPM5	80	40	DSPIOC	80	50
CPM6	90	40	DMAIOC	90	50
CPM7	100	40	CDMC	100	50
CPM8	110	40	TXC	110	50
CPM9	120	40	CKREG	120	50
CDM1	130	40	SA8b	130	50
CDM2	140	40	SA10	140	50
CDM3	150	40	TAP	150	50

Table A.7: Placement characteristics Ericsson.

Appendix A

Bibliography

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

[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.

[Aer98] Joep Aerts and Erik Jan Marinissen, Scan Chain Design for Test Time Reduction in Core-Based ICs, *Proceedings of the International Test Conference*, pp. 448-457, Washington D.C. , 1998.

[Aho87] Alfred V. Aho, John E. Hopcroft and Jeffery D. Ullman, Data Structures and Algorithms, Addison-Wesley, ISBN 0-201-00023-7, 1983.

[Bar93] Jon Barwise and John Etchemendy, The Language of First-Order Logic, *CSLI Publications*, ISBN 0-937073-99-7, 1993.

[Ben00] Alfredo Benso, Silvia Cataldo, Silvia Chiusano, Paolo Prinetto, and Yervant Zorian, A High-Level EDA Environment for Automatic Insertion of HD-BIST Structures, *Journal of Electronic Testing: Theory and Applications*, Vol. 16, No. 3, pp. 179-184, June 2000.

[Ble93] Harry Bleeker, Peter van den Eijnden and Frans de Jong, Boundary-Scan Test: A Practical Approach, *Kluwer Academic Publishers*, ISBN 0-7923-9296-5, 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.

[Cha99] Krishnendu Chakrabarty, Test Scheduling for Core-Based Systems, *Proceedings of the International Conference on Computer-Aided Design*, pp. 391-394, 1999.

[Ch00a] Krishnendu Chakrabarty, Design of System-on-a-Chip Test Access Architecture under Place-and-Route and Power Constraints, *Proceedings of the Design Automation Conference*, pp. 432-437, 2000.

[Ch00b] Krishnendu Chakrabarty, Test Scheduling for Core-Based Systems using Mixed-Integer Linear Programming, *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, October 2000.

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

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

[Che90] Kwang-Ting Chen and Vishwani D. Agrawal, A Partial Scan Method for Sequential Circuits with Feedback, *Transactions on Computers*, Vol. 39., No. 4, pp. 544-548, 1990.

[Che92] Chung-Hsing Chen, BETA: Behavioral Testability Analyzer and its Application to High-Level Test Generation and Synthesis for Testability, *Ph.D. Dissertation*, Department of Electrical Engineering, University of Illinois at Urbana-Chapaign, 1992.

[Che93] Chung-Hsing Chen and Daniel G. Saab, A Novel Behavioral Testability Measure, *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1960-1993, Vol. 12, No. 12, December 1993.

[Che94] Chung-Hsing Chen, Tanay Karnik, and Daniel G. Saab, Structural and Behavioral Synthesis for Testability Techniques, *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 777-785, Vol. 13, No. 16, June 1994.

[Chi91] Vivek Chickermane and Janak H. Patel, A Fault Oriented Partial Scan Design Approach, *Proceedings of the International Conference on Computer-Aided Design*, pp. 400-403, Santa Clara, 1991.

[Chi92] Vivek Chickermane, Jaushin Lee, and Janak 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.

[Cho97] R. Chou, K. Saluja, V. Agrawal, Scheduling Tests for VLSI Systems Under Power Constraints, *Transactions on VLSI Systems*, Vol. 5, No. 2, pp. 175-185, June 1997.

[Cor00] Luis A. Cortés, Petru Eles, and Zebo Peng, Verification of Embedded Systems using a Petri Net based Representation, *Proceedings of the International Symposium on System Synthesis*, pp. 149-155, Madrid, Spain, September 20-22, 2000.

[Cra88] G. L. Craig, C. R. Kime, and K. K. Saluja, Test SchedulingandControlforVLSIbuilt-in-self-test,*IEEE Transactions on Computers*, Vol. 37, No. 9, pp. 1099-1109, September 1988.

[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, September 7-10, 1992.

[Eri00] Ericsson, *Design document*, 2000.

[Flo97] Marie-Lise Flottes, R. Pires, and Bruno 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 No. 273*, 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*, ISBN 0-7923-9194-2, 1992.

[Gar91] M. Garg, A. Basu, T.C. Wilson, D.K. Banerji, J.C. Majithia, A New Test Scheduling Algorithm for VLSI Systems, , *Proceddings of the Symposium on VLSI Design*, pp. 148-153, January 1991.

[Gar79] M. R. Garey and, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, *W. H. Freeman*, San Fransisco, 1979.

[Gho95] Indradeep Ghosh, Anand Raghunathan, and Niraj K. Jha, Design of Hierarchical Testability of RTL Circuits Obtained by Behavioral Synthesis, *Proceedings of the International Conference on Computer Design*, Austin, October 1995.

[Gir98] Patrick Girard, Christian Landrault, Serge Pravossoudovitch, and Daniel Severac, Reducing Power Consumption During Test Application By Test Vector Ordering, *Proceedings of the International Symposium on Circuits and Systems*, pp. 296-299, Vol. 2, Austin, May 31- June 3, 1998.

[Glo86] Fred Glover, Future paths for integer programming and links to artificial intelligence, *Computer and Ops. Res., 5*, pp. 533-549 1986.

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

[Gol80] Lawrence H. Goldstein and Evelyn L. Thigpen, Controllability/Observability Analysis of Digital Circuits, *Proceedings of the Computer-Aided Design*, pp. 190-196, Minneapolis, June 1980.

[Gon76] T. Gonzales and S. Sahni, Open shop scheduling to minimize finish time, *Journal of the ACM*, Vol. 23, pp. 665-679, October 1976.

[Gru00] Flavius Gruian, Energy-Aware Design of Digital Systems , *Licentiate Thesis No. 809*, Department of Computer and Information Science, Linköpings Universitet, 2000.

[Gup90] Rajesh Gupta, Rajiv Gupta, and Melvin A. Breuer, The BALLAST Methodology for Structured Partial Scan Design, *Transactions on Computers*, Vol. 39, No. 4, April 1990, pp. 538-544.

[Gu91] Xinli Gu, Krzysztof Kuchinski, and Zebo Peng Testability Measure with Reconvergent Fanout Analysis and Its Applications, *The Euromicro Journal, Microprocessing and Microprogramming*, Vol. 32, No. 1-5, pp. 835-842, August 1991.

[Gu92] Xinli Gu, Krzysztof Kuchinski, and Zebo Peng, An Approach to Testability Analysis and Improvement for VLSI Systems, *The Euromicro Journal, Microprocessing and Microprogramming*, Vol. 35, No. 1-5, pp. 485-492, September 1992.

[Gu94] Xinli Gu, Krzysztof Kuchinski, and Zebo Peng, Testability Analysis and Improvement from VHDL Behavioral 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, *Ph.D. Dissertation No. 414*, Department of Computer and Information Science, Linköping University, Sweden, 1996.

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

[Her98] A. Hertwig and H-J Wunderlich, Low Power Serial Built-In Self-Test, *Compendium of Papers of European Test Workshop*, pp. 49-53, Sitges, Spain, May 1998.

[Het99] Graham Hetherington, Tony Fryars, Nagesh Tamarapalli, Mark Kassab, Abu Hassan, and Janusz Rajski, Logic BIST for Large Industrial Designs: Real Issues and Case Studies, *Proceedings of the International Test Conference*, pp. 358-367, September 1999.

[Hsu96a] Frank F. Hsu, Elizabeth M. Rudnick, and Janak Patel, Testability Insertion in Behavioral Descriptions, *Proceedings of the International Symposium on System Synthesis*, pp. 139-144, La Jolla, November 1996.

[Hsu96b] Frank F. Hsu, Elizabeth M. Rudnick, and Janak Patel, Enhancing High-Level Control-Flow for Improved Testability, *Proceedings of the International Conference on Computer-Aided Design*, San Jose, November 1996.

[Hal98] Jonas Hallberg, Timing Issues in High-Level Synthesis, *Ph. D. Dissertation No. 555*, Department of Computer and Information Science, Linköpings Universitet, 1998.

[Håk98] Jan Håkegård, Hierarchical Test Architecture and Board-Level Test Controller Synthesis, *Licentiate Thesis No. 676*, Department of Computer and Information Science, Linköpings Universitet, 1998.

[Jer00] Gert Jervan, Zebo Peng and Raimund Ubar, Test Cost Minimization for Hybrid BIST, *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI*, pp 283-291, Yamanashi, Japan, October 25-27, 2000.

[Jig00] Razvan Jigorea, Sorin Manolache, Petru Eles, and Zebo Peng, Modeling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML, *Proccedings of the International Symposium on Object-oriented Real-time distributed Computing*, pp. 210-213, Newport Beach, March 2000.

[Jon89] Wen Ben Jone, C. A. Papachrisou, and M. Perieria, A Scheme for Overlaying Concurrent Testing of VLSI Circuits, *Proceedings of the Design Automation Conference*, pp. 531-536, 1989.

[Kim82] C. R. Kime and K. K. Saluja, Test Scheduling in Testable VLSI Circuits, *Proceedings of the International Symposium on Fault-Tolerant Computing*, pp. 406-412, 1982.

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

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

[Kir83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.

[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*, pp. 295-300, Vol. 28, No. 1-5, March 1990.

[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 Conference,* pp. 259-266, Tallinn, November 1997.

[Lar98a] Erik Larsson and Zebo Peng, Testability Analysis of Behavioral-Level VHDL Specifications, *Compendium of Papers of the European Test Workshop,* pp. 143-144, Sitges, Spain, May, 1998.

[Lar98b] Erik Larsson, High-Level Testability Analysis and Enhancement Technique, *Licentiate Thesis No. 725*, Department of Computer and Information Science, Linköping University, Sweden 1998.

[Lar99a] Erik Larsson and Zebo Peng, A Behavioral-Level Testability Enhancement Technique, *Compendium of Papers of the European Test Workshop*, Constance, Germany, May, 1999.

[Lar99b] Erik Larsson and Zebo Peng, An Estimation-based Technique for Test Scheduling, *Proceedings of the Electronic Circuits and Systems Conference*, pp. 25-28, Bratislava, September, 1999.

[Lar00a] Erik Larsson and Zebo Peng, System-on-Chip Test Bus Design and Test Scheduling, *International Test Synthesis Workshop*, Santa Barbara, March, 2000.

[Lar00b] Erik Larsson and Zebo Peng, A Technique for Test Infrastructure Design and Test Scheduling, *Proceedings of the Design and Diagnostics of Electronic Circuits and Systems Workshop*, Smolenice Castle, Slovakia, April 2000.

[Lar00c] Erik Larsson and Zebo Peng, Test Infrastructure Design and Test Scheduling Optimization, *Informal Digest of the European Test Workshop*, Cascais, Portugal, May 2000.

[Lar00d] Erik Larsson and Zebo Peng, An Integrated System-on-Chip Test Framework, *Accepted for the Design, Automation and Test in Europe Conference*, 2001.

[Lar00e] Erik Larsson and Zebo Peng, An Efficient Test Scheduling Technique for System-on-Chip, *Submitted for publication*.

[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 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. 292-297, Dallas, June 1993.

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

[Mar98] Erik Jan Marinissen, Robert Arendsen, Gerard Bos, A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores, *Proceedings of the International Test Conference*, pp. 284-293, October 18-23, 1998.

[Mar00] Erik Jan Marinissen, Sandeep Kumar Goel, and Maurice Lousberg, Wrapper Design for Embedded Core Test, *Proceedings of the International Test Conference*, paper 34.3, pp. 911-920, Atlantic City, October 3-5, 2000.

[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, DFTAdvisor User's and Reference Manual, *Mentor Graphics*, December, 1993.

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

[Mur00] Valentin Muresan, Xiaojun Wang, Valentina Muresan, and Mirecea Vladutiu, A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling, *Proceedings of the International Test Conference*, pp. 882-891, Atlantic City, October 3-5, 2000.

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

[P1500] IEEE P1500 Web site: http://grouper.ieee.org/groups/1500/.

[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, *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, pp. 661-679, June 1989.

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

[Pet81] James Lyle Peterson, Petri net theory and the modeling of systems, *Prentice-Hall, Inc.*, ISBN 0-13-661983-5, 1981.

[Pop00] Paul Pop, Petru Eles, and Zebo Peng, Bus Access Optimization for Distributed Embedded Systems Based on Schedulability Analysis, *Proceedings of Design, Automation and Test in Europe Conference*, pp. 567-574, Paris, March 27-30, 2000,

[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.

[Ree93] Colin R. Reeves, Modern Heuristic Techniques for Combinatorial Problems, *Blackwell Scientific Publications*, ISBN 0-632-03238-3, 1993.

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

[Ses00] Sandhya Seshadri and Michael S. Hsiao, Formal Value-Range and Variable Testability Technique, *Journal of Electronic Testing, Theory and Applications*, pp. 131-145, Vol. 16, No.1/2, February/April 2000.

[Ste00] Andreas Steiningerl, Testing and Built-In Self-Test - A Survey, *Journal of System Architecture*, ISSN-1383-7621, pp. 721-747, Vol. 46, No. 9, July 2000.

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

[Sug98] Makoto Sugihara, Hiroshi Date and Hiroto Yasuura, A Test Methodology for Core-Based System LSIs, *IEICE Transactions on Fundamentals*, pp. 2640-2645, Vol. E81-A, No. 12, December 1998.

[Tho94] Thomas Thomas, Praveen Vishakantaiah and Jacob A. Abraham, Impact of Behavioral Modifications for Testability, *Proceedings of the VLSI Test Symposium*, pp. 427-432, New Jersey, April 1994.

[Tri85] Howard Trickey, Compiling Pascal Programs into Silicon, *Ph.D. Dissertation*, 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 Design Automation Conference*, pp. 490-496, Maimi 1983.

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

[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.

[Var98] Prab Varma and Sandeep Bhatia, A Structured Test Re-Use Methodology for Core-Based System Chips, *Proceedings of the International Test Conference*, pp. 294-302, Washington DC, October 1998.

[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.

[Wes92] Neil H. E.Weste and Kamran Eshraghian, Principles of CMOS VLSI Design, *Addison-Wesley*, ISBN 0-201-53376-6, 1992.

[Yan98] Tianruo Yang and Zebo Peng, An Efficient Algorithm to Integrate Scheduling and Allocation in High-Level Synthesis, *Proceedings of the Design Automation and Test in Europe Conference*, pp. 74-81, Paris, February 1998.

[Zor93] Yervant Zorian, A distributed BIST control scheme for complex VLSI devices, *Proceedings of the VLSI Test Symposium*, pp. 4-9, April 1993.