

Linköping Studies in Science and Technology

Thesis No. 1206

System-on-Chip Test Scheduling and Test Infrastructure Design

by

Anders Larsson

Submitted to Linköping Institute of Technology at Linköping University in partial
fulfilment of the requirements for the degree of Licentiate of Engineering

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

Linköping 2005

System-on-Chip Test Scheduling and Test Infrastructure Design

Anders Larsson



INSTITUTE OF TECHNOLOGY
LINKÖPING UNIVERSITY

ISBN 91-85457-61-2 , ISSN 0280-7971
PRINTED IN LINKÖPING, SWEDEN
BY LINKÖPING UNIVERSITY
COPYRIGHT © 2005 ANDERS LARSSON

Abstract

THERE ARE several challenges that have to be considered in order to reduce the cost of System-on-Chip (SoC) testing, such as test application time, chip area overhead due to hardware introduced to enhance the testing, and the price of the test equipment.

In this thesis the test application time and the test infrastructure hardware overhead of multiple-core SoCs are considered and two different problems are addressed. First, a technique that makes use of the existing bus structure on the chip for transporting test data is proposed. Additional buffers are inserted at each core to allow test application to the cores and test data transportation over the bus to be performed asynchronously. The non-synchronization of test data transportation and test application makes it possible to perform concurrent testing of cores while test data is transported in a sequence. A test controller is introduced, which is responsible for the invocation of test transportations on the bus. The hardware cost, introduced by the buffers and test controller, is minimized under a designer-specified test time constraint. This problem has been solved optimally by using a Constraint Logic Programming formulation, and a tabu search based heuristic

has also been implemented to generate quickly near-optimal solutions.

Second, a technique to broadcast tests to several cores is proposed, and the possibility to use overlapping test vectors from different tests in a SoC is explored. The overlapping tests serve as alternatives to the original, dedicated, tests for the individual cores and, if selected, they are broadcasted to the cores so that several cores are tested concurrently. This technique allows the existing bus structure to be reused; however, dedicated test buses can also be introduced in order to reduce the test time. Our objective is to minimize the test application time while a designer-specified hardware constraint is satisfied. Again Constraint Logic Programming has been used to solve the problem optimally.

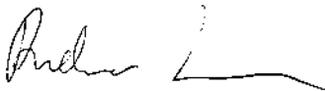
Experiments using benchmark designs have been carried out to demonstrate the usefulness and efficiency of the proposed techniques.

Acknowledgements

SPECIAL THANKS go to my supervisors Zebo Peng, Petru Eles, and Erik Larsson. Without them this thesis would not exist and they truly deserve my deepest respect for their expert knowledge and sharp minds. Their clear guidance has been very important for the results presented in this work.

Thank you all, present and former colleagues, at ESLAB and IDA for all the support and contributions to the nice atmosphere around the office and during the coffee breaks.

I would also like to mention my parents, Birgit and Lars, and my brother Peter. Their support has always been invaluable to me. Last but not least, thank you Caroline for all the joy you have brought to my life.

A handwritten signature in black ink, appearing to read 'Anders Larsson', with a long horizontal flourish extending to the right.

Anders Larsson

Linköping, November 2005.

Contents

Abstract.....	i
Acknowledgements	iii
1. Introduction	1
1.1. Test Process for SoC.....	2
1.2. Problem Formulation.....	4
1.3. Contributions.....	5
1.4. Thesis Overview	6
2. Background and Related Work.....	7
2.1. Reuse-based SoC Design.....	8
2.2. Test Challenges	11
2.3. SoC Test Access.....	12
2.3.1.Direct Access.....	12
2.3.2.Bus-based Access.....	13
2.3.3.Functional Access.....	15
2.4. Test Scheduling.....	16
2.5. Test Set Sharing.....	18
2.6. Constraint Logic Programming.....	20
2.7. Optimization Heuristics.....	22

3. Preliminaries	27
3.1. System Architecture	28
3.2. Test Access Mechanism.....	29
3.3. Test Scheduling	32
4. Buffer and Control-logic Minimization	35
4.1. System Architecture	36
4.2. Motivational Example	40
4.3. Problem Formulation	42
4.4. Constraint Logic Programming Modelling.....	45
4.5. The Tabu Search Based Algorithm.....	46
4.6. Experimental Results.....	48
4.7. Conclusions	53
5. Test Scheduling with Test Set Sharing and Broadcasting	55
5.1. Test Set Sharing	56
5.2. System Architecture	61
5.3. Motivational Example	64
5.4. Problem Formulation	67
5.5. Constraint Logic Programming Modelling.....	69
5.6. Experimental Results.....	71
5.7. Conclusions	76
6. Conclusions and Future Work.....	77
6.1. Conclusions	77
6.2. Future Work	78
References	81

Chapter 1

Introduction

THIS THESIS DEALS WITH testing of core based System-On-Chip (SoC). SoC is a technology that enables the electronic industry to provide the market with high complexity products such as mobile phones, PDAs, and DVD-players, which have a short time to market window, and hence require a short development time. The development time can be reduced if a reuse-based design methodology is used. When designing a SoC different pre-designed and pre-verified blocks of logic called cores are integrated into a system. The cores are reused or bought from core vendors and can, for instance, consist of a processor, a memory block, or an I/O peripheral. A core is usually delivered with a test set that has to be applied to the core in order to make sure that the core is free from manufacturing defects. This first chapter is used to motivate and introduce the problems of core-based testing of SoCs as well as the main contributions of this thesis.

1.1 Test Process for SoC

The trend in integrated circuit (IC) design is to build more and more complex systems and integrate them on single-chip packages. This is facilitated by the improvement in semiconductor manufacturing technology. The trend that Moore predicted, which says that the number of transistors per chip doubles every 18 months [Moo65], has now lasted for four decades. On the other hand, the number of input and output pins, so called chip terminal pins, on a chip has not been able to keep up with this trend, which means that the amount of logic that is reachable from each terminal has grown. This trend has led to many test related problems since the amount of test data, which has to be applied to the chip, is becoming very large. It has been shown that up to 40-50% of the total production cost is today spent on testing ICs for manufacturing defects [Int03].

The main purpose of testing is to verify the correctness of the manufacturing process. Testing is performed by applying test stimuli, consisting of a set of test vectors, to the chip and then comparing the test responses with the expected responses, as illustrated in Figure 1.1. The test stimuli are usually applied by using a special machine called automatic test equipment (ATE). The same machine is used to compare the test responses with the expected ones. A difference between the two responses indicates that a fault has been detected on the IC.

There are two main approaches for testing embedded cores: Built-in self-test (BIST) and applying and observing pre-computed test sets at the core terminals. In the case of BIST, an on-chip engine is responsible for generating the test stimuli and for observing the responses. Although BIST obviously simplifies the test task for the system integrator, there are several limitations: Most cores are not suitable for BIST, i.e. they contain logic that cannot be tested by randomly generated tests,

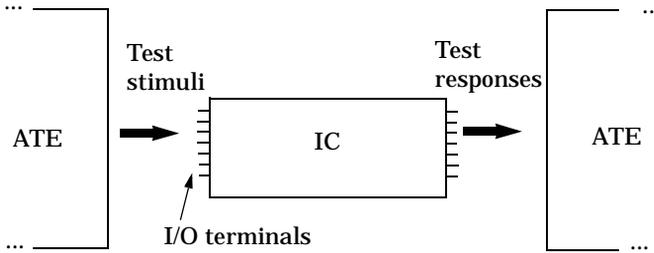


Figure 1.1: Test process.

which are usually assumed in BIST. Another drawback is the large design effort needed to achieve sufficient fault coverage for large cores.

The second approach is to use pre-computed test sets, provided usually by the core vendors, which are applied and observed via the core's terminals. The core vendor has little or no information about where their cores will be placed on the chip. Therefore they assume that all core terminals are directly accessible. The task of the system integrator is to ensure that the logic surrounding the core allows the tests to be applied, i.e. designing a suitable test access mechanism (TAM).

This test infrastructure can be implemented using, if possible, direct access between the chip I/O and the core terminals. If direct access is not possible, functional access can be used, where wires used in functional mode, such as a system bus, are utilized for test transportation during test mode. Another approach is to use various types of dedicated bus-based access mechanisms.

Irrespective of the approach used, BIST or pre-computed test sets, it is usually required that the order of the test application (the start times) are determined in such a way that the cost function for the test process is minimized. This process is called test scheduling and the cost function minimized is often related

to the test application time and/or the hardware overhead introduced from, for instance, BIST logic and wires, such as routing of a dedicated test bus.

1.2 Problem Formulation

In this thesis two test architectures are proposed, and two respective optimization problems are formulated. First, a buffer-based architecture that makes use of the existing functional bus structure for test data transportation is presented. The scheme adds buffers at each core and an additional controller. The buffers are inserted at each core allow test application to the cores and test data transportation over the bus to be performed asynchronously. This makes it possible to perform concurrent testing of cores while test data is transported on the bus. Both the buffers and the controller contribute to an increased silicon area, and the problem we solve in this thesis is to minimize this hardware overhead, without violating a given maximum test time constraint.

The second approach we propose also allows the existing functional bus structure to be reused for the test data transportation. However, in order to decrease the test time, dedicated test buses may be added to the design. To further reduce the test time the possibility to share test vectors from the individual cores tests in the system has been explored. The shared tests, if selected, are transported to the cores in a broadcasted manner so that several cores are tested concurrently. The problem, in this case, is to select appropriate tests for each core, insert test buses (if required), 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.

1.3 Contributions

In this thesis, the problem of testing core based SoCs is addressed. Two test architectures have been proposed and their respective optimization problems have been solved.

In the first architecture, buffers are introduced between each core and the functional bus and a test controller is introduced which is responsible for the invocations of tests [Lar03a], [Lar05a]. The hardware overhead introduced due to the buffers and the test controller is minimized such that a given test time is not exceeded. This problem has been solved by using a Constraint Logic Programming (CLP) formulation and a tabu search based heuristic. The experimental results, obtained using the proposed approach, shows a decrease with 27 to 55% of the cost, compared to the cost obtained using a straightforward approach. Since CLP uses an exhaustive search approach, optimization times can become exceedingly large for complex systems. The tabu search based heuristic, which is implemented to overcome the problem of long execution times, produces high quality results compared with the optimal solutions obtained using CLP. The experimental results shows that the proposed heuristic produced results which were only less than 6.1% worse than those produced by the CLP-based approach.

In the second test architecture [Lar05b], the use of test sharing and broadcasting of test vectors for core-based SoCs are proposed. The possibility to merge tests by using shared test vectors is explored. We have used a CLP technique to select suitable tests for each core in the system and schedule the selected tests such that the test application time is minimized while designer-specified hardware cost constraints are satisfied. The experimental results indicate that we can, on average, reduce the test application time with 23%.

1.4 Thesis Overview

The rest of the thesis is structured as follows. Chapter 2 gives background information and description of related work regarding core-based SoC design and test access mechanisms. The concepts of test scheduling and test set sharing are also introduced in this chapter, together with CLP and optimization heuristics.

Chapter 3 contains preliminaries that are specific for this work, such as the system model used, the test access architecture, and the test scheduling technique.

Chapter 4 describes the first test architecture and a motivational example is presented together with the formal problem formulation. Hardware overhead minimization under constrained test time has been modelled and solved using both CLP and tabu search.

Chapter 5 introduces the second proposed test architecture. Test set sharing and merging is described together with the test broadcasting technique with the possibility of adding dedicated test busses. The test time minimization problem is formulated and then solved using CLP.

Chapter 6 concludes this thesis and discusses different possible directions of future work.

Chapter 2 Background and Related Work

THIS CHAPTER INTRODUCES the reuse-based SoC design flow. The major test challenges that the test designer faces are introduced and described together with the concept of test scheduling, test set sharing, and broadcasting. The test scheduling problems can be solved by using an exhaustive approach such as CLP, or by using a heuristic. An introduction to CLP and different optimization heuristics is given at the end of this chapter. Related work in the discussed areas is also presented.

2.1 Reuse-based SoC Design

The complexity of ICs has grown tremendously during the last decades and this has forced the system integrators to think in a different way when designing their products. Short time to market has put a lot of pressure on designers under these circumstances. Designing a chip with millions of gates using conventional methods would be too time consuming and the time to market window may be missed. SoCs and reuse-based design techniques have been proposed in order to solve this problem. With the SoC approach the number of components that have to be considered at design time decreases dramatically. At the same time, the number of components to be mounted is reduced, and this significantly reduces both the physical size and the time to assemble the final product.

The system integrator is the person that designs the system and decides which components that should be used in the design. The development of a SoC is in many ways similar to the development of a System-on-Board (SoB) (Figure 2.1). 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 SoC methodology, system integrators have adopted the same reuse-based philosophy to license or use in-house pre-defined cores, which are integrated into a system. The cores are intellectual property (IP) and can consist of VHDL code or of a synthesized format such as GDSII [Rub86]. An example of a SoC is illustrated in Figure 2.2, which shows how different types of cores, such as memories (ROM, DRAM and SRAM), processors (DSP), and several ASICs (RF, MPEG, Peripherals) are integrated in one design.

One of the major differences between developing a SoB and a SoC is the way testing is performed. This is illustrated in

Figure 2.1 where the testing in the development process is shown for SoB in Figure 2.1 (a) and for SoC in Figure 2.1 (b).

In the SoB development process, all chips and components are manufactured and tested before they are mounted on the printed circuit board. Finally, after the mounting, the interconnections between the components on the board are tested.

Figure 2.1 (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

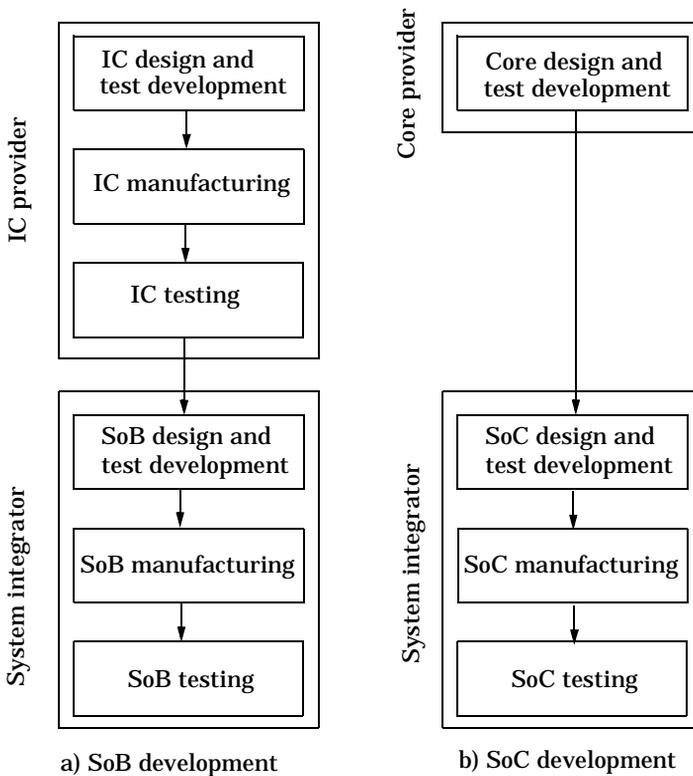


Figure 2.1: Difference between SoB and SoC development and testing [Mar99a].

is manufactured in a single step. This entails that the testing has to be postponed until all cores are integrated and connected and the chip is fabricated. This means that all the test data have to be applied at one time through the I/O terminals of the chip.

One of the problems that the test designer has to face is therefore how to transport test data inside the system. Zorian et al [Zor99] proposed a conceptual infrastructure for SoC test, as illustrated in Figure 2.2. The test *source* generates test stimuli and can be placed on-chip (LFSR, counter, or ROM) or off-chip (ATE). The test *sink* compares the test responses to the expected ones and can also be placed on-chip (signature analyzer) or off-chip (ATE). The TAM, which is always implemented on-chip, is used to transport the test from the source to the core under test (CUT) and from the CUT to the sink. The TAM can either consist of wires dedicated only for testing or of functional wires which is used both for transporting functional and test data.

The core test *wrapper* is an interface used to facilitate plug-and-play between the functional connections and/or the TAM. Different wrapper designs of the wrapper have been proposed. Marinissen et al. [Mar98] proposed a core test wrapper called *TestShell*, and Varma and Bhatia [Var98] described a wrapper called *TestCollar*. The IEEE 1500 wrapper ([IEE05], [DaS03], and [Zor97]), has widely been adopted in literature.

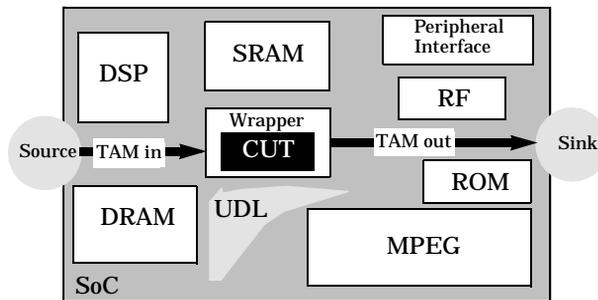


Figure 2.2: Cores and test infrastructure [Zor99].

2.2 Test Challenges

There are several challenges that the test integrator faces when designing the test strategies for a core based SoC. The main objective is to minimize the cost, C_{test} , introduced in testing. The testing should have a short application time without introducing additional hardware on the chip and still have high quality (high fault coverage). The overall test cost can be formulated as a sum of different cost components, and can be computed as follows [Nag02]:

$$C_{test} = C_{prep} + C_{exec} + C_{silicon} + C_{quality} \quad (1)$$

Here, C_{prep} is a fixed cost of test generation and preparation, and can, for instance, consist of the cost for test generation and test program preparation. C_{exec} is the cost of executing the test and is mainly dependent on the price of the test equipment and the time each chip has to spend in the ATE. $C_{silicon}$ represents the cost of the additional test hardware on the chip. Finally, $C_{quality}$ represents the cost of performance degradation during normal functionality, for instance, by introducing test points on critical paths, and test escapes.

In order to have a low test cost the test designer must try to minimize all these components. However, different trade-offs may be considered. For instance, lowering the execution time by introducing additional wiring for the transportation of the tests may reduce C_{exec} since multiple cores may be tested concurrently. However introducing additional wiring would increase the area overhead, $C_{silicon}$. The shortest test application time would be obtained if all cores would be tested in parallel, this however is almost always impossible in practice since it would require large overhead due to additional wiring and the risk of burning the chip due to that the amount of consumed power will be very large. This is why it is required to schedule the tests such that the required constraints are satisfied. In this thesis, our focus will be on the problem of lowering the cost of

C_{exec} (test application time) and $C_{silicon}$ (logic and wiring dedicated for testing).

During the following two sections, the TAM and test scheduling, which are most relevant for this thesis, will be described in more detail.

2.3 SoC Test Access

The need of a TAM has its origin in the requirement to transport test stimuli from the test source to the core under test (CUT) and of test responses to the sink. There are a number of different solutions proposed in literature that can be used for accessing the CUT, with direct access, dedicated bus-based access, and functional access, as examples. In this section, a brief description of these three architectures is presented.

2.3.1 DIRECT ACCESS

Direct access is a straightforward solution where the core terminals are directly connected to the chip level pins, which makes it possible to test the core as if it was the chip itself [Imm90]. The implementation of direct access is illustrated in Figure 2.3 where four cores are connected to the chip level pins P_{in} and P_{out} .

One of the drawbacks of this technique is the amount of wiring overhead, introduced due to the large number of cores and core terminals in SoCs. Usually the total number of the core terminals exceeds the number of chip terminals. Another drawback is that it does not provide access to the logic that is not directly accessible from the SoC I/O terminals. To solve this problem Bhatia et al [Bha96] proposed a method, CoreTest, where test points are introduced in the user defined logic (UDL), between the cores, to provide sufficient access. This is done by adding a grid of test points, which are placed around the cores. Even if the UDL logic can be reused with this method, it

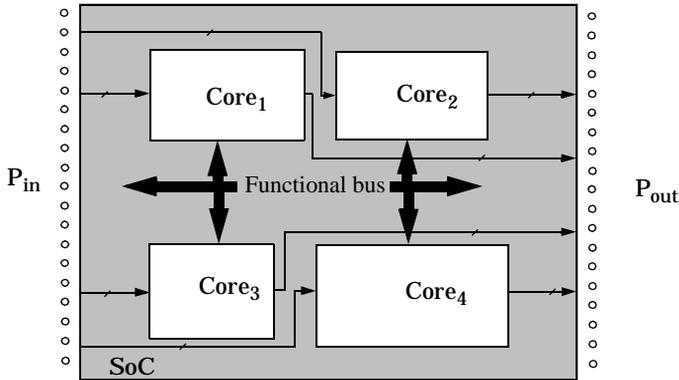


Figure 2.3: Example of direct access.

introduces hardware overhead due to both the required routing of the test grid and the additional logic in the testpoints. The main disadvantage is the disparity between the number of core terminals and chip level pins.

2.3.2 BUS-BASED ACCESS

A dedicated bus-based access mechanism can consist of one or several buses that are connected to all cores in the SoC as shown in Figure 2.4, where two dedicated test buses, bt_1 and bt_2 , are used to transport test data.

The main characteristic of a bus-based TAM is the sequential communication where each bus usually is capable of transporting test data to only one core at a time. Testing several cores concurrently entails that multiple buses are implemented or that it is possible to partition the bus wires to be connected to different cores. The dedicated bus-based access mechanism has been widely used in literature due to the modularity, and flexibility on trading off different cost factors offered by this method [Xu05].

Aerts and Marinissen [Aer98] described three different bus-based TAMs. A *multiplexed* architecture gives each core access to the full TAM width. This means that only one core can be tested at a time, which leads to long test time. The second architecture proposed in [Aer98] is called *Daisychain*, and it connects all cores through one long TAM. Bypasses have been introduced to enable parallel access to multiple cores, which can reduce the test time compared to the multiplexed architecture. In the third approach, called *distribution* architecture, the TAM bandwidth has been distributed to all the cores and each core is assigned dedicated TAM lines. In this architecture, all cores are tested concurrently and the test time for the SoC is the maximum of the individual core test times.

Varma and Bathia, proposed a combination of a multiplexed and distributed architecture, called *Test Bus* [Var98]. In this work, several multiplexed buses can be used, which then operate independantly from each other and hence allow tests to be applied concurrently, however, cores connected to the same Test Bus can only be tested sequentially.

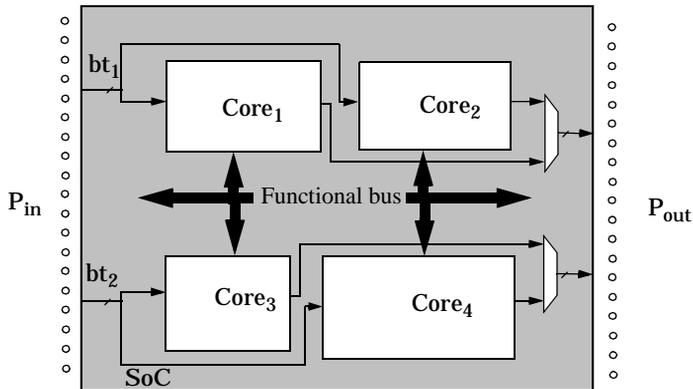


Figure 2.4: Example with two dedicated test buses.

2.3.3 FUNCTIONAL ACCESS

As opposed to the two previous methods (the direct access and the dedicated bus-based access), the functional access methodology makes use of the functional wiring, such as the functional bus or other existing functional paths between the cores, for transporting test data. Figure 2.5 illustrates the use of the functional bus for transporting tests. A special bus interface is introduced between the core and the bus in order to enhance the application of tests. One of the advantages with this method is the reduction in wiring overhead.

The Macro Test, introduced by Beenker *et al.* [Bee86], was originally developed to improve test quality by using different strategies to test different circuit architectures such as memories, PLAs, and register files. Although not originally intended, the Macro Test was later shown to work in the core based SoC domain by Marinissen and Lousberg [Mar97].

Another technique, proposed by Ghosh *et al.* [Gho00], is based on transparency of the cores in the system. The technique consists of two parts. The first part consists of core-level design

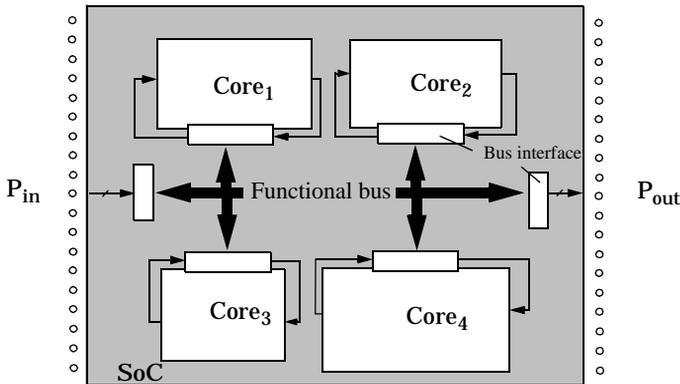


Figure 2.5: Example of functional access.

for test and test generation to make the core testable and transparent. The second part consist of chip level deign for test where the cores interconnects are analyzed. The technique can be used to design a SoC such that, the test area overhead is minimized, the test application time is minimized, or a desired trade-off between the test area overhead and the test application time. This approach requires a large design effort at the core provider side, since they assume that the core providers offer a catalogue of area/latency versions for their cores.

Harrod demonstrated the use of the AMBA bus for test purpose [Har99]. In the test mode the AMBA test interface becomes the bus master and is responsible for applying the test stimuli and to capture the responses. In functional mode, the bus is used to transport functional data between the cores in the system.

2.4 Test Scheduling

When the type of TAM is determined the test integrator is faced with another problem, namely in which order the tests for the different cores should be applied.

Test scheduling means that the start time of each test is determined and is usually done in order to minimize some predefined cost function, which often is related to the total test application time. By exploring different start times for each test it is possible to minimize the cost function while ensuring that constraints, such as power consumption and/or hardware overhead, are not violated, as illustrated in Figure 2.6. Different trade-offs may be considered during the test scheduling. For example, testing several cores in parallel can usually decrease the test time; however it will also increase the power consumption, which potentially will damage the chip [Zor93].

To further explain the scheduling process, a design example consisting of four cores is used (Figure 2.7). The system is tested

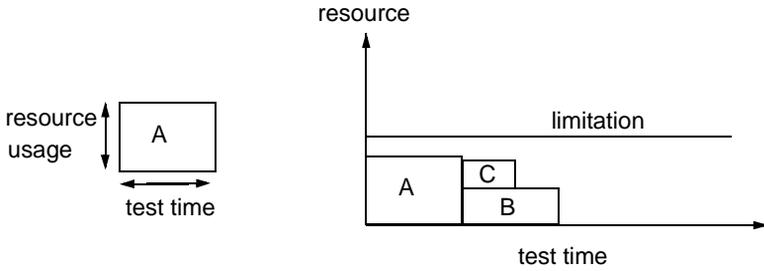


Figure 2.6: Illustration of test scheduling [Lar05c].

by applying the tests $\{Test_1, Test_2, Test_3, Test_4\}$ to the cores, $\{Core_1, Core_2, Core_3, Core_4\}$, where $Test_1$ is used to test $Core_1$, $Test_2$ to tests $Core_2$, and so on. It is assumed that each core has four scan chains, which are connected to a bus-based TAM as illustrated in Figure 2.7. The TAM is designed in such a way that it is possible to apply test stimuli to any two cores in parallel.

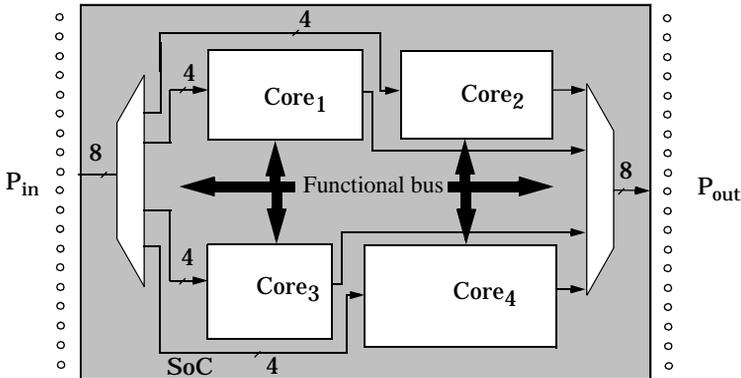


Figure 2.7: Example of dedicated bus-based access.

In this example, it is assumed that the cost function consists of the test application time, which will be minimized without violating the hardware constraint given by the maximum number of TAM wires.

The scheduling techniques can be divided into [Lar02]:

- *Non-partitioned* testing,
- *Partitioned* testing with run to completion, and
- *Pre-emptive* testing.

The three scheduling techniques are illustrated in Figure 2.8. Figure 2.8(a) shows an example of a schedule using a non-partitioned (session based) technique, proposed by Zorian [Zor93], Chou *et al.* [Cho97], and Larsson and Peng [Lar01a]. In non-partitioned scheduling no new test is allowed to start until all tests in a session are completed. This method produces long test time due to a lot of idle time.

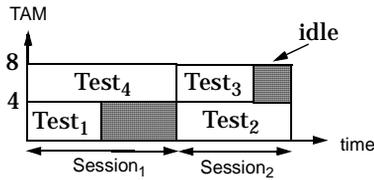
Figure 2.8(b) shows how the schedule can be improved by using a partitioned (sessionless) technique proposed by Chakrabarty [Cha01], Muresan *et al.* [Mur00], and Larsson and Peng [Lar01a]. In this technique, each test can be scheduled as soon as possible, which can decrease the test time.

In order to further optimize the schedule, a pre-emptive scheduling technique can be used. Such technique has been proposed by Iyengar and Chakrabarty [Iye01], Larsson and Fujiwara [Lar02], and Larsson and Peng [Lar03b]. The pre-emptive scheduling is illustrated in Figure 2.8(c). Here, the test intended for core 4 is pre-empted and then resumed at a later point in time using different TAM wires.

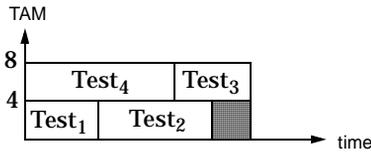
2.5 Test Set Sharing

By sharing one test among several cores it is possible to shorten the test application time significantly [Lee99], since cores that share tests can receive test vectors concurrently using shared TAM resources. The efficiency of this test set sharing method

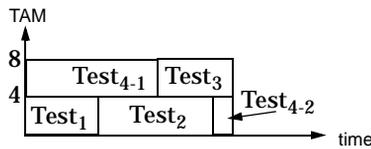
BACKGROUND AND RELATED WORK



a) Non-partitioned



b) Partitioned



c) Pre-emptive

Figure 2.8: Different test schedules for the example given in Figure 2.7.

depends on how large the merged test set is in comparison with the unmerged sets. It has been shown that the percentage of don't cares in the test sets is high (78% for un-compacted or approximately 47% even for compacted tests [Kaj01], [Cha03]). A high number of don't care bits increases the possibility of finding vectors from different sets that can be merged.

One important requirement that has to be fulfilled in order for the test set sharing method to be efficient, is that the size of the

new combined test set is smaller than the sum of the original test sets used [Lee98].

In order to benefit from the advantages of test sharing it is also required that the cores which shares the tests are connected in such a way that the tests can be broadcasted to them. How this broadcast of test vectors can be done has been shown by [Lee99], [Jia03]. The work by Lee *et al.* [Lee99] considered the circuits driven by all scan chains as one circuit during the ATPG process. In this way they can generate one single test that can be applied to all circuits simultaneously. In [Jia03], a method for generating common tests for multiple cores directly is proposed. A fault simulator is used for evaluating the fault coverage. The tests generated by the method in [Jia03] can also be utilized by the technique described in this thesis.

One additional problem that occurs when using a broadcast method is how to transport the test responses. The responses from different cores cannot be merged in the same way as the input stimuli since the opportunity to detect a fault may be lost. In [Lee98] and [Jia03] this problem has been solved by compressing the test responses by using a multiple input signature register (MISR).

2.6 Constraint Logic Programming

CLP has been introduced in the middle of the 80s, by Jaffar and Lassez [Jaf87]. CLP is a combination of logic programming and constraint solving and is suitable for solving optimization problems like scheduling, resource planning, and layout assignment, which all are examples of constrained search problems. CLP is a declarative method where the programmer describes the program in terms of conditions and relations and leaves the order of execution and assignment of variables to the solver. In short, the programmer specifies what should be solved, instead of how it should be solved.

To further explain how CLP works, let us consider the following small example (from [Mar99b]). 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 2.9. The program will determine that:

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

which is the first solution for this problem, found by the solver. This example 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

```

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 = 10000*M + 1000*O + 100*N + 10*E + Y.
```

Figure 2.9: A CLP example [Jaf87].

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, durations, 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 2.9.

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.

CLP has been used by Bieker and Marwedel [Bie95] to generate test programs for embedded processors. Zeng *et al.* [Zen01] uses CLP to generate functional test vectors.

2.7 Optimization Heuristics

In contrast to the CLP methodology, heuristics produce approximate solutions to optimization problems. They often build on a strategy of local search, where an initial feasible solution is iteratively improved by applying local modifications, so called moves, which slightly changes the solution. The *search space* consists of all possible solutions that can be considered during the search. The *neighbourhood* structure is a subset of the search space, which can be obtained by applying a single local *move*. The search is stopped if no further improvements can be made. Often, this local search produces a solution which

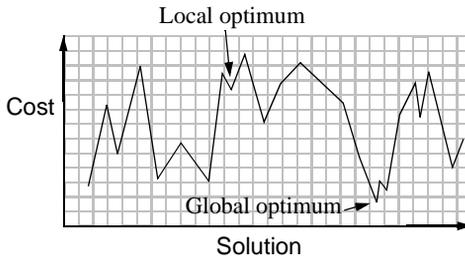


Figure 2.10: Global and local optimum for a minimization problem.

is a local minimum, that can be far from the global optimum, as illustrated in Figure 2.10. One of the main challenges when implementing a heuristic is to provide the ability to avoid to be trapped in such local minima.

In contrast to an exhaustive method, such as CLP, a heuristic searches only a very small 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.

There exists a vast variety of different optimization heuristics and many of them are developed to solve problem specific optimizations 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]. Here follows a brief description of these three heuristics.

Simulated Annealing

Simulated annealing (SA) [Kir83] has been shown to produce often close to optimal solutions to combinatorial problems, although not always in reasonable computing time.

The annealing process, which gives the name to this heuristic, is a technique used to reduce the defects in metals by using controlled cooling. The goal is to minimize the internal energy state of the metal. This is done by heating the material past its

melting point to cause atoms to move from their initial positions. During this phase the atoms randomly move around in the material, which corresponds to states with higher energy. The material is then slowly cooled back into a solid state with lower internal energy than the initial one. Slow cooling will lead to a low energy state and fast cooling to a high energy state, hence there is a trade-off between the initial temperature, the time spent during cooling and the quality of the result.

In order to apply the SA optimization, the following elements must be provided. A representation of all feasible solutions (the solution space). An annealing schedule with an initial state (temperature) and neighborhood structure that represents the different moves that can be applied to the current solution. A generator of random changes is also required. It is used at each iteration in the algorithm where a random move is allowed.

A random move is always accepted if it is associated with an improvement. A move, associated with a non-improvement state, is accepted only with a certain probability. This probability is reduced, meaning that it is unlikely to accept a non-improvement move at the end of the search. The probability corresponds to the temperature in the annealing process. At the beginning of the execution the temperature is high and large modifications are possible, while as the temperature decreases smaller and smaller modifications are allowed. When the user-defined stopping condition is met the best solution found so far is returned.

Tabu Search

Glover proposed in 1986 a new approach, called tabu search, that would overcome the problem of local optima. The main idea is to avoid local optima by accepting non-improving moves. The cyclic behavior, 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 can 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.

Genetic Algorithms

Inspired by evolutionary biology, genetic algorithms [Mic96] offer another approach to solve combinatorial optimization problems. Techniques such as inheritance, mutation, natural selection, and recombination (or combinations of these techniques) are applied to a population of candidate solutions, called individuals, forcing the optimization problem to evolve toward a better solution.

Solutions are traditionally represented by a string of '1's and '0's and the evolution starts from a population of completely random individuals. In each iteration (generation) the fitness of the population is evaluated and multiple individuals are stochastically selected and modified by mutation or recombination, to form a new population. There are a number of parameters such as mutation probability and recombination probability that the designer may use to tune the algorithm and guide the selection.

Application of Heuristics

Simulated annealing has been used by Larsson *et al.* in [Lar01b] where the test scheduling and TAM design are minimized. A tabu search based heuristic is proposed to optimize a hybrid BIST test set by Jervan *et al.* in [Jer02]. In the work done by Ebadi and Ivanov [Eba01] a genetic algorithm is used for test access architecture design and test time minimization.

CHAPTER 2

Chapter 3

Preliminaries

THE PURPOSE OF THIS CHAPTER is to introduce the background information of the rest of this thesis. In the first section, the system model used is explained, and a detailed description of how a test is applied to a core is given. It is assumed that the system consists of a number of cores that are connected to at least one functional bus. 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 test vector should be applied.

In the second section, the TAM architecture is presented, and a description of how the test vectors are transported to the cores is given. An example that illustrates the advantage of using the functional bus is also given in this section. In the third section, the schedule of tests, using the proposed architecture, is discussed.

3.1 System Architecture

In this thesis, we assume that the system under test consists of a number of cores which are connected to at least one functional bus, as illustrated in Figure 3.1. The bus-based architecture has been selected since it is the most widely used interconnect architecture for SoCs [Pol03] and several commercial buses have been developed such as CoreConnect from IBM [Cor05] and the Advanced Microcontroller Bus Architecture (AMBA) from ARM [Amb05]. The system is tested by applying to the cores a number of tests which are produced by the core providers. Also connected to the bus are the following two test components: the *source* where the test stimuli are generated and the *sink*, which is used to compare the test responses to the expected responses.

The design example depicted in Figure 3.1 consists of four cores, $Core_1$, $Core_2$, $Core_3$, and $Core_4$, which are connected to a functional bus, bf_1 . The test stimuli are transported from the *source* on the bus, to the core, through the input test pins, t_{in} . When they have been applied, the test responses are transported to the *sink* through the outputs, t_{out} . Both during normal operation (functional mode) and during testing mode,

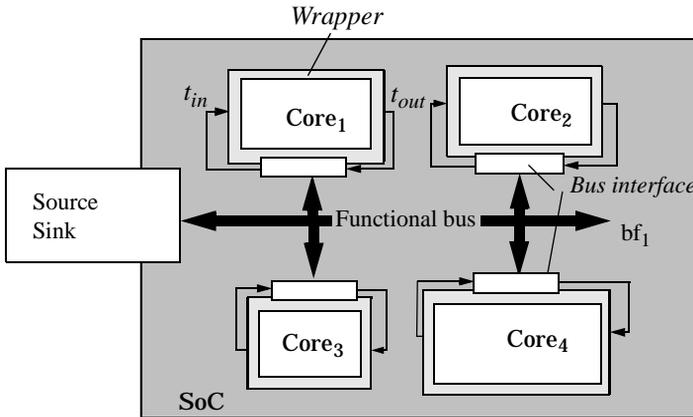


Figure 3.1: Bus-based system.

the inputs and outputs of each core are connected to the functional bus. However, when the system is in the testing mode the bus interface will receive control signals indicating when a vector should be applied. Test data is then transported to the wrapper chains.

The cores are assumed to be equipped with scan and to facilitate interfacing with the functional bus, each core has a wrapper. The scan elements at a core (scan-chains and wrapper-cells) are connected into wrapper chains, which are connected to a bus as illustrated in Figure 3.2. In this example, the core $Core_1$ from Figure 3.1 is connected to a functional bus, bf_1 . The three scan chains (sc_1 , sc_2 , and sc_3), which are of equal length ($l=9$), are connected to the bus wires.

The core $Core_1$ is tested by applying the test T_1 which consists of N test vectors, $\{v_1, v_2, \dots, v_N\}$ as illustrated in Figure 3.2 where test vector v_1 is applied to the cores three scan-chains, sc_1 , sc_2 , and sc_3 . The bus interface will receive a control signal, t_{ctrl} which indicate that the data received from the bus is a test and should be applied to the wrapper chains.

3.2 Test Access Mechanism

In this work we consider the bus-based TAM architecture. The proposed method allows the existing functional bus structure to be reused for test data transportation. The advantage of using the existing functional bus structure is that the amount of additional, dedicated test wiring is reduced. This can be illustrated by considering the example in Figure 3.3. It consists of two cores, $Core_1$ and $Core_2$, which are tested by two tests, T_1 and T_2 , respectively.

Short application time of scan patterns entails a concurrent scan-in and scan-out phase, that is, when one scan vector is shifted out, a new vector is shifted in. Therefore it is not possible to share the same bus wires for the test stimuli and test

	sc ₁	sc ₂	sc ₃
scan bit	9 ... 3 2 1	9 ... 3 2 1	9 ... 3 2 1
test vector v ₁	1 ... 1 1 0	0 ... 1 1 0	0 ... 1 0 1
v ₂	0 ... 1 0 0	0 ... 1 0 0	0 ... 0 1 0
⋮	⋮	⋮	⋮
v _N	1 ... 0 1 0	1 ... 1 1 1	1 ... 0 0 0

T₁

— Functional connection

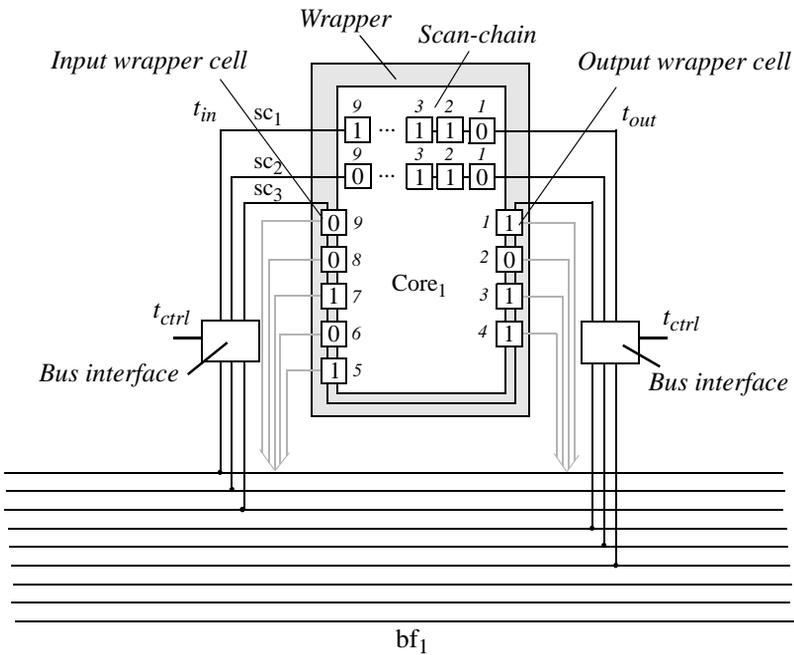
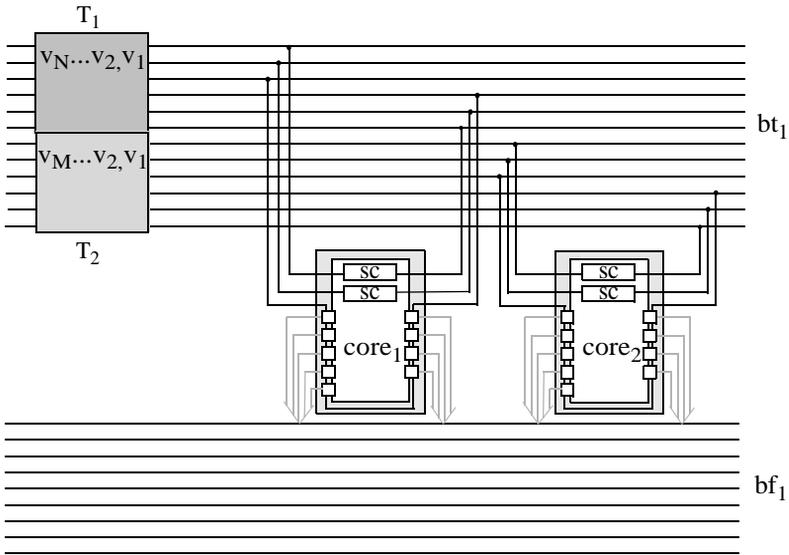


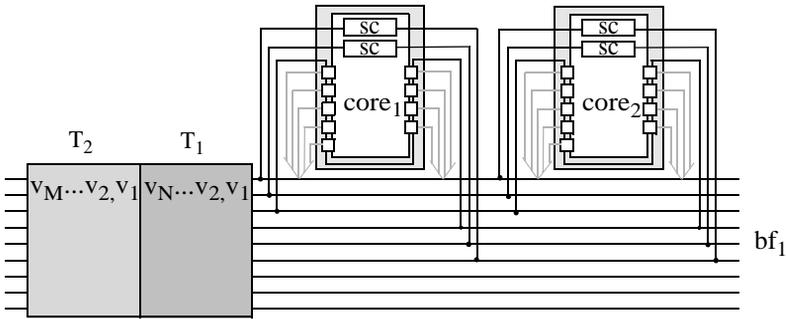
Figure 3.2: Scan testing.

responses of one core; the total number of bus wires used to test one core is twice the number of scan chains of the core. Both cores in the example in Figure 3.3 have three scan-chains each, which means that six wires are needed during the test transportation.

PRELIMINARIES



a) Test transportation on a dedicated test bus



b) Test transportation on a functional bus

Figure 3.3: Test transportation.

In Figure 3.3(a) a dedicated test bus bt_1 is used for the transportation of the test data. The test data are transported in parallel, which is possible since there are dedicated wires for

each scan-chain (one wire is used for shifting in the input stimuli and one wire is used for shifting out the responses). Note, that the functional bus is not utilized during testing in this design. In Figure 3.3(b), the tests are transported on the functional bus bf_1 and no dedicated test bus is used. The time for transporting the tests on the functional bus is larger since only one test can be transported on the bus at a time. To give exclusive access to the bus for both cores at the same time, and hence make it possible to apply the test to the cores concurrently, would significantly decrease the test time.

3.3 Test Scheduling

Using a bus for transporting test data entails usually a sequential schedule, and hence, only one core is tested at a time, as illustrated in Figure 3.4. The transportation of tests on the functional bus is shown in Figure 3.4 (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 3.4 (b)). This makes the scheduling very simple. The drawback, however, is the long test time obtained since the cores are not tested in parallel.

The following two chapters, Chapter 4 and Chapter 5, contain different solutions on how the sequential bus based test architecture can be improved in order to obtain test parallelism in the testing of cores. In Chapter 4, a buffer is introduced between the cores and the bus, which is used to temporarily store the test data during test application. In Chapter 5, a connector is used to connect the core test terminal to the functional bus wires and in contrast to the functional connection, where the full bus bandwidth often is assigned to each core, this method proposes an architecture where only a portion of the bus bandwidth is assigned to a core during testing mode. The advantage is that only the required bandwidth is

PRELIMINARIES

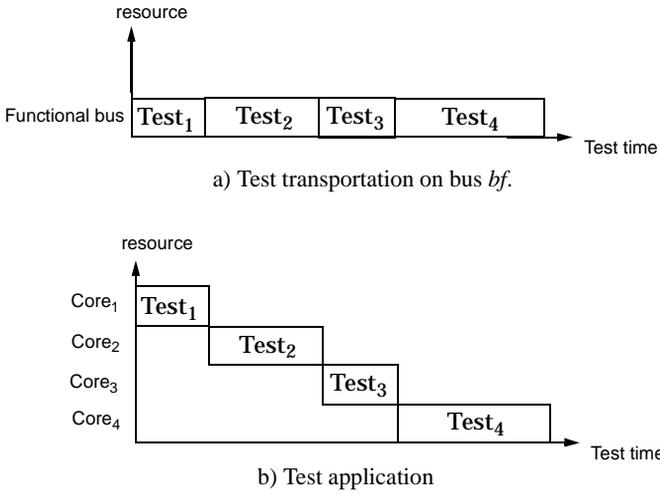


Figure 3.4: Sequential scheduling and application.

used during testing and the wires not utilized for a certain core can be used by another core at the same time. Furthermore, additional test buses can also be introduced to increase the test concurrency, whenever needed, to shorten the test application time.

CHAPTER 3

Chapter 4

Buffer and Control-logic Minimization

THE PURPOSE OF THIS CHAPTER is to describe a test architecture in which a buffer is inserted between the functional bus and each core and the tests are divided into packages. It means that tests can be applied concurrently even if the bus only allows sequential transportation. The additional hardware overhead, introduced by the proposed method, should be minimized without exceeding a, designer specified, test time constraint. The problem has been modelled and optimally solved using CLP. The results demonstrate that the cost is decreased compared with a straightforward approach.

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 from the CLP approach. The results indicate that the technique produces high quality solutions at low computational cost.

4.1 System Architecture

The example in Figure 4.1 shows a system consisting of three cores, $core_1$, $core_2$, and $core_3$, all connected to the functional bus bf . Each core, $core_p$, 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. It is assumed that the core itself handles the evaluation of the test 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.

The proposed method is based on the assumption that the test application of a core takes longer time than the test data transportation. One motivation for this assumption is that the

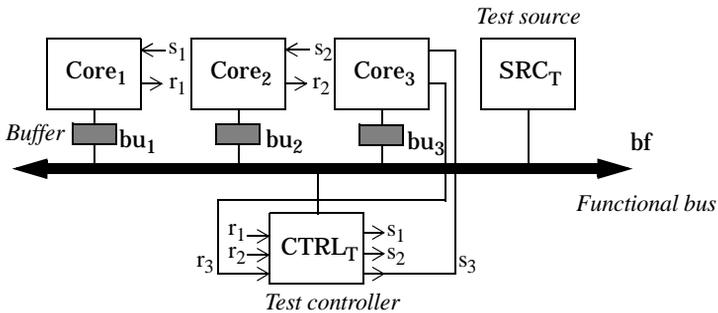


Figure 4.1: Bus-based architecture.

number of scan chains in the cores usually is smaller than the bandwidth of the bus, thus making it possible to transfer more test data per clock cycle on the bus than what can be applied to the core. This difference is further illustrated with a small example (Figure 4.2). Here a 128 bit wide ($w=128$) functional bus is connected to a $core_i$, with four scan chains through a buffer. In only one clock cycle of the bus, the buffer is fed with 128 bits of test data. The test vector is partitioned through a parallel to serial converter, to four scan chains, each with the length of 32 bits. During the next cycle, the bus can transport data to another core, $core_j$ while $core_i$ is occupied for another 32 clock cycles with the shift-in of the scan chains.

In order to make this approach more efficient the test set for each core is divided into small packages of test vectors, as illustrated in Figure 4.3. Here the test to $Core_1$ 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. This leads to a more flexible schedule, which also contributes to a possible decrease of the total test application time.

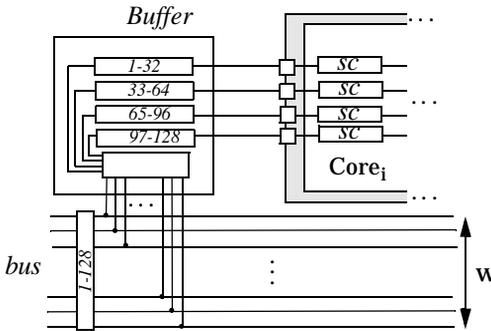
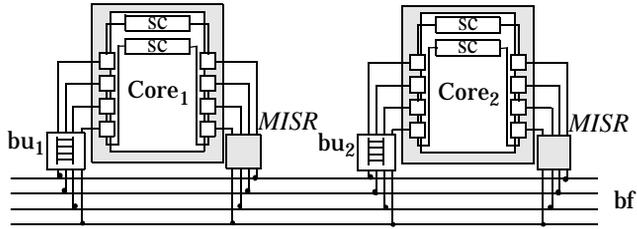
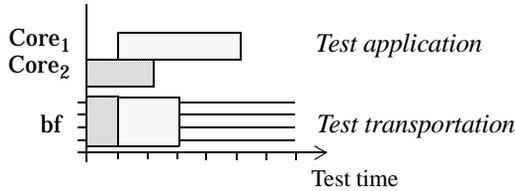


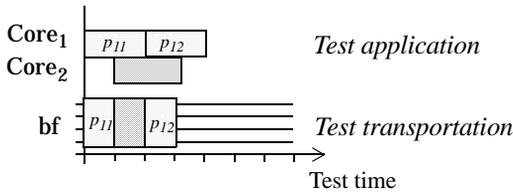
Figure 4.2: Bus and buffer connected to a core with four scan chains.



a) Test architecture with buffers and signature analyzers (MISR).



b) Test schedule and application with buffers.



c) Test schedule and application of test packages with buffers.

Figure 4.3: Test scheduling and application of test packages with buffers.

Each test T_i can be divided into m_{T_i} packages (each being a set of test vectors). There are two reasons for dividing tests into packages. As mentioned earlier, the transportation time t_i^{send-p} for a package on the bus is shorter than the application time t_i^{appl-p} . The size of the buffer does not have to be equal to the size

of the packages. This is explained by the fact that the test data in a package can be applied immediately when it arrives at the core. The buffer size bs_j , associated to a core $core_j$, is calculated with the following formula:

$$bs_j = \max(k_j \times (\tau_{start_{ij}} - \tau_{send_{ij}}) + \Delta_j), j \in (1, m_{T_i}) \quad (2)$$

where the constant k_j represents the rate 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_j 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_j is determined by the difference between t_i^{appl-p} and t_i^{send-p} , which is multiplied by the constant k_j .

The calculation of the buffer size is illustrated in Figure 4.4, which shows the bus schedule and the application of a test T_1 to core c_1 , with $t_1^{appl}=60$, $t_1^{send}=30$, $m_{T_1}=3$, and $k_1=1$. In this example the core has not finished the testing using the package, p_{12} , sent at time point $t_{send_{12}}=10$ before the package, p_{13} , sent at $t_{send_{13}}=20$ arrives at the core. This forces the buffer size to be increased. For this example the buffer size will be equal to $1 \times (40 - 20) + 10 = 30$, which is the difference between the termination of applying the last test package and the end point of transporting the corresponding package.

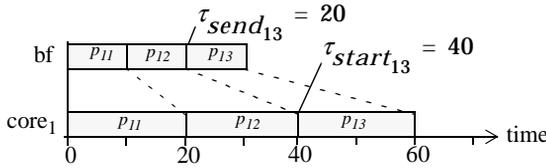


Figure 4.4: Example to illustrate time to transport and time to apply test.

4.2 Motivational Example

The following example illustrates the minimization of the buffer size and the test controller complexity. We make use of the system example in Figure 4.1, which consists of three cores $core_1$, $core_2$, and $core_3$ which are tested with three tests, T_1 , T_2 and T_3 , respectively. We have divided the tests into a total number of eight packages, all with the same application time and minimum package size, but different transportation times (Table 4.1). We assume that the maximal test application time, T_{max} for the system is given by the designer. In this example the time is 90 time units, which is the minimal time for applying these tests. This is the sum of the transportation times plus the smallest value of all Δ_i .

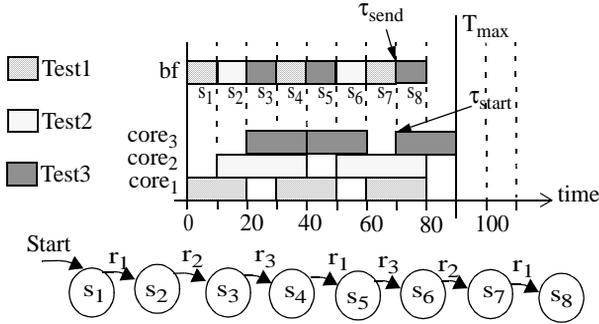
Three different schedules for the eight packages derived from the three tests are illustrated in Figure 4.5(a-c). In Figure 4.5(a) the packages are sent in such a way that the application of the previous package has finished before a new one arrives. This leads to small buffers since every package can be applied immediately as they arrive, that is $t_{startij} - t_{sendij} = 0$ for all packages. The buffer sizes for this schedule are, $bs_1 = 10$ ($bs_1 = 1 \times (0) + 10$), $bs_2 = 20$, and $bs_3 = 10$. A finite state machine is used to capture the complexity of the test controller. For realizing the schedule in Figure 4.5(a) it is required that the test controller has eight different control states.

In the second schedule, Figure 4.5(b), some packages are grouped together in pairs, which will produce larger buffers,

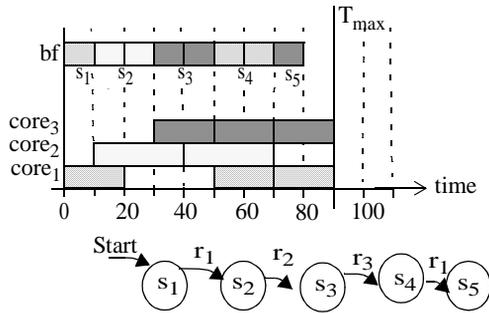
Table 4.1: Test characteristics.

Test	Nr. packages	Application time (t_i^{appl})	Transportation time (t_i^{send})	Δ_i
T_1	3	60	30	10
T_2	2	60	20	10
T_3	3	60	30	10

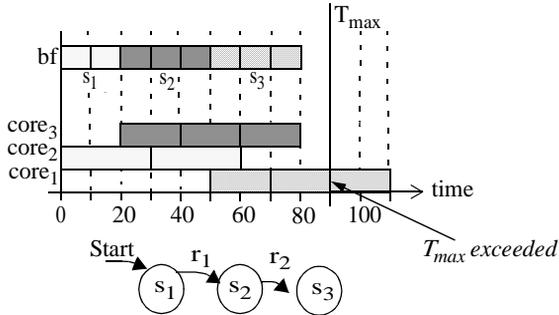
BUFFER AND CONTROL-LOGIC MINIMIZATION



a) Schedule with small buffers but a high number of control states.



b) Schedule with large buffers and few control states.



c) Schedule with large buffers and minimal number of control states.

Figure 4.5: Scheduling examples.

$bs_1 = 20$ ($bs_1 = 1 \times (70 - 60) + 10$), $bs_2 = 40$, and $bs_3 = 20$, however, the schedule in this example can be realized using only five states in the test controller.

The minimal number of control states needed for this example is three, one for each test. By using only three states it is impossible to schedule the packages in such way that the maximum allowed test application time is not exceeded, as illustrated in Figure 4.5(c).

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

4.3 Problem Formulation

The problem is formulated precisely as follows. Given is a system consisting of a set of N cores $C = \{c_1, c_2, \dots, c_N\}$, and each core, c_j , has a buffer bu_j where bs_j is the buffer size (initially bs_j is not determined). The maximal allowed test time for the system, t_{max} , is given as a constraint. Also given is the set of tests $T = \{T_1, T_2, \dots, T_N\}$, where T_i is a set of test vectors, which is to be applied to the core c_j . For each test T_j , the following information is given:

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

A test T_j is divided into a number of m_{T_j} packages, each of equal size $s^{T_i} P$. The package size $s^{T_i} P$ for a test T_i is determined as follows¹:

$$s^{T_i - P} = \left\lceil \frac{s^{T_i}}{m_{T_i}} \right\rceil \quad (3)$$

The time t_i^{appl-p} to apply a package belonging to test T_i is calculated as:

$$t_i^{appl-p} = \left\lceil \frac{t_i^{appl}}{m_{T_i}} \right\rceil \quad (4)$$

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

$$\tau_{finish_{ij}} = \tau_{start_{ij}} + t_i^{appl-p} \quad (5)$$

The objective of our technique is to find $\tau_{start_{ij}}$ and $\tau_{send_{ij}}$ for each package in such way that the total cost is minimized while satisfying the test time constraint, t_{max} . The total 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:

$$Cost_{Tot} = \alpha \times Cost_{Controller} + \beta \times Cost_{Buffer} \quad (6)$$

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

$$Cost_{Buffer} = k_1^B + k_2^B \times BufferSize \quad (7)$$

and the controller:

$$Cost_{Controller} = k_1^C + k_2^C \times CF1 \quad (8)$$

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

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

constants that represent the implementation cost parameters for the number of states 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 in the system is given by:

$$BufferSize = \sum_{i=1}^N bs_i \quad (9)$$

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

$$CF1 = K \times \{(N_i + N_o + 2 \times \lceil \log_2 N_s \rceil) \times N_t + 5 \times \lceil \log_2 N_s \rceil\} \quad (10)$$

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 .

Our problem is similar to the NP-complete multiprocessor resource constrained scheduling problem [Gar79], which is formulated as follows:

Given a set, A , of tasks, each having length $l(t)=1$, a number of m processors, a number of r resources, resource bounds B_i , where $1 \leq i \leq r$, resource requirement $R_i(t)$, $0 \leq R_i(t) \leq B_i$, for each task a and resource i , and an overall deadline, D .

Is there an multiprocessor schedule, σ , for A that meets the overall deadline D without violating the resource constraints, B_i ?

The problem formulated in this section can be translated into the above multiprocessor resource constrained scheduling problem if tasks are translate into tests, T , processors translated into cores, C , and the resources translated into hardware used to

transport the tests, which in our formulation is the bus. The resource constraint is the designer specified maximum overall test application time.

With these transformations, the objective is to find the schedule that minimizes the hardware cost without violating the resource constraint. In our formulation the overall deadline will be transformed into the test-time, which should not be exceeded.

4.4 Constraint Logic Programming Modelling

We have first modelled the system in a CLP program, consisting of two main components, *Test* and *Package* [Lar03a]. 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. $t_{start_{ij+1}} \geq t_{finish_{ij}}$,
- the start time of a package should be later or equal to the time of transmission on the bus: $t_{start_{ij}} \geq t_{send_{ij}}$,
- 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: $t_{finish_{ij}} \geq t_{start_{ij}} + T_i^{appI-P}$,
- the finish time of any test can not exceed the total test time limit, T_{max} : $t_{finish_{ij}} \leq T_{max}$

The buffer size at a core is determined by the formula presented in Section 4.1 (Eq. 2), and the cost of a solution is given by the formula in Section 4.3 (Eq. 6).

With the above constraint set, the constraint solver searches for a solution that minimizes the cost of the test.

4.5 The Tabu Search Based Algorithm

We have also implemented a tabu search based optimization heuristic for the problem described in Section 4.3 [Lar05a]. The main reasons for using tabu search, and not, for instance, simulated annealing or genetic algorithms, was tabu search's proven efficiency in solving scheduling problems, the relatively straightforward implementation, and the ability to handle additional constraints not captured in the original problem formulation.

The algorithm (Figure 4.6), consists of three steps: in step one (lines 1-9) an initial schedule is built, which is further improved in step two (10-14) and step three (15-35). The algorithm takes as additional input a minimal test time possible for the tests, t_{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 work).

In the initial step, the tests are sorted according to their application time, t_i^{appl} , and then the initial schedule is built. The slack, which is the difference between the end time of the schedule and t_{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. How a move is preformed is

BUFFER AND CONTROL-LOGIC MINIMIZATION

```
1 Step1: if  $t_{\max} < t_{\min}$  return Not schedulable
2 sort the tests T in increasing order of  $t_i^{\text{appl}}$ 
3 until all packages are applied do
4   apply package from  $T_i$ 
5   until time  $< t_i^{\text{appl}+p}$  do
6     apply package from  $T_{i+1}$ 
7     time = time +  $t_{i+1}^{\text{send}+p}$ 
8   repeat
9 repeat
10 Step2:doMark()
11 until Slack is 0
12   Delay package from MarkList
13 repeat
14 best_cost = compCost(Sched0)
15 Step3:start:
16 doMark()
17 for each pos in MarkList
18   build new schedule Schedi
19   delta_costi = best_cost - compCost(Schedi)
20 repeat
21 for each delta_costi < 0, in increasing order of delta_costi do
22   if not tabu(pos) or tabu_aspirated(pos)
23     Sched0 = Schedi
24     goto accept
25   end if
26 repeat
27 for each pos in MarkList
28   delta_costi' = delta_costi + penalty(pos)
29 repeat
30 for each delta_costi' in increasing order of delta_costi' do
31   if not tabu(pos) goto accept
32 repeat
33 accept:
34 if iterations since previous best solution < 10 goto start
35 return Sched0
```

Figure 4.6: Algorithm for test cost minimization

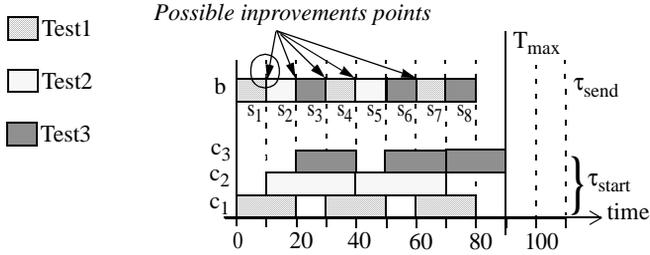
illustrated in Figure 4.7. In Figure 4.7(a), the different possible points of improvements are shown and one is selected. The move, which is selected, will reduce the number of control states since two packages will be merged. After the move selected in Figure 4.7(a) has been applied, the new schedule and the new possible points of improvements, are illustrated in Figure 4.7(b). Figure 4.7(c) shows the schedule after the move, selected in Figure 4.7(b), has been applied. The 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, 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 improvement is made for 10 consecutive tries. Also this number has been set on the basis of extensive experiments.

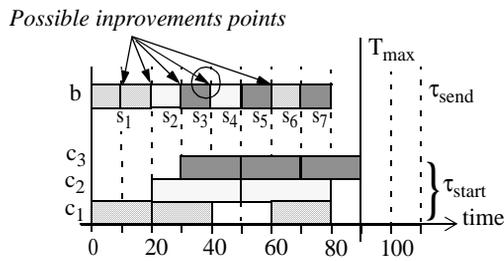
When the tabu search terminates, the solution with the lowest cost is returned.

4.6 Experimental Results

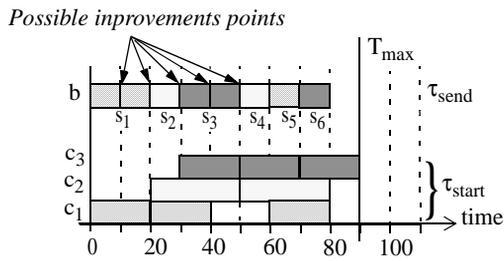
In our experiments we have used the following four designs; Ex1, Asic Z [Zor93], [Cho97], Kime [Kim82], and System L [Lar01a]. The main characteristics of the four designs, from the point of view of the problem addressed in this chapter, are presented in Table 4.2. More detailed information can be found in [Lar05d].



a) Schedule before applying a move.



b) Schedule after applying the move selected in a).



c) Schedule after applying the move selected in b).

Figure 4.7: Scheduling using proposed algorithm (Step3).

Table 4.2: Design characteristics

Design	Nr. cores	Nr. tests	Nr. packages	Min buffer size	Max buffer size
Ex1	3	3	8	80	200
Kime	6	6	20	186	680
Asic Z	9	9	38	222	838
System L	14	13	39	560	1976

We have used the CLP-tool CHIP (V 5.2.1) [Chi96] for the implementation. The experiments have been performed in two steps. In the first step the minimal test 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 time from step one is used as the time constraint, t_{max} in the second step, where the cost is minimized using the CLP approach.

The experimental results of the CLP solution are presented in Table 4.3, where the cost of our approach has been compared to the cost obtained by a straightforward approach. Column 1 lists the four designs and the maximum test time, t_{max} in Column 2. The total cost from the two approaches is presented in Column 3 and Column 4 and the cost comparison in Column 5. The results

Table 4.3: Experimental results.

Design	t_{max}	Straightforward approach	Our approach	Cost comparison
Ex1	111	116	92	-27.1%
Kime	257	625	460	-35.9%
Asic Z	294	472	319	-48.0%
System L	623	1843	1182	-55.9%

shows a decrease with 27 to 55% of the cost, which shows that our approach can decrease the cost by minimizing the buffer and controller, without exceeding the test time limitation.

Since CLP uses an exhaustive search approach, the optimization time using CLP can become large. For the largest benchmark, System L, the optimization time was more than 18 hours. In order to solve this problem a tabu search based heuristic is proposed that works for larger designs. In order to estimate the quality of the results produced by our heuristic (Section 4.5) we have compared them with those generated by solving the same optimization problem using the CLP formulation (Section 4.4). The experimental results using the CLP approach is collected in Table 4.4 where the results using our approach, also presented in Table 4.3, is complemented with the optimization time, and using the proposed heuristic in Table 4.5. The optimization time (CPU time) is presented in Column 3 and the total cost is presented in Column 4. The total cost obtained from the CLP approach is compared with the total cost from the tabu search based algorithm. As can be seen from the comparison (Table 4.6), our heuristic produced results which were only less then 6.1% 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 and produced results that, on average, were only 3.8% better.

Table 4.4: Experimental results using CLP.

Design	t_{\max}	CPU time (s)	Total cost
Ex1	111	160	92
Kime	257	27375	460
Asic Z	294	47088	319
System L	623	64841	1182

Table 4.5: Experimental results using the proposed algorithm.

Design	t_{\max}	CPU time (s)	Total cost
Ex1	111	<1	92
Kime	257	2	486
Asic Z	294	2	330
System L	623	3	1254

Table 4.6: Experimental results.

Design	Cost obtained using CLP	Cost obtained using the proposed algorithm	Cost comparison
Ex1	92	92	0%
Kime	460	486	+5,7%
Asic Z	319	330	+3,4%
System L	1182	1254	+6.1%

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. 1s for design Ex1, 2s for design Kime and Asic Z, and finally 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.

4.7 Conclusions

In this chapter we have proposed a technique to make use of the existing functional bus structure in the system for test data transportation. The advantage is that a dedicated bus for test purpose is not needed hence we reduce the cost of additional test wiring. 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 tabu search based algorithm where the test cost, given by the controller and buffer cost, is minimized without exceeding the given maximum test time. We have implemented and compared our technique with the results from a CLP approach. The results indicate that the proposed heuristic produces high quality solutions at low computational cost.

CHAPTER 4

Chapter 5

Test Scheduling with Test Set Sharing and Broadcasting

THE PURPOSE OF THIS CHAPTER is to describe a method where the existing functional bus may be reused for the test data transportation. In order to decrease the test time, we allow dedicated test buses to be added if this does not exceed the given maximum hardware cost constraint. We have also made use of a technique to share and broadcast test sets to generate better test schedules.

The first section in this chapter describes a method for test set sharing. This is followed by a description of the test architecture and a motivational example, which is used to illustrate the problem. The problem is modelled and solved using CLP. Experiments show that the overall test time can be significantly reduced when broadcasting of tests is used.

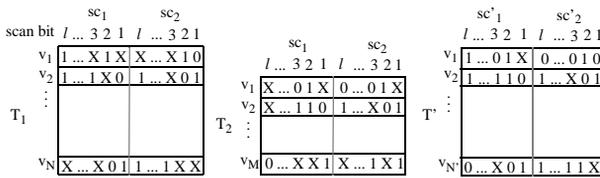
5.1 Test Set Sharing

Decreasing the test application time of a SoC entails a method that allows different cores to be tested concurrently. This can be done by, for instance, using multiple test buses. Another method would be possible if several cores use or share the same test set, which is transported and applied to the cores concurrently. There are two requirements that have to be satisfied before this method can be used. First, the cores must share the test set. This can be done by merging two or several test sets into one test set. Second, the TAM used to transport the shared test set must support broadcasting so that the cores that share the test receive it simultaneously.

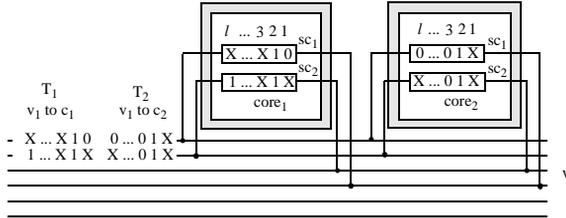
A test set that can be shared among several cores can be obtained by merging the dedicated tests for the cores. This can be done by using overlapping (identical) test vectors from the dedicated original tests. It is required that it is possible to find overlapping test vectors since the efficiency of this method depends on how large the merged test set is, in comparison with the unmerged sets. If no vectors can be merged, the new test set will have the same size as the sum of the original test sets, and the test transportation and test application time will be unchanged. The possibility of finding overlapping test vectors is increased if the vectors contain unspecified bits, so called don't cares, which can be set to either '1' or '0'. It has been shown that the amount of don't cares in the test sets is high, 78% for uncompact tests [Kaj01], [Cha03].

We have performed experiments to investigate the relationship between the number of don't care bits and the size of the merged test set [Lar05b]. For this purpose we use a straightforward pattern matching algorithm that takes two test sets, T_1 and T_2 , as input and generates a new test set T' . This is illustrated in Figure 5.1 where two cores, $core_1$ and $core_2$, are tested by two tests, T_1 and T_2 , respectively. Test T_1 consists of N test vectors, $\{v_1, v_2, \dots, v_N\}$, and test T_2 of M test vectors, $\{v_1,$

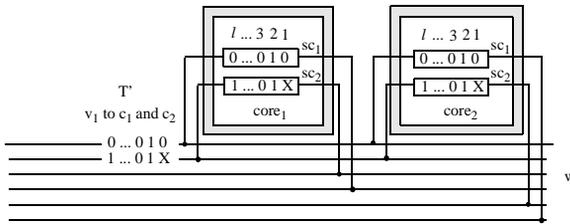
TEST SCHEDULING WITH TEST SET SHARING AND BROADCASTING



a) Merging two test sets.



b) Sequential test application.



c) Application of broadcasted test.

Figure 5.1: Merging and application of tests.

v_2, \dots, v_M . In this example, both cores have two scan chains, sc_1 and sc_2 , with equal length, l . The cores are connected to a bus with bandwidth w .

Merging one test vector v_i from Test T_1 with a test vector v_j from T_2 , where $i \in (1, N)$ and $j \in (1, M)$, is done by comparing each position in the two test vectors. As long as both have the same value or one is marked as a don't care (x), the vectors can

be merged as illustrated in Figure 5.1(a). If it is not possible to merge v_i with v_j , the next test vector, v_{j+1} , is tried until all test vectors have been investigated, i.e. when $j=M$. This process is then repeated for all v_i , i.e. when $i=N$. The test vectors, which are not possible to merge are kept intact and the size of the new test set T' is increased. In cases when the cores have scan chains with different length, the one with shorter length will be filled with don't cares.

Figure 5.1(b) shows how the cores are connected to a test bus. In this example the tests T_1 and T_2 are transported and applied sequentially. How the merged test, T' , is transported and applied is illustrated in Figure 5.1(c) and since it consists of vectors from test T_1 and T_2 , both core c_1 and c_2 will be tested concurrently.

This pattern matching algorithm has been applied to the benchmark design d695 [Cha01]. The test data is presented in Table 5.1. The test vectors (with don't cares marked) have been extracted by Kajihara and Miyase in [Kaj01]. The results from this experiment are presented in Figure 5.2(a), which shows 10 different combinations of the tests and the sizes of the merged tests. The sizes of the merged tests are compared with the size of the largest test used and the result shows an increase in size with on average only 10.94%. To illustrate the relationship between the number of don't cares and the size of the merged test set, a number of randomly generated tests were created and merged. The results depicted in Figure 5.2(b) show that when the number of care bits is in the range of 0 to 45 % the size of the merged set is still reasonably small, within an increase of 50%.

There is a relation between the possibility of merging two tests and the density of don't cares present in the tests. If the tests are compressed so that only a small percent of don't cares remains, the merging capability will significantly decrease. How the compression ratio is affected by the density of don't cares has been studied by Kinsman et al. [Kin05]. To cope with a situation

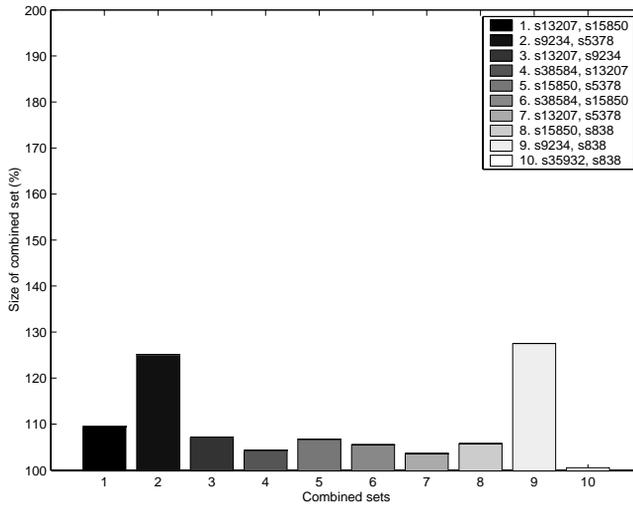
Table 5.1: Test data for d695.

Circuit (core)	Nr. test patterns	Nr. scan chains	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%

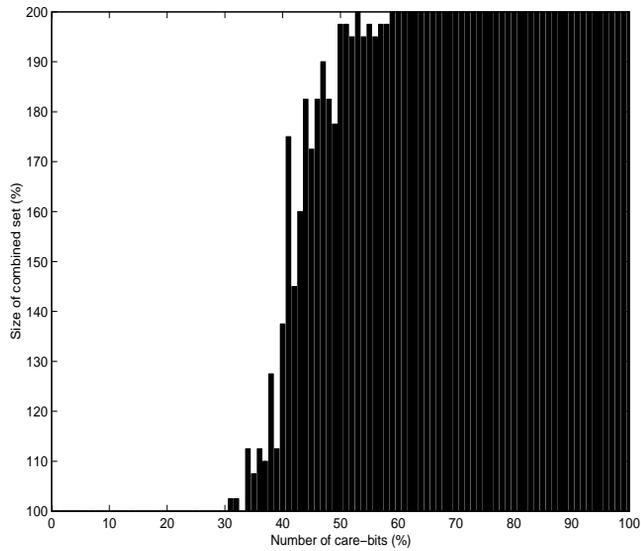
where the compression reduces the number of don't cares so that it is not possible to merge tests, the test designer can set a limit on the amount of compression of a test, so that it is still possible to use it for merging.

In order to apply the merged tests, it is required that the cores, which share the tests are connected in such a way that the tests can be broadcasted to the cores. In this work the test responses from each core will be transported on wires different from those which transported the test stimuli (on the architecture description in the following section).

CHAPTER 5



a) Different combinations from design d695.



b) Random sets with increasing number of don't cares.

Figure 5.2: Merging of test sets [Lar05b].

5.2 System Architecture

In this section we discuss the architecture that is used for test data transportation. First, we introduce the general architecture by considering a system example consisting of a set of cores (e.g. $core_1$, $core_2$, $core_3$ and $core_4$ in Figure 5.3), which are connected by one or several buses (bf_1 and bt_1 in Figure 5.3). The bus wires are connected to an ATE, which is used to apply test vectors and to analyze the responses of the tests.

A connector consisting of the logic needed for the communication and application of test data is introduced between each core and the bus. For example, $o_{4,1}$ is the connector connecting $core_4$ with bus bf_1 , as shown in Figure 5.3. It is assumed that the buses are connected to the input and output pins of the chip, and hence, directly accessible and controlled from the ATE.

The functional bus is used to transport test data from the ATE to the cores. However, if the bandwidth of the functional bus is not enough for transporting the test data in reasonable time, one or several dedicated test buses may be added to the design. A dedicated test bus for the transportation of test data will increase the transportation capacity and shorten the test time. It also offers the possibility of a trade-off between the test time and the number of wires used. A high number of wires require however large silicon area to be implemented.

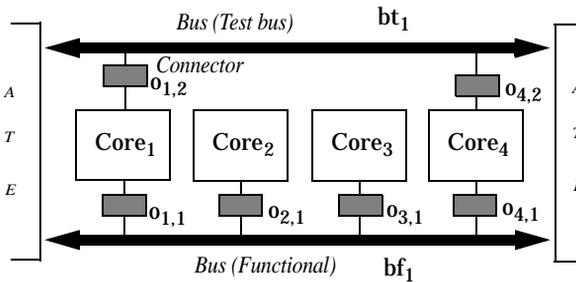


Figure 5.3: Bus based architecture.

The transportation of tests to the cores is illustrated in Figure 5.4 by considering cores $core_1$ and $core_2$ from Figure 5.3. It is assumed that both cores have three wrapper chains each, which are connected to the bus as illustrated in Figure 5.4. The test stimuli are transported from the ATE on the bus to the core through the input test pins, t_{in} . When the test stimuli have been applied, the test responses are transported back to the ATE through the outputs, t_{out} . When the system is in the functional mode, the functional inputs and outputs at each core are connected to the functional bus. When the system is in the testing mode the connectors will receive control signals, t_{ctrl} indicating when a vector should be applied.

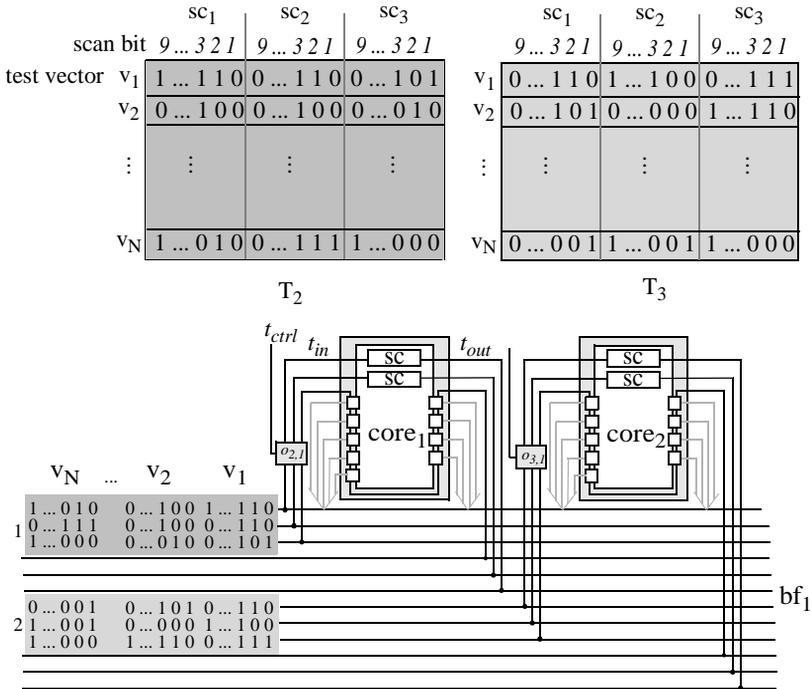


Figure 5.4: Test transportation.

Short application time of scan vectors entails a concurrent scan-in and scan-out phase, that is, when one scan vector is shifted out, a new vector is shifted in. Therefore it is not possible to share the same bus wires for the test stimuli and test responses of one core; the total number of bus wires used to test one core is twice the number of scan chains of the core, as illustrated in Figure 5.5.

The application of tests in Figure 5.5 can only be done sequentially, as long as the cores are connected to the same bus wires. To give exclusive access to the bus for both cores at the same time, and hence make it possible to apply the test to the cores concurrently, would significantly decrease the test time. However, in the example in Figure 5.6, the bandwidth of the bus is not enough and the two cores are forced to use the same wires for several connections. If the tests are applied in parallel under this situation they will overlap and hence give an impossible schedule as illustrated in Figure 5.6. If we consider that the two cores, $core_1$ and $core_2$, use a shared test set, as described in Section 5.1, which is broadcasted, the number of wires will be

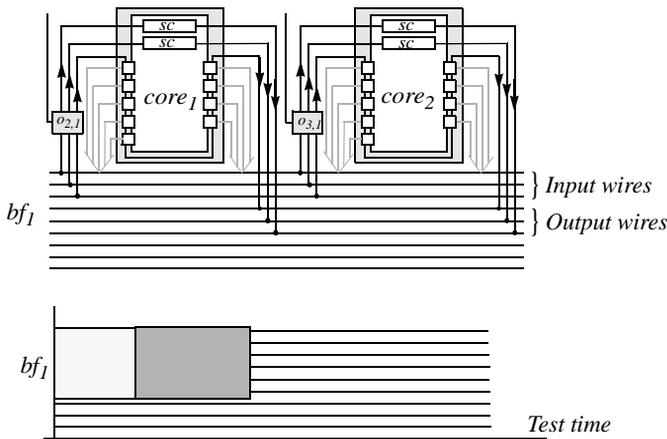


Figure 5.5: Connections and sequential schedule.

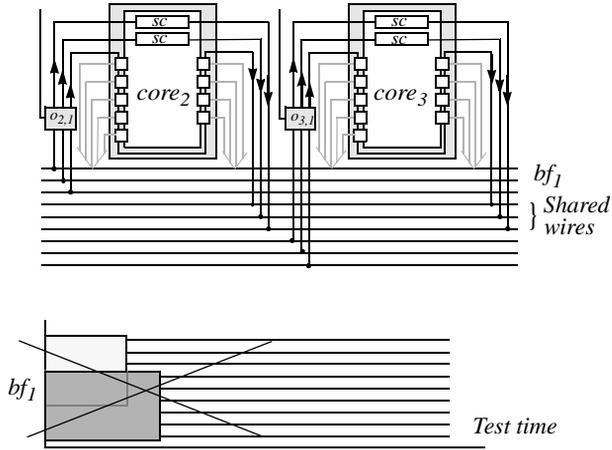


Figure 5.6: Connections where parallel schedule is not possible.

enough for transporting both the test stimuli and test responses and both cores will be tested concurrently. This is possible since the input wires will be shared by the cores and only the output responses require dedicated wires for each core, as shown in Figure 5.7.

This example illustrates that by using shared test sets, which are broadcasted, it is possible to get a shorter test time compared to a sequential application and still use a smaller amount of wires compared with a parallel application.

5.3 Motivational Example

Let us consider an example design consisting of four cores, $core_1$, $core_2$, $core_3$, and $core_4$, connected to one functional bus bf_1 with the help of the test connectors ($o_{1,1}$, $o_{1,2}$, $o_{1,3}$, and $o_{1,4}$), as shown in Figure 5.8. Each core has one test: $core_1$ is tested by T_1 , $core_2$ by T_2 , $core_3$ by T_3 , and $core_4$ by T_4 . The dedicated test set to

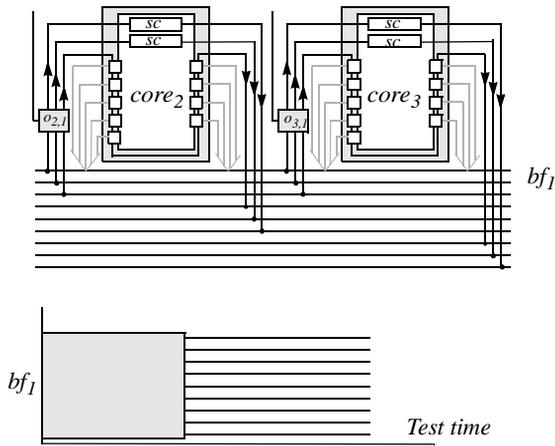


Figure 5.7: Connections and broadcasted test.

these cores has been extended with one additional test T'_5 , which is a combination of the tests T_3 and T_4 for core *core3* and *core4*. If T'_5 is selected, it is broadcasted to *core3* and *core4*.

For the sake of simplicity it is assumed that each test will occupy the whole bandwidth of the bus, which means that a single bus configuration will lead to a sequential application of the tests. In the first schedule shown in Figure 5.8(a) the shared test (T'_5) is not used, while in the second schedule, Figure 5.8(b), T'_5 is introduced and, since it can be applied to two cores, *core3* and *core4*, concurrently, the test time is decreased.

The test time may be further decreased if a dedicated test bus, bt_1 is introduced as illustrated in Figure 5.8(c and d). It will enable concurrent application of tests. Figure 5.8(c) shows an example of a mapping of cores to buses. In this example only one core is tested through the test bus and the test time has decreased compared with the example where only one bus was used. Since *core4* is tested through the test bus it is not possible

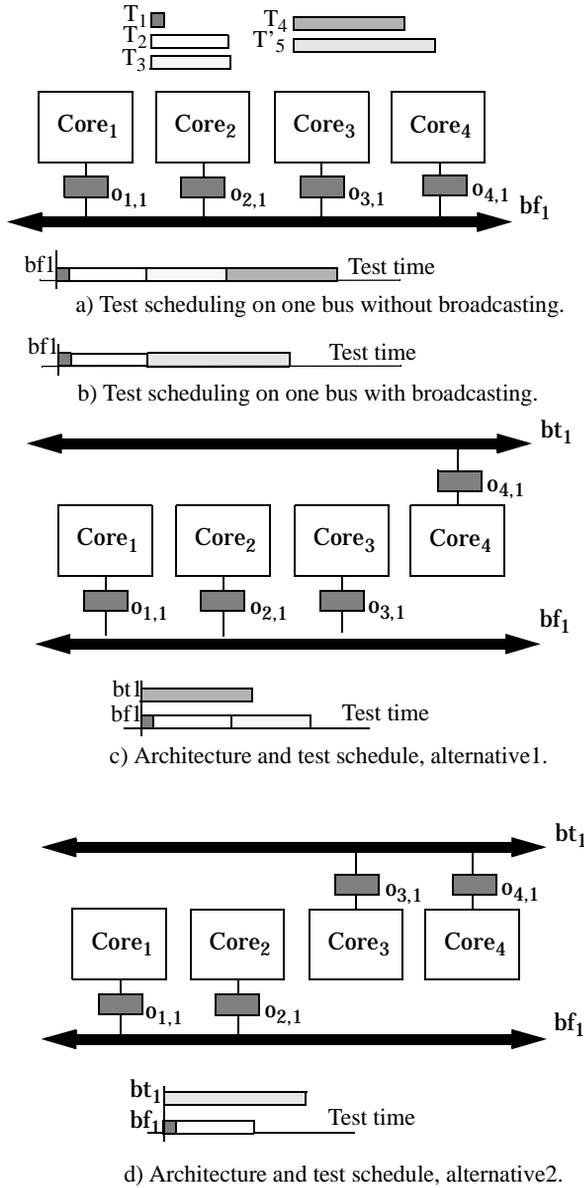


Figure 5.8: Test scheduling for different bus architectures.

to make use of the broadcast capability between $core_3$ and $core_4$. However, by connecting $core_3$ and $core_4$ to the same bus, as shown in Figure 5.8(d), the test time can be further reduced.

5.4 Problem Formulation

In this section a precise problem formulation is given as follows.

Given is a system consisting of:

- a set of N cores $C = \{c_1, c_2, \dots, c_N\}$,
- a set of M functional buses $B_F = \{bf_1, bf_2, \dots, bf_M\}$, where each bus bf_i has the bandwidth, w^{bf_i} .

Given for each core c_i , where $i = \{1, 2, \dots, N\}$, is:

- u^i the number of wrapper-chains and
- l^i the length of the longest wrapper chain.

Given is also a set of test sets $\Psi_i = \{\Gamma_1^i, \Gamma_2^i, \dots, \Gamma_X^i\}$, where each test set $\Gamma_j^i = \{T_1, T_2, \dots, T_H\}$. For a test T_k is a given number of vectors, s^{T_k} . Each core in the system is associated with a set of test sets. This is explained by considering the following example where core c_1 and c_2 have two associated test sets each, $\Psi_1 = \{\{T_1, T_2\}, \{T_3\}\}$, and $\Psi_2 = \{\{T_4, T_5\}, \{T_3, T_5\}\}$. Core c_1 is fully tested by either applying both T_1 and T_2 or only T_3 . T_3 is used to test both c_1 and c_2 , which means that it is shared and should be broadcasted.

The test time, τ_k for applying a test, T_k at core c_i is:

$$\tau_k = (1 + l^i) \times s^{T_k} + l^i \quad (11)$$

The number of wires w_k that a test makes use of when transported on a bus depends on the number of cores, z , that shares the test. If no sharing is used ($z=1$), w^i wires are used to transport test stimuli to the core, and w^i wires are used to transport test responses from the core. In total $2 \times w^i$ wires are needed when $z=1$. In the case of broadcasting, w^i wires are used

to transport test stimuli to the cores, but each core requires w^i wires to transport test responses from the core. In total $w^j + w^i \times z$. This is given by the following formula:

$$w_k = \begin{cases} 2 \times w^i, & \text{when } z=1 \\ w^j + w^i \times z, & \text{when } z \geq 2 \end{cases} \quad (12)$$

In order to apply them, the tests are transported from the ATE to the cores. For this purpose, the existing functional bus structure can be used or test buses can be added. If a test shall make use of the functional bus, a connector must be inserted between the bus and the core. If a dedicated test bus is used, the bus must have been inserted, and a connector is added between the test bus and the core. We introduce test buses and the bus connectors:

- a set of G test buses $B_T = \{bt_1, bt_2, \dots, bt_G\}$ where bus bt_i has the bandwidth w^{bt_i} ,
- test connectors $o_{i,j}$ between core c_i and bus bf_j (or bt_j).

The following hardware cost factors are considered:

- $kf_{i,j}$ is the cost of inserting a connector $o_{i,j}$ between core c_i and functional bus bf_j ,
- $kt_{i,j}$ is the cost of inserting a connector $o_{i,j}$ between core c_i and test bus bt_j ,
- k^{bt_i} is the base cost of inserting test bus bt_i .

The total hardware cost HW_{Tot} is given by:

$$HW_{Tot} = \sum_{i=1}^N \sum_{j=1}^M kf_{i,j} + \sum_{i=1}^N \sum_{j=1}^G kt_{i,j} + \sum_{j=1}^G k^{bt_j} \quad (13)$$

The following constraints are specified:

- the total bandwidth of the functional buses and inserted test buses does not exceed the bandwidth, w_{ATE} , of the ATE, that is;

$$\sum_{i=1}^M w^{bf_i} + \sum_{j=1}^G w^{bt_j} \leq w_{ATE} \quad (14)$$

where M is the number of functional buses and G the number of added test buses, and

- the total hardware cost does not exceed the designer specified hardware overhead, K_{max} that is;

$$HW_{Tot} \leq K_{max} \quad (15)$$

The optimization objective is to minimize the test application time. This is done by:

- select tests for each core,
- insert test buses (if required),
- insert connectors between cores and buses, and
- schedule the selected tests on the buses

in such a way that each core is fully tested and without violating any constraints.

5.5 Constraint Logic Programming Modelling

We have formulated our test scheduling problem as a CLP problem (Figure 5.9) [Lar05b]. The cores, their connections and information about the tests regarding the size and number of wrapper chains, are first given as input (line 2 and 3 in Figure 5.9). A number of variables used to describe a solution are then defined (4...10).

In order to find a feasible solution that minimizes the total test time (16) the program ensures that the following constraints are fulfilled (11...15):

- Each core must be connected to at least one bus (11).
- Each core must be fully tested (12).
- The hardware cost does not exceed the given maximum hardware cost (14), (Eq. 15).
- The total number of wires does not exceed the given maximum limit imposed by the bandwidth of the ATE (Eq. 14) and tests do not make use of the same wires concurrently (15).

```

1 run:-
2 Cores({1,2,3,...,NrCores}), % Get input data
3 Tests({1,2,3,...,NrTests}),
4 NrBuses::1..MaxNrBuses, % Define variables
5 Cost::1..MaxCost,
6 TestTime::1..MaxTestTime,
7 ListOfTests::0..NrTests,
8 ListOfCores::0..NrCores,
9 Schedule::0..NrTests*NrBuses,
10 Tam::1..MaxTam,
11 connect_all(Cores), % Set up constraints
12 complete_cores(Cores,Tests),
13 count_costs(Cores,Costs,Cost),
14 Cost #< MaxCost,
15 cumulative (Schedule, Duration, Resource, Tam, TestTime),
16 min_max((labeling(Schedule)),TestTime). % Find optimal solution

```

Figure 5.9: CLP formulation in CHIP for test time minimization.

We have used the following built in predicates in the CLP tool CHIP [Hen91], [Chi96], to ensure that all constraints are satisfied and the optimal solution is found:

- Cumulative (15), ensures that, at any given time, the total amount of resources does not exceed a given limit.
- Min_max (16), implements a depth-first branch and bound search for a solution with the minimal cost.
- Labeling (16), is used to assign values to variables.

Since a test T_i can be listed for several cores, a special constraint is implemented so that T_i is not scheduled more than one time as long as the cores are connected to the same bus.

5.6 Experimental Results

In our experiments we have used the following eight designs: SOC_(1..7), which are randomly generated, and the benchmark design d695 [Cha01]. The main characteristics of the eight designs can be found in Table 5.2. This table contains information about the number of cores, tests, and the minimum required hardware constraint needed. The minimum hardware constraint is the hardware cost needed in order to connect each core to a functional bus. If this constraint is not satisfied the core will not be fully accessed or tested.

The hardware cost, such as wiring and control logic needed to connect a core to a bus or to add a test bus, is assumed to be given by the designer. In the experiments, the cost of connecting a core to a functional bus is set to 10 units, the cost to connect a core to a test bus to 20 units, and the cost of adding a test bus to the system is set to 100 units. This means that adding one test bus and connect one core to it will be associated with a hardware cost of 120 units. In these designs it is also assumed that each

Table 5.2: Design characteristics

Design	Nr. cores	Nr. tests	Min. HW constraint
SOC_1	4	5	40
SOC_2	7	9	70
SOC_3	10	12	100
SOC_4	12	15	120
SOC_5	18	20	180
SOC_6	24	28	240
SOC_7	30	34	300
d695	10	12	100

system has a 64 bit wide functional bus and that each test bus, if added to the system, has a width of 32 bits.

We have used the CLP tool CHIP (V 5.2.1) [Chi96] for the implementation and we have compared the results in the cases when broadcasting is not used and when broadcasting is used.

The results when broadcasting is not used are collected in Table 5.3, and when broadcasting is used in Table 5.4. Column 1 lists the eight different designs and in the hardware constraints are listed in Column 2 . These constraints have been set so 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: the number of added test buses in Column 3, the number of test vectors used in Column 4, the minimized test time in Column 5, and the optimization time in Column 6.

Table 5.5. shows the comparison in test time between the two approaches. The experiments show that broadcasting of tests between cores can shorten the test time. The test time is decreased with 23.72% on average.

Table 5.3: Experimental results without broadcasting

Design	Hardware constraint	Nr. added test buses	Tot. nr. vectors used	Test time	CPU time (s)
SOC_1	250	1	275	8271	14
SOC_2	350	1	440	22361	76
SOC_3	400	2	539	37943	391
SOC_4	450	2	753	51946	624
SOC_5	500	2	2316	86301	1028
SOC_6	500	3	15017	1167250	2734
SOC_7	600	3	18779	1602862	4893
d695	350	2	881	24219	235

Table 5.4: Experimental results with broadcasting

Design	Hardware constraint	Nr. added test buses	Tot. nr. vectors used	Test time	CPU time (s)
SOC_1	250	1	209	6755	76
SOC_2	350	1	297	18421	132
SOC_3	400	2	423	29364	572
SOC_4	450	2	687	37526	1517
SOC_5	500	2	1723	61730	39843
SOC_6	500	3	11483	869195	62087
SOC_7	600	3	14021	1187512	95274
d695	350	2	802	18522	586

Table 5.5: Experimental results

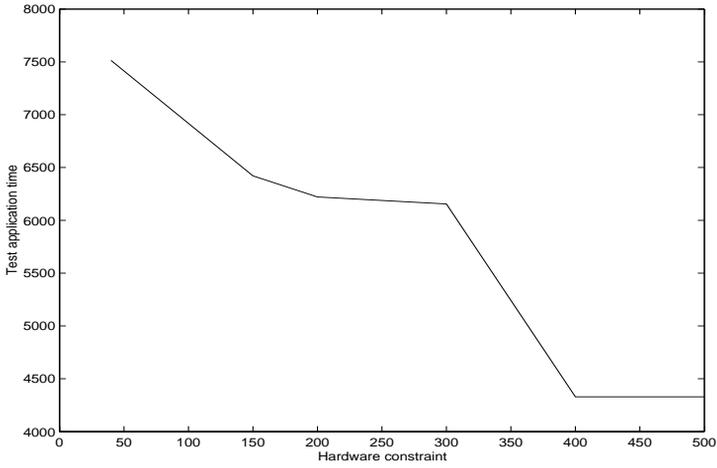
Design	Test time without using broadcasting	Test time when using broadcasting	Test time comparison
SOC_1	8271	6755	-18.33%
SOC_2	22361	18421	-17.62%
SOC_3	37943	29364	-22.61%
SOC_4	51946	37526	-27.76%
SOC_5	86301	61730	-28.47%
SOC_6	1167250	869195	-25.53%
SOC_7	1602862	1187512	-25.91%
d695	24219	18522	-23.52%

Experiments have also been made to show the impact on the test time at different hardware constraints. The test time minimization has been made with different values of the hardware constraint for two examples, SOC_1 and the benchmark design d695. The results collected in Table 5.6 show how the test time for the different designs decreases as additional test buses are added. For SOC_1 a saturation point is met when the hardware constraint reaches a point where all cores can be tested concurrently (HW constraint = 400). In Figure 5.10 the curves from this experiment showing the test application time for different hardware constraints are depicted. Figure 5.10(a) shows the graph for design SOC_1 and Figure 5.10(b) the graph for d695.

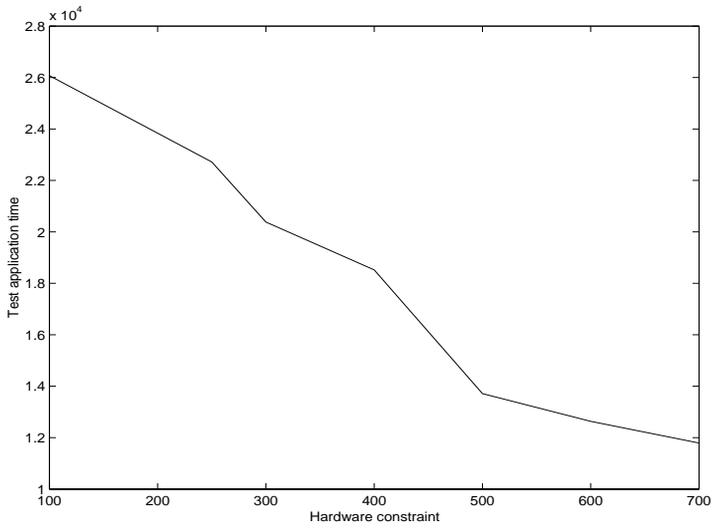
Table 5.6: Test time for different hardware constraints

Design	HW constraint	Nr. added test wires	Test time
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	64	20382
	400	64	18522
	500	96	13712
	600	128	12633
	700	160	11791

TEST SCHEDULING WITH TEST SET SHARING AND BROADCASTING



a) SOC_1.



b) d695.

Figure 5.10: Test time for different hardware constraints.

5.7 Conclusions

Decreasing the test application time for SoCs requires efficient test data transportation and concurrent test application. In this chapter a scheme to explore the high amount of don't cares present in the test sets in order to merge different tests, is proposed. The shared test set can be used as alternative to the original dedicated test for the cores.

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

We have modelled the problem and implemented it using CLP and experiments show that the overall test time can be significantly reduced when broadcasting of tests is used.

Chapter 6

Conclusions and Future Work

THIS CHAPTER summarises the thesis and gives a description of possible directions of future work.

6.1 Conclusions

As a first approach in this thesis, a technique to make use of the existing functional bus structure of the system for test data transportation is proposed. A buffer is inserted between each core and the functional bus, and the tests are divided into packages, such that they can be scheduled concurrently even if the bus only allows sequential transportation.

A tabu search based algorithm is proposed where the hardware overhead captured by the controller and buffer cost, is minimized without exceeding a given, designer-specified, maximum test time. The technique has been implemented and

is compared with the results from a CLP-based approach. The results indicate that the proposed heuristic produces high quality solutions at low computational cost.

In the second problem addressed, a scheme is proposed to explore the high amount of don't cares present in the test sets in order to merge different tests, which can be used as alternative to the original dedicated test for the cores. The proposed method allows the existing functional bus structure to be reused for the test data transportation. However, in order to decrease the test time, dedicated test buses may be added to the design. The problem formulated is to select appropriate tests 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 implemented using CLP and experiments show that the overall test time can be significantly reduced when broadcasting of tests is used.

6.2 Future Work

The problem addressed in Chapter 5 is solved by using a CLP formulation. Since CLP uses an exhaustive search approach, optimization times can become large for complex designs. A natural extension of the work is to find a heuristic that would work for larger designs.

The two techniques proposed in this thesis can be extended to include additional constraints such as power consumption, and test conflicts that occur when one part of a system has to be used while another part is tested.

Another direction would be to address the test transportation problem in NoC. NoC is gaining popularity in literature as communication platform [Ben02], [Jan03]. This is due to the limited scalability when using buses and switches in SoC. There are several interesting test transportation and test scheduling

CONCLUSIONS AND FUTURE WORK

problems that can be defined for NoC. For instance, routing of test data using the NoC infrastructure, while minimizing the test application time. Another problem would be to determine the placement of test sources and test sinks inside the NoC in such way that the routing cost and/or test time is minimized.

CHAPTER 6

References

- [Aer98] J. Aerts, E. J. Marinissen, "Scan Chain Design for Test Time Reduction in Core-based ICs", *Proceedings of International Test Conference (ITC)*, pp. 448 - 457, 1998.
- [Amb05] AMBA Specification Overview, ARM, *web site: www.arm.com/products/solutions/AMBAHomePage.html*, 2005.
- [Bee86] F. Beenker, K. van Eerdewijk, R. Gerritsen, F. Peacock, M. van der Star, "Macro testing: Unifying IC and Board Test", *IEEE Design and Test of Computers*, Vol. 3, No. 4, pp. 26-32, 1986.
- [Ben02] L. Benini, G. De Micheli, "Networks on chips: a new SoC paradigm", *Computer*, Vol. 35, No. 1, pp. 70 - 78, 2002.
- [Bha96] S. Bhatia, T. Gheewala, P. Varma, "A Unifying Methodology for Intellectual Property and Custom Logic Testing", *Proceedings of International Test Conference (ITC)*, pp. 639 - 648, 1996.

- [Bie95] U. Bieker, P. Marwedel, "Retargetable Self-Test Program Generation Using Constraint Logic Programming", *Proceedings of Design Automation Conference (DAC)*, pp. 605-611, 1995.
- [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.
- [Cha03] 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 , No 3, pp. 352-363, 2003.
- [Chi96] CHIP, System Documentation, COSYTEC, 1996.
- [Cho97] R. M. Chou, K. K. Saluja, 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.
- [Cor05] CoreConnect Bus Architecture, IBM, *web site: www.chips.ibm.com/products/coreconnect*, 2005.
- [DaS03] F. DaSilva, Y. Zorian, L. Whetsel, K. Arabi, R. Kapur, "Overview of the IEEE P1500 Standard", *Proceedings of International Test Conference (ITC)*, Vol. 1, pp. 988 - 997, 2003.
- [Eba01] Z. S. Ebadi, A. Ivanov, "Design of an Optimal Test Access Architecture using a Genetic Algorithm", *Proceedings of Asian Test Symposium (ATS)*, pp. 205-210, 2001.
- [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.

REFERENCES

- [Gho00] I. Ghosh, S. Dey, N. K. Jha, "A fast and low-cost testing technique for core-based system-chips", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 8, pp. 863 - 877, 2000.
- [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.
- [Har99] P. Harrod, "Testing Reusable IP - A Case Study", *Proceedings of International Test Conference (ITC)*, pp. 493 - 498, 1999.
- [Hen91] P. Van Hentenryck, "The CLP language CHIP: constraint solving and applications," *Compon Spring '91. Digest of Papers*, pp. 382 -387, 1991.
- [IEE05] "IEEE Std 1500-2005 Standard Testability Method for Embedded Core-based Integrated Circuits", pp. 1-117, 2005.
- [Imm90] V. Immaneni, S. Raman, "Direct Access Test Scheme - Design of Block and Core Cells for Embedded ASICs", *Proceedings of International Test Conference (ITC)*, pp. 488-492, 1990.
- [Int03] "The International Technology Roadmap for Semiconductors. 2003 Edition (ITRS2003)", *Semiconductor Industry Association*, 2003.
- [Iye01] V. Iyengar, K. Chakrabarty, "Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-Chip", *Proceedings of VLSI Test Symposium (VTS)*, pp. 368 - 374, 2001.

- [Jaf87] J. Jaffar, and J.-L. Lassez, "Constraint Logic Programming", *Proceedings of ACM Symposium on Principles of Programming Languages (POPL)*, pp. 111-119, 1987.
- [Jan03] A. Jantsch, H. Tenhunen (editors), "Networks on Chip", *Kluwer Academic Publishers*, ISBN-1-4020-7392-2, 2003.
- [Jer02] G. Jervan, Z. Peng, R. Ubar, and H. Kruus, "A Hybrid BIST Architecture and its Optimization for SoC Testing", *Proceedings of International Symposium on Quality Electronic Design*, pp. 273 - 279, 2002.
- [Jia03] J. H. Jiang, W-B. Jone, S-C. Chang, S. Ghosh, "Embedded Core Test Generation Using Broadcast Test Architecture and Netlist Scrambling", *IEEE Transactions on Reliability*, Vol. 52, Issue 4, pp.435 - 443, 2003.
- [Kaj01] S. Kajihara, 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.
- [Kim82] C. R. Kime, "Test Scheduling in Testable VLSI Circuits", *Proceedings of International Symposium on Fault-Tolerant Computing*, pp. 406 - 412, 1982.
- [Kin05] A. B. Kinsman, N. Nicolici, "Time-Multiplexed Test Data Decompression Architecture for Core-Based SOCs with Improved Utilization of Tester Channels", *Proceedings of European Test Symposium (ETS)*, pp. 196 - 201, 2005.
- [Kir83] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671-679, 1983.

REFERENCES

- [Lar01a] E. Larsson, Z. Peng, "An Integrated System-on-Chip Test Framework", *Proceedings of the Design, Automation and Test in Europe (DATE)*, pp. 138 - 144, 2001.
- [Lar01b] E. Larsson, Z. Peng, G. Carlsson, "The Design and Optimization of SOC Test Solutions", *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pp. 523 - 530, 2001.
- [Lar02] E. Larsson, H. Fujiwara, "Power Constrained Preemptive TAM Scheduling", *Proceedings of European Test Workshop (ETW)*, pp. 119 - 126, 2002.
- [Lar03a] A. Larsson, E. Larsson, P. Eles, Z. Peng, "Buffer and Controller Minimisation for Time-Constrained Testing of System-On-Chip", *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pp. 385-392, 2003.
- [Lar03b] E. Larsson, Z. Peng, "A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling", *Proceedings of International Test Conference (ITC)*, Vol. 1, pp. 1135 - 1144, 2003.
- [Lar05a] A. Larsson, E. Larsson, P. Eles, Z. Peng, "Optimization of a Bus-based Test Data Transportation Mechanism in System-on-Chip", *Proceedings of Euromicro Conference on Digital System Design (DSD)*, pp. 403-409, 2005.
- [Lar05b] A. Larsson, E. Larsson, P. Eles, Z. Peng, "SOC Test Scheduling with Test Set Sharing and Broadcasting", *To be published in proceedings of Asian Test Symposium (ATS)*, 2005.

- [Lar05c] E. Larsson, "Introduction to Advanced System-on-Chip Test Design and Optimization", *FRONTIERS IN ELECTRONIC TESTING*, Vol. 29, Springer, 2005.
- [Lar05d] E. Larsson, A. Larsson, and Z. Peng, "Linköping University SOC Test Site", <http://www.ida.liu.se/labs/eslab/soctest/>, 2005.
- [Lee98] K-J. Lee, J-J. Chen, C-H. Huang, "Using a Single Input to Support Multiple Scan Chains," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 74 - 78, 1998.
- [Lee99] K-J. Lee, J-J. Chen, 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.
- [Mar97] E.J. Marinissen, M. Lousberg, "Macro Test: A Liberal Test Approach for Embedded Reusable Cores", *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, Paper 1.2, 1997.
- [Mar98] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemans, M. Lousberg, C. Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores", *Proceedings of International Test Conference (ITC)*, pp. 284-293, 1998.
- [Mar99a] E. J. Marinissen, Y. Zorian, "Challenges in Testing Core-based System ICs", *IEEE Communications Magazine*, Vol. 37, pp.104-109, 1999.
- [Mar99b] K. Marriott, P. J. Stuckey, "Programming with Constraints - An Introduction", *MIT Press*, 1999.

REFERENCES

- [Mit93] B. Mitra, P.R. Panda, and P.P Chaudhuri, "Estimating the Complexity of Synthesized Designs from FSM Specifications", *Design & Test of Computers*, Vol 10, pp. 30-35, 1993.
- [Mic96] Z. Michalewicz, "Genetic Algorithm + Data Structure = Evolutionary Programs", *3d Edition, Springer Verlag*, 1996.
- [Moo65] G. E. Moore, "Cramming More Components Onto Integrated Circuits", *Electronics*, Vol. 38, No. 8, pp. 114-117, 1965.
- [Mur00] V. Muresan, W. Xiaojun, M. Vladutiu, "A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling", *Proceedings of International Test Conference (ITC)*, pp. 882 - 891, 2000.
- [Nag02] P. K. Nag, A. Gattiker, W. Sichao, R. D. Blanton, W. Maly, "Modeling the Economics of Testing: a DFT Perspective", *IEEE Design & Test of Computers*, Vol. 19, Issue 1, pp. 29 - 41, 2002.
- [Pol03] F. Poletti, D. Bertozzi, L. Benini, 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.
- [Rub86] S. M. Rubin, "Computer Aids for VLSI Design", *The Addison-Wesley VLSI System Series*, Vol. 6, 1986.
- [Var98] P. Varma, S. Bathia, "A Structured Test Re-Use Methodology for Core-Based System Chips", *Proceedings of International Test Conference (ITC)*, pp. 294-302, 1998.

- [Xu05] Q. Xu, N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey", *IEEE Proceedings- Computers and Digital Techniques*, Vol. 152, No. 1, pp. 67 - 81, 2005.
- [Zen01] Z. Zeng, M. Ciesielski, B. Rouzeyre, "Functional test generation using constraint logic programming", *In Proceedings of IFIP International Conference on Very Large Scale Integration (IFIP VLSI-SOC 2001)*, pp. 375-387, 2001.
- [Zor93] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices", *VLSI Test Symposium (VTS)*, pp. 4-9, 1993.
- [Zor97] Y. Zorian, "Test Requirements for Embedded Core-Based Systems and IEEE P1500", *Proceedings of International Test Conference (ITC)*, pp. 191 - 199, 1997.
- [Zor99] Y. Zorian, E. J. Marinissen, S. Dey, "Testing Embedded-Core-Based System Chips", *Computer*, Vol. 32, No. 6, pp. 52-60, 1999.