Linköping Studies in Science and Technology

Dissertation No. 1222

Test Optimization for Core-based System-on-Chip

by

Anders Larsson

INSTITUTE OF TECHNOLOGY
LINKÖPING UNIVERSITY

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

Linköping 2008

ISBN 978-91-7393-768-9, ISSN 0345-7524
Printed in Linköping, Sweden
by LiU-Tryck

Abstract

THE SEMICONDUCTOR TECHNOLOGY has enabled the fabrication of integrated circuits (ICs), which may include billions of transistors and can contain all necessary electronic circuitry for a complete system, so-called System-on-Chip (SOC). In order to handle design complexity and to meet short time-to-market requirements, it is increasingly common to make use of a modular design approach where an SOC is composed of pre-designed and pre-verified blocks of logic, called cores.

Due to imperfections in the fabrication process, each IC must be individually tested. A major problem is that the cost of test is increasing and is becoming a dominating part of the overall manufacturing cost. The cost of test is strongly related to the increasing test-data volumes, which lead to longer test application times and larger tester memory requirement. For ICs designed in a modular fashion, the high test cost can be addressed by adequate test planning, which includes test-architecture design, test scheduling, test-data compression, and test sharing techniques.

In this thesis, we analyze and explore several design and optimization problems related to core-based SOC test planning. We perform optimization of test sharing and test-data compression. We explore the impact of test compression techniques on test application time and compression ratio. We make use of analysis to explore the optimization of test sharing and test-data compression in conjunction with test-architecture design and test scheduling. Extensive experiments, based on benchmarks and industrial designs, have been performed to demonstrate the significance of our techniques.

Acknowledgements

THERE ARE MANY people, who in one way or another, have contributed to this thesis. First and foremost, a special thank you goes to my primary advisor Associate Professor Erik Larsson for all your support, inspiration, and seemingly endless patience. I also direct a special thank you to my secondary advisors Professor Zebo Peng and Professor Petru Eles. Together with Erik, you made a great team of advisors and have provided excellent guidance in how to obtain and present research results. I would also like to thank Professor Krishnendu Chakrabarty who gave me such inspiration and who was the perfect host during my stay at Duke University, USA. I would also like to thank my master thesis project student Xin Zhang for your valuable contributions regarding the implementation in Chapter 8.

The importance of kind and supportive colleagues cannot be exaggerated. Therefore, I would like to thank all present and former members of the Embedded Systems Laboratory (ESLAB) and of the Department of Computer and Information Science (IDA) at Linköping University.

To all my friends and relatives; thank you for asking and for trying to understand my research topic. But foremost, thank you for being you.

Finally, I would like to thank my family, Lars, Birgit, and Peter, for all your support and encouragements. I could not have done this without you. Last but not least, thank you Linda for all the love you give.

Anders Larsson
Linköping, November 2008

Contents

Chapter 1
Introduction

THIS CHAPTER INTRODUCES and motivates the System-on-Chip (SOC) test problem. It contains a list of the contributions and a description of the organization of the rest of the thesis.

1.1 Introduction and Motivation

Integrated circuits (ICs) are embedded nowadays in a wide range of products and systems, from consumer electronics and medical equipment to automotive and aviation systems, which usually require high availability and where the cost of failures can be immense. There has been an amazing development of ICs. The first IC available commercially was produced by Fairchild Semiconductor Corp. in 1961; it contained one transistor, three resistors and one capacitor. The everlasting improvements in semiconductor fabrication technology have led to ICs with billions of transistors. Such large ICs can contain all necessary electronic circuitry for a complete system and are referred to as SOCs. A typical SOC consists of components such as processors and peripheral devices including data transformation engines, data ports, and controllers [Cha99].

ICs can be extremely complex and time-consuming to design. In order to meet short time-to-market requirements, it is therefore common to make use of a modular core-based design approach where a system is composed of pre-

designed and pre-verified blocks of logic, so-called cores. The cores can be designed in-house or bought from core vendors, and it is the task of the system integrator to integrate them into a system.

The IC fabrication process is far from perfect and defects such as shorts to power or ground, extra materials, etc., may appear as faults and cause failures. Therefore, each manufactured IC needs to be tested. The aim of fabrication test is to ensure that the fabricated IC is free from manufacturing defects.

The general approach to test is to apply test stimuli and compare the produced responses against the expected ones. Due to the complexity of the test process, a design approach, so-called design-for-testability (DFT), aimed at making the IC more easily tested has been proposed. As each fabricated IC is tested, it is important to minimize the test application time. For example, let us assume an IC that has a test application time of 10 seconds and is fabricated in 1 million copies. The total test application time for these ICs will be 116 days. A saving of 1 second per IC leads to a reduction of the total test time with 12 days.

For modular designs, it is possible to perform modular testing where each core is tested as an individual unit. Modular test is an attractive test solution since not only the cores are reused but also their test-data. However, the designers at the core vendor have little or no information about where their cores will be placed on a SOC. It is, therefore, usually assumed that the core is directly accessible and it becomes the task of the system integrator to ensure that the logic surrounding the core allows the test stimuli to be applied and the produced responses to be transported for evaluation. In modular testing, the system integrator is faced by a number of challenges, such as test-architecture design and test scheduling.

The increasing cost for IC testing is in part due to the huge test-data volume (number of bits), test stimuli and expected responses, which can be in the order of tens of gigabits. The huge test-data volume leads to long test application time and requires large tester memory. The 2007 International Technology Roadmap for Semiconductors (ITRS) predicts that the test-data volume for ICs will be as much as 38 times larger in 2015 than it is today [Sem07]. Furthermore, the number of transistors in a single IC is growing faster than the number of I/O pins, i.e., the ratio of transistors per I/O pin is growing. This trend leads to increased test application time since more test-data have to be applied through the limited number of I/O pins. The 2007

ITRS predicts that the test application time for ICs will be about 17 times longer in 2015 than it is today [Sem07].

The importance of reducing the cost of test is further motivated by comparing the test cost with the cost of fabrication. Figure 1.1 is adapted from ITRS 1999 [Sem99] and ITRS 2001 [Sem01], and shows how the relative cost of test grows compared to the fabrication cost per transistor. As can be seen in Figure 1.1, the actual cost of test is almost constant while the cost of fabrication has been dramatically reduced over the recent years. Today, the cost of test is a significant part of the overall manufacturing cost (including the cost of fabrication and the cost of test).

The high test cost for core-based SOCs can be reduced by adequate test planning, which includes:

- test-architecture design,
- test scheduling,
- test-data compression, and
- test sharing.

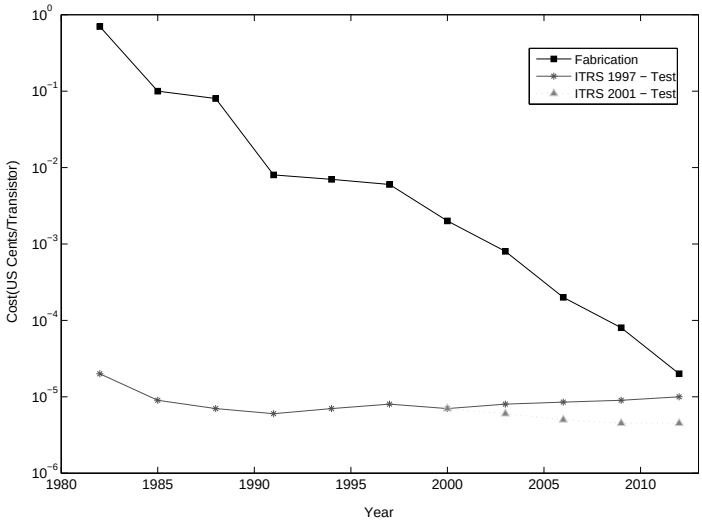


Figure 1.1: Trend in test cost versus fabrication cost per transistor.

Test-architecture design refers to the design of the hardware components that are added to achieve core isolation and core access. For example, a wrapper is usually placed around each core to achieve core access, core isolation, and to facilitate test reuse. However, the wrappers alone do not solve the test access problem, there also exists the requirement for a test access mechanism (TAM). The TAM is used for the transportation of test stimuli from the tester to the cores and of the produced responses from the cores to the tester. A TAM can be implemented by direct connections between the core terminals and the chip I/O pins, a dedicated test bus, or a functional bus.

Wrappers and TAMs are examples of test-architecture components that are added to the design to achieve modularity and efficient test-data transportation. Other examples are buffers, multiplexers, and test controllers. An adequate test-architecture potentially reduces the test application times, e.g., multiple TAMs enable concurrent test application at multiple cores, but also generates certain hardware overhead. Hence, there exists a trade-off between the amount of test-architecture that is added and the test application time. Throughout the rest of this thesis, the term test-architecture design will be used for the combined wrapper and TAM design problems.

Test scheduling is to assign the start time of each test. That is, to organize the test-data in the tester memory and to assign tests to TAMs such that some predefined cost function, e.g, the test application time, is minimized. By exploring different start times for each test it is possible to minimize the cost function while ensuring that constraints, such as hardware overhead and memory requirement, are not violated. The test scheduling can be combined (co-optimized) with the test-architecture design, e.g. by, exploring the trade-off between the test application time and the required number of TAM wires.

Test-data compression has been proposed to reduce the test-data volume and the test application time. The test-data consists of a high number of unspecified bits, so-called don't-care bits, which, together with regularities in the test-data can be explored during the compression, such that a minimal amount of test-data needs to be stored in the tester memory. The test application time can be reduced if decoders are placed on-chip, since the amount of test-data to be applied through the chip I/O pins is reduced.

In test sharing, overlapping sequences from several tests are used to create a new test. Similar to test-data compression, the general scheme of test sharing is to utilize regularities and the high number of don't-care bits in the test-data such that the shared test will have a minimal amount of test-data to be stored

in the tester memory. For test sharing, the test application time can be lowered if a TAM design that enables broadcasting of shared test is applied, since the shared test can be used to test multiple cores in parallel.

To summarize, the SOC test planning problem can be divided into four parts: (1) test-architecture design, (2) test scheduling, (3) test-data compression, and (4) test sharing. Each of the four parts is an optimization problem that is complex and hard to solve by itself. However, the optimal test plan can only be generated by considering all, or a majority of, the problems at the same time. In fact, the SOC test planning problem has been shown to belong to the group of NP-complete problems. Common for all NP-complete problems is that the execution time of algorithms to solve them optimally grows exponentially with respect to the problem size. Therefore, different optimization techniques are usually used to explore the search space for a solution with a minimized cost function. Such optimization techniques can be either exact or non-exact. Exact optimization techniques, e.g., branch and bound and constraint logic programming (CLP), will always find the optimal solution. Even if the search space can be reduced, the time to find the optimal solution using exact techniques is often too long. Therefore, non-exact optimization techniques (so-called heuristics), based on e.g., Tabu search and Simulated annealing, have been used to find sub-optimal solutions.

1.2 Contributions

In this thesis the increasing cost of test for core-based SOCs is targeted by reducing the test application time, the test-data volume, and the test-architecture hardware overhead.

Assumed is a system consisting of a number of cores where each core is delivered together with one given dedicated test. The SOC test planning problem is solved such that the given cost function is minimized. The trade-off between the test-architecture hardware overhead and the test application time is explored. The main contributions of this thesis are as follows:

- Test sharing and broadcasting of tests for core-based SOCs are addressed. The possibility to share tests, i.e. finding overlapping sequences in several tests, which are used to create a common test, is explored. The proposed technique is used to select suitable tests, individual or shared, for each core in the system and schedule the selected tests such that the test

application time is minimized under a test-architecture hardware cost constraint [Lar05b], [Lar05c], [Lar05d], [Lar06a], [Lar08e].

- The relation between test-data compression and test sharing in terms of test-data volume is explored. Since the shared test will have less don't-care bits, it is likely that it will suffer from a lower compression ratio compared to when the tests are compressed individually. This means that the size of the compressed shared test could be larger than the sum of the two separately compressed tests. The trade-off between test sharing and test-data compression in terms of test application time is explored in order to solve the SOC test planning problem. The test application time is minimized under test-architecture hardware cost and ATE memory constraints [Lar07a], [Lar07b], [Lar08d].
- For each core and its decoder, we show that the test application time does not decrease monotonically with the increasing TAM width at the decoder input or with the increasing number of wrapper chains at the decoder output. Therefore, there is a need to include the optimization of the wrapper and decoder designs for each core, in conjunction with the test-architecture design and the test scheduling at the SOC-level. A test-architecture design and test scheduling technique for SOCs that is based on core-level expansion of compressed test-data is proposed. Two optimization problems are formulated: test application time minimization under a TAM width constraint and TAM width minimization under a test application time constraint [Lar08a], [Lar08f].
- The analysis of the test application time and test-data compression ratio for different test-data compression techniques shows that the test application time and the compression ratio are not only TAM width dependant but also test-data compression technique dependant. It is, therefore, not trivial to select the optimal test-data compression technique and TAM width for a core. The overall test-data volume and test application time are minimized by test-architecture design, test scheduling, and test-data compression technique selection [Lar08b], [Lar08c].
- A test-architecture to address TAM underutilization is proposed where buffers are inserted between each core and the functional bus. A test controller is also introduced, which is responsible for the invocations of tests. The test-architecture hardware overhead due to the buffers and the

test controller is minimized such that a given test application time is not exceeded [Lar03a], [Lar04a], [Lar04b], [Lar05a], [Lar05c].

Below follows a complete list of publications by the author of this thesis which are directly related to this thesis:

- [Lar03a]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Buffer and Controller Minimisation for Time-Constrained Testing of System-On-Chip," *In Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems* (*DFT*), pp. 385–392, Boston, MA, USA, November 3–5, 2003.
- [Lar04a]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Technique for Optimization of System-on-Chip Test Data Transportation," *IEEE European Test Symposium (ETS) (Informal Digest)*, Ajaccio, Corsica, France, May 23–26, pp. 179–180, 2004.
- [Lar04b]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Technique for Optimisation of SOC Test Data Transportation," *Swedish System-on-Chip Conference (SSoCC) (Informal Digest)*, Båstad, Sweden, April 13–14, 2004.
- [Lar05a]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Constraint Logic Programming Approach to SOC Test Scheduling," *Swedish System-on-Chip Conference (SSoCC) (Informal Digest)*, Tammsvik, Stockholm, Sweden, April 18–19, 2005.
- [Lar05b]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Optimization of a Bus-based Test Data Transportation Mechanism in System-on-Chip," *In Proceedings of Euromicro Conference on Digital System Design* (*DSD*), pp. 403–409, Porto, Portugal, August 30–September 3, 2005.
- [Lar05c]: A. Larsson, "System-on-Chip Test Scheduling and Test Infrastructure Design," *Licentiate Thesis No. 1206*, Dept. of Computer and Information Science, Linköping University, ISBN: 91-85457-61-2, November 2005.
- [Lar05d]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "SOC Test Scheduling with Test Set Sharing and Broadcasting," *In Proceedings of Asian Test Symposium* (*ATS*), pp. 162–167, Kolkata, India, December 18–21, 2005.

- [Lar06a]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "SOC Test Scheduling with Test Set Sharing and Broadcasting," *Swedish System-on-Chip Conference (SSoCC) (Informal Digest)*, Kolmården, Sweden, May 4–5, 2006.
- [Lar07a]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Optimized Integration of Test Compression and Sharing for SOC Testing," *In Proceedings of Design, Automation, and Test in Europe Conference* (*DATE*), pp. 207–212, Nice, France, April 16–20, 2007.
- [Lar07b]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Heuristic for Concurrent SOC Test Scheduling with Compression and Sharing," *In Proceedings of Workshop on Design and Diagnostics of Electronic Circuits and Systems* (*DDECS*), pp. 61–66, Krakow, Poland, April 11–13, 2007.
- [Lar08a]: A. Larsson, E. Larsson, K. Chakrabarty, P. Eles, and Z. Peng, "Test-Architecture Optimization and Test Scheduling for SOCs with Core-Level Expansion of Compressed Test Patterns," *In Proceedings of Design, Automation, and Test in Europe* (*DATE*), pp. 188–193, Munich, Germany, March 10–14, 2008.
- [Lar08b]: A. Larsson, X. Zhang, E. Larsson, and K. Chakrabarty, "SOC Test Optimization with Compression Technique Selection," *Accepted for publication as a poster at the International Test Conference (ITC)*, Santa Clara, California, USA, October 28–30, 2008.
- [Lar08c]: A. Larsson, X. Zhang, E. Larsson, and K. Chakrabarty, "Core-Level Compression Technique Selection and SOC Test-Architecture Design," *Accepted for publication at the Asian Test Symposium (ATS)*, Sapporo, Japan, November 24–27, 2008.
- [Lar08d]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "SOC Test Optimization with Test Compression and Sharing," *Submitted to Journal of Electronic Testing: Theory and Applications (JETTA)*, 2008.
- [Lar08e]: A. Larsson, E. Larsson, P. Eles, and Z. Peng, "System-on-Chip Test Planning with Shared Tests," *Submitted to Journal IET Computers & Digital Techniques,* 2008.
- [Lar08f]: A. Larsson, E. Larsson, K. Chakrabarty, P. Eles, and Z. Peng, "SOC Test Planning with Core-Level Expansion of Compressed Test

Patterns, ” *Submitted to Journal of Electronic Testing: Theory and Applications (JETTA)*, 2008.

- [Lar08g]: A. Larsson, X. Zhang, E. Larsson, and K. Chakrabarty, “Optimized Test Architecture Design and Test Scheduling with Core-Level Compression Technique Selection for System-on-Chip,” *Submitted to IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems,* 2008.

1.3 Thesis Organization

The rest of the thesis is structured as follows:

Chapter 2 gives background information regarding core-based SOC design and test. Chapter 3 contains the related work that is either used in, or directly related to this thesis. Chapter 4 contains preliminaries common for the rest of the thesis.

Chapter 5 describes how test sharing and broadcasting can be used to reduce the test application time. Shared tests are generated and added as alternatives to the initially dedicated tests for the cores, and if a shared test is selected, it is transported to the cores in a broadcasted manner such that several cores are tested concurrently. For test-data transportation, a test-architecture is described, which makes use of the functional bus and added dedicated test buses. The test application time is minimized under a test-architecture hardware overhead constraint [Lar05b], [Lar05c], [Lar05d], [Lar06a], [Lar08e].

Chapter 6 describes the problem with test-architecture design and scheduling where test-data compression and test sharing are included. The work in this chapter is concentrated on the following: the relation between compression and sharing in terms of test-data volume, and the trade-off between test sharing versus test-architecture design in terms of test-application time. The test application time is minimized under test-architecture hardware overhead and ATE memory constraints [Lar07a], [Lar07b], [Lar08d].

Chapter 7 describes a test-architecture design and test scheduling technique for SOCs that is based on core-level expansion of compressed test-data. The optimization of the wrapper and decoder designs for each core are integrated with the test-architecture design and the test scheduling at the SOC-level. Two

optimization problems are formulated: test application time minimization under a TAM width constraint and TAM width minimization under a test application time constraint [Lar08a], [Lar08f].

Chapter 8 describes an analysis that highlights the impact of test-data compression technique on test application time and compression ratio are compression method dependant as well as TAM-width dependant. A technique is proposed where test-architecture design and scheduling are integrated with test-data compression technique selection for each core in order to minimize the SOC test application time and the test data volume [Lar08b], [Lar08c], [Lar08g].

Chapter 9 describes a test-architecture where buffers are inserted between each core and the functional bus to address underutilization of the TAM. A test controller, which is responsible for the invocations of tests is also inserted. The hardware overhead due to the buffers and the test controller is minimized under a test application time constraint [Lar03a], [Lar04a], [Lar04b], [Lar05a], [Lar05c].

Chapter 10 concludes this thesis and discuss possible directions of future work.

Chapter 2
Background

THIS CHAPTER PRESENTS the background related to this thesis. The chapter starts with a description of the IC design and fabrication process, which is followed by an introduction of the core-based SOC design flow. The following two sections describe the test process and core-based SOC test. Finally, an introduction to optimization techniques is presented and two optimization techniques, CLP and Tabu search, are described.

2.1 IC Design and Fabrication Process

The overall goal in IC design and fabrication is to produce ICs that contain more functionality, are faster, and have better performance, all for less cost and in less time [DeM94].

The IC design and fabrication process is illustrated in Figure 2.1. After each IC design stage, simulations are performed and the stage is repeated until the IC design meets the specification. The IC design process usually consists of the following four stages:

- behavioral synthesis,
- logic synthesis,

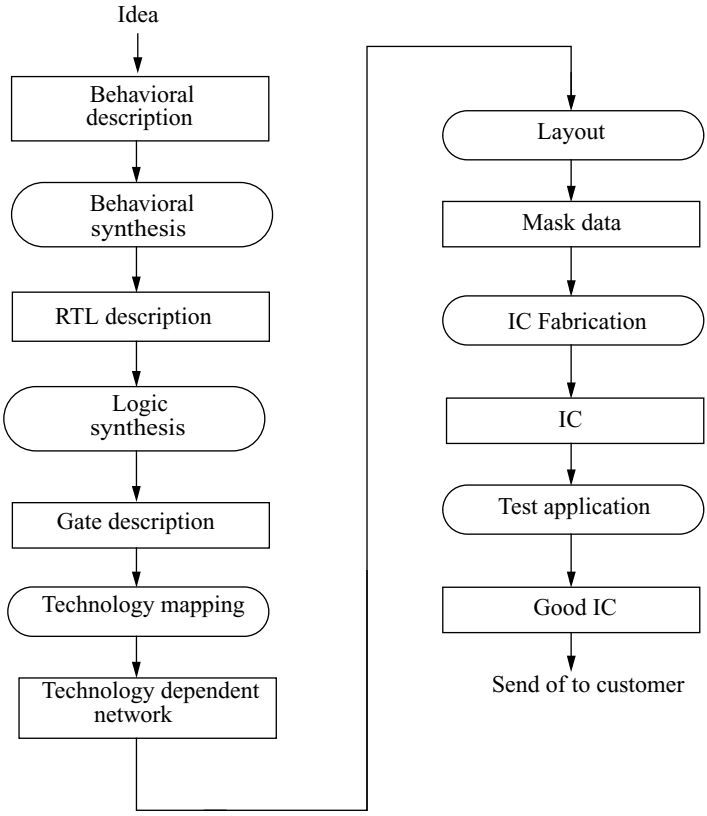


Figure 2.1: IC design and fabrication process [Mou00].

- technology mapping, and
- layout.

The IC fabrication usually consists of the following two stages:

- IC fabrication and
- test application.

From the first idea, a behavioral description is generated that describes the functionality of the IC. This description is usually written in a high level language. The behavioral synthesis takes as input a behavioral description file

and generates as output a register-transfer level (RTL) description. The RTL description specifies the flow of signals between registers, and the logical operations. The RTL specification is then used as input for the logic synthesis stage where the IC design is transformed into an implementation consisting of logic gates.

At the technology mapping stage, the transformation from gate level to physical level is performed. The IC design is transformed, during the layout stage, into layout masks that are used during the IC fabrication stage. The layout mask is used to construct the ICs through a delicate wafer fabrication process. This process is very sensitive to impurities due to the extremely small feature size, which is in the nano-scale, and despite various precautions such as clean-rooms and multiple calibrations, defects will occur. Therefore, the test application stage is used to detect defects introduced during the IC fabrication stage. Those ICs that pass this stage can be shipped to customers.

2.2 Core-Based SOC Design Flow

The core-based SOC design flow makes it possible to design ICs with multi-million gates and still meet the short time-to-market requirements.

The development of a core-based SOC is in many ways similar to the development of a System-on-a-Board (SOB). In a SOB, ICs from different IC providers are mounted on a printed circuit board and interconnected into a system. The different ICs such as processors and memories can without modification easily be reused in many different systems and products. In the core-based SOC design flow, system integrators have adopted the same reuse-based philosophy to use cores (blocks of logic), which are integrated into a system [Gup97].

The cores, which can be processors, memories, controllers, data ports, etc., are provided by various core vendors or they can be designed in-house components. For the interconnect architecture that connects the cores, the bus-based architecture is the most widely used [Pol03]. Several commercial functional buses have been developed such as CoreConnect [IBM05] from IBM, and the Advanced Microcontroller Bus Architecture (AMBA) [ARM08] from ARM.

An example of a fabricated core-based SOC is illustrated in Figure 2.2. This SOC, named PNX8550 from Nexperia, is used in set-top boxes and

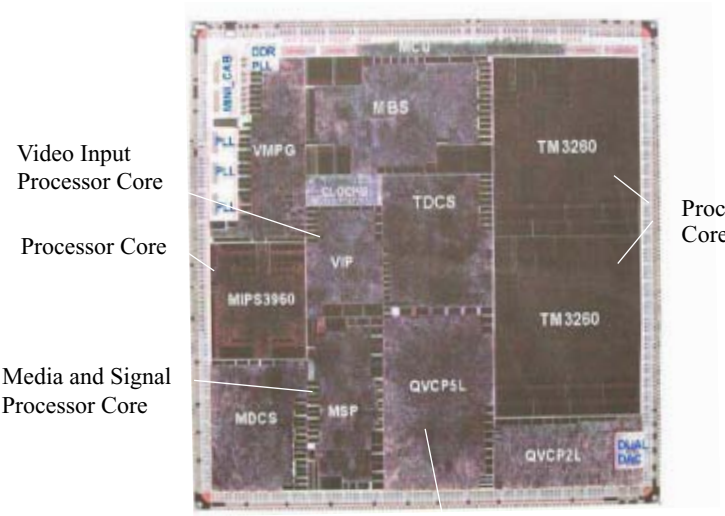


Figure 2.2: Core-based SOC layout, PNX8550 [Goel04].

digital TVs, and consists of more than 60 cores including processors, video input processor, media and signal processors, graphical processors, etc. The different cores can be easily identified as separate boxes in the layout. Such boxes will be used to represent cores throughout the rest of the thesis like in Figure 2.3, which shows an example of a core-based SOC that consists of four cores c_1, c_2, c_3, and c_4, connected to a functional bus bf_1.

2.3 Test Process

IC fabrication is far from perfect. Therefore, all ICs are tested to detect defects that might have been introduced during the fabrication process [Mou00]. The test process can be divided into the following two stages: (1) the test generation and (2) the fabrication test (test application).

Let us first describe how physical defects, such as extra or missing material, caused by dust particles on the mask, wafer surface or processing chemicals, can be detected. Physical defects manifest themselves at the electrical (circuit) level as failure modes, such as opens, shorts, and parameter degradations [Mou00]. Fault models are used to represent the effect of a failure. The effect

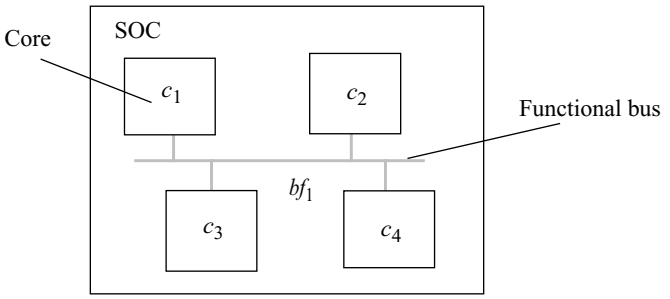


Figure 2.3: Core-based SOC with four cores: c_1, c_2, c_3, and c_4, and one functional bus bf_1.

of a failure will, at the logical level, appear as incorrect signal values. That is, how the signal is changed in the presence of a fault. One of the earliest, and most popular fault models today, is the stuck-at fault model, proposed by Eldred in 1959 [Eld59]. According to the stuck-at fault model, a defect will cause one line in the design to permanently be stuck at logic value 0 (stuck-at 0) or 1 (stuck-at 1). A stuck-at 0 fault, present at a given fault location, is detected when the stimulus data applied is a 1. The produced response will be a 0 (since the fault location is stuck at 0), which will be different from the expected response which is a 1, hence the fault is detected.

At test generation, an automatic test pattern generator (ATPG) is usually used to generate test-data for the design, including test stimuli and expected responses. The netlist (layout) of the design is given as an input to the ATPG-tool which uses sophisticated algorithms to analyze the design and generate test patterns for it. Examples of such test pattern generation algorithms are the D-algorithm [Roth67] and PODEM [Goel83].

At test application (fabrication test), it is required that the test stimuli can be applied to any given location from the inputs and that the produced responses can be propagated from any given location to the outputs. Hence, two of the most important properties of test is the observability and the controllability. The controllability is the ability of controlling the logic value at a specific location in the IC design. The observability is the ability to observe a logical value at any part of the IC design. The controllability is high for the locations close to the inputs while it is low for the locations close to the outputs. For the

observability the opposite is true, the observability is low for the locations close to the inputs and it is high for the locations close to the outputs. An IC design with 5 flip-flops (FFs), FF_1, FF_2, FF_3, FF_4, and FF_5, and a location with low controllability is illustrated in Figure 2.4.

To test an IC is a complex task, even for small ICs. In order to reduce this complexity, we can increase the controllability and observability of an IC during the design stages by adding testability features. This process is called DFT and is, usually, automatically performed using specialized design tools.

The DFT is performed in conjunction to the behavioral and logic synthesis stages in Figure 2.1. During the test pattern generation stage, the test-data used to test the fabricated IC is developed. A fault simulator is used to verify the test patterns and to measure the fault coverage. If the fault coverage is low, DFT is repeated until an acceptable fault coverage has been achieved.

The general aim of DFT is to increase the testability of an IC. Usually, DFT introduces a certain area and performance overhead. For example, it is possible to increase the observability and the controllability by inserting a direct connection, a so-called test point, between the hard-to-test fault location and an I/O pin. The test point DFT approach is straightforward, however, it does not scale as the number of hard-to-test fault locations is increased.

A more scalable DFT-technique is to use scan chain insertion, first introduced by Kobayashi *et al*. [Kob68] and later described by Williams and

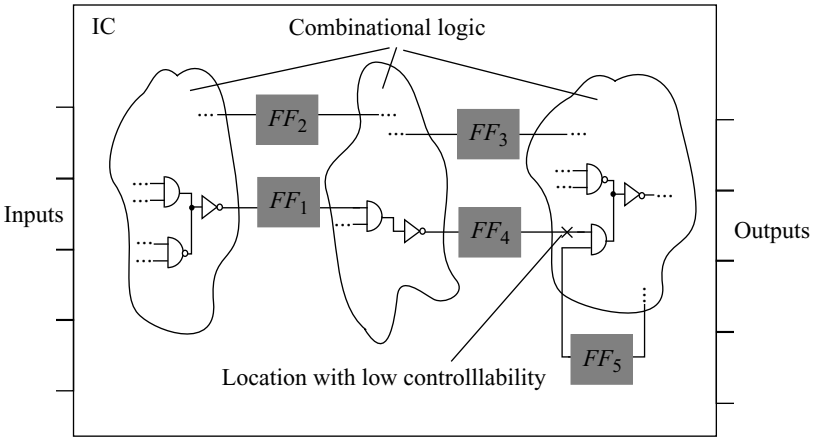


Figure 2.4: IC design and a hard-to-test location.

Parker [Wil83]. Today, scan chain design is a widely adopted DFT-technique. To make a design scanable, the FFs in the design are modified with one additional scan input, one additional scan output, and one scan enable input. The scan-modified FFs are then connected in shift registers, so-called scan chains.

In Figure 2.5, the 5 FFs in the design in Figure 2.4 have been scan-modified and connected into one scan chain. (The scan enable is not illustrated for reasons of readability.) Two additional I/O pins, $sc\text{-}in_1$ and $sc\text{-}out_1$, are added for the test stimuli shift-in and the produced responses shift-out, respectively. The location with low controllability in Figure 2.4 is now controllable from FF_4 by using the scan chain.

Scan chain testing implies that the design has two modes: functional mode and test mode. The flow of a scan cycle is as follows:

- Assert test mode, shift in test stimuli (scan-in phase) and set up the desired inputs.
- Assert functional mode and apply one clock cycle. The produced responses are now captured in the FFs and at the outputs.
- Assert test mode and shift out the produced responses (scan-out phase).

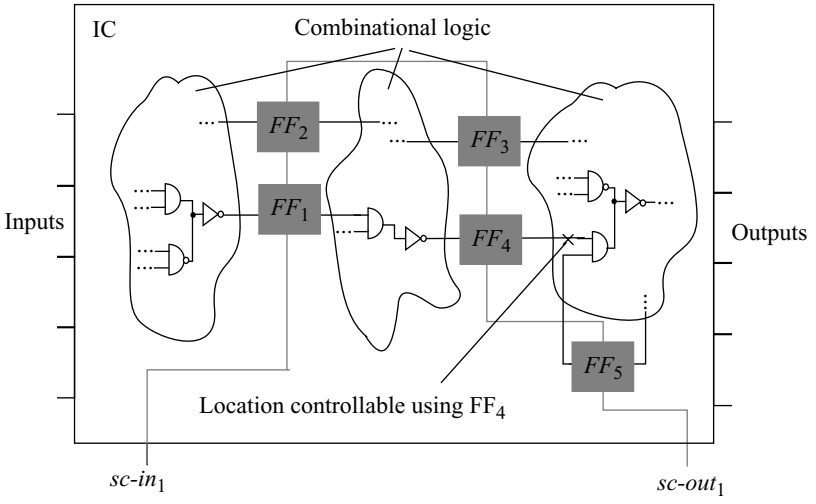


Figure 2.5: IC design with hard-to-test location controllable using one scan chain.

The test-data corresponding to the bits required for a full test stimuli shift-in, apply and capture, and shift-out of the produced responses is called a test pattern. For efficient test application, the test stimuli of the following test pattern are shifted in while the produced responses from the current test pattern are shifted out, that is, a concurrent scan-in and scan-out phase is performed. The scan test application is illustrated in Figure 2.6 using two test patterns, tp_1 and tp_2, which are applied to the IC design in Figure 2.5. The test application time for the two test patterns is 17 clock cycles. The test application time $\tau(sc)$ (number of clock cycles) for a test T used to test an IC with sc scan chains is as follows:

$$\tau(sc) = (1 + ff) \times l + ff, \tag{2.1}$$

where l is the number of test patterns that are applied and ff is the length of the longest scan chain among the sc scan chains. The rate at which the test-data is shifted is given by the scan frequency, f_{scan}.

The test application time can be lowered by using multiple scan chains as illustrated in Figure 2.7. Figure 2.7(a) shows a scan design where the 5 FFs in Figure 2.4 have been connected in one scan chain of length 5. Figure 2.7(b) shows a scan design where the 5 FFs have been connected in two scan chains, one of length 3 and one of length 2. Let us assume the IC is tested using four test patterns ($l = 4$). The test application time will be $(5 + 1) \times 4 + 5 = 29$ clock cycles for the scan design in Figure 2.7(a) and $(3 + 1) \times 4 + 3 = 19$ clock cycles for the scan design in Figure 2.7(b).

An illustration of the second stage of the test process, the fabrication test, is given in Figure 2.8. Fabrication test is usually performed using an automatic test equipment (ATE). The test stimuli and expected responses are stored in the ATE memory. Testing is performed by applying test stimuli to the device

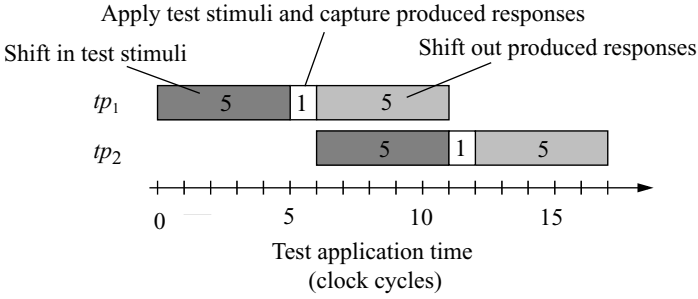


Figure 2.6: Scan test application.

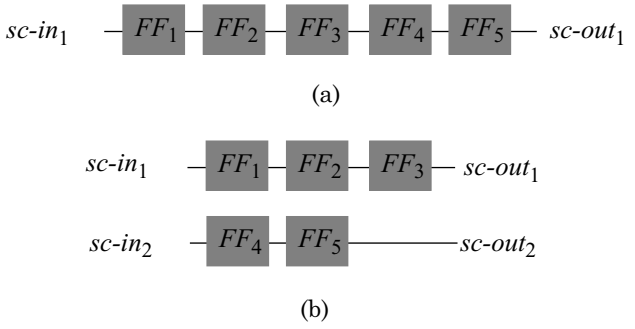


Figure 2.7: Scan chain design with (a) one scan chain and with (b) two scan chains.

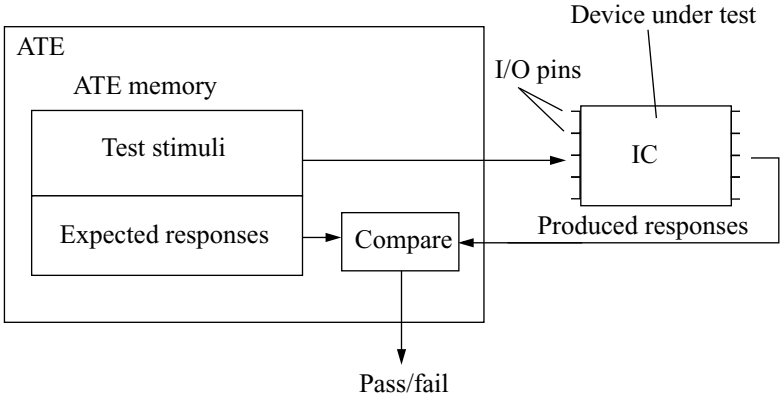


Figure 2.8: IC fabrication test process using ATE.

under test, and by comparing the produced responses to the expected ones. A difference between the expected response and the produced ones indicates that a fault is present and that the device under test should be discarded. The rate at which the test-data is applied is given by the operating frequency of the ATE, f_{ATE}.

An alternative to the ATE is to use built-in self-test (BIST). BIST is a technique where testing (test generation and test application) is performed through built-in hardware (and software) features. BIST enables in-field test

and reduces the dependency of expensive ATEs. However, BIST also contributes to hardware overhead and the quality, fault coverage, is not as high as for ATPG generated tests. For test pattern generation with BIST it is common to use a linear feedback shift register (LFSR) [Bar87] or to store pre-generated test patterns in memory. The produced responses need to be compacted, which can be done in the spatial and/or time domain [Mur96]. A multiple input signature register (MISR) is an example of a compactor in the time domain and a combinational (usually XOR network-based) compactor is an example for the space domain. In the case when MISRs are used, at the end of the testing the MISR signature is shifted out and compared with the expected signature.

2.4 Core-based SOC Test

In this section, the core-based SOC test approach with test planning is described. A core-based SOC can be tested in a modular fashion. Modular test is achieved by isolating each core in the SOC and by providing a TAM for transporting the test stimuli from the tester to the cores and the produced responses from the cores to the tester. The test-architecture design together with test scheduling and organization of the test-data in the ATE memory should be performed in such way that the test application can be done in a plug-and-play fashion. In this section we introduce test-architecture design, test scheduling, test sharing, and test-data compression.

By using a modular test approach it is possible to reduce the test application time for core-based SOCs. This reduction is illustrated using the following small example. Let us consider a core-based SOC with two cores A and B. Core A has 10 FFs and is tested using 100 test patterns while core B has 100 FFs and is tested using 10 test patterns. If modular test is not used, the scan chain in the SOC would be 110 (10 + 100) FFs long and the total number of test patterns 100 (max{10, 100}). The test application time will be equal to $(110 + 1) \times 100 + 110 = 11210$ clock cycles. If the two cores are tested one after the other using a modular approach the test application time would be the sum of the test application time of core A and core B. The test application time for core A is equal to $(10 + 1) \times 100 + 10 = 1110$ clock cycles and the test application time for core B is equal to $(100 + 1) \times 10 + 100 = 1110$ clock cycles. The total test application time is then 2220 clock cycles when modular test is used instead of 11210 clock cycles otherwise.

One of the major differences between developing an SOB and a SOC is the way testing is performed. This is illustrated in Figure 2.9 where the testing in the development process is shown for SOB in Figure 2.9 (a) and for SOC in Figure 2.9 (b). In the SOB development process, all ICs and components are fabricated and tested before they are mounted on the printed circuit board. Finally, after the mounting of components, the interconnections between the components on the board are tested. Figure 2.9 (b) shows the development and test process in the SOC methodology. In this case, it is not possible to test the cores before they are integrated in the system since the whole system is fabricated in a single step on a single die (IC). This entails that the testing has to be postponed until all cores are integrated and connected and the chip is

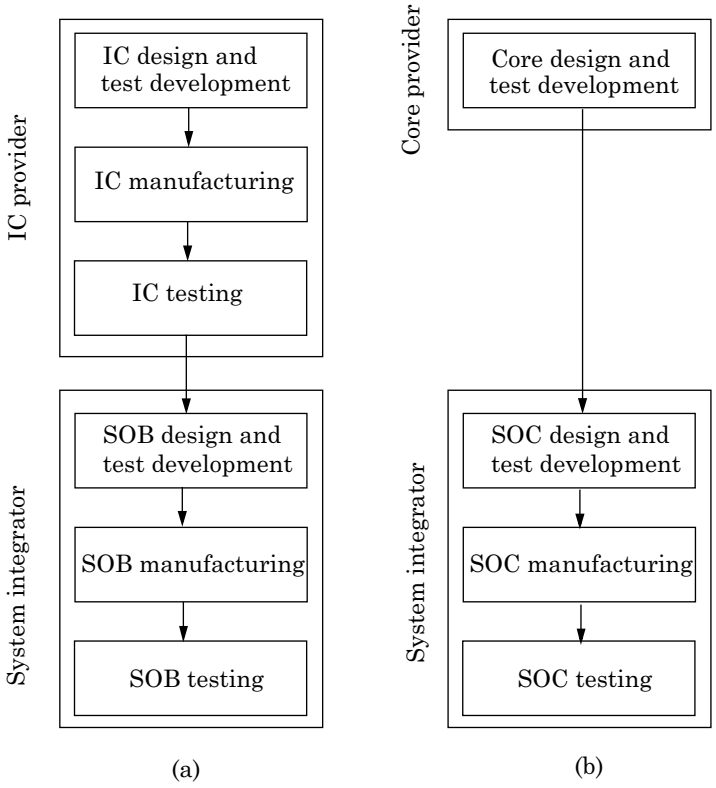


Figure 2.9: Development and test for (a) SOB and (b) SOC [Zor99].

fabricated. This means that all the test-data have to be applied at one time through a limited number of I/O pins.

2.4.1 Test-Architecture Design

A conceptual architecture, consisting of a test pattern source and sink, a TAM, and test wrapper, for modular test was introduced by Zorian *et al*. [Zor99]. The source generates/stores the test stimuli for the embedded core, and the sink stores the produced responses. The source and sink can be placed on-chip or off-chip. Test-architecture design is used to achieve core-isolation and core access required for modular test. The key components for this purpose are test wrappers and TAMs.

Cores are isolated by core test wrappers, such as specified in the IEEE Std. 1500 [DaS03], [IEEE07]. The wrapper serves three purposes: core isolation, test access, and test mode control. The IEEE Std. 1500 wrapper is illustrated in Figure 2.10. The IEEE Std. 1500 includes three registers, a wrapper boundary register (WBR), a wrapper bypass register (WBY), and a wrapper instruction register (WIR), which together provide a mechanism for core access, core isolation, and test mode control. The WBR consists of a number of input and output wrapper cells and isolates the core during test. The input wrapper cells and output wrapper cells are used to control and observe the functional inputs and functional outputs, respectively. The IEEE Std. 1500 also include one wrapper interface port (WIP) with signals used to control the WIR. By using the WIP, the core is controlled with signals such as wrapper scan input, wrapper scan output, shift enable, etc. [IEEE07]. The IEEE Std.

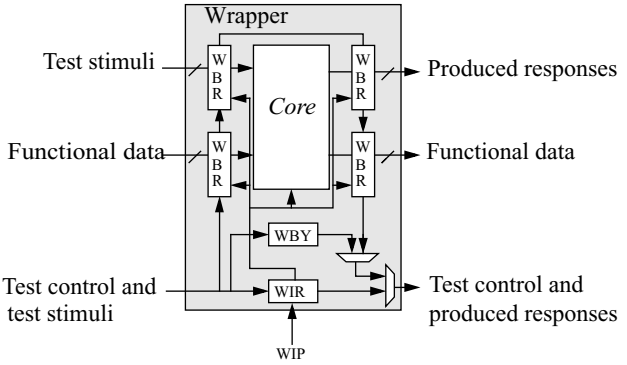


Figure 2.10: IEEE Std. 1500 wrapper.

1500 does not specify the connections of scanned elements (scan chains and wrapper cells) to the tester.

The need of a TAM, explained by Zorian *et al*. [Zor99], has its origin in the requirement to transport test stimuli from the tester to the core and of produced responses from the core to the tester. There are a number of different TAM design architectures proposed that can be used for accessing the cores during test. These TAM design architectures can be divided in two categories: (1) functional and (2) dedicated. An example of functional access is to use the functional bus as a TAM. Examples of dedicated TAMs are direct access and test bus access. Figure 2.11 shows an example of a TAM design used to access the cores in Figure 2.3. For the example in Figure 2.11, an ATE is used as test source and test sink. The test stimuli are transported from the ATE to the cores and the produced responses are transported from the cores back to the ATE.

The connections between a core and a TAM is illustrated in Figure 2.12 using a core with four scan chains of equal length, 4 FFs, 5 functional inputs, and 3 functional outputs. The core is connected to eight TAM wires and is tested using a given dedicated test T with 10 test patterns. The test stimuli are transported from the tester on the TAM wires to the core through the input test pins, *t-in*. When the test stimuli have been applied, the produced responses are transported back to the tester through the outputs, *t-out*.

The input wrapper cells, on the input side of the wrapper, will contribute to the length of the scan-in chain, while the output wrapper cells, on the output side of the wrapper, will contribute to the length of the scan-out chain. Hence, the length of the scan-in path and the scan-out path can be different. This is illustrated in Figure 2.12 where the scanned elements (scan chains, input wrapper cells, and output wrapper cells) have been formed into 4 wrapper

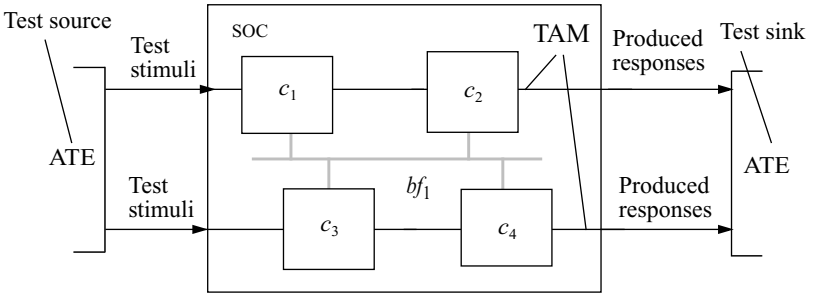


Figure 2.11: An example of a TAM design.

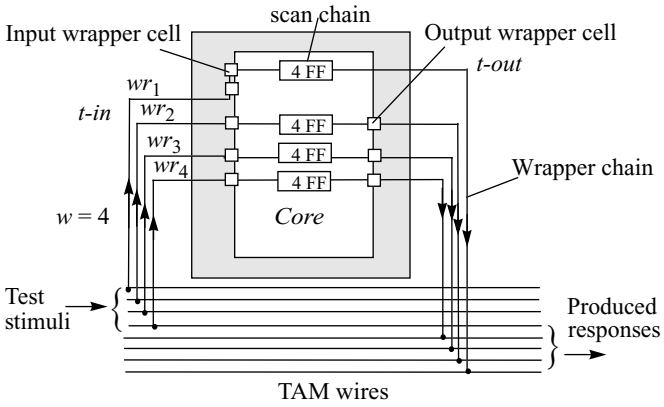


Figure 2.12: Connection of core to TAM wires using wrapper chains.

chains (we denote that as $w = 4$). For the wrapper design in Figure 2.12, 6 clock cycles are needed to shift in the test stimuli and 5 clock cycles are needed to shift out the produced responses.

The test stimuli are organized as illustrated in Figure 2.13 using one test stimuli pattern *ts*. The organization of the initial test stimuli (with don't care marked as *x*) in scan chains and inputs are illustrated in Figure 2.13(a). After designing the wrapper chains, the test-data bits are reorganized and minimum transition fill is used to balance the wrapper chains by adding extra bits, so-called idle bits, as illustrated in Figure 2.13(b). Figure 2.13(c) shows the test stimuli when applied to the four wrapper chains.

2.4.2 Test Scheduling

Test scheduling means that the start time of each test is determined in order to minimize some predefined cost function. By exploring different start times for each test it is possible to minimize the cost function while ensuring that constraints, such as hardware overhead and/or memory requirements, are not violated.

In general, tests can be applied sequentially or concurrently. In sequential test, the start time of each test is determined such that only one test is applied at a time. In concurrent test, the start time of each test can be determined such that several tests are applied at a time.

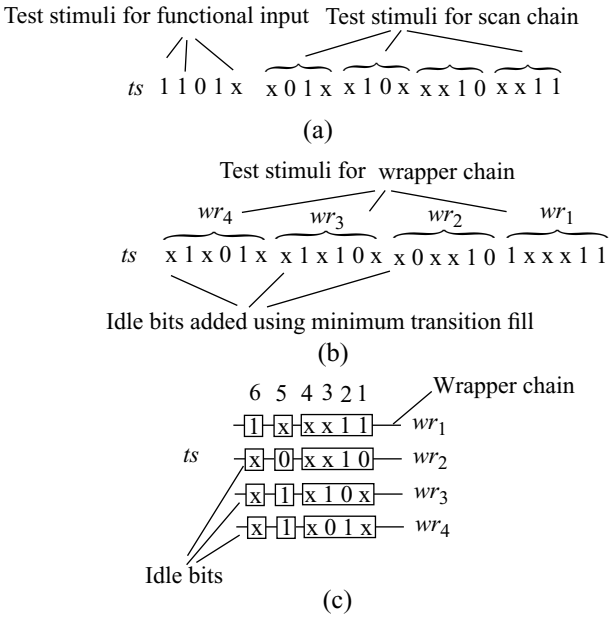


Figure 2.13: Test-data organization (a) in initially given test pattern, (b) after wrapper design and minimum transition fill, and (c) when applied to the core in Figure 2.12.

Let us illustrate sequential and concurrent testing using the four cores, c_1, c_2, c_3, and c_4, in Figure 2.3. It is assumed that the cores are tested by the given dedicated tests T_1, T_2, T_3, and T_4 in Figure 2.14, where core c_1 is tested by test T_1, core c_2 is tested by test T_2, and so forth. As illustrated in Figure 2.14, each test is associated with a test application time and a TAM width. Figure 2.15 shows an example where T_1, T_2, T_3, and T_4 are scheduled such that the test application time is minimized without violating a TAM width constraint. Figure 2.15(a) shows a sequential test schedule and Figure 2.15(b) shows a concurrent test schedule.

2.4.3 Test-Data Compression

Test-data compression has recently emerged as an efficient technique to reduce test-data volume and test application time [Tou06]. For test-data compression, the regularities and the high number of don't-care bits are explored to lower the tester memory requirement.

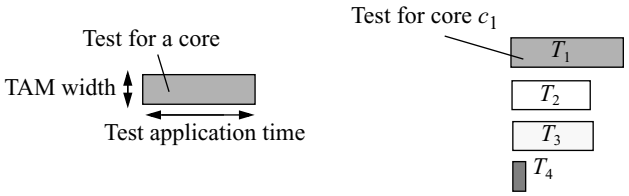


Figure 2.14: Given dedicated tests for the cores in Figure 2.3.

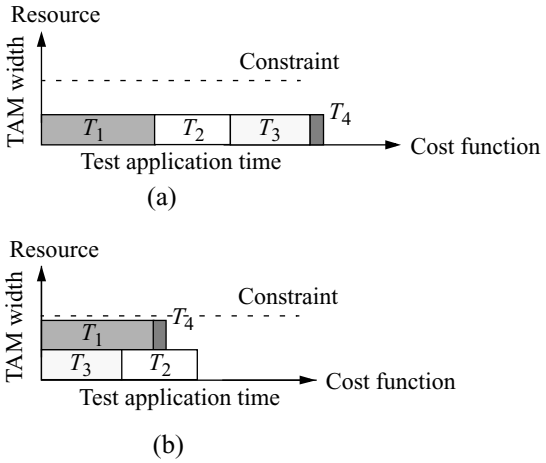


Figure 2.15: Test application time minimization at TAM width constraint using (a) sequential and (b) concurrent test scheduling.

It has been apparent in recent years, that a high number of unspecified bits, so-called don't-care bits (x), is present in the test-data. Such a don't-care bit is a bit that can be mapped to either a logical 1 or a logical 0 without affecting the quality of the test. Don't-care bits occur in the test-data partly as a consequence of the recent year's development with increasing clock frequencies that has led to IC designs with a short combinational logical depth [Wang05]. The don't-care bit density has been reported to be as high as 95%–99% [Hir03].

The general scheme is that compressed test stimuli are stored in the tester memory and, at test application, the code words are sent to the system under test, decompressed and applied. An example using an ATE as tester is

illustrated in Figure 2.16. The decoder decompresses the compressed stimuli applied from the ATE to the device under test. In Figure 2.16 the decompression is performed by expanding the n ATE channels to m scan chains, where $m >> n$.

2.4.4 Test Sharing

For test sharing, the regularities and the high number of don't-care bits are explored to lower the tester memory requirement by finding overlapping tests that have a smaller test-data volume than that of the un-shared tests. Test sharing also reduces the test application time and the TAM wire usage if the shared test is transported in a broadcasted manner.

The sharing problem is formulated as follows: for a given number of test patterns (test stimuli and expected responses), find overlapping test patterns that are used to generate a new test such that the size of the new test is minimal. An overlapping between two test patterns is found iff for each position in the sequences both tests have the same value (0, 1, x) or one is a don't-care (x).

How two test patterns can be overlapped and shared is illustrated in Figure 2.17 using test stimuli patterns ts_1 and ts_2 from two different tests. A

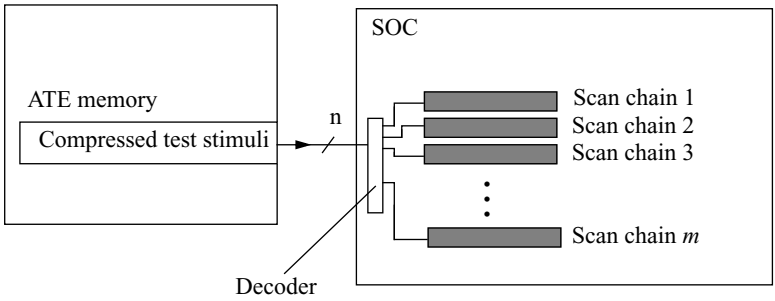


Figure 2.16: Test-architecture and ATE memory organization with stimuli compression and response compaction.

ts_1 *0xxxxxx1xx11xxxx*
ts_2 *xx0xxxxxxx1xxxx*
↓*share*
ts_new *0x0xxxx1xx11xxxx*

Figure 2.17: Sharing example.

new shared test stimulus pattern *ts_new* is generated. For this example the test-data volume to store in the tester memory is reduced by 50%. Beside the test-data volume, the test application time can also be reduced by using sharing if the cores that share the test are connected such that the shared test can be applied to the cores in parallel.

2.5 Optimization Techniques

Optimization techniques are required to solve complex combinatorial problems, such as the SOC test planning problem. In this section, two optimization techniques, CLP [Jaf87] and Tabu search [Glo89], [Glo90], are presented.

Common for all optimization techniques is that the search space, consisting of all possible solutions that can be considered during the search, is explored in the search for a solution with the lowest cost. An example of the cost variation for different solutions is illustrated in Figure 2.18. The solution with the lowest cost is called the global optimum. For combinatorial problems, such as the SOC test planning problem addressed in this thesis, there usually exists a number of local optima in the search space, as illustrated in Figure 2.18.

Optimization techniques can be either exact or non-exact. An exact optimization technique will find the optimal solution, while a non-exact optimization technique (heuristic) only searches a part of the solution space and does not guarantee that the optimal solution is found. Instead, the goal is to produce a solution that is as close to the optimal solution as possible using a limited computational effort.

Heuristics are often built on a strategy of local search, where an initial feasible solution is iteratively improved by applying local modifications, so-

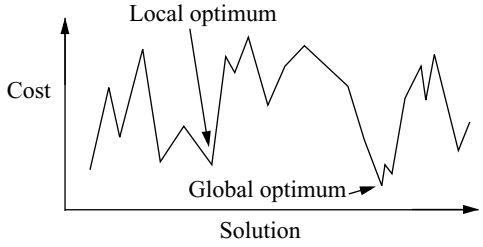


Figure 2.18: Global and local optimum for a minimization problem.

called moves, which slightly change the solution. The neighbourhood structure is a subset of the search space, which contains those solutions that can be obtained by applying a single local move. The search is terminated if no further improvements can be made. Often, the local search produces a solution which is a local minimum, that can be far from the global optimum, as illustrated in Figure 2.18. One of the main challenges when implementing a heuristic is to provide the ability to avoid to be trapped in such local minima.

Examples of exact optimization techniques are exhaustive search, branch and bound, and CLP methods. There exists a vast variety of different optimization heuristics and many of them are developed to solve problem specific optimization only. However, some are known to be applicable to a broad range of combinatorial problems. To this category belong heuristics such as Simulated annealing [Kir83], Tabu search [Glo89], [Glo90], and Genetic algorithms [Mic96].

2.5.1 Constraint Logic Programming

CLP [Jaf87] is an exact optimization technique. It is a combination of logic programming and constraint solving. CLP is a declarative method where the programmer describes the program in terms of constraints, conditions, and relations, and leaves the order of execution and assignment of variables to a solver.

To further explain the CLP technique, let us consider the following small example (from [Mar98b]). In the problem, named *SEND MORE MONEY*, each letter represents a digit, and the problem is solved by assigning integer values, in the range between 0 and 9, to the variables *S, E, N, D, M, O, R,* and *Y,* where $S \neq 0$ and $M \neq 0$, such that the following equation holds:

$$SEND + MORE = MONEY$$

The mapping of values to variables has to be one-to-one, which means that each variable has to be assigned to a value not used by any other variable. A word can be modelled as a sum of different variables, e.g. $S \times 1000 + E \times 100 + N \times 10 + D$ represents the word *SEND*. The problem can be modelled as illustrated in Figure 2.19. The program will determine that:

$$S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, and Y = 2,$$

which is the first solution for this problem, found by the solver. The example in Figure 2.19, can be extended into an optimization problem, for instance, by searching for the minimum sum of the variables.

The CLP methodology consists of three separate steps. The first is to determine a model of the problem in terms of domain variables. This is the most important step where the problem is described using a set of domain variables and the values that these variables can have, for instance, start time, duration, and resource usage. In the second step, constraints over the domain variables are defined, such as resource constraints and/or maximum hardware cost allowed. In the third, and final step, a definition of the search for a feasible solution is given. This is usually done by using a built in predicate, such as *labeling* in Figure 2.19.

During the execution of the CLP program, the solver will search for a solution by enumerating all the variables defined in step one without violating the constraints defined in step two. If required, CLP can search for an optimal solution using a branch and bound search to reduce the search space. That is, when a solution is found, satisfying all the constraints, a new constraint is added indicating that the optimal cost must be less than the cost of the current solution. If no other solution is found, the current solution has the optimal cost and is returned.

2.5.2 Tabu Search

Glover proposed, in [Glo89] and [Glo90], an approach, called Tabu search, that aims at overcoming the problem with local optima. The main idea is to avoid local optima by accepting non-improving moves. Tabu search uses three basic mechanisms in the search for the global optimum: (1) a tabu-list, (2) intensification, and (3) diversification.

```
1   smm(S,E,N,D,M,O,R,Y):-
2       [S,E,N,D,M,O,R,Y] :: [0..9],
3       constrain([S,E,N,D,M,O,R,Y]),
4       labeling([S,E,N,D,M,O,R,Y]).
5
6   constrain([S,E,N,D,M,O,R,Y]):-
7       S =/ =0,
8       M =/= 0,
9       alldifferent_neq([S,E,N,D,M,O,R,Y]),
10      1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E =
11      10000*M + 1000*O + 100*N +10*E + Y.
```

Figure 2.19: A CLP example [Jaf87].

The cyclic behaviour, that occurs when a previously visited solution is revisited, is avoided by using a short term memory called tabu-list. This memory holds a record of the recently visited solutions, which should be avoided in the next moves. The tabu tenure is a measure on how long a move should be marked as tabu. The use of tabus is effective in preventing cycling. However, it may also prohibit attractive moves and lead to a slow and time consuming search.

With Tabu search an initial solution (e.g., randomly generated) is first generated. The heuristic then moves repeatedly to a neighbouring solution. At each step, a subset of the neighbouring solutions is evaluated and the move that reduces the cost the most is selected. If there are no improving moves, the least degrading move is selected, which means that an uphill move is performed. When a move has been performed it is stored in a tabu-list of length h. The tabu-list keeps information of the h most recently visited solutions preventing the algorithm of applying them. However, it might be advantageous to return to a previous visited solution within the following h iterations. Therefore, an aspiration criterion is often introduced to permit the tabu status to be cancelled. Such an aspiration criterion is, e.g., that the move would generate a solution better than the best solution found so far. The process is stopped when a specific termination condition is satisfied, such as, a solution with an initially given cost is found or that a number of iterations has been performed.

Tabu search is often implemented using two loops. The inner loop which is called intensification and the outer loop, which is called diversification. The aim of the inner loop is to intensify the search by performing small moves (changes) to a current solution and to guide the search to a specific region where it is likely that a local (or global) optimum is located. An example of intensification strategy is to keep those solution components (e.g., assignment of cores to TAMs) that frequently occur in low-cost solutions. The aim of diversification is to force the search into a new, previously un-explored, part of the search space. Diversification can, e.g., be performed by randomly generating a new solution.

Chapter 3
Related Work

THIS CHAPTER DESCRIBES previous work that is either used in, or directly related to, this thesis. First, related work on test-architecture design, test scheduling, test-data compression, and test sharing and broadcasting, is described. Second, co-optimization techniques, including test-architecture design and test scheduling, test-architecture design and test scheduling with test-data compression, and test-architecture design and test scheduling with test-data compression and test sharing, are described. Finally, the related work is summarized.

3.1 Test-Architecture Design

In this section the related work on test-architecture design, including wrapper design and TAM design, is presented.

3.1.1 Wrapper Design

Wrapper design addresses the problem of core isolation, test access, and test mode control. Wrapper design can be divided in two parts: wrapper architecture selection and wrapper design optimization.

Marinissen *et al*. [Mar98a] proposed a wrapper architecture called TestShell and Varma and Bathia [Var98] proposed a wrapper architecture called Test Collar. The TestShell and Test Collar form the basis of the

standardized wrapper architecture IEEE 1500 [DaS03], [IEEE07], mentioned in Section 2.4. As the Test Collar and TestShell are similar, only the TestShell will be described in detail. A conceptual view of the TestShell is illustrated in Figure 3.1. The TestShell wrapper architecture has one multiplexer per functional input and one multiplexer per functional output. The multiplexer at the functional input is used to control the application of test stimuli and functional data. The multiplexer at the functional output is used to control the application of test stimuli for interconnect test, the produced responses, and the functional data.

The TestShell wrapper architecture supports four modes: (1) functional mode, (2) test mode, (3) interconnect test mode, and (4) bypass mode. The functional mode is used when the core is in normal (functional) operation and the test mode is used when the core itself is under test. The interconnect test mode refers to the test of the logic between cores and finally, the bypass is used when test stimuli and produced responses are transported to other cores through the TestShell wrapper.

Wrapper design optimization is to group the scanable elements (scan chains, input wrapper cells, and output wrapper cells) such that they can be connected to the TAM in the best possible way. The test application time $\tau_i(w)$ for a test T_i used to test a core i with w wrapper chains is as follows [Mar00]:

$$\tau_i(w) = (1 + \max\{si, so\}) \times l + \min\{si, so\}, \tag{3.1}$$

where l is the number of test patterns, si and so are the length of the longest wrapper scan-in and scan-out chain among the w wrapper chains. As given by Equation 3.1, there is a relationship between the test application time and the

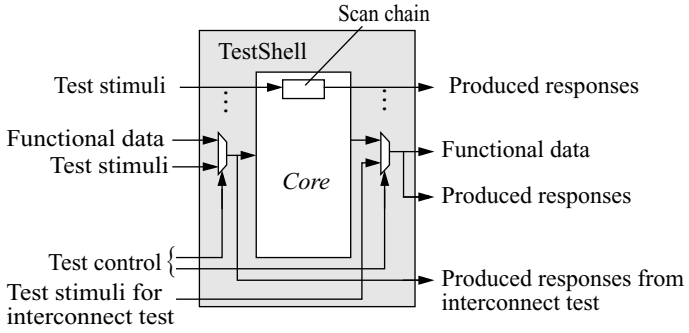


Figure 3.1: TestShell (adopted from [Mar98a]).

length of the longest wrapper scan-in and scan-out path, max{*si, so*}. The aim of the wrapper design optimization is, therefore, usually to minimize the length of the longest wrapper chain scan-in and scan-out path, max{*si, so*}.

The wrapper design optimization was addressed by Marinissen *et al*. [Mar00] and by Iyengar *et al*. [Iye01a]. The *Design_wrapper* algorithm proposed by Iyengar Iyengar *et al*. [Iye01a] is presented in Figure 3.2. The input to the *Design_wrapper* algorithm is a core c_i and a number of wrapper chains *w,* and the output is an optimized wrapper design and a test application time.

Since this algorithm is used in several places throughout the thesis, it will be described here in more detail. The *Design_wrapper* algorithm consists of three parts. In Part one (line 1–11 in Figure 3.2), the scan chains are grouped in wrapper chains such that the length of the longest wrapper chain is minimal. First, the sc_i scan chains are sorted descending according to their length ff_{ij} (line 5). The length of the longest wrapper chain S_{max} and the shortest wrapper chain S_{min} are then located (line 7–8). Each scan chain *j* is then assigned to the wrapper chain *S* whose length, after this assignment, is closest to but not exceeding the length of the current longest wrapper chain (line 9). If no such wrapper chain can be found, the scan chain *j* is assigned to the wrapper chain with the shortest length (line 11).

In Part two (line 12 –13) and Part three (line 14 –15), the input wrapper cells and output wrapper cells are assigned to the wrapper chains created in Part one. Since the length of each input wrapper cell and output wrapper cell is one, these are added to the shortest wrapper chain.

```
1  Procedure Design_wrapper
2     // Input: One core ci, number of wrapper chains w
3     // Output: A wrapper design, test application time
4     // Part one
5     Sort the sci scan chains in descending order of length
6     For each scan chain j
7        Find wrapper chain Smax with current maximum length (max{si, so})
8        Find wrapper chain Smin with current minimum length (max{si, so})
9        Assign scan chain j to wrapper chain S such that {Length(Smax) -
(Length(S) + ffij )} is minimum
10       If there is no such wrapper chain S
11          Assign scan chain j to Smin
12    // Part two
13    Assign input wrapper cells to the wrapper chains created in Part one
14    // Part three
15    Assign output wrapper cellsto the wrapper chains created in Part one
```

Figure 3.2: Design_wrapper algorithm (adopted from [Iye01a]).

The example core c_1 from the SOC in Figure 2.3 will be used for the illustration of the wrapper design optimization to minimize the test application time. Core c_1 has four scan chains *a* to *d*, as illustrated in Figure 3.3. The length of scan chain is 3 FFs for *a*, 4 FFs for *b*, 5 FFs for *c*, and 4 FFs for *d*.

First, the scan chains are sorted in descending order, according to their length. The result from this step is illustrated in Figure 3.4. The process of grouping scan chains in 2 wrapper chains ($w = 2$) is illustrated in Figure 3.5. In each iteration, one scan chain (or input/output wrapper cell) is assigned to a wrapper chain such that the length of the longest wrapper chain is minimized, hence, minimizing the term $\max\{si, so\}$ in Equation 3.1. In the example, four iterations are used, one for each scan chain, and the final result is a wrapper design where scan chains *a* and *c* are assigned to wrapper chain wr_1 and scan chains *b* and *d* are assigned to wrapper chain wr_2. For each iteration, the term $\max\{si, so\}$is presented. The final wrapper design is returned after the fourth iteration. The test application time for test T_1 using two wrapper chains is $(8 + 1) \times 3 + 8 = 35$ clock cycles.

The trade-off between test application time and the required number of wrapper chains for a core is illustrated in Figure 3.6 using core c_1 in Figure 3.3, which is tested by test T_1 in Figure 2.14. In Figure 3.6(a), the wrapper at c_1 is optimized for 3 wrapper chains ($w = 3$). The test application time $\tau_1(3)$ for applying T_1 with 3 test patterns ($l = 3$) is $\tau_1(3) = (1 + 7) \times 3 + 7 = 33$ clock cycles. In Figure 3.6(b), the wrapper at c_1 is optimized for 2 wrapper chains ($w = 2$). The test application time $\tau_1(2)$

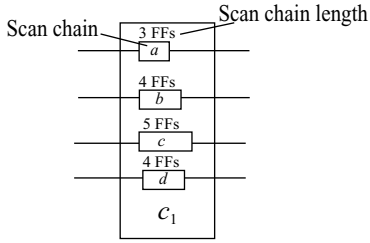


Figure 3.3: Example core c_1 with four scan chains a, b, c, and d.

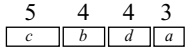


Figure 3.4: Scan chains *a, b, c,* and *d* sorted according to their length.

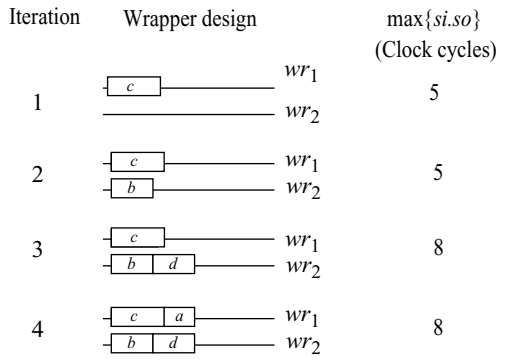


Figure 3.5: Grouping of scan chains in wrapper chains using the design algorithm in [Iye01a].

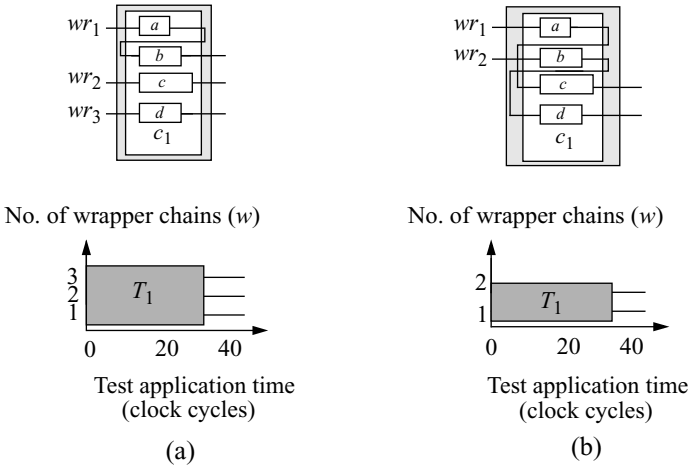


Figure 3.6: Test-architecture and test schedule (a) for a core with four wrapper chains and (b) for a core with three wrapper chains.

for applying T_1 is $\tau_1(2) = (1+8)\times 3 + 8 = 35$ clock cycles. Hence, 2 clock cycles can be saved if 3 wrapper chains are used instead of 2.

The test application time for core s38417 from the ISCAS'89 bencmark set [Brg89] using the *Design_wrapper* algorithm at various number of wrapper

chains is presented in Figure 3.7. As can be seen from these results, the test application time for a given core at various number of wrapper chains behaves as a staircase function with a number of pareto-optimal points. (The pareto-optimal points are the ones on the left most edges at each staircase level.) Hence, the test application time can be equal for different number of wrapper chains. Several pareto-optimal points are illustrated in Figure 3.7, e.g., when the number of wrapper chains is 18 and when the number of wrapper chains is 32. This staircase behaviour makes it difficult to assign the best number of wrapper chains to a core. Simply increasing the number of wrapper chains, does not always lead to a decreased test application time. For example, the test application time for 31 wrapper chains is the same as when only 18 wrapper chains are used.

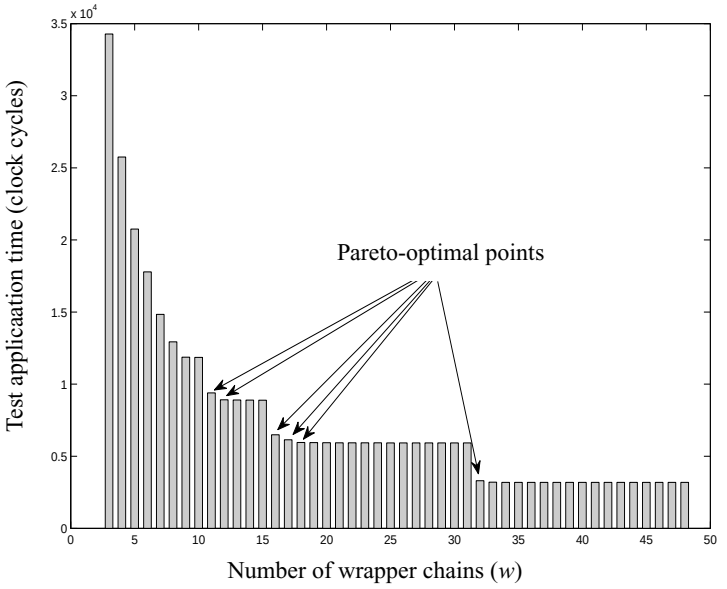


Figure 3.7: Test application time for core s38417 at various number of wrapper chains.

3.1.2 Test Access Mechanism Design

The TAM is used to provide the core-based SOC with an architecture for transporting test stimuli from the tester to the wrapped cores and transporting produced responses from the wrapped cores to the tester.

Several TAM architectures have been proposed [Aer98], [Har99], [Imm90], [Iye02b], [Mar98a], [Var98]. These TAM design architectures can be divided in two categories: (1) functional and (2) dedicated. An example of functional access is the Functional bus access [Har99]. Examples of dedicated TAM architectures are:

- Direct access [Imm90],
- Multiplexing [Aer98],
- Daisychain [Aer98],
- Distributed [Aer98],
- Test bus [Mar98a],
- TestRail [Mar98a], and
- Flexible-width architecture [Iye02b].

The example SOC in Figure 2.3 will be used for the illustration of the different TAM architectures. The Direct acccess scheme is illustrated in Figure 3.8, the Multiplexing, Daisychain, and Distributed architectures are illustrated in Figure 3.9. The Test bus and TestRail are illustrated in Figure 3.10 and the Flexible-width architecture is illustrated in Figure 3.11. The Functional bus access is illustrated in Figure 3.12

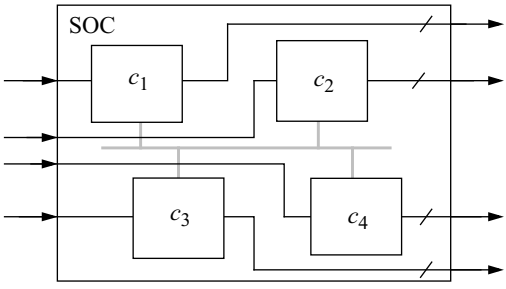


Figure 3.8: Direct access TAM architecture.

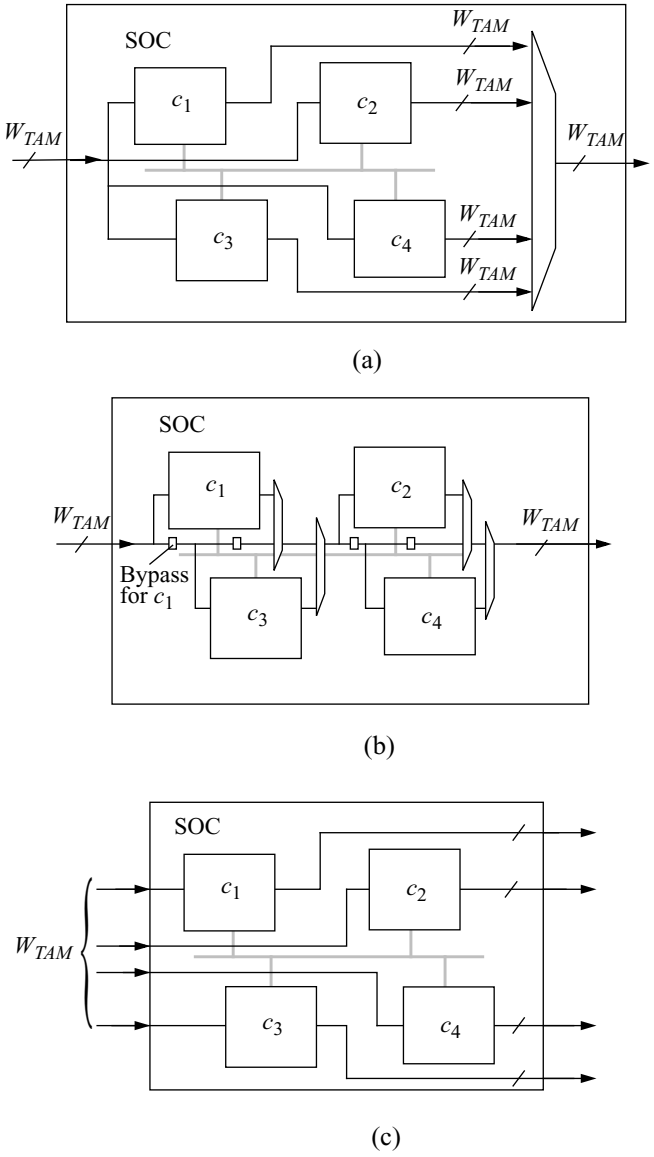


Figure 3.9: Three TAM architectures (adopted from [Aer98]): (a) Multiplexing, (b) Daisychain , and (c) Distributed.

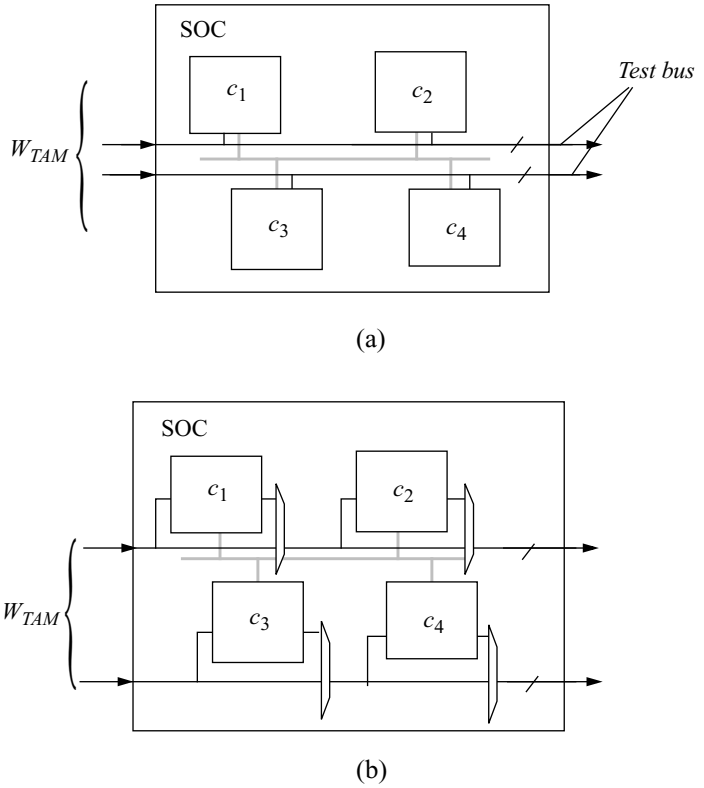


Figure 3.10: Two TAM architectures: (a)Test bus and (b) TestRail.

Immaneni and Raman [Imm90] proposed a Direct access test scheme (Figure 3.8) for core-based ASIC designs. Each core is accessed directly from the SOC I/O pins, hence, solves both the wrapper and TAM design problems. In Direct access, the number of TAM wires, W_{TAM}, is equal to the total number of core terminals in the SOC, therefore, Direct access requires a large wiring overhead when the total number of core terminals is large. In addition, for large SOCs, the number of core terminals vastly exceeds the number of I/O pins, and therefore, direct access is not applicable in practice.

Aerts and Marinissen [Aer98] proposed three TAM architectures illustrated in Figure 3.9. They are: (1) Multiplexing, (2) Daisychain, and (3) Distributed architectures. The Multiplexing architecture (Figure 3.9(a)) contains only one

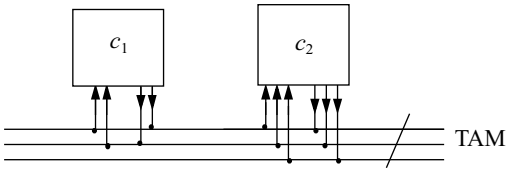


Figure 3.11: Flexible-width architecture.

TAM that connects all cores in the SOC and only one core can be accessed at a time, hence, the cores are tested sequentially. The overall test application time of the system is, therefore, the sum of all the individual core's test application times. One drawback of this architecture is that it cannot test the interconnections between cores since only one core can be accessed at a time.

The Daisychain architecture (Figure 3.9(b)) also uses one TAM to connect all cores, however, in contrast to the Multiplexing architecture, the Daisychain architecture allows multiple cores to be accessed at a time. The wrapper chains of all cores are connected into long chains, from the inputs, through all cores, to the outputs. Each core has a bypass structure to shorten the access path for each individual core. The test application typically starts by testing all cores simultaneously. The bypass structure is used when the test of one core has completed. Finally, the only core left without being bypassed is the one with the highest number of test patterns.

In the Distributed architecture (Figure 3.9(c)) each core has its own dedicated TAM, and all cores are tested in parallel. The sum of each TAM's width is the full TAM width of the system. The overall test application time for the system is given by the core with the longest test application time.

These three dedicated TAM architectures solve the TAM problem. However, they do not provide the ability of test application time minimization using elaborate test scheduling. The tests are scheduled, either all at the same time, as for the Distributed architecture, or one at a time, as for the Multiplexing architecture. Therefore, TAM architectures that support more flexible scheduling alternatives have been proposed: (1) the Test bus, (2) the TestRail, and (3) the Flexible-width architecture.

Varma and Bathia [Var98] proposed the Test bus, illustrated in Figure 3.10(a). The Test bus can be seen as a combination of the Multiplexing and Distributed architectures. Marinissen *et al*. [Mar98a] proposed the TestRail architecture. The TestRail architecture is illustrated in Figure 3.10(b)

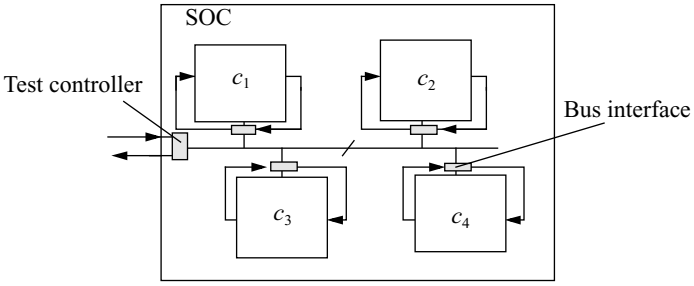


Figure 3.12: Functional bus access TAM architecture.

and can be seen as a combination of the Daisychain and the Distributed architectures. The Test bus and TestRail architectures support more flexible sceduling alternatives compared to the Distributed and the Multiplexing architectures. However, in the Test bus and TestRail architectures, the cores assigned to a TAM are connected to all wires of that TAM.

Iyengar *et al.* [Iye02b] proposed a Flexible-width architecture that allow cores to be connected in a flexible way to the TAM wires, as illustrated in Figure 3.11 using c_1 and c_2 from Figure 2.3. In this way, each TAM wire is treated as a separate unit which increases the flexibility of the test schedule. The Flexible-width architecture, however, potentially leads to an irregular organization of the test-data in the tester memory, which means that additional test control may be required.

Dedicated TAMs decrease the test application time for the system but contribute to increased wiring and hardware overhead. An alternative approach is to reuse the functional connections in the SOC as TAM. The main advantage of reusing the functional connections is that no, or few, TAM wires are required. An example showing the functional bus used as TAM is illustrated in Figure 3.12. Harrod proposed in [Har99] a method where the AMBA specification, developed by ARM, was extended to include the transportation of test-data. The hardware consists of a test harness, acting as a wrapper, which is placed around each core that is tested using AMBA and a test interface controller.

Hwang and Abraham [Hwa01] proposed a technique called Reuse of Addressable System Bus for SOC Testings (RASBuS) where on-chip

microprocessors are used to test the cores in the design and the functional bus structure (RASBuS) is used for the test transportation.

None of the proposed methods that make use of the functional bus take into consideration the hardware overhead introduced by the test harness and by the added test controller.

3.2 Test Scheduling

Test scheduling means that the start time of each test is determined, and the objective is to minimize a predefined cost function. Test scheduling techniques can be divided into the following three categories:

- *non-partitioned* testing,
- *partitioned* testing with run to completion, and
- *pre-emptive* testing.

The three test scheduling techniques are illustrated in Figure 3.13 using the four tests in Figure 2.14. In the example, it is assumed that the cost function is the test application time, which will be minimized without violating the hardware constraint given by the maximum number of TAM wires. Figure 3.13(a) shows an example of a test schedule using a non-partitioned (session based) technique, used by Zorian [Zor93], and Chou *et al*. [Chou97]. In non-partitioned test scheduling no new test is allowed to start until all tests in a session are completed. This method produces long test application times due to long periods of time when no core in the system is tested, so-called idle times.

Figure 3.13(b) shows that the test schedule can be improved by using a partitioned (sessionless) technique. Chakrabarty [Cha01] and Muresan *et al.* [Mur00] have proposed partitioned scheduling techniques. In the partitioned technique, tests are allowed to be scheduled as soon as possible, which can decrease the test application time. However, a more advanced test controller is required for the invocation of tests since more possible start times of tests can be used.

In order to further optimize the schedule, a pre-emptive test scheduling technique can be used. Such a technique has been proposed by Iyengar and Chakrabarty [Iye01b], Larsson and Fujiwara [Lar02], and Larsson and Peng [Lar03b]. The pre-emptive test scheduling is illustrated in Figure 3.13(c).

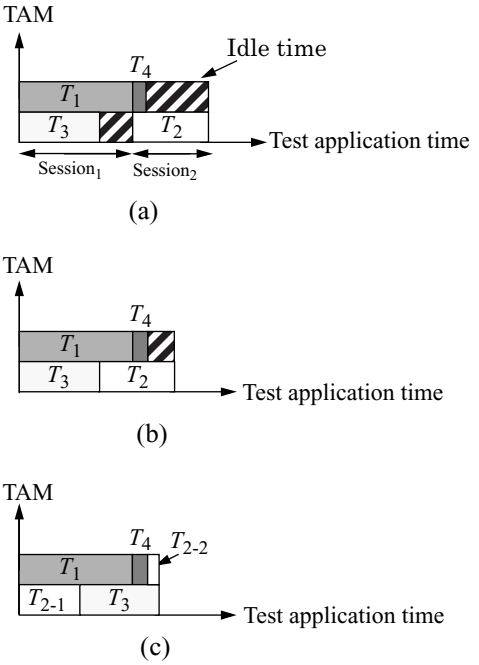


Figure 3.13: Three test scheduling techniques: (a) non-partitioned, (b) partitioned, and (c) pre-emptive.

Here, the test T_2 is pre-empted and then resumed at a later point in time using different TAM wires. Pre-emptive test scheduling can be used to reduce the idle time as illustrated in Figure 3.13(c). Pre-emptive test scheduling requires an advanced test controller. Furthermore, it is not applicable to all types of tests. For example, BIST, where the test application is started and than run to completion, is usually not possible to pre-empt.

3.3 Test-Data Compression

The aim of test-data compression is to minimize the required ATE memory. Test-data compression is usually done by exploring the regularities and the high number of don't-cares present in the test-data.

As described in Section 2.4.3, the general scheme is that compressed test stimuli are stored in the tester memory and at test application the code words are sent to the system under test where they are decompressed and applied to the cores.

Several test-data compression schemes have been investigated in literature [Jas03], [Gon04b], [Cha03a], [Raj04], [Teh05], [Wang05], [Bar01]. Jas *et al.* [Jas03] used Huffman coding, Gonciari Gonciari *et al.* [Gon04b] used Variable-length Input Huffman Coding, and Chandra and Chakrabarty [Cha03a] used Frequency-Directed Run-Length (FDR) codes. Available for test-data compression are also a number of commercialized test-data compression tools such as TestKompress from Mentor Graphics [Raj04], SmartBIST from IBM/Cadence [Koe01], and DBIST from Synopsys [Cha03c].

Usually, the decompression is associated with both hardware and ATE synchronization overhead. The hardware overhead is due to the logic required for the decoder used for the decompression of code words. The ATE synchronization overhead is due to the required communication between the on-chip decoder and the ATE. For example, it might be required to stop the application of the next code words from the ATE while the current codeword is decoded and applied. Gonciari *et al.* [Gon05] analyzed the ATE synchronization overhead and proposed an approach to reduce it. The proposed approach exploits the frequency ratio (f_{scan}/f_{ATE}). The ATE synchronization overhead is reduced by, for a given frequency ratio, inserting dummy bits in the test data and designing a distribution unit placed before the decoder.

In the rest of this section, three test-data compression techniques, which are used in this thesis, are described in detail: (1) Nine-Coded (9C) coding [Teh05], (2) Selective Encoding [Wang05], and (3) Vector Repeat [Bar01].

3.3.1 The Nine-Coded Technique

Tehranipoor *et al.* [Teh05] proposed a test-data compression technique called Nine-Coded (9C) coding. The 9C coding technique makes use of on-chip decoders for the decompression of code-words. In 9C, the test patterns are divided into K-bits blocks, where K is a constant specified by the system integrator. Each such block of K bits is then further divided in two equal halves and coded. The code words, one for each of the nine cases, are presented in Table 3.1. Column 1 lists the nine cases and Column 2 lists the

Table 3.1: 9C Coding for $K = 8$ [Teh05]

Case	Input block	Description	Decoder input	Size (bits)
1	0000 0000	All 0's	0	1
2	1111 1111	All 1's	10	2
3	0000 1111	Left half 0, right half 1	11000	5
4	1111 0000	Left half 1, right half 0	11001	5
5	1111 uuuu[a]	Left half 1, right half mismatch	11010uuuu	9
6	uuuu 1111	Left half mismatch, right half 1	11011uuuu	9
7	0000 uuuu	Left half 0, right half mismatch	11100uuuu	9
8	uuuu 0000	Left half mismatch, right half 0	11101uuuu	9
9	uuuu uuuu	All mismatch	1111uuuuuuuu	12

a. u = (0, 1, *x*)

input block (uncompressed test-data). Column 3 contains a description and the decoder input is in Column 4. The size of the compressed data for each case is listed in Column 5.

Each *K*-bits input block will be coded based on the organisation of the test-data (0's, 1's and *x*'s) in the two halves. The *x*'s in each block will be assigned 0's or 1's such that the shortest code word can be used. For example, the input block "0000 0000" and "*xxxx xxxx*" will both be coded with a single "0".

This coding scheme enables test independent coding and it can be implemented using a small decoder. An example showing the 9C coding is presented in Figure 3.14. The test stimulus TS_1 consists of 48 bits and after test-data compression using 9C 16 bits.

One disadvantage with the 9C coding is the required ATE synchronization, which is needed to stop the ATE from applying the next codeword while the current codeword is being decompressed by the on-chip decoder. Such synchronization is complex and not supported by current state-of-the-art ATEs.

3.3.2 The Selective Encoding Technique

The test-data compression scheme Selective Encoding [Wang05] makes use of on-chip decoders to expand the compressed test stimuli. The *w* input bits (TAM width) are expanded to *m* wrapper chains, as illustrated in Figure 3.15 [Wang05].

The test-data corresponding to the bits shifted into the wrapper chains in one clock cycle is called a wrapper chain slice. Each wrapper chain slice is encoded using a series of *w*-bits slice-codes. For Selective Encoding, *w* is selected as: $w = \lceil \log_2(m+1) \rceil + 2$ which means that $w << m$, hence, test-

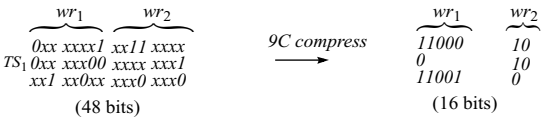


Figure 3.14: Test-data compression using 9C.

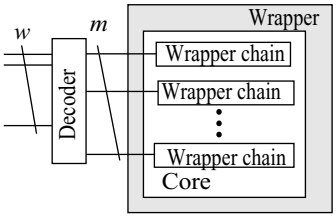


Figure 3.15: Test-data expansion for a wrapped core using Selective Encoding.

data volume and test application time are reduced. In the best case, Selective Encoding can achieve test-data compression by a factor m/w.

The coding is performed using either single-bit-mode or group-copy-mode. In single-bit-mode, each bit in a wrapper chain slice is indexed from 0 to m and the position of the target symbol is encoded using a slice-code. For example, the target symbol of 1 in the slice "*xxx*1000" is encoded as "0011" since it is positioned at index 3. In the group-copy-mode the m-bit slice is divided into $m/\lceil \log_2(m+1) \rceil$ groups. Two code words are needed to encode one group. The first code word specifies the index of the first bit in the group, and the second code word contains the test-data. Selective Encoding encodes the test-data for wrapper chain slices in every clock cycle and, therefore, the ATE synchronization problem is avoided.

3.3.3 The Vector Repeat Technique

Barnhart *et al.* [Bar01] proposed a methodology using Vector Repeat where the don't-cares are filled by repeating the last specified bit within the same scan chain, so-called repeat fill. In [Wang05], vector repeat is used in conjunction with a the Selective Encoding test-data compression technique, described in Section 3.3.2.

The idea of Vector Repeat is the following. When two or more adjacent vectors (wrapper chain slices) are identical, only one vector needs to be stored in the ATE memory and a repetition counter will record the number of time the specific vector should to be repeated. During test application, the ATE uses the repetition counter to restore the compressed test-data before it is shifted to the core. There is no need for on-chip decoder logic as the decoding is embedded in an ATE test program. As Vector Repeat does not expand the test stimuli, it is only able to achieve compression in the space domain and not in the time domain.

An example showing the Vector Repeat coding is presented in Figure 3.16. The test-data is first organized such that each row consists of the test-data for one wrapper chain. Minimum transition fill is used to make each wrapper length of equal length. Each don't care bit is assigned to a value (0 or 1) such that a maximum number of repeating vectors is acheived. With the Vector Repeat mechanism only 8 bits, out of the 48 bits, need to be stored in the ATE memory. Information about how many times each coded vector should be repeated must also be stored. For the example in Figure 3.16 16 (4×4) bits are required for the repetition counter. In total 24 (8 + 16) bits need to be stored in the ATE memory. At test application the first coded vector, "11", is repeated 7 times, the second vector, "01", is repeated 9 "00", is repeated 5 times, and the fourth vector, "10", is repeated 3 times.

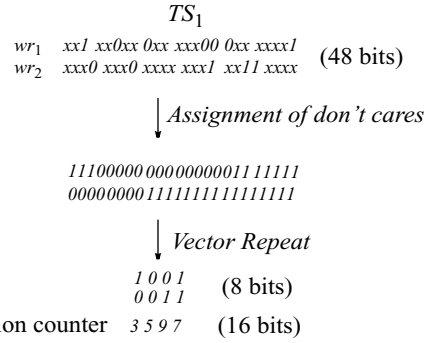


Figure 3.16: Test-data compression using Vector Repeat.

3.4 Test Sharing and Broadcasting

Regularities and the high number of don't-care bits in the test-data can be used to find overlapping test patterns from different tests. The overlapping test patterns are used to generate a new shared test, which can be shared and broadcasted to multiple cores. The aim of test-sharing is to generate a new shared test with a minimal test-data volume.

Jiang *et al.* [Jia03] proposed a method for generating common tests for multiple cores using the tests delivered by the core providers and an enhanced logic simulator. The test stimuli, intended for one particular core, are broadcasted to all cores in the system, testing them in parallel and the produced responses from each core are compacted using MISRs. The fault coverage for the system is evaluated using an enhanced fault simulator and a test-data generator is used as a complement to the tests delivered by the core providers in order to increase the fault coverage for the system. The fault simulation and test-data generation used in the proposed method are usually too time-consuming to be included in the test-architecture design and test scheduling optimization.

Lee *et al.* [Lee99] proposed a technique where all circuits in a design are considered as a single "virtual circuit". Test-data for the virtual circuit is generated and broadcasted. As in the method proposed by Jiang *et al.* [Jia03], the produced responses are compacted using MISRs.

Shinogi *et al.* [Shi05] proposed a method where test pattern overlapping is used to generate a test that is shared by all cores in the SOC. The don't-cares in the test-data are explored in the search for overlapping test patterns and a test controller used for the test stimuli application is proposed. The test controller is required since the scan chains of the cores that share a test may vary in length. The core with the shortest scan chain length must therefore wait for the core with the longest scan chain length. In the method proposed by Shinogi *et al.* [Shi05], all cores in the SOC share a test and are, by the use of broadcasting, tested in parallel. Therefore, the proposed method limits the assignments of cores to TAMs and the test scheduling.

3.5 Test-Architecture Design and Test Scheduling

The aim of co-optimizing the test-architecture design and test scheduling is to reduce test application time and/or TAM width requirement.

As illustrated in Figure 3.7, the test application time for a core behaves as a staircase when the number of wrapper chains (TAM wires) increases; therefore, it is difficult to assign the best number of wrapper chains to each and every core of a SOC. By co-optimizing the test-architecture design and test schedule the cost function (usually the test application time or TAM width) can be decreased compared to when test-architecture design and test scheduling are solved separately.

For the test-architecture design and test scheduling several trade-offs may be explored, e.g., the trade-off between the test application time and the required number of TAM wires. Let us consider an example where the four cores, c_1, c_2, c_3, and c_4, in the SOC in Figure 2.3, are tested using the given dedicated tests in Figure 2.14. Further, we assume a TAM width constraint of 8 TAM wires. Figure 3.17 shows an example where the test-architecture design and test scheduling is solved individually. The test-architecture illustrated in Figure 3.17(a) consists of one Test bus tb_1 with 8 TAM wires. All four cores are assigned to the same TAM. The wrapper design is solved for each core such that each core has 4 wrapper chains. A sequential test schedule is presented in Figure 3.17(b).

By using co-optimization various test-architecture alternatives are explored, e.g., varying the number of TAMs and the width of each TAM. For each test-architecture alternative, several test schedule alternatives can be generated and evaluated. An example of a co-optimized test-architecture and test schedule is illustrated in Figure 3.18. In Figure 3.18(a) the 8 TAM wires have been partitioned in two Test buses, tb_1 and tb_2, each with 4 TAM wires. Further, core c_1 and c_4 are assigned to tb_1 and core c_2 and c_3 are assigned to tb_2. The optimized test schedule, which is illustrated in Figure 3.18(b), has a shorter test application time as compared to the test schedule in Figure 3.17(b).

Several test-architecture design and test scheduling techniques have been proposed [Goel03], [Iye03], [Lar01], [Seh04], [Xu04], [Hus06]. Goel and Marinissen [Goel03] proposed a test-architecture design and test scheduling algorithm, named TR-Architect, that minimizes the test application time and the tester memory requirement. TR-Architect, works for the Test bus TAM architecture as well as the TestRail TAM architecture. A test-architecture independent lower bound on the test application time for a system with a given TAM width is also presented.

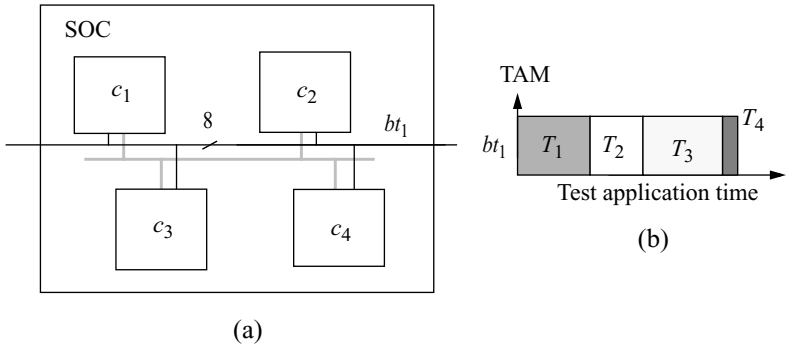


Figure 3.17: Example of a (a) test-architecture and (b) test schedule without co-optimization.

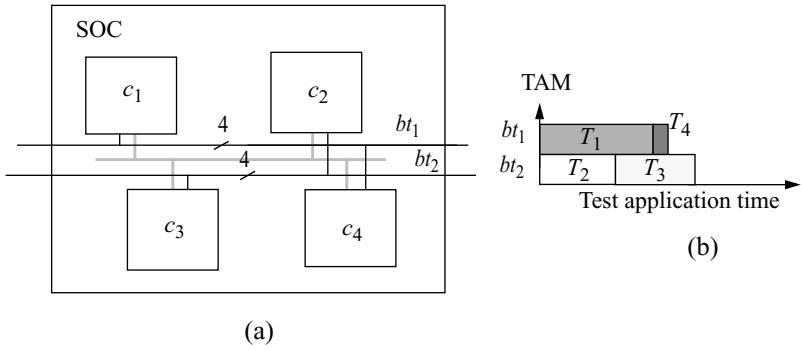


Figure 3.18: Example of a co-optimized (a) test-architecture and (b) test schedule.

Iyengar *et al*. [Iye03] proposed a technique with a TAM consisting of a flexible width Test bus that can fork and merge between cores. This means that different cores that are connected to the same TAM can at test application time utilize a different number of TAM wires. The pre-emptive test scheduling and TAM design are tightly integrated to minimize the test application time while considering test resource conflicts, preceedence, and power consumption constraints. In addition, the relation between TAM width and tester test-data volume is explored.

Larsson and Peng [Lar01] proposed an integrated SOC test framework where the test application time and the cost of TAMs are minimized, while considering constraints on tests and test resources. They included test selection, TAM design, and floor planning of test resources in the framework as well as a system test algorithm.

Sehgal *et al*. [Seh04] proposed a SOC test planning technique, including wrapper design, TAM design, and test scheduling. A Test bus TAM architecture is used. The test application time is reduced by matching the high-speed ATE channels to slower scan chains ($f_{ATE} \geq f_{scan}$) using the concept of virtual TAMs. A virtual TAM wire is an on-chip TAM wire that does not directly correspond to a particular ATE channel. This means that the number of virtual TAM wires can exceed the ATE pin count.

Xu and Nicolici [Xu04] proposed a SOC test planning technique (including test-architecture design and test scheduling) using multi-frequency virtual TAMs. It is shown that bandwidth matching can be used for the Test bus and TestRail TAM architectures under either TAM width constraints ($f_{ATE} \geq f_{scan}$) or power constraints ($f_{ATE} \leq f_{scan}$).

Hussin *et al.* [Hus06] solved the test scheduling problem, minimizing the test application time under a test power constraint, using the functional bus structure.

The related work, described in this section, focuses mainly on the reduction of the test application time while the problem with high test-data volumes is not considered directly.

3.6 Test-Architecture Design with Compression

As described in Section 2.3, the produced responses can be compacted using MISRs. The general draw-back with test response compactors is the sensitivity to unspecified values, so-called unknowns. These unknown values, which are becoming more common with technology scaling, can occur due to the use of tri-state buffers and uncontrolled and/or uninitialized memory elements [Sin03]. Additional logic must therefore be added for tolerating unknowns [Mit05]; however only a limited number of unknown bits can be handled. MISRs may also suffer from the problem with aliasing, although the probability is small. Aliasing is due to the compaction of the responses, which may cause faults to pass undetected. Further, by using a MISR the testing cannot be terminated immediately when a fault is present (abort-on-fail

testing). Instead, the testing must continue until the final signature is produced. To address these problems, an architecture that does not make use of test response compactors has been proposed [Lar07c].

Larsson and Persson [Lar07c] proposed a test-architecture for combined test-data compression and abort-on-fail test. A mask is introduced such that the don't care bits in the expected responses can be filled arbitrary. For each bit in the expected responses there is a corresponding bit in the mask. Each bit in the mask can be 0 or 1. 1 indicates that the corresponding bit in the produced response is a care bit and should be checked with the expected response otherwise it is a don't-care bit and should be masked. The test stimuli, expected responses, and mask are compressed and stored in the ATE memory. A test program, executed on a on-chip processor is used for decompression and only test independent evaluation logic is added to the SOC. The proposed approach focuses on test-architecture design and test-data compression and test scheduling is not considered.

3.7 Test-Architecture Design and Test Scheduling with Compression

The aim of integrating test-data compression with test-architecture design and test scheduling co-optimization is to reduce test application time and/or TAM width requirements by addressing the following three SOC test planning problems simultaneously: (1) test-architecture design, (2) test scheduling, and (3) test-data compression. Several such techniques have been proposed [Iye05], [Gon04a], [Wang07].

Iyengar and Chandra [Iye05] propose an approach where FDR codes is combined with test-architecture design and test scheduling. Two case studies are presented for the placement of the on-chip decoders: (1) one decoder per TAM wire and (2) one decoder per core. A rectangle packing algorithm is presented, which solves the test-architecture design and test scheduling problem.

Wang *et al*. [Wang07] integrate test-data compression with test-architecture design and test scheduling. The proposed test-architecture is illustrated in Figure 3.19 for the example SOC in Figure 2.3. An LFSR and a phase shifter are used that provide test-data to one or more cores at the same time, testing them concurrently. The external W_{TAM} TAM wires are expanded into a

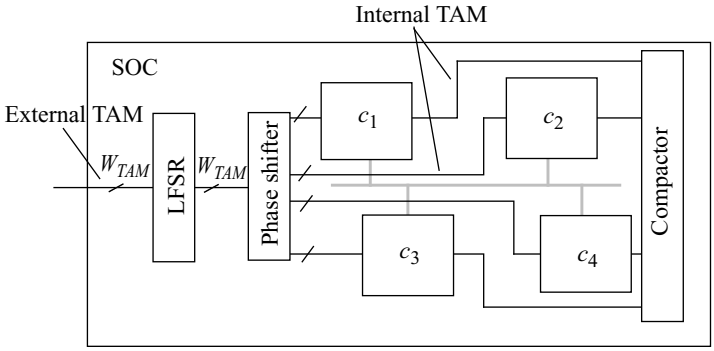


Figure 3.19: Test-architecture with test-data compression proposed by [Wang07].

number of internal TAM wires. The produced responses are compacted. The seeds, which are used for the LFSR, are calculated using the care-bits in the test-data for each core during the test scheduling. The goal is to maximize the number of care-bits that the LFSR can produce in each clock cycle, hence, minimizing the test application time.

Gonciari and Al-Hashimi [Gon4a] proposed a test-data compression driven TAM architecture design approach. The decoder consists of a shift register and an XOR-network which are used to expand a two-bit external TAM into a number of internal TAM wires.

The approaches proposed by Wang *et al*. and by Gonciari and Al-Hashimi use only one decoder which is designed at the SOC-level, therefore, there is no way to trade-off the amount of compression achieved and the test application time at core-level for the system. The proposed approaches described in this section require a large number of TAM wires to achieve an acceptable test application time for the system.

As expected, these techniques show that test-data compression leads to a reduction in test application time for the core-based SOC. However, they do not provide any quantitative insights on the test-time reduction (at the SOC-level) derived from adding a decoder for any given embedded core. Many test-data compression methods provide higher compression when a slight increase in test application time for a core is allowed through the use of a narrow TAM. In such cases, prior work does not provide any means to trade-off the

compression at the core-level considered in isolation with the test application time for a core in the overall SOC test schedule. Another drawback of previous methods is that they lead to irregular test-access architectures, which require specialized TAM optimization and test scheduling solutions at the SOC-level.

3.8 Test-Architecture Design and Test Scheduling with Compression and Sharing

The aim of integrating test-data compression and test sharing with test-architecture design and test scheduling co-optimization is to reduce test application time and/or TAM width requirements by addressing the following four SOC test planning problems simultaneously: (1) test-architecture design, (2) test scheduling, (3) test-data compression, and (4) test sharing.

Zeng and Ito [Zen06] proposed a concurrent core test approach using shared tests. Dedicated given tests for different cores are shared using a proposed sharing algorithm. A one-bit TAM is used to broadcast the shared test to the cores. A scan chain disable technique is used to restore the original test-data for each core from the shared test and a MISR is used for the compaction of the produced responses. For test application, two different strategies are proposed: by using an on-chip scan chain disable signal generator and by using an on-chip decoder. In the proposed approach, it is assumed that all cores will be connected through a common TAM wire; hence, all cores will share a test. Further, the test application time can be long for systems with large cores with many scan chains, which have to be connected into one long chain.

3.9 Summary

This section is used to summarize the related work. The section is also used to differentiate the related work from the work presented in this thesis.

For the test-architecture problem the IEEE Std. 1500 has been developed to achieve core isolation, test access, and test mode control. A *Design_wrapper* algorithm has been proposed that organizes the scan elements (scan chains, input wrapper cells, and output wrapper cells) into wrapper chains such that the test application time is minimized [Iye01a].

Several TAM architectures have been proposed. The shortest possible test application time can be achieved using Direct access. However Direct access is not applicable in practice due to the limited amount of I/O pins. The Multiplexing, Daisychain, and Distributed TAM architectures solve the problem of limited number of I/O pins. However, they do not allow a flexible test scheduling approach, since all cores are either tested one at a time or all at the same time. The Test bus and TestRail TAM architectures allow a more flexible test scheduling approach since the total number of TAM wires can be partitioned into several Test buses/TestRails. However, in the Test bus and TestRail TAM architectures, the cores assigned to a TAM are connected to all wires of that TAM, which limits the flexibility of the test scheduling. The Flexible-width architecture allows a flexible test scheduling approach as each TAM wire can be treated as a separate unit. The Flexible-width architecture, however, potentially leads to an irregular organization of the test-data in the tester memory and an advanced test controller may be required.

In terms of test scheduling, the non-partitioned test scheduling scheme does not allow any new test to start until all tests in a session are completed. This method produces long test application times due to long idle periods. The test application time can be reduced by using a partitioned (sessionless) technique, where tests are allowed to be scheduled as soon as possible. However, a more advanced test controller is required for the invocation of tests since more possible start times of tests can be used. Further optimization of the test schedule is possible by applying a pre-emptive test scheduling technique, where tests can be pre-empted and resumed at a later point in time. Pre-emptive test scheduling requires, on the other hand, an advanced test controller and is not applicable to all types of tests.

Several test-data compression schemes have been proposed that successfully reduce the test application time and test-data volumes. The benefit of test-data compression can also be further enhanced if the test-data compression is combined with SOC-level test-architecture design and test scheduling.

For test sharing and broadcasting, the benefits can be improved if they are co-optimized with SOC-level test-architecture design and test scheduling. However, techniques that make use of fault simulation are usually too time-consuming to be included in the optimization.

Several co-optimized test-architecture design and test scheduling techniques have been proposed. These techniques successfully reduce the test

application time or TAM width, while considering various resource constraints. None of these related approaches, however, consider test-data compression and/or test sharing.

For test-architecture design and test scheduling with test-data compression, several approaches described require a large number of TAM wires to achieve an acceptable test application time for the system. Most approaches uses only one decoder which is designed at the SOC-level, therefore, there is no way to trade-off the achieved test-data compression with the test application time at core-level. Further, they do not provide any quantitative insights on the test-time reduction (at the SOC-level) derived from adding a decoder for any given embedded core.

For test-architecture design with test-data compression and test sharing, only one technique where it is assumed that all cores will be connected through a common TAM wire, has been proposed. This means that all cores will be tested concurrently.

None of the related work makes use of both functional buses in combination with dedicated TAM architectures. Previous work also do not explore the trade-off between test-data compression and test sharing in terms of test-data volume. None of the related work allows test sharing to be combined with a flexible test scheduling technique.

In this thesis, we analyze and explore several design and optimization problems related to core-based SOC test planning. We perform optimization of test sharing and test-data compression. We explore the impact of test compression techniques on test application time and compression ratio. Furthermore, we make use of analysis to explore the optimization of test sharing and test-data compression in conjunction with test-architecture design and test scheduling.

Chapter 4
Preliminaries

THE PURPOSE OF this chapter is to give some preliminaries for the thesis. First, the system model is presented and a detailed description of the application of tests to cores is given. In the second section, the TAM architectures are described, and finally, the scheduling of tests is discussed.

4.1 System Model

It is assumed that a system consisting of N cores, c_1, c_1, ..., c_N, which are connected to at least one functional bus, is given. The system is tested by applying a number of tests to the cores. The test stimuli are generated/or stored in a test pattern source and the test responses are evaluated using a test pattern sink. A TAM is used to transport the test stimuli from the test source to the core and to transport the produced responses from the core to the test sink. All sequential cores are assumed to be equipped with scan chains and to facilitate interfacing with the TAM, each core has a wrapper. For each core c_i the following is given:

- sc_i - the number of scan chains,
- ff_{ij} - the number of FFs in scan chain j, where j = {1, 2, ..., sc_i},
- wi_i - the number of input wrapper cells, and

- wo_i - the number of output wrapper cells.

The total number of FFs nff_i in a core can be calculated as:

$$nff_i = \sum_{j=1}^{sc_i} ff_{ij} \quad (4.1)$$

It is assumed that each core is delivered with a dedicated test T_i, as follows.

- $T_i = \{TS_i, ER_i\}$ - a given dedicated test consisting of test stimuli TS_i and expected responses ER_i,
- $TS_i = (ts_{i1}, \ldots, ts_{il})$ - a sequence of l test stimuli patterns, where ts_{ik} consists of $nff_i + wi_i$ bits and each bit can be 0, 1, or x.
- $ER_i = (er_{i1}, \ldots, er_{il})$ - a sequence of l expected response patterns, where er_{ik} consists of $nff_i + wo_i$ bits and each bit can be 0, 1, or x.

At test application the test stimuli are transported from the test source on the TAM, to the core, through the input test pins, *t-in*, as illustrated in Figure 2.12. When they have been applied, the produced responses are transported to the test sink through the outputs, *t-out*.

Figure 4.1 illustrates, using the example SOC in Figure 2.3, which is tested using an ATE, the assumed given system architecture. Figure 4.1 also illustrates the organization of test stimuli and expected responses in the ATE memory.

4.2 Test-Architecture Design

This section describes the wrapper design and the TAM architectures that are used throughout this thesis.

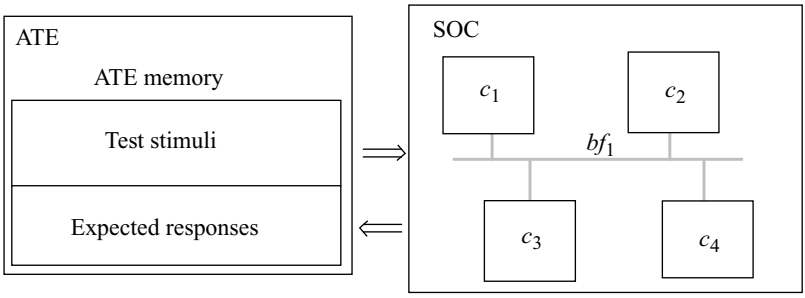


Figure 4.1: Example SOC in Figure 2.3 and ATE memory organization.

4.2.1 Wrapper Design

As described in Chapter 2 and Chapter 3, the wrapper design implies two separate issues: (1) the wrapper architecture selection and (2) the wrapper design optimization.

We assume that each core has an IEEE Std. 1500 wrapper architecture, described in Section 3.1. An example of a wrapper design is illustrated in Figure 4.2 using core c_1 and c_2 in Figure 2.3. In the example, the scan chains *a* to *d* and *e* to *g* have been grouped into three wrapper chains. We make use of the *Design_wrapper* algorithm, described in Section 3.1.1, to solve the wrapper design optimization problem.

4.2.2 Test Access Mechanism Architecture

Throughout this thesis we make use of the following three TAM architectures: (1) functional bus access, (2) Test bus, and (3) Flexible-width architecture, which are described in Section 3.1.2.

How wrapped cores are connected to the TAM wires is illustrated in Figure 4.2 using c_1 and c_2. In the example, each core has three wrapper chains that are connected to six TAM wires TAM_1 to TAM_6. In the example TAM architecture, three TAM wires are used for transporting test stimuli and three TAM wires are used for transporting produced responses.

When a MISR is used to compact the produced responses, all TAM wires can be used for the transportation of the test stimuli. An illustration of such a MISR-based test-architecture is presented in Figure 4.3 using c_1 and c_2, which are connected to three TAM wires.

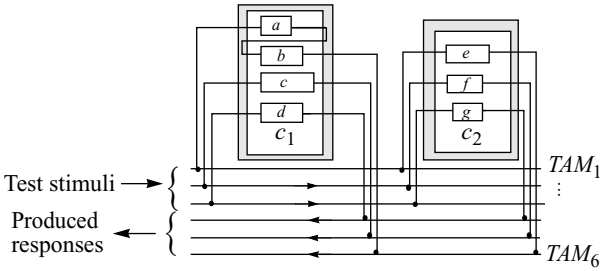


Figure 4.2: Test-architecture.

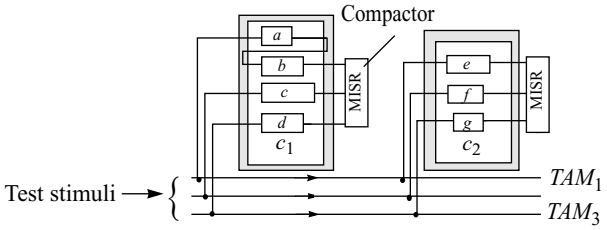


Figure 4.3: Test-architecture using MISRs.

4.3 Test Scheduling

This section describes the test scheduling techniques used. The test scheduling is described for the bus-based TAM-architecture (functional bus access and Test bus) and for Flexible-width architecture. A formula of the test application time for a system is also presented.

Using bus-based TAM architectures, such as functional bus access and Test bus, for transporting test-data usually entails a sequential schedule, and hence, only one core is tested at a time, as illustrated in Figure 4.4. The transportation of tests on the functional bus bf_1 is shown in Figure 4.4 (a). The example shows that the bus is the critical resource; it is fully occupied all the time. Still, the cores are only activated one after the other (Figure 4.4 (b)). This makes the scheduling very simple. The drawback, however, is the long test application time obtained since the cores are not tested in parallel.

When a bus-based TAM is used, concurrent test scheduling where multiple tests are scheduled in parallel is only possible by adding multiple TAMs. Such a test-architecture is illustrated in Figure 4.5 where two dedicated Test buses, bt_1 and bt_2, are used. In the example c_1 and c_2 have been assigned to bt_1 and c_3 and c_4 have been assigned to bt_2. An example of a concurrent test schedule using bt_1 and bt_2 is illustrated in Figure 4.5. The transportation of tests on the dedicated Test buses bt_1 and bt_2 is shown in Figure 4.5 (a). The corresponding test application, where c_1 is tested at the same time as c_3 and c_4, is shown in Figure 4.5 (b).

As opposed to the bus-based TAM architectures, the Flexible-width architecture allows concurrent test transportation and application even if only one TAM is used. An example of concurrent test scheduling and application using the Flexible-width architecture is shown in Figure 4.6. The

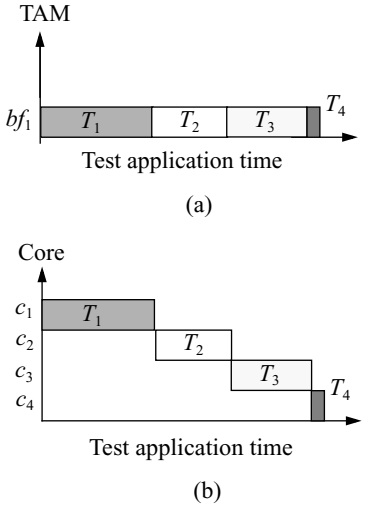


Figure 4.4: Example of (a) sequential test scheduling and (b) test application using one functional bus bf_1.

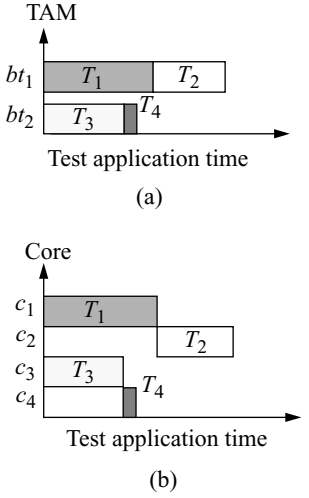


Figure 4.5: Example of (a) concurrent test scheduling and (b) test application using two Test buses bt_1 and bt_2.

transportation of tests on the TAM wires is shown in Figure 4.6(a). The corresponding test application, where c_1 is tested at the same time as c_2 and c_3, is shown in Figure 4.6(b).

We present a formula for the test application time τ_{tot} for a system with q tests, which is independent of the selected TAM architecture as:

$$\tau_{tot} = \max\{t_i + \tau_i(w)\}, \forall i, i \in \{1, 2, \ldots, q\}, \tag{4.2}$$

where t_i is the start time when the test is applied to the core c_i and $\tau_i(w)$ is the test application time when w wrapper chains are used.

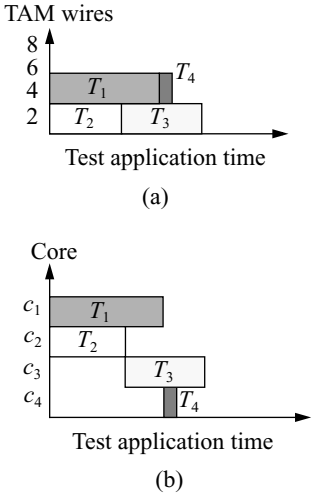


Figure 4.6: Example of (a) concurrent test scheduling and (b) test application using a Flexible-width architecture.

Chapter 5
Test-Architecture Design and Scheduling with Sharing

THIS CHAPTER PRESENTS a technique to minimize the test application time by exploring the test sharing and broadcasting of tests to multiple cores. First, the proposed technique and the used test-architecture are introduced. The test sharing problem and the proposed test sharing technique are described and are followed by an analysis of the test sharing. The broadcasting of a shared test to multiple cores is also described. The problem is motivated using an example and formulated in detail. A description of the CLP formulation used to solve the problem is presented and is followed by the experimental results and conclusions.

5.1 Introduction

In this chapter, we propose a test-architecture design and test scheduling technique to minimize the test application time by exploring the test sharing and broadcasting of tests to multiple cores.

Dedicated TAMs decrease the test application time for the system but contribute to increased wiring overhead, and hence increased hardware overhead. An alternative to adding dedicated TAMs is to reuse the functional (system) bus for the purpose of SOC testing. SOC designs often contain multiple functional buses. Such multiple functional bus system offers the opportunity for concurrent test application where two or more cores, which are connected to different functional buses, can be tested in parallel. Reusing the functional bus for SOC test purpose entails that a connector (or bus-wrapper) is added between the core and the functional bus to separate the functional mode from the test mode. Such a connector will be associated with a hardware overhead but, as opposed to the alternative with dedicated TAMs, reusing the functional bus does not require additional wiring.

As discussed in Chapter 3, several approaches have been developed in this area. Test sharing has been proposed as a method for reducing the test-data volume and test application time [Jia03], [Lee99], [Shi05], [Zen06]. Several approaches assuming a dedicated TAM for test-data transportation have been proposed [Aer98], [Goel03], [Iye03]. Functional bus for SOC testing has been proposed [Har99], [Hus06], [Hwa01].

None of the related work solves the test-architecture design and test scheduling problems at SOC-level while considering test sharing and broadcasting. Furthermore, all of the related work assume a fixed test set for each core and assume, either a dedicated TAM or the functional bus structure as a mechanism for test transportation. The high number of don't-cares in the test-data are not explored in the optimization.

In this chapter, it is assumed that given is a core-based SOC and that both functional buses and dedicated test buses can be used for test transportation. Further, a method for generating shared tests that are added as alternative tests to the cores that share the test is presented. At test application, the test stimuli of the shared test are broadcasted to all cores that share the test and the produced responses are transported on dedicated TAM wires separately. Separation of the produced responses is required since cores that share a test can output different produced responses, which cannot share the TAM in their way back to the ATE for evaluation. Consequently, the number of TAM wires

and the test application time that a core requires depend on whether a shared test is used or not. For example, a shared test will have to use fewer TAM wires for the test stimuli compared to a dedicated test for one core; as each core must be able to transport its test responses to the ATE. Hence, the test application time at core-level is longer for the shared test. However, since the shared test is used to test multiple cores in parallel, the overall test application time at SOC-level can be lower than if the cores were tested one at a time using the initially given dedicated tests. It, therefore, exists a trade-off between test sharing and test-architecture design in terms of test application time.

We explore the following two trade-offs: (1) between test sharing and test-architecture design in terms of test application time, and (2) between the test application time and the number of TAM wires used, as a consequence of adding test buses. The major contributions are as follows:

- Test sharing and broadcasting of test patterns for core-based SOCs are addressed. The shared tests serve as alternatives to the initially given dedicated tests for the cores, which means that the test is not longer fixed for one core. We also show how the efficiency of test sharing depends on the density of don't-care bits present in the tests.
- The test-architecture design and test scheduling problems are solved while minimizing the systems test application time. The test application time is minimized without exceeding a hardware overhead constraint. The proposed test-architecture offers a possibility to reuse on-chip functional connections, such as the functional bus, for test transportation, hence, reduces the hardware overhead.

5.2 Test-Architecture

The test-architecture is described using the example SOC design in Figure 2.3 consisting of core c_1, c_2, c_3, and c_4, which are tested by the given dedicated tests T_1, T_2, T_3, and T_4, respectively. In the example, the cores are connected to one functional bus bf_1.

We assume that the buses are connected to the I/O pins of the chip, and hence, directly accessible and controlled from the ATE. Furthermore, it is assumed that both functional buses and dedicated test buses can be used for transporting test stimuli and produced responses. In the example illustrated in

Figure 5.1[1] the design in Figure 2.3 has been extended with one dedicated test bus bt_1. A dedicated test bus for the transportation of test-data will increase the transportation capacity and shorten the test application time. The alternative to add dedicated test buses also offers the possibility of a trade-off between the test application time and the number of TAM wires used.

It is assumed that one or several test buses may be added to the design as long as the given hardware overhead constraint is not exceeded. Furthermore, a connector, consisting of logic needed for the communication and application of test-data is inserted between each core and the bus. For example, of_{21} is the connector connecting core c_2 with functional bus bf_1, as shown in Figure 5.1. When the system is in functional mode, the functional inputs and outputs at each core are connected to the functional bus. When the system is in testing mode the connectors will receive control signals, indicating when a pattern should be applied. The hardware cost, such as additional wiring and control logic needed to connect a core to a functional bus or a test bus, or to add a test bus, is assumed to be given by the designer.

The transportation and application of tests to the cores is illustrated in Figure 5.2 by considering cores c_1 and c_2 from Figure 5.1, which are tested by test T_1 and T_2, presented in Figure 5.3, respectively.

As described in Section 2.3, the general approach to scan testing entails a concurrent scan-in and scan-out phase, that is, when one test pattern is shifted

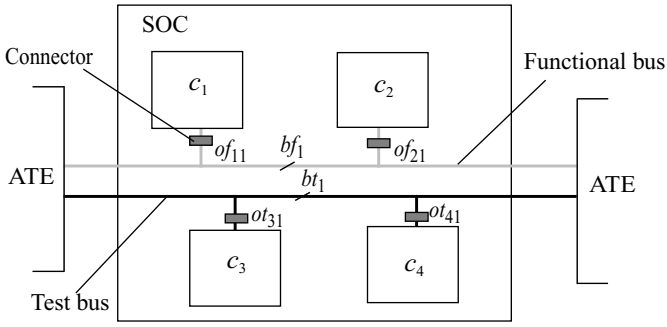


Figure 5.1: SOC test-architecture with one functional bus and one dedicated test bus.

1. Only the TAM architecture is illustrated. For this, and following examples in this chapter, it is assumed that core c_3 and c_4 are also connected to the functional bus bf_1.

out, a new test pattern is shifted in. Therefore it is not possible to share the same TAM wires for the test stimuli and produced responses of one core; the total number of TAM wires used to test one core is twice the number of wrapper chains w of the core.

In the example, presented in Figure 5.2, it is assumed that both cores are connected to the functional bus bf_1 with 6 TAM wires ($w_{TAM} = 6$), which means that the two cores can have a maximum 3 wrapper chains ($w \leq 3$). The longest wrapper scan-in and scan-out chain for the example in Figure 5.2 is for c_1 and c_2 7 clock cycles. The test application time $\tau_1(3)$ for applying the 3 test patterns in T_1 to c_1 with 3 wrapper chains is $\tau_1(3) = (1 + 7) \times 3 + 7 = 31$ clock cycles. The test application time $\tau_2(3)$

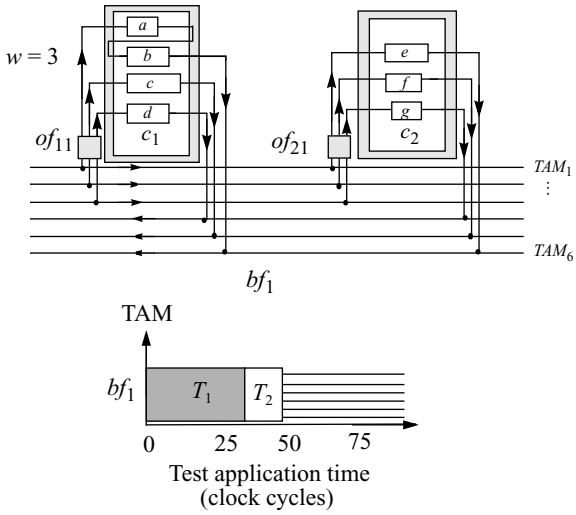


Figure 5.2: Test-architecture and test schedule.

Scan chain

T_1

	a	b	c	d
TS_1	0xx	xx11	xxxx1	xxxx
	0xx	xxxx	xxx00	xxx1
	xx1	xxx0	xx0xx	xxx0
ER_1	1xx	xxx1	xxxx0	xxxx
	11x	xxxx	xxxx1	xxxx
	xx0	xxxx	xxxxx	xxxx

T_2

	e	f	g
TS_2	xx11	xx0	0x
	xx0x	xxx	1x
	xx10	1x1	xx
ER_2	xxx0	xx0	x1
	xxx1	x1x	xx
	1xxx	x0x	x0

Figure 5.3: Initially given tests with don't-cares.

for applying the 3 test patterns in T_2 to c_2 with 3 wrapper chains is $\tau_2(3) = (1+4)\times 3+4 = 19$ clock cycles. Since the cores are tested sequentially, one at a time, the total test application time is 50 (31 + 19) clock cycles.

5.3 The Test Sharing Problem

The aim of test sharing is to lower the tester memory requirement and the test application time as discussed in Section 2.4.4.

The sharing of two tests is illustrated in Figure 5.4. The test stimuli sequences TS_1 and TS_2 from Figure 5.3 have been formed into two wrapper chains, wr_1 and wr_2. In the example, scan chains *a, c,* and *e* have been assigned to wrapper chain wr_1 and scan chains *b, d, f,* and *g* to wrapper chain wr_2 which corresponds to the architecture in Figure 5.2.

For balancing the wrapper chains before sharing, idle bits are added such that all wrapper chains have equal length (using minimum transition fill) as illustrated in Figure 5.4(a). Three possible test sequences can potentially be overlapped with ts_{11}: ts_{21}, ts_{22}, or ts_{23}. As shown in Figure 5.4(b), ts_{11} and ts_{21}, are not overlapping (there are conflicting care bits that prohibit overlapping), hence, they cannot be shared. In Figure 5.4(c) ts_{11} and ts_{22}, are overlapping and a new, shared, test sequence *ts_new* is generated.

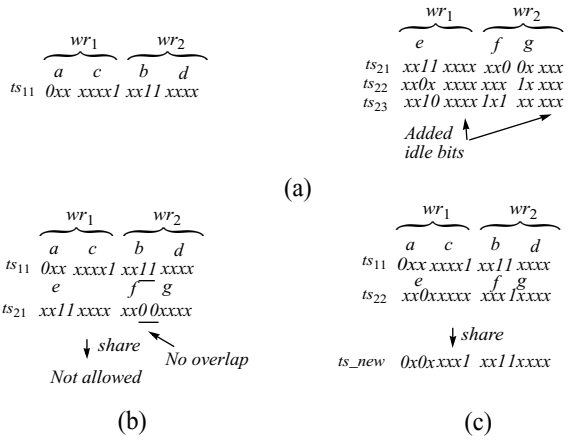


Figure 5.4: Examples of test sharing using (a) different test sequences when (b) no overlap is achieved and (c) when an overlap is found .

5.4 The Proposed Sharing Function

We introduce a function called *share* that takes two tests[1] (test stimuli and expected responses), T_i and T_j, as input and generates a new alternative test, T_k:

$$share(T_i, T_j) \rightarrow T_k \tag{5.1}$$

The *share* function, illustrated in Figure 5.5, is performed in two steps. In the first step (line 4–8), the test stimuli are sorted according to the percentage of don't-care bits, such that the sequences with the most care bits are placed first in each test. The sorting is done in order to increase the utilization (filling) of the don't-care bits in each sequence. The test with most patterns is selected as the reference, *ref_test*. In the example, test T_1 is selected. In the second step (line 9–23), the sharing is performed by finding overlapping sequences.

The search for overlapping sequences is performed using two loops. The outer loop (line 10) is used to iterate over the sequences in the reference test T_1 while the inner loop is used to iterate over the sequences in the other test T_2. Each test sequence in the reference test is compared with all test sequences in the other test using the inner loop (line 11). If an overlap sequence is found, a

```
1  Procedure Share(T1, T2)
2    // Input: Tests T1 and T2
3    // Output: A new test, new_test
4    // Step1
5    Sort(T1) // Sort T1 according to % of don't-cares
6    Sort(T2)
7    ref_test = GetRefTest(T1, T2)
8    new_test = {}
9    // Step2: Find overlapping sequences
10   For each sequence i in ref_test // ref_test = T1
11     For each sequence j in T2
12       If ts_new = Overlap(ts1i, ts2j)
13         new_test = new_test ∪ {ts_new}
14         Break
15     new_test = new_test ∪ {ts1i}
16     For each sequence j in T2
17       If ts2j is not previously added to new_test
18         new_test = new_test {ts2j}
19   Return new_test
```

Figure 5.5: The *share* function.

1. From here and through the rest of the thesis T_i is used to denote a given dedicated test or an alternative test.

new test sequence *ts_new* is generated and added to the new shared test T_k and the inner loop is terminated. Those sequences that are not subject to an overlap are copied to T_k. The size of T_k will be equal to the size of the reference test if there exists an overlapping sequence for all sequences in the reference test. If an overlap is not found the size of T_k is increased.

Figure 5.6 shows the result after using the proposed *share* function to generate shared test stimuli from TS_1 and TS_2 in Figure 5.3. For the example, ts_{11} is shared with ts_{22}, ts_{12} with ts_{21}, and ts_{13} with ts_{23}.

5.5 Analysis of Test Sharing

A high number of don't-care bits increases the possibility of identifying patterns from different tests that can be efficiently shared. We have performed experiments to investigate the relationship between the number of don't-care bits and the test-data volume of the shared test.

The *share* function has been applied to the benchmark design d695 from the ITC'02 benchmark set [Mar02]. The test patterns for each core in d695 (with don't-cares marked) have been generated by Kajihara and Miyase [Kaj01]. The test-data characteristics for d695 are presented in Table 5.1. Column 1 lists the name of each core in the system. Column 2 and Column 3 list the number of test patterns and the number of scan chains, respectively. Column 4 lists the percentage of don't-cares in the test.

The relative test-data volume when applying test sharing is given by:

$$\left(1 - \frac{((\mu_i + \mu_j) - \mu_k)}{(\mu_i + \mu_j)}\right) \times 100, \tag{5.2}$$

where μ_k ($\mu_k \geq \min\{\mu_i, \mu_j\}$) denotes the test-data volume of the new shared test, μ_i and μ_j denote the test-data volume of the un-shared (original) tests T_i and T_j, respectively.

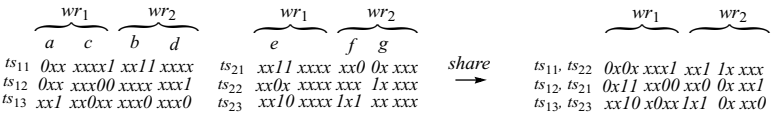


Figure 5.6: Result after applying the *share* function to TS_1 and TS_2.

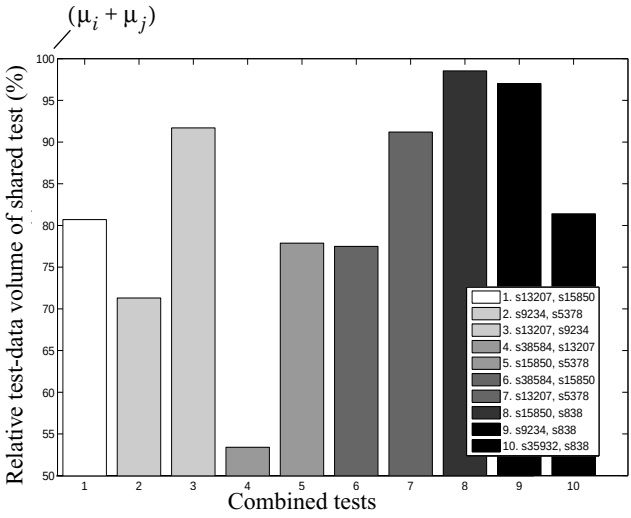
Table 5.1: Test-data characteristics for d695

Core i	No. of test patterns l	No. of scan chains sc_i	Percentage of don't-cares
c6288	12	-	0
c7552	73	-	54.31%
s838	75	1	60.93%
s9234	105	4	68.70%
s38584	110	32	80.83%
s13207	234	16	92.02%
s15850	95	16	77.22%
s5378	97	4	73.11%
s35932	12	32	36.20%
s38417	68	32	73.09%

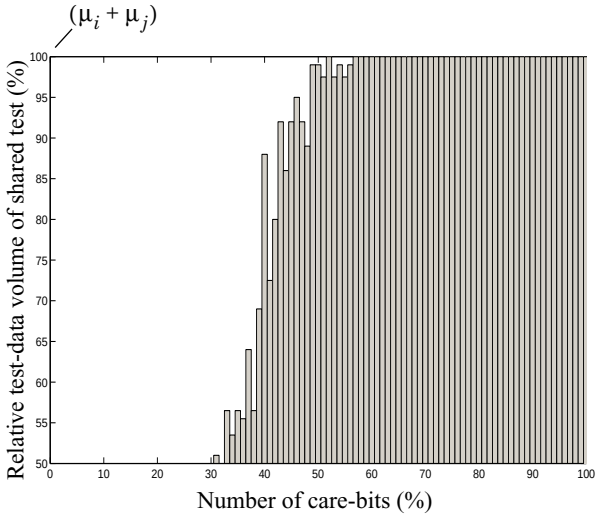
The relative test-data volumes from experiments on 10 different combinations of tests are presented in Figure 5.7(a). The relative test-data volume is between 50% and 100%. When the relative test-data volume is 50%, it means that two tests completely overlap each other. This occurs when two identical tests are applied to two identical cores. The gain in sharing is optimal as it is obvious that one test can test both cores. When the relative test-data volume is 100%, there is no gain in sharing. The two tests are not matching at all. In this case, no test patterns overlap and the test-data volume of the shared test will be equal to the total test-data volume of the un-shared tests ($\mu_i + \mu_j$). The results show that the test-data volume of the shared tests are on average 82% of the total test-data volume of the un-shared tests. This means that if test sharing is used, on average 18% less test-data needs to be stored in the tester memory.

To illustrate the relationship between the number of don't-cares and the test-data volume of the shared test, a number of tests (available at [Lar06b]) were shared. The results depicted in Figure 5.7(b) show that when the number of care bits is in the range of 0 to 50% the test-data volume of the shared test is on average only 60%. This corresponds to a saving of 40%.

Our analysis confirms the expected, that the possibility of sharing two tests is dependent on the density of don't-cares present in the tests. However, beside the density of don't-cares, the test-data volume of the input tests will also have an impact on the sharing efficiency. That is, sharing tests of equal test-data volume is more efficient than if two tests with different test-data volume are shared. This is explained using the following small example. Let us assume three tests with test-data volume 100 bits, 90 bits, and 20 bits, respectively.



(a)



(b)

Figure 5.7: Relative test-data volume of shared tests for (a) different combinations from d695 and (b) tests with increasing number of care-bits.

Sharing the test with 100 bits and the test with 20 would, in the best case, lead to a test-data volume reduction of 20 bits, corresponding to 17% (20 / 120). If the test with 100 bits instead was shared with the test with 90 bits, the best case decrease of the test-data volume would be 90 bits, corresponding to 47% (90 / 190).

5.6 Broadcasting of a Shared Test

This section describes the test-architecture and the test application time reduction when a shared test is transported in a broadcast manner.

In order to attain the possible test application time reduction of a shared test, the following two issues must be addressed: (1) It is required that the cores, which share the test, are connected in such a way that the test stimuli can be broadcasted to the cores, and (2) the produced responses from different cores cannot be shared since the sequences are different.

The first issue is solved by connecting the cores that share the same test to a common set of TAM wires. For the second issue, we make use of an architecture where the produced responses from each core will be transported to the tester on separate TAM wires. Figure 5.8 shows the test-architecture for two cores, c_1 and c_2, that share one test. In the example, T_1 and T_2 have been shared and a new shared test T_5 has been generated. The test is broadcasted to the two cores, hence testing both cores concurrently.

The number of wrapper chains w for a test or alternative test T_i depends on the number of cores z that share the test, and is given by:

$$w = \lfloor w_{TAM}/(z+1) \rfloor \tag{5.3}$$

If no test sharing is used ($z = 1$) $w = w^{TAM}/2$, where w^{TAM} is the number of TAM wires. This means that half of the TAM width is used for the transportation of test stimuli and the second half is used for produced responses as illustrated in Figure 5.2. In the case when two cores share a test ($z = 2$) $w = w^{TAM}/3$; one third of the TAM width is occupied transporting the test stimuli that are broadcasted to both cores and two thirds are used for the produced responses, one third for each core separately as illustrated in Figure 5.8.

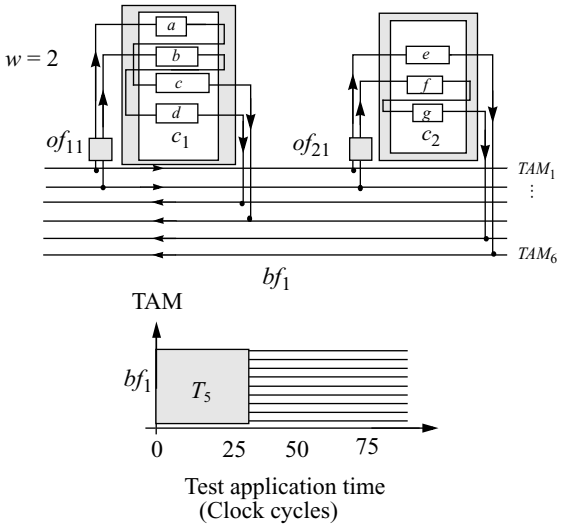


Figure 5.8: Test-architecture and test schedule with sharing.

5.7 Motivational Example

This section illustrates the two trade-offs explored: (1) between test sharing and test-architecture design in terms of test application time, and (2) between the test application time and the number of TAM wires introduced by adding test buses.

The trade-off between test sharing and test-architecture design in terms of test application time is illustrated using the cores, c_1 and c_2, in Figure 5.8. Both cores are tested concurrently using the shared test T_5, which is broadcasted to the cores. The number of wrapper chains at each core is reduced from 3, in Figure 5.2 when test sharing and broadcasting are not used, to 2. Fewer wrapper chains usually lead to longer test application time for each individual core as the scanned elements are formed in longer chains. For example, the longest wrapper scan-in and scan-out chain for the example in Figure 5.8 is 8 clock cycles for c_1 and c_2. Therefore, the test application time $\tau_2(2)$ for the shared test T_5 and 2 wrapper chains is $\tau_5(2) = (1+8)\times 3 + 8 = 35$ clock cycles. This is more than the 31 clock cycles needed to apply T_1 in the example in Figure 5.2. However, since both c_1

and c_2 are now tested concurrently, the total test application time is 31 clock cycles, which is a significant reduction from the 50 clock cycles when test sharing and broadcasting are not used. The examples in Figure 5.2 and Figure 5.8 show that by using shared tests, which are broadcasted to multiple cores, it is possible to get a shorter test application time compared to a sequential application.

The second trade-off considered in this chpater is between the test application time and the number of TAM wires used (introduced by adding test buses). This trade-off is illustrated by using the example design from Figure 5.1 consisting of four cores, c_1, c_2, c_3, and c_4. In the example all cores are connected to one functional bus bf_1 as shown in Figure 5.9.

In the first schedule shown in Figure 5.9(a) the shared test (T_5) is not used, while in the second schedule, Figure 5.9(b), T_5 is introduced and since it can be applied to the two cores c_1 and c_2 concurrently, the test application time is decreased. The test application time may be further decreased if a dedicated test bus, bt_1, is introduced as illustrated in Figure 5.9 (c and d). The dedicated test bus will enable concurrent application of tests. Figure 5.9(c) shows an alternative assignment of cores to buses. In the example only one core is tested through the test bus but the test application time has decreased compared with the example where only one bus was used. Since core c_1 is tested through the test bus it is not possible to make use of the broadcast capability between c_1 and c_2. However, a second alternative that leads to a further reduction of the test application time is to assign c_1 and c_2 to the same bus as shown in Figure 5.9(d). The example shows both that dedicated test buses can be used to reduce the test application time, and the importance of careful assignment of cores to buses.

5.8 Problem Formulation

Given is a system consisting of a number of cores as described in Section 4.1, with F functional buses, bf_1, bf_2, ..., bf_F, where each functional bus bf_i has w^{bf_i} wires. For the system we assume the following two constraints:

- W_{TAM} - the bandwidth of the ATE and
- K_{max} - the maximal allowed hardware overhead.

Also given is the *share* function, described in Section 5.4, that takes two tests as input and generates a new shared test that is added to the test sets. This

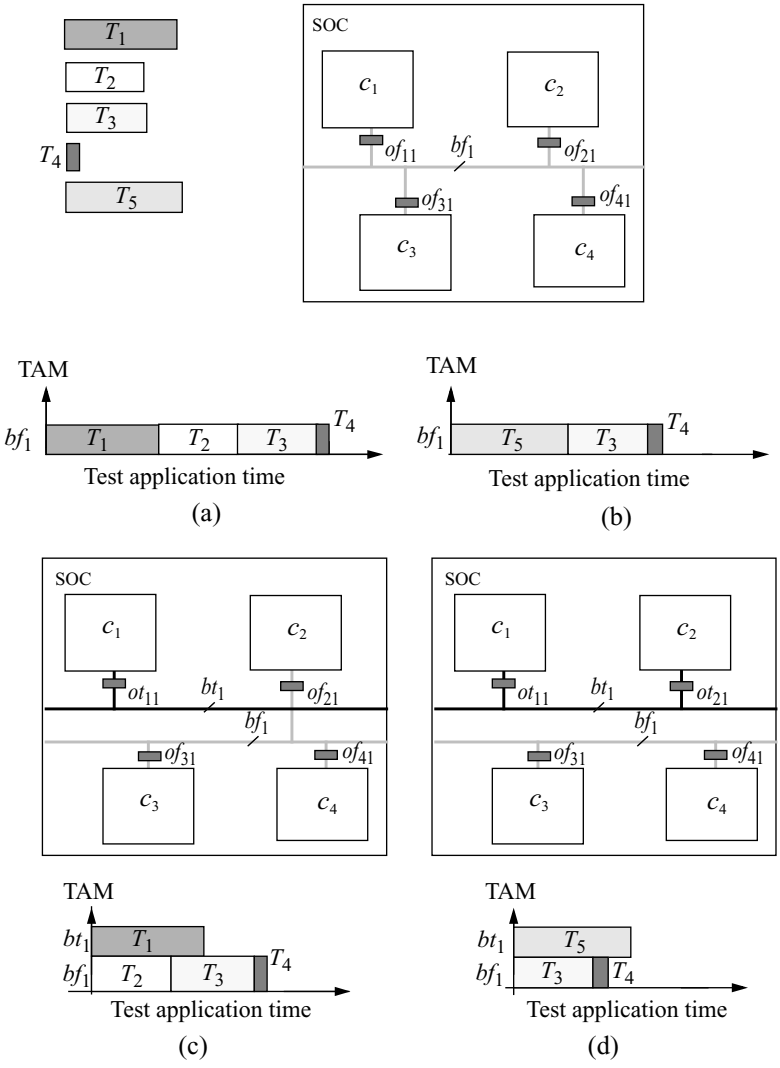


Figure 5.9: Example of (a) test-architecture design and test scheduling with one functional bus without broadcasting , (b) one functional bus with broadcasting, (c) one functional bus and one test bus, alternative 1, and (d) one functional bus and one test bus, alternative 2.

means that a core can be tested either with its dedicated given test or one of the alternative tests. The test application time $\tau_i(w)$ for a test is given by Equation 3.1 and the test application time τ_{tot} for a system with q tests is given by Equation 4.2.

As described in Section 5.2, the hardware cost of adding connectors of_{ij} between core c_i and the functional bus bf_j and ot_{ij} between core c_i and test bus bt_j are assumed to be given. The following hardware cost factors are considered:

- kf_{ij} - the cost of inserting a connector of_{ij} between core c_i and functional bus bf_j,
- kt_{ij} - the cost of inserting a connector ot_{ij} between core c_i and test bus bt_j,
- k^{bt}- the base cost of inserting test bus bt.

The total hardware cost K_{tot} is given by:

$$K_{tot} = \sum_{i=1}^{N} \sum_{j=1}^{F} \chi_{ij} \times kf_{ij} + \sum_{i=1}^{N} \sum_{j=1}^{g} \chi_{ij} \times kt_{ij} + \sum_{j=1}^{g} k^{bt}, \quad (5.4)$$

where g is the number of added test buses (determined during the optimization) and χ_{ij} ($\chi_{ij} = \{0, 1\}$) is a variable used to denote whether a connector is placed between core i and bus j or not.

The optimization objective is to:

- form the shared test alternatives,
- select at least one test for each core,
- determine how many test buses should be inserted,
- determine the number of wrapper chains w for each core,
- design the wrapper chains for each core,
- insert connectors between cores and buses, and
- schedule the transportation of selected tests on the buses

in such a way that the test application time is minimized.

The following constraints are imposed:

- The ATE bandwidth is limited to a certain value W_{TAM}, that is;

$$\sum_{i=1}^{F} w^{bf_i} + \sum_{j=1}^{g} w^{bt_j} \leq W_{TAM} \tag{5.5}$$

where F is the number of functional buses and g the number of added test buses

- The total hardware overhead cost is limited to a value K_{max}:

$$K_{tot} \leq K_{max} \tag{5.6}$$

5.9 Constraint Logic Programming Modelling

The problem has been formulated as a CLP problem. Prior to the CLP optimization, a pre-process stage is used in which the *share* function is applied to generate a number of shared tests, which are added as alternative tests for the cores. In total, q tests (dedicated tests and shared tests) are used for the system. The *Design_wrapper* algorithm, described in Section 3.1.1, and the *share* function are used to generate the test application times for various number of wrapper chains, which are stored in a look-up table used as input to the CLP program.

A description of the CLP formulation is given in Figure 5.10. The cores and information about the tests are first given as input (line 3–4 in Figure 5.10). A number of variables used to describe the solution is then defined, (line 6–12). In order to find a feasible solution that minimizes the total test application time (line 19) the program ensures that the following constraints are fulfilled (line 14–17):

- Each core must be connected to at least one functional bus or test bus (line 14).
- Each core must be tested (line 15).
- The hardware cost should not exceed the given maximum hardware cost, K_{max} (line 16), Equation 5.6.

We have used the following built in predicates in the CLP tool CHIP [Cos96], [Hen91] to ensure that all constraints are satisfied and the optimal solution is found:

```
1  run:-
2     // Get input data
3     Cores({1,2,3,... ,N}),
4     Tests({1,2,3,... ,q}),
5     // Define variables
6     g::1..MaxNrBuses,
7     K_tot::1..K_max,
8     τ_tot::1..τ_max,
9     ListOfTests::0..q,
10    ListOfCores::0..N,
11    Schedule::0..q,
12    Tam::1..W_ATE,
13    // Set up constraints
14    connect_all(Cores),
15    complete_cores(Cores,Tests),
16    count_costs(Cores,Costs, K_tot),
17    cumulative (Schedule, Duration, Resource, Tam, τ_tot ),
18    // Search for the optimal solution
19    min_max((labeling(Schedule)),τ_tot).
```

Figure 5.10: CLP formulation in CHIP for test application time minimization.

- cumulative (line 17), ensures that, at any given time, the total amount of resources does not exceed a given limit.
- min_max (line 19), implements a depth first branch and bound search for a solution with the minimal test application time.
- labeling (line 19), is used to assign values to the defined variables.

Since a test T_i can be used for several cores, a special constraint is implemented so that T_i is not scheduled more than one time as long as the cores tested by T_i share the same bus.

5.10 Experimental Results

In this section, we demonstrate the importance of integrating test sharing and broadcasting of test patterns with test-architecture design, wrapper design, and test scheduling.

For the experiments the eight designs, SOC_(1..7) [Lar06b] and the benchmark design d695, have been used. The main characteristics of the eight designs can be found in Table 5.2. Column 1 lists the designs. Column 2 and

Table 5.2: Design characteristics

Design	No. of cores N	No. of tests q
SOC_1	4	5
SOC_2	7	9
SOC_3	10	12
SOC_4	12	15
SOC_5	18	20
SOC_6	24	28
SOC_7	30	34
d695	10	12

Column 3 list the number of cores N and the number of tests q, respectively. In these designs it is assumed that each system has a 32 bit wide functional bus and that each test bus, if added to the system, has a width of 32 bits.

In the experiments the cost of connecting a core to a functional bus, kf_{ij}, is set to 10 units, the cost to connect a core to a test bus, kt_{ij}, to 20 units, and the cost of adding a test bus to the system, k^{bt}, is set to 100 units. For example, adding one test bus and connect one core to it is associated with a hardware cost of 120 units.

We have used the CLP tool CHIP (V 5.2.1) [Cos96], [Hen91] for the implementation and we have compared the cases when broadcasting is not used and when broadcasting is used.

The results are collected in Table 5.3. Column 1 lists the eight different designs. In Column 2 the hardware constraints are listed. These constraints have been set such that it is possible to add at least one test bus for each design. The following four columns, Column 3 to Column 6, contain the results from the first approach where no broadcasting is used. Column 3 lists the minimized test application time τ_{nb} and Column 4 lists the number of test buses g_{nb} added. Column 5 and Column 6 lists the number of test patterns used and the optimization time (CPU-time) required to find the optimal solution, respectively. The optimization time does not include the time to run the sharing function and the wrapper design. Column 7 to Column 10, contain the results when broadcasting is used. Column 7 lists the minimized test application time τ_b and Column 8 lists the number of added test buses g_b. Column 9 and Column 10 lists the number of test patterns and optimization time (CPU-time), respectively. The last column, Column 11, shows the comparison in test application time between the approach when no broadcasting is used τ_{nb} and the approach when broadcasting is used τ_b. The experiments show that broadcasting of tests between cores can shorten the test

Table 5.3: Test application time using broadcasting

Design	Hardware Constraint K_{max}	Without Broadcasting				With Broadcasting				Comparison $\frac{(\tau_b - \tau_{nb})}{\tau_{nb}} \times 100$
		Test application time τ_{nb} (clock cycles)	No. of added test buses g_{nb}	Total no. of test patterns used	CPU-time (s)	Test application time τ_b (clock cycles)	No. of added test buses g_b	Total no. of test patterns used	CPU-time (s)	
SOC_1	250	8271	1	275	14	6755	1	209	76	-18.33%
SOC_2	350	22361	1	440	76	18421	1	297	132	-17.62%
SOC_3	400	37943	2	539	391	29364	2	423	572	-22.61%
SOC_4	450	51946	2	753	624	37526	2	687	1517	-27.76%
SOC_5	500	86301	2	2316	1028	61730	2	1723	39843	-28.47%
SOC_6	500	1167250	3	15017	2734	869195	3	11483	62087	-25.53%
SOC_7	600	1602862	3	18779	4893	1187512	3	14021	95274	-25.91%
d695	350	24219	1	881	235	18522	1	802	586	-23.52%
									Average:	-23.72%

application time. The test application time could be decreased with 23.72% on average.

Experiments have also been made to show the impact on the test application time τ_b when broadcasting is used at different hardware constraints. The test application time minimization has been performed with different hardware constraints for SOC_1 and d695. The results collected in Table 5.4 show that the test application time for the designs decreases as additional test buses are added. Column 1 lists the two designs and Column 2 lists the hardware constraint K_{max}. Column 3 lists the number of added TAM wires used for the test buses. Finally, Column 4 lists the minimized test application time τ_b. The minimized test application time τ_b at different hardware constraints for designs SOC_1 and d695 are also presented in Figure 5.11.

As expected, these results show that adding test buses will significantly reduce the test application time. For example, for SOC_1 the test application time is reduced from 6155 clock cycles at K_{max} = 300 to 4329 clock cycles at K_{max} = 400. The test application time can still be reduced, even if no additional test bus can be added within the hardware constraint. Such reduction can for example be studied when K_{max} = 150 and K_{max} = 200 for SOC_1 where the test application time is reduced from 6421 clock cycles to 6221 clock cycles, respectively. In this case, the limited amount of which the hardware constraint was increased, did only allow additional connectors to be inserted.

Table 5.4: Test application time for different hardware constraints

Design	Hardware constraint K_{max}	No. of added TAM wires	Test application time τ_b (clock cycles)
SOC_1	40	0	7514
	150	32	6421
	200	32	6221
	300	64	6155
	400	96	4329
	500	96	4329
d695	100	0	26071
	250	32	22718
	300	32	20382
	400	32	18522
	500	64	13712
	600	64	12633
	700	128	11791

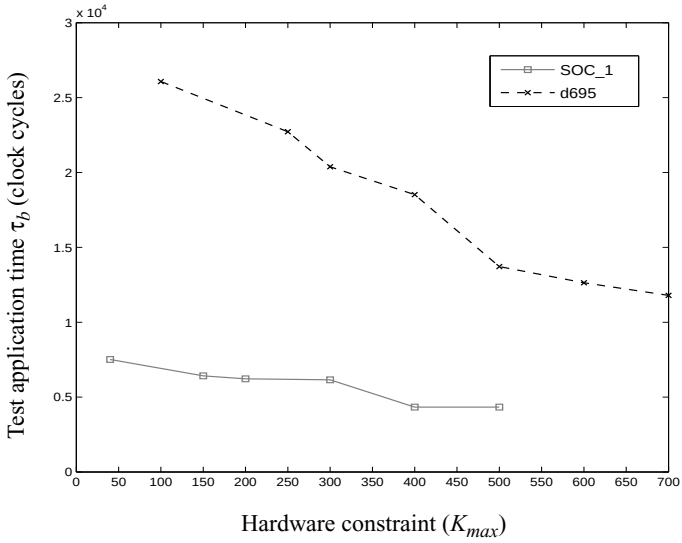


Figure 5.11: Minimized test application time τ_b at different hardware constraints K_{max} for designs SOC_1 and d695.

5.11 Conclusions

A scheme has been proposed to explore the high number of don't-cares present in the test-data to create new tests, which can be used as alternative to the original dedicated test for the cores. The new tests are shared and applied to several cores at a time.

There are a number of problems associated with the sharing of tests. For example, the test stimuli of the shared test should be broadcasted to all cores that share the test in order to reduce the test application time. Furthermore, separation of the produced responses is required since cores that share a test can output different produced responses, which cannot share TAM wires to the tester for evaluation.

The proposed method allows the existing functional bus structure to be reused for the test-data transportation. However, in order to decrease the test application time, dedicated test buses may be added to the design. The problem is to select appropriate tests for each core, design wrapper chains for

each core, insert test buses, and schedule the selected tests on the buses in such way that the test application time is minimized without exceeding the given hardware cost constraints. The problem described in this chapter has been modelled and implemented using CLP and experiments show that the overall test application time can be significantly reduced when broadcasting of tests is used. For the designs used in our experiments, the test application time was decreased with 23.72% on average.

Chapter 6
Test-Architecture Design and Scheduling with Compression and Sharing

THIS CHAPTER PRESENTS an integrated test-architecture design and test scheduling approach that utilizes both test-data compression and test sharing as mechanisms to reduce test application time and test-data volumes. First, the proposed technique and the used test-architecture are introduced. Second, the test-data compression and test sharing techniques are described. Third, the test-architecture design and scheduling technique with test-data compression and test sharing is presented. Fourth, the problem is formulated in detail and the proposed algorithm used to solve the problem is described. Finally, we present experimental results and conclusions.

6.1 Introduction

In this chapter, the test-architecture design and test scheduling as well as the test-data compression and test sharing problems, are addressed.

As discussed in Chapter 3, several test-data compression schemes have been proposed [Jas03], [Wang05], [Cha03a], [Bar01], [Raj04], [Koe01], [Cha03c], [Teh05], [Gon04b]. Several methods have been published to

combine test-data compression with TAM optimization and test scheduling [Gon04a], [Iye05], [Raj04], [Wang07]. And, several methods have been proposed to combine core-level test-data compression with SOC-level test-architecture design and test scheduling [Cha03c], [Goel03], [Gon04a], [Iye02a], [Iye03], [Iye05], [Raj04], [Seh04], [Wang07]. None of these techniques, however, considers test sharing.

For test-architecture design with test-data compression and test sharing, one technique have been propose [Zen06] where it is assumed that all cores will be connected through a common TAM wire. This means that all cores will be tested concurrently.

In this chapter, it is assumed that given is a core-based SOC with a dedicated TAM and tests for each core. A test-architecture that does not require test response compactors [Lar07c] is used. We make use of the 9C compression technique [Teh05], described in Section 3.3. This chapter is concentrated in particular to the following two issues:

- the relation between test-data compression and test sharing in terms of test-data volume, and
- the trade-off between test sharing versus test-architecture design in terms of test application time.

In order to understand the relation between test-data compression and test sharing in terms of the test-data volume, let us consider two tests. By test sharing, i.e., finding overlapping sequences in the two tests, which is used to create a new test, the amount of don't-care bits will decrease. Since the shared test will have less don't-care bits, it is likely that it will suffer from a lower test-data compression ratio compared to when the tests are compressed individually. This means that the size of the compressed shared test could be larger than the sum of the two separately compressed tests. Hence, it is not obvious to determine which tests should be shared and which tests that should be compressed.

The trade-off between test sharing and test-architecture design in terms of test application time is explained as follows: as described above, in the case of sharing, only the test stimuli are broadcasted to the cores while the produced responses are transported on separate TAM wires. Hence the TAM wire architecture will be different when using test sharing compared when test sharing is not used, consequently affecting the test application time.

The major contribution of this chapter is twofold. First, we show that the integration of test sharing and test-data compression for core-based SOCs will lead to decreased test-data volume. Second, we address the test scheduling and test-architecture design problem, exploring the trade-off between test sharing and test-data compression, while minimizing the test application time under ATE memory constraints. The efficiency of the proposed techniques has been demonstrated by experiments using ITC'02 benchmark designs.

6.2 Test-Architecture

In this section the test-architecture used for test-data transportation, decompression, and test sharing is described. In this chapter we make use of the flexible-width TAM architecture described in Section 3.1.2.

Let us first describe the common practice test-architecture, which does not make use of test-data compression. The example SOC in Figure 2.3 is illustrated in Figure 6.1, which also shows the ATE memory organization with test stimuli and expected responses when test-data compression is not used. The cores are scan tested and the scanned elements at each core are formed to wrapper chains that are connected to TAM wires. The TAM wires are connected to the ATE and are used to transport test stimuli and produced responses to and from the cores. At test application, test stimuli are sent to the SOC and the produced responses are sent to the ATE. The ATE compares the produced responses with the expected ones to determine if the chip is faulty.

For the case when test-data compression is used the test-architecture, illustrated in Figure 6.1, is extended to include a decoder, for the decompression of compressed tests, and a compactor for each core is used to compress the produced responses. The placement of the decoder and the compactors are illustrated in Figure 6.2. In this chapter we make use of a compactor free architecture proposed by Larsson and Persson [Lar07c]. The general idea is to store compressed test stimuli, compressed expected responses, and compressed test masks in the ATE, as illustrated in Figure 6.3. The mask is used to determine care bits in the test stimuli. The advantage by employing a mask is that the expected responses can be compressed in the same way as test stimuli. The compressed test stimuli, expected responses, and test masks are sent to the SOC under test and decompressed on the chip.

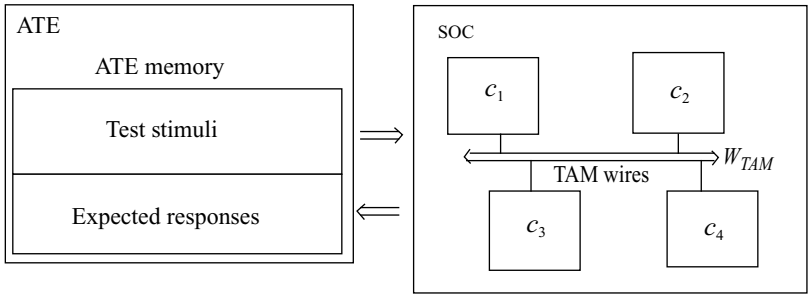


Figure 6.1: Traditional test-architecture and ATE memory organization when test-data compression is not used.

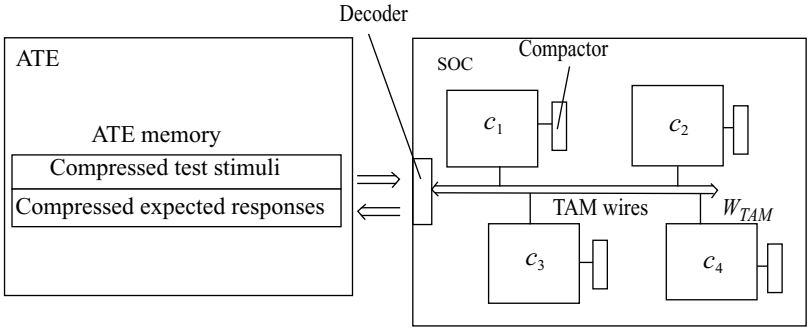


Figure 6.2: Test-architecture and ATE memory organization using stimuli test-data compression and response compaction.

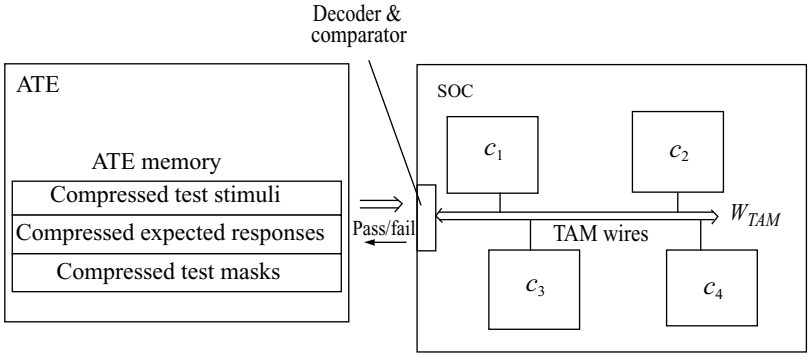


Figure 6.3: Test-architecture and ATE memory organization using stimuli and response compression [Lar07c].

Test evaluation is also performed on-chip using the comparator and a pass/fail signal is used to indicate the result of the test.

For test sharing, we make use of a similar test-architecture as described in Section 5.6. The test-architecture is illustrated in Figure 6.4 using core c_1 and c_2, which are connected to six TAM wires. Figure 6.4(a) shows a test-architecture, which does not make use of test sharing and Figure 6.4(b) shows a test-architecture when sharing is used. The tests T_1 and T_2 in Figure 6.5 consist of TS_1 and TS_2, and ER_1 and ER_2. By using a test mask (M_1, M_2 in Figure 6.5) for each test that marks the positions of each specified bit in the expected responses, it is possible to determine if the produced responses from the core are correct or not even in the presence of unspecified values, so-called unknowns. The latter is important since unknown bits in the produced responses are becoming more common with technology scaling.

A detailed description of test application using the given test-architecture is illustrated in Figure 6.6, which shows the connection of wrapper chain wr_1 to TAM wires TAM_1 and TAM_4 from Figure 6.4(a). We assume a decoder in

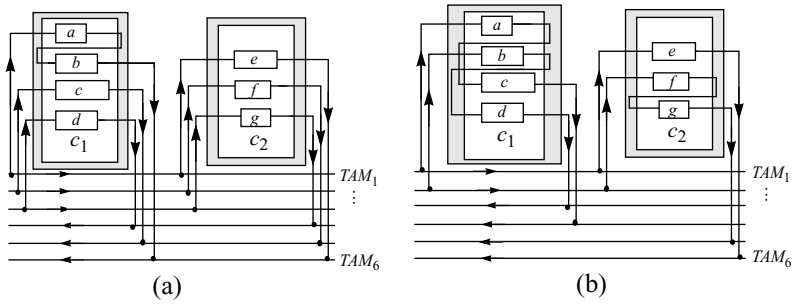


Figure 6.4: Test-architecture (a) without test sharing and (b) with test sharing.

Scan chain

T_1

	a	b	c	d
TS_1	0xx	xx11	xxxx1	xxxx
	0xx	xxxx	xxx00	xxx1
	xx1	xxx0	xx0xx	xxx0
ER_1	1xx	xxx1	xxxx0	xxxx
	11x	xxxx	xxxx1	xxxx
	xx0	xxxx	xxxxx	xxxx
M_1	100	0001	00001	0000
	110	0000	00001	0000
	001	0000	00000	0000

T_2

	e	f	g
TS_2	xx11	xx0	0x
	xx0x	xxx	1x
	xx10	1x1	xx
ER_2	xxx0	xx0	x1
	xxx1	x1x	xx
	1xxx	x0x	x0
M_2	0001	001	01
	0001	010	00
	1000	010	01

Figure 6.5: Initially given tests with don't-cares.

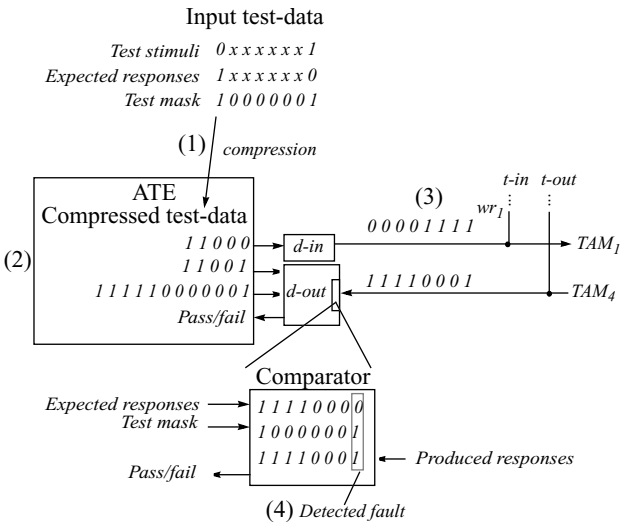


Figure 6.6: Fault detection using a comparator.

which for each TAM wire we have a decoder block. A decoder block can act as an input decoder *d-in* in the case the decoder block is configured to receive test stimuli. If the decoder block is configured as output decoder *d-out* it receives expected responses and a test mask. The test application, when test-data compression is used, is done as follows: First, the original test-data is compressed into code words, labelled as (1) in the figure, that are stored in the ATE memory (2). The test stimuli are then decompressed and applied to the wrapper chain wr_1 through the TAM wire TAM_1 (3). At the same time as the test response from the core is shifted out using TAM_4, the expected response and the mask are decompressed, and applied to the comparator, where they are used to evaluate the produced responses (4).

6.3 Test-Data Compression and Sharing

This section describes the test-data compression and the test sharing techniques used in this work. The relation between test-data compression and test sharing in terms of test-data volume is also described.

For the test-data compression, we have made use of the 9C technique [Teh05], described in Section 3.3. The test-data compression function *compress* takes a test T_i as input and generates a new compressed test T_k:

$$compress(T_i) \rightarrow T_k \tag{6.1}$$

For the test sharing, we make use of the *share* function described in Section 5.4.

The relation between test-data compression and test sharing is illustrated in Figure 6.7 using the test stimuli TS_1 and TS_2 from Figure 6.5. Using only test-data compression for the initially given TS_1 and TS_2, the total number of bits to be stored in the ATE memory is 33 (16 + 17). This is less than the 39 bits needed to store the shared and compressed test alternative. This is due to the reduced amount of don't-care bits in the shared test that, for this case, leads to a poor compression.

6.4 Test-Architecture Design and Test Scheduling

This section describes the proposed test-architecture design and test scheduling technique with test-data compression and test sharing. The underutilization of TAM wires for sequential scheduling and the proposed cuncurrent test scheduling technique are also described.

As described in Section 5.6, the cores that share a test will be connected to the same TAM wires for the test stimuli and in order to separate the different produced responses from different cores the produced responses from each core are transported on separate TAM wires. The number of wrapper chains w_i for a test or alternative test T_i depends on the number of cores z that share the test, and is given by Equation 5.3.

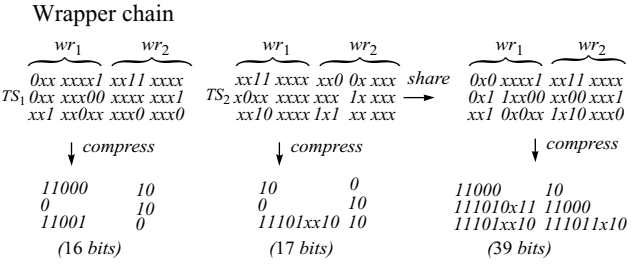


Figure 6.7: test sharing and test-data compression of tests.

In terms of test scheduling the simplest alternative is to apply the tests sequentially one after the other. In that case concurrent test application is only achieved by sharing one test between several cores. A disadvantage with a sequential approach is the potential underutilization of the TAM wires. Such underutilization occurs if the number of wrapper chains that a test uses is smaller than the TAM width. A small number of wrapper chains is used when a core has a small number of scan chains or if the scan chains are unbalanced, i.e., have a large difference in length, which may lead to few balanced wrapper chains [Iye01a]. By allowing multiple tests to be transported and applied concurrently, the TAM will be utilized more efficiently and the test application time can be reduced.

The underutilization of the TAM and the reduction of the test application time are illustrated using a small example presented in Figure 6.8 using core c_1 and c_2. In the example we assume that no test-data compression nor test sharing is used. The width of the TAM W_{TAM} is set to 8 and the TAM wires are connected with an ATE with an operating frequency f_{ATE} of 100MHZ. The number of wrapper chains that a test uses is determined such that half of the TAM wires are used for transportation of test stimuli and the second half for the produced responses (Equation 5.3). Core c_1 has 4 scan chains that are grouped into 4 wrapper chains and will occupy the full bandwidth of the TAM, however, c_2 has only 3 scan chains which means that only 6 of the TAM wires will be used when T_2 is transported, the other 2 TAM wires will not be used.

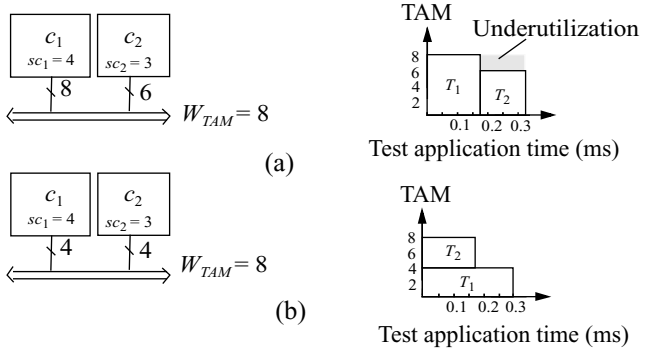


Figure 6.8: Motivational example (a) sequential test scheduling and (b) concurrent test scheduling.

In Figure 6.8(a) sequential test scheduling is used and the underutilization of the TAM is illustrated. The test application time τ_{tot} for the system using sequential test scheduling will be equal to 0.33ms. A better utilization of the TAM is illustrated in Figure 6.8(b) where the scan chains of core c_1 in this case are grouped into 2 wrapper chains. By reducing the number of wrapper chains the test application time of T_1 will be longer, however, the test application time τ_{tot} for the test schedule will be reduced from 0.33ms to 0.30ms since T_1 now can be scheduled at the same time as T_2.

The timing of the test schedule is illustrated in Figure 6.9. The tests are stored in the ATE memory and the control signals, *Comp* and *Share*, are used to determine the operation of the decoder. For example, if a test is not compressed (*Comp* = 0) the decoder is bypassed. In the example in Figure 6.9, the test stimuli sequence ts_{ij} is coded using three code words cw_1, cw_2, and cw_3. The code words are transported from the ATE to the decoder using the operating frequency f_{ATE} and the decompressed stimuli are transported and applied to the wrapper chains using the scan frequency f_{scan}.

The decompressed expected responses and mask must be synchronized with produced responses such that the comparator receives the correct sequences at correct time. We have two cases; when test-data compression is not used and when test-data compression is used. In the case when test-data compression is not used, the test-data is arranged such that the expected responses and masks are placed after the test stimuli (according to the length of the wrapper chains) in the ATE such that expected responses and masks

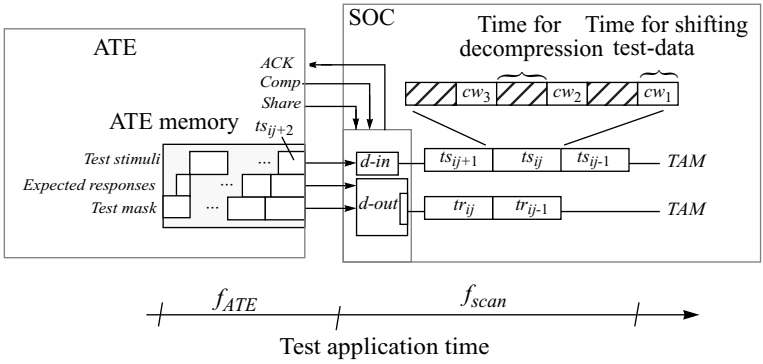


Figure 6.9: Test application using test-data compression.

arrive to the comparator when produced responses are ready. In the case when test-data compression is used, decompression takes different time depending on code words. To reduce the complexity of the synchronization between the test stimuli and expected responses and masks we assume the longest decompression time for each code word.

The synchronization when test-data compression is used is solved by applying test stimuli with a scan frequency f_{scan} that is lower than the operating frequency of the ATE, f_{ATE}. The value of f_{scan} is calculated using a constant, 9*CConst*, which is multiplied with the value of f_{ATE}. The value of 9*CConst* is given by the number of bits that each codeword contains (K=8) [Teh05], which is divided by he maximum number of clock cycles needed to apply the longest codeword that the 9C coding uses (12+8) [Teh05]. When a test that is not compressed is applied, the decoder is bypassed and the scan frequency will be the same as the ATE frequency. The value of f_{scan} is then given as follows:

$$f_{scan} = \begin{cases} f_{ATE}, & \text{when not using compression} \\ 9CConst \times f_{ATE}, & \text{when using compression} \end{cases} \quad (6.2)$$

The scan frequency f_{scan} is used to calculate the test application time $\tau_i(w)$ for a test T_i used to test a core i with w wrapper chains as follows:

$$\tau_i(w) = ((1 + \max\{si, so\}) \times l + \min\{si, so\}) / f_{scan}, \quad (6.3)$$

where *si* and *so* are the length of the longest wrapper scan-in and scan-out chain of core *i* with *w* wrapper chains, respectively and *l* is the number of test patterns. The test application time τ_{tot} for a system with *q* tests is given by Equation 4.2.

6.5 Problem Formulation

Given is a system consisting of a number of cores as described in Section 4.1 In addition, for each core c_i the following is given:

- $T_i = \{TS_i, ER_i, M_i\}$ - an initially given dedicated test consisting of test stimuli TS_i, expected responses ER_i, and test masks M_i,
- $TS_i = (ts_{i1}, \ldots, ts_{il})$ - a sequence of l test stimuli patterns, where ts_{ik} consists of $nff_i + wi_i$ bits and each bit can be 0, 1, or *x*,

- $ER_i = (er_{i1}, ..., er_{il})$- a sequence of l expected response patterns, where er_{ik} consists of $nff_i + wo_i$ bits and each bit can be 0, 1, or x,
- $M_i = (m_{i1}, ..., m_{il})$- a sequence of l mask patterns, where m_{ik} consists of $nff_i + wo_i$ bits and each bit can be 0 or 1. 1 indicates that the corresponding bit in the produced response is a care bit and should be checked with the expected response otherwise it is a don't-care bit and should be masked.

Also given for the system is the number of TAM wires, W_{TAM}.

For the ATE the following is given:

- μ_{ATE} - the number of bits that can be stored in the ATE memory,
- f_{ATE} - the clock frequency of the ATE.

Further, we assume that the *share* and *compress* functions, described in Section 5.4 and Section 6.3, respectively, have been used to generate a number of test alternatives per core.

The *share* and *compress* functions are used to generate new tests that are added to the list of alternative tests. Which test alternative can be used to test which core is explained using two initially given dedicated tests T_1 and T_2. The alternative tests are presented in Table 6.1. Column 1 lists the alternative tests and Column 2 and 3 list which core is tested by each test (marked as *X* in the table). For example, core c_1 can be tested using T_1, T_3, T_5, or T_6 (one test is sufficient in our approach). Column 4 lists the function(s) used to generate the test.

In order to solve the test selection problem a number of q possible alternative tests is generated for the system using the initially given dedicated tests. To illustrate the computational complexity of the test selection problem, we consider a system consisting of N cores. The number of tests generated using the dedicated given tests and the *share* function equals the sum of the number of possible k-subsets (where $k = \{1, 2, ..., N\}$) of a set of size N. For

Table 6.1: Test alternatives per core

Alternative test	Core c_1	Core c_2	Note
T_1	X		*Initially given*
T_2		X	*Initially given*
T_3	X		*compress(T_1)*
T_4		X	*compress(T_2)*
T_5	X	X	*share(T_1,T_2)*
T_6	X	X	*compress(share(T_1,T_2))*

example, a system consisting of 2 cores ($N = 2$) has the following k-subsets of cores: {1}, {2}, and {1, 2}, i.e., as each core has one dedicated test there are three different alternative tests. Since each such alternative test can be either compressed or not compressed the sum is multiplied by two. Hence, the number of possible test alternatives for two cores is equal to six. The value of q is then given as:

$$q = \left(\sum_{k=1}^{N} \binom{N}{k} \right) \times 2 \tag{6.4}$$

Given the above, our problem is to select one test alternative for each core c_i, determine the test-architecture (form the wrapper chains and determine TAM wire usage), and start time t_i such that the test application time τ_{tot} is minimized without exceeding the memory constraint μ_{ATE}.

6.6 Proposed Algorithm

This section describes the search space reduction using a *Maximum Share Ratio* (*MSR)* constant. This is followed by a description of the solution to the test selection, test-architecture design and test scheduling problem, first using CLP and second, using a Tabu search-based algorithm. For the CLP and the Tabu search-based algorithm, the *Design_wrapper* algorithm, described in Section 3.1.1, and the *share* and *compress* functions are used to generate the test application time for various number of wrapper chains, which are stored in a look-up table used as input to the algorithm.

In order to avoid excessively large optimization times due to the large number of possible test alternatives q, we limit the number of alternative tests for the system by restricting the number of possible permutations for the test sharing. In this work we restrict the generation of new tests by only considering possible 2-subset combinations during the sharing of tests. In addition, we do not consider those permutations that have minor effect on the ATE memory requirement. Only those tests that have similar size in term of number of sequences and scanned elements are shared. We, therefore, define *MSR* as follows:

$$MSR = 100 - \left(\frac{\max\{\mu_i + \mu_j\}}{\mu_i + \mu_i} \right) \times 100, \tag{6.5}$$

where μ_i is given as the number of bits in the test T_i (stimuli and expected responses) and μ_j is given as the number of bits in the test T_j, which are considered during the test sharing. This ratio is further illustrated by the example presented in Figure 6.10, where *MSR* is calculated for two different combinations of tests. Merging test T_i ($\mu_i = 100$) with T_j ($\mu_i = 20$) will lead to a maximum decrease of only 17% of the memory, while merging T_i with T_k ($\mu_i = 90$) potentially reduces the size with 47%. By setting a limit on the *MSR* during the pre-process stage it is possible to avoid those alternatives that have little possibility to be part of the optimal solution and therefore will not be explored during the optimization process.

6.6.1 Constraint Logic Programming Modelling

This section describes the solution to the test selection, test-architecture design and test scheduling problem using CLP. Since CLP uses an exhaustive search approach we restrict the CLP formulation to only consider sequential scheduling of tests.

The CLP-tool CHIP [Cos96], [Hen91] has been used for the implementation and the built-in predicates *labeling* and *min_max* are used for the enumeration and search for the optimal solution. A short description of the program is depicted in Figure 6.11. The variables such as the test application time τ_{tot} and used memory μ_{tot} are defined (line 2–6) and two new predicates, *sum_test_time* and *sum_test_mem* (line 8–9), have been implemented that calculate the test application time and the required memory for a specific solution. In line 8 the constraint expressing that the memory used is less than the size of the ATE memory is defined, and finally the optimization is done by using *labeling* for the enumeration inside the *min_max* predicate (line 13).

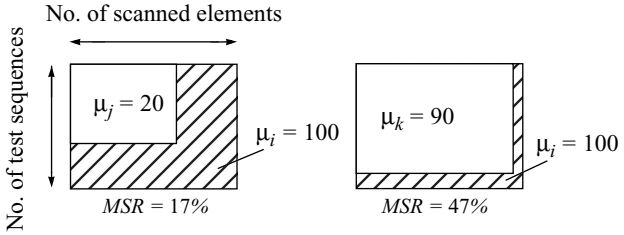


Figure 6.10: Maximum Share Ratio for different tests.

```
1  run:-
2     // Define variables and get input data
3     τ_tot::0..100000,
4     μ_tot::0..100000,
5     get_max_mem(μ_ATE),
6     get_input_tests(InputTests),
7     get_input_cores(InputCores),
8     // Set up constraints
9     sum_test_time(InputTests, InputCores, τ_tot),
10    sum_test_mem(InputTests, InputCores, μ_tot:),
11    μ_tot: #<=μ_ATE,
12    //Search for the optimal solution
13    min_max((labeling(InputCores)), τ_tot).
```

Figure 6.11: CLP formulation in CHIP for test application time minimization.

6.6.2 A Tabu Search-based Algorithm

The restriction of the CLP to only consider sequential scheduling of tests can be relaxed by replacing CLP with a heuristic. For this purpose we solve the problem using a Tabu search-based technique.

An overview of the proposed Tabu search-based algorithm is presented in Figure 6.12. The inputs are the cores $\{c_1, c_2, ..., c_N\}$, the test alternatives $\{T_1, T_2, ..., T_q\}$, the TAM width W_{TAM}, and the ATE memory μ_{ATE} constraint. The produced outputs are the test-architecture, a test schedule, and the test application time τ_{tot}.

The outer loop is used to generate a new, diversified solution in order to escape from local minima. The diversified solution is passed on to the inner loop where the search for a minimal test application time is done. For each iteration of the inner loop, test scheduling and TAM wire assignment using a Bottom-Left-Decreasing (BLD) algorithm [Lesh04] is performed. When a termination condition, defined later in this section, is met, the Tabu search-based heuristic is stopped and the test-architecture and test schedule with a minimized test application time are returned as output.

The pseudo code for the Tabu search-based algorithm is presented in Figure 6.13 and Figure 6.14. Figure 6.13 presents the initial solution and the inner loop and in Figure 6.14 the outer loop is presented. The initial solution is created by using the dedicated test for the testing of each core (line 5–6 in Figure 6.13). Since no compression is used, this initial solution is likely to violate the ATE memory limit. In such a case, the initial solution is modified

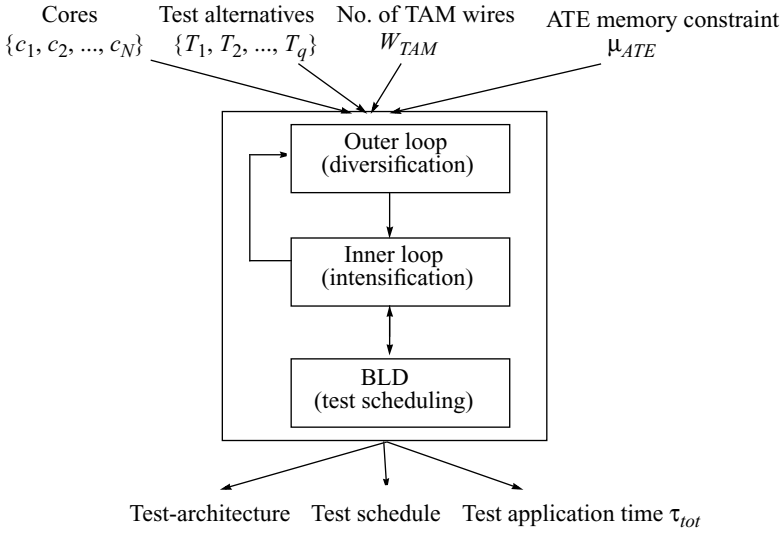


Figure 6.12: Overview of the Tabu search-based algorithm.

by randomly changing some of the dedicated tests to a compressed test. This change is done using a random function that is repeated for a maximum given number *MAX_ITERATIONS* of times (line 8–12). When a valid initial solution has been found, the Tabu search will continue the search for a better solution by exploring the neighborhood (line 15–48).

Below follows a description of the neighborhood and the search for improved solutions. Each core is assigned with a list of tests, *core_test_list* that consists of test alternatives that can be used to test the core. A solution consists of N tests where each position in the solution is associated with a specific core and contains one test from that core's *core_test_list*. In the heuristic, the neighbourhood is determined by the possible changes of test for each core and is defined as follows: A test $T_i(k)$, where k is used to denote the position of the test in the *core_test_list*, can be replaced with either the test at position k - 1 or the test at position k + 1.

The neighbourhood is illustrated in Figure 6.15 using the example system in Figure 6.8. In Figure 6.15 the current solution contains T_1 and T_4, which are used to test c_1 and c_2, respectively. Figure 6.15 also shows the *core_test_list* for c_2. The *core_test_list* shows that the possible moves for the test T_4 are T_2 (k - 1) and T_5 (k + 1). The same principle is applied for all positions in the

```
1  Procedure TabuSearchBLD (Part one)
2    //Inputs: Cores, test alternatives, TAM width, and ATE memory constraint
3    //Outputs: Test-architecture, test schedule, test application time
4    //Generate initial solution
5    For each core c_i
6      solution ∪ {T_i}
7    iterations = 0
8    While MemoryExceeded(solution)
9      GenRandomCompressSolution(solution)
10     iterations++
11     If iterations > MAX_ITERATIONS
12       Return "No solution found"
13   best_solution = solution
14   // Start the inner loop
15   Start:
16   moves[] = GenNeighborhoodSolutions(solution)
17   CalculateDeltaTATAndSort(solution, moves)
18   For each move m_j
19     delta_tat = GetDeltaTAT(m_j)
20     If delta_tat < 0
21       new_solution = GetNewSolution(solution, m_j)
22       If MemoryExceeded(new_solution)
23         delta_tat = delta_tat + mem_penalty
24       If m_j not in tabu_list or GetTATBLD(new_solution) < GetTATBLD(best_solution)
25         IncrFrequeny(m_j)
26         solution = new_solution
27         Goto Accept
28   For each move m_j
29     UpdateMoveTAT(m_j, GetFrequency(m_j))
30   For each move m_j
31     new_solution = GetNewSolution(solution, m_j)
32     If MemoryExceeded(new_solution)
33       delta_tat = delta_tat + mem_penalty
34     If m_j not in tabu_list or GetTATBLD(new_solution)< GetTATBLD(best_solution)
35       IncrFrequency(m_j)
36       solution = new_solution
37       Goto Accept
38   m_1 = GetMoveFromTabuList(tabu_list)
39   new_solution = GetNewSolution(solution, m_1)
40   IncrFrequency(m_1)
41   Accept:
42   If GetTATBLD(solution) <GetTATBLD(best_solution) and !MemoryExceeded(solution)
43     iterations_without_better = 0
44     best_solution = solution
45   Else
46     iterations_without_better++
47   If iterations_without_better < MAX_INNER_LOOP
48     Goto Start
```

Figure 6.13: Tabu search heuristic (initial solution and inner loop).

```
1  Procedure TabuSearchBLD (Part two)
2    //Outer loop (diversification)
3    If restarts < MAX_OUTER_LOOP
4      restarts++
5    iterations_without_better = 0
6    If CyclesDetected(solution)
7      Goto Stop
8    //Generate diversified solution
9    no_to_change = n*divers_ratio/100
10   While(divers_count < no_to_change)
11     GenDiversifiedSolution(solution, divers_count)
12     divers_count++
13   Goto Start
14   Stop:
15   Return best_solution
```

Figure 6.14: Tabu search heuristic (outer loop).

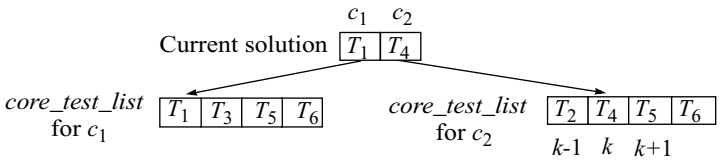


Figure 6.15: Neighborhood definition.

current solution, which means that each test in the current solution will be associated with two possible moves.

The reason for using this neighbourhood is that it will lead to small changes of the current solution, hence, the search will continue in the same region of the solution space (intensification). For example, one shared test is likely to be changed to another shared test. An alternative neighbourhood could be to randomly select a test. Such random move, however, will lead to a bigger change of the current solution and could result in a move to a different region of the solution space.

When a move has been applied, it is marked as a tabu to avoid cycling. A move will be marked as tabu for a constant *MAX_TABUS* iterations, which is determined using experiments. For each move, a *delta_tat* value is calculated (line 17) that corresponds to the decrease of the test application time when that move is applied to the current solution. The moves are also sorted decreasing according to their *delta_tat* value (line 17). The *delta_tat* value is calculated

using a BLD algorithm, described later in this section. The BLD algorithm is used to schedule the selected tests and assign TAM wires to each core such that the test application time is minimized. If *delta_tat* is less than zero a new solution is generated (line 21).

In order to make the search efficient, solutions that violate the ATE memory constraint can be accepted. If such a move is accepted it is penalized using a *mem_penalty* parameter (line 23). The *mem_penalty* parameter is defined as follows:

$$mem\_\ penalty\ =\ MEM\_\ CONST \times \tau_{tot} \tag{6.6}$$

where *MEM_CONST* is an experimentally determined parameter.

If the move is not in the tabu-list or if the move would generate a solution better than the best solution found so far, the current solution is assigned the new solution (line 26). In such case the frequency, i.e., the number of times the move has been applied, is increased, and the solution is accepted (line 41–48).

If no improving move (the *delta_tat* is more than zero) is found the search continues by recalculating the *delta_tat* considering the frequency of the moves. In this step, moves with a high frequency are considered to likely be part of a good solution, hence, they will get priority when the search continues (line 30–37). If no improving move can be found, the move that leads to the smallest increase of the test application time is assigned to the current solution (line 38), which means that an uphill move will be applied. The inner loop is stopped if no improving move is found for *MAX_INNER_LOOP* consecutive tries.

While the inner loop is used to search for a solution by making small changes to the current solution, the outer loop, presented in Figure 6.14, will diversify the search by generating a new solution, which is dramatically different from the current solution. This diversification will force the search into a new region of the solution space that is not reachable using the inner loop. The outer loop is executed for a maximum of *MAX_OUTER_LOOP* iterations (line 3 in Figure 6.14). The value of *MAX_OUTER_LOOP* is defined as follows:

$$MAX\_\ OUTER\_\ LOOP\ =\ \alpha + \beta \times N \tag{6.7}$$

where α and β are tuned experimentally. The reason for not having a fixed value of *MAX_OUTER_LOOP* is to allow the search to be executed for longer time for large examples.

The diversification is done by randomly changing a number of the tests in the current solution (line 9–12). The number of tests that are changed is given by a variable *divers_ratio*, which ranges from 0 to 100. A *divers_ratio* = 100 means that all tests in the solution will be replaced, *divers_ratio* = 50 means that 50% of the tests will be replaced. The *divers_ratio* will have a large value in the beginning, which means that solutions from different regions of the solution space will be generated. The diversified solution is then improved by using the inner loop. The outer loop also has a mechanism to detect if a cycle has occurred (line 6). If a cycle is detected in the outer loop, the algorithm is stopped and the best solution found and the test application time τ_{tot} are returned (line 15).

The selected tests are then scheduled and assigned to TAM wires according to a BLD algorithm, which has been implemented in the *GetTATBLD* function used to acquire the test application time for a solution. The pseudo code for the BLD algorithm is presented in Figure 6.16. First, the tests for the solution, which is given as input to the algorithm, are sorted decreasingly according to their TAM usage (line 5 in Figure 6.16). The tests that occupy the most of the TAM bandwidth will be placed first. At this point, all redundant, shared, tests are removed leaving *n_tests* distinct tests to be scheduled. For example, a shared test will be listed for all cores that share the test. However, only one instance of the shared test needs to be scheduled.

Each test is then scheduled as early as possible while leaving as much empty TAM wires as possible. The first test will be selected and scheduled at time zero. If the TAM wires are not fully utilized a second loop is used to

```
1  Procedure GetTATBLD(solution)
2    //Calculate the test application time using BLD scheduling
3    τtot = 0
4    used_TAM = 0
5    tests[n_tests] = SortTestsTAM(solution)
6    For each test Ti in tests
7      If NotScheduled(Ti)
8        ScheduleTestAtBottomLeft(Ti)
9        Update(τtot, used_tam)
10     While(used_tam < WTAM)
11       Tj = search_test(tests, WTAM- used_tam)
12       ScheduleTestAtBottomLeft(Tj)
13       Update(τtot, used_tam)
14   return τtot
```

Figure 6.16: BLD scheduling algorithm.

search for a test, which can be scheduled at the same time. Once a test has been scheduled the test application time is updated (line 13). The search is repeated until the bandwidth is fully utilized or no other test can be scheduled. When all tests have been scheduled the test application time τ_{tot} is returned.

6.7 Experimental Results

In this section the significance of integrating in one framework both test sharing and compression with test-architecture design and scheduling is demonstrated by experiments. Two sets of experiments have been performed. First, to show the significance of integrating test sharing and compression, the proposed CLP formulation of the problem described in Section 6.6.1 is used. Second, to show the importance of concurrent test scheduling, we use the proposed Tabu search algorithm described in Section 6.6.2.

For the experiments we have used the following four ITC'02 benchmark designs: *d695*, *g1023*, *p34395*, and *p93791* [Mar02], consisting of 10, 14, 19, and 32 cores, respectively. The input characteristics for the designs are collected in Table 6.2 where Column 1 contains the name of the design and Column 2 the number of tests given as input. Column 3 lists the amount of memory required to store the original dedicated test stimuli and expected responses for the given tests. The last column, Column 4, contains the number of TAM wires, which is specified by us.

For the *d695* design the test stimuli and expected responses (with don't-care bits marked) are given [Kaj01]. We have generated test stimuli and expected responses for designs *g1023*, *p34395*, and *p93791* such that the amount of don't-care bits is 95% [Lar06b]. We assume, in these experiments that the designs are tested using an ATE with a frequency f_{ATE} of 100MHz and after running extensive experiments, the *MSR* threshold, described in Section 6.6, is set to 35%.

Table 6.2: Benchmark Characteristics

Design	No. of input tests	Memory requirement (kbits)	No. of TAM wires (W_{TAM})
d695	10	3398	48
g1023	14	4789	60
p34392	19	125332	60
p93791	32	1122802	60

In the first set of experiments, the CLP formulation in Section 6.6.1 is used. The test application time is minimized using CLP under ATE memory constraints using the following four techniques: no compression and no sharing (NC, NS), only compression (C), only sharing (S) and both compression and sharing (C, S). The memory constraint M_{ATE} has been determined by multiplying the amount of memory required for each design (given in Table 6.2 Column 3) with a constant, *MConst.* In total, three experiments have been performed each with different ATE memory constraint, *MConst* = 1*, MConst* = 2/3, and *MConst* = 1*/3.*

The experimental results for the CLP approach (using sequential test scheduling) are collected in Table 6.3. Column 1 lists the different designs and Column 2 the different techniques for each design respectively. Column 3 lists the total number of test alternatives considered during the optimization. The following columns, 4 to 12, list the memory constraint, the test application time, and the CPU-time for each of the three experiments, respectively.

The results obtained using the integrated approach, with both compression and sharing (denoted C, S in Table 6.3), are compared against the following three techniques. First, using no compression and no sharing (NC, NS), i.e., only the dedicated, initially given, tests are used to test the system. Second, using only compression (C) and third, using only sharing (S).

Without using either sharing nor compression, results are only obtained when the ATE memory is large enough as in Experiment 1 (*MConst* = 1). When reducing the ATE memory size as in Experiment 2 (*MConst* = 2/3), sharing only is sufficient to decrease the amount of memory used for the design *d695* and *p34392*, for the design *g1023* the compression technique must be applied to obtain a solution. The compression technique is required for all three designs in Experiment 3 (*MConst* = 1/3) since only sharing does not decrease the required memory sufficiently.

When using compression the test size is reduced and less ATE memory is used but the test application time is increased due to the slower scan frequency (Equation 6.2). For all three experiments, the results show a decrease in the test application time when sharing is used. Experiment 3 shows that, by using both sharing and compression, it is possible to considerably reduce the test application time when using a small ATE memory. When using a large ATE memory such that sharing only (S) is able to obtain a solution our method is not able to further decrease the test application time, however, our proposed

Table 6.3: Test Application Time for Different ATE Memory Constraints using CLP

Design	Technique (No compression (NC), No sharing (NS), Compression (C), Sharing (S))	No. of test alternatives	Experiment 1 (MConst = 1)			Experiment 2 (MConst = 2/3)			Experiment 3 (MConst = 1/3)		
			Memory constraint M_{ATE} (kbits)	Test application time τ_{tot} (ms)	CPU-time (s)	Memory constraint M_{ATE} (kbits)	Test application time τ_{tot} (ms)	CPU-time (s)	Memory constraint M_{ATE} (kbits)	test application time τ_{tot} (ms)	CPU-time (s)
d695	NC, NS	10	3398	0.49	0.2	2265	n.s.[a]	n.s.[a]	1132	n.s.[a]	n.s[a].
	C	20		0.49	0.2		0.82	0.8		1.22	0.3
	S	20		0.36	1.0		0.36	0.9		n.s.[a]	n.s.[a]
	C, S	40		0.36	1.0		0.36	0.9		0.44	1.0
g1023	NC, NS	14	4789	0.56	0.2	3193	n.s.[a]	n.s.[a]	1596	n.s.[a]	n.s.[a]
	C	28		0.56	0.3		0.90	1.6		1.28	0.5
	S	35		0.47	15.9		n.s.[a]	n.s.[a]		n.s.[a]	n.s.[a]
	C, S	70		0.47	51.9		0.55	29.8		0.94	40.4
p34392	NC, NS	19	125332	14.72	0.3	83551	n.s.[a]	n.s.[a]	41784	n.s.[a]	n.s.[a]
	C	38		14.72	216.2		23.68	204.4		33.30	46.9
	S	37		10.93	313.3		10.93	5.8		n.s.[a]	n.s.[a]
	C, S	74		10.93	2255.6		10.93	1902.8		16.43	437.7

a. No solution (n.s.)

integrated technique always produces solutions that are equal or better compared to when sharing or compression is used separately or when none of them is used.

The results also show the trade-off between the test application time obtained using the proposed approach, and the amount of optimization time needed. In general, the optimization time is increased using our approach since the complexity (the number of test alternatives) is increased.

In the second set of experiments we show the importance of concurrent test scheduling. The experimental results using the Tabu search heuristic are presented in Table 6.4. Column 1 lists the designs, Column 2 lists the number of test alternatives considered during the optimization, and Column 3 lists the ATE memory constraint. Column 4 to 9 lists the test application time and optimization time (CPU-time) for three different optimization strategies. Column 4 to 7 contain the results obtained using sequential test scheduling optimized using CLP and the proposed Tabu search respectively. Column 8 and 9 contain the results when concurrent test scheduling is used and optimized using Tabu search. The parameters used in the Tabu search heuristic have been determined, using extensive experiments, as follows: *MAX_TABUS* = 15, *MAX_INNER_LOOP* = 10, $\alpha = 50$, $\beta = 3$, and *MEM_CONST* = 0.4.

The results show that the proposed Tabu search generates solutions which are close to the optimal solution generated using CLP. For the design *p93791*, CLP was not able to find the optimal solution in reasonable time and therefore a time-out is used to terminate the algorithm. The timeout is set to 5 hours and when this time is reached the best solution found so far is reported. In the case when a small memory is used, CLP was not able to find any solution for *p93791* after 5 hours. On average, the test application time using Tabu search is only 8.2% longer than the optimal solution (the results from *p93791* is not included as CLP does not give a solution) and Tabu search requires much shorter optimization time for medium and large designs. Only for the smallest design, *d695*, the CLP outperforms Tabu search in terms of optimization time. The results also show an average of 15% decrease of the test application time when concurrent test scheduling is used, compared with using sequential test scheduling.

Table 6.4: Test Application Time using Proposed Tabu search-based Heuristic

Design	No. of test alternatives	Memory constraint M_{ATE} (kbit)	CLP (sequential test scheduling)		Tabu search heuristic (sequential test scheduling)		Tabu search heuristic (concurrent test scheduling)	
			Test application time τ_{tot} (ms)	CPU-time (s)	Test application time τ_{tot} (ms)	CPU-time (s)	Test application time τ_{tot} (ms)	CPU-time (s)
d695	40	1132	0.44	1.0	0.47	8.4	0.47	12.7
		2265	0.36	0.9	0.37	8.7	0.35	11.8
g1023	70	1596	0.94	40.4	1.07	18.9	0.81	29.6
		3193	0.55	29.8	0.63	18.4	0.42	17.2
p34392	74	41784	16.43	437.7	19.32	26.6	15.34	50.5
		83551	10.93	1902.8	10.95	45.5	9.56	73.2
p93791	214	374267	n.s.[a]	18000.0[b]	306.09	387.6	306.09	382.0
		748534	117.24	18000.0[b]	175.30	348.7	169.13	363.5

a. No solution (n.s.)
b. Terminated by time-out.

6.8 Conclusions

In this chapter we integrated test-data compression, test sharing, test-architecture design and test scheduling with the objective to minimize the test application time under ATE memory constraint. We assume a core-based system with given tests per module and we define a technique for test-data compression and test sharing to find the best test alternatives for the testing of each core such that the test application time is minimized for the system. The efficiency of our approach has been demonstrated with experiments on several ITC'02 designs. The experimental results show the importance of integrating both test-data compression and test sharing. Furthermore, experiments demonstrate that the proposed Tabu search-based algorithm is able to find solutions that are close to the optimal with much shorter optimization times than the CLP-based approach. The results also show that the test application time can be further decreased when concurrent test scheduling is used as opposed to sequential test scheduling.

Chapter 7
Compression Driven Test-Architecture Design and Scheduling

THIS CHAPTER PRESENTS a test-architecture design and test scheduling approach for SOCs that is based on core-level expansion of compressed test patterns. First, the proposed technique and the used test-architecture with test-data compression are introduced. Second, the analysis of test-data compression is described. Third, the problem is formulated in detail and the proposed algorithm used to solve the problem and a lower bound on test application time are described. Finally, experimental results are presented and conclusions are drawn.

7.1 Introduction

In this chapter, we present a co-optimization technique that reduces the SOC test application time (TAM width) by test-architecture design, scheduling, and decoder design.

As discussed in Chapter 3, several approaches have been developed [Iye05], [Gon04a], [Wang07]. Most approaches use only one decoder which is designed at the SOC-level, therefore, there is no way to trade-off the achieved

test-data compression with the test application time at core-level. Further, the related work do not provide any quantitative insights on the test-time reduction (at the SOC-level) derived from adding a decoder for any given embedded core.

In this chapter, the optimization of the wrapper and decoder designs for each core are integrated with the test-architecture design and the test scheduling at the SOC-level. Analysis of test-data compression shows that, for each core and its decoder, the test application time does not decrease monotonically with the increasing width of TAM at the decoder input or with the increasing number of wrapper chains at the decoder output. Therefore, there is a need to include the optimization of the wrapper and decoder designs for each core, in conjunction with the test-architecture design and the test scheduling at the SOC-level. A test-architecture design and test scheduling technique for SOCs that is based on core-level expansion of compressed test-data is proposed. The proposed approach leads to regular TAMs and it is able to leverage the large body of work that has been developed recently for TAM optimization and test scheduling. Two optimization problems are formulated: test application time minimization under a TAM width constraint and TAM width minimization under a test application time constraint

7.2 Test-Architecture

In this section the test-architecture using core-level expansion of compressed test patterns is described.

In the previous chapter, one decoder was used to decompress the test-data for all cores in the SOC. In this chapter we make use of a test-architecture with one decoder per core. Figure 7.1 shows the test-data compression architecture for a single wrapped core. The compressed test-data for the core, which is stored in the ATE, is sent via w TAM wires to the decoder at the core under test. The decoder takes at each clock cycle w^{TAM} input bits and expands them into w bits ($w^{TAM} < w$), which feed w wrapper chains. The scan chains and the wrapper input and wrapper output cells at the core are accessed using the w wrapper chains.

We use Selective Encoding [Wang05], described in Section 3.3.2, as a representative test-data compression method at the core-level. The decoder for a core is placed between its wrapper and the TAM as illustrated in Figure 7.1.

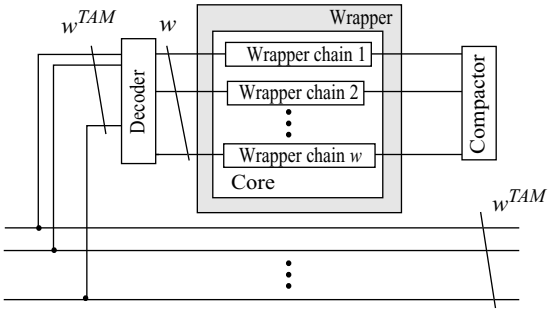


Figure 7.1: Test-data compression architecture for a wrapped core.

In this way, the number of inputs to the decoder is determined on the basis of not only the test-data compression achieved for the test, but also the test-data volume (and test-data compression achieved) for the other cores in the SOC. In this chapter, it is the test stimuli that are subject for the test-data compression, test-architecture design, and test scheduling. This work can be combined with a suitable method for test response compaction.

Selective Encoding makes use of on-chip decoders to expand the compressed test stimuli. For a non-modular SOC, the n compressed test stimuli bits are expanded to m scan chains where $m > n$; hence the scan chains are shorter and therefore the test application time is lower. For a core-based SOC, w^{TAM} input bits (TAM width) are expanded to w wrapper chains (see Figure 7.1) [Wang05]. For Selective Encoding, w^{TAM} is given as:

$$w^{TAM} = \lceil \log_2(w + 1) \rceil + 2 \qquad (7.1)$$

It is assumed that W_{TAM} wires are available. The test-architecture for a system is illustrated in Figure 7.2 using the SOC in Figure 2.3. In the example, each of the four cores has one decoder and one compactor. The inputs of each decoder are connected to the TAM wires. The W_{TAM} wires have been partitioned in two Test bus TAMs ($g = 2$). Cores c_1 and c_2 are connected to a TAM of width $w_1{}^{TAM}$ and c_3 and c_4 are connected to a TAM of width $w_2{}^{TAM}$. For core-based SOCs, it is favourable to place the decoder for a core near the core as it reduces routing cost as $w^{TAM} < w$ (see Figure 7.1).

The hardware cost for the selective encoding test-data compression technique is small. The synthesized controller part of the decoder contains only 5 FFs and 23 combinational gates. The other parts of the decoder are synthesized separately since they depend on w^{TAM} and w. For $w = 1024$ and

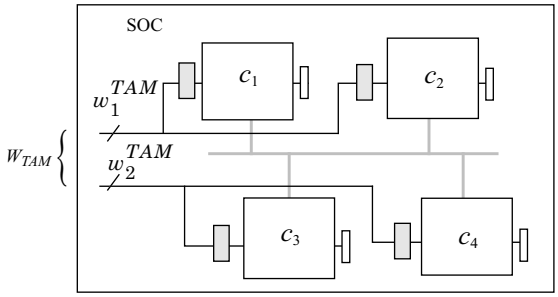


Figure 7.2: Example of a test-architecture using test-data compression for the SOC in Figure 2.3.

w^{TAM} = 13, the synthesized decoder contains 6409 gates and 1035 FFs. For larger than million-gate designs, this corresponds to a hardware cost overhead of less than 1% [Wang05].

7.3 Analysis of Test-Data Compression

A number of industrial cores were analyzed in respect to test application time. For the experiments, we have implemented the *Design_wrapper* algorithm described in Section 3.1.1, and for the filling of don't-care bits, we have made use of minimum transition fill. The scheme has been as follows: First, the wrapper chains have been formed for a given TAM width, and then, based on the wrapper design, the test stimuli bits are arranged accordingly and filled according to the minimum transition fill scheme. After that, the test stimuli are compressed.

For every core, we considered all possible values of w^{TAM} and w and we evaluated the test application time $\tau(w^{TAM}, w)$. We found a similar behaviour for all cores [Wang05]. We present the results for the industrial core named ckt-7. We present the analysis for two steps: (1) the test application time for various number of wrapper chains at a fixed TAM width and (2) the test application time at various TAM widths.

Figure 7.3 shows, for ckt-7, the test application time when the TAM width is fixed to 10 bits; hence w^{TAM} = 10 and w varies between 128 and 255 (Equation 7.1). It is expected that the test application time decreases as the number of wrapper chain increases; for example, the test application time at w

= 255 is lower than the test application time when $w = 128$. However, it is less obvious that the minimum test application time τ_{min} is obtained for 253 wrapper chains, and not for the maximum number of wrapper chains ($w = 255$) as expected. Figure 7.3 shows that when the goal is to find the lowest test application time for a core, it is not sufficient to simply assign the largest number of wrapper chains. In fact, the test application time varies much between the best w and the worst. Even though there is a global trend where the test application time is decreased with an increased number of wrapper chains there are many local regions where the test application time can increase with an increase in the number of wrapper chains.

In step two we have determined the value of w that gives the minimum test application time for each TAM width w^{TAM}. The minimum test application time for each TAM width is plotted in Figure 7.4. It is interesting to note that the test application time does not necessarily decrease as the TAM width (inputs to the decoder) increases. Actually, the test application time can increase as the TAM width increases. Figure 7.4 clearly shows that the test application time at TAM width 11 is lower than at TAM widths 12 and 13.

There are three reasons for the behavior highlighted in Figure 7.3 and Figure 7.4. First, the test data itself will be slightly different for different wrapper chain architectures due to the fact that extra bits (so-called idle bits) must be added to balance wrapper chains. Second, the reorganization of the test data for a different number of wrapper chains will impact the characteristics (i.e., the distributions of 1s, 0s, and xs) of the test-data, and therefore also the amount of test-data compression achieved. Third, in the best case, Selective Encoding can achieve test-data compression by a factor w/w^{TAM}, where w^{TAM} is given by Equation 7.1. This means that the achieved compression does not only depend on the test-data that is compressed but also on the number of TAM wires w^{TAM} and the number of wrapper chains w. For example, a test for a core with 127 wrapper chains ($w = 127$) will have a maximum test-data compression ratio of 14.1 (127/9). If the number of wrapper chains is increased to 128, the maximum test-data compression ratio reduced to 12.8 (128/10).

The analysis has shown that it is not straightforward to optimally design decoders at core-level to minimize the test application time. Furthermore, at

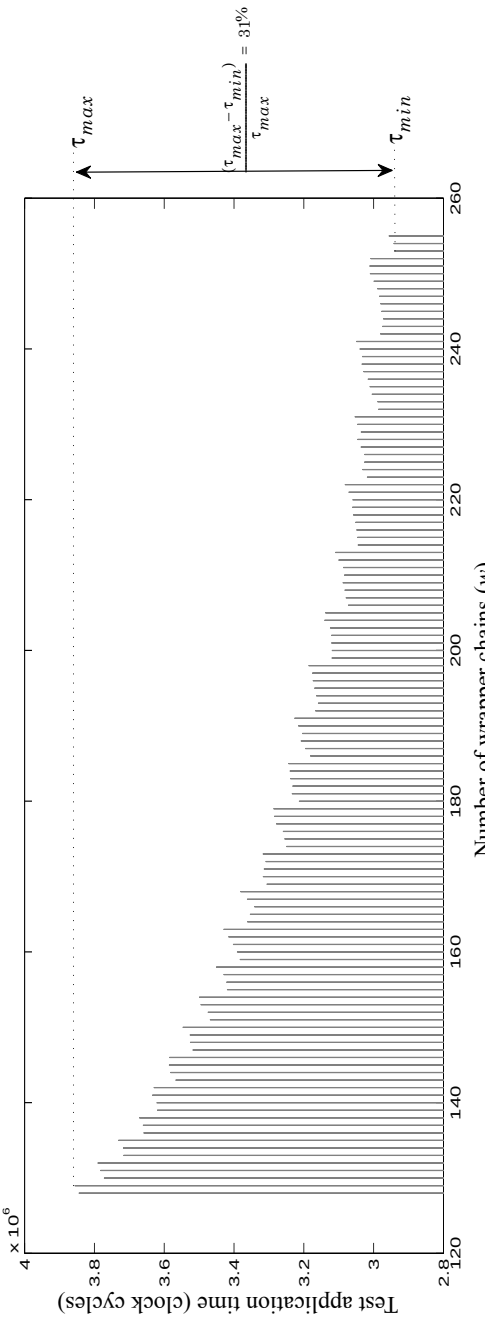


Figure 7.3: The non-monotonic variation of test application time with the number of wrapper chains at TAM width 10 for core ckt-7.

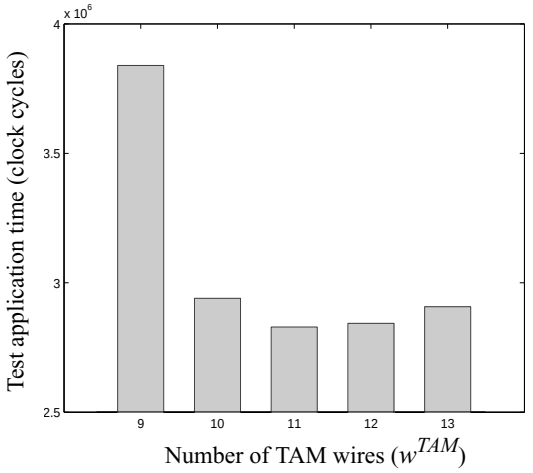


Figure 7.4: Lowest test application time at various TAM widths for ckt-7.

the SOC-level, the values of w^{TAM} and w for each core should be determined such that the overall test-data volume and corresponding test application time are at a minimum; hence all cores must be considered together, which makes the problem more complex.

7.4 Problem Formulation

Given is a system consisting of a number of cores as described in Section 4.1. The test application time τ_{tot} for a test schedule with q tests is given by Equation 4.2. In this chapter it is assumed that each core is tested by applying one test ($q = N$). The total test-data volume for a SOC with N cores is:

$$\mu_{tot} = \sum_{i=1}^{N} \mu_i(w^{TAM}, w) \tag{7.2}$$

We formulate two SOC-level optimization problems as follows:

- Ψ_{time} - For a given SOC, minimize the total test application time τ_{tot} without exceeding a TAM width constraint, W_{TAM}.

- Ψ_{TAM} - For a given SOC, minimize the total TAM width W_{tot} without exceeding a test application time constraint, τ_{max}.

For Ψ_{time}/Ψ_{TAM} the optimization objective is as follows: For a given SOC with a given TAM width W_{TAM}, partition the TAM into Test buses and determine each TAM's width, assign the cores to the TAMs, schedule the transportation of tests, and design the decoder for each core, such that the system's test application time/total TAM width is minimized.

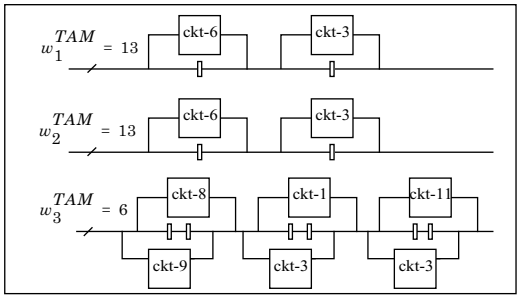
7.5 Proposed Algorithm

Both Ψ_{time} and Ψ_{TAM} are solved by considering test-architecture design and test scheduling, and involve the following goals: (1) partition the top-level (SOC) test-access wires into TAMs, (2) define the width w_j^{TAM} of each TAM j, (3) assign cores to TAMs, and (4) design a wrapper for each core. If test-data compression is taken into account, we also need to determine where to place the decoders as well as the widths of their inputs and output interfaces, i.e., the value of w^{TAM} at the decoder input and the value w at the decoder output.

Figure 7.5 shows three test-architecture alternatives for one industrial design. In Figure 7.5(a), the test-architecture and the test schedule are optimized but test-data compression is not used. The test application time is 32460913 clock cycles. Figure 7.5(b) shows an alternative scheme where test-architecture design and test scheduling are optimized assuming a decoder per TAM. The test application time is lowered to 10711883 clock cycles. However, the TAMs used to access the cores are extremely wide. Figure 7.5 (c) shows a scheme where a decoder is placed at each core. The test application time is the same as in Figure 7.5(b); however, the on-chip TAMs are much narrower. For modular SOCs it is, therefore, favourable to place the decoder for a core near the core as it reduces routing cost as $w^{TAM} < w$ (see Figure 7.1).

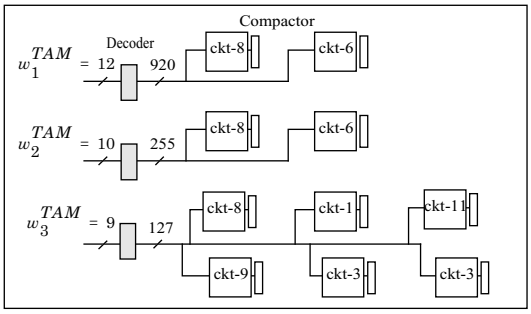
We use heuristic methods to solve the test application time and TAM width minimization problems. We are given a dedicated test for each and the SOC-level TAM width W_{TAM}. The basic heuristic procedure consists of four steps:

1. *Wrapper chain design*. For a given core and its test length, wrapper-design places the scanned elements (scan chains, input and output wrapper cells) into wrapper chains with the corresponding test application time. We have made use of the *Design_wrapper* algorithm (described in



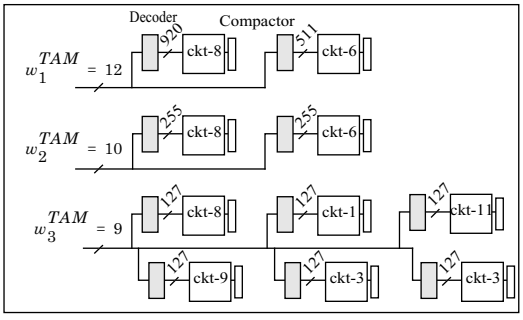
Test application time
τ_{tot} = 32460913

(a)



Test application time
τ_{tot} = 10711883

(b)



Test application time
τ_{tot} = 10711883

(b)

Figure 7.5: Test-architecture design and test application time (a) without using test-data compression, (b) with TDC and one decoder per TAM, and (c) with TDC and one decoder per core.

Section 3.1.1). For each core, we generate a number of wrapper design alternatives.

2. *Decoder design.* We make use of the Selective Encoding test-data compression scheme [Wang05]. As discussed above, we generate all alternatives for the decoder input/output mapping. For each combination of w_i^{TAM} and w, we have for a core i the test application time $\tau_i^c(w^{TAM}, w)$.
3. *Test-architecture design*. The input for this step is a TAM width (W_{TAM}) and the output is a TAM design. Figure 7.5(b) shows an example where the given TAM width, $W_{TAM} = 31$, has been partitioned into 3 TAMs ($g = 3$) of width $w_1^{TAM} = 12$, $w_2^{TAM} = 10$, and $w_3^{TAM} = 9$.
4. *Test scheduling.* Given a test-access architecture, we sort the cores based on test application time such that the core with longest test application time is first, and then we traverse the set of cores and assign a core to a TAM such that the resulting increase in test application time is the least. For each core, we have a lookup table from step 1 and 2 to find its test application time at various TAM widths, and we try each TAM width; hence the computational complexity for g TAMs and N cores is $O(N \times g)$.

7.6 Lower Bound on Test Application Time

A lower bound computation on test application time was defined by Goel and Marinissen [Goel03] for a given test-access architecture. The lower bound is calculated as follows: For each core, the optimal number of wrapper chains is selected such that the test application time is minimized. The test area, defined by the number of wrapper chains multiplied with the test application time, is summed for all cores. The lower bound test application time is obtained by dividing the sum of test areas with the available TAM width W_{TAM}.

We define a lower bound on test application time for a given TAM width W_{TAM} to also include test-data compression. For each core i, we select the optimal number of wrapper chains w and TAM width w^{TAM} and for each combination of w and w^{TAM} we select the optimal alternative between when test-data compression is used and without test-data compression. The lower bound test application time τ_{lb} is calculated as

$$\tau_{lb}(W_{TAM}) = \left\lceil \frac{\sum_{i=1}^{N} \min\left\{\forall w_i^{TAM}, w_i \ \min\left\{\tau_i^{nc}\left(w_i^{TAM}\right) \times w_i^{TAM}, \tau_i^{c}\left(w_i^{TAM}, w_i\right) \times w_i^{TAM}\right\}\right\}}{W_{TAM}} \right\rceil, \quad (7.3)$$

where N is the number of cores, $\tau_i^{nc}(w_i)$ is the test application time of core i without test-data compression, and $\tau_i^{c}(w_i^{TAM}, w_i)$ is the test application time when test-data compression is used. Similarly, the lower bound TAM width W_{lb} for a given test application time constraint τ_{max} is defined as

$$W_{lb}(\tau_{max}) = \left\lceil \frac{\sum_{i=1}^{N} min\left\{\forall w_i^{TAM}, w_i \ min\left\{\tau_i^{nc}\left(w_i^{TAM}\right) \times w_i^{TAM}, \tau_i^{c}\left(w_i^{TAM}, w_i\right) \times w_i^{TAM}\right\}\right\}}{\tau_{max}} \right\rceil \quad (7.4)$$

7.7 Experimental Results

In this section, we demonstrate the importance of integrating test-architecture design, test scheduling, and test-data compression at the SOC-level. We have implemented the technique described above and we have carried out extensive experiments.

We made use of benchmark design d695 and four designs, System1, System2, System3, and System4, which are composed of industrial cores described in detail in [Wang05]. The characteristics of each design are presented in Table 7.1. Column 1 lists the designs, Column 2 to Column 4 list the number of cores, gates, and FFs respectively. Column 5 lists the initially given test-data volume. The d695 SOC is composed of ISCAS'89 cores. These cores are fairly small. The number of scan-chains is in all cases less than 32, the number of test patterns is in the range 12 to 234, and the density of care bits is on average 66%. For the industrial designs, the number of scan

Table 7.1: Design characteristics

Design	No. of cores	No. of gates (1M)	No. of FFs (100k)	Initial given test-data volume V_i (Mbits)
d695	10	n.r.[a]	0.06	0.340
System1	10	7.13	5.39	6547
System2	40	16.74	10.74	7967
System3	80	40.24	26.46	20731
System4	120	48.58	33.64	24853

a. Not reported

cells ranges from 10000 to 110000, the test-data volume is in the order of tens of Gigabits, and the care-bit density is less than 5%.

We have carried out two sets of experiments. First, we minimize the test application time and the test-data volume for a given TAM width constraint, corresponding to problem Ψ_{time}, and second, we minimize the TAM width at a given test application time constraint, corresponding to problem Ψ_{TAM}. For each experiment, we compare test-architecture design and test scheduling with and without test-data compression. The results obtained using our proposed approach with test-data compression optimized at SOC-level is compared to the results obtained without test-data compression and to the results obtained with test-data compression optimized at core-level. The results obtained with test-data compression optimized at core-level is generated using the proposed test-architecture design and test scheduling optimization. For each core the optimal number of TAM wires is selected such that the test-data compression is optimized, (hence, no test-data compression optimization at SOC-level is performed).

For the first experiment, we compare test application time results with our approach, with test-data compression optimized at SOC-level τ_{SOC}, to those reported by Wang *et al*. [Wang07] and by Sehgal *et al*. [Seh04]. The comparisons for the d695 design are given in Table 7.2. Column 1 lists the TAM width constraint. Column 2 and Column 3 list the test application time $\tau_{[Wang07]}$ and $\tau_{[Seh04]}$, respectively. Column 4 shows the test application time obtained using our approach. The last two columns compare the test application time obtained in this work with those reported in prior work.

The results show that our approach produces better solutions than those reported by Sehgal *et al*. at a narrow TAM width constraint. For wider TAM width both our approach and the approach proposed by Sehgal *et al*. are able to find the solution with the minimal test application time corresponding to the test application time for the largest core in the design. For the results by Wang *et al*., no solutions are reported for narrow TAM width constraint. Our approach is able to produce a solution which is better than the one produced by the approach proposed by Wang *et al*. at TAM width 161. At TAM width 186 both the approach proposed by Wang *et al*. and our approach are able to find the solution with the minimal test application time corresponding to the test application time for the largest core in the design. Narrow TAM widths are likely for SOCs because of the need to carry our reduced pin-count testing (RPCT), especially for wafer test where contact fails must be minimized.

Table 7.2: Test application time with TAM width constraint for d695

TAM width constraint (W_{TAM})	[Wang07]	[Seh04]	Proposed		
	Test application time $\tau_{[Wang07]}$ (clock cycles)	Test application time $\tau_{[Seh04]}$ (clock cycles)	Test application time τ_{SOC} (clock cycles)	Comparison $\tau_{SOC}/\tau_{[Wang07]}$	Comparison $\tau_{SOC}/\tau_{[Seh04]}$
28	n.a.[a]	24701	16139	n.a.[a]	0.65
42	n.a.[a]	18564	12269	n.a.[a]	0.66
56	n.a.[a]	12192	11714	n.a.[a]	0.96
70	n.a.[a]	10432	10437	n.a.[a]	1.00
84	n.a.[a]	9869	9870	n.a.[a]	1.00
98	n.a.[a]	9869	9870	n.a.[a]	1.00
112	n.a.[a]	9869	9870	n.a.[a]	1.00
161	11049	n.a.[a]	9870	0.89	n.a.[a]
186	9870	n.a.[a]	9870	1.00	n.a.[a]

a. Not available

Moreover, larger TAM widths are undesirable because they lead to higher routing complexity. In fact, the approach proposed by Wang *et al*. suffers from high routing complexity because of the need to route the LFSR outputs to different cores in the SOC.

The results reported by Wang *et al*. and by Sehgal *et al*. are only available for comparison for the benchmark design d695. Sehgal *et al*. reports results for a design consisting of 9 industrial cores. This design, however, is not available for comparison.

Table 7.3 shows, at various TAM width constraints (W_{TAM}), the test application time and CPU-time (execution time to produce the solution). We compare the minimized test application time τ_{SOC} obtained when test-data compression is used and optimized at the SOC-level, against both the minimized test application time τ_{core} obtained with test-data compression optimized at core-level only and with the minimized test application time τ_{nc} without making use of test-data compression . The minimized test application time τ_{SOC} is also compared to the lower bound on test application time τ_{lb}.

The results in Table 7.3 are organized as follows. Column 1 lists the designs, Column 2 the TAM width constraint, and Column 3 the lower bound test application time. The following six columns lists the test application time and CPU-time for the case without test-data compression, with test-data compression optimized at core-level, and with test-data compression optimized at SOC-level, respectively. The last three columns highlight the comparisons. Column 10 lists the comparison between the test application time obtained using test-data compression optimized at SOC-level τ_{SOC} an

Table 7.3: Test application time results with TAM width constraint

Design	TAM width constraint W_{TAM}	Lower bound τ_{lb} (1000 clock cycles)	Without test-data compression		With test-data compression optimized at core-level		With test-data compression optimized at SOC-level		Time reduction factor		
			τ_{nc} (1000 clock cycles)	CPU-time (s)	τ_{core} (1000 clock cycles)	CPU-time (s)	τ_{SOC} (1000 clock cycles)	CPU-time (s)	τ_{lb}/τ_{SOC}	τ_{nc}/τ_{SOC}	τ_{core}/τ_{SOC}
d695	16	26.269	44.745	3.91	51.885	0.01	29.301	0.05	0.90	1.53	1.10
	32	13.134	22.538	65.11	25.654	0.47	14.252	0.64	0.92	1.58	0.99
	48	8.756	15.474	336.09	18.880	13.39	12.269	0.13	0.71	1.26	1.06
	64	6.567	12.034	16013.3	15.217	190.67	10.437	0.09	0.63	1.15	1.37
System1	16	20130	409351	0.29	28827	0.03	28323	0.12	0.71	14.45	1.02
	32	10065	204745	3.68	17991	1.62	10697	0.31	0.94	19.14	1.68
	48	6710	136532	0.81	10497	7.54	7580	0.01	0.89	18.01	1.38
	64	5033	102431	13.27	8250	123.28	7580	0.14	0.66	13.51	1.09
System2	16	30363	498618	1.27	44224	0.06	42684	0.50	0.71	11.68	1.04
	32	15181	249309	148.75	24844	2.14	15547	1.33	0.98	16.04	1.60
	48	10121	166306	1495.87	15244	51.80	11150	1.26	0.91	14.92	1.37
	64	7591	124789	57.96	13140	878.67	7855	10.28	0.97	15.89	1.67
System3	16	79461	1296889	2,52	116119	0.18	112766	1,03	0.70	11.50	1.03
	32	39730	648403	298,26	59181	6.42	41621	2,77	0.95	15.58	1.42
	48	26487	432238	3041,56	31327	157.20	29067	2,56	0.91	14.87	1.08
	64	19865	324320	10260,10	25582	2709.28	20737	20,72	0.96	15.64	1.23
System4	16	94460	1554672	32.35	137123	0.35	133277	1.56	0.71	11.66	1.03
	32	47230	777643	465.53	69236	12.85	49168	4.15	0.96	15.82	1.41
	48	31487	518413	9594.34	35080	318.66	34365	3.86	0.92	15.09	1.02
	64	23615	388931	16090.20	28242	5521.62	24687	30.93	0.96	15.75	1.14
								Average for all designs	0.85	12.44	1.24
								Average for industrial designs only	0.86	15.20	1.26

the lower bound test application time τ_{lb} (τ_{lb}/τ_{SOC}). Column 11 lists the comparison between the test application times obtained using test-data compression optimized at SOC-level and the test application time obtained when test-data compression is not used τ_{nc} (τ_{nc}/τ_{SOC}). Finally, Column 12 lists the comparison between the test application time obtained using test-data compression optimized at SOC-level and the test application time obtained using test-data compression optimized at core-level τ_{core} (τ_{core}/τ_{SOC}).

The achieved test-data volume results without using test-data compression μ_{nc}, with test-data compression optimized at core-level μ_{core}, and with test-data compression optimized at SOC-level μ_{SOC}, are presented in Table 7.4. The results in Table 7.4 are organized as follows. Column 1 lists the design, Column 2 the initially given test-data volume (μ_i), and Column 3 the TAM width constraint. Column 4 to Column 6 list the test-data volume obtained without test-data compression, with test-data compression optimized at core-level, and with test-data compression optimized at SOC-level, respectively. The last three columns contain the comparisons μ_i/μ_{SOC}, μ_{nc}/μ_{SOC}, and μ_{core}/μ_{SOC}.

The results from the first experiment show the importance of co-optimizing test-data compression, test-architecture design, and test scheduling for SOCs. On average, our approach with test-data compression optimized at SOC-level results in a 12.44x (15.20x) reduction in test application time compared when test-data compression is not used. (The results in parenthesis are for SOCs that are crafted from industrial cores only.) The corresponding reduction in test-data volume is on average 12.72x (15.52x). Our approach, when test-data compression is optimized at SOC-level results also in a 1.24x (1.26x) reduction in test application time compared when test-data compression optimized at core-level is used. For the test-data volume, our approach with test-data compression optimized at SOC-level, μ_{SOC}, produced results in the same range as the one obtained using test-data compression optimized at core-level, μ_{core}.

In the second experiment, we minimize the TAM width for a given test application time constraint. The results from the TAM width minimization experiment are presented in Table 7.5 and Table 7.6. Table 7.5 shows the TAM widths and CPU-times under various test application time constraints. The results in Table 7.5 are organized as follows. Column 1 lists the designs, Column 2 the test application time constraint (τ_{max}), and Column 3 the lower bound TAM width (W_{lb}). The following six columns list the TAM width and

Table 7.4: Test-data volume results with TAM width constraint

Design	Initial given test-data volume μ_i (Mbits)	TAM width constraint W_{TAM}	Without test-data compression	With test-data compression optimized at core-level	With test-data compression optimized at SOC-level	Data volume reduction factor		
			μ_{nc} (Mbits)	μ_{core} (Mbits)	μ_{SOC} (Mbits)	μ_i/μ_{SOC}	μ_{nc}/μ_{SOC}	μ_{core}/μ_{SOC}
d695	0.340	16	0.696	0.430	0.475	0.72	1.47	0.91
		32	0.699	0.430	0.425	0.80	1.65	1.01
		48	0.719	0.430	0.454	0.75	1.58	0.95
		64	0.723	0.430	0.589	0.58	1.23	0.73
System1	6547	16	6548	340	432	15.16	15.16	0.79
		32	6548	340	338	19.35	19.36	1.00
		48	6549	340	345	18.96	18.97	0.98
		64	6550	340	345	18.96	18.97	0.98
System2	7967	16	7973	509	667	11.95	11.96	0.76
		32	7973	509	493	16.16	16.17	1.03
		48	7974	509	529	15.05	15.07	0.96
		64	7978	509	496	16.07	16.09	1.03
System3	20731	16	20740	1337	1777	11.67	11.67	0.75
		32	20739	1337	1325	15.65	15.66	1.01
		48	20736	1337	1391	14.91	14.91	0.96
		64	20740	1337	1320	15.70	15.71	1.01
System4	24853	16	24861	1585	2112	11.77	11.77	0.75
		32	24871	1585	1564	15.89	15.90	1.01
		48	24864	1585	1641	15.14	15.15	0.97
		64	24873	1585	1567	15.86	15.87	1.01
					Average for all designs	12.55	12.72	0.93
					Average for industrial designs only	15.52	15.52	0.94

CPU-time for the case without test-data compression (W_{nc}), with test-data compression optimized at core-level (W_{core}), and with test-data compression optimized at SOC-level (W_{SOC}). The last three columns highlight the comparisons. Column 10 lists the comparison between the TAM width obtained using test-data compression optimized at SOC-level and the lower bound TAM width (W_{lb}/W_{SOC}). Column 11 lists the comparison between the TAM width obtained using test-data compression optimized at SOC-level and the TAM width obtained when test-data compression is not used (W_{nc}/W_{SOC}). Finally, Column 12 lists the comparison between the TAM width obtained using test-data compression optimized at SOC-level and the TAM width obtained using test-data compression optimized at core-level (W_{core}/W_{SOC}).

The results on test-data volumes are collected in Table 7.6. The columns in Table 7.6 are organized as follows: Column 1, 2, and 3, list the design, the initial given test-data volume, and the test application time constraint, respectively. Column 4 to Column 6 list the test-data volume obtained without test-data compression, with test-data compression optimized at core-level, and with test-data compression optimized at SOC-level, respectively. Column 7, 8, and 9, list the comparisons, μ_i/μ_{SOC}, μ_{nc}/μ_{SOC}, and μ_{core}/μ_{SOC}.

On average, our approach with test-data compression optimized at SOC-level provides a 2.33x (3.23x) reduction in TAM width compared to when no test-data compression is used, and a 1.40x (1.38x) reduction compared to when test-data compression optimized at core-level is used. The reduction in test-data volume is on average 9.21x (11.33x) compared to when no test-data compression is used. The test-data volume reduction when test-data compression optimized at SOC-level is used compared to the initial test-data volume is, for the second experiment, 3.12x (4.71x).

We further illustrate the importance of optimization of test-data compression at SOC-level compared to when the test-data compression is optimized at core-level in Figure 7.6, where we show the deviation from the lower bound test application time and from the lower bound TAM width for the largest design, System 4.

Figure 7.6(a) shows, at various TAM width constraints, the deviation from the lower bound test application time when test-data compression is optimized at core-level (τ_{core} - τ_{lb}) and when the test-data compression is optimized at SOC-level (τ_{SOC} - τ_{lb}). The deviations are large for narrow TAM widths since it is difficult to partition the narrow width in multiple TAMs. Therefore, many cores will be tested sequentially. This situation corresponds to a case when

Table 7.5: TAM width results with test application time constraint

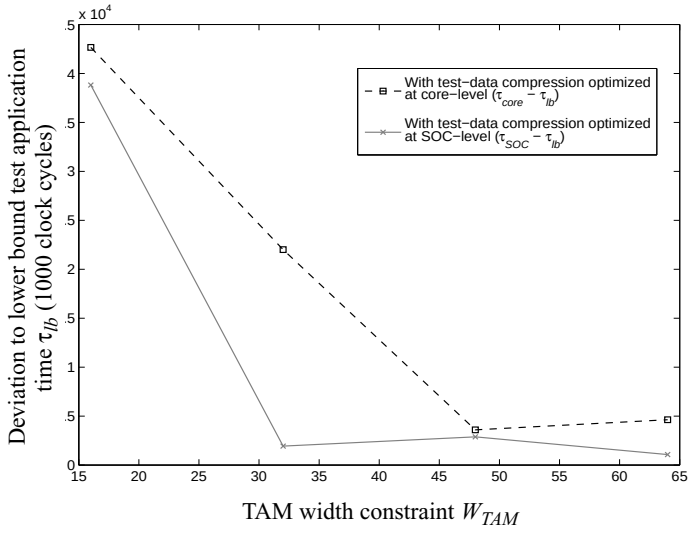
Design	Test application time constraint τ_{max} (1000 clock cycles)	Lower bound TAM width W_{lb}	Without compression		With test-data compression optimized at core-level		With test-data compression optimized at SOC-level		Width reduction factor		
			W_{nc}	CPU-time (s)	W_{core}	CPU-time (s)	W_{SOC}	CPU-time (s)	W_{lb}/W_{SOC}	W_{nc}/W_{SOC}	W_{core}/W_{SOC}
d695	70.00	7	11	2.37	11	0.02	7	0.81	1.00	1.57	1.57
	60.00	8	12	5.65	11	0.02	8	1.20	1.00	1.50	1.38
	50.00	9	15	18.52	18	0.21	11	5.29	0.82	1.36	1.64
	40.00	11	18	44.04	19	0.30	14	12.32	0.79	1.29	1.36
System1	500000	1	14	21.70	12	0.03	6	0.90	0.17	2.33	2.00
	100000	4	n.s.[a]	14400[b]	12	0.03	9	3.28	0.44	n.s.[a]	1.33
	50000	7	n.s.[a]	14400[b]	12	0.03	10	4.94	0.70	n.s.[a]	1.20
	25000	13	n.s.[a]	14400[b]	23	0.80	19	96.34	0.68	n.s.[a]	1.21
System2	250000	2	32	10852.00	12	0.08	8	9.37	0.25	4.00	1.50
	100000	5	n.s.[a]	14400[b]	12	0.08	9	13.64	0.56	n.s.[a]	1.33
	50000	10	n.s.[a]	14400[b]	12	0.08	11	30.04	0.91	n.s.[a]	1.09
	25000	20	n.s.[a]	14400[b]	24	6.99	23	1215.08	0.87	n.s.[a]	1.04
System3	1000000	2	21	1461.44	12	0.16	7	7.57	0.29	3.00	1.71
	500000	3	n.s.[a]	14400[b]	12	0.16	8	17.89	0.38	n.s.[a]	1.50
	100000	13	n.s.[a]	14400[b]	23	15.61	18	196.73	0.72	n.s.[a]	1.28
	50000	26	n.s.[a]	14400[b]	35	424.40	30	13814.2	0.87	n.s.[a]	1.17
System4	1000000	2	25	6483.78	12	0.24	7	11.45	0.29	3.57	1.71
	500000	4	n.s.[a]	14400[b]	12	0.24	8	27.05	0.50	n.s.[a]	1.50
	250000	7	n.s.[a]	14400[b]	12	0.24	9	41.80	0.78	n.s.[a]	1.33
	100000	16	n.s.[a]	14400[b]	23	31.58	19	1253.54	0.84	n.s.[a]	1.21
								Average for all designs	0.64	2.33	1.40
								Average for industrial designs only	0.58	3.23	1.38

a. No solution (n.s.)
b. Terminated by time-out

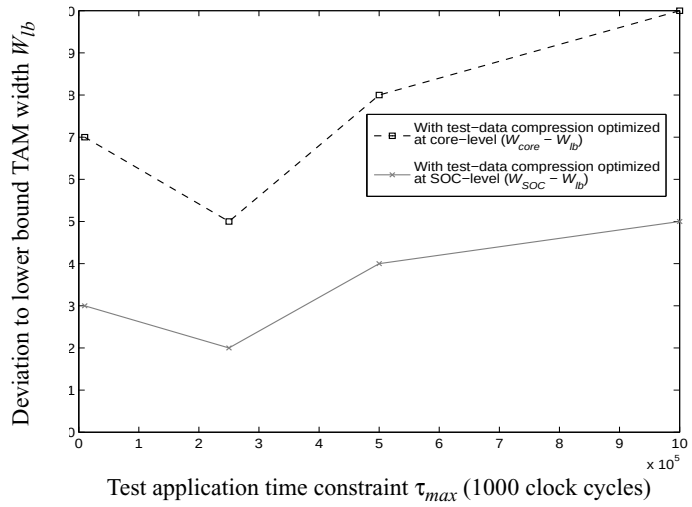
Table 7.6: Test-data volume results with test application time constraint

Design	Initial given test-data volume μ_i (Mbits)	Test application time constraint τ_{max} (1000 clock cycles)	Without compression	With test-data compression optimized at core-level	With test-data compression optimized at SOC-level	Data volume reduction factor		
			μ_{nc} (Mbits)	μ_{core} (Mbits)	μ_{SOC} (Mbits)	μ_i/μ_{SOC}	μ_{nc}/μ_{SOC}	μ_{core}/μ_{SOC}
d695	0.340	70.00	0.691	0.441	0.441	0.77	1.57	0.98
		60.00	0.697	0.468	0.468	0.73	1.49	0.92
		50.00	0.693	0.456	0.456	0.74	1.52	0.94
		40.00	0.696	0.441	0.441	0.77	1.58	0.98
System1	6547	500000	6547	2621	2621	2.50	2.50	0.13
		100000	n.s.[a]	511	511	12.82	n.s.[a]	0.67
		50000	n.s.[a]	359	359	18.23	n.s.[a]	0.95
		25000	n.s.[a]	347	347	18.89	n.s.[a]	0.98
System2	7967	250000	7973	1057	1057	7.54	7.54	0.48
		100000	n.s.[a]	670	670	11.89	n.s.[a]	0.76
		50000	n.s.[a]	497	497	16.02	n.s.[a]	1.02
		25000	n.s.[a]	515	515	15.47	n.s.[a]	0.99
System3	20731	1000000	20738	4721	4721	4.39	4.39	0.28
		500000	n.s.[a]	2749	2749	7.54	n.s.[a]	0.49
		100000	n.s.[a]	1625	1625	12.76	n.s.[a]	0.82
		50000	n.s.[a]	1363	1363	15.21	n.s.[a]	0.98
System4	24853	1000000	24869	5666	5666	4.39	4.39	0.28
		500000	n.s.[a]	3300	3300	7.53	n.s.[a]	0.48
		250000	n.s.[a]	2100	2100	11.83	n.s.[a]	0.75
		100000	n.s.[a]	1745	1745	14.24	n.s.[a]	0.91
					Average for all designs	9.21	3.12	0.74
					Average for industrial designs only	11.33	4.71	0.69

a. No solution (n.s.)



(a)



(b)

Figure 7.6: Deviation (a) to lower bound test application time at TAM width constraint and (b) to lower bound TAM width at test application time constraint for System4.

objects with a fixed size are packed into a small bin, which consequently may lead to an inefficient packing. For wider TAMs, the deviations are smaller since a wide TAM more easily can be partitioned into several TAMs such that multiple cores can be tested in parallel. This situation corresponds to a case when the objects are packed into a large bin, which in this case may lead to a more efficient packing.

Figure 7.6(b) shows, at various test application time constraints, the deviation from the lower bound TAM width when test-data compression is optimized at core-level (W_{core} - W_{lb}) and when the test-data compression is optimized at SOC-level (W_{SOC} - W_{lb}). For longer test application time constraints the deviation is increased. This is due to the fact that the long test application time leads to an extremely small number of TAM wires for the lower bound.

The results of the two experiments show that our approach produces results close to the lower bound. For the test application time minimization our approach is on average 15% (14%) from the lower bound (0.85x (0.86x)). For TAM width minimization our approach was on average 36% (42%) from the lower bound. This quite large number is explained by the longer test application time constraint, which leads to a very small TAM width for the lower bound. The longer test application time constraint is due to long CPU-times for the approach without test-data compression, used for comparison, for the industrial systems, System1 to System4. A time-out, set to 4 hours, was used to limit the CPU-time for the experiment. The results show that our approach is close to the lower bound when the test application time constraint is shorter, which, in fact, is the situation of interest in practice. For d695, the TAM width required by our approach using test-data compression optimized at SOC-level was on average only 10% from the lower bound. The results also indicate that our approach with test-data compression optimized at SOC-level is computationally effective. The CPU-time was less than one minute, even for the system with the largest number of cores and a wide TAM width.

7.8 Conclusions

To reduce both test application time and test-data volume, we propose a co-optimization technique for test-architecture design, test scheduling and test-data compression, based on core-level expansion of compressed test patterns. We analyzed, for a set of industrial cores, the inputs and outputs of the decoder

in relation to test application time and we found a non-monotonically decreasing behavior. We therefore propose a technique where we explore the trade-off between the test application time and test-data compression at core-level for each core and at SOC-level simultaneously. The proposed approach leads to regular test-access architectures and is able to leverage the large body of work that has been developed recently for test-architecture optimization and test scheduling. We have implemented the technique and compared it with previous work. We have also compared the approach with test-architecture design and test scheduling using SOCs crafted from industrial cores and the results show that we can get a test application time reduction on average 15x, a test-data volume reduction on average 16x, and a TAM width reduction on average 3x.

Chapter 8
Test-Architecture Design and Scheduling with Compression Technique Selection

THIS CHAPTER PRESENTS a technique to combine test-architecture design and test scheduling with test-data compression technique selection for each core in order to minimize the SOC test application time and the test-data volume. First, the proposed technique and the used test-architecture is introduced. Second, the different test-data compression technique alternatives are described and analyzed and the problem and the proposed algorithm are then presented. Finally, we present experimental results and make conclusions.

8.1 Introduction

As will be shown in this chapter, the performance of various test-data compression methods, with respect to compression ratio an test application time is different from method to method and it also depends on the actual TAM width. Thus, there is no single compression scheme that is optimal with respect to test application time reduction and test-data compression, for all TAM widths. We therefore propose a technique where we integrate core

wrapper design, test-architecture design and test scheduling with test-data compression method selection for each core in order to minimize the test application time and the test-data volume.

In the previous chapter, the test-data compression driven test-architecture design and test scheduling problem was solved while assuming one given test-data compression technique (Selective Encoding). In this chapter, the test-data compression driven test-architecture and test scheduling problem is extended to include the test-data compression technique selection. Since the proposed algorithm in Chapter 7 uses a semi-exhaustive approach to solve the test-architecture problem, the optimization time (CPU-time) becomes long if the test-data compression technique selection would be included. Therefore, we propose in this chapter, a new algorithm with a greedy heuristic for the test-architecture design that also includes the test-data compression technique selection.

We analyze the test application time and test-data compression ratio for three compression techniques. For a given core, we find for each technique different characteristics on compression ratio and test application time. And as the characteristics depend on the bitwidth assigned to a core, it is difficult to find the optimal bitwidth for each individual core in an SOC when the test-architecture is to be designed. We therefore present an optimization technique that for a given SOC, finds the best test-data compression technique for each core, designs the core wrapper, defines the test-architecture, and schedules the tests such that the SOC's overall test application time and test-data volume are minimized.

8.2 Test-Architecture

In this section the test-architecture using test-data compression technique selection is described. As in the previous chapter, the compressed test stimuli are stored in the ATE memory and are decoded at test application. The produced responses can be compacted on-chip and it is assumed that W_{TAM} wires are available for the transportation of test stimuli to the cores.

A typical test-architecture design using test-data compression technique selection is illustrated in Figure 8.1 using the SOC in Figure 2.3. Here, the W_{TAM} wires have been partitioned in two Test bus TAMs ($g = 2$) of widths ${w_1}^{TAM}$ and ${w_2}^{TAM}$, respectively. Core c_1 and c_2 have been assigned to the TAM

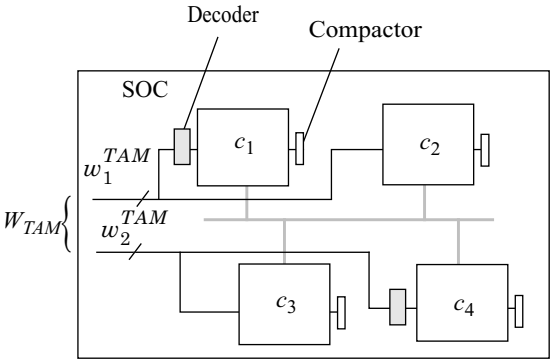


Figure 8.1: Example of a test-architecture with optimized decoders for each core for overall minimal test application time and test-data volume for the SOC in Figure 2.3.

with ${w_1}^{TAM}$ wires, and core c_3 and c_4 have been assigned to the TAM with ${w_2}^{TAM}$ wires. Furthermore, the test-data compression technique selection has defined decoders for core c_1 and c_4 while core c_2 and c_3 are tested without using on-chip decoders.

For the test-data compression, we have made use of the following alternatives: Selective Encoding and Vector Repeat, which are described in Section 3.3, and the combination of Selective Encoding and Vector Repeat (SE_VR). For SE_VR, the test-data is first compressed by using Selective Encoding and then compressed by applying Vector Repeat.

8.3 Analysis of Test-Data Compression

In this section, we analyze the test application time and test-data volume for the three test-data compression techniques, Selective Encoding, Vector Repeat and SE_VR, at various TAM widths using a number of cores.

In Chapter 7, we analyzed the test-data compression achieved using Selective Encoding in terms of the inputs (TAM width) and outputs (wrapper chains) of the decoder in relation to test application time and we found a non-monotonically decreasing behavior. For the analysis in this chapter, the best number of wrapper chains at each TAM width is selected.

The experiments have been performed on the cores in d695 [Mar02] and on the industrial cores [Wang05]. We found a similar behaviour for all 20 cores, however, we choose to present the results for the d695 core s9234 and the industrial core ckt-7. The results concerning test application time at various bandwidths are presented for the two cores in Figure 8.2, and the results with respect to test-data volume are reported in Figure 8.3.

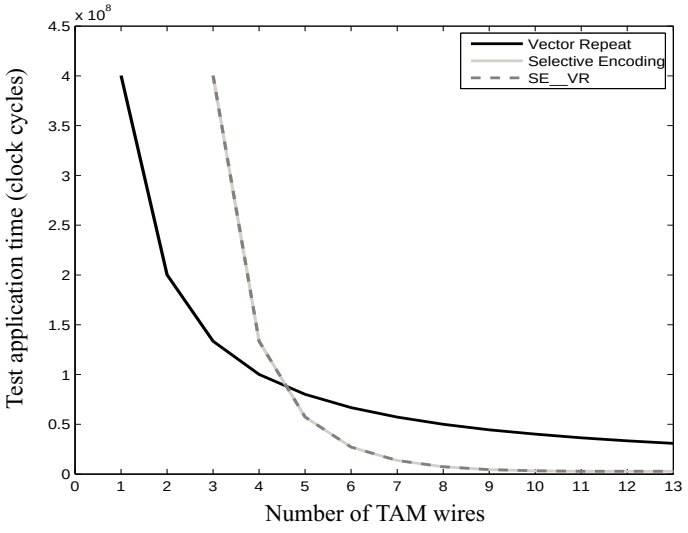
The results in Figure 8.2 show that the test application times for Selective Encoding and SE_VR are always the same as Vector Repeat does only compress in space domain and no compression is performed in the time domain. When comparing the test application time for the three compression techniques, Vector Repeat is better for lower TAM widths while Selective Encoding and SE_VR are better for wider TAM widths. Further, Selective Encoding cannot be applied to a narrow TAM as the technique requires a minimum of three TAM wires. Selective Encoding requires two TAM wires for control of the decoder, hence a minimum of three TAM wires are required for one wrapper chain. In summary, there is no compression technique that produces test application times that are best for any TAM width. For example, in Figure 8.2(a), the test application time of Vector Repeat is better than that of Selective Encoding and SE_VR at TAM width 4 while at TAM width 8 it is the other way around.

The test-data volume, obtained using different compression techniques for various TAM widths, is shown in Figure 8.3. As for the test application time, the test-data volume is not constant at various TAM widths. For Vector Repeat the compression ratio decreases, (the compressed test-data volume increases,) at wider TAMs. This is due to the fact that it is more difficult to find overlapping vectors when the slice (TAM width) increases. The test-data volume decreases for Selective Encoding as TAM width increases. However, as shown in Figure 8.3(b), the compression ratio gets worse for wider TAMs.

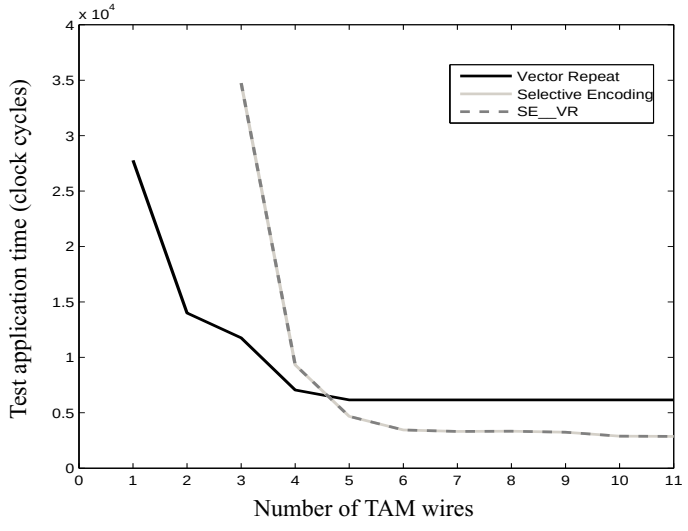
The analysis on test application time and test-data volume requirement for the compression techniques shows that there is no single compression technique that produces the best results in terms of test application time (Figure 8.2) as well as test-data volume (Figure 8.3) for all TAM widths.

In conclusion:

- it is not trivial to select the test-data compression scheme that produces the lowest test application time and the best compression ratio, and

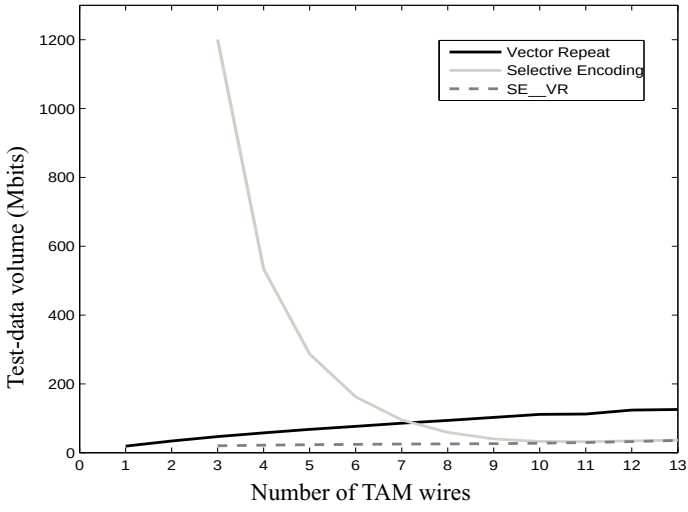


(a)

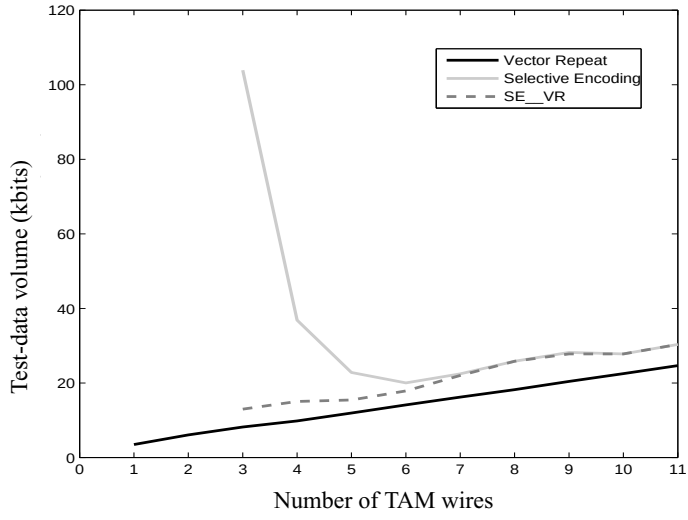


(b)

Figure 8.2: Test application time using different compression techniques at various TAM widths for (a) ckt-7 and (b) s9234.



(a)



(b)

Figure 8.3: Test-data volume using different compression techniques at various TAM widths for (a) ckt-7 and (b) s9234.

- for a core-based SOC, where the test-architecture is to be designed and several cores are to be assigned to the same TAM, it is not trivial to find the TAM widths such that they best fit all cores.

Therefore, there is a need to include the selection of test-data compression scheme when the test-architecture and test schedule are defined in order to minimize the SOC's overall test application time and the test-data volume.

8.4 Problem Formulation

Given is a system consisting of a number of cores as described in Section 4.1 Furthermore, it is assumed that a set of compression techniques is available:

R = {No Compression, Selective Encoding, Vector Repeat, SE_VR}.

For each compression technique r, where $r \in R$, we can easily determine:

- $\tau_i(w^{TAM}, w, r)$ - the test application time using test-data compression technique r at w number of TAM wires and m number of wrapper chains.
- $\mu_i(w^{TAM}, w, r)$ - the compressed test-data volume using compression technique r at w number of TAM wires and w number of wrapper chains.

For Selective Encoding, w^{TAM} is the TAM width and the core's decoder input, and w is the decoder output and the number of wrapper chains. For Vector Repeat and No Compression where no decoder is used, $w = w^{TAM}$, while for Selective Encoding and SE_VR, w is an input parameter.

Given $\tau_i(w^{TAM}, w, r)$ and $\mu_i(w^{TAM}, w, r)$, the cost $Cost_i(w^{TAM}, w, r)$ for a core i at w^{TAM} number of TAM wires, w wrapper chains using compression technique r is:

$$Cost_i(w^{TAM}, w, r) = \alpha \times \tau_i(w^{TAM}, w, r) + \beta \times \mu_i(w^{TAM}, w, r), \quad (8.1)$$

where α ($\{0 \le \alpha \le 1\}$) and β ($\{0 \le \beta \le 1\}$) are used to set the weight of the test application time and the test-data volume, respectively. The value of α and β are set such that $\alpha + \beta = 1$. The minimum cost $MinCost_i$ for a core i is finally given as:

$$MinCost_i = \min\{Cost_i(w^{TAM}, w, r)\}, \forall w^{TAM} \forall w \forall r \quad (8.2)$$

The problem is formulated as follows: For a given SOC with a given TAM width W_{TAM}, partition the TAM and determine each TAM's width, assign the cores to the TAMs, schedule the transportation of tests, and select a

compression technique for each core, such that the system's cost is minimized. The system's cost $Cost_{SOC}$ is given by:

$$Cost_{SOC} = \alpha \times \tau_{tot} + \beta \times \mu_{tot} \quad (8.3)$$

In this chapter it is assumed that each core is tested by applying one test ($q = N$). The test application time τ_{tot} for a test schedule with N cores is:

$$\tau_{tot} = \max\{t_i + \tau_i(w^{TAM}, w, r)\}, \forall i, i \in \{1, 2, \ldots, N\}, \quad (8.4)$$

where t_i is the start time when the test is applied to the core i, and the total test-data volume for the SOC with N cores is:

$$\mu_{tot} = \sum_{i=1}^{N} \mu_i(w^{TAM}, w, r) \quad (8.5)$$

8.5 Proposed Algorithm

A pre-process stage is used to generate, for each core, a number of wrapper and decompression design alternatives. For the wrapper design we have made use of the *Design_wrapper* algorithm (described in Section 3.1.1). For the decompression design, we make use of the Selective Encoding, the Vector Repeat and the SE_VR. We generate all alternatives for the decoder input/output mapping. For each compression technique r and each combination of w and m, we have for a core i the test application time $\tau_i(w^{TAM}, w, r)$.

The proposed algorithm consists of three procedures; initialization, compression technique selection, and test-architecture design and test scheduling. The three procedures are executed as illustrated in Figure 8.4 and detailed below.

For all iterations in the optimization loop a modified solution is generated and evaluated. From the current TAM architecture a new TAM architecture alternative is generated by merging TAMs. The merging of TAMs is explained as follows. Consider two TAMs TAM i and TAM j as candidates for a merge. A new TAM, with w_{new}^{TAM} TAM wires, will be generated where $w_{new}^{TAM} = w_i^{TAM} + w_j^{TAM}$. All cores, that previous to the merge were assigned to TAM i and TAM j, will after the merge be assigned to the new TAM.

The optimization loop is stopped when no new TAM architecture alternative and test schedule can be generated such that the system's cost is reduced. The rest of this section consists of a detailed description of each procedure.

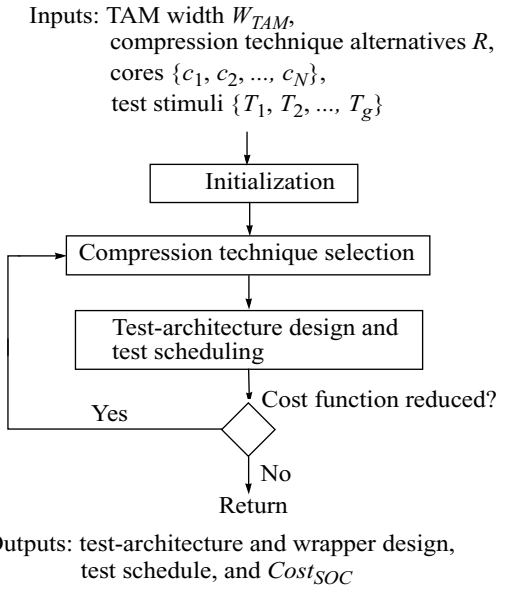


Figure 8.4: Flow graph of the proposed algorithm.

8.5.1 The Initialization Procedure

In the initialization procedure, the initial test-architecture (TAM design and wrapper design) and test schedule are designed. In the compression technique selection procedure a compression technique will be selected for each core. Finally, the test-architecture design and test schedule procedure and the compression selection procedure are used in an optimization loop.

The pseudo code for the initialization procedure is presented in Figure 8.5. The test transportation is sequential for each TAM. Hence, the test application time τ_{tam} for a TAM connected to N cores is given as:

$$\tau_{tam} = \sum_{i=1}^{N} \tau_i(w^{TAM}, w, r) \tag{8.6}$$

For the TAM architecture design, we have two different cases. First, the number of cores N is larger than the given TAM width W_{TAM}, and second, when N is less or equal to W_{TAM}. If the number of cores N is larger than the number of TAM wires W_{TAM} (line 2 in Figure 8.5), than the number of TAMs

```
1   Procedure Initialization()
2       If N > W_TAM
3           g = W_TAM
4           For each TAM j
5               w_j^TAM = 1
6               If CalculateTestTime(TAM j) is the shortest
7                   CoreMax = FindMaxTime()
8           Assign(CoreMax, TAM i)
9       Else // N ≤ W_TAM
10          g = N
11          For each TAM j
12              w_j^TAM = 1
13              CoreMax = FindMaxTime();
14              Assign (CoreMax, TAM j)
15          RestOfTAMWires = W_TAM - N
16          For each RestOfTAMWires
17              If CalculateTestTime(TAM j) is the longest
18                  w_j^TAM = w_j^TAM + 1
19  End
```

Figure 8.5: Initialization procedure.

g is set to W_{TAM}. Each core is assigned to one TAM according to the core's test application time such that the overall test application time is minimized (line 4–8).

If N is smaller or equal to W_{TAM} (line 9), than g is set to N. One core and one TAM wire is assigned to each TAM (line 10–14). The TAM wires, *RestOfTAMWires*, which so far, have not been assigned to any core, are assigned to the TAMs based on the TAMs' test application time, such that the overall test application time is minimized (line 15–17).

8.5.2 Compression Technique Selection Procedure

For a given test-architecture and test schedule, each core will be assigned to one compression technique such that the system's cost is minimized. The pseudo code for the compression technique selection procedure is presented in Figure 8.6.

Three loops are used for the selection of the test-data compression technique alternative. The first loop j (line 2 in Figure 8.6) is used to iterate over the g TAMs, the second loop i (line 3) iterates over the cores that are connected to TAM j, and finally, the third loop (line 4) iterates over the available compression technique alternatives.

```
1   Procedure CompressionTechniqueSelection()
2       For each TAM j
3           For each core i connected to TAM j
4               For each compression technique alternative r
5                   AssignCompression(w_j^TAM w_i, r)
6                   TmpCost_SOC = CalculateCost()
7                   If TmpCost_SOC < Cost_tot
8                       Cost_tot = TmpCost_SOC
9                   Else
10                      UnAssignCompression(w_j^TAM w_i, r)
11  End
```

Figure 8.6: Compression technique selection procedure.

The function *AssignCompression* (line 5) is used to assign a compression technique alternative r to a core i connected to a TAM with TAM width w_j^{TAM}. The selected compression technique is accepted if the cost is reduced (line 7–8). If the cost is not reduced, another compression technique alternative will be evaluated. If no compression technique that reduces the cost can be found, the current compression technique for the core is retained.

8.5.3 Architecture Design and Scheduling

The pseudo code for the test-architecture design and test scheduling procedure is presented in Figure 8.7. An outer loop (line 2 in Figure 8.7) is iterated as long as further merging and selection of compression technique alternatives leads to a reduction of the system's cost.

For each iteration of the outer loop, the g TAMs are sorted decreasingly according to the cost (line 3). Two inner loops (line 6–17) are used to select the TAMs that are candidates for a merge such that the TAM with the highest cost is selected to be merged with the TAM with the lowest cost. For each new TAM design the compression technique selection procedure is invoked (line 9). If a merge is found that leads to a reduction of the system's cost the inner loops are stopped and the outer loop is repeated. The test-architecture design and test scheduling procedure is stopped when no TAM design alternative is found such that the system's cost is reduced.

```
1   Procedure TestArchitectureDesignTestScheduling()
2       While Cost_SOC is reduced
3           SortTAMsDescByCost()
4           i = 1
5           j = g
6           For each TAM i
7               For each TAM j
8                   Merge(TAM i, TAM j)
9                   CompressionTechniqueSelection()
10                  Cost_New = CalculateCost()
11                  If Cost_New < Cost_SOC
12                      Cost_SOC = Cost_New
13                      g = g- 1
14                  Else
15                      UndoMerge(TAM i, TAM j)
16                  j = j + 1
17              i = i + 1
18  End
```

Figure 8.7: Test-architecture design and test scheduling procedure.

8.6 Experimental Results

In this section, we demonstrate the importance of integrating test-architecture design, wrapper design, test scheduling, and test-data compression technique selection.

We have carried out experiments on the benchmark design d695 [Mar02], and on four designs, System5, System6, System7, and System8, crafted using industrial cores, which are described in detail by Wang *et al*. [Wang05]. The characteristics for each design are presented in Table 8.1. Column 1 lists the design and Column 2 lists the number of cores. Column 3 and Column 4 list the number of FFs and the initial given test-data volume, respectively.

We minimize each system's cost $Cost_{SOC}$ for various TAM width W_{TAM} constraints. For each TAM width constraint, we run three different experiments; when $\alpha = 1$ and $\beta = 0$ that corresponds to test application time

Table 8.1: Design characteristics

Design	No. of cores	No. of FFs	Initial given (uncompressed) test-data volume (Mbits)
System5	10	246,581	20,801
System6	30	739,743	62,404
System7	60	1,479,486	124,808
System8	100	2,465,810	208,014
d696	10	6,348	0.34

minimization, when $\alpha = 0$ and $\beta = 1$ that corresponds to test-data volume minimization, and finally when $\alpha = 0.5$ and $\beta = 0.5$.

We compare our proposed algorithm (*PA*) with four other approaches; No Compression (NC), only Vector Repeat (VR), only Selective Encoding (SE), and only combined Selective Encoding plus Vector Repeat (SE_VR).

First, we graphically illustrate the importance of co-optimizing test-architecture design, test scheduling, and compression technique selection using two experiments. For the first experiment we use System6 with 16 TAM wires and for the second experiment we use System7 with 32 TAM wires. The results, test application time and test-data volume, for System6 and System7 are presented in Figure 8.8 and Figure 8.9, respectively. In Figure 8.8 (a) and Figure 8.9 (a), only the test application time is minimized ($\alpha = 1$ and $\beta = 0$). In Figure 8.8 (b) and Figure 8.9 (b), only the test-data volume is minimized ($\alpha = 0$ and $\beta = 1$). Finally, in Figure 8.8 (c) and Figure 8.9 (c), both the test application time and the test-data volume are minimized ($\alpha = 0.5$ and $\beta = 0.5$).

As can be seen in Figure 8.8 and Figure 8.9, when minimizing only the test application time the obtained solution may have a very large test-data volume and vice versa. Regardless of the objective of the minimization, test application time, test-data volume, or both test application time and test-data volume, our proposed approach, with compression technique selection produced the best solution for System6 and System7.

The rest of the results from the experiments for System5 to System8 and for d695 are collected in Table 8.2 to Table 8.6. Table 8.2 to Table 8.6 show for each system at various TAM width constraints, the test application time τ_{tot} and test-data volume μ_{tot}. The results in Table 8.2 to Table 8.6 are organized as follows: Column 1 lists the compression technique used, Column 2 lists the test application time and data factors, and Column 3 lists the TAM width constraint. Column 4 and Column 5 list the test application time and test-data volume for each compression technique. The last two columns highlight the comparison ratios. Column 6 lists the comparison (τ_{NC}/τ_{tot}) between the test application times τ_{tot} obtained when test-data compression is used and the test application time τ_{NC} obtained when test-data compression is not used. Column 7 lists the comparison (μ_{NC}/μ_{tot}) between the test-data volumes μ_{tot} obtained using test-data compression to the test-data volume μ_{NC} obtained when test-data compression is not used.

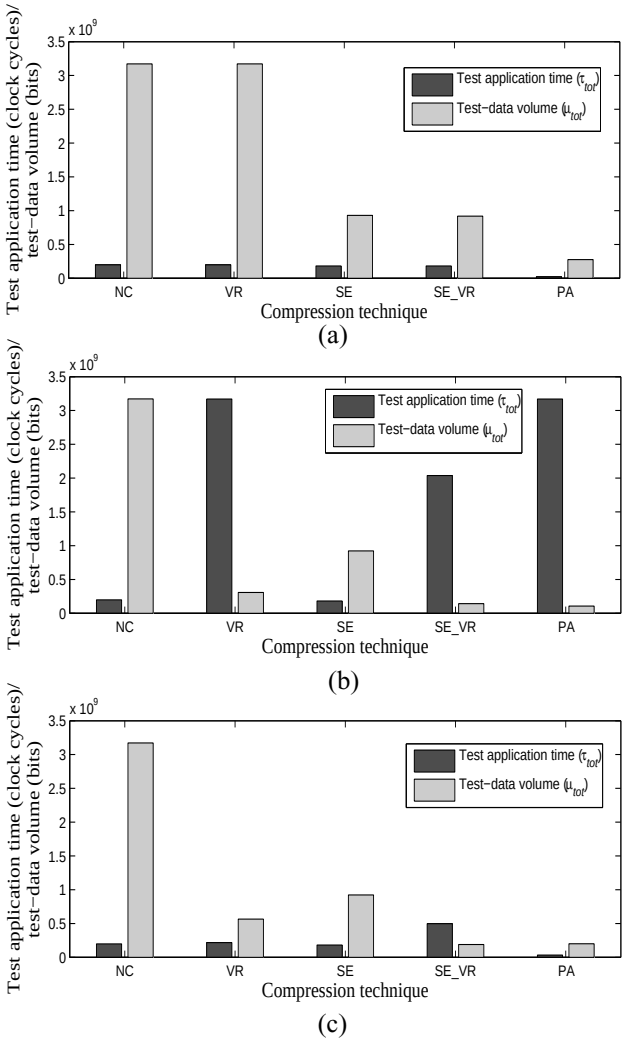


Figure 8.8: Test application time and test-data volume for System6 with 16 TAM wires and different compression techniques (No Compression (NC), Vector Repeat (VR), Selective Encoding (SE), Selective Encoding and Vector Repeat (SE_VR), and our Proposed Approach (PA)) when (a) only the test application time is minimized, (b) only the test-data volume is minimized, and (c) both test application time and test-data volume are minimized.

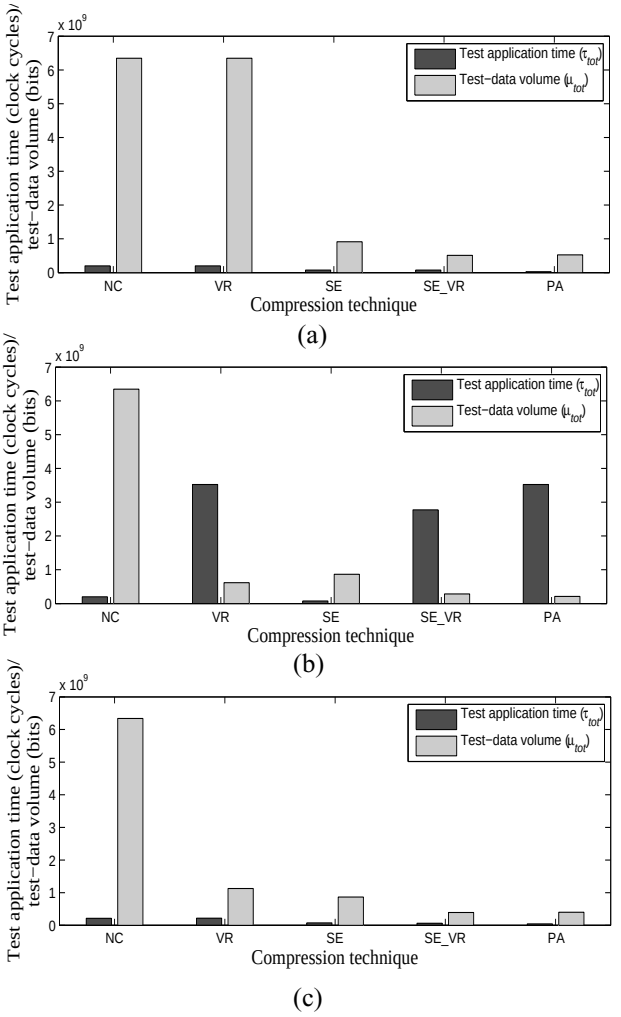


Figure 8.9: Test application time and test-data volume for System7 with 32 TAM wires and different compression techniques (No Compression (NC), Vector Repeat (VR), Selective Encoding (SE), Selective Encoding and Vector Repeat (SE_VR), and our Proposed Approach (PA)) when (a) only the test application time is minimized, (b) only the test-data volume is minimized, and (c) both test application time and test-data volume are minimized.

Table 8.2: Experimental results for System5

Technique	Test application time, data factor (α, β)	TAM width (W_{TAM})	Test application time τ_{tot} (1000 clock cycles)	Test-data volume μ_{tot} (Mbits)	Comparison of test application time τ_{NC}/τ_{tot}	Comparison of test-data volume μ_{NC}/μ_{tot}
NC	1, 0	8	132,158	1,057	1.00	1.00
VR			132,158	1,057	1.00	1.00
SE			19,809	158	6.67	6.67
SE_VR			19,809	63	6.67	6.67
PA			19,809	63	6.67	6.67
NC	0, 1	8	132,159	1,057	1.00	1.00
VR			1,056,690	103	0.13	0.13
SE			19,809	158	6.67	6.67
SE_VR			1,025,670	47	0.13	0.13
PA			1,056,610	35	0.13	0.13
NC	0.5, 0.5	8	132,159	1,057	1.00	1.00
VR			143,467	174	0.92	0.92
SE			19,809	158	6.67	6.67
SE_VR			19,809	63	6.67	6.67
PA			19,809	63	6.67	6.67
NC	1, 0	16	66,130	1,057	1.00	1.00
VR			66,130	1,057	1.00	1.00
SE			44,315	279	1.49	1.49
SE_VR			44,315	68	1.49	1.49
PA			7,386	92	8.95	8.95
NC	0, 1	16	66,154	1,057	1.00	1.00
VR			1,056,690	103	0.06	0.06
SE			44,315	279	1.49	1.49
SE_VR			661,933	47	0.10	0.10
PA			1,056,610	35	0.06	0.06
NC	0.5, 0.5	16	66,130	1,057	1.00	1.00
VR			76,167	238	0.87	0.87
SE			44,315	279	1.49	1.49
SE_VR			45,937	62	1.44	1.44
PA			11,153	67	5.93	5.93
NC	1, 0	32	35,133	1,058	1.00	1.00
VR			35,133	1,058	1.00	1.00
SE			8,352	116	4.21	4.21
SE_VR			8,352	74	4.21	4.21
PA			6,294	91	5.58	5.58
NC	0, 1	32	35,133	1,058	1.00	1.00
VR			626,467	103	0.06	0.06
SE			8,352	116	4.21	4.21
SE_VR			489,589	47	0.07	0.07
PA			626,457	35	0.06	0.06
NC	0.5, 0.5	32	35,133	1,058	1.00	1.00
VR			63,143	236	0.56	0.56
SE			8,352	116	4.21	4.21
SE_VR			7,952	66	4.42	4.42
PA			17,427	59	2.02	2.02

Table 8.3: Experimental results for System6

Technique	Test application time, data factor (α, β)	TAM width (W_{TAM})	Test application time τ_{tot} (1000 clock cycles)	Test-data volume μ_{tot} (Mbits)	Comparison of test application time τ_{NC}/τ_{tot}	Comparison of test-data volume μ_{NC}/μ_{tot}
NC	1, 0	8	396,478	3,170	1.00	1.00
VR			396,478	3,170	1.00	1.00
SE			59,428	474	6.67	6.68
SE_VR			59,428	190	6.67	16.67
PA			59,428	190	6.67	16.67
NC	0, 1	8	396,478	3,170	1.00	1.00
VR			3,170,080	308	0.13	10.28
SE			59,428	474	6.67	6.68
SE_VR			3,077,010	141	0.13	22.46
PA			3,169,840	106	0.13	30.05
NC	0.5, 0.5	8	396,478	3,170	1.00	1.00
VR			396,636	622	1.00	5.10
SE			59,428	474	6.67	6.68
SE_VR			59,428	190	6.67	16.67
PA			59,428	190	6.67	16.67
NC	1, 0	16	198,461	3,172	1.00	1.00
VR			198,461	3,172	1.00	1.00
SE			181,948	931	1.09	3.41
SE_VR			181,948	919	1.09	3.45
PA			22,158	275	8.96	11.54
NC	0, 1	16	198,461	3,172	1.00	1.00
VR			3,170,080	308	0.06	10.29
SE			181,948	923	1.09	3.44
SE_VR			2,038,160	141	0.10	22.48
PA			3,169,840	106	0.06	30.07
NC	0.5, 0.5	16	198,461	3,172	1.00	1.00
VR			217,030	565	0.91	5.62
SE			181,948	923	1.09	3.44
SE_VR			497,355	190	0.40	16.72
PA			33,459	201	5.93	15.81
NC	1, 0	32	99,349	3,173	1.00	1.00
VR			99,349	3,173	1.00	1.00
SE			24,284	383	4.09	8.29
SE_VR			24,284	243	4.09	13.07
PA			13,522	266	7.35	11.93
NC	0, 1	32	99,452	3,174	1.00	1.00
VR			2,091,340	308	0.05	10.29
SE			24,284	370	4.10	8.58
SE_VR			1,412,550	141	0.07	22.49
PA			2,091,120	106	0.05	30.08
NC	0.5, 0.5	32	63,729	3,170	1.00	1.00
VR			133,497	548	0.48	5.78
SE			7,513	256	8.48	12.39
SE_VR			20,994	192	3.04	16.49
PA			27,281	174	2.34	18.24

Table 8.4: Experimental results for System7

Technique	Test application time, data factor (α, β)	TAM width (W_{TAM})	Test application time τ_{tot} (1000 clock cycles)	Test-data volume μ_{tot} (Mbits)	Comparison of test application time τ_{NC}/τ_{tot}	Comparison of test-data volume μ_{NC}/μ_{tot}
NC	1, 0	8	792,751	6,339	1.00	1.00
VR			792,751	6,339	1.00	1.00
SE			118,856	949	6.67	6.68
SE_VR			118,856	380	6.67	16.67
PA			118,856	380	6.67	16.67
NC	0, 1	8	792,956	6,340	1.00	1.00
VR			6,340,150	617	0.13	10.28
SE			118,856	949	6.67	6.68
SE_VR			6,154,010	282	0.13	22.46
PA			6,339,670	211	0.13	30.05
NC	0.5, 0.5	8	792,751	6,339	1.00	1.00
VR			792,751	1,243	1.00	5.10
SE			118,856	949	6.67	6.68
SE_VR			118,856	380	6.67	16.67
PA			118,856	380	6.67	16.67
NC	1, 0	16	396,478	6,340	1.00	1.00
VR			396,478	6,340	1.00	1.00
SE			396,115	1,988	1.00	3.19
SE_VR			396,115	1,962	1.00	3.23
PA			44,315	550	8.95	11.53
NC	0, 1	16	396,922	6,345	1.00	1.00
VR			6,340,150	617	0.06	10.29
SE			396,115	1,974	1.00	3.21
SE_VR			4,658,400	282	0.09	22.48
PA			6,339,670	211	0.06	30.07
NC	0.5, 0.5	16	396,636	6,339	1.00	1.00
VR			396,636	1,243	1.00	5.10
SE			396,115	1,975	1.00	3.21
SE_VR			527,337	378	0.75	16.77
PA			66,918	401	5.93	15.79
NC	1, 0	32	198,461	6,345	1.00	1.00
VR			198,461	6,345	1.00	1.00
SE			74,375	913	2.67	6.95
SE_VR			74,375	510	2.67	12.44
PA			28,547	526	6.95	12.05
NC	0, 1	32	198,461	6,345	1.00	1.00
VR			3,525,450	617	0.06	10.29
SE			74,375	867	2.67	7.32
SE_VR			2,771,640	282	0.07	22.48
PA			3,525,400	211	0.06	30.07
NC	0.5, 0.5	32	215,936	6,340	1.00	1.00
VR			217,030	1,129	0.99	5.61
SE			74,375	867	2.90	7.31
SE_VR			60,996	392	3.54	16.17
PA			43,970	401	4.91	15.79

Table 8.5: Experimental results for System8

Technique	Test application time, data factor (α, β)	TAM width (W_{TAM})	Test application time τ_{tot} (1000 clock cycles)	Test-data volume μ_{tot} (Mbits)	Comparison of test application time τ_{NC}/τ_{tot}	Comparison of test-data volume μ_{NC}/μ_{tot}
NC	1, 0	8	1,321,250	10,566	1.00	1.00
VR			1,321,250	10,566	1.00	1.00
SE			198,093	1,581	6.67	6.68
SE_VR			198,093	634	6.67	16.67
PA			198,093	634	6.67	16.67
NC	0, 1	8	1,321,590	10,567	1.00	1.00
VR			10,566,900	1,028	0.13	10.28
SE			19,8093	1,581	6.67	6.68
SE_VR			10,256,700	470	0.13	22.46
PA			10,566,100	352	0.13	30.05
NC	0.5, 0.5	8	1,321,250	10,566	1.00	1.00
VR			1,321,250	2,072	1.00	5.10
SE			19,8093	1,581	6.67	6.68
SE_VR			19,8093	634	6.67	16.67
PA			19,8093	634	6.67	16.67
NC	1, 0	16	660,630	10,566	1.00	1.00
VR			660,630	10,566	1.00	1.00
SE			660,622	3,299	1.00	3.20
SE_VR			660,622	3,257	1.00	3.24
PA			73,860	916	8.94	11.53
NC	0, 1	16	661,537	10,575	1.00	1.00
VR			10,566,900	1,028	0.06	10.29
SE			660,622	3,280	1.00	3.22
SE_VR			7,645,020	470	0.09	22.48
PA			10,566,100	352	0.06	30.07
NC	0.5, 0.5	16	660,630	10,566	1.00	1.00
VR			660,630	2,072	1.00	5.10
SE			660,622	3,280	1.00	3.22
SE_VR			880,523	634	0.75	16.67
PA			111,530	669	5.92	15.79
NC	1, 0	32	330,768	10,575	1.00	1.00
VR			330,768	10,575	1.00	1.00
SE			134,263	1,575	2.46	6.72
SE_VR			134,263	854	2.46	12.39
PA			42,843	916	7.72	11.54
NC	0, 1	32	330,768	10,575	1.00	1.00
VR			6,060,690	1,028	0.05	10.29
SE			134,263	1,496	2.46	7.07
SE_VR			4,344,410	470	0.08	22.48
PA			6,059,890	352	0.05	30.07
NC	0.5, 0.5	32	333,200	10,567	1.00	1.00
VR			333,254	1,816	1.00	5.82
SE			134,263	1,496	2.48	7.06
SE_VR			123,269	648	2.70	16.30
PA			67,183	669	4.96	15.79

Table 8.6: Experimental results for d695

Technique	Test application time, data factor (α, β)	TAM width (W_{TAM})	Test application time τ_{tot} (1000 clock cycles)	Test-data volume μ_{tot} (kbits)	Comparison of test application time τ_{NC}/τ_{tot}	Comparison of test-data volume μ_{NC}/μ_{tot}
NC	1, 0	8	85	667	1.00	1.00
VR			85	667	1.00	1.00
SE			47	349	1.82	1.91
SE_VR			47	345	1.82	1.93
PA			47	345	1.82	1.93
NC	0, 1	8	87	678	1.00	1.00
VR			634	49	0.14	13.98
SE			51	322	1.71	2.11
SE_VR			647	221	0.13	3.07
PA			634	49	0.14	13.98
NC	0.5, 0.5	8	87	678	1.00	1.00
VR			85	100	1.01	6.78
SE			49	323	1.75	2.10
SE_VR			64	258	1.35	2.62
PA			85	100	1.01	6.78
NC	1, 0	16	46	701	1.00	1.00
VR			46	701	1.00	1.00
SE			33	442	1.40	1.58
SE_VR			33	373	1.40	1.88
PA			33	373	1.40	1.88
NC	0, 1	16	51	787	1.00	1.00
VR			634	49	0.08	16.22
SE			49	316	1.04	2.49
SE_VR			647	221	0.08	3.57
PA			634	49	0.08	16.22
NC	0.5, 0.5	16	51	787	1.00	1.00
VR			59	117	0.86	6.73
SE			46	318	1.10	2.48
SE_VR			35	259	1.43	3.04
PA			59	117	0.86	6.73
NC	1, 0	32	26	728	1.00	1.00
VR			26	728	1.00	1.00
SE			16	401	1.60	1.81
SE_VR			16	400	1.60	1.82
PA			16	400	1.60	1.82
NC	0, 1	32	31	872	1.00	1.00
VR			372	49	0.08	17.98
SE			27	316	1.12	2.76
SE_VR			374	221	0.08	3.95
PA			372	49	0.08	17.98
NC	0.5, 0.5	32	27	787	1.00	1.00
VR			42	130	0.65	6.05
SE			23	318	1.21	2.47
SE_VR			20	255	1.35	3.08
PA			42	130	0.65	6.05

On average, our proposed approach, with test-data compression selection, results in a 6.33x reduction in test application time (when only the test application time is considered in the optimization). The corresponding reduction in test-data volume is on average 27.26x. When both the test application time and the test-data volume were optimized the test application time was reduced by 4.41x and the test-data volume was reduced by 14.44x.

The proposed algorithm assumes that wrapper designs are available for all compression techniques, at all TAM/wrapper chains alternatives, for all cores. The process of generating these alternatives is quite time consuming.

However, once this information is available, our algorithm is computationally effective. The CPU-time (execution time to produce the solutions, excluding the time for core wrapper design and test-data compression) is very short. For the largest design, System8 and the widest TAM width constraint $W_{TAM} = 32$, the CPU-time was less than 1 second.

For the experiments presented in this section, we have varied α and β between 0, 1, and 0.5. Suitable values of α and β can be extracted using additional experiments. For example, a designer of such experiment can use the following three steps: (1) setting $\alpha = 1$ ($\beta = 0$), (2) setting $\alpha = 0$ ($\beta = 1$), and (3) repeatedly increasing/decreasing the value of α/β.

8.7 Conclusions

We have analyzed the test application time and test-data compression ratio for the test-data compression schemes Selective Encoding, Vector Repeat and the combination of Selective Encoding and Vector Repeat for a number of ISCAS cores and industrial cores (in total 20 cores). The analysis shows that the test application time and the test-data compression ratio are compression method dependant as well as TAM width dependant. It is, therefore, not trivial to select the optimal compression scheme for a core. Further, the behavior on test application time and test-data compression ratio are independent; hence it is difficult to select the optimal TAM width for a given core such that both test application time and compressed test-data are reduced. The problem becomes even more difficult for a core-based SOC as cores assigned to the same TAM must have the same bandwidth. We, therefore, proposed a technique to integrate test-data compression selection with test-architecture design and test scheduling. Our technique selects test-data compression technique for each core, designs the core wrapper, defines the number and widths of each TAM, and schedules the testing of the cores on the test-architecture such that the test-application time and the test-data volume are reduced. We have performed experiments on several SOCs that are crafted from industrial cores. The experimental results demonstrate that the proposed method leads to significant reduction in test-data volume, on average 26.56x, and test application time, on average 6.14x.

Chapter 9
Test Hardware Minimization under Test Application Time Constraint

THIS CHAPTER PRESENTS a test-architecture in which buffers are inserted between the functional bus and the cores. First, the proposed technique and the used test-architecture are introduced. Second, the proposed test-architecture and test scheduling using buffers are described. The hardware overhead minimization is illustrated using a motivational example, which is followed by the problem formulation. The problem has been solved using a CLP formulation and by using a Tabu search-based algorithm. Experimental results are presented and conclusions are drawn.

9.1 Introduction

We propose a test-architecture design and test scheduling technique that utilizes the functional bus as TAM. Different strategies have been proposed to solve the test-architecture design and/or the test scheduling problem [Aer98], [Goel03], [Iye02b], [Lar01], [Mar98a], [Seh04], [Var98], [Xu04]. The main disadvantage with these approaches is that they require additional TAM wiring overhead.

The main advantage of reusing the functional connections is that no, or few, TAM wires are required and several such techneques have been proposed [Har99], [Hwa01]. However, none of the proposed methods that make use of the functional bus take into consideration the hardware overhead introduced by test harness (wrapper) and/or the added test controller.

We propose a technique that makes use of the existing functional buses for the test data transportation inside the SOC. The proposed technique is based on the assumption that one or more cores in the system have a test application time that is longer than the test transportation time. This is, e.g., true when the number of scan chains in a core is smaller than the bandwidth of the TAM, thus making it possible to transfer, in a given period of time, more test-data on the TAM than can be applied to the core.

The proposed test-architecture makes use of buffers that are inserted between the functional bus and each core and the tests are divided into packages. By using buffers, tests can be applied concurrently even if the bus only allows sequential transportation. Furthermore, a test controller is proposed, which is responsible for the invocation of transmissions of the tests on the bus. We have dealt with the test-architecture design and test scheduling problems and developed a technique to minimize the test controller and buffer size.

First, the problem has been modelled and solved optimally using CLP. Since CLP uses an exhaustive search approach, the optimization time can become long for complex designs. Therefore, a Tabu search-based algorithm is proposed that works for larger designs, and is compared with the results attained from the CLP approach.

9.2 Test-Architecture

In this section, the proposed test-architecture using a functional bus access TAM is described.

The example in Figure 9.1 shows a system consisting of three cores, c_1, c_2, and c_3, all connected to the functional bus *bf*. Each core is associated with a buffer bu_i placed between the core and the bus. Also connected to the bus are two test components, SRC_T and $CTRL_T$. We assume that the tests are all produced in the test source SRC_T and the test controller $CTRL_T$ is responsible for the invocation of transmissions of the tests on the bus. A finite state machine is used to capture the complexity of the test controller. It is assumed

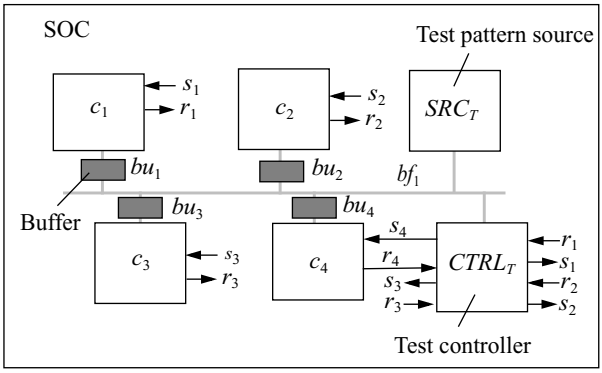


Figure 9.1: Bus-based architecture with buffers, one test pattern source, and one test controller.

that the core itself handles the evaluation of the produced responses, by, for example, a multiple-input signature analyser (MISR), and, thus, the cores act as the test sink. The information needed for the final test result evaluation is also sent via the bus.

The test controller is a finite state machine sending a signal s_i to each core indicating when it will receive a package of test-data. The signal, s_i, is also sent to the test source, SRC_T, indicating when a test should be transmitted on the bus. When the core has received the package, it sends a signal r_i to the controller, indicating that the controller can continue to transmit packages to another core.

9.3 Test-Architecture Design and Test Scheduling using Buffers

This section describes the proposed test-architecture where buffers are inserted between the cores and the functional bus. Further, the test scheduling, which makes use of the difference between the test application time and the test transportation time, and the calculation of the buffer size are described.

The proposed technique is based on the assumption that the test application of a core takes longer time than the test-data transportation. This difference is further illustrated with a small example (Figure 9.2). Here, a 20 bit wide ($w^{TAM} = 20$) functional bus bf_1 is connected to core c_1, with four scan chains

through a buffer bu_1. In only one clock cycle of the bus, the buffer is fed with 20 bits of test-data. The test-data is partitioned through a parallel to serial converter, to four scan chains, each with the length equal to the length of the longest scan chain. For the example, the length of the longest scan chain is 5 bits. During the next cycle, the bus can transport data to another core while core c_1 is occupied for another 5 clock cycles with the shift-in of the scan chains.

In order to make this approach more efficient the test stimuli for each core are divided into small packages, as illustrated in Figure 9.3 using core c_1 and c_2 in Figure 9.1. Figure 9.3(a) shows the connection of c_1 and c_2 to bf_1 using buffers bu_1 and bu_2. In Figure 9.3(b), a schedule is presented where c_1 is tested before c_2. In the example, the tests have not been divided into packages. Therefore, the test of c_1 is postponed until the transportation of test T_2 to core c_2 has finished. In Figure 9.3(c), c_1 is tested before c_2 and test T_1 has been divided into two separate packages, p_{11} and p_{12}, which then are scheduled in order but without a fixed interval between the packages. For this example, both core c_1 and c_2 are tested concurrently. The examples in Figure 9.3(b) and Figure 9.3(c) show that dividing tests into packages leads to a more flexible schedule, which also contributes to a possible decrease of the total test application time.

Each test T_i can be divided into m_{T_i} packages (each being a set of test vectors). As mentioned earlier, the transportation time $\tau_i^{send\text{-}p}$ for a package on the bus is shorter than the application time $\tau_i^{appl\text{-}p}$. The size of the buffer does not have to be equal to the size of the packages. This is explained by the

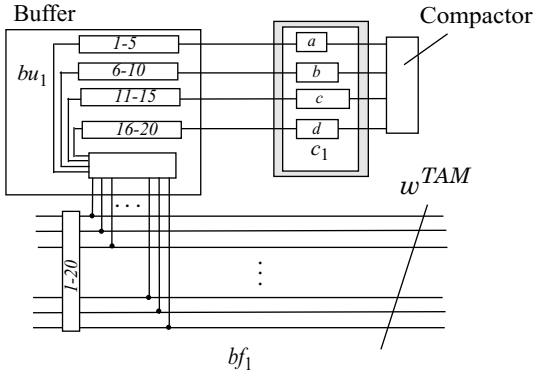


Figure 9.2: Functional bus, bf_1, and buffer connected to core, c_1.

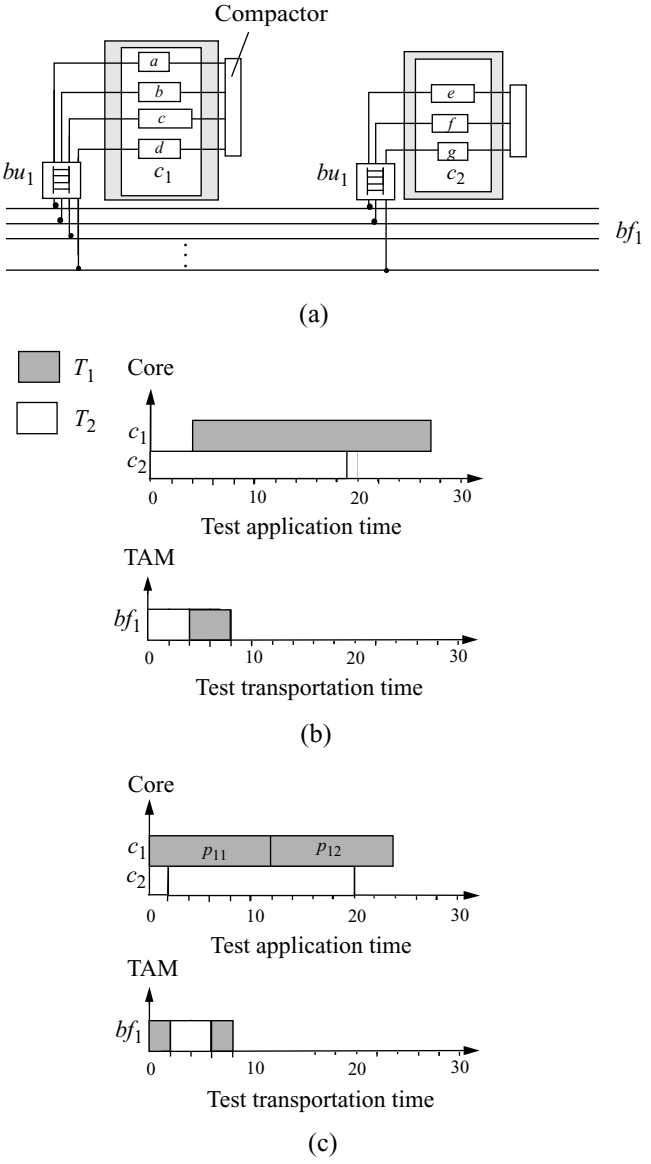


Figure 9.3: Example of (a) test-architecture with buffers and compactors and (b) test scheduling without packages and (c) with packages.

fact that the test-data in a package can be applied immediately when it arrives at the core. The buffer size bs_i, associated to a core i, is calculated with the following formula:

$$bs_i = \max(\lambda_i \times (t_{start_{ij}} - t_{send_{ij}}) + \Delta_i), j \in (1, m_{T_i}) \quad (9.1)$$

where the constant λ_i represents the rate (bits per clock cycle) at which the core can apply the test, the time $t_{start_{ij}}$ is the scheduled start time of the application of the package j from test T_i at the core, and $t_{send_{ij}}$ is the start time for sending the package on the bus. The constant Δ_i represents the leftover package size, which is the size of the test vectors that remain in the buffer after the transportation of the package terminates. This constant Δ_i is determined by the difference between $\tau_i^{appl\text{-}p}$ and $\tau_i^{send\text{-}p}$, which is multiplied by the constant λ_i.

The calculation of the buffer size is illustrated in Figure 9.4, which shows the bus schedule and the application of a test T_1 to core c_1, with $\tau_1^{appl} = 23$ clock cycles (Equation 3.1), $\tau_1^{send} = 4$ clock cycles, $m_{T_1} = 2$, and $\lambda_1 = 1$. In the example the core has not finished the testing using the package, p_{11}, sent at time point $t_{send_{11}} = 0$ before the package, p_{12}, sent at time point $t_{send_{12}} = 2$ arrives at the core. This forces the buffer size to be increased. For the example the buffer size will be equal to $1 \times (12 - 2) + 10 = 20$ bits, which is the difference between the termination of applying the last test package and the end point of transporting the corresponding package.

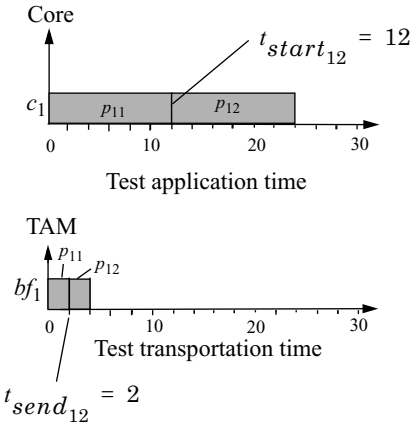


Figure 9.4: Example to illustrate time to transport and time to apply test using test T_1.

9.4 Motivational Example

The problem solved in this chapter explores the trade-off between the complexity of the test controller, given by the number of states N_s and number of transitions N_t, and the total buffer size bs_{tot}. This section describes the problem using a motivational example.

For this motivational example, the system in Figure 2.3 is used, which consists of four cores c_1, c_2, c_3, and c_4 which are tested by the four tests, T_1, T_2, T_3, and T_4, in Figure 2.14, respectively. We have divided the tests into a total number of seven packages. The test characteristics are presented in Table 9.1. Column 1 lists the tests and Column 2 lists the number of packages for each test m_{T_i}. Column 3 and Column 4 list the test application time $\tau_i^{appl\text{-}p}$ and test transportation time $\tau_i^{send\text{-}p}$ for a package, respectively. Column 5 lists the constant Δ_i. We assume that the test application time constraint, τ_{max}, for the system is given by the designer. In the example the test application time constraint is 24 clock cycles, which is the minimal time for applying these tests.

Two different schedules for the seven packages derived from the four tests are illustrated in Figure 9.5. In Figure 9.5(a), the packages are sent in such a way that the minimal number of control states is needed. For realizing the schedule in Figure 9.5(a) it is required that the test controller has four different control states $N_s = 4$, one for each test, and four transitions $N_t = 4$. This test schedule leads to a large buffer requirement, the total buffer size, bs_{tot}, given by the sum of bs_i, where $i = \{1, 2, 3, 4\}$. The total buffer size for the test schedule in Figure 9.5(a) is 53 ($bs_1 = 20$, $bs_2 = 16$, $bs_3 = 16$, $bs_4 = 1$) bits. Furthermore, the test schedule in Figure 9.5(a) leads to a long total test

Table 9.1: Test-data characteristics for the motivational example

Test	No. of packages m_{T_i}	Test application time $\tau_i^{appl\text{-}p}$ (clock cycles)	Test transportation time $\tau_i^{send\text{-}p}$ (clock cycles)	Δ_i
T_1	2	12	2	10
T_2	2	10	2	8
T_3	2	10	2	8
T_4	1	2	1	1

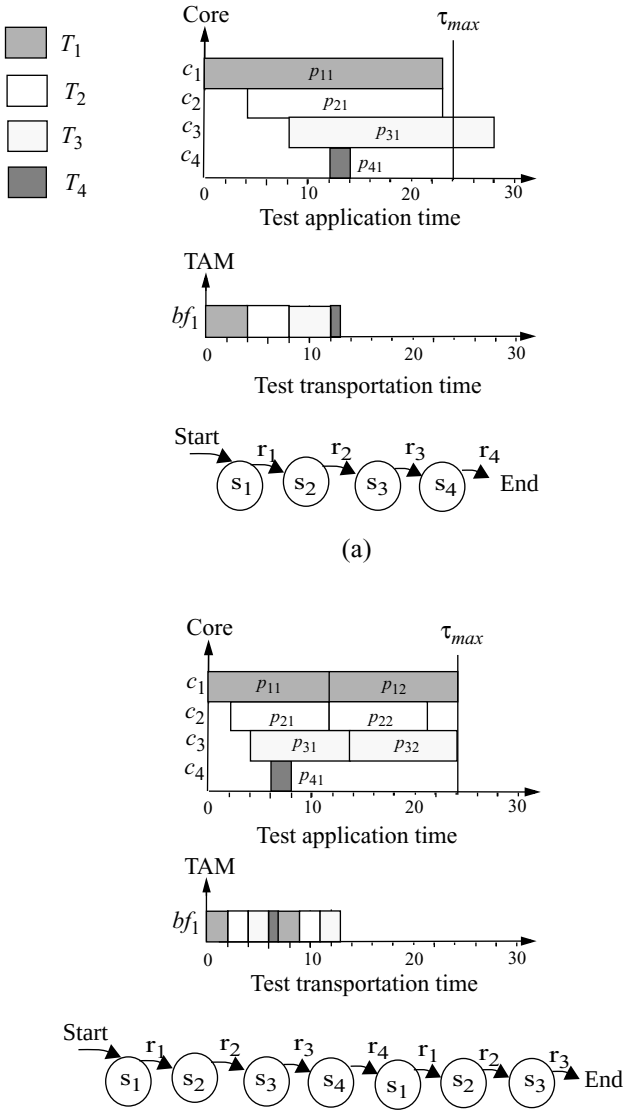


Figure 9.5: Scheduling examples with (a) small buffers and a high number of control states and (b) large buffers and few control states.

application time, τ_{tot}, such that the test application time constraint, $\tau_{max} = 24$ clock cycles, is violated.

In the second schedule, Figure 9.5(b), a maximal number of control states is used. The maximum number of control states is seven, which is equal to the total number of packages. For realizing the schedule in Figure 9.5(b) it is required that the test controller has seven different control states $N_s = 7$ and seven transitions $N_t = 7$. This test schedule leads to a small buffer requirement. The total buffer size, bs_{tot}, for the test schedule in Figure 9.5(b) is 38 ($bs_1 = 15$, $bs_2 = 11$, $bs_3 = 11$, $bs_4 = 1$) bits. Furthermore, the test schedule in Figure 9.5(b) leads to a short total test application time and the test application time constraint is not violated.

This example illustrates the trade-off between the complexity of the test controller, given by the number of states, and the buffer size. A small test controller with few states requires large buffers while a small buffer size requires many states in the test controller.

9.5 Problem Formulation

In this section, a detailed problem fromulation is presented. Given is a system consisting of one functional bus, bf_1, and a number of cores as described in Section 4.1. Each core, c_i, has a buffer bu_i where bs_i is the buffer size (initially bs_i is not determined).

The maximal allowed test application time for the system, τ_{max}, is given as a constraint and for each test T_i, the following information is given:

- the test application time τ_i^{appl} is the time needed to apply the test to core i,
- the test transportation time τ_i^{send} is the time needed to transport T_i from the test source SRC_T via the bus to core i,
- the size s^{T_i} is the number of test vectors in test T_i.

A test T_i, is divided into a number of m_{T_i} packages, each of equal size $s^{T_i\text{-}p}$. The package size $s^{T_i\text{-}p}$ for a test T_i is determined as follows[1]:

$$s^{T_i-p} = \left\lceil \frac{s^{T_i}}{m_{T_i}} \right\rceil \tag{9.2}$$

1. The last test package may have a smaller number of test vectors than $t_i^{size\text{-}p}$. We assume that this package is filled with arbitrary vectors.

The time $\tau_i^{appl\text{-}p}$ to apply a package belonging to test T_i is calculated as:

$$\tau_i^{appl-p} = \left\lceil \frac{\tau_i^{appl}}{m_{T_i}} \right\rceil \tag{9.3}$$

Associated to each package p_{ij} of test T_i where $j \in (1, m_{T_i})$, are three time points, $t_{start_{ij}}$, $t_{send_{ij}}$, and $t_{finish_{ij}}$. The time to send, $\tau_{send_{ij}}$, represents the start of the transmission of package, p_{ij}, on the bus. The time, $t_{start_{ij}}$, is the time to start the application of the test at the core c_i. Finally, $t_{finish_{ij}}$ is the time when the whole package has been applied. The finish time, $t_{finish_{ij}}$, is given by the following formula:

$$t_{finish_{ij}} = t_{start_{ij}} + \tau_i^{appl-p} \tag{9.4}$$

The complexity of the test controller $CTRL_T$ is given by the following formula described in [Mit93]:

$$CF1 = K \times \{(N_i + N_o + 2 \times \lceil \log_2 N_s \rceil) \times N_t + 5 \times \lceil \log_2 N_s \rceil\} \tag{9.5}$$

where N_i is the number of inputs, N_o the number of outputs, N_s the number of states and N_t the number of transitions. The formula estimates the complexity of a finite state machine in equivalent two-input NAND gates. In this work the number of inputs N_i and outputs N_o is equivalent to the number of cores and the number of transitions N_t is equal to the number of states N_s, which is equal to the number of packages, see Figure 9.5.

The objective of our technique is to find $t_{start_{ij}}$ and $t_{send_{ij}}$ for each package in such a way that the total hardware cost K_{tot} is minimized while satisfying the test application time constraint, τ_{max}. The total hardware cost for the test is computed by a cost function that consists of the system's total buffer size and the complexity of the controller given as follows:

$$K_{tot} = \alpha \times K_{CTRL} + \beta \times K_{Buffer} \tag{9.6}$$

where α and β are two coefficients used to set the weights of the controller and the buffer cost. The hardware cost of the buffers is given as:

$$K_{Buffer} = k_1^B + k_2^B \times bs_{tot} \tag{9.7}$$

and the controller:

$$K_{CTRL} = k_1^C + k_2^C \times CF1 \tag{9.8}$$

where the constants k_1^C and k_1^B are constants reflecting the base cost, which is the basic cost for having a controller and buffers, respectively, and k_2^C and k_2^B are design-specific constants that represent the implementation cost parameters for the complexity of the test controller and the buffer size. The buffer size is translated into estimated silicon area expressed by the number of NAND gates used.

The total buffer size bs_{tot} in the system is given by:

$$bs_{tot} = \sum_{i=1}^{N} bs_i \tag{9.9}$$

where N is the number of cores in the system.

9.6 Proposed Algorithm

This section describes the hardware cost minimization techniques, first using CLP and second, using a Tabu search-based algorithm.

9.6.1 Constraint Logic Programming Modelling

We have modelled the system in a CLP program, consisting of two main components, *Test* and *Package*. The *Test* component contains all given information for the tests and is used as the input to the program. In order to find a feasible solution that minimizes the total cost, the program ensures that a number of different constraints are fulfilled. These constraints are:

- the packages belonging to the same test have to be sent in a given order, i.e. $t_{start_{ij+1}} \geq t_{finish_i}$,
- the start time of a package should be later or equal to the time of transmission on the bus: $t_{start_{ij}} \geq t_{send_i}$,
- the time when a package has been completely applied to the core is equal to the time it starts the application plus the time used for application: $t_{finish_{ij}} \geq t_{start_{ij}} + \tau_i^{appl-p}$,
- the finish time of any test cannot exceed the total test application time constraint, τ_{max}: $t_{finish_{ij}} \leq \tau_{max}$.

The buffer size at a core is determined by the formula (Equation 9.1) presented in Section 9.3, and the hardware cost of a solution is given by the formula in Section 9.5 Equation 9.6.

With the above constraint set, the constraint solver searches for a solution that minimizes the hardware cost K_{tot} of the test.

9.6.2 Tabu Search-based Algorithm

We have also implemented a Tabu search-based optimization heuristic for the problem. The algorithm *HWMinimization* presented in Figure 9.6, takes as input: cores $\{c_1, c_2, ..., c_N\}$, tests $\{T_1, T_2, ..., T_q\}$, and the test application time constraint, τ_{max}. The produced outputs are the test-architecture, a test schedule, and the hardware cost K_{tot}. The *HWMinimization* algorithm consists of three steps: in step one (line 4–11 in Figure 9.6) an initial schedule is built, which is further improved in step two (line 12–15) and step three (16–34). The algorithm takes as additional input a minimal test application time possible for the tests, τ_{min}, which is the theoretical minimal time needed for transportation and application of the tests, with unlimited buffer and controller cost. This value can be computed by a CLP model in a very short time (less than one second for each of the experiments used in this chapter).

In the initial step, the tests are sorted descending according to their application time, τ_i^{appl}, and then the initial schedule is built. The slack, which is the difference between the end time of the schedule and τ_{max}, is calculated. In step two, the initial schedule is improved by distributing the slack between the packages, hence, decreasing the buffer size. After this step the slack is zero. The schedule is then further improved in step three, where a Tabu search-based strategy is used to find the best solution.

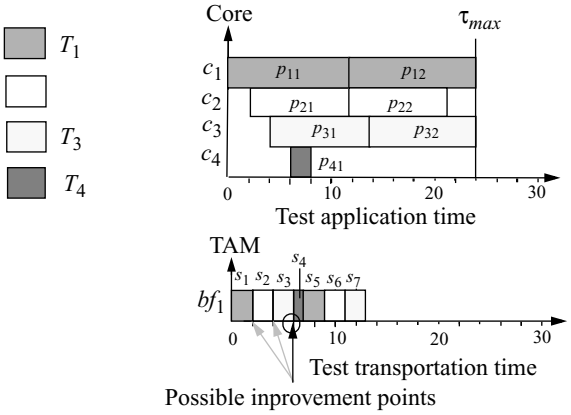
In our algorithm the neighborhood is determined by the possible points of improvements in the schedule. These can be points which decrease the buffer size by splitting a package, or decrease the controller cost by merging packages. Each possible improvement point is defined as a move, which, after it has been applied, is marked as a tabu. The application of a move is illustrated in Figure 9.7. In Figure 9.7(a), the different possible points of improvement are shown and one is selected. The move, which is selected, will reduce the number of control states since two packages will be merged. After the move selected in Figure 9.7(a) has been applied, the new schedule and the new possible points of improvement, are illustrated in Figure 9.7(b). The

```
1   Procedure HWMinimization
2   //Inputs: Cores, tests, test application time constraint (τ_max)
3   //Outputs: Test-architecture, test schedule, hardware cost (K_tot)
4   Step1: If τ_max < τ_min
5       Return "Not schedulable"
6   sort the tests in increasing order of τ_i^appl-p
7   While all packages not applied
8       apply package from T_i
9       While time < τ_i^appl-p do
10          apply package from T_i+1
11          time = time + τ_{i+1}^{send-p}
12  Step2:DoMark()
13  While slack > 0
14      delay package from mark_list
15  best_cost = CompCost(sched_0)
16  Step3:
17  Start:
18  DoMark()
19  For each pos in mark_list
20      build new schedule sched_i
21      delta_cost_i = best_cost - CompCost(sched_i)
22  For each delta_cost_i< 0, in increasing order of delta_cost_i
23      If not Tabu(pos) or TabuAspirated(pos)
24          Sched_0 = Sched_i
25          Goto accept
26  For each pos in MarkList
27      delta_cost_i' = delta_cost_i + Penalty(pos)
28  For each delta_cost_i' in increasing order of delta_cost_i'
29      If not Tabu(pos)
30          Goto Accept
31  Accept:
32  If iterations since previous best solution < max_iter
33      Goto Start
34  Return Sched_0
```

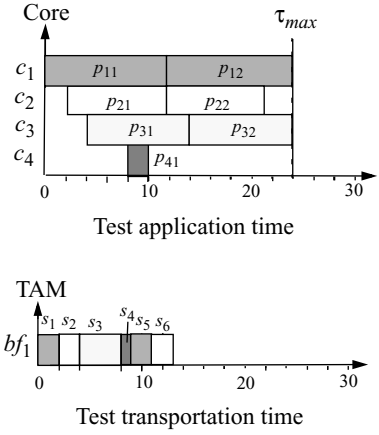
Figure 9.6: Algorithm for test hardware cost minimization

move, selected in each iteration, is the one which reduces the cost the most, however, a move that increase the cost is accepted if no other move is possible.

The tabu tenure *max_tabus*, that is the number of iterations when a move is kept as tabu, is set to seven. This value has to be long enough to prevent cycling without driving the solution away from the global optimum. Extensive experiments were carried out to find this value of the tenure. The tabu is aspirated if the cost of the obtained schedule is the best obtained so far. In order to find a good solution, an outer loop iterates until no further



(a)



(b)

Figure 9.7: Scheduling using proposed algorithm (Step3) with (a) possible improvement points and (b) after applying the selected move from (a).

improvement is made for *max_iter* = 10 consecutive tries. Also this number has been set on the basis of extensive experiments.

When the *HWMinimization* algorithm terminates, the solution (test-architecture and test schedule) with the lowest cost is returned.

9.7 Experimental Results

In our experiments we have used the following three designs: Asic Z [Zor93], [Chou97], Kime [Kime83], and System L [Lar01]. The main characteristics of the three designs, from the point of view of the problem addressed in this chapter, are presented in Table 9.2. Column 1 lists the name of the designs and Column 2 lists the number of cores N. The number of tests q and the total number of packages are presented in Column 3 and Column 4, respectively. Column 5 and Column 6 list the minimal buffer size bs_{min} and the maximum buffer size bs_{min}, respectively.

We have used the CLP-tool CHIP (V 5.2.1) [Cos96], [Hen91] for the implementation. The experiments have been performed in two steps. In the first step the minimal test application time is obtained assuming no division of the tests into packages, which corresponds to the traditional approach assumed by several existing test scheduling techniques. For experimental purposes the obtained test application time from step one is used as the test application time constraint, τ_{max}, in the second step, where the cost is minimized using the CLP approach.

The experimental results for the CLP solution are presented in Table 9.3, where the total cost of our proposed approach K_{PA} has been compared to the total cost obtained by the traditional approach K_{TA}. Column 1 lists the name of the designs and Column 2 lists the test application time constraint, τ_{max}. The total cost from the two approaches is presented in Column 3 and Column 4 and the cost comparison in Column 5. The results shows a decrease with 26 to 35% and with an average of 31% of the cost, which shows that our approach can decrease the cost by minimizing the buffer and controller, without exceeding the test application time limitation.

Since CLP uses an exhaustive search approach, the optimization time using CLP can become very large. For the largest benchmark, System L, the optimization time was more than 18 hours. The Tabu search-based heuristic

Table 9.2: Design characteristics

Design	No. of cores N	No. of tests q	Total no. of packages	Min buffer size bs_{min}	Max buffer size bs_{max}
Kime	6	6	20	186	680
Asic Z	9	9	38	222	838
System L	14	13	39	560	1976

Table 9.3: Hardware cost obtained using CLP

Design	τ_{max}	Traditional approach K_{TA}	Proposed approach K_{PA}	Cost comparison $\frac{(K_{PA}-K_{TA})}{K_{TA}} \times 100$
Kime	257	625	460	-26.4%
Asic Z	294	472	319	-32.4%
System L	623	1843	1182	-35.9%
			Average:	-31.6%

(Section 9.6.2), on the other hand, works for larger designs. In order to estimate the quality of the results produced by the Tabu search-based heuristic we have compared them with those generated by solving the same optimization problem using the CLP formulation (Section 9.6.1).

The experimental results using the CLP approach and the Tabu search-based algorithm are collected in Table 9.4. Column 1 lists the designs and Column 2 lists the test application time constraint, τ_{max}. The total cost using the proposed approach, K_{PA}, and the optimization time (CPU-time) for the CLP formulation are listed in Column 3 and Column 4, respectively. The total cost using the proposed Tabus search based algorithm, K_{TS}, and the optimization time are listed in Column 5 and Column 6, respectively. Finally, Column 7 lists the cost comparison between the cost produced with CLP and the cost obtained using the Tabu search-based algorithm.

As can be seen from the cost comparison, our Tabu search-based algorithm produced results which were on average only 4.9% worse then those produced by the CLP-based approach. However, the heuristic proposed in this paper take 3s for the largest example, while the CLP-based solver was running up to 18 hours.

Table 9.4: Hardware cost obtained using Tabu search-based algorithm

Design	τ_{max}	CLP		Tabu search-based algorithm		Cost comparison $\frac{(K_{TS}-K_{PA})}{K_{PA}} \times 100$
		Total cost K_{PA}	CPU-time (s)	Total cost K_{TS}	CPU-time (s)	
Kime	257	460	27375	486	2	5.3%
Asic Z	294	319	47088	330	2	3.4%
System L	623	1182	64841	1254	3	6.1%
					Average:	4.9%

We have also compared our results with the results produced by the CLP solver after the same time as our proposed algorithm needed, i.e. 2s for design Kime and Asic Z, and 3s for System L. For this experiment, the CLP is used as a heuristic where a timeout is used to stop the search and the best solution found so far is returned. This comparison showed that our Tabu search-based algorithm on average produced solutions that were 10.2% better.

9.8 Conclusions

A technique to make use of the existing functional bus structure in the system for test-data transportation is proposed. The advantage is that a dedicated TAM for test purpose is not needed hence we reduce the cost of additional test wiring. On the other hand, we insert buffers and divide the tests into packages, which means that tests can be applied concurrently even if the TAM only allows sequential transportation. We have proposed a CLP and a Tabu search-based algorithm where the test cost, given by the controller and buffer cost, is minimized without exceeding the given test application time constraint. The results indicate that the proposed heuristic produces high quality solutions at low computational cost.

Chapter 10
Conclusions and Future Work

THIS CHAPTER CONCLUDES the thesis and discusses possible directions of future work.

10.1 Conclusions

The increasing test-data volumes that are needed for fabrication test of System-on-Chip (SOC) circuits is a major problem since it leads to long test application time and high tester memory requirement. It is possible to address this problem by using test-architecture design, test scheduling, test-data compression, and test sharing and broadcasting.

In this thesis, the test application time, which is highly related to test cost, is minimized by using co-optimization of test-architecture design and test scheduling, which is extended to also include test sharing and broadcasting, test-data compression, and test-data compression technique selection. The test application time is minimized such that given resource constraints, such as TAM width and tester memory, are not exceeded. In addition, a test-architecture is proposed where buffers are inserted between each core and the functional bus. The test hardware overhead is minimized such that a given test application time is not exceeded. The main contributions of the thesis are as follows:

In Chapter 5 a technique has been proposed to explore the high amount of don't-cares present in the tests in order to create new tests to share, which can be used as alternatives to the original dedicated tests for the cores. The proposed method allows also the existing functional bus structure to be reused for the test-data transportation. In order to decrease the test application time, the test-architecture allows shared tests to be broadcasted and dedicated test buses may be added to the design. The problem is to select appropriate tests for each core, design wrapper chains for each core, insert test buses, and schedule the selected tests on the buses in such way that the test application time is minimized without exceeding the given hardware cost constraints. The problem has been modelled and solved using Constraint Logic Programming (CLP) and experiments show that the overall test application time can be significantly reduced when test sharing and broadcasting of tests are used.

In Chapter 6 a technique has been proposed that integrates test-data compression, test sharing, test-architecture design and test scheduling with the objective to minimize the test application time under ATE memory constraint. The work in Chapter 6 is concentrated in particular to the relation between test-data compression and test sharing in terms of test-data volume, and to the trade-off between test sharing versus test-architecture design in terms of test application time. The problem has been solved using both a CLP formulation and a Tabu search-based algorithm.

In Chapter 7 and Chapter 8, the analysis of test application time and test-data compression ratio for different test-data compression techniques shows that the test application time and the compression ratio are not only test-data compression technique dependant but also TAM width dependant. It is therefore not trivial to design the decoder and to select the optimal test-data compression technique and TAM width for a core. The overall test application time is minimized by test-architecture design, test scheduling, and test-data compression technique selection.

In Chapter 9, we have proposed a technique to make use of the existing functional bus structure in the system as TAM. We insert buffers and divide the tests into packages to address underutilization of the TAM. The buffers and packages enables concurrent test application even if the TAM only allows sequential transportation. We have proposed a CLP formulation and a Tabu search-based algorithm where the test cost, given by the controller and buffer cost, is minimized without exceeding a given test application time constraint.

Each of the problems described in this thesis has been modelled and implemented and extensive experiments have been performed to demonstrate the significance of each proposed approach.

10.2 Future Work

There are several possible extensions to the work presented in this thesis. In this section possible extensions, directly related to the work are presented. A few possible directions of future work that are beyond the scope of this thesis will also be discussed.

Here follows a list of possible extensions for each of the problems described in Chapter 5 to Chapter 9:

- The problem in Chapter 5 is only solved using a CLP modelling formulation. Since CLP uses an exhaustive search approach a heuristic technique is required to generate solutions for large designs.
- The problem in Chapter 6 can be extended to include the increased control overhead due to the concurrent test scheduling approach.
- The problem in Chapter 7 can be extended such that the pre-process stage of generating test alternatives is included in the optimization loop.
- The problem in Chapter 8 can be extended to include additional test-data compression technique alternatives that are considered in the optimization.
- The problem in Chapter 9 can be extended to include multiple functional buses. A more advanced test controller can be used.

Common for the proposed approaches in Chapter 5 to Chapter 9 is the use of a pre-process stage that solves the wrapper design problem. One extension, is therefore, to develop an optimization technique that includes the wrapper design optimization stage.

The rest of this section will be used to discuss about possible extensions that are beyond the scope of this thesis. This discussion will include functional self-test, at-speed BIST, thermal and power aware test optimization, and tolerance to transient faults.

Functional self-test is a test strategy where the programmable cores, such as processors, are used as test stimuli generators and produced response analyzer

for other cores in the system. The work presented in this thesis can be extended to include functional self-test in the test-architecture design and test scheduling, where precedence constraint are added such that the processor cores are tested before they are used to test other cores in the system.

Technology scaling generates both power and thermal problems. There is a close relationship between power consumption and the junction temperature. This relationship is due to the fact that the IC will use more current as it gets hotter, which results in more self-heating that eventually can lead to junction temperatures high enough to melt the package and possibly damage not only the IC under test but also the ATE. Therefore, the power consumption should be considered during the optimization.

In this thesis, we addressed the detection of stuck-at faults. However, in deep submicron technology, delay faults and noise faults may occur. To detect such faults several consequtive test patterns are needed to be applied with a specific timing to achive the desired bahavoir so that the faults can be detected. The order of the test patterns is important in order to detect delay fault and noise faults, which means that the proposed *share* function may not be suitable since it assumes that the test patterns can be applied in arbitrary order. Therefore, a new *share* function is needed that ensures that the intended order of which test patterns are applied is maintained.

References

[Aer98] J. Aerts and E. J. Marinissen, "Scan Chain Design for Test Application Time Reduction in Core-based ICs," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 448–457, 1998.

[ARM08] "AMBA Specification Overview," ARM, *Web site: www.arm.com/products/solutions/AMBAHomePage.html*, 2008.

[Bar87] P.H. Bardell, W.H. McAnney, and J. Savir, "Built-In Test for VLSI: Pseudorandom Techniques," *John Wiley and Sons*, 1987.

[Bar01] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B.Keller, and B. Koenemann, "OPMISR: The Foundation for Compressed ATPG Vectors," *In Proceedings of IEEE International Test Conference*, pp. 748–757, 2001.

[Brg89] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *In Proceedings of IEEE International Symposium on Circuits and Systems*, Vol. 3, pp. 1929–1934, 1989.

[Cha99] H. Chang, L. R. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, "Surviving The Soc Revolution - A Guide to Platform-based Design," *Kluwer Academic Publishers*, ISBN 0-7923-8679-5, 1999.

[Cha01] K. Chakrabarty, "Optimal Test Access Architectures for System-on-a-Chip," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 6, pp. 26–49, 2001.

[Cha03a] A. Chandra and K. Chakrabarty, "A Unified Approach to Reduce SOC Test-Data Volume, Scan Power and Testing Time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, Issue 3, pp. 352–363, 2003.

[Cha03c] M. Chandramouli, "How to Implement Deterministic Logic Built-In Self-Test (BIST)," *Compiler: A Monthly magazine for technologies world-wide*, Synopsys, January 2003.

[Chou97] R. M. Chou, K. K. Saluja, and V. D, Agrawal, "Scheduling Tests for VLSI Systems Under Power Constraints," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 5, Issue 2, pp. 175–185, 1997.

[Cos96] "CHIP, System Documentation," COSYTEC, 1996.

[DaS03] F. DaSilva, Y. Zorian, L. Whetsel, K. Arabi, and R. Kapur, "Overview of the IEEE 1500 Standard," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 988–997, 2003.

[DeM94] G. De Micheli, "Synthesis and Optimization of Digital Circuits," *McGraw-Hill Science/Engineering/Math*, ISBN: 0-07-016333-2, 1994.

[Eld59] R. D. Eldred, "Test Routines Based on Symbolic Logical Statements," *Journal of the ACM,* Vol. 6, No. 1, pp. 33–36, 1959.

[Gar79] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," *W. H. Freeman And Company*, New York, 1979.

[Glo89] F. Glover, "Tabu search - part I," *ORSA Journal on Computing*, Vol. 1, pp. 190–260, 1989.

[Glo90] F. Glover, "Tabu search - part II," *ORSA Journal on Computing*, Vol. 2, pp. 4–32, 1990.

[Goel83] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 215–222, March 1981.

[Goel03] S. K. Goel and E. J. Marinissen, "SOC test-architecture design for efficient utilization of test bandwidth," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 8, Issue 4, pp. 399–429, 2003.

[Goel04] S. K. Goel, C. Kuoshu, E. J. Marinissen, T. Nguyen, and S. Oostdijk, "Test Infrastructure Design for the Nexperia™ Home Platform PNX8550 System Chip," *In Proceedings of IEEE Design, Automation and Test in Europe (DATE)*, pp. 108–113, 2004.

[Gon04a] P. T. Gonciari and B. Al-Hashimi, "A Compression-Driven Test Access Mechanism Design Approach," *In Proceedings of IEEE European Test Symposium (ETS)*, pp. 100–105, 2004.

[Gon04b] P. T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-chip Test Data Compression/Decompression," *In Proceedings of IEEE Design, Automation and Test in Europe (DATE)*, pp. 604–611, 2002.

[Gon05] P. T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Synchronization Overhead in SOC Compressed Test," *IEEE Transactions on Very Large Scale Integration (VLSI) System*s, Vol. 13, Issue 1, pp. 140–152, 2005.

[Gup97] R.K. Gupta and Y. Zorian, "Introduction to Core-based System Design," *IEEE Design & Test of Computers*, Vol. 14, No. 4, 1997.

[Har99] P. Harrod, "Testing Reusable IP - A Case Study," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 493–498, 1999.

[Hen91] P. Van Hentenryck, "The CLP language CHIP: constraint solving and applications," *Compcon Spring '91. Digest of Papers*, pp. 382–387, 1991.

[Hir03] T. Hiraide, K.O. Boateng, H. Konishi, K. Itaya, M. Emori, H. Yamanaka, and T. Mochiyama, "BIST-Aided Scan Test - A New Method for Test Cost Reduction," *In Proceedings of IEEE VLSI Test. Symposium (VTS)*, pp. 359–364, 2003.

[Hus06] F. A. Hussin, T. Yoneda, H. Fujiwara, and A. Orailoglu, "Power-Constrained SOC Test Schedules through Utilization of Functional Buses," *In Proceedings of IEEE International Conference on Computer Design (ICCD)*, pp. 230–236, 2006.

[Hwa01] S. Hwang and J.A. Abraham, "Reuse of Addressable System Bus for SOC Testing," *In Proceedings of IEEE International ASIC/SOC Conference*, pp. 215–219, 2001.

[IBM05] "CoreConnect Bus Architecture," IBM, *Web site: www.chips.ibm.com/products/coreconnect,* 2005.

[IEEE07] "IEEE 1500 Standard for Embedded Core Test (SECT)," *Web site: http://grouper.ieee.org/groups/1500*, 2007.

[Ing05] U. Ingelsson, S. K. Goel, E. Larsson, and E. J. Marinissen, "Test Scheduling for Modular SOCs in an Abort-on-Fail Environment," *In Proceedings of IEEE European Test Symposium (ETS),* pp. 8–13, 2005.

[Imm90] V. Immaneni and S. Raman, "Direct access test scheme-design of block and core cells for embedded ASICs," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 488–492, 1990.

[Iye01a] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 1023–1032, 2001.

[Iye01b] V. Iyengar and K. Chakrabarty, "Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-Chip," *In Proceedings of IEEE VLSI Test Symposium (VTS)*, pp. 368–374, 2001.

[Iye02a] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip," *Journal of Electronic Testing: Theory and Applications (JETTA)*, Vol.18, No. 2, pp. 213–230, 2002.

[Iye02b] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Recent Advances in Test Planning for Modular Testing of Core-Based SOCs," *In Proceedings of IEEE Asian Test Symposium (ATS)*, pp. 320–325, 2002.

[Iye03] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Test Data Volume Reduction for System-on-Chip," *IEEE Transactions on Computer*s, Vol. 52, No. 12, pp. 1619–1632, 2003.

[Iye05] V. Iyengar and A. Chandra, "Unified SOC Test Approach Based on Test-Data Compression and TAM design," *In Proceedings of IEE Computer and Digital Techniques*, Vol. 152, Issue 1, pp. 82–88, 2005.

[Jaf87] J. Jaffar, and J.-L. Lassez, "Constraint Logic Programming," *In Proceedings of ACM Symposium on Principles of Programming Languages (POPL)*, pp. 111–119, 1987.

[Jas03] A. Jas, J. Ghosh-Dastidar, M. Ng, and N. Touba, "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding," *IEEE Transactions on Computer-Aided Design (TCAD)*, Vol. 22, pp. 797–806, 2003.

[Jia03] J.H. Jiang, W-B. Jone, S-C. Chang, and S. Ghosh, "Embedded Core Test Generation Using Broadcast Test-Architecture and Netlist Scrambling," *IEEE Transactions on Reliability*, Vol. 52, No. 4, pp. 435–443, 2003.

[Kaj01] S. Kajihara and K. Miyase, "On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits," *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 364–369, 2001.

[Kime83] C. R. Kime, "Test Scheduling in Testable VLSI Circuits," *In Proceedings of IEEE International Symposium on Fault-Tolerant Computing (FTSC)*, pp. 406–412, 1982.

[Kir83] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, pp. 671–679, 1983.

[Kob68] A. Kobayashi, S. Matsue, and H. Shiba, "Flip-Flop Circuit with FLT Capability," *In Proceedings of (IECEO) conference*, pp. 962, 1968.

[Koe01] B. Koenemann, C. Banhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding," *In Proceedings of IEEE Asia Test Symposium (ATS)*, pp. 325–330, 2001.

[Lar01] E. Larsson and Z. Peng, "An Integrated System-on-Chip Test Framework," *In Proceedings of IEEE Design, Automation and Test in Europe (DATE)*, pp. 138–144, 2001.

[Lar02] E. Larsson and H. Fujiwara, "Power Constrained Preemptive TAM Scheduling," *In Proceedings of IEEE European Test Workshop (ETW)*, pp. 119–126, 2002.

[Lar03a] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Buffer and Controller Minimisation for Time-Constrained Testing of System-On-Chip," *In Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems* (*DFT*), pp. 385–392, 2003.

[Lar03b] E. Larsson and Z. Peng, "A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling," *In Proceedings of IEEE International Test Conference (ITC)*, Vol. 1, pp. 1135–1144, 2003.

[Lar04a] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Technique for Optimization of System-on-Chip Test Data Transportation," *In Proceedings of IEEE European Test Symposium (ETS) (Informal Digest)*, pp. 179–180, 2004.

[Lar04b] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Technique for Optimisation of SOC Test Data Transportation," *Swedish System-on-Chip Conference (SSoCC) (Informal Digest)*, 2004.

[Lar05a] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Constraint Logic Programming Approach to SOC Test Scheduling," *Swedish System-on-Chip Conference (SSoCC) (Informal Digest)*, 2005.

[Lar05b] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Optimization of a Bus-based Test Data Transportation Mechanism in System-on-Chip," *In Proceedings of IEEE Euromicro Conference on Digital System Design* (*DSD*), pp. 403–409, 2005.

[Lar05c] A. Larsson, "System-on-Chip Test Scheduling and Test Infrastructure Design," *Licentiate Thesis No. 1206*, Department of Computer and Information Science, Linköping University, ISBN: 91-85457-61-2, 2005

[Lar05d] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "SOC Test Scheduling with Test Set Sharing and Broadcasting," *In Proceedings of IEEE Asian Test Symposium* (*ATS*), pp. 162–167, 2005.

[Lar06a] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "SOC Test Scheduling with Test Set Sharing and Broadcasting," *Swedish System-on-Chip Conference (SSoCC) (Informal Digest)*, 2006.

[Lar06b] E. Larsson, A. Larsson, and Z. Peng, "Linköping University SOC Test Site," *Web site: http://www.ida.liu.se/labs/eslab/soctest*, 2006.

[Lar07a] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "Optimized Integration of Test Compression and Sharing for SOC Testing," *In Proceedings of IEEE Design, Automation, and Test in Europe Conference* (*DATE*), pp. 207–212, 2007.

[Lar07b] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "A Heuristic for Concurrent SOC Test Scheduling with Compression and Sharing," *In Proceedings of IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems* (*DDECS*), pp. 61–66, 2007.

[Lar07c] E. Larsson and J. Persson, "An Architecture for Combined Test Data Compression and Abort-on-Fail Test," *In Proceedings of IEEE Asia and South Pacific Design Conference (ASP-DAC)*, pp. 726–731, 2007.

[Lar08a] A. Larsson, E. Larsson, K. Chakrabarty, P. Eles, and Z. Peng, "Test-Architecture Optimization and Test Scheduling for SOCs with Core-Level Expansion of Compressed Test Patterns," *In Proceedings of IEEE Design, Automation, and Test in Europe* (*DATE*), pp. 188–193, 2008.

[Lar08b] A. Larsson, X. Zhang, E. Larsson, and K. Chakrabarty, "SOC Test Optimization with Compression Technique Selection," *Accepted for publication as a poster at the IEEE International Test Conference (ITC)*, 2008.

[Lar08c] A. Larsson, X. Zhang, E. Larsson, and K. Chakrabarty, "Core-Level Compression Technique Selection and SOC Test-Architecture Design," *Accepted for publication at the IEEE Asian Test Symposium (ATS)*, 2008.

[Lar08d] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "SOC Test Optimization with Test Compression and Sharing," *Submitted to Journal of Electronic Testing: Theory and Applications (JETTA),* 2008.

[Lar08e] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "System-on-Chip Test Planning with Shared Tests," *Submitted to Journal of Electronic Testing: Theory and Applications (JETTA),* 2008.

[Lar08f] A. Larsson, E. Larsson, K. Chakrabarty, P. Eles, and Z. Peng, "SOC Test Planning with Core-Level Expansion of Compressed Test Patterns, " *Submitted to Journal IET Computers & Digital Techniques,* 2008.

[Lar08g] A. Larsson, X. Zhang, E. Larsson, and K. Chakrabarty, "Optimized Test Architecture Design and Test Scheduling with Core-Level Compression Technique Selection for System-on-Chip," *Submitted to IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems,* 2008.

[Lee99] K-J. Lee, J-J. Chen, and C-H. Huang, "Broadcasting Test Patterns to Multiple Circuits*," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.18, No.12, pp. 1793–1802, 1999.

[Lesh04] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher, "Exhaustive Approaches to 2D Rectangular Perfect Packings," *Elsevier Science Direct, Information Processing Letters,* Vol. 90, Issue 1, pp. 7–14, 2004.

[Mar98a] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, H. Lousberg, and C. Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 284–293, 1998.

[Mar98b] K. Marriott and P. J. Stuckey, "Programming with Constraints - An Introduction," *MIT Pres*, ISBN 10: 0-262-13341-5, 1998.

[Mar00] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test," *In Proceedings of IEEE International Test Conference (ITC),* pp. 911–920, 2000.

[Mar02] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 519–528, 2002.

[Mic96] Z. Michalewicz, "Genetic Algorithm + Data Structure = Evolutionary Programs," *3d Edition, Springer Verlag*, ISBN: 3-540-60676-9, 1996.

[Mit93] B. Mitra, P.R. Panda, and P.P Chaudhuri, "Estimating the Complexity ofSynthesized Designs from FSM Specifications," *Design & Test of Computers*, Vol 10, pp. 30–35, 1993.

[Mit05] S. Mitra, M. Mitzenmacher, S. S. Lumetta, and N. Patil, "X-Tolerant Test Response Compaction," *IEEE Design & Test of Computers*, Vol. 22, Issue 6, pp. 566–574, 2005.

[Mou00] S. Mourad and Y. Zorian, "Principles Of Testing Electronic Systems," *Wiley-Interscience*, ISBN: 0-471-31931-7, 2000.

[Mur96] B. T. Murray and J. P. Hayes, "Testing ICs: getting to the core of the problem," *Computer,* Vol. 29, Issue 11, pp. 32–38, 1996.

[Mur00] V. Muresan, W. Xiaojun, and M. Vladutiu, "A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 882–891, 2000.

[Pol03] F. Poletti, D. Bertozzi, L. Benini, and A. Bogliolo, "Performance Analysis of Arbitration Policies for SoC Communication Architectures", *Kluwer Journal on Design Automation for Embedded Systems*, Vol. 8, No. 2, pp. 189–210, 2003.

[Raj04] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded Deterministic Test," *IEEE Transactions on Computer Aided Design*, Vol. 23, pp. 776–792, 2004.

[Roth67] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Transactions on Electronic Computers*, Vol. EC-16, pp. 567–580, 1967.

[Seh04] A. Sehgal, V. Iyengar, and K. Chakrabarty, "SOC Test Planning Using Virtual Test Access Architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 12, Issue 12, pp. 1263–1276, 2004.

[Sem99] Semiconductor Industry Association. "International Technology Roadmap for Semiconductors (ITRS)," *Web site: http://www.itrs.net/reports.html*, 1999.

[Sem01] Semiconductor Industry Association. "International Technology Roadmap for Semiconductors (ITRS)," *Web site: http://www.itrs.net/Links/2001ITRS/Home.htm*, 2001.

[Sem07] Semiconductor Industry Association. "International Technology Roadmap for Semiconductors (ITRS)," *Web site: http://www.itrs.net/Links/2007ITRS/Home2007.htm*, 2007.

[Shi05] T. Shinogi, Y. Yamada, T. Hayashi, T. Yoshikawa, and S. Tsuruoka, "Parellel Core Testing with Multiple Scan Chains by Test Vector Overlapping," *In Proceedings of IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test*, pp. 204–207, 2005.

[Sin03] O. Sinanoglu and A. Orailoglu, "Compacting Test Responses for Deeply Embedded SoC Cores," *IEEE Design & Test of Computers*, Vol. 20, pp. 22–30, 2003.

[Teh05] M. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-Coded Compression Technique for Tesing Embedded Cores in SoCs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*,Vol. 13, Issue 6, pp. 719–731, 2005.

[Tou06] N. A. Touba, "Survey of Test Vector Compression Techniques," *IEEE Design & Test of Computers*, Vol. 23, Issue 4, pp. 294–303, 2006.

[Var98] P. Varma and S. Bathia, "A Structured Test Re-Use Methodology for Core-Based System Chips," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 294–302, 1998.

[Wang05] Z. Wang and K. Chakrabarty, "Test Data Compression for IP Embedded Cores Using Selective Encoding of Scan Slices," *In Proceedings of IEEE International Test Conference (ITC)*, pp. 581–590, 2005.

[Wang07] Z. Wang, K. Chakrabarty, and S. Wang, "SoC Testing Using LFSR Reseeding, and Scan-Slice-Based TAM Optimization and Test Scheduling," *In Proceedings of IEEE Design, Automation and Test in Europe (DATE)*, pp. 201–206, 2007.

[Wil83] T. W. Williams and K. P. Parker, "Design for Testability - A survey," *IEEE Transactions on Computers*, Vol. 71, No. 1, pp. 98–112, 1983.

[Xu04] Q. Xu and N. Nicolici, "Multi-Frequency Test Access Mechanism Design for Modular SOC Testing," *In Proceedings of IEEE Asian Test Symposium (ATS)*, pp. 2–7, 2004.

[Zen06] G. Zeng and H. Ito, "Concurrent Core Test for SOC Using Shared Test Set and Scan Chain Disable," *In Proceedings of Design, Automation and Test in Europe (DATE)*, Vol. 1, pp. 1–6, 2006.

[Zor93] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," *In Proceedings of IEEE VLSI Test Symposium (VTS)*, pp. 4–9, 1993.

[Zor99] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing Embedded-Core-Based System Chips," *IEEE Computer*, Vol. 32, No. 6, pp. 52–60, 1999.

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall:** Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91-7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91-7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91-7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation,1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372-805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov***:* Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies,1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity

of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani**: Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms,1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund**: Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld**: Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén**: Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Langugaes from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis,1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and

Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi**: Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer**: Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-suppor-ted Inter-organisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informa-

tionsystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Emperical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004. ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linquistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modellling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005. ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.

No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.

No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.

No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.

No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.

No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.

No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.

No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.

No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.

No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.

No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.

No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.

No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.

No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.

No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.

No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.

No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.

No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.

No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.

No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.

No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.

No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.

No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.

No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.

No 1139 **Per Wikberg:** Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.

No 1143 **Mehdi Amirijoo:** QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.

No 1150 **Sanny Syberfeldt:** Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.

No 1155 **Beatrice Alenljung:** Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.

No 1156 **Artur Wilk**: Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.

No 1183 **Adrian Pop:** Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.

No 1185 **Jörgen Skågeby:** Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.

No 1187 **Imad-Eldin Ali Abugessaisa:** Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.

No 1204 **H. Joe Steinhauer:** A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.

No 1222 **Anders Larsson:** Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.

Linköping Studies in Statistics

No 9 **Davood Shahsavani:** Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.

No 10 **Karl Wahlin:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN: 978-91-7393-792-4

Linköping Studies in Information Science

No 1 **Karin Axelsson:** Metodisk systemstrukturering- att skapa samstämmighet mellan informa-tionssystemarkitektur och verksamhet, 1998. ISBN-9172-19-296-8.

No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998. ISBN-9172-19-299-2.

No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN-91-7219-606-8.

No 4 **Owen Eriksson:** Kommunikationskvalitet hos in-

formationssystem och affärsprocesser, 2000. ISBN 91-7219-811-7.

No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X

No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.

No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.

No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN91-7373-736-4.

No 9 **Karin Hedström:** Spår av datoriseringens värden - Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.

No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.

No 11 **Fredrik Karlsson:** Method Configuration - method and computerized tool support, 2005, ISBN 91-85297-48-8.

No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.