SOLUTIONS FOR EMERGING PROBLEMS IN MODULAR SYSTEM-ON-A-CHIP TESTING

SOLUTIONS FOR EMERGING PROBLEMS IN MODULAR SYSTEM-ON-A-CHIP TESTING

BY
QIANG XU
AUGUST 2005

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DOCTOR OF PHILOSOPHY (2005) McMaster University
(Electrical and Computer Engineering) Hamilton, Ontario

TITLE: Solutions for Emerging Problems in Modular System-on-a-Chip Testing

AUTHOR: Qiang Xu

SUPERVISOR: Nicola Nicolici

NUMBER OF PAGES: xvii, 211

Acknowledgments

Finally the long journey has come to the end, and I am really glad that I have now the opportunity to express my gratitude for all the people who have made this thesis possible.

The first person I would like to thank is my supervisor, Nicola Nicolici, whose encouragement, enthusiasm to high-quality work and motivating discussions helped me in all the time of research for and writing of this thesis. In particular, I am deeply grateful to the support and freedom he gave me on my summer travel. It was a great pleasure to me to conduct this thesis under his supervision.

Needless to say, I am grateful to all of my colleagues in the computer-aided design and test research group at McMaster University, Bai Hong Fang, Henry Ko, David Lemstra, Adam Kinsman, David Leung and Ehab Anis, those with whom I have shared both happiness and sadness as a graduate student. I also would like to thank Theo Gonciari and Paul Rosinger from University of Southampton. In particular, I would like to express my appreciation to Theo Gonciari for his early work and insightful comments on this thesis. Krishnendu Chakrabarty from Duke University also provided constructive suggestions on this thesis and I am deeply indebted to him. Many thanks also go to the faculty, administrative and technical members in Department of Electrical and Computer Engineering at McMaster University, whose assistance was vital for the research. In particular, I would like to express my gratitude to Cheryl Gies, who is of great help whenever needed.

My father Zhiqing Xu, my mother Qinghua Zhang, my wife Yujie Cui and my sister Dongdong Xu have loved, trusted and supported me all the time in my life. I cannot live through this Ph.D period without their love and patience. Words cannot express my love and gratitude to my family.

Abstract

Manufacturing test has established itself as an enabling technology for the system-on-a-chip (SOC) design paradigm. Although the IEEE Std. 1500 for embedded core test and prior work on test access mechanism design ease the SOC testing process, there are several emerging problems not treated explicitly or cost-effectively by the state-of-the-art methods. For example, the number of clock domains is stepping-up and the specific test access and isolation problems posed by multi-frequency cores have not been treated explicitly; detection of timing failures, which are predominant in deep sub-micron technologies, requires support from the test access architectures for at-speed application of two successive patterns; glue logic defined by system integrators for product differentiation is increasing in size and its test can adversely influence either the chip area or testing time; the continuous growth in the size of SOCs leads to multiple levels of design hierarchy, which imposes an additional constraint on the design and reuse of test architectures.

This dissertation shows how at-speed test application, without test hazards, can be achieved for cores with multiple clock domains, by embedding small custom logic in the wrapper. It also introduces novel test architectures for SOCs containing unwrapped logic blocks without any loss in fault coverage, by combining dedicated bus-based test access mechanism and functional interconnects for test data transfer. With some minor changes, this new test architecture can be utilized to effectively apply two-pattern test for core-based SOCs, which is essential for detecting delay faults and CMOS stuck-open faults. Test access mechanism design algorithms for the new SOC test architectures are also proposed and design trade-offs between test area and testing time are discussed. Finally, this dissertation presents a new framework for the design space exploration of multi-level test access mechanisms, which is important for future hierarchical SOCs that reuse past-generation SOCs with existing test infrastructure as internal mega-cores.

List of Abbreviations

ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
CMOS	Complementary Metal Oxide Silicon
CTL	Core Test Language
CUT	Core under Test
DFT	Design for Testability
FF	Flip-Flop
FIFO	First-In First-Out
FSM	Finite State Machine
HDL	Hardware Description Language
I/O	Input/Output
IC	Integrated Circuit
ILP	Integer Linear Programming
IP	Intellectual Property
ITRS	International Roadmap for Semiconductors
LFSR	Linear Feedback Shift Register
LSSD	Level-Sensitive Scan Design
MFCW	Multi-Frequency Core Wrapper
MISR	Multiple Input Signature Analyzer
NC-PI	Non-Controlled Primary Inputs
PC-PI	Parallelly-Controlled Primary Inputs

PCB	Printed Circuit Board
PI	Primary Input
PLL	Phase-Locked Loop
PO	Primary Output
PSI	Pseudo-Input
PSO	Pseudo-Output
RAM	Random Access Memory
SA	Signature Analyzer
SC-PI	Serially-Controlled Primary Inputs
SECT	Standard for Embedded Core Test
SFCW	Single-Frequency Core Wrapper
SFF	Scanned Flip-Flop
SOC	System-on-a-Chip
TAM	Test Access Mechanism
TAT	Test Application Time
TDC	Test Data Compression
TPG	Test pattern generator
UDL	User Defined Logic
VC	Virtual Core
VLSI	Very Large Scale Integration
VTB	Virtual Test Bus
VTB-DIU	Virtual Test Bus De-Multiplexing Interface Unit
VTB-MIU	Virtual Test Bus Multiplexing Interface Unit
WBR	Wrapper Boundary Register
WBY	Wrapper Bypass Register
WIC	Wrapper Input Cell
WIR	Wrapper Instruction Register
WOC	Wrapper Output Cell
WPI	Wrapper Parallel Input
WPO	Wrapper Parallel Output

WSC	Wrapper Serial Control
WSI	Wrapper Serial Input
WSO	Wrapper Serial Output
Wrapper SC	Wrapper Scan Chain

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Semiconductor manufacturing technology has advanced significantly over the past several decades and continues to do so. The International Technology Roadmap for Semiconductors (ITRS) [69] projects that integrated circuits (ICs) with multi-billion transistors will be manufactured by the end of this decade, with feature sizes in the range of $50nm$ and clock frequencies in the range of $10GHz$. In order to fully exploit the capabilities of the advanced process technologies and at the same time meet the shrinking product development schedules, the electronic industry is moving toward a design flow that integrates pre-designed and pre-verified cores into system-on-a-chip (SOC) platforms, based on the ”reuse” philosophy [86]. Manufacturing test, a key step at the back end of the implementation flow of very large scale integrated (VLSI) circuits, used to isolate good chips from the defective ones, has become an essential technology that enables the fabrication yield, certifies the product quality and influences the final cost of the device [17].

Recently a vast body of research [171] has been endeavored to address the SOC testing challenges. Nevertheless, several emerging problems in SOC testing are not addressed and the known methods cannot be easily adapted to solve them in a cost-effective way due to their simplified assumptions. For example, embedded cores are considered to have only a single clock domain or design hierarchy is assumed to be flattened during test development. While these assumptions were valid in the past, they are becoming obsolete for state-of-the-art SOC designs. As a consequence, the topic of this dissertation is to tackle the emerging problems in modular SOC testing.

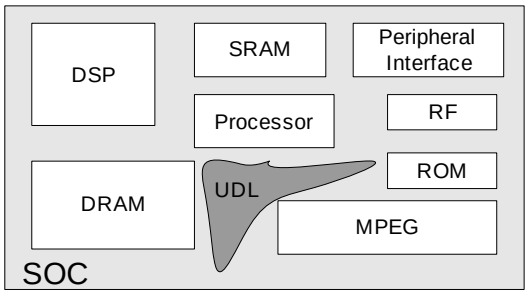


Figure 1.1: General Architecture of Core-Based System-on-a-Chip.

This chapter is organized as follows. We begin with an introduction to the core-based SOC design paradigm in Section 1.1. Section 1.2 illustrates the SOC test challenges and elaborates on several open questions that motivate this research work. Finally, Section 1.3 outlines the main contributions of this research and presents the organization of the thesis.

1.1 Core-Based SOC Design

The benefits of integrating an entire system on a single chip are manyfold. The reduction in chip count decreases the system cost because the cost of integrating multiple components is usually higher than the cost of the single SOC. The interconnect distance between components on a single silicon die is much shorter than on a board, which not only reduces timing delays and power dissipation, thus boosting the system performance, but also increases the system reliability. Further, a high integration is particularly desirable for applications where the product size is crucial, such as embedded mobile devices (e.g., cell phones or portable music players).

The main limitation of SOCs is the implementation time. To address this, SOC development is based on the design reuse philosophy, where two parties are involved: core providers and system integrators. Core providers create libraries of pre-designed and pre-verified building blocks, known as embedded cores, virtual components or macros. Embedded cores can be digital logic blocks, memories or analog (including radio frequency) circuits. Examples of cores include microprocessors, digital signal processors, analog-to-digital converters, network controllers, flash memories and peripherals. System integrators

put the SOC together by combining the available cores and their custom user-defined logic (UDL) [54], as shown in Figure 1.1.

Embedded cores come in different forms: soft, hard or firm. A soft core comes in the form of synthesizable hardware description language (HDL) code and has the advantage of being able to be easily re-targeted to different technologies. The trade-off for this flexibility and portability is unpredictability in area, performance and power. Hard cores, on the contrary, come with physical layout information and hence are technology-dependent. Despite the lack of flexibility, they are optimized for timing or power and have well-known performance parameters. Firm cores provide a trade-off between soft and hard cores. They are gate-level netlists and hence they are more predictable than soft cores, however, their size, aspect ratio and pin location can be tuned according to the system integrator's needs. Cores are often products of technology, software, and know-how that are subject to patents and copyrights. Hence, a core block represents intellectual property (IP) that the core provider licenses to the system integrator [54]. Therefore, the system integrator, who is in charge of core integration, verification, and also manufacturing test of the entire SOC (including the IP-protected internal cores), is not always entitled to make any changes to the core and is forced to reuse it "as is" (i.e., as a black box), knowing only the core's functionality without any implementation details.

In addition to the reusable cores from third-parties, SOCs often include UDLs for product differentiation and core interfacing. Because the UDLs are custom logic designed from scratch, the system integrator has full knowledge about both its functionality and structure.

1.2 Thesis Motivation

Although the design process for core-based SOCs is conceptually analogous to traditional board design, the manufacturing test process is fundamentally different [180]. In the traditional system-on-board approach, as shown in Figure 1.2(a), the IC provider performs chip design, manufacturing and test. The system integrator can assume the ICs are fault-free and he is only responsible for the design, assembly and test of the printed circuit board (PCB). Therefore, testing is usually limited to the interconnects between chips. In contrast, for the SOC approach, as shown in Figure 1.2(b), the cores are not yet manufactured when

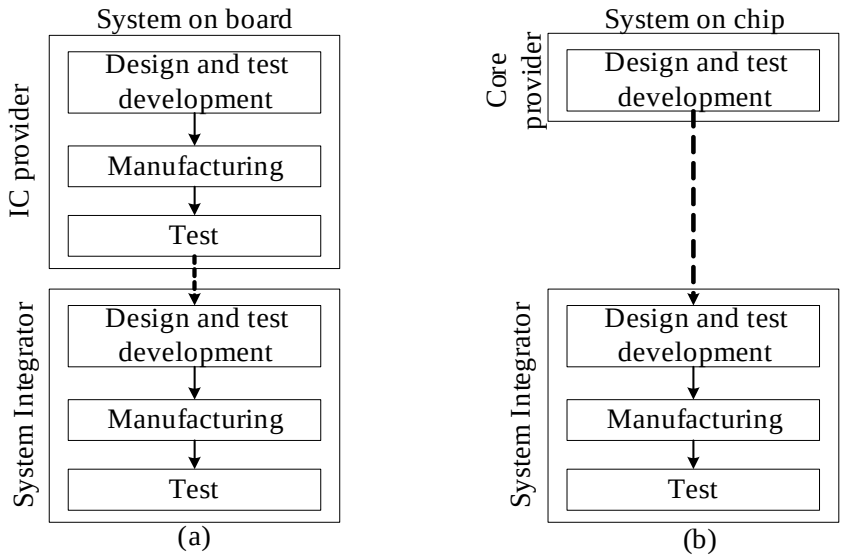


Figure 1.2: Design Development Variations between Board and Chip [180].

the system integrator puts them together, thus the core providers cannot test their products for manufacturing defects. Manufacturing test can only be done by the system integrator once the chip is fabricated. This makes the testing of embedded cores a joint responsibility of both core providers and system integrators. Since the system integrator is usually forced to deal with cores as black boxes as discussed above, the core provider has to provide the model, the design for testability (DFT) [26] structures and the corresponding test vectors. The main difficulty for the system integrator is to provide a reliable and scalable test infrastructure for accessing the cores. This is required, since, contrary to PCBs where direct physical access to the chip pins is available, in SOCs the cores are embedded in the chip and thus, no direct access to the cores' terminals is available.

To enable the reuse of tests when a core gets embedded in multiple different SOCs, as well as to enable interoperable core-based testing of SOCs, the IEEE Std. 1500, i.e., IEEE standard for embedded core test (SECT) [67, 55], has been developed by the IEEE P1500 working group [117] (detailed in Section 3.1). A module-level test wrapper, similar to IEEE 1149.1 boundary-scan structures [132], is suggested to surround each embedded core, which allows inter-core and intra-core tests to be carried out via test access mechanisms (TAMs). IEEE Std. 1500 also standardizes a test information transfer model, i.e.,

IEEE 1450.6 core test language (CTL) [84]. IEEE Std. 1500, however, does not standardize the core's internal test methods or DFT structure, nor SOC test integration and optimization because of the differences in the test requirements for different technologies and the design styles of different cores and SOCs. Therefore, numerous strategies and algorithms in SOC test architecture design and optimization, test scheduling and test resource partitioning techniques have been proposed in the literature in order to reduce test cost of the SOC, based on the IEEE Std. 1500 test infrastructure.

Although the IEEE Std. 1500 and prior work in core-based SOC testing ease the task of system integrators for SOC test, with the ever increasing design complexity, many new problems are emerging:

- *How to deal with the increasing number of clock domains inside embedded cores?* Many embedded cores operate internally with multiple frequencies. In functional mode care has been taken so that only safe transitions exist between the different clock domains. In test mode, however, if specific test clocking strategies are not employed, the test data might be corrupted due to clock skew. How to provide a reliable test strategy for such embedded multi-frequency cores is an open question.

- *How to apply delay tests for embedded cores?* With the continuous increase in speed of SOCs, verifying only logic correctness during manufacturing test is not sufficient to guarantee high quality products. To find out timing-related defects, dedicated at-speed delay test is required, which typically needs the application of consecutive test vector pairs. This consecutive testability, however, is not covered by prior research in modular SOC test architecture design.

- *How to effectively and efficiently test unwrapped logic blocks?* Embedded cores may have a large number of primary inputs (PIs) and primary outputs (POs) , which can incur a large DFT area overhead when wrapping them. More importantly, IEEE Std. 1500 wrapper may also result in performance penalty because each wrapper cell adds a multiplexer on circuit's functional path. As a result, unlike the assumptions made by prior work that all embedded cores are 1500-wrapped, system integrators usually prefer to keep certain cores unwrapped in practice. In addition, the unwrapped UDLs

are typically distributed on chip and may contain many storage elements. The inter-core test strategy provided by IEEE Std. 1500 treats these unwrapped logic blocks (including both unwrapped cores and UDLs) as non-scanned circuits and hence cannot provide sufficient controllability and observability for the circuits' internal memory elements, thus leading to fault coverage loss. Motivated by the above fact, a modular test strategy that provides the same testability for unwrapped logic blocks as when they are 1500-wrapped is required.

- *How to effectively and efficiently test SOCs with multiple levels of hierarchy?* Most of the prior work in this field assumes that the SOC hierarchy is flattened during test. This assumption, however, is becoming unrealistic. This is because, with the increase of reusability, sometimes the older-generation SOCs are themselves used as embedded cores in the new-generation SOC designs, thus leading to a hierarchical SOC design paradigm. Therefore, novel test strategies for hierarchical SOCs need to be developed.

Based on the above observations, the contributions and organization of this dissertation are presented in the next section.

1.3 Thesis Organization and Contributions

The remainder of this dissertation is organized as follows. First the background for this research work is introduced in Chapter 2, including the basic VLSI manufacturing test concepts, various delay fault testing and multi-frequency testing strategies. Chapter 3 describes IEEE Std. 1500 in detail and presents a comprehensive overview of prior work in core-based SOC testing domain. Various test access strategies and the relevant solutions available for the design and optimization of SOC test architectures are reviewed.

Motivated by the fact that most SOCs today include embedded cores that operate in multiple clock domains, Chapter 4 shows how at-speed test application can be achieved without test hazards by embedding custom logic in the multi-frequency core wrapper. The introduced limited DFT hardware can synchronize the external tester channels with the core's internal scan chains in the shift mode, and provide safe at-speed capture via a careful

capture window design. In addition, wrapper optimization algorithms are proposed, which efficiently utilize the available tester bandwidth and guarantee low test application time (TAT) within the given power ratings.

Although the at-speed testing strategy proposed in Chapter 4 is able to uncover some timing-related defects, to increase circuit reliability and manufacturing yield, system integrators are constrained to develop dedicated delay tests since more delay faults are emerging with the advancement of the VLSI fabrication technology. Therefore, in Chapter 5, we propose a novel architecture for two-pattern test of core-based SOCs, which can be used to achieve a reliable coverage of delay faults and complementary metal oxide semiconductor (CMOS) stuck-open faults. The proposed solution combines the dedicated bus-based test access mechanism and functional interconnects for test data transfer, in order to provide full controllability of the wrapper input cells in the two consecutive clock cycles required by two-pattern testing. This technique, however, introduces test schedule conflicts among different cores and hence increases the overall SOC test application time. Consequently, new algorithms for TAM design and test scheduling are proposed and design trade-offs between DFT area and testing time are discussed.

The main idea behind the proposed architecture for two-pattern test in Chapter 5 is a modular strategy that applies the test stimuli for the CUT through the functional interconnect. Following the same idea, Chapter 6 proposes solutions for reducing the number of wrapper boundary register cells without loss of the SOC test architecture's modularity and scalability for one-pattern tested cores. By utilizing the functional interconnect topology and the wrapper cells of the surrounding cores to transfer test stimuli and test responses, the wrapper of some cores can be removed without affecting the controllability/observability of the SOC. We present novel modular SOC test architectures for concurrently testing both the 1500-wrapped cores and unwrapped logic blocks. Since wrapper cells of cores that transfer test stimuli and test responses for unwrapped logic blocks become shared resources during test, conflicts arise during test scheduling that will negatively impact the SOC TAT. As a consequence, to alleviate this problem, we also present optimization algorithms for the proposed SOC test architectures.

Chapters 4, 5 and 6 present solutions for several emerging problems in testing SOCs

that have one level of hierarchy (SOC and cores). With the increase of reusability, past-generation SOCs might be used as embedded cores in next-generation designs, and hence an emerging problem is how to test such SOCs that have multiple levels of hierarchy. This problem is tackled in Chapter 7, which describes a new framework for the design and optimization of an hierarchical SOC test architecture. We first explore the concept that TAMs on the same level of design hierarchy employ multiple frequencies for test data transportation. Then, we extend this concept to hierarchical SOCs and, by introducing frequency converters at the inputs and outputs of the hierarchical cores, the proposed solution not only removes the constraint that the system level TAM width must be wider than the internal TAM width of hierarchical cores, but also facilitates rapid exploration of the trade-offs among TAT, test power and DFT hardware.

Finally, Chapter 8 summarizes the presented research work and concludes this thesis.

Chapter 2

Background

This chapter presents the background knowledge in VLSI testing, the process of identifying defects in an IC, for a better understanding of this dissertation. We first describe the manufacturing test concepts in Section 2.1, which lay the foundation for SOC testing. Sections 2.3 and 2.2 present the advancement of knowledge in delay fault testing and multi-frequency testing over the past decades, respectively. Finally Section 2.4 concludes this chapter. Some of the descriptions and figures in this chapter are excerpted from [13, 158].

2.1 VLSI Testing Concepts

The basic principle of manufacturing test is illustrated in Figure 2.1. *Circuit under test* can be the entire chip or only a part of the chip (e.g., a logic block or a memory core). *Input test vectors* are binary patterns applied to the inputs of the circuit and the associated *output responses* are the values observed on the outputs of the circuit. Using a *comparator*, output responses are checked against the expected *correct response data*, which are obtained through simulation prior to design tape-out. If all the output responses match the correct response data, the circuit has passed the test and it is labeled as fault-free, otherwise, the circuit is said to be erroneous, and designers can either make a complete failure diagnosis or simply throw it away. *Automatic test equipment* (ATE), a powerful computer and measurement system operating under the control of a *test program*, is typically used to control the process of testing VLSI devices.

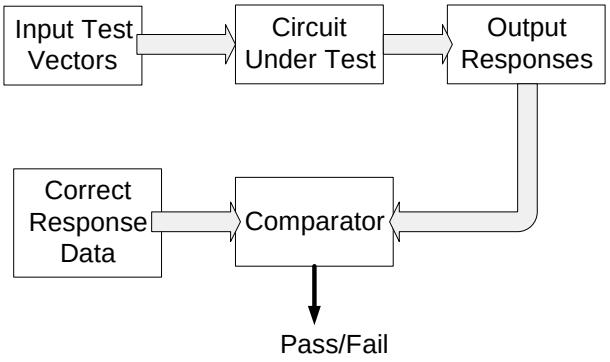


Figure 2.1: Principle of VLSI Testing.

Manufacturing test method used to identify defective chips can be divided into *functional test* and *structural test* . A complete functional test usually needs to check every entry of the truth table, which leads to extremely long testing time and makes it infeasible for complex digital systems [13] (one exception is the test of semiconductor memories due to their regularity and simple operations). Structural test, on the other hand, depends on the structure of the design. One of the greatest advantages of structural test is that it allows us to develop structural search algorithms to achieve efficient testing strategies, based on the fault modeling technique, discussed in the following.

2.1.1 Defect vs. Fault Modeling

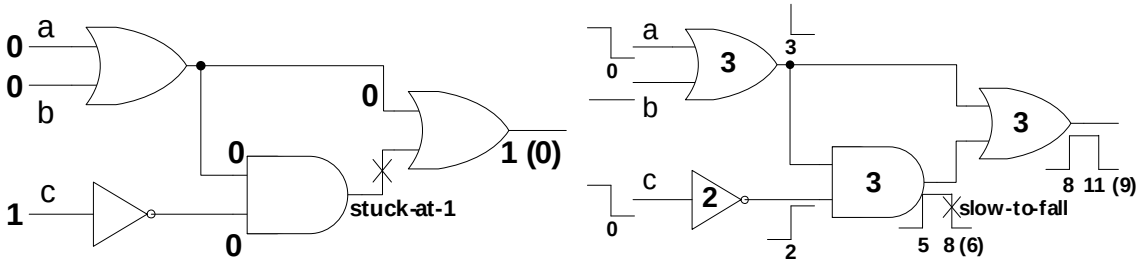
A defect in an electronic system is the unintended difference between the implemented hardware and its intended design [13]. Various kinds of defects exist in VLSI circuits [60]:

1. Process Defects - for example, missing contact window, parasitic transistors, oxide breakdown.

2. Material Defects - for example, bulk defects (cracks, crystal imperfection), surface impurities.

3. Age Defects - for example, dielectric breakdown, electromigration.

4. Package Defects - for example, contact degradation, seal leaks.



(a) (b)

Figure 2.2: Commonly Used Fault Models: (a) Stuck-at Fault; (b)Delay Fault.

However, it is very hard to generate tests for every possible type of physical defects. Because the defects lead to faulty behaviors, they are usually modeled at a certain level of design abstraction for efficient testing, such as behavioral level, logic/gate level or transistor level. Fault models at behavioral level usually have no clear correlation to manufacturing defects and hence are used more often in design verification rather than manufacturing test. Transistor level fault models are also known as technology-dependent faults and are mainly used in analog circuit testing. Fault models based on logic gates (i.e., circuit is modeled as an interconnection of boolean gates, called netlist) are technology-independent and over time have been proven to be very efficient for testing digital circuits [13]. In the following we discuss some commonly-used fault models.

- **Stuck-at Fault**:

  Stuck-at fault model, specifically, single stuck-at fault model is the fundamental fault model in digital testing, and it is also the most popular fault model used in the industry. In stuck-at fault model, each connecting line in the structural netlist can have only two types of faults: stuck to a logic value 0 (SA0) or 1 (SA1), as shown in Figure 2.2(a). In general, several stuck-at faults can be simultaneously present in the circuit, however, because the combinations of multiple stuck-at faults are prohibitively large, it is common practice to model only single stuck-at faults [13]. This single stuck-at fault model (based on the assumption that only one line is faulty at a time) has been extensively studied and is shown to be very effective to verify the logic correctness of the circuit [13].

- **CMOS Stuck-Open and Stuck-Short Faults**:

  Case [14] found that the input-output behavior of a faulty CMOS logic circuit cannot be exactly represented by the stuck-at fault model. Since a CMOS logic gate consists of more than one transistor, instead of modeling faults at the gate level, these fault models assume just one transistor to be *stuck-open* or *stuck-short*. The effect of stuck-open faults is to produce a floating state at the output of the faulty logic gate. The effect of a stuck-short fault is to produce a power supply to ground conducting path.

- **Delay Fault**:

  A digital system may also have timing failures without logic errors, i.e., it fails to operate at the specified speed but can produce correct outputs at slower or faster speed. Digital systems are usually synchronized by clock signals. Therefore, if the combinational logic elements cannot attain steady state within a clock period, the excessive propagation delay violates the *setup time* constraint of storage elements (i.e., flip-flops or latches). Alternatively, if the propagation delay through combinational logic is too short, it violates the *hold time* constraint of the storage elements. We mainly consider the former case because defects leading to excessive delay happen more often during manufacture. For example, as shown in Figure 2.2(b), because of the transition delay the output signal turns low at 11 instead of 9, if the period of system clock is 10, the storage element next to this logic will load the wrong value.

  Commonly used delay fault models include the *transition fault model* (also called *gross-delay fault model*) and the *path delay fault model*. Transition fault model is based on the assumption that only a single gate delay is changed. It has the advantages of easy test set generation and comparably small test set size, but it is less accurate due to its simplistic assumption. Path delay fault model, however, considers the cumulative propagation delay of a combinational path and hence is much more accurate, but the test generation is extremely hard and normally requires a large number of test patterns. In addition, although path delay fault test generation has been researched for two decades, a large part of path delay faults remains untestable for state-of-the-art designs [98].

Different fault models may require different kinds of test pattern application strategies. For example, each stuck-at fault can be detected by a single test pattern. The delay fault, however, requires to examine signal transitions and hence consecutive test vector-pairs are needed, in which the first vector V_1 initializes the circuit while the second vector V_2 causes the desired transitions. Similarly, CMOS stuck-open fault is also detected by consecutive test vector-pairs, where V_1 is used to initialize the circuit appropriately and V_2 detects a stuck-at fault on the gate output. CMOS stuck-short fault is usually tested by quiescent current (I_{DDQ}) measurement because of its specific fault behavior.

2.1.2 Test Pattern Generation

The task of the automatic test pattern generation (ATPG) process is to find a set of excitation vectors that cover a sufficient large portion of the fault set by the adopted fault model. To achieve this, ATPG algorithms first inject a fault into the circuit, and then use a variety of mechanisms to activate the fault and cause its effect to propagate through the hardware and manifest itself at circuit outputs [13]. ATPG is an extremely complex process, which may take weeks even months to achieve a high fault coverage for large designs, even with the help of specific test hardware.

In essence ATPG algorithms can be divided into combinational ATPG algorithms and sequential ATPG algorithms. The first and seminal combinational ATPG algorithm is D-algorithm proposed by Roth [147], which established the calculus and algorithms for ATPG using D-cubes. The next significant improvement is Goel's PODEM algorithm [43]. He efficiently used path propagation constraints to limit the ATPG algorithm search space. Fujiwara and Shimono's FAN algorithm [38, 39] further restricts the ATPG search space significantly by efficiently constraining the backtrace and considering signal information. Sequential ATPG algorithms are even more complex due to the unknown internal memory states at the beginning of test. As a result, the test must first initializes the internal states, and after test inputs are applied for several clock cycles the final states of memory elements must be propagated to primary outputs. The lack of controllability and observability to the internal nodes dramatically increases the test generation complexity. Commonly used algorithms for sequential ATPG are backward justification procedures through time-frame expansion [13]. To decrease the number of time-frames and hence ATPG complexity, it is

common practice to limit the sequential steps during test, using specific test hardware, as discussed in Section 2.1.3.

Since a test pattern can potentially detect several faults at the same time, ATPG algorithms are usually combined with fault simulation techniques during test generation. One possible approach is to start from a random set of test patterns. Fault simulation then determines how many faults are detected and these faults are then declared to be covered and removed from the fault set. After this trial random pattern phase, an ATPG algorithm is executed to target the remaining undetected faults. Whenever a deterministic test pattern is acquired, fault simulation is utilized again to find other faults that can be detected by this pattern. This process is repeated over and over until a desired fault coverage is achieved. It should be noted that the test patterns generated from ATPG process feature a large portion of ”don’t-care” bits (i.e., they can be assigned to any logic value without affecting the fault coverage of the test set), and test data volume can be dramatically reduced by manipulating these unspecified bits using efficient compaction and/or compression methods.

2.1.3 Design for Testability (DFT)

DFT is the design process which embeds special hardware for testing purpose only. With the ever increasing transistor to pin ratio in VLSI circuits, test generation is impractical without introducing DFT logic. The main purpose of DFT is to increase the controllability (i.e., the ability to set a particular logic signal to a 0 or a 1 within the circuit) and the observability (i.e., the ability to observe the state of a logic signal within the circuit) of the circuit’s internal node. Two widely accepted DFT methodologies are illustrated in this section: scan based DFT and built-in self-test (BIST).

Scan based DFT: The main idea for scan based DFT is to obtain controllability and observability for internal flip-flops. This is done by replacing all or part of the flip-flops of the circuit with scanned flip-flops (SFFs). In the test mode, these scanned flip-flops functionally form one or more long shift registers (called *scan chains*). The input/output (I/O) of these scan chains are connected to primary I/Os of the chip, hence, all the SFFs can be set to any desired state by shifting those logic states into the scan chains. Similarly, the states of these SFFs can be observed by shifting out the contents of the shift registers.

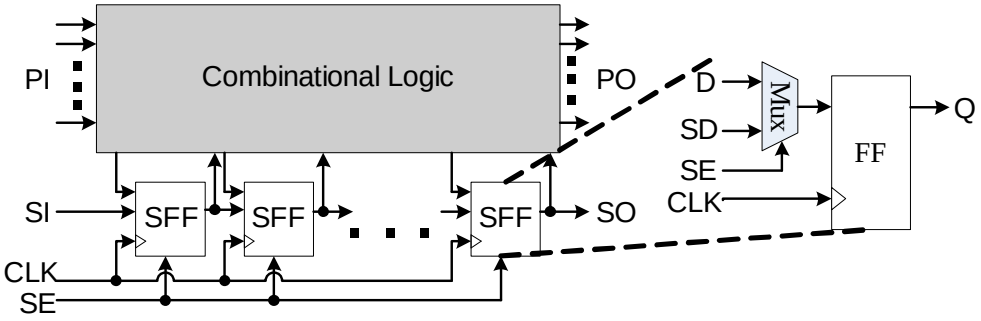


Figure 2.3: A Single Scan Chain Design Schematic.

There are various types of implementation of SFFs, e.g., mux-based SFF, double latched SFF, level sensitive scan latches SFF [1, 13]. The one illustrated in Figure 2.3 and considered in this dissertation is the mux-based SFF, which is composed of a multiplexer and a D flip-flop. The SFF has two data inputs: the original data input *D* and the scan-data input *SD*, a scan enable signal *SE* controls which input is stored in the flip-flop. The scan chain is constructed by connecting the data output *Q* of one SFF to *SD* signal of the following SFF, as illustrated in the figure. Applying a test vector using scan based DFT implies three steps: (i) scanning-in the test stimuli, i.e., SE is set to 1 (test mode), the data is loaded from the scan input *SI* into the scan chain in N_{sc} clock cycles, where N_{sc} is the scan chain length; (ii) capturing the circuit responses, i.e., SE is set to 0, the data is loaded from the combinational logic outputs in 1 clock cycle; and (iii) scanning-out the test responses, i.e., similar to scan-in where the circuit responses are scanned out. It should be noted that the last step is usually performed simultaneously with the first step in a pipelined fashion, where new control values are loaded.

The amount of time needed to perform the above procedure for all the test patterns in a test set is mainly dependent on the scan chain length and the number of test patterns. To reduce test application time, flip-flops are usually arranged in multiple scan chains in which test data can be shifted in/out in parallel. When these multiple scan chains have varying length, the TAT for one test pattern is dependent on the length of the longest scan chain. Each scan chain requires dedicated *SI* and *SO* pins, if extra pins are not available, however, the scan I/O pins can be multiplexed with the normal primary I/O pins under the control of the test control signal SE.

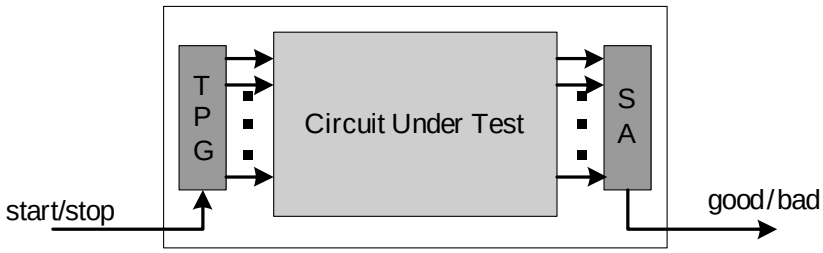


Figure 2.4: Conceptual BIST Design Scheme.

Full-scan design, in which all flip-flops are replaced by SFFs, forms the foundation on which most other DFT techniques are built and is considered the best DFT discipline [13], however, at the expense of penalties in area and performance. To reduce the overhead of full-scan design, sometimes only part of flip-flops inside the circuit are transformed into SFFs (called *partial-scan*) [3, 24]. Since most embedded cores are full-scanned today to achieve high fault coverage, partial-scanned cores are not considered in this dissertation.

It is important to note that chips with scan design in test mode might dissipate more power than they do in normal mode [177], because more transitions might happen in test mode than in the normal mode. As a result, to avoid destructive testing and at the same time reduce testing time, test engineers must trade-off scan shift frequency and test power dissipation when selecting their test strategy. This is especially important for complex chips that contain a large number of flip-flops.

BIST: The motivation behind BIST is to decrease the test cost by using low-cost ATEs. In traditional test, test stimuli are loaded to the chip from ATE and test responses are shifted out and compared with the correct response data stored in the ATE memory. As transistor to pin ratio and circuit operating frequencies continue to increase, there is a growing gap between the ATE capabilities and circuit test requirements (especially in terms of speed and volume of test data). Moreover, the increasingly long time for test pattern generation is also a major concern in today's testing. The enormous cost of high-end ATE and its own limitations make BIST technology an attractive alternative to external test for complex chips.

In a BISTed circuit, part of the circuit is used to test the circuit itself, i.e., a test pattern generator (TPG) generates a set of test stimuli and a signature analyzer (SA) analyzes the

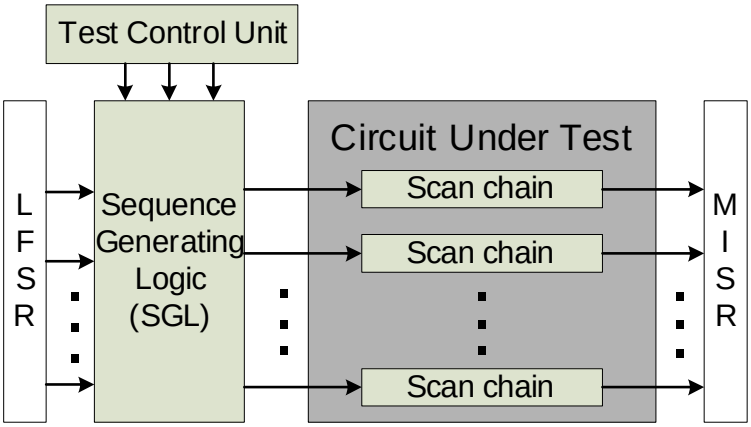


Figure 2.5: Scan-Based BIST Architecture.

test responses and makes the decision of good/bad chip, as shown in Figure 2.4. BIST needs only an inexpensive tester to initialize BIST circuitry and to inspect the final results, which obviously decreases the test execution cost. The most common BIST setup is to use a linear feedback shift register (LFSR) as the TPG, and a single input signature analyzer (SISR) or a multiple input signature analyzer (MISR) as the SA [4, 13].

Logic BIST can be further divided into pseudo-random BIST and deterministic BIST. The former implies that the TPG is allowed to run freely for a given number of clock cycles, which may cause low fault coverage because of the random-resistant faults inside of the circuit. The latter implies that the TPG is controlled using techniques such as ”bit-flipping” [169], ”bit-fixing” [160], ”re-seeding” [148] and ”pattern-mapping” [23] in order to generate deterministic test vectors [59, 88]. When the two are combined, mixed-mode BIST solutions are derived. The main challenge for logic BIST lies in the computational overhead required to synthesize compact and scalable TPGs and SAs, such that high fault coverage is achieved in low testing time with limited interaction to external equipment. The state-of-the-art logic BIST DFT is to combine scan design and BIST in a mixed-mode solution, i.e., combining the pseudo-random BIST and deterministic BIST methodology to generate the scan inputs of the circuit under test, as shown in Figure 2.5.

2.1.4 Cost of Test

ITRS [69] anticipated that, if the current trends are maintained, the cost of testing a transistor will approach and may even exceed the cost of manufacturing it at the end of this decade. Research in VLSI testing is in essence to find reliable yet low cost test strategies.

Nag et al. [127] pointed out that the cost of test (C_{test}) can be computed as follows:

$$C_{test} = C_{prep} + C_{exec} + C_{silicon} + C_{quality} \tag{2.1}$$

where:

- C_{prep}: the fixed costs of test generation, tester program creation, and any design effort for incorporating test-related features (all nonrecurring costs, including software systems). Test generation cost depends also on die area and personnel cost. Tester program preparation is a function of test generation cost. Test-related design effort is a function of the extra silicon area required to incorporate testability-enhancing features such as scan and BIST.

- C_{exec}: the cost of test execution. It is mainly dependent on ATE execution cost, including tester setup time, tester execution time, capital cost of the tester (as a function of die area and number of I/O pins) and capital equipment depreciation rate.

- $C_{silicon}$: the cost required to incorporate DFT features. It is a function of wafer size, die size, yield, and the extra area required by DFT.

- $C_{quality}$: the cost of imperfect test quality. That is, the profit loss from performance degradation caused by the added DFT circuitry, the cost of test escape (the relation between fault coverage and fault occurrence - or simply yield), and cost of good dies being deemed faulty by imperfect test.

Test engineers must make trade-offs among the above costs and apply a test strategy to decrease the overall cost. For example, introducing more DFT logic can reduce C_{exec} at the expense of an increase in $C_{silicon}$ and $C_{quality}$. A high delay fault coverage normally can reduce $C_{quality}$ at the expense of increased C_{prep} and C_{exec}.

2.2 Scan Design with Multiple Clock Domains

After introducing the basic concepts of VLSI testing, this section describes the main difficulties of testing designs with multiple clock domains and discusses the corresponding solutions, which lay the foundation for one of the focuses of this dissertation.

Many of today's complex chips, especially communications chips often have many asynchronous data streams coming into them, making multiple clocks a common necessity. Furthermore, design for low power often requires different parts of the chip to operate at their minimum permitted frequency. Therefore, designs with tens or even hundreds of clock domains are no longer isolated cases in industry at present [164].

Ideally, clock signal triggers all the memory elements it drives simultaneously. However, due to a variety of process and environment variations, the relative arrival time for different register points is different. This spatial variation in arrival time is commonly referred to as *clock skew* [136], which is caused by static mismatches in the clock paths and differences in the clock load. Clock skew often limits the operational speed of the circuit, and can invalidate the operation of the circuit in some circumstances. While clock skew within the same clock domain can be well controlled below a desired limit through clock tree synthesis, clock skew between different clock domains, however, is often indefinite, and hence transitions between different clock domains are usually made safe with the help of *synchronizers*[1] or First-in First-out (FIFO) [136] memory elements. Although clock skew problem is avoided through careful design in normal functional mode, when the storage elements are chained in test mode, clock skew can still occur between different clock domains [158]. For scanned circuit, we can separate this problem into two subproblems: clock skew during shift and clock skew during capture, discussed in the following sections.

2.2.1 Avoiding Clock Skew during Scan Shift

During the shift operation, the scan chain operates as a shift register. When scan chains are built from flip-flops within a single clock domain, they can operate correctly without clock skew problem. However, when scan chains are built from flip-flops within multiple

[1] A circuit implemented to resolve asynchronous signal as a 1 or a 0 is called a synchronizer [136].

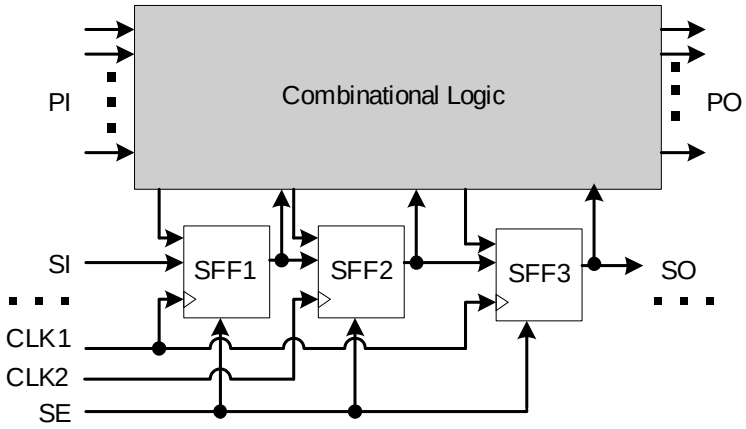


Figure 2.6: Hazard of Clock Skew during Scan Shift.

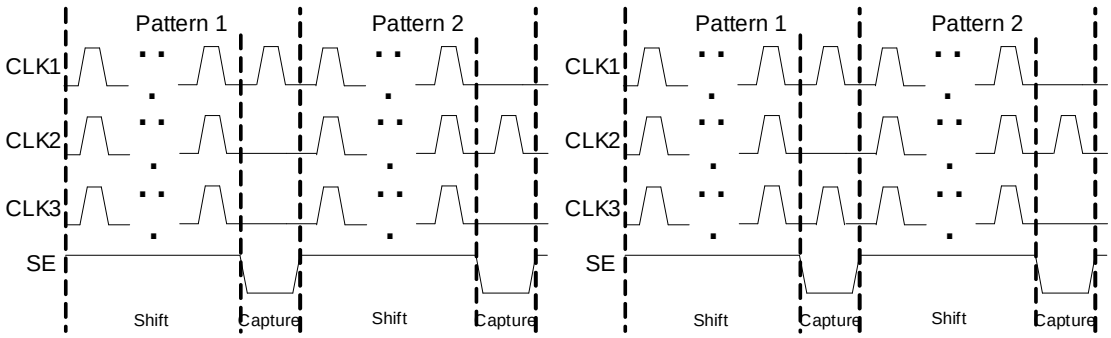
clock domains, clock skew may occur and corrupt test data, as described in the following example.

Example 2.1 *As shown in Figure 2.6, SFF1 and SFF3 are clocked with* CLK1*, while SFF2 is clocked with* CLK2 *in the functional mode. Although during test all the three flip-flops are clocked with the same test clock, this test clock traverses their original clock trees. As a result, the test clock to the two domains may still be misaligned.*

- *If CLK1 triggers before CLK2. In the first clock cycle, a value will be loaded into SFF1, and then in the same clock cycle this value will be loaded incorrectly into SFF2. The error is propagated in the following pulse of CLK1, where SFF3 will load the incorrect value from SFF2. Hence, SFF2 in this case operates rather as a combinational buffer instead of a storage element.*

- *If CLK1 triggers after CLK2. In this case, SFF1 and SFF2 can register the correct value, however, SFF3 now suffers the same problem as what SFF2 did in the previous situation. Since CLK2 is triggered before CLK1, SFF3 loads the value one cycle earlier.*

Fortunately, for scan design with multiple clock domains, the problem of clock skew during shift can be efficiently solved by one of the following techniques:



(a) Pulsing single clock domain (b) Pulsing independent clock domains together

Figure 2.7: Timing Diagram for Multi-Frequency Design with Combinational ATPG Help.

- Having separate scan chains for each clock domain. This is the safest and easiest solution. Unfortunately, the number of scan chains in a design is usually limited by the affordable scan I/O number. If the number of clock domains exceeds this limitation, the designer has to resort to other solutions. In addition, this solution often leads to unbalanced scan chain length, which will increase test application time.

- Using *level-sensitive scan design* (LSSD) SFFs [13] rather than mux-based SFFs to build scan chains. LSSD has an additional clock for test purpose and hence does not suffer from clock skew during shift problem [34]. However, this additional feature of LSSD comes with two costs. First, It requires the routing of the additional test clock with stringent timing accuracy. Second, LSSD SFFs are larger than mux-based SFFs.

- Grouping the flip-flops from the same clock domain together and adding lockup latches in between different clock domains. Adding the lockup latches as buffers causes half cycle delay and hence reduces the possibility of registering values in the earlier cycle. This strategy is commonly used in industry.

2.2.2 Avoiding Clock Skew during Scan Capture

The above techniques used to solve clock skew problem during scan shift, unfortunately, cannot solve the skew problem during scan capture. This is because there can be paths

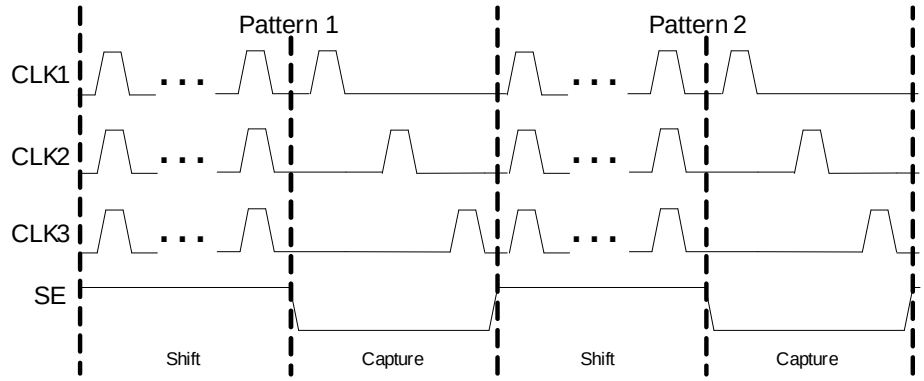


Figure 2.8: Timing Diagram for Multi-Frequency Design with Sequential ATPG Help.

between clock domains in both directions, and simply skewing the clock inputs might not help. Hardware approach (e.g., d-mimic model [158]) to solve this problem requires even larger scan cells with one more clock signal during capture compared to LSSD SFFs, and is not utilized in practice since IC vendors do not provide this type of cells. As a result, we mainly discuss the solutions for this problem from the ATPG perspective here.

The scan capture strategy in multi-frequency testing is strongly related to the ATPG technique. To illustrate this, we use an example circuit with three clock domains. Each clock domain has one scan chain and hence eliminates the clock skew problem during shift. The clock skew during capture can be avoided by one of the following techniques [158]:

- The easiest and safest way to avoid clock skew during capture is to pulse only one clock per pattern, as shown in Figure 2.7(a). This method leads to short ATPG running time, however, at the expense of a large number of test patterns. Since the example design has three clock domains, this method requires up to three times pattern count of the design with single clock.

- If clock domain analysis is done prior to ATPG and two or more clock domains are shown to be independent, these clock signals can be treated as equivalent in ATPG process and hence these clock domains can be safely captured together, which decreases the number of test patterns. As shown in Figure 2.7(b), CLK1 and CLK3 are pulsed together during capture while CLK2 is pulsed alone which can decrease

pattern count without the help of complex sequential ATPG tools.

- If sequential ATPG is utilized, the capture phase can now have multiple clock cycles, as shown in Figure 2.8. This *clock concatenation* method leads to the most compact test pattern set at the expense of increased ATPG running time. Designers often combine the clock domain analysis with this method. In addition, designers often make trade-offs between ATPG complexity and test pattern count. For example, if the design has eight clock domains, the first capture phase can pulse four clocks, and the second capture phase pulses the other four clocks, and then repeat the same process. The pattern count might be increased, however, the ATPG runtime will decrease exponentially because of the much smaller number of sequential depth. It should be noted that many advanced ATPG tools [80, 110] that support clock concatenation have been available for the last few years.

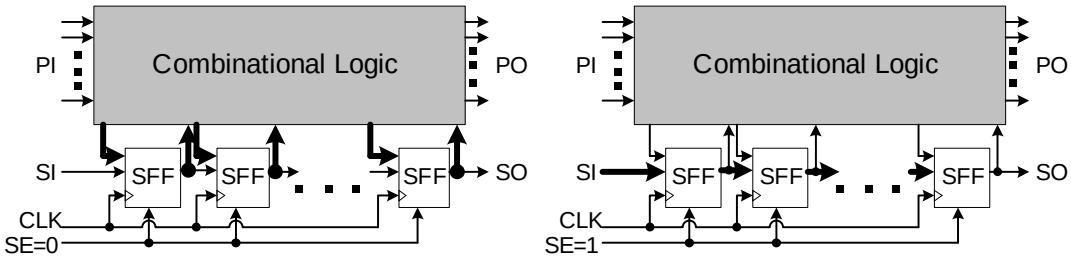
2.3 Delay Fault Testing Strategies

In addition to multi-frequency testing, another focus of this dissertation is how to efficiently and effectively detect timing-related defects in SOC devices. Consequently, we discuss several fundamental techniques for testing delay faults in this section.

There are mainly three delay fault testing strategies for scanned circuits: *enhanced scan*, *broadside testing* (also called *functional justification* or *last-shift capture*), and *skewed-load testing* (also called *scan shifting* or *last-shift launch*).

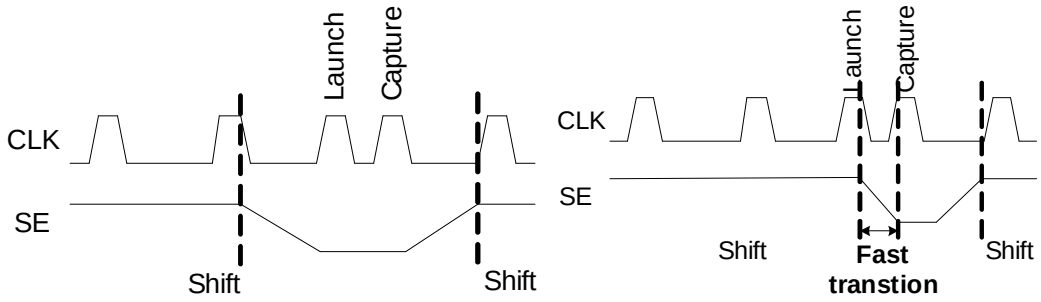
DasGupta et al. [28] described a Scan-Hold Flip Flop (SHFF), which stores two bits instead of just one bit. Although SHFF was originally designed to isolate the scan and non-scan portions of a circuit, this technique finds more applications in applying vector-pairs for delay fault testing, called enhanced scan. Enhanced scan flip-flops allow a pre-scan of any arbitrary vector pairs and the two vectors are applied to the circuit in consecutive clock cycles, which makes the test generation much easier by considering combinational logic only. The disadvantages of using enhanced scan flip-flops are the associated high DFT area overhead, long test application time and high volume of test data.

Generating tests for delay faults with standard SFFs (i.e., each SFF stores only one bit)



(a) Launch for broadside testing (b) Launch for skewed-load testing

Figure 2.9: Data Flow of Launch Pattern for Broadside Testing vs. Skewed-Load Testing.



(a) Broadside testing timing (b) Skewed-Load testing timing

Figure 2.10: Timing Diagrams of Broadside Testing vs. Skewed-Load Testing.

corresponds to a two time-frame sequential circuit test generation [98]. Primary inputs are fully controllable in both time frames. For the states of the internal flip-flops, the first vector V_1 is scanned in and hence fully controllable, the second vector V_2 can be acquired by either (i) applying a one-bit shift to the scan register, called skewed-load testing; or (b) propagating V_1 through the combinational logic in the functional mode, called broadside testing. The data flow of launching pattern for the two techniques is shown in Figure 2.9, and the timing diagrams are shown in Figure 2.10.

To effectively detect timing-related defects, the delay faults need to be activated at rated speed. However, to perform such at-speed test, for both broadside testing and skewed-load testing, it is not necessary to load/unload data at functional frequencies, but only the launch and capture must be done at-speed [59, 98]. The low frequency shifting not only avoids destructive testing with high test power during scan shift phase, but also makes high-speed

scan chain design unnecessary. How to apply at-speed launch and capture, however, leads to the specific advantages and disadvantages in broadside testing and skewed-load testing. The test generation in skewed-load testing is simpler because the circuit can be treated as purely combinational, while broadside testing requires sequential ATPG and usually leads to longer test generation time and more test patterns. The main drawback for skewed-load testing is that the control signal *SE* needs to operate at rated speed because it must toggle before and after the capture edge of the high-speed clock [109, 111, 134]. The strict delay and skew design requirement for SE signal is a big challenge for physical design of the chip. However, the timing of SE in broadside testing is not critical because both launch and capture occur in normal functional mode. In addition, test application via skewed-load testing delivers tests that may not be sensitizable in the normal operation, which can cause unnecessary yield loss [145]. In contrast, broadside testing limits the space of the possible consecutive patterns to only those that affect the timing in the normal mode, thus covering the worst-case operational behavior of the manufactured circuit.

2.4 Concluding Remarks

Although the introduction of fault modeling, ATPG, various DFT techniques, and many delay fault testing and multi-frequency testing strategies, discussed in this chapter, has significantly improved the test quality and reduced the cost of VLSI testing, innovative solutions are required to apply the existing test methodologies for core-based SOCs in a cost-effective way. This is mainly because, system integrators typically have very little knowledge of the structure of embedded cores, which makes the test of embedded cores a joint responsibility of both core providers and system integrators [180]. Motivated by the fact that challenges for testing core-based SOCs have become major contributors to the widening gap between design capability and manufacturability [17], a vast body of research work has been endeavored to this domain recently, including the industry standardization effort, the IEEE Std. 1500, as discussed in detail in the following chapter.

Chapter 3

Previous Work

Zorian et al. [180] proposed a conceptual infrastructure for SOC test, as illustrated in Figure 3.1. The basic elements of this SOC test infrastructure are:

- *Test source and sink.* The test source generates test stimuli and the test sink compares the actual test responses to the expected responses. The test source and sink can be off-chip ATE, on-chip BIST hardware, or even an embedded microprocessor [140]. It is possible to have several test sources and sinks at the same time. It is also possible that the source and the sink are not of the same type. For example, an embedded core's test source can be ATE, while its test sink is a MISR located on-chip.

- *Test access mechanism (TAM)* facilitates the transport of test stimuli from the source to the core-under-test (CUT) and of the test responses from the CUT to the sink.

- *Core test wrapper* connects the core terminals to the rest of the chip and to the TAM, which isolates the embedded core from its environment during test. It facilitates modular testing at the cost of an additional area and potential performance overhead.

Based on this conceptual test infrastructure, recently numerous test strategies and algorithms in SOC test architecture design and optimization, test scheduling and test resource partitioning have been proposed in the literature. This chapter provides a comprehensive overview of the research in this domain. IEEE Std. 1500, a standard that aims at improving ease of reuse and facilitating inter-operability for SOCs, is introduced in detail in Section

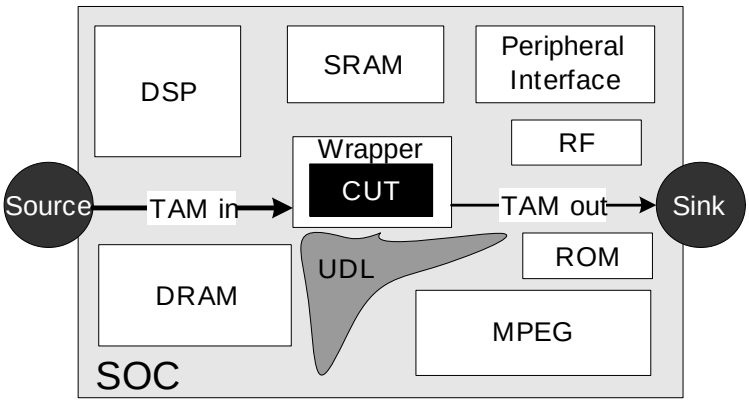


Figure 3.1: Conceptual Infrastructure for SOC Testing [180].

3.1. Section 3.2 describes different test access methods and compares their advantages and disadvantages from the test cost standpoint. Then in Section 3.3 we overview the relevant solutions available for the design and optimization of test architectures. Section 3.4 briefly discusses the various test resource partitioning techniques. Finally, Section 3.5 concludes this chapter.

3.1 IEEE Std. 1500

IEEE Std. 1500 concentrates on standardizing *the test access to embedded cores* and *the core test knowledge transfer*. The corresponding two main elements of this standard, detailed in this section, are a scalable core test wrapper [117], designed by the P1500 scalable architecture task force, and a core test language (CTL) [83], developed by the P1500 CTL task force.

3.1.1 Scalable Core Test Wrapper

IEEE 1500 wrapper, as shown in Figure 3.2, is a thin shell around a core that allows that core and its environment to be tested independently. The wrapper has three main operational modes [117, 118] : (i) functional mode, in which the wrapper is transparent and operates as if not existing, (ii) *inward-facing* test modes, in which test access is provided to the core itself and (iii) *outward facing* test modes, in which test access is provided to the

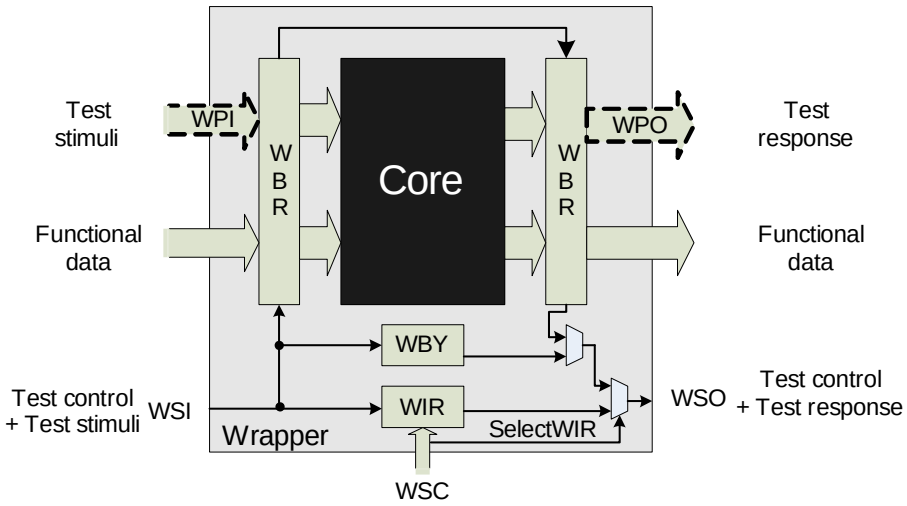


Figure 3.2: IEEE 1500 Wrapper Architecture [117].

circuitry outside the core. The wrapper has a mandatory one-bit input/output pair, wrapper serial input (WSI) and wrapper serial output (WSO), and optionally one or more multi-bit input/output pairs, wrapper parallel input (WPI) and wrapper parallel output (WPO). The wrapper also comprises wrapper boundary register (WBR) to provide controllability and observability for the core terminals and wrapper bypass register (WBY) to serve as a bypass for the test data access mechanism. In addition, the wrapper has wrapper serial control (WSC) port and an internal wrapper instruction register (WIR), used to control the different operational modes of the wrapper. We describe these elements in detail as follows:

- *Wrapper instruction register (WIR)*

  The WIR controls the operation of the wrapper. It determines the wrapper operational mode, the wrapper data register being accessed, and whether serial or parallel test access is used. It has a dual register implementation, consisting of a shift register and an equally sized update register. Instructions are scanned into the shift register and become active only when clocked into an update register.

- *Wrapper serial control (WSC)*

  The WSC facilitates the load of instructions corresponding to different operational modes into WIR, and also controls the operation of WIR. For example, the WSC

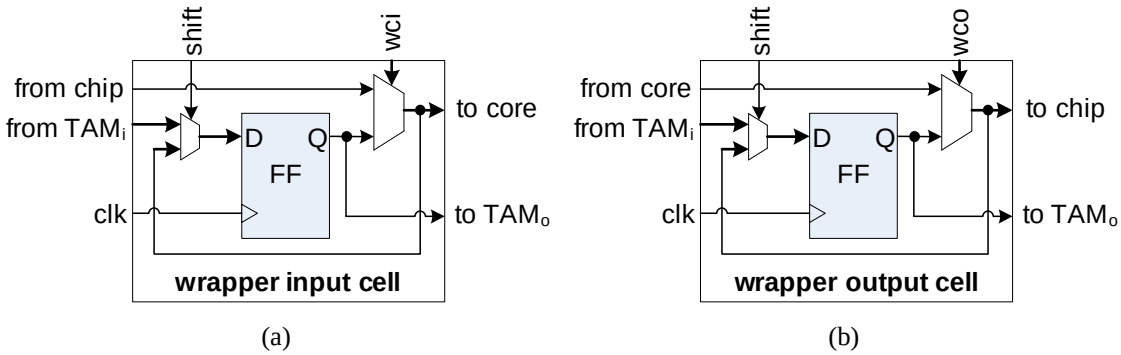


Figure 3.3: Wrapper Cell Design for (a) Core Input and (b) Core Output Terminal [117].

determines whether WSI and WSO is used to load/unload test instructions or test data. The WSC consists of the following six signals:

- *WRCK*: Clock signal for WIR, WBY and WBR.
- *WRSTN*: Asynchronous reset signal used to place the wrapper into normal functional mode when asserted.
- *SelectWIR*: Control signal that determines whether WSI and WSO are used for test instructions or test data.
- *CaptureWR, ShiftWR and UpdateWR*: Control signals when asserted, a capture, shift or update operation will be enabled for the selected wrapper register.

- *Wrapper boundary register (WBR)*

  The WBR is composed of *wrapper boundary register cells* (or simply, *wrapper cells*). Figure 3.3 shows simple implementation examples of wrapper input cell (WIC) and wrapper output cell (WOC). It is important to note that, while the basic functionality of the wrapper cells is standardized, how to implement them is not defined by IEEE Std. 1500. This flexibility allows the extension of functionality of wrapper cells.

  Test data can be fed through either serially between WSI and WSO or parallelly between WPI and WPO with a user-defined TAM width. When in the serial test mode, for core internal testing, the WBR is combined with core-internal scan chains

(if any) to shift in/out test data. While for inter-core testing, the WBR alone forms one shift register. When in the optional parallel test mode, multiple wrapper scan chains (Wrapper SCs) are constructed to reduce test application time for both core internal and external testing. How to partition and re-order the core-internal scan chains to get optimized wrapper scan chains, however, is not defined by IEEE Std. 1500.

- *Wrapper bypass register* (WBY)

  The WBY serves as a bypass for the serial test access mechanism between WSI and WSO, which reduces the test data shifting time when multiple 1500-wrapped cores are daisychained into one serial TAM. Likewise, when parallel test access is enabled, a parallel bypass register can be implemented in the wrapper to shorten the test access to other cores.

3.1.2 Wrapper Instruction Set

The various operational modes of IEEE 1500 wrapper are determined by the WSC signals and the instructions loaded into the WIR. The IEEE Std. 1500 defined a minimum set of mandatory wrapper instructions and also some optional instructions, here we mainly discuss the wrapper instructions with the following modes:

1. *Normal Mode*: The mandatory *wBYPASS* instruction configures the wrapper to work in its functional mode. In addition, the serial TAM connects WBY in between WSI and WSO to shorten test access to other cores.

2. *InTest mode*: *wCORETEST*, the main instruction to test the core's internal logic, is a flexible instruction that allows any core test to execute. For example, test stimuli/response can be fed through WSI-WSO, WPI-WPO, or even BIST inside the core. Two other core test instructions *wCORETESTWS* and *wCORETESTS* are defined additionally to standardize two types of testing using serial TAM [117].

3. *ExTest mode*: *wExTESTS* is a mandatory instruction to be implemented in order to

test the circuitry (i.e., interconnects and/or UDL) between cores. This serial instruction avoids the risk of not having enough pins to activate all of the wrappers. Alternatively, the optional *wExTESTP* instruction which allows parallel external testing is needed if the circuitry between cores is complex.

It should be noted that the wrapper instruction set can be extended with user-defined instructions, which suit the specific test requirements of the core.

3.1.3 Core Test Language (CTL)

CTL is a language for capturing and expressing test-related information for reusable cores [83, 84]. The system integrator should be able to create a complete SOC test with the help of CTL, i.e., the core test wrapper can be constructed and the appropriate TAM can be determined according to the test constraints described in CTL. Once the test infrastructure is in place, the CTL description can be translated from the core boundary to the SOC boundary.

CTL is developed on top of IEEE Standard Test Interface Language (STIL)[66]. Therefore, similar to STIL, CTL is both human- and computer-readable. As a result, it not only can be used for documentation purpose, but also can be used by automatic test integration tools. In addition, by using macro statements (M statements) in CTL rather than vector statements (V statements) in STIL, the test patterns in CTL can be reused without modification whatsoever because the system integrator can modify the vector application protocol according to his own requirements.

After introducing IEEE Std. 1500, in the following sections we discuss various SOC test architecture design and optimization techniques proposed in the literature. These techniques are not suitable for standardization and hence are not covered by IEEE Std. 1500, due to the fact that the test requirements differ for various technologies and design styles of distinct SOCs.

3.2 SOC Test Access

One of the major challenges for core-based SOC testing is to design an efficient TAM to link the test sources and sinks to the CUT [178]. There are a number of solutions for accessing the embedded cores from chip's I/O pins [179]: (i) direct parallel access via pin muxing; (ii) serial access and core isolation through a boundary scan-like architecture (also called isolation ring access mechanism); (iii) functional access through functional busses or transparency of embedded cores; and (iv) access through a combination of core wrappers and dedicated test busses. In this section, we review the above test access strategies and compare their advantages and disadvantages from the cost standpoint discussed in Section 2.1.4.

3.2.1 Direct Access

The simplest approach to test a core is to multiplex the core terminals to the chip level pins so that test patterns can be applied and observed directly [68]. The CUT can be tested as if it is the chip itself and hence the test sets/program can be reused almost without modification. It is also unnecessary to isolate the CUT using core test wrapper. While this approach apparently solves the TAM problem, due to the large number of cores and high number of core terminals, it introduces a large routing overhead. Furthermore, this approach does not scale well when the number of core terminals exceeds the number of SOC pins. In addition, since this approach does not provide access for the shadow logic of the CUT, it also results in degraded fault coverage.

Bhatia et al. [10] proposed a grid-based *CoreTest* methodology that uses a test-point matrix. Three types of test points were introduced: storage elements, embedded cells and observation test points. Direct parallel test access is provided through the "soft" netlist to these test points via the SOC I/O pins. The basic idea is to place bi-directional test points at the input and output of the embedded core and matrix accessible storage elements and observation test points in the UDL to provide sufficient fault coverage. This methodology allows the test logic and wiring used for testing UDL to be shared for testing cores and hence it reduces DFT area. However, new library cells for the proposed test points are required and global routing of the test grid is also necessary. In addition, test point selection

and test matrix assignment require an additional development effort.

3.2.2 Isolation Ring Access

IEEE 1149.1 test architecture is a widely-used DFT technique to simplify the application of test patterns for testing interconnects at the board or system level [1]. A serial scan chain is built around every chip which allows indirect yet full access to all the I/O pins. In core-based SOC testing, system testability can be obtained in a similar way, by isolating each core using boundary scan and serially controlling and observing the I/Os of the core. This ring access mechanism can be implemented either internally by the core provider or externally by the system integrator. In [165], Whetsel presented a test access architecture, which utilizes the 1149.1 test access port (TAP) for core-level DFT and a novel TAP Linking Module for the overall SOC test control.

When compared to the direct access scheme, the isolation ring access has a significant lower routing overhead and supports testing cores with more ports than chip pins. Another important advantage lies in the effective core isolation from its surrounding logic, which not only protects the inputs of the CUT from external interference and the inputs of its superseding blocks from undesired values (e.g., bus conflicts), but it also facilitates the test of the surrounding logic by putting the core into the external test mode. However, the main shortcoming of this technique is the long test application time due to the limited bandwidth caused by serial scan access. To decrease the hardware cost and performance overhead (i.e., a multiplexer delay for every path to and from the core) associated with full isolation ring, Touba and Pouya [161] described a method for designing partial isolation rings. This method avoids adding logic on critical paths, without affecting the fault coverage, by justifying a part of the test vectors through the surrounding UDL. Later in [135], the same authors proposed to modify the output space of the UDL and completely eliminate the need for an isolation ring for certain cores. Instead of scanning core test vectors into an isolation ring, the core test vectors are justified by controlling the inputs of the UDL. Despite avoiding the high number of multiplexers, the main limitation of the solution presented in [135, 161] lies in its computational complexity. This is because prior to deciding which input/output ring elements need to be inserted or removed, an analysis needs to be performed

to check whether each test vector can be functionally justified. Therefore, the extensive usage of ATPG for this analysis reduces the the scalability of this methodology.

3.2.3 Functional Access

Functional access involves justifying and propagating test vectors through the available mission logic. Making use of the existing functional paths as test paths may significantly reduce the hardware cost. For a bus-based design, a large number of embedded cores are connected to the on-chip bus. Harrod [57] described a test strategy employed by ARM that enables test access through reusing the 32-bits functional bus. Test harness is introduced to isolate the AMBA-compliant core and communicate with the functional bus during test. In the test mode, the AMBA test interface controller becomes the AMBA bus master and is responsible for applying the test stimuli and capturing the test responses. The proposed approach is best suited for cores that are functionally tested and have a small number of non-bus I/Os. In addition, only one core is allowed to be tested at a time in this methodology, which may lead to long testing time.

Beenker et al. [7, 12] introduced a rather different test strategy, utilizing the module transparency (e.g., existing functional paths) for test data transfer. Although this *Macro Test* was developed originally to improve test quality by testing ”macros” with different circuit architectures (e.g., logic, memories, PLAs and register files) using different test strategies, later Marinissen and Lousberg [119] showed that the approach is also useful for testing core-based SOCs. The techniques described in Macro Test for introducing transparency, however, are rather ad hoc. Ghosh et al. [42] assumed that every core has a transparent mode in which data can be propagated from inputs to outputs in a fixed number of cycles. Their approach is based on the concept of finding functional test paths (F-paths) [37] through test control/data flow extraction. Although the proposed method successfully lowered the test area and delay overheads, it requires functional description of each core to make it transparent, which in most cases is not available or it is hard to extract. In addition, because test data cannot be pipelined through a core in this method, the potentially large transparency latency of each core (number of cycles needed to propagate test data from inputs to outputs of the core) may incur a relatively large test application time. To address the

above issues, in [41] the same authors extended their approach by describing a trade-off between the silicon area consumed by the design-for-transparency hardware and propagation latency. They suggested that the core providers offer a catalogue of area/latency versions for their cores, from which the system integrator can select one in order to meet the test access needs of its neighboring cores. By associating a user-defined cost function with the transparent test paths in the core connectivity graph, the system integrator is able to select the version of cores that achieve an optimized test solution at the system level. This approach, however, requires a large design effort at the core provider side. Another limitation of [41, 42] stems from the difficulty to handle other popular DFT schemes, such as scan or BIST, in the SOC.

Solutions from [41, 42] require that all the I/Os of a core to be simultaneously, though indirectly, controllable/observable. Ravi et al. [142] showed that this complete controllability and observability is unnecessary and proposed to provide them on an "as needed basis". Their approach allows for complex core transparency modes and is able to transfer test data to and from the CUT in a more aggressive and effective manner. For example, if the behavior of a core is such that input data at times t_i and t_j combine to generate output data at time t_k, [142] can achieve the transparency objective without additional DFT cost, which is unlike [41, 42] that have to resort to the use of test multiplexers. This is achieved through transparency analysis using non-deterministic finite-state automata and transparency enhancement via symbolic justification and propagation based on a regular expression framework. Since integrated system synthesis with testability consideration is able to output a more efficient architecture compared to performing test modifications post-synthesis, later in [141] Ravi and Jha combined the synthesis flow of MOCSYN [30] with the symbolic testability analysis in [142] so that testability costs are considered along with other design parameters during synthesis.

The ability to apply an arbitrary test sequence to core's inputs and observe its response sequence from the core's outputs consecutively at the speed of the system clock is important for non-scanned sequential circuits. It also facilitates an increase in the coverage of non-modeled and performance-related defects. Yoneda and Fujiwara [172] introduced the concept of consecutive transparency of cores, which guarantees consecutive propagation of arbitrary test stimuli/responses sequences from the core's inputs to the core's outputs

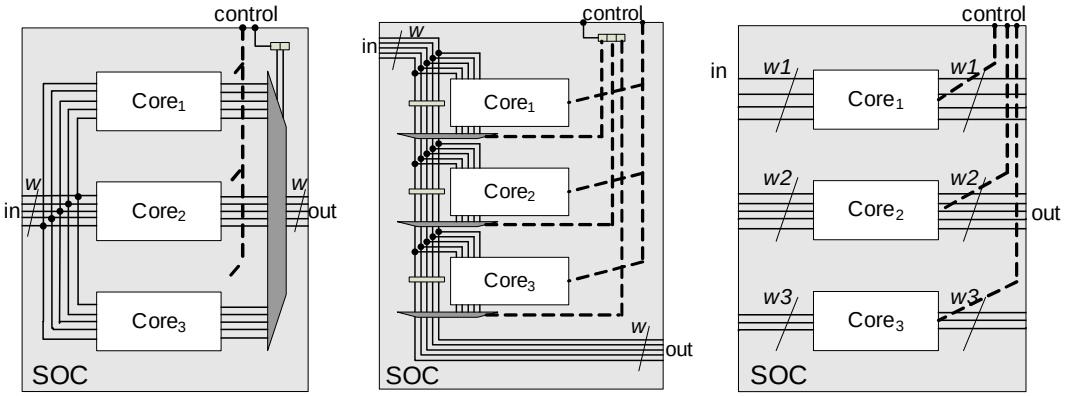
with a propagation latency. An earlier synthesis-for-transparency approach presented by Chakrabarty et al. [18] had tackled the same problem, i.e., it made cores single-cycle transparent by embedding multiplexers. When compared to [18], the area overhead for making cores consecutively transparent with some latency instead of single-cycle transparent is generally lower [172]. In order to further decrease the hardware cost, the same authors [173] proposed an integer linear programming (ILP) technique to make soft cores consecutively transparent with as few bypass multiplexers as possible. This is achieved by efficiently exploiting the data path description. Later, in [174], area and time co-optimization methodology for the TAM design problem was given using an ILP technique. In [131], Nourani and Papachristou presented a similar technique to core transparency, in which cores are equipped with a bypass mode using multiplexers and registers. They modeled the system as a directed weighted graph, in which the accessibility of the core input and output ports is solved as a shortest path problem. While this approach eases the problem of finding paths from the SOC inputs to the CUT, it requires packetization of test data (to match the bit width of input and output ports), and the help of serialization/de-serialization bit-matching circuits.

3.2.4 Dedicated Bus-Based Access

Since time-to-market is the overriding goal of core-based design, the ease with which cores can be designed and tested is crucial. As a result, the increased size of the logic and routing resources consumed by dedicated test infrastructure is acceptable for large SOC designs.

Aerts and Marinissen [2] described three basic scan-based test access architectures for core-based SOCs, as shown in Figure 3.4:

- **Multiplexing architecture:** All cores connect to all the TAM input pins and get assigned full TAM width. A multiplexer is added to select which core is actually connected to the TAM output pins, as shown in Figure 3.4(a). Only one core can be accessed at a time, and hence the total test application time is the sum of all the individual core test application times. Since core external testing needs to access two or more cores at the same time, it is hard to test circuits and interconnects between cores using this architecture.



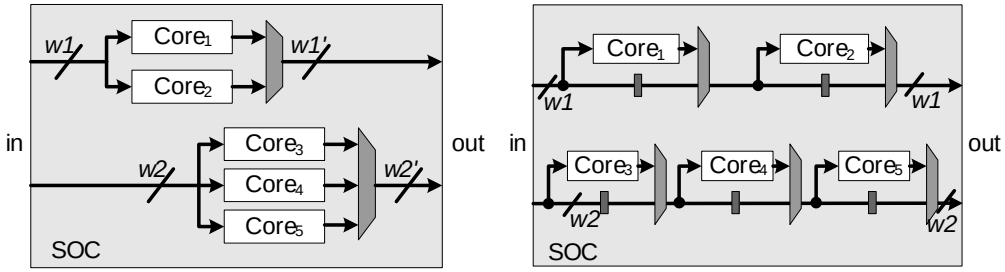
(a) Multiplexing architecture (b) Daisychain architecture (c) Distribution architecture

Figure 3.4: Three Basic Scan-Based SOC Test Architectures ([2]).

- **Daisychain architecture:** Long TAMs are constructed over all the cores from the TAM input pins to the TAM output pins. By-passes are introduced to shorten the access paths to individual cores, as shown in Figure 3.4(b). Cores can be tested either simultaneously or sequentially in this architecture, however, concurrent testing increases test power with almost no test application time reduction. Due to the bypass mechanism, external testing can be done efficiently using this architecture.

- **Distribution architecture:** The total TAM width is distributed to all the cores, and each core gets assigned its private TAM lines, as shown in Figure 3.4(c). Hence the total TAM width has to be at least as large as the number of cores. Cores are tested concurrently in this architecture, and the test application time of the SOC is the maximum of individual core test application times. In order to minimize the SOC test time, the width of an individual TAM should be proportional to the amount of test data that needs to be transported to and from a core connected to the TAM.

Based on the above basic architectures, two more popular test architectures, which support more flexible test schedules, were proposed. The *Test Bus* architecture proposed by Varma and Bhatia [162] can be seen as a combination of the Multiplexing and Distribution architectures, as shown in Figure 3.5(a). A single Test Bus is in essence the same as the Multiplexing architecture. Multiple Test Buses on an SOC operate independently (as in



(a) Test Bus architecture (b) TestRail architecture

Figure 3.5: Test Bus and TestRail Architectures.

the Distribution architecture) and hence allow for concurrent testing, however, cores connected to the same Test Bus can only be tested sequentially. Cores connect the Test Bus through a proposed test wrapper called *"test collar"* in order to facilitate test access and test isolation. However, the test collar does not support test width adaptation and external testing of its surrounding logic. It should be noted that the width of the input and output test busses can be different in the original Test Bus architecture [162], although most of the later approaches consider them equal.

Marinissen et al. [113] proposed *TestRail* architecture, which can be seen as a combination of the Daisychain and Distribution architectures, depicted in Figure 3.5(b). A single TestRail is in essence the same as the Daisychain architecture. Multiple TestRails on an SOC operate independently (as in the Distribution architecture). Cores on each TestRail can be tested sequentially as well as concurrently. A test wrapper called *TestShell* is proposed to connect each core to the TestRail. TestShell has four mandatory modes: function mode, IP test mode, interconnect test mode and bypass mode. A core specific test mode can be introduced, in addition to the four mandatory modes, if required. An advantage of the TestRail architecture over the Test Bus architecture is that it allows access to multiple or all wrappers simultaneously, which facilitates core external testing [48].

3.2.5 Cost Analysis for Different SOC Test Access Strategies

As discussed in Section 2.1.4, the cost of test (C_{test}) can be computed as $C_{test} = C_{prep} + C_{exec} + C_{silicon} + C_{quality}$, where C_{prep}, C_{exec}, $C_{silicon}$ and $C_{quality}$ are the nonrecurring test

Access Strategy	C_{prep}	C_{exec}	$C_{silicon}$	$C_{quality}$
Direct Access	Low	Medium	High	High
Isolation Ring Access	Medium	High	Medium	Medium
Transparency-based Access	High	Medium	Low	Low
Bus-based Access	Medium	Medium	Medium	Medium

Table 3.1: Test Cost Comparison for Different Access Strategies.

development cost, test execution cost, DFT hardware cost and imperfect test quality cost, respectively. To reduce C_{prep} in SOC testing, the test architecture must be optimized automatically in a short time, even for SOCs with a large number of cores. In addition, the translation of core-level tests to system-level tests should also be easily automated. $C_{silicon}$ is mainly dependent on the test access strategy and the amount of added test-related resources. To reduce $C_{quality}$, system integrator should try to decrease the performance overhead brought by DFT features and also balance fault coverage with yield loss. In particular, the system integrator must provide an efficient testing strategy for UDL and interconnects. To reduce C_{exec}, system integrators should avoid using high-end ATE and also minimize test data volume and test application time by efficient test architecture design and optimization and test scheduling techniques.

A generic comparison of the test cost for different test access strategies is shown in Table 3.1. Direct access scheme has the advantage of making core test translation unnecessary (low C_{prep}), however, it introduces large routing overhead (high $C_{silicon}$) and has difficulty with testing UDL and interconnects (high $C_{quality}$). The main disadvantage of ring access scheme is its long test application time (high C_{exec}). Although functional access through core transparency introduces the least DFT routing/area and performance overhead (low $C_{silicon}$ and $C_{quality}$), it is very difficult to design the system-level test access scheme (high C_{prep}). Modular test architecture using bus-based test access mechanism, being flexible and scalable, appears to be the most promising and cost-effective, especially for SOCs with a large number of cores. Its cost-effectiveness stems also from the fact that using design space exploration frameworks, the system integrator can trade-off different test cost parameter (e.g., C_{exec} against $C_{silicon}$). For bus-based TAMs, an acceptable $C_{quality}$ is guaranteed by providing test access to all the blocks in the SOC with the help of core test wrappers and the dedicated test busses. Within a satisfactory test development time (C_{prep}), the main

source of SOC test cost reduction lies in the reduction of C_{exec} and $C_{silicon}$. The two most important factors in C_{exec} are the volume of ATE test data and the test application time. Test data volume affects both the number of scan clock cycles and the test data management on the ATE. For example, if test data volume exceeds the ATE buffer size, buffer reload is necessary, which takes longer than test application and hence it dramatically increases the overall testing time. In addition, the test application time also depends on the test data bandwidth (the product of number of ATE channels and transfer rate), and more importantly, the TAM distribution to cores and the test schedule. Therefore, the reduction in C_{exec} can be achieved through efficient test architecture optimization and test scheduling techniques, which also affect the routing cost and the amount of DFT hardware, and hence $C_{silicon}$.

Most of the relevant research on SOC testing has been focused on dedicated test bus access (due to its modularity, scalability and flexibility in trading-off different cost factors) and it was concentrated in two main directions:

- *Test Scheduling and Test Architecture Optimization.* Given the number of TAM lines or test pins, efficient test architectures and test schedules are determined in a short development time, under constraints such as test power and physical layout. Numerous algorithms have been proposed on this topic and they are surveyed in Section 3.3.

- *Test Resource Partitioning.* By moving some test resources from ATE to the SOC, both the volume of test data that needs to be stored in the ATE buffers and the SOC testing time can be reduced. Since this technique is out of the focus of this dissertation, we just briefly overview the relevant work in this direction in Section 3.4.

3.3 Test Scheduling and Test Architecture Optimization

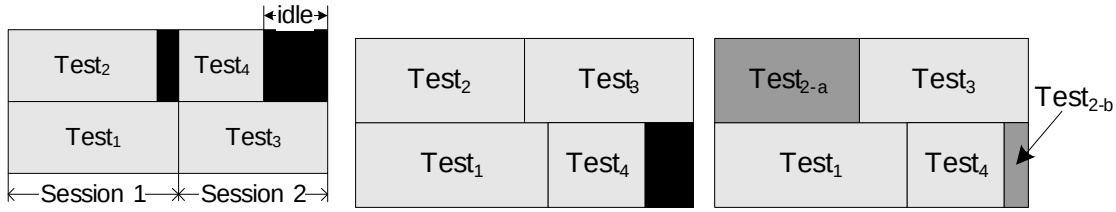
The optimization of modular test architectures (e.g., Test Bus and TestRail) and test scheduling have been subject to extensive research. For an SOC with specified parameters for its cores, a test architecture and a test schedule are designed to minimize the test cost, which

must account for test application time and the amount of DFT on-chip resources (both logic and routing).

Various constraints need to be considered during test scheduling, which is the process that determines the start and end test times for all the cores. For example, testing more cores in parallel usually can decrease TAT, however it will also increase the test power, which may lead to destructive testing [177]. In addition, *concurrency constraints* may exist due to sharing of test resources (e.g., a wrapped core cannot be tested at the same time with its surrounding unwrapped UDL because both of them use the wrapper boundary cells during test). Furthermore, in certain cases a user-defined partial ordering, called *precedence constraints*, is helpful in test scheduling. For instance, the core tests which are more likely to fail can be scheduled in front of the core tests that are less likely to fail. This helps to reduce the average test application time when "abort-at-first-fail" strategy is used [82]. Because test architecture optimization and test scheduling are strongly interrelated, next we formulate the integrated test architecture optimization and test scheduling problem (P_{taos}) as follows:

Problem P_{taos}: Given the number of available test pins N_p for the SOC, the peak test power constraint P_{peak}, the user-defined precedence constraint U_{prec}, the test set parameters for each core, including the number of input terminals i_c, the number of output terminals o_c, the number of test patterns p_c, the number of scan chains s_c and for each scan chain k the length of it $l_{c,k}$ (for cores with fixed-length internal scan chains) or the total number of scan flip-flops f_c (for cores with flexible-length internal scan chains), determine the wrapper design for each core, the TAM resources assigned to each core and a test schedule for the entire SOC such that: (i) the sum of the TAM width used at any time and the test control pin count does not exceed N_p; (ii) Power, concurrency and precedence constraints are met; and (iii) the SOC test cost $C_{test} = \alpha \cdot T + (1-\alpha) \cdot A$ is minimized, in which T is the SOC testing time, A is the DFT cost of the test architecture and α is a parameter specified by the system integrator in optimization.

Problem P_{taos} is strong NP-hard. This is because, when $\alpha = 1$ and there are no power, concurrency and precedence constraints, problem P_{taos} is reduced to problem P_{NPAW} in [73], which was proven to be NP-hard. Because of its complexity, only a part of P_{taos} was addressed in individual prior works. We start by reviewing various test scheduling techniques



(a) Session-based schedule (b) Sessionless schedule (c) Preemptive schedule

Figure 3.6: Test Scheduling Technique Categorization.

with fixed DFT hardware, then we continue with wrapper design methods and we conclude with the integrated wrapper/TAM co-optimization and test scheduling approaches.

3.3.1 Test Scheduling

Test scheduling is the process that allocates test resources (i.e., TAM lines) to cores at different time in order to minimize the overall test application time, while at the same time satisfying the given constraints. As shown in Figure 3.6, test scheduling techniques can be divided into [35, 100]:

- Session-based testing.
- Sessionless testing with run to completion.
- Preemptive testing, in which a core's test can be interrupted and resumed later.

Early session-based test scheduling techniques such as [25, 177] result in long test application time. This is because the division of the schedule into test sessions leads to large idle times, as shown in Figure 3.6(a). Hence, most of the proposed approaches for SOC testing fall into the other two sessionless test schemes. Preemptive testing (e.g., [70, 105]) can decrease test application time. It is useful in particular in constraint-driven test scheduling where there is more idle blocks in the schedule (due to additional constraints) and an entire core test is frequently not able to fit in these idle blocks. It should be noted, however, not all core test (e.g., memory BIST) can be preempted. In addition, the system integrator must consider the added overhead caused by the use of test preemption, i.e., the extra control and time to stop and resume the preemptive core tests.

Several techniques for SOC test scheduling, independent of TAM optimization, have been proposed in the literature. Sugihara et al. [156] addressed the problem of selecting a test set for each core from a set of test sets provided by the core vendors, and then schedule the test sets in order to minimize the test application time. The problem was formulated as a combinatorial optimization problem and solved using a heuristic method. It is assumed that each core has its own BIST logic and external test can only be carried out for one core at a time. Chakrabarty [15] generalized the problem and assumed the existence of a test bus. He formulated the problem as an m-processor open shop scheduling problem and solved it using a mixed-linear integer programming (MILP) technique. Marinissen [112] addressed test scheduling problem from the test protocol expansion point of view. A test protocol is defined at the terminals of a core and describes the necessary and sufficient conditions to test the core. He modeled the test scheduling problem as a No-Wait Job Shop scheduling problem and solved it using a heuristic.

Power-constrained test scheduling is essential in order to avoid destructive testing. Zorian [177] presented a power-constrained BIST scheduling process by introducing a distributed BIST controller. In [25], Chou et al. proposed a method based on approximate vertex cover of a resource-constrained test compatibility graph. Muresan et al. [124] modeled the power-constrained scheduling problem as job scheduling problem under the constraint some jobs must be executed exclusively. Then left-edge algorithm, list scheduling and force-directed scheduling are proposed to solve the problem. Larsson and Peng [104] tried to reorganize scan chains to trade-off scan time against power consumption. Ravikumar et al. [144] proposed a polynomial-time algorithm for finding an optimum power-constrained schedule which minimizes the test time. Later in [143], the authors considered SOCs composed of soft and/or firm cores, and presented an algorithm for simultaneous core selection and test scheduling. In both papers, they assumed that BIST is the only methodology for testing individual cores. Zhao and Upadhyaya [176] considered the test resources (e.g., test buses and BIST engines) as queues and the core tests to be scheduled as the job entering corresponding queue. The power-constrained test scheduling problem was then formulated as the single-pair shortest path problem by representing vertices as core tests, directed edges between vertices as a segment of a schedule sequence, and the edge weight as the core testing time at the end of the segment. An efficient heuristic was proposed to solve

this problem.

3.3.2 Wrapper Design and Optimization

It is important to note that IEEE Std. 1500 standardizes only the wrapper interface, as shown in Figure 3.2. Therefore the internal structure of the wrapper can be adapted to the specific SOC test requirements.

Core wrapper design mainly involves the construction of wrapper SCs . A wrapper SC usually comprises a number of wrapper boundary cells and/or core internal scan chains. The number of wrapper SCs is equal in number to the TAM width. Since the test application time of a core is dependent on the maximum wrapper SC length, the main objective in wrapper optimization is to build balanced wrapper SCs. For soft cores or firm cores, in which the internal scan chain design is not decided yet or can still be changed, balanced wrapper SCs are guaranteed to be constructed (lock-up latches may need to be inserted when wrapper SCs contain memory elements from multiple clock domains). For hard cores, in which the implementation of the internal scan chains is fixed, wrapper SCs are constructed by concatenating the set of core internal scan chains and wrapper boundary cells.

Marinissen et al. [114] first addressed the wrapper optimization problem of designing balanced wrapper SCs for hard cores. Two polynomial-time heuristics were proposed. The *LPT* (Largest Processing Time) heuristic, originally used for Multi-Processor Scheduling problem, was adapted to solve the wrapper optimization problem in a very short computational time. Since the number of internal scan chains and core I/Os are typically small, the computational complexity of LPT heuristic is quite small. The authors proposed another *COMBINE* heuristic which obtained better results by using LPT as a starting solution, followed by a linear search over the wrapper SC length with the First Fit Decrease heuristic. Iyengar et al. [73] proposed the *Design_wrapper* algorithm based on the Best Fit Decreasing heuristic for Bin Packing problem, which tries to minimize core test application time and required TAM width at the same time. They also showed an important feature of wrapper optimization for hard cores, i.e., the test application time varies with TAM width as a ”staircase” function. According to this feature, only a few TAM widths between 1 and

W_{max}, the maximum number of TAM width, are relevant when assigning TAM resources to hard cores and these discrete widths are called *Pareto-optimal* points.

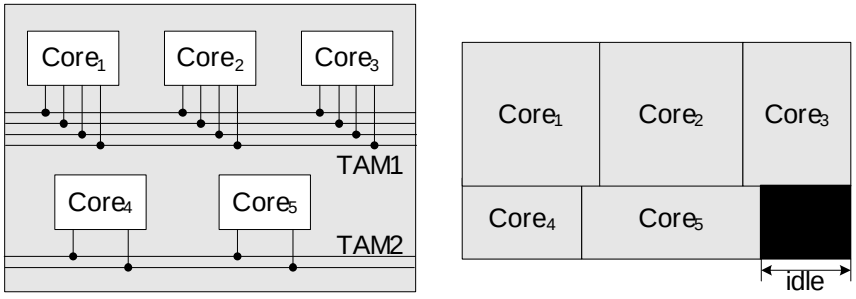
Koranne [92, 95] described a design of reconfigurable core wrapper which allows a dynamic change in the TAM width while executing the core test. This is achieved by placing extra multiplexers at the input and output of each reconfigurable scan chain. He also presented a procedure for the automatic derivation of these multiplexers using a graph representation of core wrappers. Instead of connecting the outputs of each scan chain to multiplexers, Larsson and Peng [105] presented a reconfigurable power-conscious core wrapper by connecting the inputs of each scan chain to multiplexers. This approach combines the reconfigurable wrapper [92, 95] with the scan chain clock gating [128] concepts. The reconfigurable core wrapper is useful for cores with multiple tests, where each of the tests has different TAM width requirements. For example, a core might have three tests: a BIST test for internal memory, a scan test for stuck-at faults, and a functional test for some un-modeled detects. The scan test needs a higher test bandwidth than the BIST test, while the requirement of the functional test is in the middle.

In [163], Vermaak and Kerkhoff presented a 1500-compatible wrapper design for delay fault testing, based on the digital oscillation test method. To be able to use this method, they introduced extra multiplexers and a cell address register to each wrapper cell. This approach for delay testing is only suitable for combinational cores, because only paths between the core's inputs and outputs are tested. In [44], Goel described a wrapper architecture for hierarchical cores, which allows for parallel testing of the parent and child cores, at the cost of an expensive wrapper cell design.

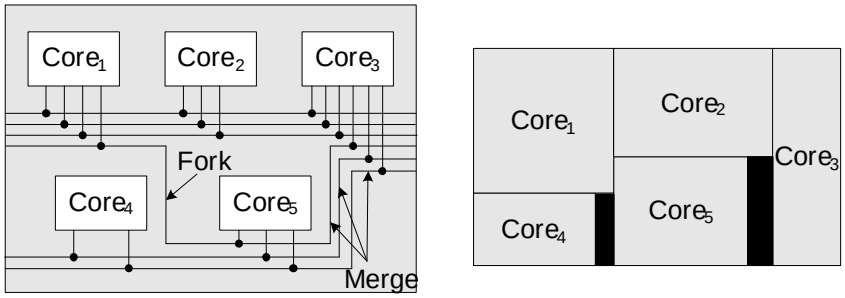
Since wrapper design, TAM optimization and test scheduling all have direct impacts on the SOC test cost, they should be considered in conjunction to achieve the best result. Consequently, the next section will review the state-of-the-art techniques for the integrated problem.

3.3.3 Integrated Wrapper/TAM Co-Optimization and Test Scheduling

The bus-based TAM architecture can be categorized into two types [76]:



(a) Fixed-width TAM bus



(b) Flexible-width TAM bus

Figure 3.7: TAM Bus Categorization.

- *Fixed-width Test Bus architecture*, in which the total TAM width is partitioned among several test buses with fixed-width, as shown in Figure 3.7(a). It operates at the granularity of TAM buses and each core in the SOC is assigned to exactly one of them.

- *Flexible-width Test Bus architecture*, in which TAM lines are allowed to fork and merge instead of just partitioning, as shown in Figure 3.7(b). It operates at the granularity of TAM lines and each core in the SOC can get assigned any TAM width as needed.

A number of approaches have been proposed for optimizing both architectures. Iyengar et al. [78] advocated flexible-width architecture to be more effective because it improves the TAM wire utilization. They argued that cores are frequently not assigned with a Pareto-optimal TAM width for fixed-width architecture, which wastes part of the test resources. Another reason is that, test scheduling is usually more tightly integrated with TAM design

in flexible-width architecture design, while in fixed-width architecture design it is often performed after TAM design by shifting tests back and forth to satisfy the given constraints. However, due to the complexity of the SOC test problem it cannot be generalized that the test cost of the flexible-width architecture is lower than the cost of the fixed-width architecture. This is because of the following reasons. Firstly, the Pareto-optimal TAM width only exists in hard cores for which internal scan chain design is fixed. For soft or firm cores, there is no waste of test resources when assigning them to any fixed-width TAM bus. Secondly, because of the strong NP-hard attribute of P_{taos}, the size of the solution space for this problem is enormous, even with the fixed-width TAM constraint. Hence, the effectiveness of an approach is to a large extent dependent on the effectiveness of the search algorithm embedded in the approach instead of the architecture. Thirdly, and most importantly, because cores on the same TAM bus can share the same scan enable signal in fixed-width architecture, the test control pin requirement is usually lower for the fixed-width architecture than for the flexible-width architecture. These test control pins will reduce the available TAM width for test data transfer [49], which will impact significantly the SOCs with a large number of cores.

Fixed-width Test Bus architecture Optimization

The main optimization techniques for fixed-width Test Bus architecture include integer linear programming (ILP), graph-based heuristics, and merge-and-distribution heuristics.

Iyengar et al. [73] first formulated the integrated wrapper/TAM co-optimization problem and broke it down into a progression of four incremental problems in order of increasing complexity. For the fixed-width architecture, core assignment to a TAM bus can be represented as a binary value, and the width of each TAM partition is an integer value between 1 and W_{max}, where W_{max} is the maximum TAM width. They can be treated as variables in an ILP model. The main drawback of the ILP method is that the computation time increases exponentially because of the intractability of the problem and hence the method is not suitable for SOCs with a large number of cores and/or TAM partitions. To decrease the CPU running time, the same authors proposed to combine efficient heuristics and ILP methods [74]. By pruning the solution space the computation time is significantly reduced. However, the proposed approach returned in longer test application time than ILP in most

cases. In [154], Sehgal et al. presented another optimization method based on Lagrange multipliers and achieved better results.

Instead of optimizing the finish time of the last core test, Koranne [94] proposed to optimize the average completion time of all the TAM partitions. Then he reduced this revised problem to the minimum weight perfect bipartite graph matching problem (not NP-hard) and proposed a polynomial time method to solve it. He also considered the TAM partitioning problem within the search space of his method, by projecting the ratio of the bitwidth[1] requirement for each core to the total bitwidth of the SOC (i.e., the sum of all cores' bitwidths). However, this projection method introduces the restriction that the overall TAM width of the SOC must exceed the total number of cores. By modeling the test sources/sinks as sources/sinks in a network, TAMs as channels with capacity proportional to their width, and each core under test as a node in the network, Koranne [93] formulated the test scheduling problem as a network transportation problem. The overall test application time of the SOC is thought of as the time taken to transport the test data between cores and the test source/sink, assuming the capture time is negligible as compared to the scan shifting time. A 2-approximation algorithm was proposed to solve this problem by using the results of single source unsplittable flow problem.

While the above approaches concentrate on Test Bus architecture, Goel and Marinissen [47] addressed the same problem for fixed-width TestRail architecture with the constraint that the total TAM width must exceed the number of embedded cores inside the SOC. This constraint is removed in [46] and a new efficient heuristic $TR-Architect$ is presented, which works for both cores having fixed-length and cores having flexible-length scan chains. Next, the same authors extended $TR-Architect$ in [48] to support both Test Bus and TestRail architectures. They also presented the lower bounds of SOC test application time in this work. Since $TR-Architect$ is adapted to solve the problems presented in Chapters 6 and Chapter 7, we briefly discuss its main idea here, for the sake of self-containment, please refer to [48, 51] for more details and terminology. $TR-Architect$ has four main steps. The basic idea is to divide the total TAM width over multiple cores based on their test data volume. The algorithm first creates an initial test architecture by assigning

[1]bitwidth is the minimum TAM width value beyond which there is no decrease in the test time for a given core.

value 1 to each core's TAM width. Since the overall test application time of the SOC T_{soc} equals the *bottleneck* TAM with the longest test application time, in the second and the third steps, the algorithm iteratively optimizes T_{soc} through merging TAMs and distributing freed TAM resources. Either two *non-bottleneck* TAMs are merged with less TAM width to release freed TAM resources to the bottleneck TAM, or the bottleneck TAMs is merged with another TAM to decrease T_{soc}. In the last step the algorithm tries to further minimize T_{soc} by placing one of the cores assigned to the bottleneck TAM to another TAM. In [50], Goel and Marinissen extended their $TR-Architet$ algorithm to minimize both testing time and wire length. For a given TAM division, based on a proposed simple yet effective wire length cost model, they described a heuristic to determine the ordering of the cores on each TAM bus so that the overall TAM wire length is minimized. By including the wire length cost into the optimization procedure of $TR-Architect$, a test architecture that can co-optimize time and routing overhead was obtained. Later in [49], the same authors discussed the influence of test control on test architecture optimization. Given the more practical test pin constraint N_p instead of total TAM width constraint W_{max}, the $TR-Architect$ is extended again to a control pin-constrained version. Finally, Krundel et al. [99] described a Test Architecture Specification language, in which the user can express various test constraints. They then modified $TR-Architect$ to incorporate the ability to satisfy these user constraints, by providing them as inputs to the tool.

Flexible-width Test Bus architecture Optimization

For flexible-width Test Bus architectures, several core test representations and their corresponding optimization techniques are summarized next.

Huang et al. [62] mapped the TAM architecture design to the well-known two-dimensional bin packing problem. Each core was modeled as a rectangle, in which its height corresponds to the test application time, its width corresponds to the TAM width assigned to the core, and its weight corresponds to the power consumption during test. The objective is to pack these rectangles into a bin of fixed width (SOC pins), such that the bin height (total test application time) is minimized, while not exceeding the power constraint. A heuristic method based on the Best Fit algorithm was then proposed to solve the problem. Later in [61], the authors described a new heuristic which gives better results. Another

advantage of this method is that, it supports multiple tests for each core. Iyengar et al. [76] described another heuristic *TAM_Schedule_Optimizer* for the rectangle packing problem without considering power constraints. Since this algorithm is adapted to solve the problems presented in Chapters 5 and Chapter 6, we briefly discuss its main idea here, for the sake of self-containment, please refer to [76, 78] for more details and terminology. *TAM_Schedule_Optimizer* first finds out the pareto-optimal TAM widths for all cores with fixed-length scan chains. Next, a "*preferred TAM width*" for each core is identified from these pareto-optimal TAM widths, such that the core's TAT is within a small percentage of its testing time at a maximum allowable TAM width. The test for each core is then scheduled using the preferred width, as long as there are enough TAM lines available. If the number of available TAM lines is insufficient to schedule any new tests, the resulting idle time is filled using several heuristics that insert tests to minimize the idle time. Whenever a currently-running test completes, the number of available TAM lines is incremented, and the algorithm repeats the scheduling process for the remaining tests. By taking pareto-optimal TAM width into account, *TAM_Schedule_Optimizer* is shown to be more effective than the algorithm presented in [62]. Next in [75], Iyengar et al. extended their algorithm to incorporate precedence and power constraints, while allowing a group of tests to be preemptable. They also discussed the relationship between TAM width and tester data volume in this work, so that the system integrator can identify an effective total TAM width for the SOC. Later in [79], the same authors considered to minimize the ATE buffer reloads and multi-site testing, again by extending their rectangle packing algorithm. Since idle time on a TAM wire between useful bits appears as idle bits on the corresponding ATE channel, they tried to move these idle time to the end of the TAM as suggested also in [53], so that these bits do not need to be stored in the ATE. Although the test application time might be increased for a single chip, the total test data volume is reduced and hence the average testing time can be decreased when multi-site testing is employed. In [181], Zou et al. used *sequence pairs* to represent the placement of the rectangles, borrowed from the place-and-route literature [123]. They then employed the simulated annealing optimization technique to find a satisfactory test schedule by altering an initial sequence pair and rectangle transformation. Their approach is shown to be more effective in terms of test application time than earlier methods, however, constraint-driven scheduling was not considered.

To include the peak power constraint into their optimization procedure, Huang et al. [65] proposed to model the core as a 3-D cube, where the TAM width, peak power and core testing time represent the three dimensions. They then formulated the integrated wrapper/TAM co-optimization and test scheduling problem under power constraints as a restricted 3-D bin-packing problem and proposed a heuristic to solve it. Following this idea, when additional constraints exist, the problem formulation can be extended as a bin-packing problem with appropriate number of dimensions.

Koranne and Iyengar [97] proposed a *p*-admissible representation of SOC test schedules based on the use of k-tuples. Compared with rectangle and 3-D cube representations, *p*-admissible representation has the benefits to be readily amenable to several optimization techniques, such as tabu-search, simulated annealing, two-exchange, and genetic algorithms. A greedy random search algorithm was presented to find an optimal test schedule.

By utilizing the reconfigurable wrapper concepts [92, 95], Koranne modeled the core test schedule as a malleable job rather than a static job [96]. A new polynomial time algorithm was then presented using a combination of a network flow algorithm [93] and a malleable job scheduling algorithm. In [101, 105], Larsson et al. showed the test scheduling problem is equivalent to independent job scheduling on identical machines, when reconfigurable wrapper and preemptive scheduling are used. Consequently, a linear time heuristic, which is able to produce a close-to lower bound solution, was proposed.

In [103], Larsson and Peng presented an integrated test framework by analyzing test scheduling under power and test resource constraints along with TAM design. They formulated the test architecture optimization problem as a set of equations and proposed a polynomial-time algorithm to solve them. Later in [106], a simulated annealing algorithm was proposed to reduce the CPU run time for large SOCs. The authors are one of the first researchers who considered almost all aspects of core-based SOC testing in an unified approach. However, the problem is over simplified by the assumption of linear dependence of testing time and power on scan chain subdivision.

Zhao and Upadhyaya [175] addressed the power-constrained test scheduling problem based on a graph-theoretic formulation. They constructed the power-constrained concurrent core tests and use the test compatibility information to settle the preferred TAM width for the tests. Dynamic test partitioning techniques are proposed to dynamically assign

the TAM width and reduce the explicit idle time. In [155], Su and Wu proposed another graph-based approach to solve the same problem. First the test compatibility graph (TCG) is built based on given test resource conflicts. TCG is a complete graph if no predetermined test resource conflicts exist. The authors proved that the problem can be reduced to finding a partial graph G of the TCG that satisfies: (1) G is an interval graph, (2) each maximum clique of G satisfies the power constraint, and (3) the overall test application time is minimized. To reduce the computation time of searching the solution space, a tabu search based heuristic is used for rapid exploration. In [64], instead of optimizing test schedule, Huang et al. proposed to use the test scheduling information as a constraint to optimize the total number of SOC test pins. The authors formulated this constraint-driven pin mapping problem as a chromatic number problem in graph theory and as a dependency matrix partitioning problem used in pseudo-exhaustive testing. A heuristic algorithm based on clique partitioning was proposed to solve the NP-hard problem. Larsson and Fujiwara [102] presented a test resource partitioning and optimization technique that produces a test resource specification, which can serve as an input to the test automation tool. The routing overhead was considered in the cost model, by using a single point to represent each core, and calculating the length of a TAM wire with a Manhattan distance function. The authors proposed a technique for the iterative improvement of a test specification by using Gantt charts. When the SOC contains a large number of cores and test resources, however, the search for an optimal solution will become computationally expensive.

3.4 SOC Test Resource Partitioning

As discussed in Section 2.1.3, BIST generates test stimuli and compares test responses on-chip. It is an alternative approach to ATE-based external testing, however, due to limited fault coverage of pseudo-random techniques, the use of external test resources is still required. This leads to the new test resource partitioning approaches [21], where on-chip embedded test resources interact with the external test equipment. Although the original work described in this dissertation does not rely on test data compression, the following summary of the main directions in the area is presented in order to provide a self-contained overview of low cost system-on-a-chip testing. Besides, combining the methods described

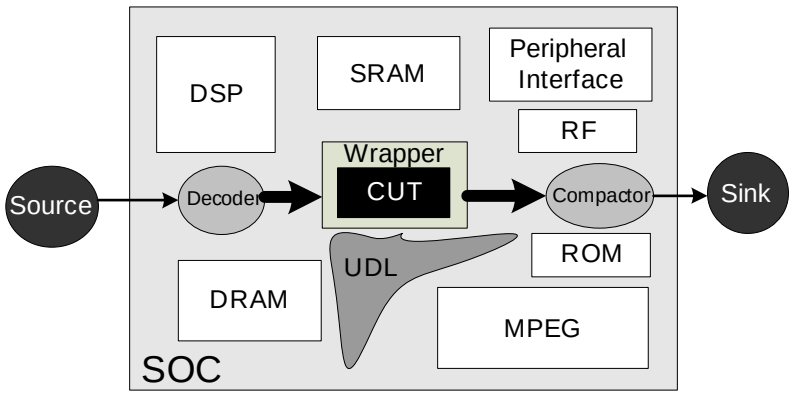


Figure 3.8: Conceptual Infrastructure for SOC Testing with Test Data Compression.

in this thesis with test data compression is an interesting topic for future work, as pointed out in the final chapter.

Full scan is the mainstream DFT technique employed by most of the chip designers and embedded core providers. For full-scanned embedded cores, the test cubes generated by ATPG tools feature a large number of unspecified (”don’t-care” or X) bits which can be assigned arbitrary values, even after static and/or dynamic compaction. Various test data compression (TDC) techniques significantly reduce the test data volume stored in ATE by exploiting these don’t care bits. TDC mainly involves two parts: test stimuli compression and test response compaction. A conceptual architecture is illustrated in Figure 3.8, in which the decoder is used to decompress the compressed test vectors sent from ATE and the compactor is used to compress the test responses. It should be noted that TDC only reduces the test data volume that needs to be transferred between ATE and the chip. It does not reduce the number of test patterns applied to the chip, which is different with traditional test set compaction techniques [1] and has the advantage of being able to detect un-modeled defects.

3.4.1 Test Stimuli Compression

LFSR-reseeding, was first proposed by Koenemann in [90]. Instead of storing the fully specified test patterns on ATE, he proposed to store the LFSR seeds, whose size is only slightly higher than the number of the maximum care bits in the test cubes. As a follow up

to this work, many approaches [6, 31, 91, 138] were proposed to improve the encoding efficiency and/or reduce the possibility of a phenomenon called ”test pattern lockout” caused by the linear dependencies in the LFSR sequences. The above techniques are embedded in the ATPG flow or exploit the core’s structural information which may not be suitable for the IP-protected core-based design flow.

An effective TDC approach suitable for SOC testing is based on data compression using coding techniques, which exploit the following features.

- The ”don’t care” bits in test cubes can be filled with arbitrary values to form long run of 0’s or 1’s.
- There is a lot of similarity between test vectors because of the structural relationship among faults in the circuit. By carefully ordering the test sets, successive test vectors will differ in only a small number of bits. Then, the information about how the vectors differ, instead of the original vectors, can be encoded to reduce test data volume.

These techniques compress test data without requiring the structural information from the core provider (which is necessary for integrated compression, ATPG and fault simulation) and therefore they fit well in the core-based design flow.

To achieve compressed SOC test set, in statistical coding-based TDC techniques, data is first decomposed into either fixed-length or variable-length blocks and a code word, also of either fixed or variable length, is assigned to each block. The basic idea is to assign frequent blocks to a comparably smaller code word. Various coding techniques [20, 19, 52, 81, 159] have been proposed in the literature. Several dictionary-based TDC techniques [89, 108, 166] have also been presented recently. While the statistical coding schemes use a statistical model of the test data and encode blocks according to their frequencies of occurrence, dictionary-based methods select strings of the blocks to establish a dictionary, and then encode them into equal-size tokens [149]. When selecting TDC strategies, system integrators need to trade-off the size of the decompression logic, test data compression ratio and test power consumption.

3.4.2 Test Response Compaction

Test response compaction is the process that reduces the volume of test responses sent from the CUT to the ATE. Unlike the lossless compression techniques used for test stimuli, test response compaction techniques are typically lossy, i.e., some faults that can be detected by the original test response may be masked by the compactor. This phenomenon is referred to as *aliasing*. An additional problem lies in the fact that many designs produce unknown states (also called X states) in test responses. Sources of these unknown states include bus contention, interactions between multiple clock domains or uninitialized memory elements. Although inserting test points helps to eliminate unknown states in test responses, this intrusive DFT technique is not preferable for core-based design. Further, the test response compactor should have also support for reliable diagnosis, in addition to very low aliasing and the ability to handle responses with unknown states.

Based on their structures, test response compactors can be categorized into three types [139]: infinite memory compactors (also called time compactors or sequential compactors), memoryless compactors (also called space compactors or combinational compactors) and finite memory compactors. Infinite memory compactors, such as MISRs or cellular automata, have been widely used in BIST environments [1]. By inserting an MISR between the scan chain outputs and bi-directional scan pins, Barnhart et al. [5] presented a method to compact scan data during the scan-out operation. A characteristic feature of the above compactor is the infinite impulse response property, which allows the compaction ratios to be a huge number from 10^6 to 10^8 with a very small aliasing possibility. If the test responses contain unknown states, these unknown states will need to be masked to avoid the corruption of the response signature. Memoryless compactors are combinational circuits. They compact a large number of scan outputs, m, to a much smaller number, k, through a compaction circuitry (usually XOR networks). Various linear and nonlinear compactors [122, 133] have been proposed in the literature. Memoryless compactors can handle unknown states in test responses without circuit modification. However, the compaction ratio is much smaller when compared with the infinite memory compactors. Finite memory compactors contain memory elements, however they have a finite impulse response. In [139], Rajski et al. first introduced this type of compactor: the convolutional compactor.

The finite memory compactors can achieve compression ratios in between that of memoryless and infinite memory compactors. They also can handle unknown states and support diagnosis, at the cost of a slightly higher DFT area.

3.5 Concluding Remarks

This chapter introduces the IEEE Std. 1500 and surveys prior work in the SOC test architecture design and optimization field. Although effective today, with the growing complexity of SOC designs and the aggressive shrinking feature size of semiconductor technologies, new methodologies are still required to be developed to address the emerging challenges. For example, care must be taken to avoid test data corruption for embedded cores containing multiple clock domains. This problem has not been taken into account in the IEEE Std. 1500 and prior research effort. Therefore, in the next chapter, we show how at-speed test application can be achieved without test hazards by embedding custom logic in a novel multi-frequency core wrapper.

Chapter 4

Wrapper Design for Multi-Frequency IP Cores

This chapter addresses the testability problems raised by IP cores with multiple clock domains. The key factor of the proposed solution is a novel core wrapper architecture used to synchronize the external tester channels with the core's internal scan chains in the shift mode, and provide at-speed test control in the capture mode. Section 4.1 gives preliminaries for the research work presented in this chapter, outlines the motivation behind it and summarizes its contributions. The detailed multi-frequency wrapper architecture is presented in Section 4.2. Next in Section 4.3, we discuss how to optimize the proposed wrapper in terms of testing time under power constraints, using integer linear programming (ILP) model and fast heuristics. Experiments on a hypothetical yet representative multi-frequency core are conducted in Section 4.4. Finally, Section 4.5 concludes this chapter.

4.1 Preliminaries and Summary of Contributions

With the ever increasing design complexity, many embedded cores operate internally using multiple frequencies. For instance, in a video processing SOC design [164], all the embedded cores have at least three clock domains and one of them has 12 clock domains inside.

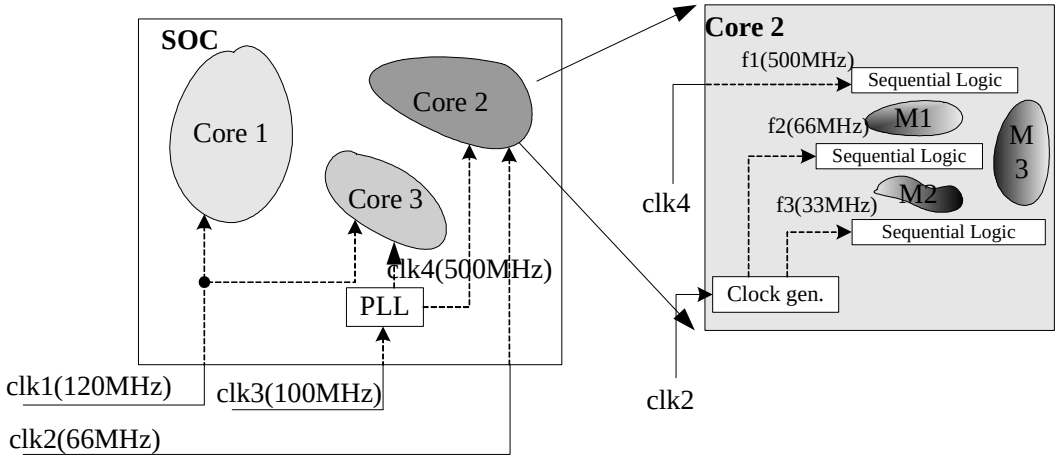


Figure 4.1: An Example Multi-Frequency SOC.

To illustrate a multiple-frequency core-based SOC, Figure 4.1 shows a simple hypothetical design that comprises three cores with three different physical clocks. In addition, Core 2 consists of modules (M1,M2,M3) operating at different frequencies (f1,f2,f3). A *physical clock* is a chip-level clock, e.g., it can come from an external oscillator or an on-chip phase-locked loop (PLL). All the internal clocks generated from the same physical clock are considered to be a part of the same physical clock domain. The multi-frequency modules communicate with each other through asynchronous handshake signals, synchronization logic or FIFO memory blocks [27]. Although multi-frequency embedded cores present advantages, such as reduced power consumption and silicon area, because of the clock skew and synchronization problems they require special attention during test, as discussed in Section 2.2.

A few hardware approaches based on BIST have been proposed to address at-speed multi-frequency testing problem. The solution presented in [126] used a dedicated clock generator to shift the multi-frequency scan chains at their corresponding clock frequencies. To avoid clock skew during capture, retiming latches or dedicated two-phase/two-edge clocking schemes were used between different clock domains. In [125] the test data was shifted in/out at-speed by reusing the existing clock tree on chip and a programmable *scan mode* signal unit was used to control the capture of the circuit responses. The solutions proposed in [11, 59] employ a rather different approach that separates the clocking for scan

and capture in two phases, by multiplexing the clock signals for each phase. The main difference between these two approaches lies in the design of the *capture window*. In [11], in the capture mode, all the rated clocks are applied iteratively in a number of intervals equal to the number of clock domains. In each iteration the selected clock signal will propagate to a scan chain only if its value is lower than the rated clock of the respective scan chain. In [59], a more flexible capture window was used, which consists of captures in different clock domains and some shift operations to create inter-domain at-speed capture. The functional clock of each of the domains is used to obtain a shift followed by a capture. In addition, the shift operation for all the scan chains can work at any of the on-chip frequencies. Four types of elements, composed of a scan flip-flop and an extra latch or flip-flop, are introduced in between transition-hazard clock domains[1] in [150]. The extra latch or flip-flop is controlled by an external test signal and hence a two-phase clocking scheme is applied in the test mode. The technique can be effectively applied in low frequency scan test. However, since the relation between the two clocks is dependent on the maximum clock skew and the longest propagation delay, it is difficult to ensure a non-overlapping clocking scheme for at-speed test.

Despite the fact that BIST is the primary solution for at-speed multi-frequency testing, there are a number of issues which arise in the SOC paradigm from the core provider and system integrator inter-operability perspective. There are four main cases: the system integrator will receive a (*i*) BIST-ed core, a (*ii*) BIST-ready core, a (*iii*) scan-testable core, or a (iv) functional-testable core. With the exception of the BIST-ed multi-frequency cores, in order to deliver test patterns (using *low-speed testers*) to the IP-protected cores and to perform rapid at-speed test (using *high-speed on-chip generated clocks*) without exceeding their power ratings, the system integrator is constrained to design a multi-frequency core wrapper (MFCW). Unlike the existing approaches which assume that designs/cores can be BIST-ed for multi-frequency test, i.e., the structural netlist can be modified with extra hardware to guarantee valid multi-frequency capture, our approach is suitable for IP-protected cores where the SOC integrator is in charge of developing the core test strategy.

The system integrator can use test wrapper design to tackle the multi-frequency test

[1] If there are no data transfers between two clock domains or data transfers between two clock domains are safe during capture, then they are called *transition-free*. Otherwise, they are called *transition-hazard* clock domains.

problem, however the existing single-frequency core wrapper (SFCW) designs (e.g., [75, 92, 105, 113, 114]) are not directly adaptable to at-speed multi-frequency testing. This is because, as discussed in 2.2, if memory elements from different clock domains, such as core's internal latches/flip-flops and wrapper boundary registers, are captured at the same time, then clock skew may occur and corrupt test data. While in the functional mode, signals crossing different clock domains are made skew-tolerant through special handshake hardware (e.g., brute-force synchronizers or FIFOs) or protocols, avoiding clock skew during test is a major challenge. Although by grouping the flip-flops triggered by the same clock together and adding lock-up latches in between different clock domains, clock skew problem during shift can be solved (for mux-based scan approach), clock skew during at-speed capture still might occur and corrupt the test responses. Therefore, careful attention must be paid to the design of the capture window and a special support must be provided within the core wrapper architecture in order to guarantee that IP protection of embedded cores is not violated. In addition to the capture window design, the frequency at which the test data is fed to the core wrapper scan chains is of great importance, since it determines the trade-off between test application time and test power dissipation. On the one hand, if no attention is paid to the multi-frequency core wrapper design and the test data is fed into scan chains at high shift frequency then the outcome of the testing process may lead to the permanent damage of the SOC due to the excessive test power. On the other hand, if the test data is fed into the scan chains at low shift frequency then large test application time may be required, thus leading to an increased test execution cost. New features for scaling the shift frequency should be provided *within the core wrapper architecture* in order to efficiently utilize the available tester bandwidth while meeting the constraints on the maximum internal shift frequency that guarantees low testing time within the given power ratings.

In this chapter, we present how to solve the above emerging testing problem for the non-BISTed IP-protected multi-frequency embedded cores (i.e., determining reliable and cost-efficient shift frequencies and designing at-speed multi-frequency capture windows without any structural modifications to the core). The two main contributions of this chapter, detailed in Sections 4.2 and 4.3, are as follows:

- we propose a novel wrapper architecture for embedded cores containing multiple

clock domains. The proposed MFCW architecture, using limited hardware overhead, can trade-off the number of ATE channels, scan time and test power and, at the same time, it implements a capture window for at-speed stuck-at fault testing that can efficiently work with cores for which ATPG techniques described in [80, 110] have been applied. It is also important to note that, since clock skew problem during capture is avoided by making transition-hazard clock domains capture at different time, the proposed approach is not affected by different synchronizer designs [27];

- we also present ILP model and efficient heuristics to optimize the proposed MFCW architecture in terms of test application time under average power constraints. System integrators can easily trade-off the testing time, routing overhead and test power consumption based on the proposed wrapper optimization techniques, when selecting their test strategies.

4.2 Multi-Frequency Core Wrapper Design

This section presents the multi-frequency core wrapper architecture, by focusing on the case when all the internal clocks are generated from a single physical clock. As summarized at the end of this section, the proposed solution can easily be extended to the general case when a core has multiple physical clocks.

To perform at-speed testing, although it is not necessary to load/unload data at functional frequencies, the launch and capture must be done at-speed [59]. To facilitate this, the physical test clock needs to function at the highest frequency f_h during launch/capture. Since most of the available testers cannot provide test data at the highest on-chip frequency, we need to provide a mechanism which can use low-speed testers to transfer test data (shift phase), yet perform test application using high-speed on-chip functional frequencies (capture phase). To achieve this, the tester must synchronize with an on-chip low frequency f_l, derived from the on-chip high functional frequency f_h (e.g., coming from PLL), which is used to shift in/out test data from/to the ATE. In the following the details of the proposed wrapper architecture is given.

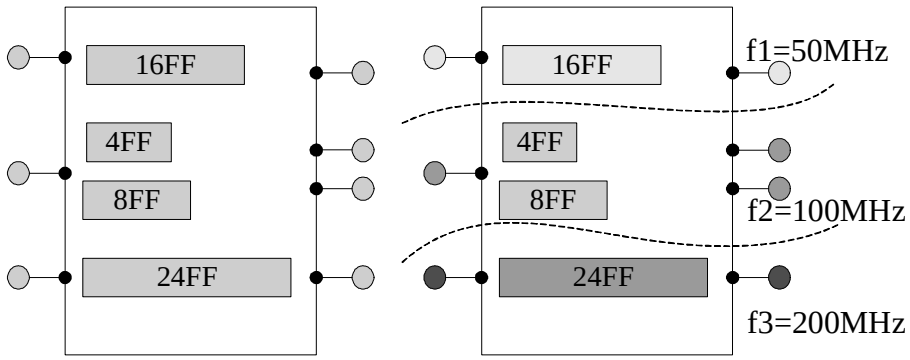


Figure 4.2: Single/Multiple Frequency Cores.

Wrapper Scan Chains in Multiple Frequency Groups: An example multiple frequency core is shown in Figure 4.2, where in addition to labeling the scan chains with their lengths, the associated functional frequencies are also provided. As introduced in Section 3.3.2, core wrapper design is mainly concerned with the construction of wrapper SCs such that the testing time is minimized. The wrapper SCs are composed from the wrapper input/output cells and the internal scan chains. When using SFCW, the testing time (in seconds) is a function of the longest wrapper SCs [114] and the single shift frequency f_{shift}, given by the following equation:

$$\tau(C_s) = \{(1+\max(wsc_{in}, wsc_{out})) \times N_v + \min(wsc_{in}, wsc_{out})\} / f_{shift} \quad (4.1)$$

where wsc_{in}/wsc_{out} are the lengths of the maximum input/output wrapper SC, respectively, and N_v is the number of test vectors in the test set for single-frequency core C_s. For multi-frequency core C_m, care is taken not to combine scan chains belonging to different frequency groups, and consequently the testing time of C_m is not only dependent on the lengths of the wrapper SCs but also on the shift frequency f_{shift_i} for different clock domain i (note, f_{shift_i} is not necessarily the same as its functional frequency). For multi-frequency core test, since cores with different clock domain configurations will have different capture window design, hence different lengths of capture cycles, the testing time of the multi-frequency core can be formulated by the following equation:

$$\tau(C_m) = \max_i \{\max(wsc_{in\_i}, wsc_{out\_i}) \times N_v + \min(wsc_{in\_i}, wsc_{out\_i})\} / f_{shift\_i} + t_c \times N_v \quad (4.2)$$

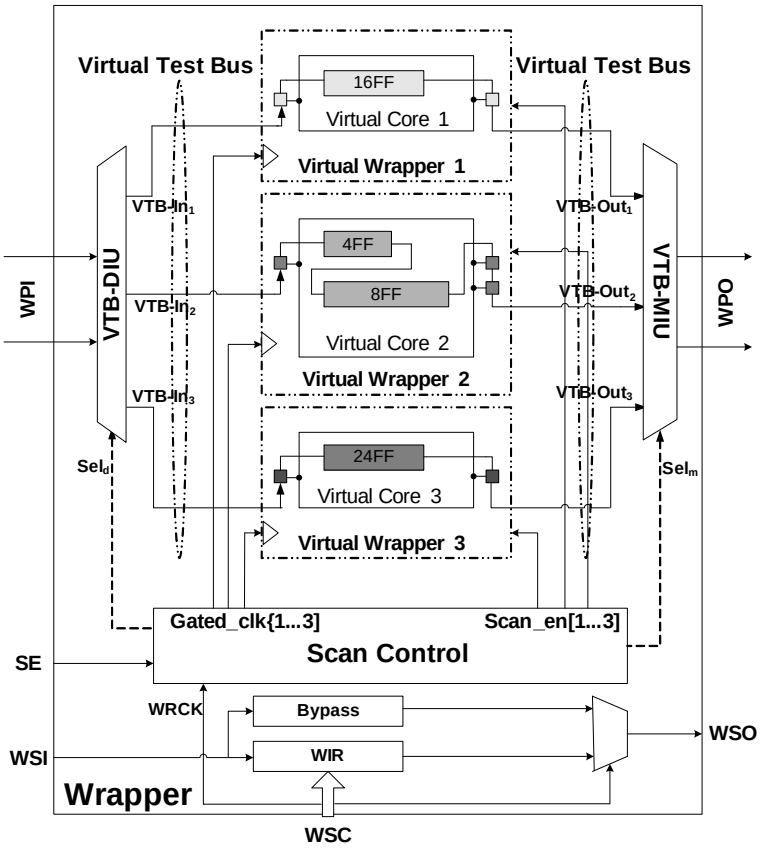


Figure 4.3: An Example Multi-Frequency Core Wrapper.

where the wsc_{in_i}/wsc_{out_i} are the lengths of the maximum input/output wrapper SC of clock domain i (with shift frequency f_{shift_i}) from core C_m, t_c is the time spent on capture phase for each pattern and N_v is the number of test vectors. To compute the wrapper SCs for the multi-frequency core, a single-frequency wrapper design algorithm can be employed and adapted for each frequency group, as shown later in Section 4.3.

Core Wrapper Interface: A multi-frequency core wrapper for the example core shown in Figure 4.2(b) is depicted in Figure 4.3 (InTest mode is illustrated). When compared to a IEEE 1500 single-frequency core wrapper, the proposed multi-frequency core wrapper has the same interface signals: serial input/output (WSI/WSO), parallel input/output (WPI/WPO), and the wrapper serial control (WSC) port, and hence is compliant with the IEEE Std. 1500. Additional off-chip test clocks (i.e., ATE supplied) or on-chip generated

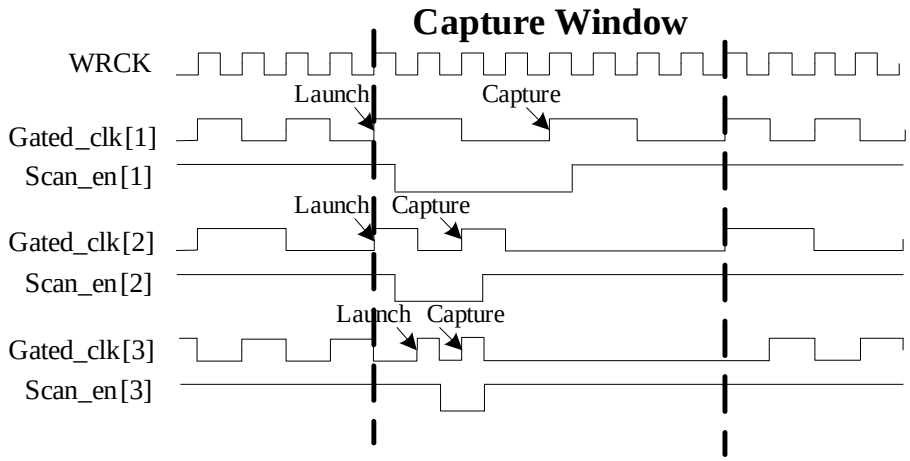


Figure 4.4: At-Speed Multi-Frequency Testing Timing Diagram with a Single Physical Clock.

test clocks can also be included in the multi-frequency interface.

Core Wrapper Architecture: Although the MFCW interface is the same as the SFCW one, the internal structure of the proposed MFCW differs significantly from the SFCW. Logic blocks belonging to different frequency domains are grouped and marked in the figure as *virtual cores*, and for each virtual core (VC) a *virtual wrapper* (single-frequency core wrapper), containing the wrapper SCs for the respective group, is assigned. The virtual wrapper is connected to the interface through virtual test bus (VTB) lines. We assume that the system integrator uses parallel TAM lines (WPI/WPO) for testing multi-frequency cores (this is optional for SFCW design). WPI is connected to a *virtual test bus de-multiplexing interface unit (VTB-DIU)*, which drives data in the *virtual test bus lines*. Similarly, WPO is connected to a *virtual test bus multiplexing interface unit (VTB-MIU)*, which collects the data from the virtual test buses. The motivation behind using VTB-DIU and VTB-MIU is to let the MFCW be able to work with any arbitrary TAM width. The operation of VTB-DIU and VTB-MIU will be explained later in this section.

Scan Control Block: Scan Control block is a key part of the proposed MFCW, since it is used to generate the gated clocks (*Gated_clk*) necessary for shift and capture phases and scan enable signals (*Scan_en*) required by each virtual core in the capture phase. Since for at-speed testing, it is not necessary to load/unload test data at the rated frequencies, the

shift frequency is used to trade-off testing time against the average test power dissipation during scan shifting. Unlike [58], we do not speed up the test data loading/unloading to its functional frequency through serializing/de-serializing technique since this will reduce the tester channel capacity and increase test power. Rather, we load/unload test data from/to the ATE at a slower frequency f_t and distribute it to different virtual core *i* at distinct shift frequency f_{shift_i} using the proposed wrapper architecture, which is switched to the functional frequency in the capture window (see Figure 4.4). In addition, by making the distinct shift clocks have a different clock phase, the peak power consumption during scan shift phase can also be decreased. For the example core from Figure 4.2(b), if we assume the maximum tester frequency is 120MHz, we will synchronize the ATE with the on-chip clock with the maximum functional frequency ($f_3 = 200MHz$). To simplify the hardware implementation we select the ratio of $\frac{f_t}{f_{shift\_i}}$ as two's exponent. Therefore, the ATE operational frequency is selected as $f_t = 100MHz$, assuming core test power using f_t is within its power rating. It is also important to note that, due to the nature of launch from last shift at-speed testing strategy used in this chapter, scan enable signal must be able to switch at-speed (see signal $Scan\_en[3]$ in Figure 4.4), which requires either routing this signal as a clock tree or pipelining it to distribute the delay across several clock cycles [134]. However, the proposed methodology can be easily extended to support launch from capture (i.e., broadside testing) when two-pattern test is used to cover delay faults. In addition, since transition-hazard VCs are captured at different time in the capture window, the previous-captured VC will pass data to the later-captured VC and overwrite part of its shifted data. We assume in this chapter that advanced ATPG tools, as described in [80, 110], are used to take care of such situations.

By grouping flip-flops from the same clock domain into separate scan chains we eliminate the problem of clock skew during shift. However, to avoid the clock skew problem during capture, we employ a capture window. In the capture window separate capture clocks are generated for each VC. When compared to [59], to adapt capture window to the core provider/system integrator model, the proposed solution is not programmable. Rather, its control is *embedded in the core wrapper architecture*. The clock switching between shift clock and capture clock is made glitch-free in the Mux using techniques described

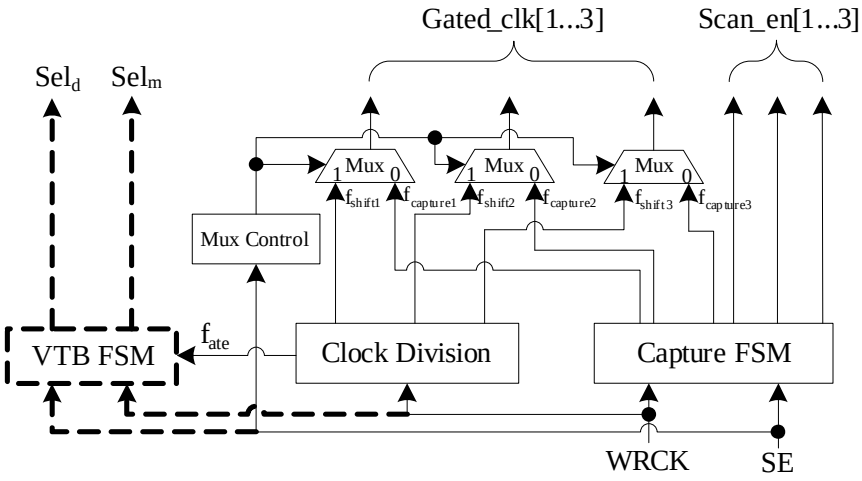


Figure 4.5: Block Diagram of Scan Control Block.

in [137]. In addition, based on core provider's information, the transition-free clock domains are captured at the same time (this decreases test generation complexity), while the transition-hazard clock domains are captured at different times to avoid the inter-domain clock skew. This is achieved through carefully controlling the *Gated_clk* and *Scan_en* signals using the *Capture FSM* shown in Figure 4.5. The generation of these two signals justifies the use of the highest frequency physical clock instead of the slow tester clock as the core wrapper clock signal *WRCK*. If the shift frequencies f_{shift} is lower than the tester frequency f_t then an internal control finite state machine (*VTB FSM*) is used to generate the mux select signal for VTB-DIU and VTB-MIU (see Figure 4.5). The testing timing diagram for the example core from Figure 4.2(b) is shown in Figure 4.4. It should be noted that clock domains 2 and 3 are assumed to be transition-free and hence they can be safely captured simultaneously, while the clock domain 1 will capture data at a different time to eliminate the test invalidation problem arising from clock skew during capture.

VTB-DIU and VTB-MIU Blocks: VTB-DIU block is used to synchronize the input test data and to transfer the test vectors into the corresponding virtual cores. If shift frequencies f_{shift} for all VCs are selected to be the same as the ATE frequency f_t, then we can simply connect each TAM line to its corresponding VTB line through a flip-flop (used to register the last shift launch bit). However, if lower shift frequencies is used, then VTB

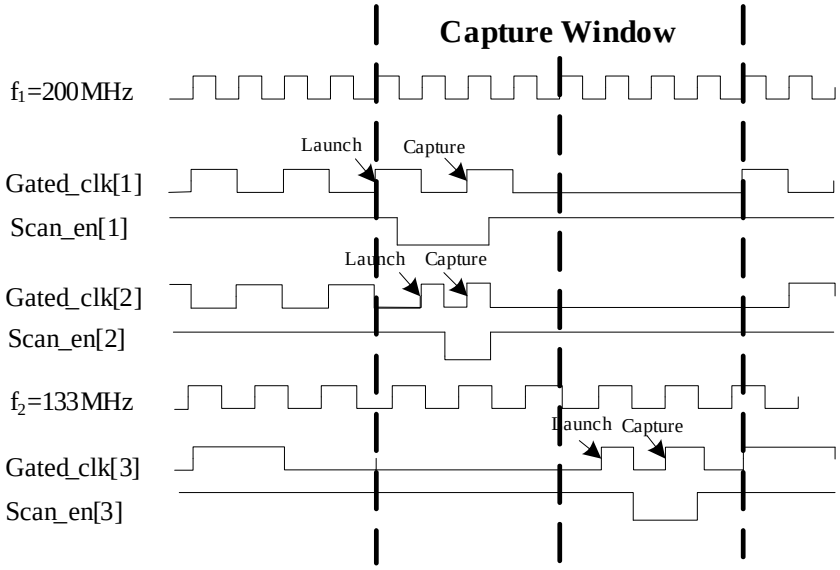


Figure 4.6: Timing Diagram for At-speed Multi-Frequency Testing with Multiple Physical Clocks.

FSM is needed to control the de-multiplexing unit. Again, flip-flops are used in the block to register the last shift launch bits. It is obvious that by changing the Sel_d at f_t, the test data from the TAM lines are loaded into the corresponding internal flip-flops in an interleaved mode first and then into all VTB lines at f_{shift} with a latency of one clock cycle. It is important to note that the last shift launch bit registered in VTB-DIU block is shifted in at the correct time decided by the Scan Control block, which is an essential feature required by at-speed test through last shift launch. The multiplexing unit is the opposite of the de-multiplexing unit, i.e., it is used to synchronize the output test data and transfer the test responses to the corresponding TAM lines. Note that, both VTB-DIU and VTB-MIU are active only in the test mode and hence do not infer any additional performance penalty when compared to the standard IEEE 1500 wrapper.

MFCW With Multiple Physical Clocks: So far it was shown how a novel core wrapper architecture can address at-speed multi-frequency core test with only one physical clock domain. For the general problem, where the core comprises several physical clock domains, we still divide the core under test into VCs belonging to different clock domains

and we can still use distinct shift frequencies for different VCs. The main difference, however, lies in the design of the capture window. To reach at-speed testing without corrupting test data, we propose to separate the capture window into several separate sub-capture windows, corresponding to each physical clock domain. This is achieved by letting all the physical clocks connect to the Scan Control block and utilizing counters to control the interval between different sub-capture windows. By separating the launch/capture for transition-hazard physical clock domains in the capture window, we guarantee a risk-free at-speed test for each virtual core. For example, consider an embedded core with 3 virtual cores VC_1, VC_2, VC_3, that operate at 100MHz, 200MHz and 133Mhz separately. Suppose $f_t = 100MHz$ (division of $f_{VC_2} = f_1$ = 200 MHz), then a possible timing diagram is shown in Figure 4.6, which shows two separate sub-capture windows: one for clock domains 1 and 2, which are transition-free, and one for clock domain 3.

4.3 Multi-Frequency Core Wrapper Optimization

Since the proposed MFCW design enables the scan chains for different clock domains to shift data at distinct frequencies, thereby saving TAT under tight power constraints, we propose a new wrapper optimization procedure to minimize TAT. The problem can be stated as follows:

Problem $P_{mfw-opt}$: Given the test set parameters for the multi-frequency core, including

- the number of clock domains N_C;
- for each clock domain (virtual core) i, the number of primary inputs, primary outputs, and bidirectional I/Os, the number of scan chains and scan chain lengths for fixed-length scan chains (or the number of scan cells when scan chains are flexible); the number of test patterns and the average power consumption P_i;
- the maximum allowed average test power P_{ave};
- the ATE shift frequency f_t;
- the external TAM width W_{ext};

determine the wrapper design for the core, including

- the shift frequency f_{shift_i} for each clock domain i, $1 \leq i \leq N_C$;
- the number of virtual test bus lines W_i for each clock domain i, $1 \leq i \leq N_C$;
- the wrapper scan chain design

such that the TAT of the core is minimized, the average test power during scan shifting at any time does not exceed P_{ave}, and the internal scan bandwidth matches the external scan bandwidth.

In this section, we first develop an ILP model and solve it using a public software *lp_solve*[2]. Due to the computational cost of the ILP method, we also introduce an efficient heuristic to solve the $P_{mfw-opt}$ problem. Despite its computational complexity, the ILP model is useful to generate optimal solutions for small problem instances, and evaluate the effectiveness of the heuristic method by comparing these exact solutions to the heuristic solutions. The computation time can be reduced by LP-relaxation, whereby some carefully chosen integer variables are allowed to take non-integer values. This results in useful lower bounds on the testing time, which are presented in Section 4.4.

4.3.1 Wrapper Optimization Using an ILP Model

Suppose the possible shift frequencies for each VC are $f_{shift\_i} \in \{F_1, F_2, ..., F_M\}$, which satisfy (i) $F_{k+1} = \frac{F_k}{2}, k \in \{1, 2, ..., M-1\}$ (the "divided by a power of 2" relationship guarantees easy hardware implementation); and (ii) $F_1 \times 1 + F_M \times (N_c - 1) \leq f_t \times W_{ext}$, i.e., the external scan bandwidth exceeds the internal bandwidth when the number of VTB lines for every VC is 1 and one clock domain shifts at F_1, while all the other clock domains shift at F_M. Hence, when the number of trial frequencies M is given (we assume $M = 4$ in this chapter), the values of $F_1, ..., F_M$ can be pre-determined based on the above constraints.

Let W_i denote the number of virtual test bus lines assigned to clock domain i. Now the maximum possible value of W_i will be $W_{max} = \frac{f_t}{f_M} \times W_{ext} - N_c + 1$. We are able to pre-calculate $T_i(F_k, j)$, which is the test application time for clock domain i, when W_i is equal to j and f_{shift_i} is equal to F_k. We consider that the given value of P_i is the power consumption for domain i when shifted at F_M. Let us define the binary variable δ_{ij} as $\delta_{ij} = 1$ only if $W_i = j$, where $j \in \{1, 2, ..., W_{max}\}$. In addition, let us define the binary variable θ_{ik} as

[2]In our experiments we use *lp_solve* from http://elib.zib.de [151].

$\theta_{ik} = 1$ only if domain i is given a shift frequency F_k, where $k \in \{1, 2, ..., M\}$. Then the TAT of the core is:

$$T_{core} = max_i\{\Sigma_{j=1}^{W_{max}}\Sigma_{k=1}^{M}\delta_{ij}\theta_{ik}T_i(F_k, j)\} \tag{4.3}$$

The following constraints must be satisfied:

1. $\Sigma_{j=1}^{W_{max}}\delta_{ij} = 1$, $1 \leq i \leq N_c$, i.e., every virtual core is assigned to exactly one virtual test bus.
2. $\Sigma_{k=1}^{M}\theta_{ik} = 1$, $1 \leq i \leq N_c$, i.e., every virtual core is shifted in exactly one frequency.
3. $\sum_{i=1}^{N_c}\Sigma_{k=1}^{M}\theta_{ik} \times P_i \times \frac{F_k}{F_M} \leq P_{ave}$, i.e., the power rating is not exceeded.
4. $\sum_{i=1}^{N_c} W_i \times f_{shift\_i} \leq W_{ext} \times f_t$, i.e., the external scan bandwidth is not exceeded.

Since we have

$$W_i = \sum_{j=1}^{W_{max}} \delta_{ij} \times j \tag{4.4}$$

$$f_{si} = \sum_{k=1}^{M} \theta_{ik}F_k = \sum_{k=1}^{M} \theta_{ik}2^{M-k}F_M \tag{4.5}$$

constraint 4 can be converted to:

$$\sum_{i=1}^{N_c}\sum_{j=1}^{W_{max}}\sum_{k=1}^{M} 2^{M-k}\delta_{ij}\theta_{ik}j \leq W_{ext} \times (\frac{f_t}{F_M}) \tag{4.6}$$

The non-linear term $\delta_{ij}\theta_{ik}$, must be linearized so that we can use the linear programming tools to solve this problem. This is done by introducing a new binary variable $\lambda_{ijk} = \delta_{ij}\theta_{ik}$ with additional constraints, which yields the following ILP model:

Objective: Minimize T_{core}, subject to the following constraints:

1. $\Sigma_{j=1}^{W_{max}}\Sigma_{k=1}^{M}\lambda_{ijk}T_i(F_k, j) \leq T_{core}, \ 1 \leq i \leq N_c$
2. $\Sigma_{j=1}^{W_i^{max}}\delta_{ij} = 1, \ 1 \leq i \leq N_c$
3. $\Sigma_{k=1}^{M}\theta_{ik} = 1, \ 1 \leq i \leq N_c$

4. $\sum_{i=1}^{N_c}\sum_{k=1}^{M}2^{M-k}\theta_{ik}P_i \leq P_{ave}$

5. $\sum_{i=1}^{N_c}\sum_{j=1}^{W_{max}}\sum_{k=1}^{M}2^{M-k}\lambda_{ijk}j \leq W_{ext}\times(\frac{f_t}{F_M})$

6. $\delta_{ij}+\theta_{ik}-\lambda_{ijk}\leq 1,\ 1\leq i\leq N_c, 1\leq j\leq W_{max}, 1\leq k\leq M$

7. $\delta_{ij}+\theta_{ik}-2\lambda_{ijk}\geq 0,\ 1\leq i\leq N_c, 1\leq j\leq W_{max}, 1\leq k\leq M$

It should be noted that with the binary attribute of δ_{ij} and θ_{ik}, the above constraints 6 and 7 effectively constrain $\lambda_{ijk}=\delta_{ij}\theta_{ik}$. The number of variables Num_v and constraints Num_c for this ILP model are $N_cW_{max}+N_cM+N_cMW_{max}$ and $2N_cMW_{max}+2N_c+2$, respectively. Because Num_v and Num_c can easily be in the range of thousands for a core with a large number of N_c and/or W_{ext}, using an ILP solver to obtain the optimal TAM configuration requires large computation time. Hence in the next section we introduce an efficient heuristic for problem $P_{mfw-opt}$, which can achieve near-optimal result within seconds.

4.3.2 Wrapper Optimization Using Fast Heuristics

The algorithm for MFCW design (*MFCWD*) takes as inputs the ATE frequency (f_t), the test parameters of the multi-frequency core (C), the TAM width (W_{ext}), the pre-determined possible shift frequency $\{F_1,...,F_M\}$, the number of clock domains N_c and the maximum test power consumption P_{ave}, and it outputs the wrapper design VC, including the shift frequency f_{shift_i} and the number of VTB lines VTB_{VC_i} for each virtual core VC_i. The pseudocode for this procedure is shown in Figure 4.7.

The algorithm initializes the virtual cores, by assigning to each VC the wrapper input cells, the scan chains and the wrapper output cells which operate in its clock domain (line 1). In line 2 all the VTB lines are initialized to operate at the lowest possible frequency F_M. Line 3 computes the power consumption P_{curr} (at this moment P_i is the power consumption for clock domain i when shifted at F_M) and if $P_{curr} > P_{ave}$ then the program exits because it cannot satisfy the power consumption constraint (line 4). Otherwise, each virtual core VC_i is first allocated one VTB line and then the single frequency core wrapper design (*Design_wrapper* [73]) is performed to get an initial testing time (lines 5-8), which will be used as the starting point for virtual test bus line allocation (lines 9-21).

Algorithm 4.1: *MFCWD*

INPUT: C, W_{ext}, f_t, N_c, *NoWeights*, $\{F_1, ..., F_M\}$, P_{ave}
OUTPUT: $VC = \{VC_i | i = 1...N_c\}$, f_{shift_i}

1. Initialize VC;
2. $N_{vtb} = W_{ext} \times \frac{f_t}{F_M}$; $N_{assigned\_vtb} = 0$;
3. $P_{curr} = \sum_{i=1}^{N_c} P_i$;
4. **if**($P_{curr} > P_{ave}$) **exit**;
. /*First assign every VC 1-bit VTB line*/
5. **for** i from 1 to N_c {
6. $f_{shift\_i} = F_M$; $VTB_{vc_i} = 1$;
7. $N_{assigned\_vtb}$++;
8. **do** SFCWD;
. }
. /*Wrapper optimization with different power weight*/
9. **for** *powerWeight* from 0 to $NoWeights - 1$ {
10. **while** ($N_{vtb} > N_{assigned\_vtb}$) {
11. find VC_{max} with TAT $\tau_{max} = \max\{\tau_i\}$ for all VCs;
12. copy VC_{max} to VC_{temp};
13. $N_{temp} = N_{assigned\_vtb}$;
14. compute P_{curr};
15. $AssignVTBtoVC(VC_{temp}, P_{curr}, P_{ave}, N_{vtb}, N_{temp}, powerWeight)$;
16. **if** ($\tau_{temp} < \tau_{max}$) {
17. copy VC_{temp} to VC_{max};
18. $N_{assigned\_vtb} = N_{temp}$;
. } **else** {
19. **break**;
. }
. }
20. $T_{shift} = \max\{\tau_i\}$ for all VCs;
21. record the VC design with the minimum T_{shift};
. }
22. **return** f_{shift_i}, VC;

Figure 4.7: Pseudocode for Multi-Frequency Wrapper Design.

Algorithm 4.2: *AssignVTBtoVC*

INPUT: VC^{temp}, P_{curr}, P_{ave}, N_{temp}, N_{vtb}, *powerWeight*
OUTPUT: τ_{temp}, f_{temp}, $VTB_{VC^{temp}}$, N_{temp}

```
1. τ_orig = τ_temp;
2. P_orig = P_temp; P_others = P_curr − P_orig;
3. while ((τ_temp ≥ τ_orig) && (N_vtb > N_temp)) {
4.    f_temp = F_M; VTB_VC_temp = VTB_VC_temp × f_temp / F_M;
5.    VTB_VC_temp++; N_temp++;
6.    noTrials = M;
7.    minCost = ∞;
.     /*Find the shift frequency with the minimum cost*/
8.    while (−−noTrials > 0) {
9.       do SFCWD;
10.      compute τ_temp, P_temp;
.       /*Build the cost function*/
11.      currCost = (τ_temp − τ_orig)
.                 + powerWeight / normalWeight × (P_temp − P_orig);
12.      if (τ_temp < τ_orig && currCost <= minCost) {
13.         minCost = currCost;
14.         record the current virtual core wrapper design; }
15.      if (VTB_VC_temp%2 == 0 && P_others + 2P_temp <= P_ave) {
16.         f_temp = f_temp × 2;
17.         VTB_VC_temp = VTB_VC_temp ÷ 2;
18.      } else {
19.         break; }
.     }
.  }
20. return τ_temp, f_temp, VTB_VC^temp, N_temp;
```

Figure 4.8: Procedure for Assigning VTB Lines to the Bottleneck Virtual Core.

Depending on N_{vtb}, the algorithm proceeds as follows. First, all the virtual cores are sorted based on their TAT and the bottleneck virtual core VC_{max} (with longest TAT) is identified (line 11). Then the following steps will iteratively assign the remaining VTB lines to virtual cores. The basic idea is to assign more virtual test bus lines to the bottleneck virtual core and at the same time try different possible shift frequencies. Although increasing

the frequency will lower TAT, if the current bottleneck VC is assigned a higher frequency without considering the increase in power, a suboptimal solution may be obtained because the available power budget for the next iteration is reduced. To account for this problem, we build a cost function that combines TAT and power, and we select the shift frequency that can obtain the minimum cost instead of minimum TAT. This is done in Algorithm 4.2, which assigns VTB lines to the bottleneck VC. *NoWeights* number of power weights in the cost function are tried and we select the one which gives the shortest TAT (line 21).

In Algorithm *AssignVTBtoVC* (shown in Figure 4.8), only one VTB line that operates at F_M is assigned each time. As a result, the bottleneck VC is first transformed to a temporary VC which operates at F_M (line 4). The cost function is built as in line 11, in which *normalWeight* is a constant used to match the TAT and the power consumption into comparable values. In our experiments, we select $NoWeights = 100$ and $normalWeight = 200$[3] to limit the run time to a few seconds. Inside the inner loop (lines 8-19), the algorithm selects the shift frequency that minimize the cost and at the same time satisfies the power constraint (lines 12, 15). Whenever a VTB line is assigned, SFCW design algorithm is performed again to get the new testing time (line 9). This program exits when the TAT of the bottleneck VC is reduced or all the virtual test bus lines are assigned with no TAT reduction.

The worst-case complexity of the SFCW design algorithm *Design_wrapper* is shown to be $O(sc \cdot \log sc + sc \cdot W_{ext})$ in [73], where sc is the number of internal scan chains. The worst-case complexity of the proposed *MFCWD* algorithm is $O(\sum_{i=1}^{N_c} sc_i \cdot \log sc_i + W_{ext} \cdot sc_{max} \cdot \log sc_{max} + W_{ext}^2 \cdot sc_{max})$, where sc_i and sc_{max} are the number of internal scan chains for clock domain i and the maximum number of scan chains of all clock domains, respectively. The computational complexity is therefore linear in the number of clock domains and quadratic in the number of external TAM wires. For a core with a fixed number of clock domains, it is quadratic in the number of external TAM wires.

[3] We have also tried with $NoWeights = 1000$ and $normalWeight = 2000$ in the experiments and obtained the same results.

$f_{func}(MHz)$	N_{in}	N_{out}	N_{bi}	P	N_{sc}	$SC_{length,\ i}$
200	109	32	72	2572	16	{168 168 166 166 163 163 163 163 162 162 162 162 151 151 151 151}
133	144	67	72	450	3	{150 150 150}
120	89	8	72	930	10	{93 93 93 93 93 93 93 93 93 93}
75	111	31	72	1314	6	{219 219 219 219 219 219}
50	117	224	72	2605	5	{521 521 521 521 521}
33	146	68	72	576	11	{82 82 82 81 81 81 18 18 17 17 17}
25	15	30	72	40	4	{10 10 10 10}

Table 4.1: hCADT01 Clock Domain Information.

4.4 Experimental Results

Since no existing approaches have tackled the multi-frequency embedded core testing problem, it is difficult to provide a one to one comparison to previous work. We have decided to analyze the trade-offs of the proposed solution, in terms of the number of ATE channels, testing time, the number of internal VTB lines, and test power dissipation. Therefore, we present results here for a hypothetical, but representative multi-frequency embedded core hCADT01. This core has seven clock domains as shown in Table 4.1, where f_{func} denotes the functional frequency; N_{in}, N_{out}, N_{bi} and N_{sc} are the number of inputs, outputs, bidirectionals and scan chains in the specific clock domain, respectively; the length of each scan chain in clock domain i is shown in column $SC_{length,\ i}$; and P is the power consumption when shifting at 100MHz and is calculated as $P_i = \sum_{j=1}^{|SC_{length,i}|}(l_j|l_j \in SC_{length,i})$ (We assume the power consumption of a VC is proportional to the number of memory elements in it). Note that since the maximum internal frequency for the experimental core is $f_{max} = 200MHz$, and we assume that the maximum operational frequency of the ATE is 120MHz in our experiments, the ATE will shift test data at $f_t = 100MHz$, thus synchronizing with a division of f_{max}.

Table 4.2 presents the necessary shifting time for each test pattern and the number of VTB lines of hCADT01 with varied TAM width W_{ext}, when different power constraints P_{ave} are considered. T_{lb} is acquired by the ILP method using a public linear programming solver lp_solve [151] when the number of possible shift frequencies $M = 4$. Using the heuristic presented in Section 4.3.2, T_{shift_1} and N_{vtb_1} are the TAT and number of VTB

P_{ave}	W_{ext}	T_{lb}	M=1		M=4	
			T_{shift_1}	N_{vtb_1}	T_{shift_4}	N_{vtb_4}
1500	16	20.84	41.68	23	20.84	39
	14	20.84	41.68	23	20.84	39
	12	20.84	41.68	23	20.84	39
	10	20.84	41.68	23	20.84	39
	8	20.84	41.68	23	20.84	39
	7	20.84	41.68	23	20.84	39
	6	20.84	41.68	23	20.84	39
	5	21.24	41.68	23	25.04	34
	4	29.76	41.68	23	29.76	30
	3	41.68	41.68	23	41.68	22
	2	59.88	63.52	16	59.88	18
	1	116.04	127.04	16	116.04	12
4500	16	6.72	10.42	23	7.44	54
	14	7.51	10.42	23	8.88	51
	12	8.79	10.42	23	10.42	31
	10	10.42	12.78	20	11.62	25
	8	12.82	15.88	16	14.88	22
	7	14.73	20.84	23	15.63	23
	6	17.52	20.84	23	19.2	26
	5	20.85	25.56	20	23.24	19
	4	29.01	31.76	16	29.01	14
	3	38.36	41.68	23	38.36	21
	2	58.02	63.52	16	58.02	11
	1	116.04	127.04	16	116.04	11
∞	16	6.4	7.94	16	7.44	26
	14	7.33	10.42	23	8.88	26
	12	8.73	10.42	23	10.42	25
	10	10.33	12.78	20	11.62	19
	8	12.81	15.88	16	14.88	20
	7	14.73	20.84	23	15.63	15
	6	17.52	20.84	23	19.18	22
	5	20.85	25.56	20	23.24	19
	4	29.01	31.76	16	29.01	11
	3	38.36	41.68	23	38.36	21
	2	58.02	63.52	16	58.02	11
	1	116.04	127.04	16	116.04	11

Table 4.2: Test Application Time and Number of VTB lines for hCADT01 with Different TAM Width under Various Power Constraints.

lines obtained when $M = 1$ (i.e., all VCs are constrained to shift at the same frequency); while T_{shift_4} and N_{vtb_4} stands for the values acquired when $M = 4$. We can observe that $T_{lb} = T_{shift\_4}$ when $W_{ext} \leq 4$, which shows the heuristic yields optimal results in these cases. When the external TAM width is larger, *lp_solve* does not run to completion in 10 hours, using a 900MHz Pentium III PC with 256MB memory. As a result, for $W_{ext} > 4$, the lower bounds are obtained using LP-relaxation (the variables θ_{ik} and hence also λ_{ijk} in the ILP model were "relaxed" to reals). Due to the nature of LP-relaxation these lower bounds are *not* "tight", which implies that they may *not* be reachable with integer values. Nevertheless, we can observe that the proposed heuristics generate close values to them, which proves the effectiveness of the proposed heuristics. As a comparison, the execution time of the heuristic is, however, only a few seconds.

Even when there is no power constraint (i.e., $P_{ave} = \infty$), we can observe from Table 4.2 that the shifting time is reduced for almost all the given TAM widths from 1 to 16 if we let different clock domains shift at distinct frequencies. We can also observe that T_{shift_4} is much shorter than T_{shift_1} when the power constraint is tighter. For example, when the given TAM width is $W_{ext} \geq 6$ and the power constraint $P_{ave} = 1500$, T_{shift_4} is only half of T_{shift_1}. This is because all the VCs are constrained to shift at $12.5MHz$ to meet the power requirements in the single-frequency shift architecture ($M = 1$), and clock domain 5 ($f_{func} = 50MHz$) dominates with TAT=41.68μs. For $M = 4$, clock domain 5 is able to shift at $25MHz$ which results in TAT=20.84μs, while still meeting the power constraint. However, the number of internal VTB lines N_{vtb_4} is larger than N_{vtb_1}, thus leading to a larger routing overhead. There are also many cases that both T_{shift_4} and N_{vtb_4} are improved, for example, when $W_{ext} = 4$ and $P_{ave} = 4500$. As a result, the proposed multi-frequency wrapper architecture facilitates system integrators to trade-off scan shifting time, routing overhead and power consumption when selecting their test strategy.

In terms of DFT area overhead, VTB-DIU and VTB-MIU blocks need one flip-flop for each virtual test bus line and additional logic for multiplexing/de-multiplexing if the shift frequency is lower than the tester frequency. The capture window size and the number of clock domains decide the hardware overhead of the scan control block. Even for cores with a high number of clock domains the DFT area overhead is in the range of hundreds of gates, which is insignificant compared with the size of today's complex multi-frequency cores.

4.5 Concluding Remarks

In this chapter, we proposed a novel wrapper design for testing IP cores with multiple clock domains, which, by means of a capture window, facilitates multi-frequency at-speed testing, while accepting data from a low-speed tester, at the expense of small on-chip area overhead. We also proposed an ILP model and efficient heuristics to optimize the wrapper in terms of test application time under a given average test power constraint. By allowing scan chains in different clock domains to shift test data at distinct frequencies, system integrators can easily trade-off the testing time, routing overhead and test power consumption when selecting their test strategies.

Although the at-speed testing strategy proposed in this chapter is able to uncover some timing-related defects, with the shrinking feature size of today's process technology, it is inevitable to develop dedicated delay tests to increase circuit reliability and manufacturing yield. Hence, in the next chapter, we present a new test architecture for core-based SOCs containing two-pattern tested cores, which is necessary to detect delay faults and CMOS stuck-open faults.

Chapter 5

Two-Pattern Test of Core-Based SOCs

Existing approaches for modular manufacturing test of core-based SOC devices do not provide any explicit mechanism for delivering two-pattern tests, which is necessary to achieve a reliable coverage of delay and stuck-open faults. Although wrapper input cells can be enhanced with two memory elements to address this problem, this will incur a large DFT area overhead. This chapter proposes a novel architecture for broadside two-pattern test of core-based SOCs, without any loss in fault coverage and without increasing the size of the wrapper input cells. The proposed solution combines the dedicated bus-based test access mechanism and functional interconnects for test data transfer, which provides full controllability of the wrapper input cells in the two consecutive clock cycles required by two-pattern test. New algorithms for test access mechanism design and test scheduling are also proposed to optimize the proposed test architecture in terms of testing time.

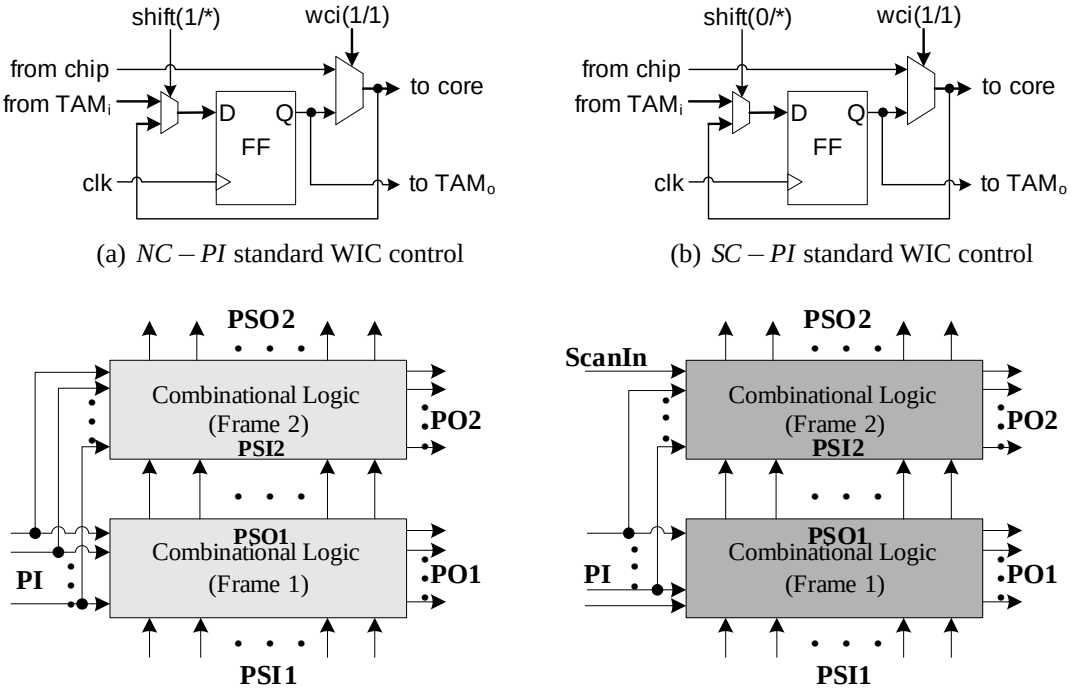
The organization of this chapter as follows. Section 5.1 gives preliminaries for the research work presented in this chapter, outlines the motivation behind it and summarizes the contributions of it. Section 5.2 introduces our proposed SOC test architecture for two-pattern test and describes the necessary DFT support at the core level. Next in Section 5.3, we adapt an existing wrapper/TAM co-optimization algorithm [76] to optimize the proposed architecture at the system level. Section 5.4 contains experimental results for a revised version of ITC'02 benchmark SOCs [115], and finally Section 5.5 concludes this chapter.

5.1 Preliminaries and Summary of Contributions

As discussed in Chapter 2, one way to model VLSI chip's physical defects is to abstract them as stuck-at faults [13]. Stuck-at fault model has been extensively studied and was shown to be effective in verifying the logic correctness of digital circuits. However, digital ICs are generally synchronized using clock signals, which may also have timing failures without logic errors. With the shrinking feature size of CMOS technology, this type of timing-related defects appear more often in fabricated chips [40]. Although application of stuck-at fault tests at rated-speed (as shown in Chapter 4) can uncover some of the delay defects, it was shown in [36, 121] that this technique is not sufficient. Functional test can be used to address timing verification and even detection of un-modeled defects, nevertheless its main drawback lies in the low fault coverage for today's complex circuits, whose transistor to pin ratio is continuing to increase. What's more, as pointed out in [85] by Kapur and Williams, a higher test quality for each core is required to achieve acceptable overall quality of the SOC, when compared to the case that the core itself is a chip. As a result, to increase circuit reliability and manufacturing yield through speed sorting, system integrators are constrained to develop dedicated delay fault tests and the associated testing strategies.

As discussed in Section 2.3, to apply delay fault tests, at least two ordered patterns in *consecutive* clock cycles are necessary: the first *launch* (*initialization*) pattern V_1 initializes the circuit to a certain state, and then the second *capture* (*excitation*) pattern V_2 provokes the fault and captures its effect to the outputs. In addition to testing delay faults, two-pattern test can also be used to detect the CMOS stuck-open fault, which is used to model the defects that cause the transistors to be permanently off [146]. The two techniques to apply two-pattern test for designs using standard SFFs, i.e., broadside testing and skewed-load testing, are both widely accepted in practice with their specific strengths and limitations. We mainly consider broadside testing technique for core-based SOC here.

Since there has been extensive research on DFT and ATPG techniques for two-pattern tests over the last couple of decades [13], the question is how do the existing methods adapt to core-based SOCs [180]? This adaption is an open issue, since, in addition to the standard test-quality problems, core-based SOCs present new challenges, in particular in



(a) *NC – PI* standard WIC control

(b) *SC – PI* standard WIC control

(c) *NC – PI* Two-pattern ATPG model

(d) *SC – PI* Two-pattern ATPG model

Figure 5.1: Wrapper Input Cells and Combinational ATPG Models for Broadside Testing when Adapting Existing Approaches to Control Embedded Core's Primary Inputs.

terms of test development for core providers and TAM design and optimization for system integrators.

One way to deliver two-pattern test to embedded cores is to exploit SOC's architecture-specific information and to reuse on-chip functional interconnect as TAM. Various functional access strategies [18, 41, 131, 142, 172] discussed in Section 3.2.3 can be used or adapted to apply two-pattern test. Regardless of their potential benefits in the long term, unless implemented automatically using a reliable test tool flow, these architecture-specific DFT methodologies do not provide reusability, flexibility and inter-operability, and hence are not desirable. On the other hand, although recent research advances (discussed in Section 3.3.3) and standardization efforts for modular SOC manufacturing test (i.e., IEEE Std. 1500 [67, 55]) support structural test automation, to the best of our knowledge, most prior work in this domain is based on one-pattern test, i.e., no explicit mechanism for two-pattern test is provided. The main challenge to apply two-pattern test for embedded cores lies in

the fact that the cores' inputs are difficult to be fully controlled in consecutive clock cycles with standard IEEE 1500 wrapper cells (shown in Figure 3.3(a)), thus resulting in fault coverage loss if broadside testing for delay faults is used. Based on the analysis in [85], if the system integrator wants to achieve 90% delay fault coverage, when there are five cores requiring delay tests, each core requires a delay fault coverage of about 98%. To achieve such high fault coverage, the PIs of the embedded cores must be fully controllable in both launch and capture cycle. The core provider can implement the WIC with two memory elements to store the consecutive patterns. This method, however, incurs large DFT area overhead when the input count of the embedded cores is large. Therefore, in the following we discuss how to adapt the existing methodologies for two-pattern tests with standard IEEE 1500 WIC design, which motivates our proposed test architecture.

In broadside testing, the pseudo-input part of the excitation vector (i.e., the part that is loaded in the internal flip-flops) is generated through functional justification. However, in order to emulate the functional core behavior, we analyze two options for controlling the PIs of the embedded core, which can reuse the existing strategies for TAM design and optimization without any need for considering the two-pattern application as a special case:

- *Non-Controlled Primary Inputs (NC-PI):* This test scenario assumes that PIs are scanned for fully controlling the initialization vector, however they keep the same value (frozen) for the excitation vector; the control of the WICs and the associated broadside ATPG model are shown in Figures 5.1(a) and 5.1(c), where labels on the $shift$ and *wci* multiplexer controls show the values during the launch/capture cycle;

- *Serially-Controlled Primary Inputs (SC-PI):* After the PIs are scanned in for the initialization vector, in order to obtain the excitation vector, they are updated through an extra shift (using TAM data as input for the first WIC) during the launch cycle; the control of the WICs and the associated broadside ATPG model are shown in Figures 5.1(b) and 5.1(d);

Since $NC-PI$ and $SC-PI$ control mechanisms can reuse the existing TAM design and optimization algorithms, an obvious question is why do we need a new test architecture? The answer lies in the *quality of delay tests*. While the delay fault coverage loss caused

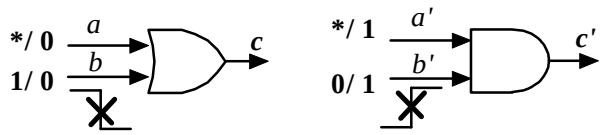
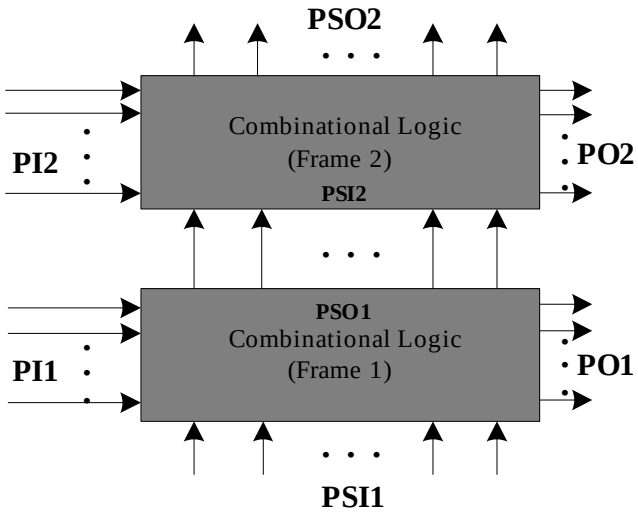


Figure 5.2: Fault Coverage Loss with $NC-PI$ and $SC-PI$ ATPG Model.

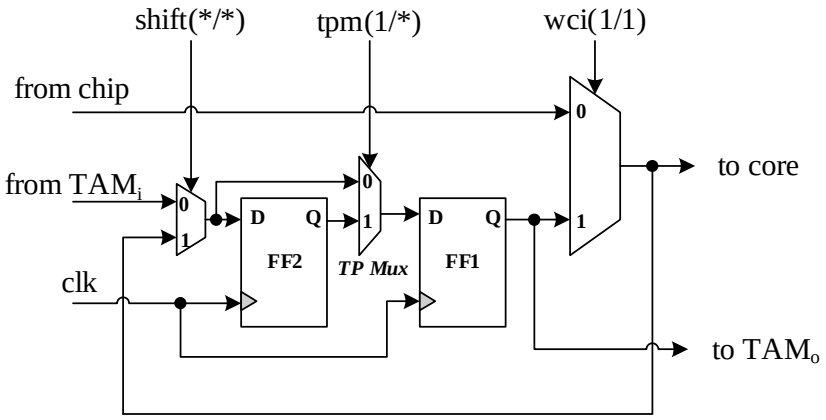
by the connection between pseudo-output 1 and pseudo-input 2 *must* be ignored (these are redundant faults and will never be activated during functional operation, which is an advantage of broadside testing over skewed-load testing and enhanced-scan [145]), the coverage loss due to PI sharing between the two time frames (see both $NC-PI$ and $SC-PI$ in Figures 5.1(c) and 5.1(d), respectively) is unacceptable.

The limitation of the $NC-PI$ and $SC-PI$ control mechanisms can be easily illustrated by testing the b and b' inputs of the OR and AND gates for falling and rising transitions, respectively (see Figure 5.2). For example, if OR input b precedes input a in the wrapper scan chain, then the falling transition on input b is untestable by $NC-PI$ and $SC-PI$, since either freezing ($NC-PI$) or shifting ($SC-PI$) the PI value, $b=1$ required for initialization will conflict with $a=b=0$ necessary for excitation in the following second capture cycle. One might argue that this problem can be solved by structural modifications (i.e., wrapper cell reordering), however, this may be prohibited due to routing constraints and it also brings additional design effort, due to its dependence on test set. In addition, two-pattern test set compaction for $NC-PI$ and $SC-PI$ control mechanisms will introduce additional constraints (caused by freezing and shifting), which are difficult to satisfy when the care-bit density in each vector is increasing (i.e., when test set size decreases).

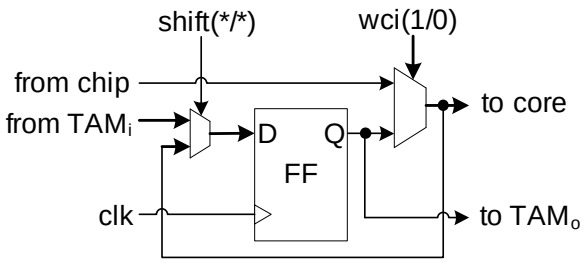
Therefore, to ensure a high delay fault coverage, a *Parallelly-Controlled Primary Inputs* (PC-PI) ATPG model as shown in Figure 5.3(a) is necessary. This $PC-PI$ ATPG model guarantees that *any* arbitrary primary input value can be justified for both initialization and excitation vectors. The easiest way to implement such ATPG model is to double-buffer the core's wrapper input cells (called *enhanced wrapper input cell*), as depicted in Figure 5.3(b). When the CUT is in one-pattern test mode (e.g., stuck-at test), the load through two flip-flops will incur a needless clock cycle for each core input. Hence a multiplexer *TP*



(a) *PC – PI* two-pattern ATPG model [56]



(b) *PC – PI* enhanced WIC control



(c) *PC – PI* standard WIC control

Figure 5.3: PC-PI ATPG Model and The Two Corresponding Wrapper Input Cell Designs.

mux is introduced to bypass $FF2$ in one-pattern test mode to reduce its loading time. As observed in Figure 5.3(b), each enhanced WIC introduces an extra flip-flop and an extra multiplexer (that is, an extra SFF) compared with standard WIC design. Since embedded cores have no pin constraints and may have a large number of input ports, using the enhanced WIC may incur a large DFT area overhead.

To achieve the same delay fault coverage without using enhanced WIC, in this chapter, we describe an approach that is able to implement PC-PI ATPG model with standard WIC design. The two main contributions of this chapter, detailed in Sections 5.2 and 5.3, are as follows:

- we propose a novel $Producer - CUT$ test architecture for two-pattern tested cores based on the PC-PI control mechanism shown in Figure 5.3(c). Without increasing the wrapper input cell size, we achieve the same fault coverage by letting the second test vector V_2 justified through a parallel load from the core's *producer*[1];
- to handle the extra test scheduling conflicts arisen from test resource sharing for two-pattern tested cores in this new test architecture, we also present effective and efficient heuristics to optimize it in terms of test application time.

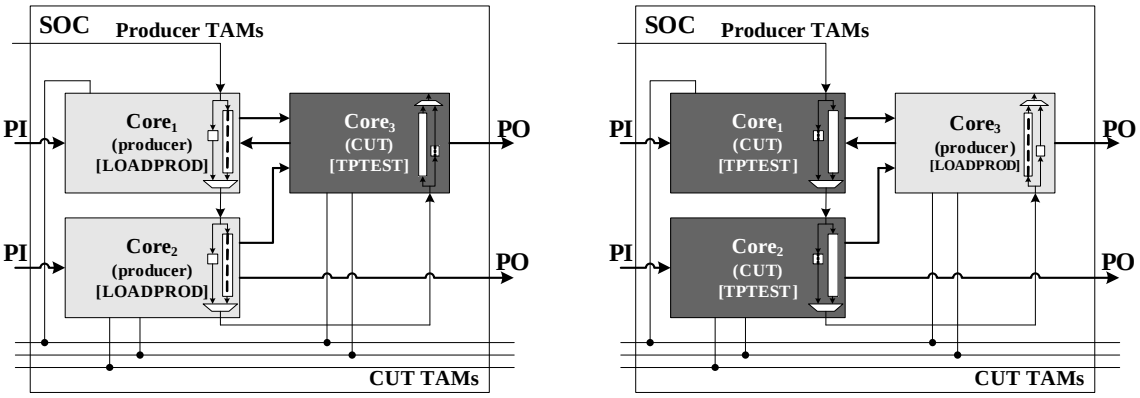
5.2 Proposed Architecture for Two-Pattern Test

The proposed approach considers, as a starting point, that all embedded cores in the SOC are 1500-wrapped [67]. In addition, UDLs are also treated as 1500-wrapped cores. In this section, we first introduce the generic two-pattern testing process using the proposed architecture, and then we discuss the necessary DFT support for the proposed methodology.

5.2.1 The Two-Pattern Testing Process

The proposed approach uses the WICs of the CUT to control the PIs of the launch vector and it exploits the WOCs of its producer cores to control the PIs of the excitation vector.

[1] *For a given $Core_i$, the producers are the cores which feed its primary inputs in the normal (functional) mode.*

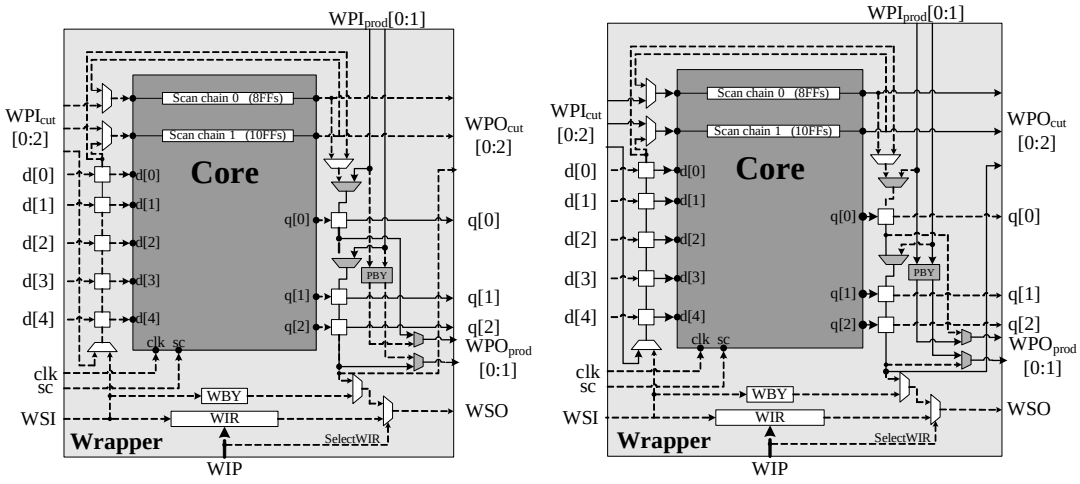


(a) Session 1 for two-pattern test of core 3 (b) Session 2 for two-pattern test of cores 1 and 2

Figure 5.4: Proposed Producer-CUT Architecture for SOC Two-Pattern Test.

As shown in Figure 5.4, this test session-level Producer-CUT dichotomy leads to a division of the SOC's TAM into producer TAMs and CUT TAMs, detailed in Section 5.3.

The proposed broadside two-pattern testing process can be explained as follows. The IEEE Std. 1500 instruction set is extended to support two custom operational modes, *Load-Prod* for the producer cores and *TpTest* for the two-pattern tested CUT. In the TpTest mode, the loading of the initialization vector into the internal scan chains and the application of the first initialization vector are done in the same way as for the InTest mode. However, since TpTest requires another capture cycle, an internal control mechanism for applying the second excitation vector must be provided. On the one hand, the PSIs of the excitation pattern (the internal scan chain part) is generated through functional justification. On the other hand, the PI part is provided using the functional inputs by setting the producer cores in the producer LoadProd mode. To speed up this loading time, we propose that only the WOCs of the producer cores are connected as scan chains in the producer TAM lines. This can be seen in Figures 5.4(a) and 5.4(b), where two test sessions are required to test a hypothetical SOC for delay faults or CMOS stuck-open faults. To ensure two consecutive controllable PI vectors, the CUT's WBR multiplexer control signals are switching between the WICs of the CUT for initialization and the WOCs of the producers for excitation. By exploiting the existing producers' WOCs for storing the PI part of the excitation vector, an *emulation of the enhanced-scan* is provided only for the PIs of each two-pattern tested core.



(a) LoadProd mode (b) Bypass mode

Figure 5.5: An Example Producer Core in LoadProd and Bypass Modes.

5.2.2 1500-Compatible Core Wrapper Design for Two-Pattern Test

Additional DFT hardware is necessary to decode the newly-introduced instructions for two-pattern test and ensure the proper activities of the core wrapper in TpTest or LoadProd mode.

To support the proposed TpTest mode which applies two consecutive test patterns as in the PC-PI ATPG model, in the wrapper of two-pattern tested core we need to add some logic to decode the newly-introduced TpTest instruction. It is important to note, however, the at-speed switch of the *wci* signal (see Figure 5.3(c)) for delay fault testing requires the core wrapper to be controlled by a rated-speed clock signal. To support the proposed LoadProd mode for the producer cores, however, in addition to the extra logic to decode the new LoadProd instruction, the wrapper needs to be revised to be able to load test data into only its wrapper output cells. As shown in Figure 5.5, for the example core, two producer TAM lines are used to load test data into its WOCs in LoadProd mode (the real lines indicate which paths are enabled). In any other mode (Figure 5.5(b) shows Bypass mode), the producer TAM lines bypass the core through a parallel bypass register (PBY). Hence, four extra multiplexers and a 2-bits bypass register are added to the wrapper in this example, which is much smaller when compared to the enhanced WIC design that implements an extra SFF for each core input terminal.

To assess the impact of the new LoadProd and TpTest instruction on the wrapper area, a public processor core was synthesized to 0.18 micron TSMC technology [157]. When compared to a standard IEEE 1500 implementation, the additional overhead for two-pattern test is under 1%.

5.3 Proposed Architecture Optimization

Having introduced the $Producer-CUT$ SOC test architecture supported by the new LoadProd and TpTest instructions, the $PC-PI$ ATPG model, and the wrapper design required on the core provider's side, this section concentrates on how to optimize the proposed architecture, which is necessary on the system integrator's side.

5.3.1 Test Conflicts

CUT in TpTest mode needs the cooperation of its producers to supply the second excitation vector, which introduced extra test conflicts as follows:

- *Producer-CUT Conflict*: Producers and the CUT **cannot** be tested at the same time. This is because, the producer needs to utilize its WOCs to capture its test responses, however, at the same time, the CUT needs the producer's WOCs to provide test stimuli. If they are tested concurrently the test data will be corrupted. For example, in Figure 5.4, $Core_3$ should not be tested with $Core_1$ and $Core_2$ concurrently.

- *Shared-Producer Conflict*: Two cores which directly connect to the same producer(s) for the excitation vector, **cannot** be tested at the same time, this is because, both of them require the WOCs of the same producer(s) to provide the test stimuli. For example, in Figure 5.4, if Core 2 also gets inputs from $Core_3$, then $Core_1$ and $Core_2$ should not be tested concurrently.

- *Shared-Bus Conflict*: When two cores are not directly connected, but they communicate through functional busses, they may imply the test conflicts described above and hence **may not** be able to be tested concurrently. For example, as shown in Figure 5.6, since Core 1, Core 2 and Core 4 all can transmit data to the bus, it is possible to

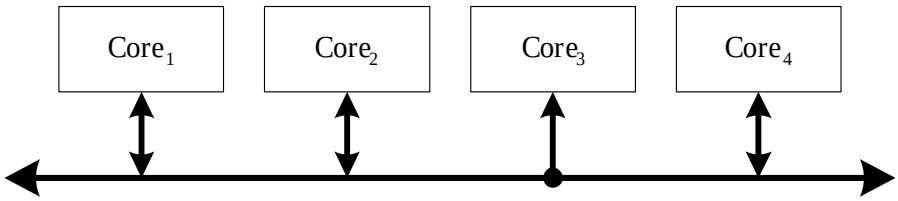


Figure 5.6: Test Conflicts for Multiple Cores on The Same Functional Bus.

select any of them to be the producer of the other cores on bus to provide the second excitation vector. We name this type of producer as a *bus producer*. Suppose $Core_2$ is selected to be the producer of $Core_1$ and $Core_4$ serves as the producer of $Core_3$, obviously $Core_1$ and $Core_2$, $Core_3$ and $Core_4$ cannot be tested concurrently because of the producer-CUT test conflict. However, $Core_1$ and $Core_3$ can be tested at the same time, this is because their producers ($Core_2$ and $Core_4$, respectively) can be loaded at the same time and *only* during the launch/capture cycles, the functional bus needs to be shared, which is negligible compared to the long scan shifting time.

If there are test conflicts between cores, then these cores are called *incompatible cores* and cannot be scheduled concurrently during test. By analyzing the test conflicts between each core, we construct a test incompatibility graph (TIG) by treating each core as a node and by adding an edge between two nodes if the cores are incompatible. This TIG is used in test scheduling Algorithm (*TpTest_Schedule*) described in Section 5.3.3.

5.3.2 TAM Division into Producer and CUT Groups

Producers and CUTs should be fed from **two separate** TAM groups, otherwise additional *indirect* test conflicts maybe introduced. This can be seen from the following example:

Example 5.1 *Suppose the test data is transferred using shared TAM lines between producers and CUTs during two-pattern test. $Core_1$ and $Core_2$ are connected to separate TAM lines, however $Core_1$ shares its TAM lines with $Core_3$, which is a producer to $Core_2$. To test $Core_2$ in TpTest mode, we need $Core_3$ in LoadProd mode, since it shares TAM lines with Core*1*, loading a pattern in $Core_1$ is prohibited at this time, although it uses separate TAM*

lines to $Core_2$. Hence, this indirect *resource conflict leads to test conflict between $Core_1$ and $Core_2$.*

A neat solution to this problem, which is important in particular for complex SOCs with a large number of two-pattern tested cores, is to divide the TAM lines into two groups: G_{prod} for loading the PIs of the excitation patterns in the producers' outputs and G_{CUT} for loading the initialization patterns and unloading the test responses from the CUT. In this way, for the above example, $Core_3$ in LoadProd mode does not affect loading $Core_1$ since they use TAM lines from different TAM groups (hence no additional test conflict between $Core_1$ and $Core_2$ exist).

For the G_{CUT} group, a flexible-width Test Bus architecture is used to support efficient test scheduling. For G_{prod} group, however, we use a daisychain architecture, i.e., long scan chains are constructed over the output terminals of the cores that serve as producers during test, as shown in Figure 5.4. Bypasses are introduced in order to shorten the loading time because only a few cores serve as producers at a specific test session. The main reason for using the daisychain architecture for G_{prod} group is to simplify the control complexity. When a producer core is in the LoadProd mode, the producer TAM lines go through the core's output wrapper boundary cells, otherwise they go through bypass registers (note, it is unnecessary to introduce extra producer bypass instruction because it is compatible with the standard IEEE 1500 Bypass mode). As a result, although the two-pattern test of CUT involves several producers, these producers can be controlled by the LoadProd instructions independently and no extra control signals need to be supplied. In addition, the daisychain architecture for G_{prod} can almost always give a near optimal loading time for a given producer TAM width W_{prod}. Suppose the number of the outputs of a producer is N_o, then its loading time will be $\lceil \frac{N_o}{W_{prod}} \rceil$. As long as $W_{prod} \leq N_o$ (which is realistic in most of the cases), there is no waste for G_{prod} TAM resources except the few bypass cycles. This leads to a near optimal loading time for producers in each test session.

5.3.3 Two-Pattern Test Scheduling

The introduction of Producer-CUT architecture and TAM division into G_{prod} and G_{CUT} groups, leads to new test scheduling algorithms, as explained in this section. Note, we do

Algorithm 5.1 - TpTest_Optimization

INPUT: C_{set}, R, W_{ttl}
OUTPUT: W_{prod}, W_{CUT}, $schedule$, T_{soc}

1. Assign bus producers;
2. $TIG = Construct\_TIG(R)$;
3. For W_{CUT} from $W_{ttl} - 1$ downto 1 {
4. $W_{prod} = W_{ttl} - W_{CUT}$;
5. $testing\_time = TpTest\_Schedule(C_{set}, TIG, W_{prod}, W_{CUT})$;
6. $minTime$ = min{all $testing\_time$};
7. Record W_{prod_min} and the associated $schedule$;
. }
8. $W_{prod} = W_{prod\_min}$;
. $W_{CUT} = W_{ttl} - W_{prod}$;
. $T_{soc} = minTime$;
9. **return** W_{prod}, W_{CUT}, $schedule$, T_{soc};

Figure 5.7: Pseudocode for Optimizing SOC with Two-Pattern Tested Cores.

not consider test scheduling constraints introduced by precedence relationship, preemption and power, which can be addressed using the techniques presented in [70].

Problem P_{TP-opt}*: Given the test set parameters for each core, including the number of primary inputs, primary outputs, bidirectional I/Os, type of test (i.e., one-pattern test or two-pattern test), test patterns and scan chains, and each scan chain length, the functional relationships between cores R, the total TAM width* W_{ttl} *for the SOC, determine the width of each TAM group (*W_{prod}, W_{CUT} *corresponding to* G_{prod}, G_{CUT}*), the assigned TAM width and the wrapper design for each core, and a test schedule for the entire SOC such that:* (i) *the total number of TAM lines used at any time does not exceed* W_{ttl}*; and* (ii) *the overall SOC testing time is minimized.*

The proposed algorithm *TpTest_Optimization* to solve P_{TP-opt} is shown in Figure 5.7. The inputs are the set of cores (C_{set}), the total TAM width (W_{ttl}) and the functional interconnect relationship between cores (R). The outputs are the number of TAM lines allocated to each group W_{prod} and W_{CUT}, the wrapper design for each core, SOC test schedule $schedule$ and the overall test application time T_{soc}. The optimal TAM division, i.e., the combination

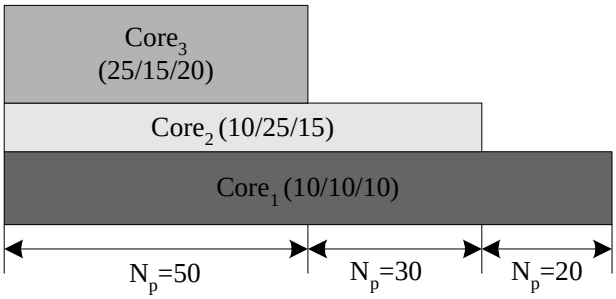


Figure 5.8: Loading Time for Different Patterns ($L_{prod}/wsc_{in}/wsc_{out}$).

of W_{prod} and W_{CUT} that gives the minimum T_{soc} of the SOC, is acquired through enumeration. The enumerative algorithm begins by assigning the core with the least number of outputs (except itself) as the bus producers for two-pattern tested cores on busses (line 1), then based on the test conflicts determined by functional interconnect relationship between cores (R), a test incompatibility graph (TIG) (discussed in Section 5.3.1) is created (line 2). Next, inside the loop (lines 3 to 7) the algorithm will find the optimal TAM division and the system TAT T_{soc}, by enumerating W_{CUT} from the maximum possible value $W_{ttl}-1$ down to 1.

It should be noted that during the enumeration process, we do not need to do TAM design for G_{prod} group because it is already fixed using the daisychain architecture. To optimize G_{CUT} TAM group, we adapt an existing generalized rectangle packing algorithm *TAM_Schedule_Optimizer* [76, 78]. The key novel feature in our approach is that, due to the usage of daisychain architecture for loading producers' outputs, a *dynamic adaptation* of the existing algorithms is necessary, as discussed in the following paragraphs.

Dynamic Rectangle Representation: For one-pattern test of a 1500-wrapped core, when a specific wrapper optimization method is used, the core testing time is a fixed value with a given TAM width. As a result, the core test can be represented as a *static* rectangle, in which the height represents the TAM width and the width stands for the testing time. However, when reusing functional interconnect to transfer test data for two-pattern tested cores, its TAT does not only depend on the time to load its own producers (L_{prod}), wrapper scan-in chains (wsc_{in}) and unload its wrapper scan-out chains (wsc_{out}). The TAT

also depends on the time necessary to load/unload all the other concurrently-tested cores' producers and wrapper scan in/out chains because of the daisychain architecture used for G_{prod} group. To keep the control and computational complexity low, we propose to *align* test patterns for all the concurrently-tested two-pattern tested cores. That is, if several two-pattern tested cores are scheduled at the same time, then the overlapped test patterns for these cores will have the same start times and have the same loading/unloading time for each pattern, called *LoadSize*.

$$LoadSize = max\{\sum L_{prod}, max\{wsc_{in}\}, max\{wsc_{out}\}\} \tag{5.1}$$

where bypass cycles are ignored. Obviously, the core which needs the least time to load its test stimuli will wait for all its concurrently scheduled cores to complete loading. Hence, if for the given core the test schedule changes s times, then for each subset of patterns p_s (for the s distinct divisions of the time allocated to the given core) the testing time will be computed based on the two-pattern tested cores scheduled in each of these s divisions. Since the loading time for each CUT in TpTest mode is variable with its schedule, the rectangles cannot be pre-computed. This can be observed from the following example.

Example 5.2 *Suppose $Core_1$, $Core_2$ and $Core_3$ are two-pattern tested cores and they are scheduled as in Figure 5.8, the loading time for their producers, wrapper scan-in chains and wrapper scan-out chains are shown in the figure. Since the producers and CUTs are connected using the daisychain architecture, without considering the bypass cycles, for the first 50 patterns, the LoadSize for all the three cores will be $max\{L_{prod\_1} + L_{prod\_2} + L_{prod\_3}, wsc_{in\_1}, wsc_{in\_2}, wsc_{in\_3}, wsc_{out\_1}, wsc_{out\_2}, wsc_{out\_3}\} = 45$ clock cycles. However, for the next 30 patterns, once the test for $Core_3$ has been completed, the LoadSize is $max\{L_{prod\_1} + L_{prod\_2}, wsc_{in\_1}, wsc_{in\_2}, wsc_{out\_1}, wsc_{out\_2}\} = 25$ clock cycles. The same reasoning is applied for the last 20 patterns, when $Core_1$ is not concurrently tested with any other cores, where the LoadSize will be 10 instead.*

Adapted Dynamic Rectangle Packing: Figure 6.10 shows the pseudocode for algorithm *TpTest_Schedule*. The algorithm takes the core list C_{set}, TIG and the TAM division as inputs, and it generates the *schedule* for each core and the SOC testing time T_{soc}. The proposed algorithm is based on a generalized rectangle packing algorithm [76, 78] and we only show the differences with respect to the original algorithm.

Algorithm 5.2 - TpTest_Schedule

INPUT: C_{set}, TIG, W_{prod}, W_{CUT}
OUTPUT: *schedule*, T_{soc}

1. Compute collection R_o of rectangles for one-pattern tested core set C_o;
2. Initialize(C_o, d, p);
3. Set $C_{unfinished} = C_{set}$; $w\_avail = W_{CUT}$; (see [76, 78])
4. **While** $C_{unfinished} \neq \emptyset$ {
5. **if** $w\_avail > 0$ {
6. Find unscheduled two-pattern tested core set C'_t;
7. Compute collection R'_t of dynamic rectangles for C'_t;
8. Initialize(C'_t, d, p);
9. Schedule compatible one-pattern tested cores that can be assigned
. preferred TAM width or two-pattern tested cores; (see [76, 78])
10. Schedule compatible one-pattern tested cores that can use the
. resulting idle TAM wires; (see [76, 78])
11. Update L_{prod}, *loadSize*;
12. Update *test_time* for scheduling two-pattern tested cores;
13. } **else** {
14. Update *next_time*;
15. Finish the scheduling core test C_i with ending time *next_time*;
16. Update *this_time*;
17. $w\_avail += w_{tam\_C_i}$;
18. $C_{unfinished} -= \{C_i\}$;
. }
. }
22. **return** *schedule*, T_{soc};

Figure 5.9: Procedure for Test Scheduling with Given Widths for Each TAM Group.

As described earlier, for two-pattern tested cores the test **cannot** be pre-computed and represented as a static rectangle. Its TAT (the width of the rectangle) varies with its schedule and hence its rectangle representation is computed dynamically (line 7). Because of the same reason there are also no static "preferred TAM widths" for two-pattern tested cores. In [76, 78], the procedure *Initialize* is used to compute the preferred width for each core, in which parameters d and p are used to select appropriate "preferred width" for each core and are usually manually selected for SOCs with different available TAM widths to get a better

result. Since we need to call this procedure many times with different W_{CUT} (see Algorithm 1), it is unlikely that a manual selection will lead to an optimal value. Consequently, in our implementation we have fixed the two parameters to $d = 2$ and $p = 1.0$. This may result in a different schedule and a slightly longer testing time in some cases when compared to the result from [76, 78]. Whenever a core is scheduled (line 9, 10), the available TAM width *w_avail* will be deducted with the value of the assigned G_{CUT} TAM width for the core. Once a two-pattern tested core is scheduled, L_{prod}, *loadSize* for the currently scheduled cores in TpTest mode need to be updated (line 11) and their TATs are recalculated (line 12). Once there is no available TAM resources for current test session (i.e., $w\_avail = 0$) the currently scheduled core with the minimum TAT completes its scheduling (line 15), its TAM resources will be released and the algorithm will try to find another unscheduled core which can use the freed TAM lines.

5.4 Experimental Results

To investigate the implication of the proposed approach on the DFT area savings and its impact on SOC testing time, experiments are carried out on ITC'02 SOC benchmark circuits. As described in Section 5.3, the total TAM lines are divided into Producer-CUT TAM groups (G_{prod}/G_{CUT}) in the proposed test architecture. Since different divisions (configurations) will generate different test schedules, we need to obtain the optimal configuration which leads to the minimum TAT. Using the SOCs' specifications detailed in Section 5.4.1, the above issues are investigated in the following three experiments:

Experiment 1 discusses the DFT area savings of the proposed prodcuer/CUT test architecture (Section 5.4.2);

Experiment 2 illustrates the variable TAT with different W_{prod}/W_{CUT} TAM configurations (Section 5.4.3);

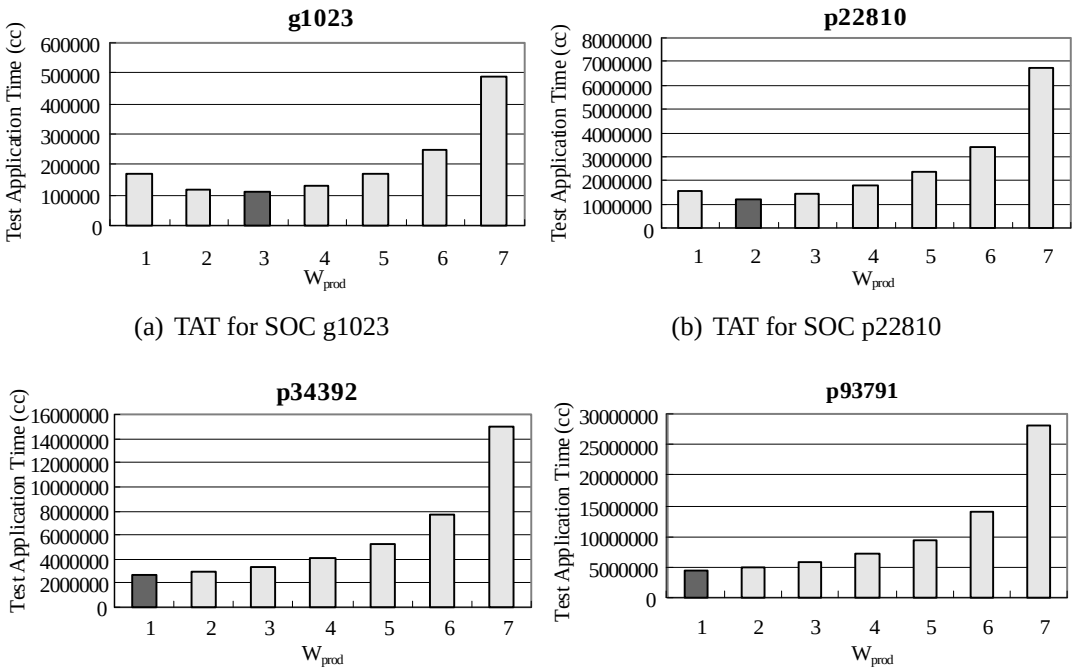
Experiment 3 compares the testing time when using the new TpTest methodology with the case when all cores are one-pattern tested, assuming the test pattern count is the same (Section 5.4.4);

5.4.1 SOC Specifications

Four SOCs: g1023, p22810, p34392 and p93791, which are originally part of the ITC'02 *SOC test benchmarking initiative* [115], are used in our experiment. g1023 is a comparatively small hypothetical SOC, while p22810, p34392 and p93791 are large industrial SOCs. Since the functional interconnects are not provided in the benchmark files, we have decided to randomly generate them to support the proposed approach, including the direct connection between cores and functional busses. We have assumed that the SOCs have $Round(\frac{N_c}{10})$ busses and each bus has a random number p $(3 \leq p \leq min(N_c, 8))$ of cores attached to it, where N_c is the total number of cores in the SOC. In addition, all cores on busses are assumed to be able to transfer data to and from the bus, and hence each one of them can be a bus producer to others. We have also assumed that every core has a random number of $q(1 \leq q \leq 3)$ producers. In addition, we assume that all the cores with internal scan chains are tested using the proposed TpTest modes, while the remaining non-scanned cores are tested using InTest mode. Hence, 12 of 14 cores in g1023, 22 of 28 cores in p22810, 4 of 19 cores in p34392 and 13 of 32 cores in p93791 are selected to be tested in TpTest mode. In addition, we assume that the number of test patterns is equal to the one provided in [115] when cores are two-pattern tested. It should be noted, however, that in reality the number of patterns for delay faults is usually much higher than when targeting single stuck-at faults.

5.4.2 Experiment 1: DFT Area Savings

The area of the core wrapper is mainly determined by the size of the WBR cells. As discussed in Section 5.1, the enhanced WIC design introduces an extra SFF when compared to the standard WIC design. The area of a typical SFF implementation is about ten equivalent 2-input NAND gates. Therefore, the DFT area savings of the proposed architecture can be calculated as $\sum_i^{N_{tp}} \{N_{in}^i + N_{bi}^i\} \times 10$, in which N_{tp}, N_{in}^i and N_{bi}^i are the number of two-pattern tested cores, the number of inputs for two-pattern tested core $Core_i$ and the number of bidirectionals for $Core_i$. Based on the above formula, using the proposed test architecture, SOC g1023, p22810, p34392 and p93791 save 14910, 23820, 4840 and 31320 equivalent 2-input NAND gates, respectively.



(a) TAT for SOC g1023

(b) TAT for SOC p22810

(c) TAT for SOC p34392

(d) TAT for SOC p93791

Figure 5.10: SOC TATs with Variable Producer-CUT TAM Width Configurations.

Because these savings do not come at no expense, the implications on the SOC TAT are discussed in the following sections.

5.4.3 Experiment 2: Optimal Producer-CUT TAM Configuration

Figure 5.10 presents the TATs of the four SOCs, for different widths of the producer TAM (W_{prod}), when the total TAM width W_{ttl} is fixed to 8. To give an exact TAM width division the functional interconnects are fixed in this experiment (i.e., we have randomly generated the functional interconnect only once). It can be seen that the TAT of g1023 is minimum when W_{prod} is 3, the TAT of p22810 is minimum when W_{prod} is 2, while the TATs for p93791 and p34392 are minimum when W_{prod} is 1. This variation is due to the relationship between the total number of the producers' outputs and the internal SFF in the SOCs. On the one hand, in the case of g1023, the number of SFF is comparable to the number of the producers' outputs, hence a large amount of TAT is necessary to load producers' outputs,

thus leading to a higher number of producer TAM lines. On the other hand, for p93791 and p34392, the SFF number is significantly larger than the number of producers' outputs. As a result, the time necessary to load the internal scan chains dominates the SOC TAT and hence only one TAM line is necessary to load the producers' WOCs. The number of SFFs in SOC p22810 is larger than the number of producers' outputs, but the difference is not as large as in the case of p34392 and p93791. Hence two TAM lines are used to load producers' WOCs give the optimum SOC TAT. It should also be noted that the SOC TAT variation with W_{prod} is a convex function based on our observation, i.e., it decreases until it reaches its minimum for the *optimal* W_{prod}, after which point if W_{prod} is further increased then the TAT will grow as well. This is because, once sufficient TAM resources are used to load producers' WOCs, the SOC TAT will be dominated by the loading of the internal wrapper scan chains (i.e., G_{CUT} group). Hence the further decrease of W_{CUT} will obviously lead to an increase of the overall TAT of the SOC.

5.4.4 Experiment 3: Test Schedule and Test Application Time

Tables 5.1, 5.2, 5.3 and 5.4 present results for TpTest of the four benchmark SOCs g1023, p22810, p34392 and p93791 when varying the total TAM width W_{ttl} (note only results for the optimal TAM division are reported). Since the SOC TAT is affected by the functional interconnects, we ran the algorithm for 100 randomly generated interconnects. T_{ave}, $T_{\max}$ and $T_{\min}$ denote the average, maximum and minimum TAT, respectively. The percentage change in TAT using the proposed test architecture is calculated using the formula $\Delta T(\%) = \frac{T_{ave}-T}{T} \times 100$, where T is the result for two-pattern test with enhanced WICs, using the algorithm from [76, 78].

It can be seen that the average SOC TAT increases about 68% for SOC g1023, 43% for SOC p22810, 42% for SOC p34392 and 14% for SOC p93791, respectively. The increase is due to: (i) W_{prod} TAM lines used to load the excitation vector; (ii) Test resource conflicts between cores as described in Section 5.3.1. Note, the increase varies based on the SOC structure (including the functional interconnects, the number of cores in TpTest mode and the core sizes).

	g1023				
	InTest [76]	NEW TpTest			
W_{ttl}	T (cc)	T_{ave} (cc)	$T_{\max}$ (cc)	$T_{\min}$ (cc)	$\Delta T(\%)$
8	76218	126976	152613	107117	+66.60
16	42106	65326	75813	57149	+55.15
24	27906	43906	54325	38518	+57.34
32	21967	33971	41268	28505	+54.65
40	17925	29057	34622	24838	+62.10
48	14794	27236	33549	22353	+84.10
56	14794	26728	33287	21676	+80.67
64	14794	26538	33223	21275	+79.38

Table 5.1: TAT Comparison of The Two Two-Pattern Test Methodologies for g1023.

	p22810				
	InTest [76]	NEW TpTest			
W_{ttl}	T (cc)	T_{ave} (cc)	$T_{\max}$ (cc)	$T_{\min}$ (cc)	$\Delta T(\%)$
8	973995	1251036	1393431	1171321	+28.44
16	504440	653548	742651	607702	+29.56
24	368493	491847	559716	441247	+33.48
32	281438	397183	484224	332677	+41.13
40	241237	337832	410470	291417	+40.04
48	200595	301209	375937	250480	+50.16
56	174485	286088	360592	248916	+63.96
64	174485	278992	360592	239484	+59.89

Table 5.2: TAT Comparison of The Two Two-Pattern Test Methodologies for p22810.

Having reported the average TAT increase for the four SOCs with 100 random functional interconnects, we present the different test schedules of the four SOCs for InTest and TpTest in Figure 5.11 and Figure 5.12. To get the exact schedule the functional interconnect is also fixed in this experiment and only the schedule with optimal W_{prod}/W_{CUT} configuration is shown. For SOC g1023, the total TAM width is 16 and 4 TAM lines are used to load producers' WOCs in TpTest mode. This obviously increases TAT, in addition to the reason that the *LoadSize* for each core is not solely dependent on its assigned TAM width, but also depends on the producers of the other cores that are scheduled at the same time. For SOC p22810 with a total TAM width of 16, producers' WOCs are loaded from 3 TAM lines. Because many two-pattern tested cores are scheduled to be tested concurrently,

	p34392				
	InTest [76]	NEW TpTest			
W_{ttl}	T (cc)	T_{ave} (cc)	$T_{\max}$ (cc)	$T_{\min}$ (cc)	$\Delta T(\%)$
8	2198975	2715071	2752472	2708766	+23.47
16	1077812	1245226	1322366	1114607	+15.53
24	838643	1018869	1164818	855407	+21.49
32	544579	863964	1140994	704813	+58.65
40	544579	836397	1140994	567044	+53.59
48	544579	836274	1140994	567044	+53.56
56	544579	836274	1140994	567044	+53.56
64	544579	835413	1140994	567044	+53.41

Table 5.3: TAT Comparison of The Two Two-Pattern Test Methodologies for p34392.

the load size for those cores may increase. The TAT increases in this case by approximately 32 percent. For SOC p34392, only 1 TAM line is used to load producers' WOCs for a total TAM width of 32 to get the optimal TAT in TpTest mode. Although there are only 4 cores in TpTest mode in this SOC, the TAT increases by 85 percent (from 544579 to 1009879). The main reason for this high increase is because these 4 cores are the largest cores inside the SOC, and the time spent on testing them dominates the overall TAT of the SOC. For this specific functional interconnect in our experiment, $Core_1$, $\{Core_2, Core_{10}\}$ and $Core_{18}$ are incompatible, from Figure 5.12(b) a good part of the testing time (from 153844 to 545326) is wasted in TpTest mode, which is used effectively in InTest mode, however. As shown in Section 5.4.2, the savings in DFT area for p34392 are not significant and, as a result, the system integrator may prefer to use the enhanced WIC design for p34392 to decrease SOC TAT. Nevertheless, unlike p34392, for SOC p93791, the sizes of the cores are similar and, consequently, none of the cores will dominate the whole SOC TAT. As a result, although test conflicts exist between cores, the size of idle rectangles are not too large (Figure 5.12(d)). In this case 2 of 32 TAM lines are used to transfer producers' WOCs and TAT increases by only about 12 percent.

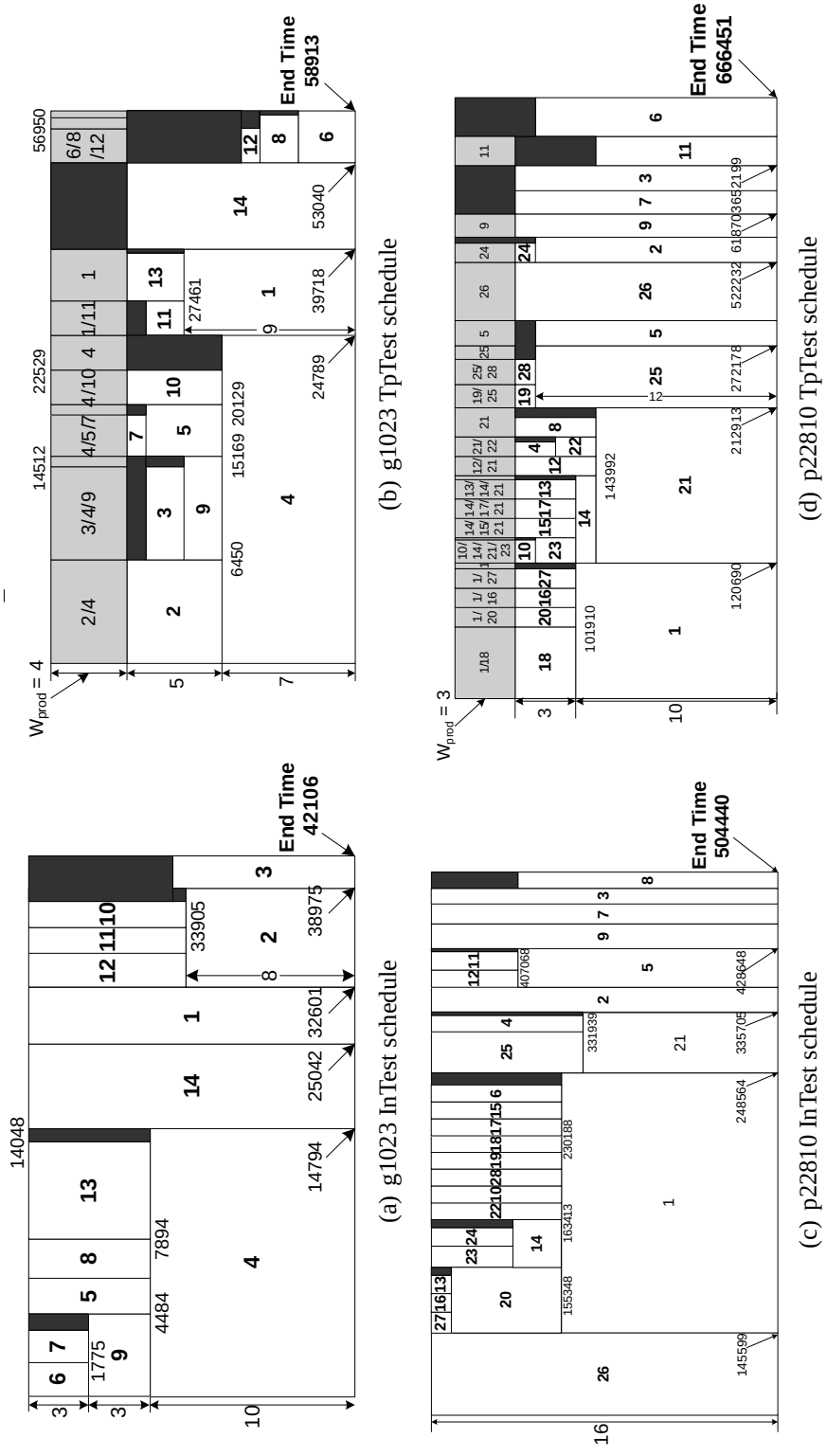
	p93791				
	InTest [76]	NEW TpTest			
W_{ttl}	T (cc)	T_{ave} (cc)	$T_{\max}$ (cc)	$T_{\min}$ (cc)	$\Delta T(\%)$
8	3684114	4228052	4323478	4198829	+14.76
16	1888950	2227268	2386166	2107380	+17.91
24	1302093	1506468	1679554	1287985	+15.70
32	1066517	1206364	1275858	1168159	+13.11
40	880381	1057245	1188497	898920	+20.09
48	694260	841771	979567	754301	+21.25
56	627575	641982	669208	624541	+2.30
64	580610	623852	643842	588030	+7.45

Table 5.4: TAT Comparison of The Two Two-Pattern Test Methodologies for p93791.

5.5 Concluding Remarks

Motivated by the difficulty to deliver high quality delay fault tests to embedded cores, this chapter has presented a novel *Producer-CUT* test architecture for core-based SOCs containing two-pattern tested cores. It was shown how IEEE 1500 wrapper instruction set can be extended with LoadProd instruction for producer cores and TpTest instruction for CUT cores, in order to ensure full controllability of the two-pattern tested core's primary inputs in two consecutive cycles. Solutions to address the optimization of Producer-CUT architecture have also been elaborated. When compared to the case that enhanced WICs are used for two-pattern test, it was demonstrated that the proposed architecture can deliver the same fault coverage with less DFT area overhead and limited SOC TAT penalty.

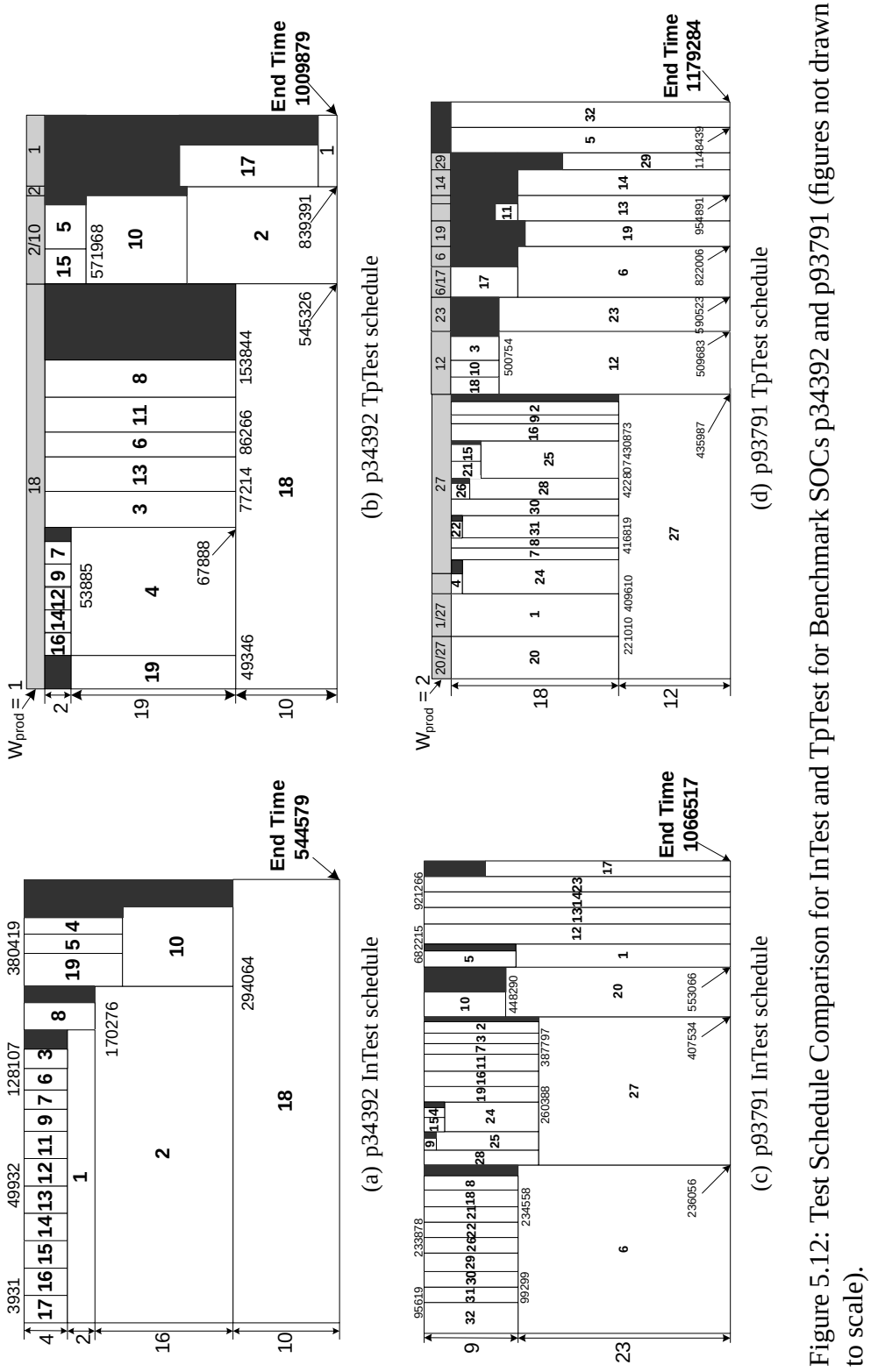
The main idea behind the proposed architecture for two-pattern test is a modular strategy that applies the test stimuli for the CUT through the functional interconnect. Following the same idea, for a one-pattern tested CUT, if we apply (observe) the test stimuli (responses) for its PIs (POs) both through the functional interconnects, we can provide the same testability for this CUT without wrapping it, thus reducing the area overhead and performance penalty brought by the IEEE 1500 wrapper. This observation motivates the modular SOC testing with reduced wrapper count work presented in the next chapter.



(a) g1023 InTest schedule (b) g1023 TpTest schedule

(c) p22810 InTest schedule (d) p22810 TpTest schedule

Figure 5.11: Test Schedule Comparison for InTest and TpTest for Benchmark SOCs g1023 and p22810 (figures not drawn to scale).



(a) p34392 InTest schedule (b) p34392 TpTest schedule

(c) p93791 InTest schedule (d) p93791 TpTest schedule

Figure 5.12: Test Schedule Comparison for InTest and TpTest for Benchmark SOCs p34392 and p93791 (figures not drawn to scale).

Chapter 6

Modular SOC Testing with Reduced Wrapper Count

By utilizing the functional interconnect topology and the WBRs of the surrounding cores to transfer test stimuli and test responses, this chapter shows that some core wrappers can be removed without affecting the controllability/observability and hence the testability of the SOC. We denote such cores without WBRs as *light-wrapped* cores and we present novel modular SOC test architectures for concurrently testing both 1500-wrapped and light-wrapped cores. Since the WBRs of cores that transfer test stimuli and test responses for light-wrapped cores become shared test resources, similar to two-pattern test for embedded cores discussed in Chapter 5, conflicts arise during test scheduling that will negatively impact the test application time. As a consequence, novel optimization algorithms are also presented to alleviate this problem.

The sequel of this chapter is organized as follows. Section 6.1 gives preliminaries for the research work presented in this chapter, outlines the motivation behind it and summarizes the contributions of it. In Section 6.2, we present a novel SOC test architecture with reduced wrapper count and provide the corresponding optimization algorithms. While this new architecture is more effective when the number of light-wrapped cores is large, in Section 6.3, we show how to adapt the TestRail architecture for testing SOCs containing a relatively small number of light-wrapped cores. Next, Section 6.4 discusses the experimental results, and finally, Section 6.5 concludes this chapter.

6.1 Preliminaries and Summary of Contributions

Most prior works on modular SOC test architecture design and optimization (discussed in Section 3.3) assume that all the cores attached to the TAM wires are fully 1500-wrapped, i.e., wrapper cells are placed on all the functional input and output terminals. While this guarantees core isolation during test, and hence high test quality, some embedded cores may have high pin count, and consequently the DFT area overhead associated with the wrappers will increase the cost of the test. Moreover, since both core's inputs and outputs are buffered in the wrapper, at least two sets of multiplexers are required to switch between the functional and test mode of operation. If placed on the critical paths, these multiplexers will lower the maximum operating frequency, thus having a direct impact on the SOC's functional timing performance.

Since the gate count of the unwrapped IP cores (due to area constraints or timing violations) can be large, the question is how can they be tested effectively? One approach is to treat them as interconnect circuitry in between 1500-wrapped cores and test them as non-scanned sequential logic, as discussed in [102, 105]. If there is a large number of memory elements in the unwrapped logic blocks, this approach may present several problems. Firstly, when compared to the scanned version of the unwrapped logic blocks, the fault coverage may significantly decrease despite the increased computational time required for sequential ATPG. This is unacceptable according to the analysis given by Kapur and Williams [85], in which they show a higher test quality for each core is required to achieve acceptable overall quality of the SOC, when compared to the case that the core itself is a chip. Secondly, due to sequential ATPG, the test pattern count will grow, which may also lead to an increase in the overall testing time because the test stimuli/responses are not directly controlled/observed (they have to be shifted in/out through other cores' wrapper cells). Another approach presented in [135, 161] is to justify the unwrapped logic block's test vectors through its surrounding UDL, without affecting the fault coverage. Although this solution is effective in reducing the DFT overhead, its main limitation lies in the fact that it reduces the reusability of the core test sets, since new test sets are required whenever the unwrapped logic block is used in a different SOC environment.

To maintain the controllability/observability of the embedded cores without wrapper

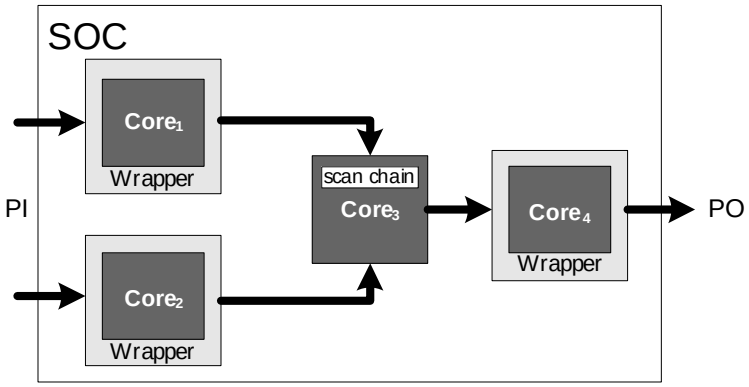


Figure 6.1: Full Controllability and Observability for $Core_3$ without Wrapper Cells.

cells, and to enable test reuse at the same time, we introduce a new concept called ***light wrapper*** and explains how light wrappers can be employed to address the above issues.

6.1.1 Light Wrapper

From the system integrator's standpoint, to test the embedded cores and their interconnects, full controllability and observability needs to be provided at the inputs and outputs of each core. It is important to note to ensure modularity and scalability, the controllability and observability should be test set independent. To achieve this, however, it is not necessary to wrap all the cores' terminals with WBR cells, since the system integrator can also exploit the functional interconnect between cores to transfer the test data. To illustrate this observation, ***producers*** and ***consumers*** are introduced. For a given $Core_i$, its producers are the cores which feed its primary inputs (same as the "producer" defined in Chapter 5) and its consumers are the cores which capture its primary outputs in the normal (functional) mode. Figure 6.1 shows a part of an SOC, where $Core_3$ is not wrapped with WBR cells, however all its producers ($Core_1, Core_2$) and its consumer ($Core_4$) are 1500-wrapped. For InTest of $Core_3$, the controllability of its input terminals is provided through its producers' WOCs, while the observability of its output terminals is provided through its consumer's WICs. In other words, we can shift in its test stimuli through the WOCs of $Core_1$ and $Core_2$, feed in the test stimuli into $Core_3$ through its normal functional path, and then capture its test

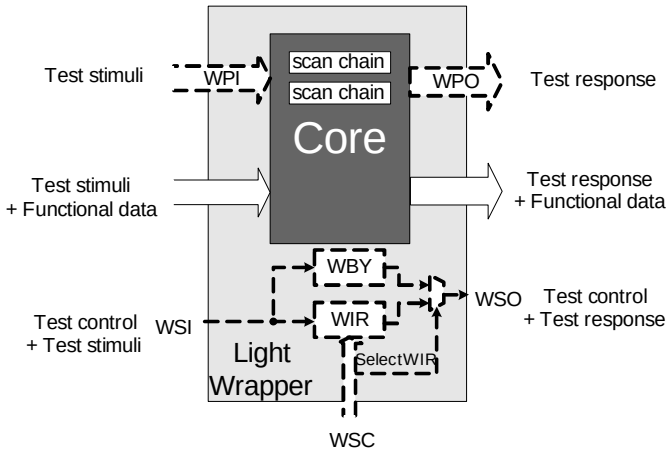


Figure 6.2: Light Wrapper without Wrapper Cells.

responses and shift them out through the WICs of $Core_4$. Note, $Core_3$ cannot achieve a high fault coverage using ExTest of $Core_1$, $Core_2$ and $Core_4$ because the state of the internal scan chain in $Core_3$ cannot be controlled and observed in ExTest mode. Since we apply test stimuli and capture test responses through functional paths, all the interconnects are tested implicitly, and hence we do not need to perform ExTest for $Core_3$. It should be noted, however, this implicit testing of interconnects loses the diagnostic information that differentiates between defects in $Core_3$'s interconnects and defects in $Core_3$'s internal logic.

To summarize the above-explained observation, a core that does not require consecutive test vectors (e.g., targeting stuck-at fault), can be tested without wrapping its terminals as long as all its producers and consumers are 1500-wrapped. If the core does not have other test modes except InTest and ExTest mode, then it does not need a wrapper at all (Figure 6.2). If the core has other test modes, for example, it contains RAM or ROM blocks and it has an additional built-in self-test (BIST) mode to test these internal memories, then, it needs a light wrapper without WBRs to support these additional modes, i.e., the light wrapper must include WIR and the WSC port to control the operational mode of the core. From now onwards *light-wrapped cores* will refer to cores which do not need a wrapper at all or cores with a light wrapper, since both of them remove all the wrapper cells, which

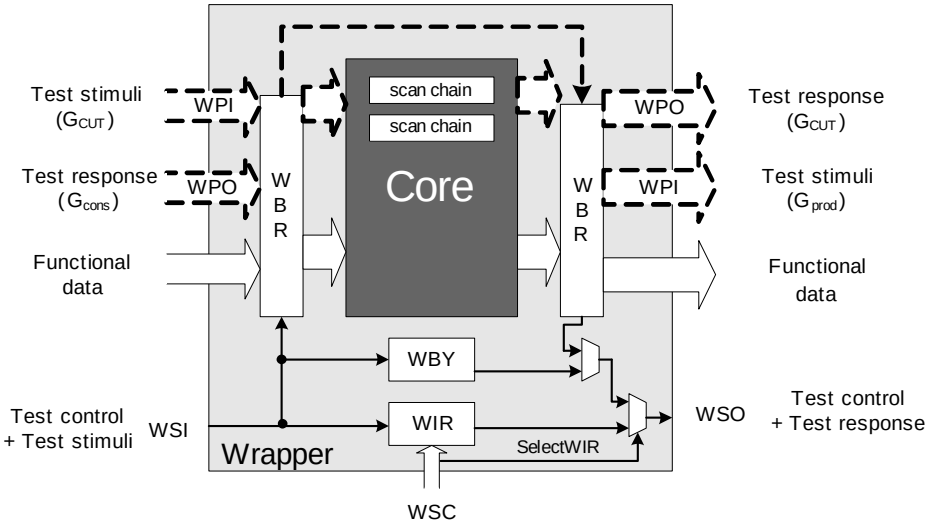


Figure 6.3: Revised IEEE 1500-Compliant Wrapper for Producer/Consumer Cores.

in turn reduce DFT area and may improve the SOC's functional performance. The light-wrapped core requires either WSI/WSO or WPI/WPO to shift in the test stimuli and shift out the test responses to and from its internal scan chains. It may also include a serial or parallel bypass register (WBY) to enable a shortened test access path to other cores, if necessary. It is interesting to note that, if the light-wrapped core does not have internal scan chains, it can be treated as a UDL and the proposed test strategy for it is in essence a standard ExTest strategy.

When the number of light-wrapped cores is large, to speed up the test data transfer, we propose to update producers and consumers with a 1500-compliant wrapper shown in Figure 6.3. In this revised wrapper architecture, test stimuli for the PIs of the CUT are loaded through the producers' WOCs only; while the test responses for the POs of the CUT are unloaded through its consumers' WICs only. In addition to the DFT hardware modification to support light-wrapped core testing, the IEEE 1500 instruction set also needs to be extended in this revised wrapper architecture for producer/consumer cores. New instructions *LoadProd* for producer cores and *UnloadCons* for consumer cores are introduced. Moreover, if a core serves as both producer and consumer at the same time, an additional *LoadUnloadNbrs* instruction is required to transfer test data both in and out of its WBR

cells. These instructions are used to set the producer/consumer in the appropriate operational mode to shift in/out test stimuli/responses. For the example shown in Figure 6.1, $Core_3$ has a light wrapper (Figure 6.2) and the core wrappers for its producers ($Core_1$ and $Core_2$) and its consumer ($Core_4$) can be revised to support the previously explained transfer mechanisms (see Figure 6.3 whose parameters are detailed in Section 6.2.2).

When the number of light-wrapped cores is relatively small, the amount of test data need to be transferred through the WBRs of producers and consumers is also comparably small. Hence, it is acceptable to use the ExTest configuration of the standard IEEE 1500 wrapper to load/unload the wrapper cells of the producers and consumers (i.e., TAMs go through all wrapper cells), and also there is no need to introduce extra test instructions. However, to enable parallel ExTest and to access the internal SFFs of the light-wrapped cores, we need to adapt the TestRail architecture to support testing light-wrapped cores, as detailed in Section 6.3.

6.1.2 Summary of Contributions

In this chapter, novel test architectures and the associated optimization techniques for SOCs containing light-wrapped cores are presented, which facilitate a rapid and concurrent test of 1500-wrapped cores and light-wrapped cores. The two main contributions of this chapter, detailed in Sections 6.2 and 6.3, are as follows:

- we propose a novel $Producer - CUT - Consumer$ test architecture for SOC design containing a large number of light-wrapped cores and its associated optimization algorithms. By dividing TAMs into three separate groups (for producers, CUTs and consumers, respectively), implicit test conflicts are eliminated and thus the negative impact on testing time because of test resource sharing is minimized. In addition, based on the functional relationship among embedded cores, an algorithm that maximizes the number of light-wrapped cores is also presented;

- for SOC designs containing a relatively small number of pre-determined light-wrapped cores, we show how to adapt the TestRail architecture to access the internal SFFs of light-wrapped cores in the Parallel ExTest mode, and how to optimize it in terms of test application time.

6.2 Producer-CUT-Consumer Architecture for Testing Light-Wrapped Cores

Having introduced the light wrapper concept and outlined its applicability to 1500-based testing, this section focuses on its implications on SOC test architecture and test scheduling when system integrators want to maximize the number of light-wrapped cores. Note, in this scenario, the test architecture and test scheduling algorithm for SOCs containing light-wrapped cores are similar to the ones proposed in Chapter 5 for two-pattern tested cores because both of them use the revised version of wrapper for test data transfer through producers (consumers). For the sake of self-containment in this chapter, however, we still elaborate all the details here.

To clarify all the issues related to testing these light-wrapped cores, we provide a hypothetical SOC, called m4953, with 9 cores and a system bus connecting 3 cores. The number of scan chains n_{sc} and the functional interconnects of these cores are shown in Figure 6.4[1]. Note, the test infrastructure of the SOC has not been implemented yet and hence it is not shown in the figure. Additional test parameters will be given in the experimental section. The name of this SOC follows the benchmark naming convention presented in [116], where *m* refers to McMaster University and the number 4953 denotes its test complexity.

6.2.1 Test Conflicts Caused by Sharing Producers/Consumers

Before proposing a new SOC test architecture, we analyze the conflicts introduced by the inter-operability of light-wrapped cores and their 1500-wrapped producers and consumers. In the InTest mode, all the 1500-wrapped cores can be tested concurrently, as long as they use different TAM lines (assuming cores on the same TAM are tested in sequential order

[1] to make the drawing clear, the cores are not placed in the increasing numerical order.

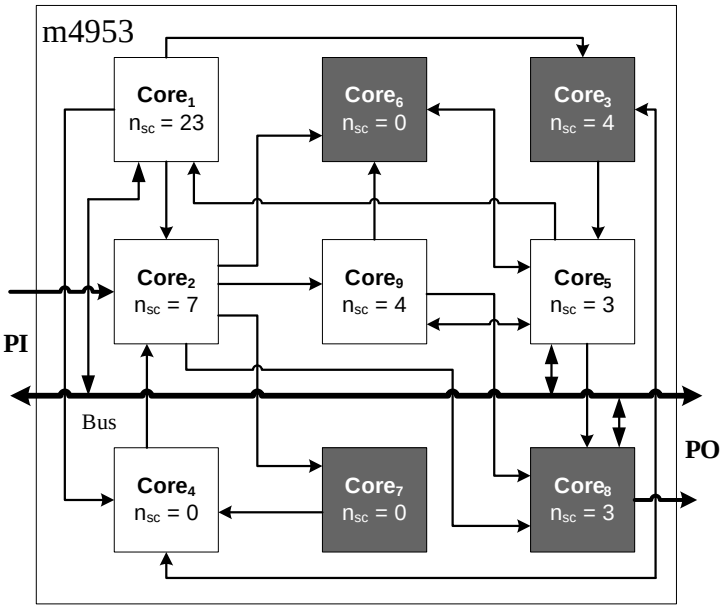


Figure 6.4: Example SOC: m4953.

and there is no other test scheduling constraints, such as precedence relationship, preemption and power constraints). However, because testing light-wrapped cores is dependent on their producers and consumers, TAM lines conflicts are not the only ones which limit the test concurrency. Instead, there are five new types of test conflicts, described as follows:

- *Producer-CUT Conflict*: Producer(s) and the CUT **cannot** be tested at the same time. For example, in Figure 6.4, if $Core_6$ is a light-wrapped core, $Core_2$, $Core_5$ and $Core_9$ should not be tested at the same time as $Core_6$. This is because, the producer needs to utilize its WOCs to capture its test responses, however, at the same time, the CUT needs the producer's WOCs to provide test stimuli. If they are tested concurrently, the test data will be corrupted.

- *CUT-Consumer Conflict*: The CUT and consumer(s) **cannot** be tested at the same time. For example, in Figure 6.4, if $Core_2$ is a light-wrapped core, $Core_6$, $Core_7$, $Core_8$ and $Core_9$ should not be tested at the same time. This is because, the consumer needs to utilize its WICs to deliver test stimuli, however, at the same time, the CUT needs the consumer's WICs to capture the test responses. If they are tested

concurrently, the test data will be corrupted.

- *Shared-Producer Conflict*: Two light-wrapped cores which connect directly (i.e., on a dedicated non-shared set of lines) to the same producer **cannot** be tested at the same time. For example, in Figure 6.4, if $Core_7$ and $Core_8$ are light-wrapped cores, they cannot be tested at the same time because both of them require the WOCs of $Core_2$ to provide the test stimuli.

- *Shared-Consumer Conflict*: Two light-wrapped cores which connect directly to the same consumer **cannot** be tested at the same time. For example, in Figure 6.4, if $Core_3$ and $Core_6$ are light-wrapped cores, they cannot be tested at the same time because both of them need the WICs of $Core_5$ to capture the test responses.

- *Shared-Bus Conflict*: If the producer(s) or consumer(s) connect to the light-wrapped CUT through functional buses, they might imply the previous described test conflicts and hence **may not** be tested at the same time. For example, in m4953, if $Core_1$ and $Core_5$ are two light-wrapped cores connected to the system bus, they cannot be tested at the same time because both of them need the I/O WBRs of $Core_8$ to provide test stimuli or capture the test responses at the same time. However, if we have another wrapped core connected to the bus, for example $Core_4$, then $Core_1$ and $Core_5$ can be tested together because we can use $Core_4$ as the producer and consumer of $Core_1$, and $Core_8$ as the producer and consumer of $Core_5$. By sharing the bus lines in consecutive times, there is only one clock cycle test application penalty per test pattern (using the same system bus to transfer test data), which is insignificant for scan-based testing.

6.2.2 TAM Division into Three Groups: Producer, CUT and Consumer

The previous section has outlined the test conflicts which, if not taken into consideration, may corrupt the test data and render the test useless. Other types of conflicts may appear if the test data is transferred using shared TAM lines between producers, CUT and consumers. To avoid this type of conflicts, which may adversely influence the overall testing time of

the SOC, dividing the TAM lines into three groups is proposed, motivated by the following examples:

Example 6.1 *Consider the SOC m4953 shown in Figure 6.4 and let us assume that* $Core_1$ *and* $Core_2$ *are 1500-wrapped cores and* $Core_6$ *is a light-wrapped core, which needs* $Core_2$ *as a producer. We assume that* $Core_1$ *and* $Core_2$ *share the same TAM lines (*TAM_{w1}*) and* $Core_6$ *connects to a different TAM (*TAM_{w2}*). Since for testing* $Core_6$ *we need to use both* TAM_{w1} *and* TAM_{w2} *to transfer test data, loading a test pattern for* $Core_1$ *is prohibited while loading the stimulus for* $Core_6$*. As a result, there is a test conflict between* $Core_1$ *and* $Core_6$ *even though they connect to different TAM lines and have no functional relationship. This* indirect *TAM resource conflict may prohibit the overall test concurrency for light-wrapped cores in a large SOC, which will ultimately lead to testing all the light-wrapped cores separately, and thus resulting in very large testing time.*

Sharing TAM lines between producers, CUTs and consumers, may also increase the test control complexity for Test Bus architecture, as illustrated in the following example.

Example 6.2 *In the case of m4953 shown in Figure 6.4, if* $Core_2$ *is a light-wrapped core, then after the test stimuli are loaded in the WOCs of* $Core_1$ *and* $Core_4$*, we must apply them at the same time. We also need to capture the test responses in the WICs of* $Core_6$*,* $Core_7$*,* $Core_8$ *and* $Core_9$ *at the same time before shifting it out. If the TAM lines are shared between producers, CUTs and consumers, all of these operations introduce additional synchronization issues and consequently they may increase not only the testing time, but also the test control complexity.*

To address the above problems, we propose to divide the TAM lines into three groups: G_{prod}, G_{CUT} and G_{cons} used to load the producers, CUTs and consumers, respectively. This division will remove the additional test conflicts discussed in Example 6.1 and test control complexity discussed in Example 6.2. Using the setup from Example 6.1, testing $Core_6$ will need the assistance of $Core_2$ to provide the test stimuli. If the output WBRs of $Core_2$ is loaded through G_{prod} and, although $Core_1$ and $Core_2$ share the same TAM resources in G_{CUT}, then $Core_1$ can still be tested at the same time as $Core_6$.

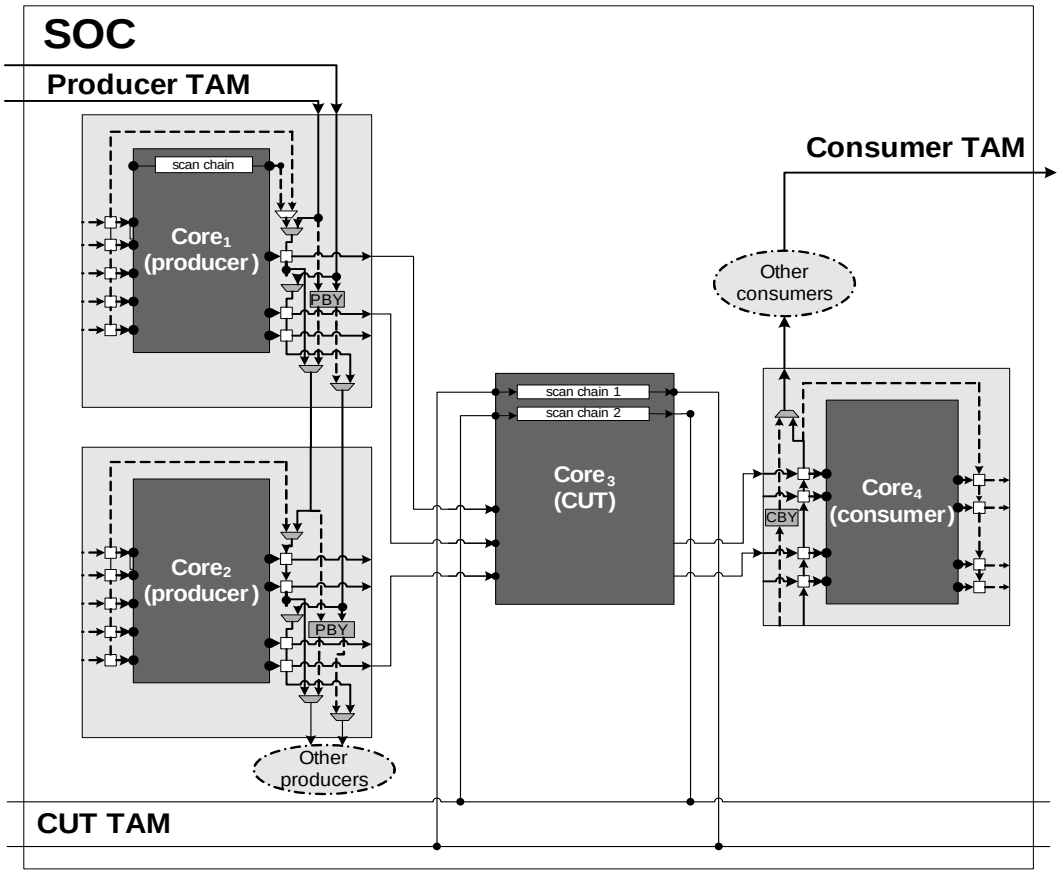


Figure 6.5: Proposed Test Architecture for an Example SOC Containing Light-Wrapped Cores.

For G_{CUT} group we use a flexible-width Test Bus architecture, as introduced in Section 3.3.3. For G_{prod} and G_{cons} group, however, we use the Daisychain architecture [2], i.e., long scan chains are constructed over all the producer cores' output terminals and all the consumer cores' input terminals, as depicted in Figure 6.5. Producer bypass registers and consumer bypass registers (PBY and CBY in the Figure) are introduced in order to shorten the loading/unloading time because only a few cores serve as producers or consumers at a specific test session. The main reason for using the Daisychain architecture for G_{prod} and G_{cons} group is to simplify the control complexity. When a producer (consumer) core is in LoadProd (UnloadCons) mode, the producer (consumer) TAM lines go through the

core's wrapper boundary cells, otherwise they go through the bypass register (note, it is unnecessary to introduce extra bypass instruction for producers and consumers). As a result, although testing light-wrapped cores involves several producers and consumers, they can be controlled by the LoadProd and UnloadCons instructions independently. In addition, the Daisychain architecture for G_{prod} (G_{cons}) TAM groups can almost always give a near optimal loading (unloading) time, for a given TAM width W_{prod} (W_{cons}). Suppose the number of the outputs of a producer is N_o, then its loading time will be $\lceil \frac{N_o}{W_{prod}} \rceil$. As long as $W_{prod} \leq N_o$ (which is realistic in most of the cases), there is no waste for G_{prod} TAM resources, except the few bypass cycles, which leads to a near optimal loading time for its producers. The same holds for G_{cons} unloading. It is essential to note that the TAT of a light-wrapped core is dependent on all the three TAM groups' architectures and the proposed TAM division into three groups facilitates concurrent testing of 1500-wrapped and light-wrapped cores, which is exploited by the algorithms described in the following section.

6.2.3 Proposed Algorithms for Wrapper/TAM Co-Optimization

The introduction of light-wrapped cores, producers, consumers and TAM division into three groups, requires the development of new algorithms for wrapper/TAM co-optimization, as explained in this section. We formulate the new problem to be solved as follows.

Problem $P_{LWT-opt}$: Given the test set parameters for each core (including the number of primary inputs, primary outputs, bidirectional I/Os, test patterns and scan chains, and each scan chain length), the total TAM width W_{ttl} for the SOC and the wrapper design constraints C_w, determine the width of each TAM group (W_{prod}, W_{CUT} and W_{cons} corresponding to G_{prod}, G_{CUT} and G_{cons}), the TAM width and the wrapper design for each core, and a test schedule for the entire SOC such that: (i) the wrapper design constraints C_w are satisfied; (ii) the total number of light-wrapped cores is maximized; (iii) the total number of TAM lines used at any time does not exceed W; and (iv) the overall SOC TAT is minimized.

There are mainly three types of wrapper design constraints C_w: (i) if the critical paths appear between cores then, to avoid performance penalty, some cores must be light-wrapped; (ii) if some of the cores are provided with IEEE 1500 wrappers and, due to their location

and size, their overhead does not affect the performance or the cost of the SOC, then there is no reason to make them light-wrapped. (iii) if a core is two-pattern tested (e.g., targeting delay faults or CMOS stuck-open faults) as discussed in Chapter 5, which employs the producers' WOCs to apply the second consecutive pattern, double-buffering in the WBRs of the core and its producers is necessary; hence, in this case, the CUT and all its producers must be 1500-wrapped. It is important to note, if there are user-defined logic (UDL) blocks in the SOC, the system integrator has two choices: either make the UDL blocks 1500-wrapped, and then use them as an input to the algorithms described in this section for problem $P_{LWT-opt}$; or treat the UDL blocks as light-wrapped cores, i.e., they must satisfy the first wrapper design constraint when solving $P_{LWT-opt}$. In either case there is no loss in fault coverage of UDLs since, by construction, it is ensured that each light-wrapped core is controlled by its producers and observed by its consumers. Therefore, the proposed solution can also be used as an alternative to ExTest for *concurrently* testing wrapped cores and UDLs.

In the rest of this section we first present the top level algorithm for solving $P_{LWT-opt}$ and then we give details on the new procedures and concepts specific to our approach.

TAM Division And Test Scheduling: *LightTest_Optimization*, the proposed algorithm to solve $P_{LWT-opt}$ is shown in Figure 6.6. The inputs are the set of cores (C_{set}), TAM width (W_{ttl}), functional interconnect relationship between cores (R), wrapper design constraint (C_w), and a weight parameter (*weight*), used in pruning the search space. The outputs are the number of TAM lines allocated to each TAM group, wrapper type ($wrapper\_type$) and design for each core, SOC test schedule (*schedule*) and the overall test application time for the entire SOC (T_{soc}). The optimal TAM division, i.e., the combination of W_{prod}, W_{CUT} and W_{cons} that gives the minimum TAT of the SOC, is acquired through enumeration. The enumerative algorithm begins by determining which cores must be light-wrapped (line 1), according to the functional interconnect relationship R and pre-defined wrapper design constraint (C_w) of the SOC. Based on the generated wrapper type (light-wrapped or not) for each core and the test conflicts determined by functional interconnect relationship among cores (R), a test incompatibility graph (TIG) is created (line 2). Next, the algorithm will enumeratively find the optimal TAM division and the minimum system TAT T_{soc}. In the inner loop (lines 5 to 9), the local minimum TAT *localmin* for a fixed total

Algorithm 6.1 - LightTest_Optimization

INPUT: C_{set}, R, W_{ttl}, C_w, *weight*
OUTPUT: W_{prod}, W_{CUT}, W_{cons}, *wrapper_type*, *schedule*, T_{soc}

1. $wrapper\_type = Decide\_Wrapper\_Type(C_{set}, R, C_w)$;
2. $TIG = Construct\_TIG(wrapper\_type, R)$;
3. For W_{CUT} from $W_{ttl} - 2$ downto 1 {
4. $W_{prod\_plus\_cons} = W_{ttl} - W_{CUT}$;
5. **For** W_{prod} from 1 to $W_{prod\_plus\_cons} - 1$ {
6. $W_{cons} = W_{prod\_plus\_cons} - W_{prod}$;
7. $testing\_time = LightTest\_Schedule(C_{set}, TIG, W_{prod}, W_{CUT}, W_{cons})$;
8. *localmin* = min{all *testing_time*};
9. Record $W_{prod\_localmin}$, $W_{cons\_localmin}$;
. }
10. **if** ($localmin > weight \times globalmin$) {
11. **break**; } /*Prune search space*/
12. *globalmin* = min{all *localmin*};
13. Record $W_{prod\_globalmin}$, $W_{cons\_globalmin}$;
. }
14. $W_{prod} = W_{prod\_globalmin}$;
. $W_{cons} = W_{cons\_globalmin}$;
. $W_{CUT} = W_{ttl} - W_{prod} - W_{cons}$;
. $T_{soc} = globalmin$;
15. **return** W_{prod}, W_{CUT}, W_{cons}, *wrapper_type*, *schedule*, T_{soc};

Figure 6.6: Pseudocode for Optimizing Producer-CUT-Consumer Test Architecture.

width of $W_{prod} + W_{cons}$ ($W_{prod\_plus\_cons}$) is computed. In the outer loop (lines 3 to 13) the algorithm searches for *globalmin*, among the *localmin* values, by enumerating W_{CUT} from the maximum possible value $W - 2$ to 1. During our initial experiments it was observed that *localmin* is nearly a *convex* function with respect to W_{CUT}. That is, it keeps decreasing until it reaches a local minimum value, at which point it starts increasing. This convex attribute can be explained by the fact that when W_{CUT} has a small value, the TAT is dominated by the time to transfer test data through G_{CUT} (for justification see Equation 6.1 explained later in this section). Increasing W_{CUT}, and hence decreasing W_{prod} + W_{cons}, will finally break

this bottleneck. The TAT starts to increase when the time required to load/unload the producer/consumer output/WICs starts to dominate the scan time for G_{CUT}. There are some variations around the local minimum value, which can be justified by the heuristic nature of the dynamic rectangle packing (explained later in this section). Hence, to prune the search space we enumerate the *localmin* values in the opposite direction (i.e, from $W-2$ to 1), since we want to discard the large *localmin* values. To accommodate the variations around the minimum value we use a parameter *weight* (a real value slightly greater than 1) (lines 10 and 11). It should be noted that during the enumeration process, we do not need to do TAM design for G_{prod} and G_{cons} groups, since the implementation of the Daisychain architectures for these two groups is straightforward once W_{prod} and W_{cons} are determined. To generate a TAM design for G_{CUT}, we adapt an existing generalized rectangle packing algorithm *TAM_Schedule_Optimizer* [76, 78]. Due to the usage of the Daisychain architecture for producers/consumers, a *dynamic adaptation* of this existing algorithms is necessary. We elaborate on each of the main steps of the top-level algorithm in the following paragraphs.

The worst case complexity of algorithm *LightTest_Optimization* is $O(W_{ttl}^2 \times C(LTS))$, where $C(LTS)$ is the worst case complexity of algorithm *LightTest_Schedule*, which will be detailed at the end of this section.

Decide Wrapper Type: Not all the cores need to be 1500-wrapped in an SOC, however, to provide full controllability and observability, each light-wrapped core needs to be surrounded by 1500-wrapped cores, i.e., all its producers and consumers must be wrapped. The pseudocode for deciding the wrapper type is shown in Figure 6.7. The algorithm takes the set of cores C_{set}, the functional interconnect relationship R and the wrapper design constraints C_w as the inputs, and it outputs the wrapper type for each core $i \in C_{set}$. First, the cores which need to be wrapped by 1500-compliant wrappers according to direct functional relationship (i.e., dedicated non-shared communication lines) are identified (lines 1 to 10). In the first loop (lines 1 to 6), we initialize the wrapper status and wrap the cores according to wrapper constraints, if any. For all the other cores, the wrapper is first set to a light-wrapped type and a variable called *test_dependency* is initialized to the sum of its unwrapped producers and consumers (note, if one core serves as both a producer and a consumer for another core, it is not to be counted twice). This variable is used to indicate core's test requirements as a light-wrapped core; if this number is large, it means that when

Algorithm 6.2 - Decide_Wrapper_Type

INPUT: C_{set}, R, C_w
OUTPUT: *wrapper_type*

```
/* According to direct functional interconnects */
1. For each Core i ∈ C_set {
2.    if (C_w exist) {
3.       Wrap core i according to its constraint
4.    } else {
5.       Set Is_Light_Wrapped_i = true;
6.       Initialize test_dependency_i;
.     }
. }
7. While (test_dependency_i != 0 for any core i ∈ C_set) {
8.    Find Core j with the maximum test_dependency;
9.    Set Is_Light_Wrapped_j = false; test_dependency_j = 0;
10.   Update test_dependency of its producers and consumers;
.  }
/* According to functional bus interconnects */
11. For each functional bus
12.   if (No core on the bus is wrapped)
13.      Wrap the core with the least number of I/Os;
14. return wrapper_type for each core;
```

Figure 6.7: Procedure for Deciding the Wrapper Type of Each Core.

this core is light-wrapped, we need a large number of 1500-wrapped neighbor cores to test it. For example, in the case of m4953, $Core_2$ has 2 producers ($Core_1$ and $Core_4$), and 4 consumers ($Core_6$, $Core_7$, $Core_8$ and $Core_9$), hence its *test_dependency* is initialized to 6. If $Core_2$ is a light-wrapped core, we need to wrap all its 6 neighbors. As a result, it is better to wrap $Core_2$ with a 1500-compliant wrapper. Therefore, the algorithm finds the cores with a large *test_dependency* and wraps them as 1500-compliant (lines 8 and 9). Whenever a core is decided to be wrapped as 1500-compliant, its *test_dependency* is set to 0 because it does not require any other cores to facilitate its test; the *test_dependency* of all its light-wrapped producers/consumers is deducted by 1 (line 10). When functional busses are used, at least one core on each functional bus must be wrapped as 1500-compliant to test all the other

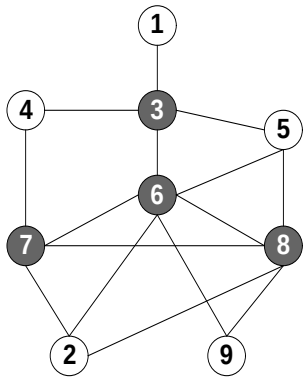


Figure 6.8: Test Incompatibility Graph for SOC m4953.

light-wrapped cores on the bus (lines 11 to 13). The algorithm will find a core with the least number of inputs and outputs to wrap, in order to decrease the time required to load/unload the test stimuli/responses. To illustrate the outcome of the proposed algorithm, in the case of m4953, $Core_3$, $Core_6$, $Core_7$ and $Core_8$ are selected to be light-wrapped, as shown by the shaded boxes in Figure 6.4.

Construct the Test Incompatibility Graph (TIG): If there are test conflicts between two cores (see Section 6.2.1), then these two core tests cannot be scheduled at the same time and they are denoted as *incompatible* cores. We construct a test incompatibility graph TIG by treating each core as a node and adding an edge between two nodes if they are incompatible. This TIG is used in Algorithm 6.3 (*LightTest_Schedule*). The TIG generated for m4593 is shown in Figure 6.8. As shown in the figure, edges illustrating incompatibility can exist only between two light-wrapped cores or between a light-wrapped core and its producers/consumers. Two 1500-wrapped cores are always compatible during test because they do not need each other's help to test their internal logic.

***Dynamic* Rectangle Representation:** For a one-pattern tested 1500-wrapped core, if the assigned TAM width is given, the TAT to apply the entire test set is determined by Equations 4.1 (for single-frequency core) and 4.2 (for multi-frequency core). When a specific wrapper optimization method is used, the core testing time is a fixed value and hence the core test can be represented as a *static* rectangle, in which the height represents the TAM width and the width stands for the testing time. However, for a light-wrapped core,

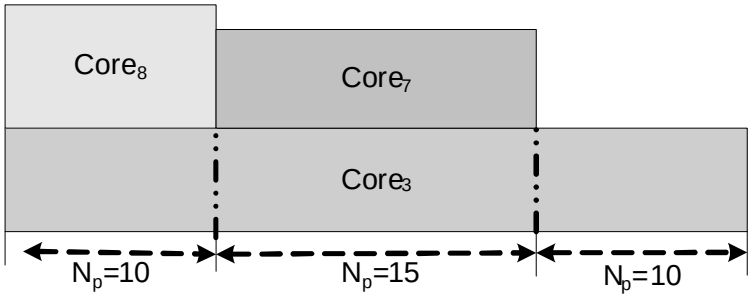


Figure 6.9: Test Application Time for Light-Wrapped Cores.

its TAT T_l does not only depend on the time to load/unload its own producers (L_{prod}), consumers (L_{cons}) and internal scan chains (L_{in}); *the TAT also depends on the time necessary to load/unload all the concurrently-tested light-wrapped cores' producers/consumers*. To keep the control and computational complexity low, similar to the test strategy for two-pattern tested cores discussed in Chapter 5, we propose to *align* test patterns for all the concurrently-tested light-wrapped cores and hence T_l is calculated as in Equation 6.1, where bypass cycles are ignored.

$$T_l = \sum_s (1 + max\{\sum L_{prod}, \sum L_{cons}, max\{L_{in}\}\}) \times (p_s + 1) \tag{6.1}$$

As a result, if for a given light-wrapped core the test schedule changes s times, then for each subset of patterns p_s (for the s distinct divisions of the time allocated to the given core) the TAT will be computed based on the light-wrapped cores scheduled in each of these s divisions. The following example is used to better illustrate the computation of T_l.

Example 6.3 *In the case of m4953, $Core_3$ is compatible with light-wrapped cores $Core_7$ and $Core_8$. Let us assume $Core_7$ and $Core_8$ are selected to be scheduled at the same test time with $Core_3$, as shown in Figure 6.9 (the given number of test patterns for these three cores has been selected only to illustrate this example). The time necessary to apply a pattern for $Core_3$ is updated each time the schedule changes. For the first 10 patterns, the shifting time for each pattern of both $Core_3$ and $Core_8$ will be $max\{L_{prod\_3} + L_{prod\_8}, L_{cons\_3} + L_{cons\_8}, L_{in\_3}, L_{in\_8}\}$. However, for the next 15 patterns, once the test for $Core_8$ has been completed and $Core_7$ is scheduled concurrently with $Core_3$, the shifting time for each pattern will be $max\{L_{prod\_3} + L_{prod\_7}, L_{cons\_3} + L_{cons\_7}, L_{in\_3}, L_{in\_7}\}$. The same reasoning is*

applied for the last 10 patterns when $Core_3$ is not concurrent with any other light-wrapped cores. The shifting time for each pattern will be $max\{L_{prod\_3}, L_{cons\_3}, L_{in\_3}\}$. The variations in the shifting time for each of the 3 divisions of the schedule for $Core_3$ can be differentiated using a $loadSize = max\{\sum L_{prod}, \sum L_{cons}, max\{L_{in}\}\}$, determined by all the concurrently-tested light wrapped cores.

From the above discussion, we can see that the T_l for a light-wrapped core changes *every time* when the schedule is updated. This *dynamic* attribute leads to a *dynamic rectangle representation* of the light wrapped core's test and hence a dynamic rectangle packing algorithm for test scheduling, shown in the following.

Adapted Dynamic Rectangle Packing: To concurrently test the light-wrapped cores and 1500-wrapped cores, for the CUT TAM group we use an *LightTest_Schedule* algorithm (the pseudocode is shown in Figure 6.10). The algorithm takes the core list C_{set}, TIG and the TAM division as inputs, and it generates the *schedule* for each core and the overall TAT of the SOC.

As described earlier, for light-wrapped cores the test **cannot** be pre-computed and represented as a static rectangle; its TAT (the width of the rectangle) varies with its schedule, hence its rectangle representation is computed dynamically (lines 7 and 15). In addition, since the TAT of the light-wrapped cores may change dynamically with its schedule, there are no "preferred TAM widths" for them. In [76] the procedure *Initialize* (line 2) was used to compute the preferred width for each core, in which parameters d and p were sometimes manually selected for SOCs with different available TAM widths to get a better result; since we need to call this procedure many times with different W_{CUT} (see Algorithm 6.1), it is unlikely that a manual selection will lead to an optimal value. Consequently, in our implementation we have fixed the two parameters to $d = 2$ and $p = 1.0$ (these two values give a generally good "preferred TAM width"); this may result in a different schedule and a slightly longer TAT in some cases when compared to the result in [76]. Line 3 initializes the cores that have not finished their schedule $C_{unfinished}$, the current available TAM width w_{avail}, and the current start time for unscheduled cores $this\_time$, respectively. In line 9, the algorithm tries to schedule either a 1500-wrapped core with preferred TAM width or a light-wrapped core with the maximum allowable test pattern count that is able to fit in the

Algorithm 6.3 - LightTest_Schedule

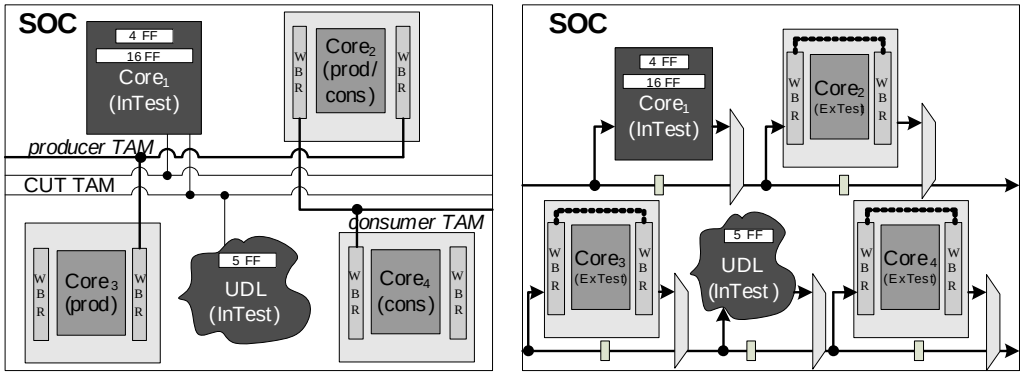
INPUT: C_{set}, TIG, W_{prod}, W_{CUT}, W_{cons}
OUTPUT: *schedule*, *testing_time*

1. Compute collection R_p of rectangles for 1500-compliant core set C_p;
2. Initialize(C_p, d, p); /*C_p is the 1500-compliant core set*/
3. Set $C_{unfinished} = C_{set}$; $w\_avail = W_{CUT}$; $this\_time = 0$; (see [76, 78])
4. **While** $C_{unfinished} \neq \emptyset$ {
5. **if** $w\_avail > 0$ {
6. Find unscheduled light-wrapped core set C'_d;
7. Compute collection R'_d of dynamic rectangles for C'_d;
8. Initialize(C'_d, d, p);
9. Schedule compatible 1500-wrapped cores that can be assigned
. preferred TAM width or compatible light-wrapped cores; (see [76, 78])
10. Schedule compatible 1500-wrapped cores that can use the resulting
. idle TAM wires; (see [76, 78])
11. Update L_{prod}, L_{cons}, $loadSize$;
12. Update $test\_time$ for scheduling light-wrapped cores;
13. } **else** {
14. Update $next\_time$;
15. Find unscheduled light-wrapped cores $C"_d$ with
. **no** internal scan chains;
16. Compute collection $R"_d$ of dynamic rectangles for $C"_d$
17. Initialize($C"_d$, d, p);
18. Schedule compatible cores in $C"$ not exceeding $next\_time$;
19. Update L_{prod}, L_{cons}, $loadSize$;
20. Update $test\_time$ for scheduling light-wrapped cores;
21. Update $next\_time$;
22. Finish the scheduling core test C_i with ending time $next\_time$;
23. Update $this\_time$;
24. $w\_avail += w_{tam\_C_i}$;
25. $C_{unfinished} -= \{C_i\}$;
26. Update $this\_time$;
. }
. }
27. **return** *schedule*, *testing_time*;

Figure 6.10: Procedure for Test Scheduling with Given Widths of Each TAM Group.

idle rectangle (since test patterns for concurrently-tested light-wrapped cores are aligned). When scheduling a light-wrapped core, its rectangle size is determined as following. The TAM width for this core (its height) is the minimum value that minimizes *loadSize* and the TAT of the core (its width) is calculated using Equation 6.1. Once a core is scheduled (lines 9 and 10), the available CUT TAM width *w_avail* will be deducted the value of the assigned CUT TAM width for the core. Whenever a light-wrapped core is scheduled, L_{prod}, L_{cons}, *loadSize*, for the currently scheduled light-wrapped cores, need to be updated (lines 11 and 19) and the TAT will be recalculated (lines 12 and 20). If a light-wrapped core has no internal scan chains inside and hence it does not need any TAM lines in the G_{CUT} group, we may be able to schedule it even when the available CUT TAM width $w\_avail = 0$ in G_{CUT} (lines 15 to 20). This is because only G_{prod} and G_{cons} resources are necessary. Once there is no core able to be tested starting with *this_time*, the currently scheduled core with the minimum TAT will be finished (line 24), its TAM resources are released and *this_time* advances to its finishing time; the algorithm tries to schedule another core with this TAM resources. Note, due to test conflicts, we are only able to select a compatible core to be scheduled at any time (lines 9, 10 and 18); this is done through checking whether there is an edge in TIG between the cores currently under test and the to-be-scheduled core.

The worst case complexity $C(LTS)$ of algorithm $LightTest\_Schedule$ can be estimated as follows. The while loop in Line 4 of Figure 6.10 is executed N_c times, where N_c is the number of cores of the SOC. In each such execution, a linear search in the $O(N_c)$ core set is used to find the next core to be scheduled. In addition, these cores are also examined $O(N_l)$ to determine whether they are compatible with the currently-scheduled light-wrapped cores (lines 9, 10 and 15). Moreover, in each such execution, a collection of $O(W_{CUT})$ rectangles are generated for $O(N_l)$ light-wrapped cores. The complexity of rectangle generation using *Design_wrapper* is $O(sc \log sc + sc \cdot k)$, in which *sc* is the number of scan chains in the light-wrapped core and *k* is the TAM width [73]. As a result, the worst case complexity $C(LTS)$ is $O(N_c^2 \cdot N_l^2 \cdot W_{CUT} \cdot (sc \log sc + sc \cdot W_{CUT}))$.



(a) Producer-CUT-Consumer architecture (b) TestRail architecture with NO added control

Figure 6.11: Comparison of Test Architectures for SOCs with Light-Wrapped Cores - A Simple Example.

6.3 Adapting TestRail Architecture for Testing Light-Wrapped Cores

One of the design aims in the previous section is to maximize the number of light-wrapped cores. This, however, can lead to very long test application time, which, in some situations, may not be the preferred option for the system integrator. One may argue that from the test reuse standpoint, as long as the existence of the IEEE 1500 wrappers does not violate the design constraints, it is better to employ them, regardless of the overhead that they may incur. Hence, a practical situation is that the system integrator analyzes the design requirements and SOC constraints and determines the wrapper type, for each embedded IP core, on a case by case basis.

Since the TestRail architecture (discussed in Section 3.2.4) supports loading cores on individual TestRails both sequentially and concurrently, it automatically supports the access of a light-wrapped core's producers and consumers. Therefore, no dedicated TAM resources are necessary for shifting in/out producers' WOCs and consumers' WICs, as it is the case in Section 6.2. For the TestRail architecture, when a light-wrapped core is under test, we simply set its producers and consumers in the ExTest mode, and set itself in the InTest mode if it is a scanned core, as shown for an hypothetical SOC in Figure 6.11(b) . No new test modes, and hence no additional test commands, need to be introduced in the

Algorithm 6.4 - LightTRDesign

INPUT: C_{set}, R, W_{ttl}
OUTPUT: TR, T_{soc}

1. **for**$(i = 0; i < loopCnt; i++)$ {
2. *BuildCostFunction*;
3. *DesignTestRail*;
4. *ScheduleLightCores*;
5. *RescheduleInRail*;
6. *RescheduleBetweenRails*;
7. record the TestRail architecture TR with lowest T_{soc};
. }
8. **return** TR, T_{soc};

Figure 6.12: Pseudocode for Optimizing TestRail Architecture with Light-Wrapped Cores.

wrapper instruction set. When using the TestRail architecture both the WICs and WOCs of the producers and consumers are loaded through the TestRails used by the wrapped cores. Therefore we do not need to differentiate them and, in this section, they are both regarded as the *test partners* of the light-wrapped cores. Because the light-wrapped cores and their test partners might be placed in different TestRails, separate TestRails cannot operate independently any more. This necessitates a new optimization algorithm, described in this section. The problem of minimizing test application time of the TestRail architecture for SOCs with light-wrapped cores, called $P_{lightTR-opt}$, can be formulated as follows:

Problem $P_{lightTR-opt}$: Given the test set parameters for each core and UDL (including the number of primary inputs, primary outputs, bidirectional I/Os, number of test patterns, number of scan chains, and scan chain lengths), the wrapper property of each core and UDL (light-wrapped or 1500-wrapped), the total TAM width W_{ttl} for the SOC, determine the set of TestRails R, the width $w(r)$ of each TestRail r, the wrapper design for each core, and a test schedule for the entire SOC such that: (i) Every core or UDL is assigned to not more than one TestRail; (ii)$\sum_{r \in R} w(r) \leq W_{ttl}$; (iii) the overall SOC test application time T_{soc} is minimized.

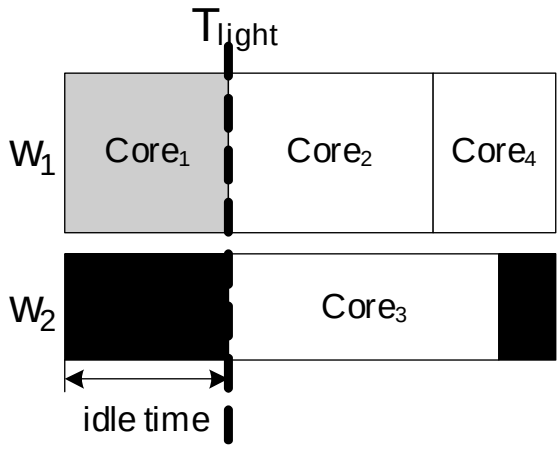
The total test application time T_{soc} for the TestRail architecture is the maximum of the

test application times of all the individual TestRails. The 1500-wrapped cores connected to a TestRail r are assumed to be tested sequentially, i.e., while a core is tested, all other cores connected to the same TestRail are bypassed. The light-wrapped cores can be either tested sequentially or concurrently, depending on which alternative saves test application time. It is important to note that light-wrapped cores that do not have internal scan chains, do not need to be assigned to any TestRails because its test stimuli/responses can be fully controlled/observed from its test partners. The test application time of a light-wrapped core cannot be determined until all its test partners (which are 1500-wrapped cores) have already been assigned to dedicated TestRails. To decrease the complexity of the $P_{lightTR-opt}$ problem, we first separate the schedules for 1500-wrapped cores and light-wrapped cores and then merge them at a later stage of the algorithm.

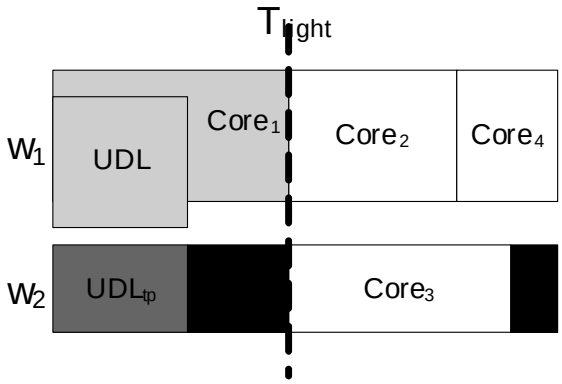
The proposed top-level algorithm *LightTRDesign* (shown in Figure 6.3) is a loop procedure with an upper limit on the exploration time decided by the predefined value (*loopCnt*). It takes the SOC core set C_{set} and the total TAM width W_{ttl} as inputs, and it outputs the set of TestRails TR and the overall test application time T_{soc}. *LightTRDesign* has four main steps (lines 3-6). For each of the iterations, the cost function is different which implies that we explore *loopCnt* different TestRail architectures and select the one which leads to the lowest test application time. This is important because the initial test architecture determines the effectiveness of the subsequent optimization procedures. In the results reported in this paper $loopCnt = 500$, which *gives a good balance in between the quality of the results and CPU execution times that are in the seconds range*. Step *DesignTestRail* determines a set of TestRails and their widths, based on the cost function generated from *BuildCostFunction*. Step *ScheduleLightCores* schedules the light-wrapped cores in front of 1500-wrapped cores for the previously determined TestRail architecture. The last two steps (*RescheduleInRail* and *RescheduleBetweenRails*) try to optimize the overall test application time for the SOC by exploiting the idle times within TestRails. In the following we illustratively show the specific features of our overall algorithm and the main steps.

Test Conflicts: Because testing the internal logic of the light-wrapped cores uses the 1500-wrapped test partners, three types of test conflicts are introduced:

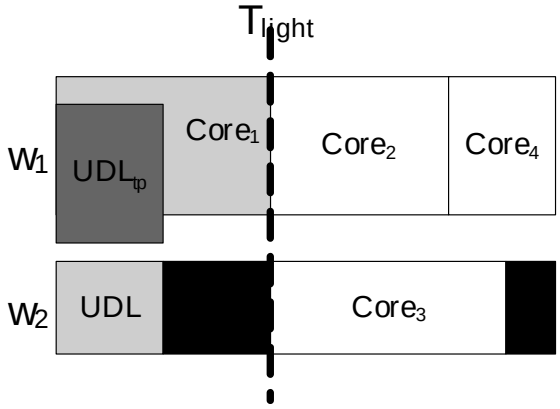
- *Partner-CUT Conflict*: The light-wrapped CUT and its test partners **cannot** be tested at the same time. This is because, both of them need to utilize the WBRs of the test



(a) UDL to-be-Scheduled



(b) UDL scheduled on TestRail 1



(c) UDL scheduled on TestRail 2

Figure 6.13: Comparison of Scheduling UDLs on different TestRails.

partner to shift in/out test stimuli/responses.

- *Shared-Partner Conflict*: Two light-wrapped cores which connect directly (i.e., on a dedicated non-shared set of lines) to the same test partner **cannot** be tested at the same time. This is because, as in the above case, sharing of the partner's WBR for test data transfer is prohibited.
- *Shared-Bus Conflict*: If the test partner(s) connect to the light-wrapped core through functional buses, they might imply the previous described test conflicts and hence **may not** be tested at the same time.

Data Structure: The data structure used to store the schedule information for each core is as follows:

Data structure core schedule

---

1. *TestRail*; /* The TestRail that the core is on */
2. *begin*; /* Schedule begin time of the core */
3. *end*; /* Schedule end time of the core */
4. *isScheduled*; /* Whether the core has been scheduled */
5. *inCompatibleCores*; /* The cores that cannot be scheduled concurrently */
6. $finishedPatterns$; /* The number of patterns that finished schedule*/
7. $finishedTime$; /* The test application time of the finished patterns*/

The *inCompatibleCores* list for every core is initialized in a pre-processing step based on the functional interconnects and wrapper properties. For a 1500-wrapped core, once it was assigned to a TestRail, its test application time is determined, however, its schedule sequence on the TestRail is not decided yet. We still consider it unscheduled ($isScheduled = false$) and therefore its *begin* and *end* times will be updated after the schedule of light-wrapped cores has been resolved. Since the test of light-wrapped cores involves its test partners, a light-wrapped core schedule might affect many other cores scheduled at the same time. Hence, when a light-wrapped core i has completed its schedule, $finishedPatterns$ and $finishedTime$ at $end(i)$ are updated for all the other concurrently scheduled cores.

Test application time for light-wrapped cores: When a new light-wrapped core is scheduled, all the cores that are tested at the same time as this light-wrapped core, will change their test application time. This is illustrated using the following example:

Example 6.4 *The hypothetical SOC shown in Figure 6.11 contains two light-wrapped cores: $Core_1$ and the UDL. When using TestRail architecture shown in Figure 6.11(b), based on the functional interconnect, $Core_2$ is test partner of $Core_1$, while $Core_3$ and $Core_4$ are test partners of UDL. Suppose the test pattern count for $Core_1$ and UDL are N_1 and N_{udl}, respectively ($N_1 > N_{udl}$). The TestRail architecture can be determined by the procedure DesignTestRail and, as shown in Figure 6.13(a), $Core_1$ will be assigned to TestRail 1. At this time, the load size for each pattern of $Core_1$ is $loadsize_1 = N_{sc1\_W_1} + \lceil N_{io2}/W_1 \rceil + 1$, where $N_{sc1\_W_1}$ is the maximum scan chain length of $Core_1$ after scan chain stitching to match TAM width W_1 and N_{io2} is the number of wrapper boundary cells of its test partner $Core_2$ (the value 1 stands for the bypass cycle on $Core_4$). Since $Core_1$ and $Core_2$ are on the same TestRail 1, the schedule of $Core_1$ is independent of TestRail 2. If the UDL is assigned on TestRail 1, as shown in Figure 6.13(b), since it shares the same TestRail with $Core_1$ and they are tested concurrently, the load size for each of the overlapped test patterns is $loadsize_{overlapped} = max\{N_{sc1\_W_1} + N_{scudl\_W1} + \lceil N_{io2}/W_1 \rceil + \lceil N_{io4}/W_1 \rceil, \lceil N_{io3}/W_2 \rceil\}$, where $N_{scudl\_W_1}$ stands for the maximum scan chain length of UDL after scan chain stitching to match TAM width W_1. If the UDL is assigned on TestRail 2, as depicted in Figure 6.13(c), however, although $Core_1$ and UDL are not on the same TestRail, their test partners share the same TestRail. Therefore, the load size for each of the overlapped test patterns will be $loadsize_{overlapped} = max\{N_{sc1\_W_1} + \lceil N_{io2}/W_1 \rceil + \lceil N_{io4}/W_1 \rceil, N_{scudl\_W_2} + \lceil N_{io3}/W_2 \rceil\}\}$. In Figures 6.13(b) and 6.13(c) the rectangle UDL_{tp} stands for the time required to load the test patterns of UDL. In both cases after the UDL has finished its schedule, the load time for each of the remaining $N_1 - N_{udl}$ test patterns for $Core_1$ is $loadsize_1$.*

6.3.1 Determine the TestRail Architecture

In procedure **DesignTestRail** we determine the set of TestRails TR and the width of each TestRail $w(r)$ by optimizing TestRail architecture only for 1500-wrapped cores. Since the testing time of light-wrapped cores is dependent on its test partners' shifting time, we

need to consider it in this step in order to get an initial TestRail architecture more suitable for assigning light-wrapped cores in the following steps of the top-level Algorithm 1. Therefore, we use *BuildCostFunction* to try different costs in every iteration of Algorithm 1. Given a TestRail r with 1500-wrapped cores C and for each core $c_i \in C$ the time it serves as test partners is t_i and its number of wrapper cells is N_w, the cost function is $Cost = T(r) + \alpha \times \sum_i (t_i \times N_w)$, where $T(r)$ is the test application time for the TestRail, and α is a cost weighting scalar that is varied by the system integrator in solution space exploration. In our implementation, α is selected to be incremented by 0.1 in each iteration (e.g., for a $loopCnt = 500$, α varies from 0 to 49.9), which gives us good results with execution time of seconds. $TR-Architect$ [48] is revised to optimize *Cost* instead of testing time only in procedure DesignTestRail for the test architecture exploration.

6.3.2 Schedule Light-Wrapped Cores

The procedure *ScheduleLightCores*, as shown in Figure 6.14, schedules the light-wrapped cores onto a given TestRail architecture and tries to reduce the overall test application time. It takes the set of TestRails TR and the light-wrapped core set C_{light} as inputs, and it outputs the updated TestRail set TR' with all the light-wrapped cores scheduled on them and the overall test application time of the light-wrapped cores T_{light}. In this procedure, we schedule all the light-wrapped cores in front of 1500-wrapped cores in each TestRail, and there is no schedule overlap between any 1500-wrapped cores and light-wrapped cores so that we do not need to consider the Partner-CUT conflicts. In lines 1 and 2 we initialize TR', the unscheduled core set $C_{unScheduled}$ and the currently scheduled core set $C_{scheduling}$. Inside the loop, the light-wrapped cores are scheduled (line 3-23). The procedure first finds a core i compatible with $C_{scheduling}$ (i.e., it does not have any shared-partner conflicts with the cores in $C_{scheduling}$) with maximum test pattern count (line 4). Then if core i is a non-scanned core, it will not be assigned to any TestRail. In this case the procedure only updates the schedule of all the affected cores (line 6-7). If core i is a scanned core, the procedure will search through all the TestRails and try to assign the core to the TestRail r^* which leads to the minimum test application time. The time must account for the sum of all the affected light-wrapped cores $C_{affected}$ and the maximum test application time

Algorithm 6.5 - ScheduleLightCores

INPUT: TR, C_{light}
OUTPUT: TR', T_{light}

1. set $TR' = TR$;
2. set $C_{unScheduled} = C_{light}$; $C_{scheduling} = \emptyset$;
3. **while**($C_{unScheduled} != \emptyset$) {
4. **if** a compatible core i can be found with maximum N_p {
5. **if** (core i is non-scanned core) {
6. find the cores $C_{affected}$ whose schedule are affected;
7. $compTime(C_{affected}, TR')$;
8. } **else** {
9. Set $t(r^*)$ maximum value;
10. **for all** $r \in TR'$ {
11. $r_{temp} = r \cup \{i\}$; $TR_{temp} = TR' \setminus \{r\} \bigcup \{r_{temp}\}$;
12. find the TestRails $TR_{affected}$ whose schedule are affected;;
13. find the cores $C_{affected}$ whose schedule are affected;
14. $t(r_{temp}) = compTime(C_{affected}, TR_{temp}) + T(TR_{affected})$;
15. **if**($t(r_{temp}) < t(r^*)$) {
16. $r^* = r_{temp}$; $TR^* = TR_{temp}$;
. }
. }
17. $TR' = TR^*$;
. }
18. $C_{scheduling} = C_{scheduling} \bigcup \{i\}$;
19. $isScheduled(i) = true$; $C_{unScheduled} = C_{unScheduled} \setminus \{i\}$;
20. } **else** {
21. find core j in $C_{scheduling}$ such that $end(j)$ is minimum;
22. $C_{scheduling} = C_{scheduling} \setminus \{j\}$;
23. update $N_{finished}$, $T_{finished}$ for all cores in $C_{scheduling}$;
24. }
. }
25. set $T_{light} = max_{i \in C_{light}} end(i)$;
26. **return** TR', T_{light};

Figure 6.14: Procedure for Scheduling Light-Wrapped Cores onto TestRail Architecture

$T(TR_{affected})$ for all the 1500-wrapped cores on the affected TestRails $TR_{affected}$ (line 10-17). Taking $T(TR_{affected})$ into consideration is very important since it helps us to avoid the assignment of the light-wrapped cores which will affect badly the TestRail that already has a large testing time for the 1500-wrapped cores. After scheduling core *i*, $C_{scheduling}$ and $C_{unScheduled}$ are updated (line 18-19). If a compatible core with $C_{scheduling}$ cannot be found, the schedule of core *j* in $C_{scheduling}$ with the minimum *end* time will be finished, the number of finished test patterns and test application time of all the other cores in $C_{scheduling}$ are updated (line 21-23). The procedure is repeated until all the light-wrapped cores are scheduled. In line 25, the overall test application time of light-wrapped cores T_{light} is computed, which also gives the begin time for 1500-wrapped cores in every TestRail.

An example schedule, for a hypothetical SOC with five 1500-wrapped cores and three light-wrapped cores, after the *ScheduleLightCores* step is shown in Figure 6.17(a). It can be seen the overall test schedule is divided into separate test sessions for light-wrapped cores and 1500-wrapped cores, and hence incurs a relatively large idle time.

6.3.3 ReSchedule 1500-Wrapped Cores within TestRail

The procedure **RescheduleInRail**, as shown in Figure 6.15, tries to move the 1500-wrapped cores to the idle time created by scheduling light-wrapped cores, in order to reduce the overall test application time of the longest TestRail. Since the core set on every TestRail will not change, this rescheduling will not affect the core schedule on other TestRails. For each TestRail *r*, the procedure searches through all the idle ranges one by one (controlled by $upperLimit$), to reschedule the 1500-wrapped cores. First the idle time is computed (line 4-8). If the idle range ends at T_{light}, then all the remaining 1500-wrapped cores can move forward to the beginning of this idle range. Hence T_{idle} is set as maximum value (line 5-6). Then an unscheduled core *i* with maximum test application time which can fit in the idle range is found and scheduled (line 9-11). If such a core cannot be found, $upperLimit$ will be updated to the end time of this idle range so that the procedure will try the next idle range (line 13). Once all the idle ranges are searched, the procedure will update the schedule of the remaining 1500-wrapped cores (line 16).

As illustrated in Figure 6.17(b), after the **RescheduleInRail** step the 1500-wrapped

Algorithm 6.6 - RescheduleInRail

INPUT: TR, T_{light}
OUTPUT: TR'

```
1. for all r ∈ TR' {
2.    set upperLimit = 0;
3.    while (true) {
4.       if an idle range < idleBegin, idleEnd > on r can be found
.             such that idleBegin ≥ upperLimit {
5.          if (idleEnd == T_light) {
6.             set T_idle maximum value;
7.          } else {
8.             T_idle = idleEnd − idleBegin;
.          }
9.          if a unscheduled 1500-wrapped core i in r can be found
.              such that T_i < T_idle AND T_i is the maximum {
10.            begin(i) = idleBegin; end(i) = begin(i) + T_i;
11.            upperLimit = end(i);
12.         } else {
13.            upperLimit = idleEnd;
.          }
14.      } else {
15.         break;
.        }
16.      update the schedule of the remaining 1500-wrapped cores;
.     }
.  }
17. return TR';
```

Figure 6.15: Procedure for Rescheduling 1500-Wrapped Cores within TestRail

cores PC_1 on TestRail1 and PC_2 on TestRail2 are rescheduled. As a consequence, the TestRail3 becomes the new bottleneck TAM, which shortens the overall test application time of the SOC.

Algorithm 6.7 - RescheduleBetweenRails

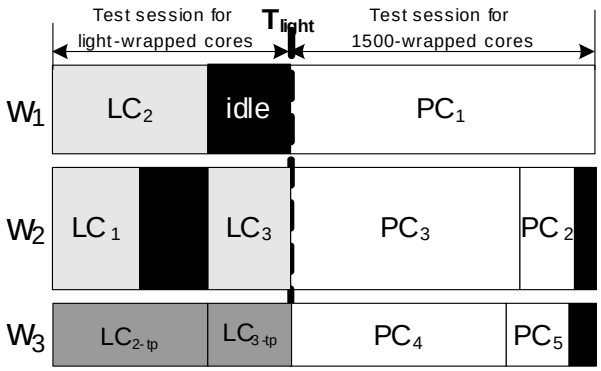
INPUT: TR
OUTPUT: TR', T_{soc}

1. $isImproved = true$;
2. **while** (isImproved) {
3. find r_{max} for which $T(r_{max}) = max_{r \in TR}\, T(r)$;
4. find cores C on r_{max} which do not serve as test partners;
5. **if** ($C == \emptyset$) {$isImproved = false$; **break**;}
6. find core i^* in C for which $T_{i^*} = min_{i \in C} T(i)$;
7. Set $isFound = true$;
8. **for all** $r \in TR \setminus \{r_{max}\}$ {
9. **for all** idle ranges $< idleBegin, idleEnd >$ on r {
10. $T_{idle} = idleEnd - idleBegin$;
11. **if** ($TestTime(i^*, r) \leq T_{idle}$) {
12. schedule core i^* to the idle range;
13. $r_{max} = r_{max} \setminus \{i^*\}$; $r = r \bigcup \{i^*\}$;
14. $isFound = true$;
15. **break**;
. }
. }
16. **if** (isFound) **break**;
. }
17. if (!isFound) {isImproved=false};
. }
18. $T_{soc} = max_{r \in TR'} T(r)$;
19. **return** TR', T_{soc};

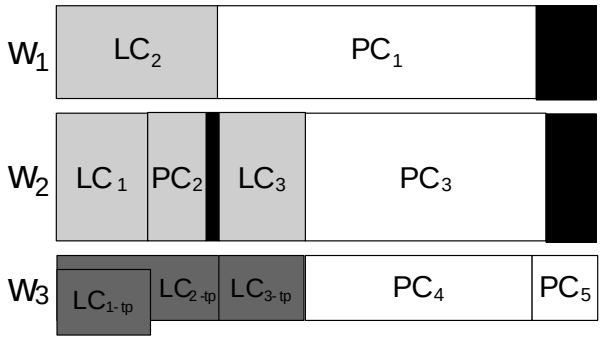
Figure 6.16: Procedure for Rescheduling 1500-Wrapped Cores in between Different TestRails

6.3.4 ReSchedule Wrapped Cores in between Different TestRails

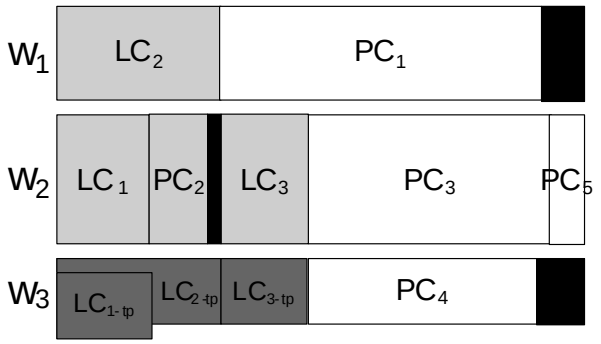
The procedure *RescheduleBetweenRails*, as shown in Figure 6.16, attempts to reduce the test application time of a given TestRail architecture by moving the 1500-wrapped cores from the bottleneck TestRail to another TestRail, provided that it reduces the overall test application time. If a 1500-wrapped core serves as a test partner for light-wrapped cores, placing it to a different TestRail will affect core schedule on other TestRails. As a result, we



(a) After step *ScheduleLightCores*



(b) After step *RescheduleInRail*



(c) After step *RescheduleBetweenRails*

Figure 6.17: *LightTRDesign* Algorithm for an Example SOC with Five 1500-Wrapped Cores and Three Light-Wrapped Cores.

only consider moving those cores which do not serve as test partners for any light-wrapped cores in this procedure. The procedure is iterative. In each iteration, it first identifies the bottleneck TestRail r_{max} (Line 3). Line 4 finds all the 1500-wrapped cores C on r_{max} which do not serve as test partners of light-wrapped cores. Then the procedure searches through other TestRails to see whether there is sufficient idle time to fit in core i^*, whose test application time is the shortest in C (line 8-17). For each TestRail r, the procedure searches through every idle range. Note, the idle time between r and r_{max} is also one of the idle ranges. If such a TestRail can be found, core i^* will be rescheduled on the new TestRail (line 14). The procedure exits when C is empty or no beneficial re-assignment can be found.

For the example from the previous two subsections, after step *RescheduleBetweenRails*, the core PC_5, originally on bottleneck TestRail3, is rescheduled to TestRail2 which reduces the test application time, as shown in Figure 6.17(c).

6.4 Experimental Results

The purpose of our experiments is to find out (i) When system integrators want to maximize the number of light-wrapped cores, how much DFT area can be saved without affecting the test quality and what are the implications of these savings on testing time; (ii) When system integrators pre-determined the set of light-wrapped cores, what are the testing times for the two proposed test architectures. As a result, in addition to the hypothetical SOC m4953, benchmark SOCs from the ITC'02 *SOC test benchmarking initiative* ([115]) are used in our experiments. Since the functional interconnects are not provided in the original benchmark files, similarly to the configuration described in Section 5.4.1, we have randomly generate them to support the proposed approach, including the direct connection between cores and functional busses. We have assumed that the SOCs have $Round(\frac{N_c}{10})$ busses and each bus has a random number p $(3 \leq p \leq min(N_c, 8))$ of cores attached to it, where N_c is the total number of cores in the SOC. In addition, all cores on busses are assumed to be able to transfer data to and from the bus, and hence each one of them can be a bus producer to others. We have also assumed that every core has a random number of $q(1 \leq q \leq 3)$ producers (consumers are generated from the producer-CUT relationship automatically).

	N_{in}	N_{out}	N_{bi}	N_{sc}	SC_{length}
$Core_1$	35	50	28	23	5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4
$Core_2$	117	84	36	7	221 221 221 221 221 200 178
$Core_3$	144	47	72	4	149 149 149 147
$Core_4$	202	60	20	0	
$Core_5$	126	25	40	3	300 299 299
$Core_6$	29	40	6	0	
$Core_7$	6	242	0	0	
$Core_8$	136	12	28	3	160 159 159
$Core_9$	194	157	6	4	149 149 149 147

Table 6.1: Test Parameters of SOC m4953.

To investigate the implication of the proposed architectures on the DFT area savings and its impact on SOC testing time, four experiments are carried out, as follows.

Experiment 1 compares the test schedule obtained for SOC m4953 when all the embedded cores are 1500-wrapped against the case when 4 cores are light-wrapped using the proposed Producer-CUT-Consumer architecture (Section 6.4.1);

Experiment 2 analyzes the number of cores that can be light-wrapped and the number of wrapper cells that can be saved (Section 6.4.2);

Experiment 3 discusses the TAT implications of using the Producer-CUT-Consumer architecture compared with the serial ExTest strategy (Section 6.4.3);

Experiment 4 presents the TAT comparison between using the Producer-CUT-Consumer architecture and the TestRail architecture when the light-wrapped cores are pre-determined by system integrators (Section 6.4.4);

Note, it is assumed that no wrapper design constraints exist in Experiments 2 and 3, and the proposed algorithm from Section 6.2 determines the wrapper type of each core, the optimal TAM division and the test schedule. While in experiment 4, the wrapper type for each embedded core is pre-determined by the system integrator.

6.4.1 Experiment 1: Test Schedule Comparison for m4953

First we investigate our pruning technique used for rapidly dividing the available TAM lines into three separate groups: producer, CUT and consumer for SOC m4953. The test parameters for the cores in m4953 are shown in Table 6.1, in which N_{in}, N_{out}, N_{bi} and N_{sc}

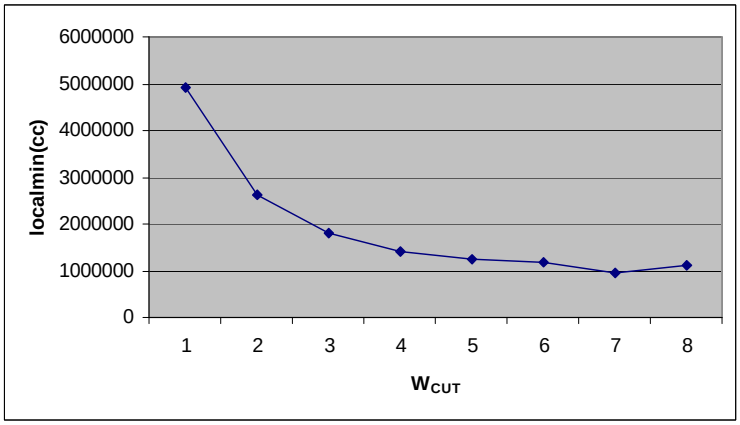


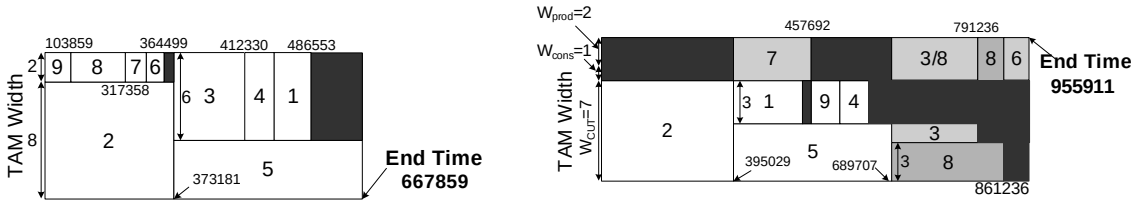
Figure 6.18: Test Application Time Variation with W_{CUT}.

denote the number of inputs, outputs, bidirectionals and scan chains in the specific core, respectively. The length of each scan chain is shown in column SC_{length}.

The TAT variation with W_{CUT} for m4953 is depicted in Figure 6.18 (given the total TAM width $W_{ttl} = 10$). Using the proposed *TAM_Division_Schedule* algorithm proposed in Section 6.2, we obtain the minimum TAT of 955911 clock cycles for $W_{CUT} = 7$, $W_{prod} = 2$ and $W_{cons} = 1$. For this particular case we deal with a convex function and the first identified local minimum is the only global minimum. If we set $weight = 1.1$ (see Algorithm 6.1), the search for the minimum TAT will start from $W_{CUT} = 8$ and stop at $W_{CUT} = 6$. This will prune the search space and hence reduce the computational time to a few seconds even for large SOCs, while getting the best possible TAM division and schedule.

In Figure 6.19(a), we present the test schedule obtained for m4953 when all the cores are 1500-wrapped. When applying the new $Decide\_Wrapper\_Type$, if no wrapper design constraints exist, $Core_3$, $Core_6$, $Core_7$ and $Core_8$ are selected to be light-wrapped, and the test schedule is shown in Figure 6.19(b). We can observe that the SOC TAT increases by approximately 45%, due to the following three main reasons:

1. the test conflicts introduced in the light-wrapped SOC test model cause more idle rectangle areas in the bin. For example, although $Core_6$ can fit into the rectangle area above $Core_2$, due to the producer-CUT conflict $Core_6$ is incompatible with $Core_2$ and hence they cannot be scheduled at the same time;



(a) Test schedule when all the cores are 1500-wrapped using [76]

(b) Test schedule when $Core_3$, $Core_6$, $Core_7$ and $Core_8$ are light-wrapped using the proposed method

Figure 6.19: Test Schedule Comparison for SOC m4953.

2. the number of TAM lines used to access the 1500-compliant cores and the internal scan chains of light-wrapped cores (G_{CUT}) are decreased from 10 to 7. To support light-wrapped core testing, two TAM lines are used for loading producers' outputs and one TAM line is used to unload consumers' inputs;

3. when two light-wrapped cores are scheduled at the same time, the load time for each overlapped pattern may increase. In this example, light-wrapped $Core_3$ and $Core_8$ are tested concurrently, and the load time of the overlapped test patterns is dominated by the loading time of the producer outputs;

It can be observed in Figure 6.19(b), that $Core_7$ is scheduled from the same starting time as $Core_1$ even when the available number of CUT TAM lines $G_{CUT} = 0$. This is because $Core_7$ has no internal scan chains and test data can be transferred only using G_{prod} and G_{cons}.

6.4.2 Experiment 2: Reduction in 1500-Wrappers and WBRs

Table 6.2 shows the reduction in the number of 1500-wrapped cores and the number of WBRs for the 100 random-generated interconnects for four benchmark SOCs [115], when using the *Decide_Wrapper_Type* procedure to maximize the number of light-wrapped cores. N_c is the total number of cores, while N_{max_l}, N_{min_l} and N_{ave_l} denote the maximum, minimum and average number of light-wrapped cores, respectively. N_{wbr}, is the total number of WBRs and N_{max_wbr}, N_{min_wbr} and N_{ave_wbr} denote the maximum, minimum and average number of WBRs that are removed. The percentage reductions are defined

SOC	N_c	N_{max_l}	N_{min_l}	N_{ave_l}	$\Delta N_l(\%)$	N_{wbr}	N_{max_wbr}	N_{min_wbr}	N_{ave_wbr}	$\Delta N_{wbr}(\%)$
g1023	14	7	4	5.77	41.21	3587	2367	504	1479.11	41.24
p34392	19	10	7	8.29	43.63	1884	1436	336	804.22	42.69
p93791	32	16	12	13.51	42.22	7697	4235	1983	2871.82	37.31
t512505	31	16	10	12.62	40.71	8503	5035	1798	3287.67	38.66

Table 6.2: The Number of Light-Wrapped Cores for Benchmark SOCs.

as $\Delta N_l(\%) = \frac{N_{ave\_l}}{N_c} \times 100$ and $\Delta N_{wbr}(\%) = \frac{N_{ave\_wbr}}{N_{wbr}} \times 100$. To provide the same test quality for core-based SOCs, a high number of cores (approximately 40%) does not need to be wrapped with WBR cells. Based on functional interconnect topology there are cases where the maximum number of light-wrapped cores can be half of the total number of cores (see column 3 in Table 6.2). More importantly, the number of WBRs that can be removed varies from hundreds for smaller benchmarks to thousands for the larger ones. Given the fact that each WBR can have an equivalent of 10 to 60 logic gates [167] (depending on the number of flip-flops and the modes used for each cell), we believe that the proposed solution can yield significant savings in DFT area. Because these savings do not come at no expense, the implications on the testing time are discussed next.

6.4.3 Experiment 3: Testing Time for Producer-CUT-Consumer Architecture

To investigate the implications on test application time using Producer-CUT-Consumer architecture, we compare the results from Section 6.2 against the case when the light-wrapped cores are tested sequentially using serial EXTEST *after* the test of all the wrapped cores. When the light-wrapped cores are tested sequentially, it is assumed that the entire parallel TAM bandwidth is allocated to their internal scan chains.

Tables 6.3, 6.4, 6.5 and 6.6 present test application time results when varying the total TAM width W_{ttl} (note, only results with the optimal TAM divisions are reported). T_{ave_se}, $T_{\max\_se}$ and $T_{\min\_se}$ denote the average, maximum and minimum TAT for the 100 random circuits when serial ExTest is used to test the light-wrapped cores. T_{ave_p}, $T_{\max\_p}$ and $T_{\min\_p}$ denote the average, maximum and minimum TAT for the 100 random circuits when the

	SOC g1023								
	[76]	Serial ExTest Architecture				Producer-CUT-Consumer Architecture			
W_{ttl}	T (cc)	T_{max_se} (cc)	$T_{\min\_se}$ (cc)	T_{ave_se} (cc)	ΔT_{se} (%)	T_{max_p} (cc)	$T_{\min\_p}$ (cc)	T_{ave_p} (cc)	ΔT_p (%)
8	66423	3164423	270004	1089174	+1539.75	362484	87109	188585	+183.92
16	35156	3149277	256732	1072909	+2951.85	180895	42895	91760	+161.01
24	24018	3141727	253263	1066494	+4340.39	120686	28234	63304	+163.57
32	19620	3141727	249221	1064404	+5325.10	93622	21665	49676	+153.19
40	14794	3141727	246090	1063731	+7090.29	75094	19567	42305	+185.96
48	14794	3141727	246090	1063597	+7089.38	64360	19054	37667	+154.61
56	14794	3141727	246090	1063479	+7088.58	58141	19054	34589	+133.80
64	14794	3141727	246090	1063299	+7087.37	52435	18265	31997	+116.28

Table 6.3: Test Application Time Comparison for g1023.

	SOC p34392								
	[76]	Serial ExTest Architecture				Producer-CUT-Consumer Architecture			
W_{ttl}	T (cc)	T_{max_se} (cc)	$T_{\min\_se}$ (cc)	T_{ave_se} (cc)	ΔT_{se} (%)	T_{max_p} (cc)	$T_{\min\_p}$ (cc)	T_{ave_p} (cc)	ΔT_p (%)
8	2198975	21926765	3186491	11207946	+409.69	6174198	2807634	4019375	+82.78
16	1075242	21095812	2369160	10339305	+861.58	3148405	1396602	2049255	+90.59
24	838643	20859213	2183218	10248657	+1122.05	2163488	998371	1477729	+76.20
32	544579	20565149	2133872	10114582	+1757.32	1682451	838643	1208688	+121.95
40	544579	20565149	2133872	10110141	+1756.51	1495912	601822	1064984	+95.56
48	544579	20565149	2133872	10110141	+1756.51	1393674	563150	979985	+79.95
56	544579	20565149	2133872	10110141	+1756.51	1323567	544579	923876	+69.95
64	544579	20565149	2133872	10110141	+1756.51	1286667	544579	895010	+64.35

Table 6.4: Test Application Time Comparison for p34392.

proposed producer-CUT-consumer architecture is used. The percentage changes are calculated using the formula $\Delta T_{se}(\%) = \frac{T_{ave\_se} - T}{T} \times 100$ and $\Delta T_p(\%) = \frac{T_{ave\_p} - T}{T} \times 100$, where T is the test application time result obtained using the algorithm described in [76]. It should be noted that since we did not manually select the d and p parameters, T is slightly different when compared to the result reported in [76]. As seen in all the tables, in almost all the cases ΔT_{se} is much higher than ΔT_p, especially when the total TAM width W_{ttl} is large. This shows the effectiveness of the proposed test architecture. We can observe that T_{ave_se} does not change a lot with the variation of W_{ttl} when serial ExTest is used for testing

	SOC p93791								
	[76]	Serial ExTest Architecture				Producer-CUT-Consumer Architecture			
W_{ttl}	T (cc)	T_{max_se} (cc)	$T_{\min\_se}$ (cc)	T_{ave_se} (cc)	ΔT_{se} (%)	T_{max_p} (cc)	$T_{\min\_p}$ (cc)	T_{ave_p} (cc)	ΔT_p (%)
8	3642176	20864571	5302688	10878060	+198.67	7131063	4566696	5192296	+42.56
16	1901700	19538337	3858733	9513093	+400.24	3577099	2259929	2598762	+36.65
24	1233570	19052632	3306718	9059614	+634.42	2409185	1554572	1827146	+48.12
32	1052919	18943120	3132505	8936536	+748.74	1826561	1159523	1313552	+24.75
40	869430	18849275	3000622	8801580	+912.34	1426151	963158	1132204	+30.22
48	640190	18644076	2842435	8658272	+1252.45	1210150	787964	973060	+52.00
56	598231	18642092	2802460	8636190	+1343.62	1145604	601717	795703	+33.01
64	544052	18587913	2745080	8599643	+1480.67	1006763	583079	680408	+25.06

Table 6.5: Test Application Time Comparison for p93791.

light-wrapped cores. This is because the single-bit loading/unloading time for producers/consumers dominates the overall TAT of the SOC and the increase of W_{ttl} does not help in shortening it. While for the proposed Producer-CUT-Consumer architecture, when W_{ttl} is increased the algorithm will distribute more TAM lines to the bottleneck TAM group and leads to decreased TAT. One exception in the experiments is when $W_{ttl} = 8$ for SOC t512505, where the serial ExTest gives better result. This is because the number of internal memory elements is much larger than the number of cores' I/Os in this SOC. When the W_{ttl} is small, test data transportation into the CUT is the bottleneck. Since the proposed architecture requires at least 1 TAM line for G_{prod} and 1 TAM line for G_{cons}, only 6 TAM lines are left in G_{CUT} to transfer test data to/from the cores' internal memory elements, while all 8 TAM lines can be used for the same duty when serial ExTest is employed.

In can be seen in Tables 6.3, 6.4, 6.5 and 6.6, the average increase in TAT over [76] can vary from about 4% to 186% when the proposed architecture is used. For g1023 the penalty is higher than for the other SOCs. This is because, in addition to the reasons analyzed earlier in Experiment 1, the number of internal scanned flip flops in g1023 is comparable to the number of the producers'/consumers' outputs/inputs. Hence a large amount of time is necessary to load/unload test stimuli/responses, which imposes a high number of TAM lines assigned to producer/consumer TAMs. This leads to less TAM lines for CUTs to transport test data to/from the internal scan chains of all the cores. It can also be observed

	SOC t512505								
	[76]	Serial ExTest Architecture				Producer-CUT-Consumer Architecture			
W_{ttl}	T (cc)	T_{max_se} (cc)	$T_{\min\_se}$ (cc)	T_{ave_se} (cc)	ΔT_{se} (%)	T_{max_p} (cc)	$T_{\min\_p}$ (cc)	T_{ave_p} (cc)	ΔT_p (%)
8	23550880	28194691	24501542	26024702	+10.50	30181825	29513530	29758333	+26.36
16	11451554	17259395	13291295	14906314	+30.17	13786212	12024617	12836625	+12.10
24	10530995	17256024	13263325	14891063	+41.40	12697603	10453470	11157350	+5.95
32	6740743	13703678	8066245	9816884	+45.64	8528143	6277675	7005682	+3.93
40	5228420	13703678	7960790	9801760	+87.47	7523723	5228420	5961920	+14.03
48	5228420	13703678	7960790	9801760	+87.47	7523723	5228420	5958941	+13.97
56	5228420	13703678	7960790	9801760	+87.47	7523723	5228420	5958844	+13.97
64	5228420	13703678	7960790	9801760	+87.47	7523723	5228420	5958789	+13.97

Table 6.6: Test Application Time Comparison for t512505.

that the difference between the maximum and minimum TAT for different functional interconnect topologies may be very high, which is due to the unbalanced sizes of the cores inside the SOCs. For example, there are three large cores in p34392 ($Core_2$, $Core_{10}$ and $Core_{18}$). When these large cores are light-wrapped and the functional interconnect topology causes plenty of test conflicts between them, then the TAT will increase significantly. However, this penalty in TAT can be greatly improved simply by wrapping the large cores (that are involved in many test conflicts) with 1500-compliant wrappers. For SOC p93791, the sizes of the cores are medium and hence no core dominates the whole SOC TAT. As a result, although test conflicts exist between cores the idle time is not too large (in average the increase in TAT is about 37%). The TAT overhead for SOC t512505 is the smallest (in average about 12%) in the four benchmark SOCs. This is because one large core ($Core_{31}$) dominates the TAT of the entire SOC, and the additional time used to test the other incompatible cores is insignificant.

6.4.4 Experiment 4: Testing Time for Adapted TestRail Architecture

To investigate the implications on test application time using the adapted TestRail architecture, we compare the results from Section 6.3 against the one obtained from 6.2 for two benchmark SOCs, p34392 and p93791, with different light-wrapped core configurations. For the given interconnects (we randomly generated the functional interconnect only once),

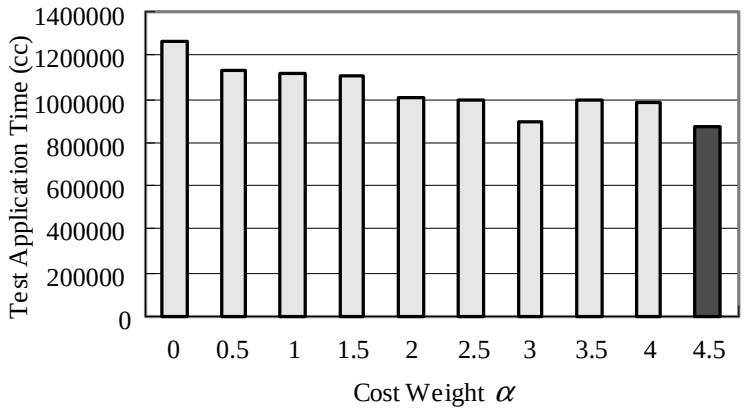


Figure 6.20: Test Application Time Variation for p34392 with Different *costweight*.

at most 8 cores in p34392 and 12 cores in p93791 can be light-wrapped.

First, for p34392 when $W_{ttl} = 32$, we analyze the test application time variation with different values of the cost weight α (from 0 to 4.5), used by the *BuildCostFunction*. As shown in Figure 6.20, the variation can be large, which justifies the need to try different starting TestRail architectures for each of the *loopCnt*=500 iterations of the top-level algorithm *LightTRDesign*. The large number of trials for the initial architecture effectively compensates for the lack of optimization in the *DesignTestRail* step.

Tables 6.7 and 6.8 show the test application time comparison between the two proposed test architectures for SOCs containing light-wrapped cores. We have used four different light-wrapped core configurations for benchmark SOCs p34392 and p93791. For each SOC, when the number of light-wrapped cores (N_l) is increased we keep the light-wrapped cores from the previous experiments (with the lower number of light-wrapped cores). When the overall TAM width is small ($W_{ttl} = 4, 8$ or 16 in this experiment), in almost all the cases, the test application time for the adapted TestRail architecture is lower than the one using Producer-CUT-Consumer architecture. This is because, separate producer and consumer TAMs have to be designed to shift in/out the test stimuli/responses to/from the producers and consumers of the light-wrapped cores, for the Producer-CUT-Consumer architecture from. This leads to a decrease in the number of TAM lines for the CUT TAM group, which are used to shift in/out all the test data for 1500-wrapped cores and the internal scan chains in light-wrapped cores. In contrast, for the adapted TestRail architecture, all the TAM lines

are used to load the test data for both 1500-wrapped cores and the internal scan chains of the light-wrapped cores. As a result, for Producer-CUT-Consumer architecture, when W_{ttl} is small, the CUT TAM group dominates the test application time of the entire SOC, since its available bandwidth is low. When increasing W_{ttl}, CUT TAM width no longer determines the bottleneck for the SOC test schedule, as it can be observed in Tables 6.7 and 6.8. Therefore, the improvements of the Adapted TestRail architecture disappear when increasing the W_{ttl} and we attribute the few contradictory cases to the fact that the fast heuristic search engines cannot always lead to near-optimal results.

From Tables 6.7 and 6.8, it can also be seen that when the number of light-wrapped cores N_l is relatively small, the adapted TestRail architecture leads to shorter test application time, while Producer-CUT-Consumer architecture gives better results when N_l is large. This is because, again, in the TestRail-based method, there are no dedicated producer and consumer TAM groups. When N_l is relatively small, the test data to be shifted for the light-wrapped cores is also small. In this case the load time for all the light-wrapped cores that are scheduled at the same time is low and thus the testing time is dominated by the time needed to load data in cores' internal scan chains. When N_l is large, whenever a light-wrapped core is scheduled, it is very likely that the schedule of several other concurrently tested light-wrapped cores will be affected (and hence the TAM resources cannot be used to test other cores). In addition, when considering all the loading time for all the test partners for all the light-wrapped cores scheduled at the same time, the test application time for the overlapped patterns is significantly increased. While for Producer-CUT-Consumer architecture, since dedicated producer and consumer TAMs are used to shift in/out the test data for the light-wrapped cores, the CUT TAM resources can be fully exploited to test the wrapped cores, and hence it is more efficient when N_l is large. For SOC p34392 when $W_{ttl} = 32$, the result obtained in Producer-CUT-Consumer architecture is better even when $N_l = 2$ or $N_l = 4$. This is because, in these two cases, core 18, which is quite large and dominates the test application time of the entire SOC, is 1500-wrapped. If the proposed method places the schedule of the light-wrapped cores in front of core 18, the test application time will further increase. When $W_l = 6$ and $W_l = 8$, core 18 is light-wrapped and the previous discussion can be followed, i.e., the adapted TestRail architecture gives better results when N_l is relatively small.

	SOC p34392											
	$N_l = 2$			$N_l = 4$			$N_l = 6$			$N_l = 8$		
W_{ttl}	T (cc)	T_{new} (cc)	ΔT (%)	T (cc)	T_{new} (cc)	ΔT (%)	T (cc)	T_{new} (cc)	ΔT (%)	T (cc)	T_{new} (cc)	ΔT (%)
4	7276835	4288630	-41.06	7276835	4598506	-36.81	7153222	4636891	-35.18	7474887	5276385	-29.41
8	2781117	2199507	-20.91	2781117	2387461	-14.15	2781117	2384181	-14.27	3226091	2923023	-9.39
16	1396602	1163432	-16.70	1396602	1201762	-13.95	1396602	1324457	-5.17	1396603	1560145	+11.71
24	838643	840216	+1.88	838643	889207	+6.03	969392	933378	-3.72	1243747	1356128	+9.04
32	544579	637660	+17.09	607678	726572	+19.57	765643	756336	-1.22	1119960	1175547	+4.96
40	544579	544579	0	570229	571600	+2.40	572794	726772	+26.88	1036450	1077662	+3.98
48	544579	544579	0	544579	544579	0	544579	622163	+14.25	1013245	1043252	+2.96
56	544579	544579	0	544579	544579	0	544579	605659	+11.22	999723	1027602	+2.79
64	544579	544579	0	544579	544579	0	544579	582992	+7.05	996929	1028265	+3.14

Table 6.7: TAT Comparison between The Two Proposed Test Architectures for p34392 with Different Light-Wrapped Core Configurations.

W_{ttl}	SOC p93791											
	$N_l = 4$			$N_l = 6$			$N_l = 8$			$N_l = 12$		
	T (cc)	T_{new} (cc)	ΔT (%)	T (cc)	T_{new} (cc)	ΔT (%)	T (cc)	T_{new} (cc)	ΔT (%)	T (cc)	T_{new} (cc)	ΔT (%)
4	13942991	7915588	-43.23	13894497	8395010	-39.58	13876650	8747510	-36.96	13831009	9924700	-28.24
8	4796178	4048644	-15.59	4841393	4333418	-10.49	4839660	4567028	-5.63	5072044	5174018	+2.01
16	2318569	2105982	-9.19	2383685	2246607	-5.75	2383686	2366156	-0.74	2566209	2879062	+12.19
24	1551868	1382726	-10.90	1568659	1562007	-0.42	1594786	1647152	+3.28	1803658	1969220	+9.18
32	1192928	1035950	-13.16	1233571	1201271	-2.62	1233572	1264935	+2.54	1278032	1515097	+18.55
40	980098	858994	-12.36	974074	951174	-2.35	937510	1046972	+11.68	1128936	1206425	+6.86
48	809219	736656	-8.97	841379	868083	+3.17	841379	844862	+0.41	937252	1089628	+16.26
56	628907	635909	+1.11	696490	683161	-1.91	714528	786894	+10.13	749536	945671	+26.17
64	592361	548069	-7.48	580302	623143	+7.38	616617	656602	+6.48	678392	898136	+32.29

Table 6.8: TAT Comparison between The Two Proposed Test Architectures for p93791 with Different Light-Wrapped Core Configurations.

6.5 Concluding Remarks

This chapter addresses the problem of effectively and efficiently testing SOCs containing light-wrapped cores. Two scenarios according to the number of light-wrapped cores are considered. When this number is large, we propose a novel $Producer - CUT - Consumer$ test architecture, similar to the one presented in Chapter 5. When this number is comparably small, we show how to adapt the TestRail architecture to obtain an effective test strategy by accessing the internal SFFs of light-wrapped cores in Parallel ExTest mode. We also present how to optimize the two proposed architectures in terms of test application time.

So far we have addressed several emerging problems for testing SOCs that have one level of hierarchy (SOC and cores), including how to test multi-frequency IP cores, how to test SOCs containing two-pattern tested cores and how to test SOCs containing light-wrapped cores. With the increase of reusability, past-generation SOCs might be used as embedded cores in next-generation designs, and hence an emerging problem is how to test such SOCs that have multiple levels of hierarchy. This problem is tackled in the following chapter.

Chapter 7

Multi-Frequency TAM Design for Hierarchical SOCs

The emergence of hierarchical cores in SOC design presents new challenges to electronic test automation. This chapter describes a new framework for designing TAMs for modular testing of such SOCs with multiple levels of hierarchy. We first explore the concept that TAMs on the same level of design hierarchy employ multiple frequencies for test data transportation. Then, we extend this concept to hierarchical SOCs and, by introducing frequency converters at the inputs and outputs of the hierarchical cores, the proposed solution not only removes the constraint that the system level TAM width must be wider than the internal TAM width of the hierarchical cores, but also facilitates rapid exploration of the trade-offs between the test application time and the required DFT area. Experimental results for the ITC'02 SOC Test Benchmarks show that the proposed TAM design algorithms increase the size of the solution space that is explored which, in turn, will lower the test application time, when compared to the existing solutions.

The organization of this chapter is as follows. Section 7.1 reviews the related approaches and outlines the main contributions of our work. In Section 7.2 we show how multiple frequency test data transportation can be used for flattened SOCs, which is then extended to hierarchical SOCs in Section 7.3. Section 7.4 presents the experimental results and finally Section 7.5 concludes the chapter.

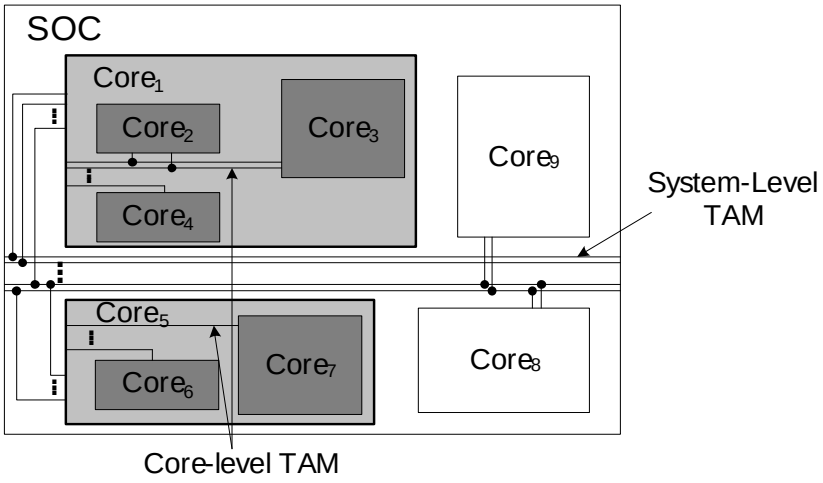


Figure 7.1: Hierarchical SOC Example.

7.1 Preliminaries and Summary of Contributions

With the capability to integrate tens or even hundreds of millions of transistors onto a silicon die [69], SOC designs are becoming too large and complex to have only one level of hierarchy (SOC and cores), since the capabilities of electronic design automation tools and computing resources improve at a slower pace. As a result, state-of-the-art SOC designs often have multiple levels of hierarchy [32, 45, 116, 130]. These hierarchical cores may be composed of several smaller in-house/external cores due to functional requirements [44], or simply old-generation SOCs. In this chapter, we refer to such hierarchical cores in SOC designs as ”mega-cores” and a lower-level core embedded in them as their ”child core”.

Most of the relevant research in core-based SOC testing (see Section 3.3.3) assumes that the hierarchy of the SOC is flattened during test, and hence only a single-level TAM is needed. This assumption is becoming unrealistic because, as illustrated in Figure 7.1, the hierarchical mega-cores may have the TAM already hard-wired inside. In addition, a straightforward extension of the known TAM design algorithms will not necessarily lead to effective solutions for testing hierarchical SOCs. Moreover, most of the prior research on test architecture optimization assumes that the ATE operates at the same frequency (f_{ext}) as the internal cores' scan chain frequencies (f_{tam}). This assumption has been shown to be inefficient when there is a mismatch between the ATE capability and the internal scan

chain speed [87]. Multi-frequency TAM design, in which both the TAM width and the TAM running frequency assigned to each core are co-optimized, can facilitate SOC test cost reduction. A limited number of research approaches have been presented to address the above issues separately in [154, 152] (for multi-frequency TAM design) and in [8, 9, 44, 72, 107, 153] (for hierarchical SOC testing). Prior to outlining our contributions we will summarize the relevance and limitations of these methods. Once the known art is presented we stress the distinguishing features of the approach presented in this chapter.

7.1.1 Related Work on Multi-Frequency TAM Design

The mismatch between the frequencies of the external ATE and internal DFT logic leads to under-utilization of the available resources, which in turn will affect the cost of test. To address this problem, Khoche [87] proposed the *Bandwidth Matching* technique. *Bandwidth* is defined as the product of the width and the frequency of a scan architecture. Using serialization/deserialization frequency converters, a high bandwidth source/sink (e.g., ATE) can be connected to multiple low bandwidth sinks/sources (e.g., TAMs) as long as the bandwidth matches. As a follow up, Sehgal et al. [154] proposed to match ATE channels with high data rates to low-speed core scan chains using virtual TAMs. These virtual TAMs, however, are working at the same frequency. Later in [152], the ATE channels with high data rates are used to directly drive SOC TAM wires, and the heuristic approach based on rectangle packing from [76] was extended to optimize the dual-speed TAM architecture. [152, 154] were proposed mainly to fully exploit the capability of state-of-the-art ATEs to drive different channels at different data rates. Most of the ATEs currently in use, however, do not have this feature and therefore cannot employ the proposed techniques.

In fact, multi-frequency TAM design has the benefits to reduce SOC test cost even for low-end ATEs. This is because, in core-based SOC framework, the scan frequency for an embedded core is variable only within a dedicated range, which is often different for various cores. When single-frequency TAM design is utilized, f_{tam} is a compromise among the frequency ranges of all embedded cores and the external ATE. Since f_{tam} has a direct impact on test application time, TAM wire length and test power of the SOC, if integrated into test architecture optimization, the system integrator has a larger solution space to explore the

trade-off between these factors and hence a better solution can be achieved. Moreover, there may be cases when frequency ranges of multiple cores have no intersection at all, thus constraining system integrators to design multi-frequency TAMs to supply data to these cores at different rates. Therefore, prior to tackling the hierarchical SOC test problem, we first investigate a more general multi-frequency TAM design approach in a flattened SOC framework.

7.1.2 Related Work on Hierarchical SOC Testing

Only limited work has been done for testing SOCs that contain mega-cores. Benso et al. [9] proposed a hierarchical-distributed-data BIST (HD^2BIST) TAM architecture, which allows test access through design hierarchies. Another TAM architecture presented by Benabdenbi et al., called CAS-BUS [8], also considered the existence of hierarchical mega-cores. In [107], Li et al. presented a hierarchical test scheme for SOCs with heterogeneous core tests, in which the proposed hierarchical test manager (HTM) is able to handle mega-core testing. All the above articles just briefly discussed how to provide test access for mega-cores. Recently, two 1500-compliant wrapper architectures for hierarchical cores were proposed, in which mega-core level TAM optimization was addressed. Goel [44] introduced a new wrapper cell design that allows a core to operate in InTest and ExTest mode concurrently and hence allows parallel testing of cores at different levels of hierarchy. His method resulted in reduced testing time for hierarchical SOCs, at the cost of a larger DFT area overhead. In [153], Sehgal et al. presented a general architecture for hierarchical core wrappers using conventional IEEE 1500 wrapper cells [117] and described various modes of operation of the wrapper. The proposed wrapper design is reconfigurable in order to operate efficiently in all the test modes. In [44, 153], mega-core level TAMs are assumed to be ”soft”, i.e., they are not fixed and system integrators are in charge of their design and optimization. In this chapter, however, we consider the case that mega-core level TAMs are ”hard”, i.e., their implementation is fixed ”as is”: not only the TAM architecture inside is pre-designed, but also the test schedule of all its child cores is pre-determined. The test of the mega-core itself is also implemented by the core provider (unlike in [44, 153] where it is implemented by system integrators), which could be a mega-core level ExTest (similar

to SOC ExTest to test interconnecting logic) or a ”hard” implementation of the test strategy proposed in [44, 153].

In another approach for hierarchical SOC testing [72], Iyengar et al. presented how to extend known wrapper/TAM optimization methods for flattened SOCs [73, 74] to multi-level TAM optimization in two different design transfer models. For the *interactive* design transfer model (i.e., the core provider can change the TAM based on the request of system integrators), a set of mega-core instances with different TAM widths need to be delivered, which obviously affects the reuse methodology and inhibits the development of hard mega-cores. To address this issue, *non-interactive* design transfer model can be employed (i.e., the cores are taken off-the-shelf and integrated into designs ”as is”). However, in this case, the system integrator has to design a wider system-level TAM to fork out to the pre-designed core-level TAMs. In addition, the constrained TAM width of the mega-core leads to very inflexible test schedule, and hence increased test application time.

7.1.3 Summary of Contributions

In this chapter, we explore the suitability of exploiting multi-frequency test data transportation to lower the test application time in hierarchical SOCs with hard mega-cores. The two main contributions of this chapter, detailed in Sections 7.2 and 7.3, are as follows:

- we first present a new multi-frequency TAM design algorithm for flattened SOCs; unlike [154, 152] we do not consider only the case when the ATE is faster than the scan chains, but we also examine the case when low-speed ATEs are used for high-speed scan chains, which may reduce also the routing overhead as long as power ratings are not exceeded; this is achieved by matching the bandwidth at the TAM level and refining an effective exploration engine [48, 51], which, ultimately, reduces test application time;

- we then extend the multi-frequency concepts to a multi-level TAM design algorithm for hierarchical SOCs; to fully reuse the *hard* mega-cores in a non-interactive design transfer model, we examine the usage of two types of frequency converters that can match a higher number of core-level TAM lines to a lower number of system-level

TAM lines; thereafter, by proposing a new design flow, in contrast to [72], we provide the system integrator the option to trade the DFT area against savings in test application time;

7.2 Multi-Frequency TAM Design for Flattened SOCs

Prior to addressing the test of hierarchical SOCs, we first explain how multi-frequency test data transportation can reduce the testing time for flattened SOCs (i.e., all the embedded cores are on the same level of hierarchy). We start by describing the architecture, then we formulate the problem to be solved and propose an extension of $TR-architect$ [48, 51] to support multi-frequency TAMs. Finally, we illustrate the benefits of the proposed approach on a benchmark SOC [116].

7.2.1 Multi-Frequency Test Architecture

Figure 7.2 shows the proposed test access architecture for flattened SOCs. To match the bandwidth of 6 external tester channels running at 100MHz to 3 internal TAM lines running at 100MHz, 2 internal TAM lines running at 50MHz and 8 internal TAM lines running at 25MHz, we place serialization/deserialization frequency converters at the input/output of the TAM groups. To keep the area overhead low and to facilitate our proposed algorithm (detailed in Section 7.2.2), we consider that the relationship between frequencies is a power of 2. This will lead to a straightforward serial/parallel shift register implementation for the frequency converters. Although the ratio between any two frequencies is a power of 2, this does not necessarily imply that the ratio between the number of tester channels and any TAM group is a power 2, since we can have separate TAM groups working at the same frequency. This is illustrated in the following example that shows the advantage of using multiple frequencies for test data transportation.

Example 7.1 *Consider a hypothetical SOC with 6 cores. The TAMs for these 6 cores are on the same level and the test schedules are shown in Figure 7.3. The original test schedule (with all TAMs operating at $f_{tam} = f_{ext}$) is shown in Figure 7.3(a), where four TAMs are employed (w_i is the width of TAMi) and TAM4 is the bottleneck TAM, i.e., the*

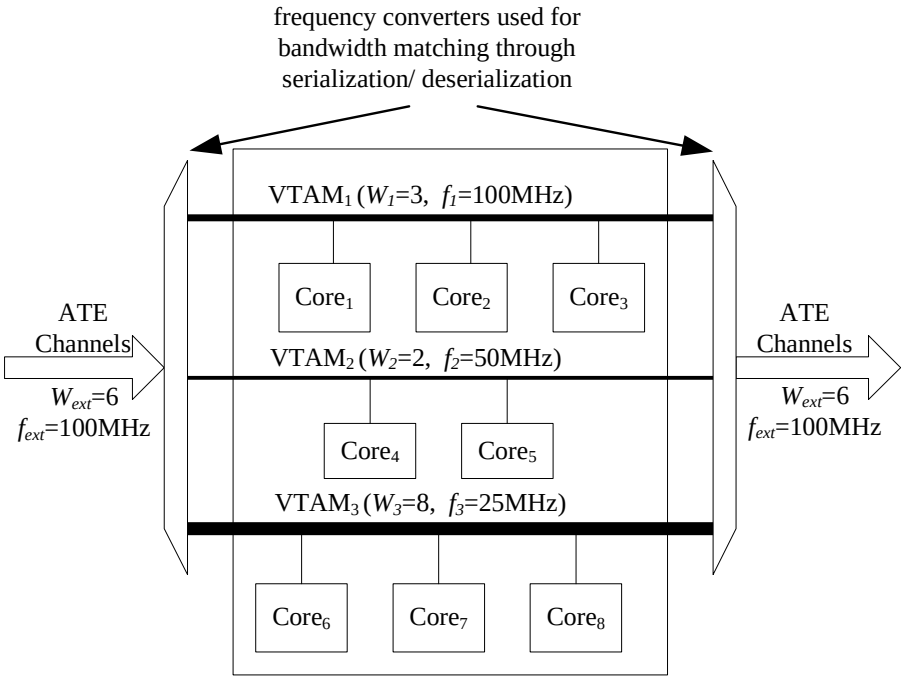


Figure 7.2: Proposed Multi-Frequency Test Architecture for Flattened SOCs.

overall test application time is dominated by it. It can be seen the idle time for the non-bottleneck TAMs is large. If, however, multi-frequency virtual TAMs are designed using the bandwidth matching technique with $f_{tam} \leq f_{ext}$ *(low frequency* f_{tam} *synchronized to a divisor of* f_{ext}*), as shown in Figure 7.3(b), TAM1, TAM2 and TAM3 operate at* $f_{ext}/2$*,* f_{ext} *and* $f_{ext}/2$ *respectively. The original bottleneck TAM takes less time with* $w_3 = 5$ *operating at* $f_{ext}/2$ *and the overall test application time is decreased with the new bottleneck TAM2. Unlike in [154], the serial/parallel converters used for bandwidth matching are placed at the TAM level, which may introduce additional routing overhead, however it will lead to more flexibility in the test schedule (i.e., the number of low speed TAM lines does not need to be a multiple by a power of 2 of the number of high speed tester lines). For example, in Figure 7.3(b), the width of the TAM bus used by Core 3 operating at* $f_{ext}/2$ *is* $w_3 = 5$*. Note, since the virtual TAMs are generated internally, the number of SOC pins for test purpose remains the same. If the TAMs can operate at higher frequency than the ATE (*$f_{tam} \leq 2 f_{ext}$ *in this example) and the power ratings are not exceeded then both the testing time and the routing overhead of the SOC can be decreased, as shown in Figure 7.3(c). This also*

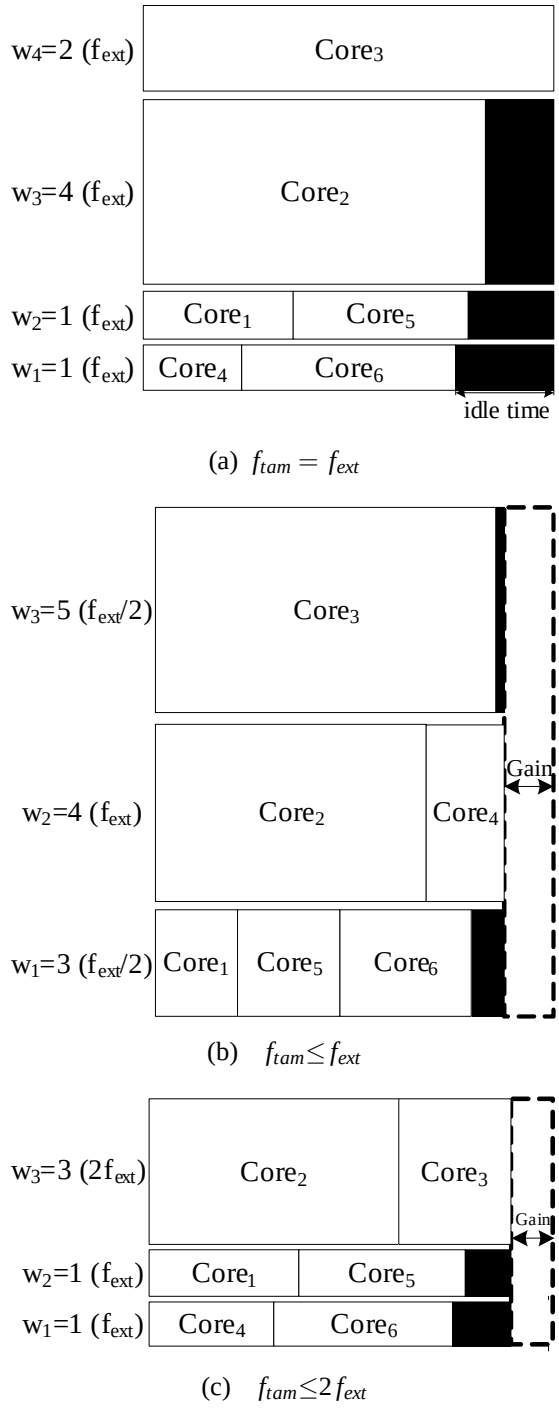


Figure 7.3: The Benefits of Multi-Frequency TAM Design.

Data Structure *TAM*

1. $width(i)$ /* width of TAM i */
2. $coreList(i)$ /* List of cores tested on TAM i */
3. $freqRatio(i)$ /* The freq. ratio between tester and TAM i*/
4. $T_{tam}(i)$ /* Test application time of TAM i */

facilitates high speed scan testing (scan frequency at several hundred MHz as shown in [58, 168]) using low-end ATEs, and, in terms of design methodology, the main difference to low speed testing using high speed ATEs lies in using parallel/serial conversion on the input and serial/parallel conversion on the output. When using low speed ATEs for high speed scan chains it is assumed that a high speed clock, generated on-chip, is used to serialize the low speed data arriving from the ATE.

7.2.2 Multi-Frequency TAM Design Algorithm

The multi-frequency TAM design problem, addressed in this section, is formulated as follows.

Problem $P_{mf-TAM-opt}$: Given the maximum TAM width W_{ttl} for the SOC, the operating frequency of the ATE f_{ext}, the test set parameters for each core, including the number of input terminals, the number of output terminals, the number of test patterns, the number of scan chains and for each scan chain its length (for cores with fixed-length internal scan chains) or the total number of SFFs (for cores with flexible-length internal scan chains), determine the width of each virtual TAM, the shift frequency f_{tam} of each virtual TAM, the wrapper design for each core, and a test schedule for the entire SOC such that: (i) the bandwidth of the internal TAM used at any time does not exceed the bandwidth of the ATE; and (ii) the overall SOC test application time is minimized.

Because the single frequency TAM design problem, which is known to be *NP-hard* [16], is a special case of the above one, in the following we proposed an heuristic algorithm to solve it. In our implementation we have adapted *TR-Architect* ([48, 51]) to multi-frequency TAM design, which is able to support both the TestRail and Test Bus architectures.

Algorithm 7.1 - mfTAMDesign

INPUT: C, $layoutConstraint$, W_{ttl}
OUTPUT: TAM_{design}, T_{soc}

1. **Compute** $initVTWidth$;
2. **Assign** $multiple = 1$;
3. for i from 1 to N_{mul} $\{$
4. $W_{vt} = multiple \times initVTWidth$;
5. $maxFreqRatio = multiple$;
6. $T_{soc}(i) = mf - TR - Architect(W_{vt}, maxFreqRatio, C)$;
7. T_{soc} = min(all $T_{soc}(i)$ with $W_{soc(i)} <= layoutConstraint \times W_{ttl}$);
8. Record TAM_{design} with testing time T_{soc};
9. $multiple = multiple \times 2$;
. $\}$
10. **return** (T_{soc}, TAM_{design})

Figure 7.4: Pseudocode for SOC Test Architecture Optimization Using Multi-Frequency TAMs.

Data Structure: The data structure *TAM* contains four elements. We define frequency ratio of the TAM as $freqRatio = \frac{f_{ext}}{f_{tam}}$, and T_{tam} is the testing time of a TAM measured in ATE clock cycles. Suppose the test application time of the TAM in its operating frequency is T_{of} clock cycles, then $T_{tam} = T_{of} \times freqRatio$. This data structure is updated whenever TAM merging, freed wires assignment or core test reshuffle during test scheduling happens.

Proposed Top-Level Algorithm: In $mfTAMDesign$ (shown in Figure 7.2.2), we first compute the initial Virtual TAM width using the bandwidth matching technique. Then we iteratively initialize the virtual TAM width W_{vt} as 2's exponent of the $initVTWidth$ (line 4). N_{mul} is a constant to stop the search in solution space (in our experiments $N_{mul} = 8$ provides a good trade-off between the computation time, which is at most within a few seconds even for large SOCs, and the quality of the final solution in terms of test application time). The maximum frequency ratio (*maxFreqRatio*) constrains the lowest possible frequency f_{min}= $f_{ext}/maxFreqRatio$ for current virtual TAM width (line 5). In line 6 we call an adapted TR-Architect algorithm $mf - TR - Architect$ to get the test application time $T_{soc}(i)$ with current virtual TAM width W_{vt}. When compared to [48, 51] the main differences are in

Algorithm 7.2 - mergeMFTAMs

INPUT: $TAM1$, $TAM2$, $vtWidth$
OUTPUT: $mergedTAM$, T_{tam}

1. $coreList_{tam} = coreList_{tam1} \bigcup coreList_{tam2}$;
2. **Assign** $freqRatio_{tam} = maxFreqRatio$;
3. **Assign** $W_{tam} = vtWidth$;
4. **for** all frequency ratios {
5. **compute** $T_{tam}(freqRatio_{tam})$;
6. $T_{tam} = \min$ { all $T_{tam}(freqRatio_{tam})$ };
7. Record $mergedTAM$ with minimum testing time T_{tam};
8. $freqRatio/ = 2$;
9. $W_{tam}/ = 2$;
. }
10. **return** ($mergedTAM$, T_{tam})

Figure 7.5: Procedure for Merging Multi-Frequency Virtual TAMs.

merging multi-frequency TAMs and distributing freed TAM resources, which are detailed in Algorithms 7.2 and 7.3. Since a large number of virtual TAMs may lead to routing overhead, we use a variable $layoutconstraint$ to restrict the number of virtual TAMs $W_{soc} \leq layoutconstraint \times W_{ttl}$ and we choose the TAM design TAM_{design} with the minimum T_{soc} without exceeding the layout constraint W_{soc}. In this chapter we assume that all the channels of the ATE run at the same frequency f_{ext}, however, if the tester has channels with different speeds, by applying the bandwidth matching translations, the number of ATE channels visible by the SOC can be re-calculated and provided as input to our algorithm.

Merging Multi-frequency TAMs: The procedure to merge multi-frequency virtual TAMs is shown in Figure 7.2.2. It enumeratively merges TAMs with different frequency/width configuration and selects the one with minimum T_{tam}. The algorithm starts by merging the cores on $TAM1$ and $TAM2$ (line 1) and then the frequency ratio $freqRatio_{tam}$ and the width of the TAM W_{tam} are initialized to $maxFreqRatio$ and $vtWidth$ respectively (lines 2 and 3). In the loop (lines 4-9) we enumeratively change the frequency/width configuration and compute $T_{tam}(freqRatio_{tam})$. Finally the TAM configuration $mergedTAM$ with minimum T_{tam} is returned.

Algorithm 7.3 - distributeVT

INPUT: R, $freedVT$
OUTPUT: R', $freedVT'$

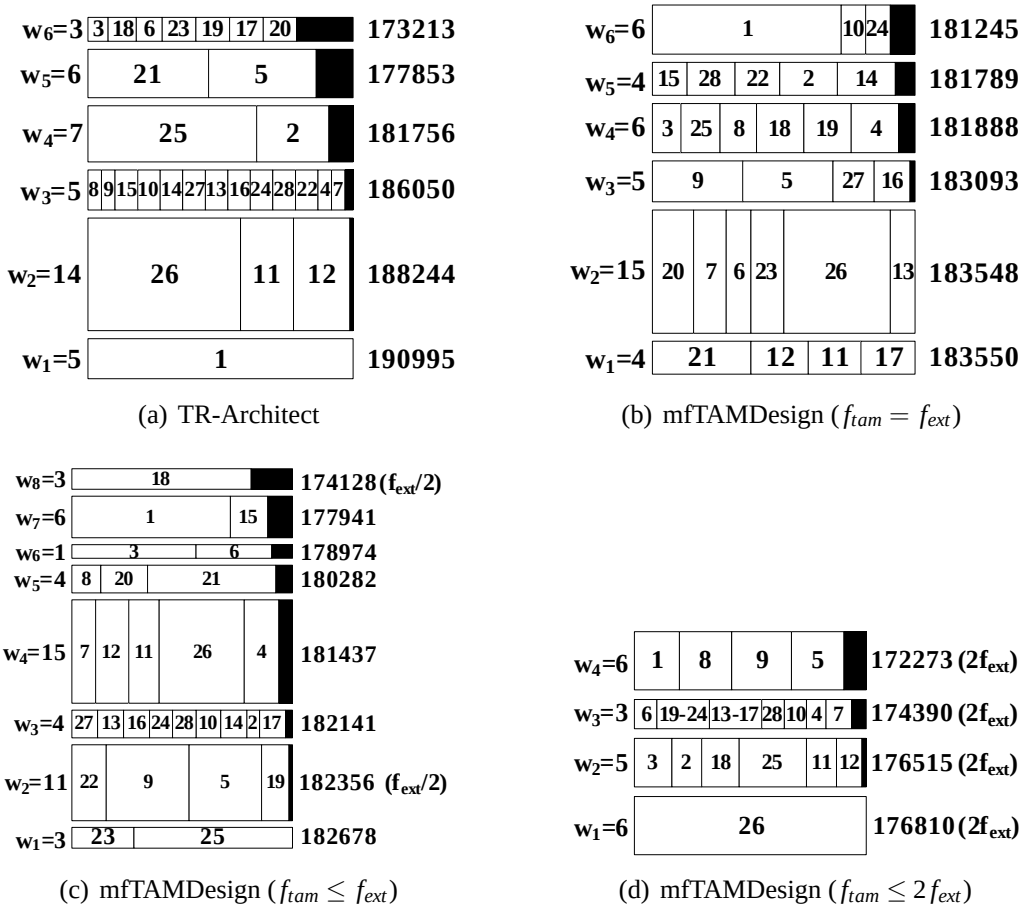
```
1. while (freedVT > 0) {
2.    find r_max with maximum T_r;
3.    wireWidth = maxFreqRatio ÷ freqRatio_{r_max};
4.    r_temp = r_max;
5.    assign isImproved = false;
6.    for(width = 1; width <= wireWidth; width* = 2) {
7.       if(freedVT < width) break;
8.       freqRatio_{r_temp} = maxFreqRatio;
9.       width_{r_temp} += width;
10.      compute minTime = T_{r_temp};
11.      if(T_{r_temp} < T_{r_max}) {
12.         r_max = r_temp;
13.         isImproved = true;}
14.      while ((freqRatio_{r_temp} ≥ 2)&&(width_{r_temp}%2 == 0)) {
15.         freqRatio_{r_temp} /= 2;
16.         width_{r_max} /= 2;
17.         compute T_{r_temp};
18.         if ((T_{r_temp} < T_{r_max})&&(T_{r_temp} < minTime)) {
19.            minTime = T_{r_temp};
20.            r_max = r_temp;
21.            isImproved = true;
22.         } else {
23.            continue; }
.        }
24.      if (isImproved == true) {
25.         freedVT -= width;
26.         break;
27.      } else if (width==wireWidth) {
28.         width_{r_max} += 1;
29.         freedVT -= width; }
.     }
30.   if (freedVT < wireWidth) && (!isImproved) break;
.  }
31. freedVT' = freedVT;
32. return (R', freedVT');
```

Figure 7.6: Procedure for Distributing Freed Virtual TAM Lines.

Distributing Free Virtual TAM Resources: Since the proposed solution does not restrict the number of lines in a TAM group working at a lower frequency to be a power of 2 (see example w_1, w_2 and w_3 in Figure 7.3), there is added flexibility for TAM merging, which ultimately turns out in more free virtual TAM resources that can be used for test application time reduction. The procedure *distributeVT*, shown in Figure 7.2.2, is used to iteratively distribute virtual TAM resources to reduce the idle time. The algorithm iteratively finds the bottleneck TAM (line 2) and then it computes the wire width (*wireWidth*) of this virtual TAM in terms of virtual TAM lines operating at f_{min} (line 3). Inside the *for* loop (lines 6-29), the algorithm tries to assign the least number of virtual TAM wires which can reduce T_{soc}. Whenever an additional virtual TAM wire is assigned to the bottleneck TAM, the algorithm tries to find whether the test application time can be decreased with different configuration of frequency and TAM width (lines 14-23). If T_{soc} is reduced (lines 24-26), the algorithm will loop back to find the new bottleneck TAM and start again. Even if T_{soc} is not decreased by assigning *wireWidth* of virtual TAM lines, the algorithm will still distribute this number of virtual TAM lines to the bottleneck TAM (lines 27-29) since this may provide a good starting point for the next iteration within TR-Architect [48, 51]. The algorithm terminates when $freedVT = 0$ or the test application time of the bottleneck TAM cannot be decreased and at the same time $freedVT < wireWidth$ (line 30). The freed virtual TAM wires $freedVT'$ can be used within the future iterations of TR-Architect to decrease T_{soc}. The benefits of using multi-frequency TAMs are illustrated next.

7.2.3 A Case Study for Benchmark SOC p22810

Figure 7.7 shows the test schedules for the benchmark SOC p22810 [116] with $TR-Architect$ and $mfTAMDesign$ algorithms, assuming fixed-length scan chains, the Test Bus architecture and a total TAM width of 40. With $TR-Architect$ (Figure 7.7(a)) six TAMs are obtained and the maximum test application time is 190995 clock cycles. When using $mfTAMDesign$ with $f_{tam} = f_{ext}$, a more balanced TAM design (less idle time) is achieved (183550 clock cycles) without introducing any hardware overhead (Figure 7.7(b)). *This proves that embedding the proposed multi-frequency TAM exploration in TAM merging engine of $TR-Architect$ [48, 51], savings in test application time can be achieved in the*



(a) TR-Architect (b) mfTAMDesign ($f_{tam} = f_{ext}$)

(c) mfTAMDesign ($f_{tam} \leq f_{ext}$) (d) mfTAMDesign ($f_{tam} \leq 2f_{ext}$)

Figure 7.7: Comparison of TAM Design for Flattened SOC p22810 with $W_{ttl} = 40$.

single frequency case. When additional routing overhead is affordable the test application time can be further decreased to 182678 clock cycles, in which eight TAMs are designed with seven additional virtual TAM lines (Figure 7.7(c)). When the power ratings allow the scan chains to run at $2f_{ext}$, 176810 clock cycles are obtained with only 20 internal virtual TAM lines (Figure 7.7(d)). Frequency converters, implemented as shift registers, are needed for each of the 14 low speed virtual TAM lines working at $f_{ext}/2$ in Figure 7.7(c) and every of the 20 high speed virtual TAM lines working at $2f_{ext}$ in Figure 7.7(d).

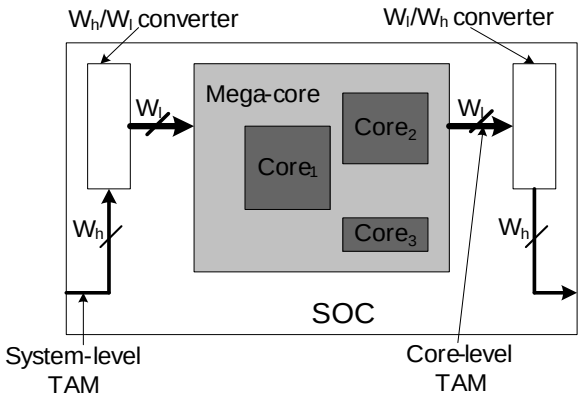


Figure 7.8: Proposed Hardware Architecture for Testing Hard Mega-Cores.

7.3 Multi-Frequency TAM Design for Hierarchical SOCs

In this section we show how multi-frequency test data transportation can facilitate the reuse of hard mega-cores in a non-interactive design transfer model, without a negative impact on test application time. We first examine two types of frequency converters that can match a higher number of core-level TAM lines to a lower number of system-level TAM lines. Then we introduce a new design flow for the system integrator and we illustrate the benefits of the proposed approach on a hierarchical benchmark SOC [116].

7.3.1 Matching the Bandwidth for Hard Mega-Cores

Suppose the core-level TAM width within the mega-core is W_l and it operates at frequency f_l and the external system-level TAM width W_h is assigned to the mega-core, which operates at f_h (not necessarily to be the same as ATE frequency f_{ext}). When $W_h = W_l$, the system-level TAM connects to the core-level TAM directly with the same frequency. If $W_h < W_l$, by introducing W_h/W_l (W_l/W_h) frequency converters on the input (output) of the mega-core, the core-level TAM operates at a lower frequency $f_l = \frac{f_h \times W_h}{W_l}$, as shown in Figure 7.8. Based on the relationship between W_h and W_l, we define the two types of converters used in this chapter as follows:

- *Type I converters* are pairs of shift registers and the corresponding control logic for bandwidth matching. They are used when W_h is a divisor of W_l.

- *Type II converters* are pairs of buffers with depth $\frac{W_l \times W_h}{gdiv}$ and the corresponding control logic for bandwidth matching, where $gdiv$ is the greatest common divisor of W_l and W_h. They are used when W_h is not a divisor of W_l.

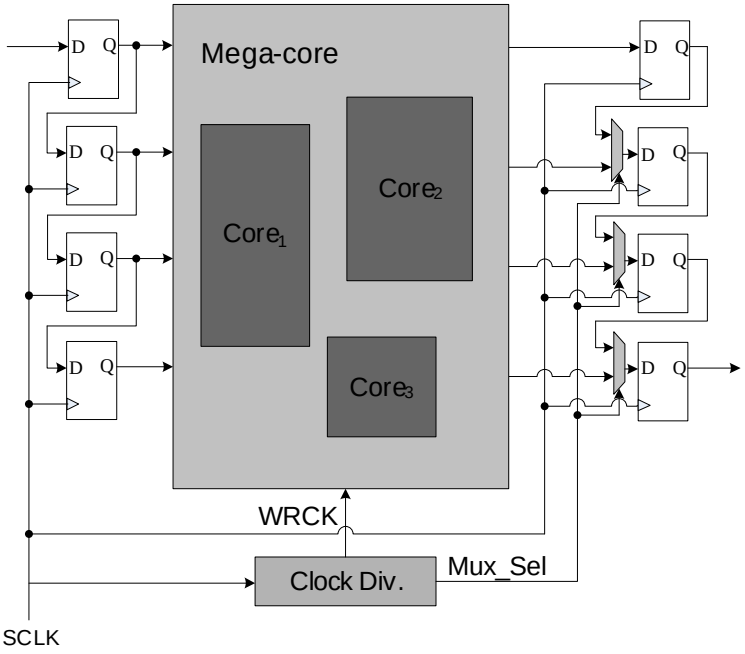
In Example 7.2, we show the implementation of a type I 1/4 (4/1) frequency converter and a type II 3/4 (4/3) frequency converter and analyze the impact to use these frequency converters for SOC testing. It should be noted that, for a general W_h/W_l (W_l/W_h) frequency converter, the hardware implementation is based on the same concepts outlined in the following example.

Example 7.2 *Consider a hard mega-core with core-level TAM width $W_l = 4$. The test application time is T* core clock cycles.

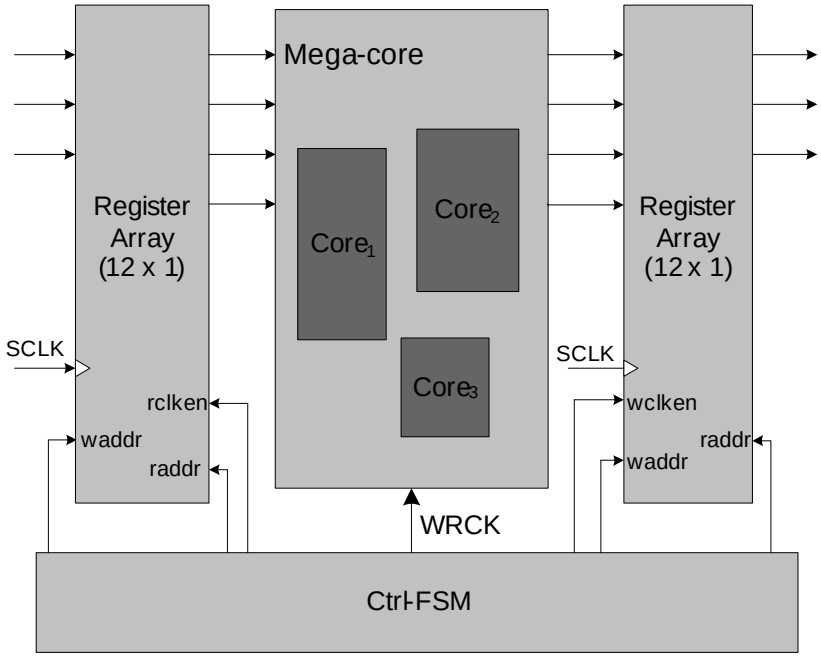
- *If the system-level TAM width assigned to the mega-core is $W_h = 1$, then the type I 1/4 (4/1) frequency converter can be implemented as shown in Figure 7.9(a). For the shift register implementation, only eight flip-flops and a clock division unit are needed to map the 1 system-level TAM line to the 4 core-level TAM lines, in which Clock Div. (see Figure 7.9(a)) generates the mega-core wrapper test clock signal WRCK and also the mux control signal Mux_Sel for the 4/1 frequency converter. The test application time for the mega-core will be $4T$* system clock cycles *(since the core-level TAM operates at $f_h/4$).*

- *If, however, 3 system-level TAM lines are assigned to this mega-core ($W_h = 3$), then the type II 3/4 (4/3) frequency converters can be implemented as shown in Figure 7.9(b). Two 12×1 register array serves to transfer data between the system-level TAM and core-level TAM. The* ctrl-FSM *is a finite state machine that counts the number of writes/reads to/from the buffer and controls the increment, decrement or hold of the buffer address. It also generates the WRCK signal with frequency $f_l = \frac{3}{4} f_h$. The test application time for the mega-core will be $\frac{4}{3}T$ system clock cycles, however, this type II converter takes more area than type I converter.*

For both type I and type II frequency converters, the size of the buffer to map test data from the system-level TAM to the core-level TAM is $S_{buffer} = LCM(W_h, W_l) \times 2$, where

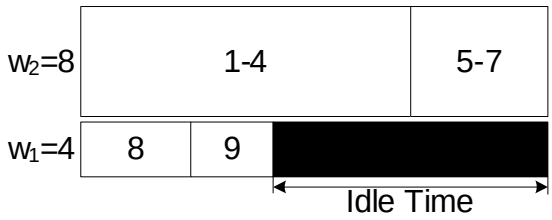


(a) 1/4 and 4/1 type I converters

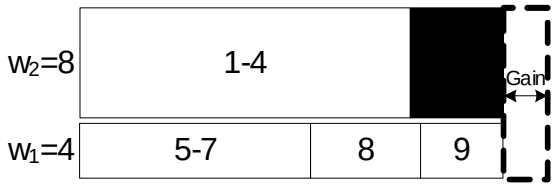


(b) 3/4 and 4/3 type II converters

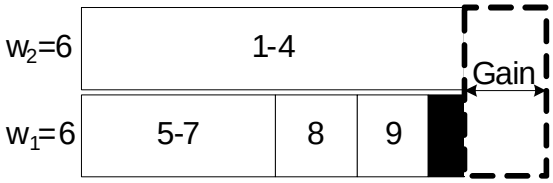
Figure 7.9: Type I and II Frequency Converters Used for Bandwidth Matching.



(a) No bandwidth matching.



(b) Type I converters.



(c) Type II converters.

Figure 7.10: The Benefits of Multi-Frequency TAM Design for Hierarchical SOCs.

W_h is the system-level TAM width assigned to the mega-core, W_l is internal TAM width of the mega-core and *LCM* stands for the least common multiplier. For the control logic part, if ignoring the small combinational logic inside, a type I converter requires $\frac{W_h}{W_l}$ flip-flops, while a type II converter requires $\lceil logS_{buffer} \rceil$ flip-flops, which is generally small when compared to the buffer size. As a result, we use S_{buffer} as the added DFT area constraint of the system-level exploration algorithm proposed later in this section.

Prior to introducing the design flow for the system integrator, we show, using an example, the benefits to use type I and II converters for hierarchical SOC testing.

Example 7.3 *Consider a hypothetical hierarchical SOC having two system-level cores* $Core_8$ *and* $Core_9$*, and two system-level mega-cores* $Core_1$ *and* $Core_5$ *(labeled as 1-4 and*

5-7), as shown in Figure 7.1. Suppose the mega-core provider has implemented "hard" (i.e., non-alterable) TAMs within the mega-cores and the internal TAM width of both $Core_1$ and $Core_5$ is 8. If the system integrator is constrained to use a system-level TAM width of 12, the test schedule using [72] is shown in Figure 7.10(a), where the two mega-cores need to be tested sequentially and compose the bottleneck TAM of the SOC (i.e., the overall test application time is dominated by TAM2). However, if we introduce type I converters next to the I/Os of the mega-core wrappers, in order to divide by 2 the system-level TAM width assigned to mega-core $Core_5$, the overall test application time will be reduced because $Core_5$ is placed in the same TAM group as $Core_8$ and $Core_9$. This case is shown in Figure 7.10(b) and it can be seen that TAM1 has become the bottleneck TAM. If type II converters are used for both $Core_1$ and $Core_5$, the test application time can be further decreased, as shown in Figure 7.10(c), because 6 system-level TAM lines are assigned to each mega-core. However, two pairs of 6/8 and 8/6 type II converters are required, which leads to significantly more area than in the case of Figure 7.10(b), where a single pair of 4/8 and 8/4 type I converters is required.

While it is clear that converters of both types can be employed to reduce test application time when hard mega-cores are used, the question is how much DFT area needs to be introduced? Since type I converters incur less overhead than type II converters, in the following section we introduce a new design flow for the system integrator, which facilitates an easy trade-off between the reduction in test application time and the DFT area necessary to achieve it.

7.3.2 Design Flow for System Integrator

DFT area to implement frequency converters is considered as a constraint to the multi-level TAM design problem, defined next.

Problem $P_{ml-TAM-opt}$: Given the maximum system-level TAM width W_{ttl} for the hierarchical SOC, the DFT area overhead constraint C_{area} to implement frequency converters, the test set parameters for each top-level core, including the number of input terminals, the number of output terminals, the number of test patterns, the number of scan chains and for each scan chain its length (for cores with fixed-length internal scan chains) or the total

number of SFFs (for cores with flexible-length internal scan chains), the test parameters for each "hard" mega-core, including the pre-specified core-level TAM width W_l and its test application time T for its child cores and the mega-core itself, determine the width and shift frequency of each system-level TAM, the wrapper design for each core, the test resources to map the SOC system-level TAM to each core-level TAM within the mega-core, and a test schedule for the entire SOC, such that: (i) the bandwidth of the internal TAM used at any time does not exceed the bandwidth of the ATE; (ii) the total buffer size of all the converters used for mega-cores does not exceed C_{area}; and (iii) the overall SOC test application time is minimized.

The basic intuition behind the proposed solution to design of multi-level TAMs is to trade the test application time and the type and size of the frequency converters used to match the bandwidth between the SOC TAM lines and the hardwired mega-cores' TAM lines. Once the previous step is completed, mega-cores can be treated as regular cores and a multi-frequency variation of $TR-Architect$ [48, 51], described in Section 7.2, is used to solve the flattened SOC TAM problem. In the following we explain the cost model necessary for constrained design space exploration and the proposed design flow for the system integrator.

Cost model for mega-core: By building a cost model for each mega-core using both test application time and hardware cost, the test scheduling algorithm tries to optimize them simultaneously. For each mega-core *i*, the overall test cost is formulated as $C_i = T_i + S_{buffer\_i} \times costWeight$, where T_i is its test application time, S_{buffer_i} is the buffer size necessary to implement frequency conversion and costWeight is a weighting factor that is varied by the system integrator during the solution space exploration.

Design flow for the system integrator: The design flow is shown in Figure 7.3.2. The algorithm starts by allowing both type I and type II frequency converters to be used (line 2). Inside the internal loop (lines 5-23), the algorithm tries to find the schedule with the lowest test application time while fulfilling the area constraint. If the constraint cannot be met, then the algorithm will try again only with type I converters (line 20). If the constraint still cannot be met, there will be no converters that can guarantee that area constraint is met and the design flow will become, in principle, the same as in [72] (line 24-28). Finally, the

Algorithm 7.4 - Design Flow for the System Integrator

1. get the core test parameters for non-hierarchical cores;
2. set $typeIIAllowed = true$; $typeIAllowed = true$;
3. set $isConstraintMet = false$;
4. **while** (!isConstraintMet) {
5. **if** ($typeIAllowed$ **OR** $typeIIAllowed$){ /*Converters are allowed, try different area cost weight w_n*/
6. **for** each iteration n in N iterations{
7. **for** each top-level mega-core i in the SOC {
8. Set $w_n = n \times p$;
9. Get the pre-specified TAM width and test application time;
. /* Let $T_i(w_j)$ and $C_i(w_j)$ be the SOC-level test application time
. and overall cost of Core i with TAM width w_j */
10. Set $T_i(w_j) = T_i$, $C_i(w_j) = T_i$, for $w_j \geq W_i$;
11. **for** ($w_j < W_i$) {
12. **if** ($typeIIAllowed$) { Set $T_i(w_j) = \frac{T_i \times W_i}{w_j}$; Set $C_i(w_j) = T_i(w_j) + S_{buffer\_i} \times w_n$; }
13. **else if** ($typeIAllowed$) { Set $T_i(w_j) = T_i \times N_f$; Set $C_i(w_j) = T_i(w_j) + S_{buffer\_i} \times w_n$; }
14. }
. }
15. partition W_{ttl} system-level TAM among cores using $mfTAMDesign$;
. }
16. **if** (test schedule with $BufferSize \leq C$ exists) {
17. record the test schedule with minimum time;
18. $isConstraintMet = true$;
19. } **else** {
20. $isConstraintMet = false$;
21. if($typeIIAllowed$) $typeIIAllowed = false$;
22. else $typeIAllowed = false$; }
23. } **else** { /*No converters is allowed*/
24. **for** each top-level mega-core i in the SOC {
25. Get the pre-specified TAM width and test application time;
26. Set $T_i(w_j) = T_i$, $C_i(w_j) = T_i$, for $w_j \geq W_i$;
27. **for** ($w_j < W_i$) { set $T_i(w_j) = \infty$; $C_i(w_j) = \infty$; }
. }
28. Partition W_{ttl} system-level TAM among cores using $mfTAMDesign$;
. }
. }
29. Implement system-level TAM architecture;

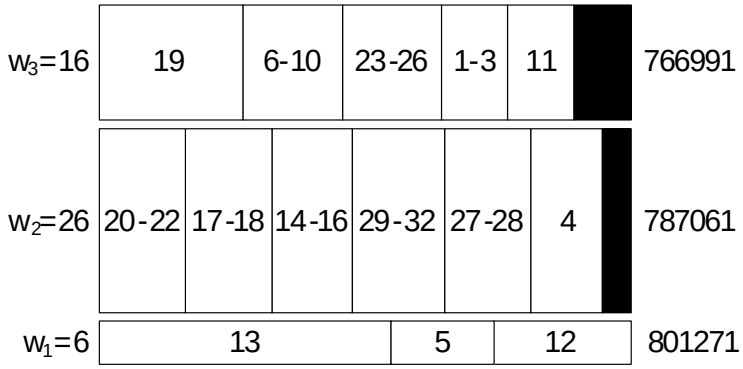
Figure 7.11: Design Flow for Testing SOCs with Multiple levels of Hierarchy.

system-level TAM architecture which can meet the area constraint C_{area} with the lowest test application time is implemented (line 29). It is important to note that even when DFT area constraints are tight, the test scheduling flexibility leads to savings in test application time, as shown in our experiments.

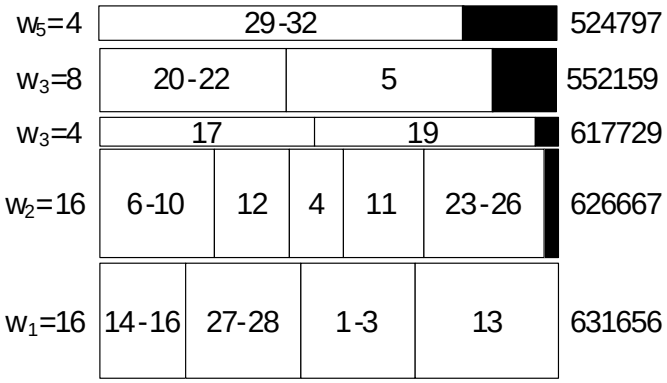
In the following, we place the focus on the internal loop of the algorithm where the frequency converters are used (line 5-23). Since the internal core-level TAM width W_i cannot be changed, if $w_j \geq W_i$, w_j of the system-level TAM lines connect to the core-level TAM lines directly and hence the test application time equals T_i (line 10). When $w_j < W_i$ the test application time of the mega-core is dependent on what type of test resources the system integrator can introduce (line 12-13). Note, if type I converter is introduced, only w of w_j system-level TAM lines can be mapped to the core-level TAM lines, where w is the greatest factor of W_i less than w_j ($N_f = W_i/w$). After we get the information on TAM width/test application time pair for each core or mega-core, in line 18 we partition the system-level TAM using the flattened TAM optimization algorithm *mfTAMDesign*, which was described in the previous section. The cost weight w_n in the overall cost is varied several times in order to explore a high number of solutions (line 6-13). w_n is computed in line 8, in which p is a scaling factor used to avoid the usually large difference between the buffer size measured in flip-flop count and the test application time measured in clock cycles. For the results reported in this chapter, we have mostly used $p = 50$ and the number of iterations is $N = 20$, which gives good results with computation times in the range of seconds.

7.3.3 A Case Study for Benchmark SOC p93791

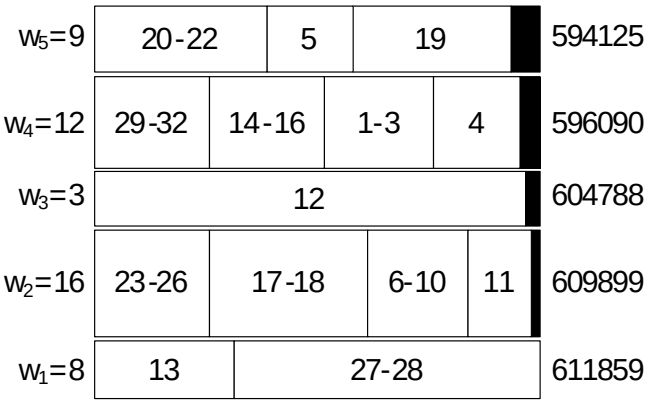
Figure 7.12 shows the test schedule for the hierarchical benchmark SOC p93791 with total system-level TAM width of 48 and when the core-level TAM width for each mega-core is fixed at 16. When no frequency converters are implemented, the test application time is 801271 clock cycles, as shown in Figure 7.12(a). Although the idle time does not seem to be large, since mega-cores 14, 17, 20, 27 and 29 are on the TAM with width 26, only 16 system-level TAMs connect to their core-level TAM (while the other 10 system-level TAM lines are wasted, which is obviously inefficient). When the area constraint is set as



(a) No bandwidth matching.



(b) Type I converters.



(c) Type II converters.

Figure 7.12: Comparison of Multi-Level TAM Design for SOC p93791 with $W_{ttl} = 48$.

100, type I converter for mega-cores 20 and 29 is used and the test application time is reduced to 631656 clock cycles. Frequency conversion is achieved using a shift register implementation, which needs 64 flip-flops for the buffer and some control logic. When type II converters for mega-cores 1, 14, 20 and 29, and type I converter for mega-cores 27 are employed, the test application time can be further decreased to 611859 clock cycles, however, with a larger hardware cost. In total 608 flip-flops are required for the buffers to implement bandwidth matching for the entire SOC. It should be noted that this SOC has a large number of mega-cores when compared to other designs, which is the main source of the increased test area caused by type II converters. Since, in principle, a mega-core is supposed to be large and the top-level SOC that includes it should be even larger, we consider that the area penalty introduced by the type II converters can be acceptable, especially when ATE buffer depth is low and reductions in test application time are essential to avoid ATE buffer reloading.

7.4 Experimental Results

To investigate the implication of the proposed solution on the trade-off between test application time and DFT area overhead, benchmark SOCs from the ITC'02 SOC test benchmarking initiative [115, 116], again, are used in our experiment. We first show the savings in test application time with multi-frequency TAM design in Section 7.4.1 for the flattened version of the benchmark SOCs. Next, we present the benefits of the proposed multi-level TAM design for four hierarchical benchmark SOCs.

7.4.1 Experiment 1: Testing Time for Flattened SOCs

In this experiment, we have used the flattened versions, i.e., the hierarchy has not been taken into consideration, of the benchmark SOCs p22810, p34392 and p93791 from the ITC'02 benchmark suite [115]. For TestRail architecture, assuming fixed-length scan chains, we compare the test application time of *mfTAMDesign* with the serial schedule from TR-Architect [48, 51]. For Test Bus architecture, assuming fixed-length scan chains, we compare the results of *mfTAMDesign* with six representative approaches: (1) the generalized

SOC	W_{ttl}	$f_{tam} = f_{ext}$		$f_{tam} \leq f_{ext}$		$f_{tam} \leq 2f_{ext}$	
		TR-Architect [48]	$mfTAM$	$mfTAM_l$	W_{vt-l}	$mfTAM_h$	W_{vt-h}
p22810	16	476301	**454443**	**437976**	**23**	**437078**	**11**
	24	310249	310249	**308117**	**32**	**295793**	**15**
	32	226640	**223466**	223466	32	**218988**	**23**
	40	190995	190995	**186123**	**45**	**177454**	**20**
	48	160221	160221	160221	48	**154059**	**32**
	56	145417	145417	145417	56	**128492**	**28**
	64	133405	133405	133405	64	**111733**	**32**
p34392	16	1032049	**1010821**	1010821	16	**975968**	**12**
	24	721244	**701838**	**666908**	**33**	**653204**	**15**
	32	552746	**551249**	**544579**	**33**	**505411**	**16**
	40	544579	544579	544579	33	**430019**	**20**
	48	544579	544579	544579	33	**333454**	**33**
	56	544579	544579	544579	33	**293805**	**30**
	64	544579	544579	544579	33	**272290**	**33**
p93791	16	1853402	**1824376**	1824376	16	**1783085**	**8**
	24	1240305	**1210081**	1210081	24	**1196230**	**15**
	32	940745	940745	**906878**	**37**	906878	37
	40	786608	**726616**	**721859**	**42**	721859	42
	48	628977	**611730**	611730	48	**605041**	**24**
	56	530059	530059	530059	56	**522527**	**28**
	64	461128	461128	**455738**	95	**453439**	37

Table 7.1: TAT Comparison for TestRail Architecture with Fixed-Length Scan Chains.

rectangle-packing (GRP) from [76], (2) the simulated annealing algorithm from [181], (3) ILP and exhaustive enumeration from [73], (4) the heuristic Par eval from [74], (5) the Lagrange multipliers from [154] and (6) TR-Architect from [48, 51]. The first two methods used flexible-width TAM architecture, while the others employed fixed-width TAM architecture. For flexible-length scan chains we compare our results against [48, 51]. Test application time is given in ATE clock cycles (note, when $f_{tam} \geq f_{ext}$ the result may need to be rounded up to an integer).

Tables 7.1, 7.2, 7.3, and 7.4 present experimental results for SOCs with fixed-length scan chains for the TestRail architecture, fixed-length scan chains for the Test Bus architecture, flexible-length scan chains for the TestRail architecture, and flexible-length scan

SOC	W_{ttl}	$f_{tam} = f_{ext}$							$f_{tam} \leq f_{ext}$		$f_{tam} \leq 2f_{ext}$	
		Flexible-width		**Fixed-width**								
		GRP [76]	**SA_2** [181]	**ILP/enum** [73]	**Par eval** [74]	**Lagrange** [154]	**TR-Architect** [48]	$mfTAM$	$mfTAM_l$	W_{vt-l}	$mfTAM_h$	W_{vt-h}
p22810	16	489192	438619	462210	468011	434922	458068	**433485**	**430052**	**23**	**427119**	**8**
	24	330016	*293019*	361571	313607	313607	299718	**296365**	296365	24	**287327**	**15**
	32	245718	219923	312659	246332	245622	222471	**218035**	218035	32	**215026**	**23**
	40	199558	*180004*	278359	232049	194193	190995	**183550**	**182678**	**47**	**176810**	**20**
	48	173705	*151886*	278359	232049	164755	160221	**155851**	155851	48	**148183**	**24**
	56	157159	*132812*	268472	153990	**145417**	**145417**	**145417**	145417	56	**126487**	**28**
	64	142342	*112515*	260638	153990	133628	**133405**	**133405**	**128706**	**71**	**109018**	**32**
p34392	16	1053491	*965252*	**998733**	1033210	1021510	1010821	1006155	1006155	16	**975338**	**8**
	24	759427	*657561*	720858	882182	729864	680411	**663193**	663193	24	**646382**	**15**
	32	544579	544579	591027	663193	630934	551778	**544579**	544579	32	**503078**	**16**
	40	544579	544579	**544579**	**544579**	**544579**	**544579**	**544579**	544579	32	**392441**	**20**
	48	544579	544579	**544579**	**544579**	**544579**	**544579**	**544579**	544579	32	**331597**	**24**
	56	544579	544579	**544579**	**544579**	**544579**	**544579**	**544579**	544579	32	**292262**	**30**
	64	544579	544579	**544579**	**544579**	**544579**	**544579**	**544579**	544579	32	**272290**	**32**
p93791	16	1932331	*1765797*	**1771720**	1786200	1775586	1791638	1791638	1791638	16	**1759710**	**8**
	24	1310841	*1178397*	1187990	1209420	1198110	**1185434**	**1185434**	1185434	24	**1177054**	**12**
	32	988039	893892	**887751**	894342	936081	912233	906878	902716	42	895819	16
	40	794027	718005	**698583**	741965	734085	718005	718005	718005	40	706798	20
	48	669196	*597182*	**599373**	**599373**	**599373**	601450	601450	601450	48	**592717**	**24**
	56	568436	*510516*	514688	514688	514688	528925	**513564**	513564	56	**511158**	**28**
	64	517958	*451472*	460328	473997	472388	**455738**	**455738**	455738	64	**451358**	**42**

Table 7.2: TAT Comparison for Test Bus architecture with Fixed-Length Scan Chains.

SOC	W_{max}	$f_{tam} = f_{ext}$		$f_{tam} \leq f_{ext}$		$f_{tam} \leq 2f_{ext}$	
		TR-Architect	mfTAM	$mfTAM_l$	W_{vt-l}	$mfTAM_h$	W_{vt-h}
p22810	16	448179	448179	**436281**	**20**	**431242**	**11**
	32	223368	**220591**	**218084**	**33**	218084	33
	48	150455	150455	**146706**	**70**	146706	70
	64	112695	**110922**	**110922**	**64**	**109042**	**33**
p34392	16	1003345	1003345	1003345	16	**986688**	**8**
	32	505783	**490246**	490246	24	490246	24
	48	347948	347948	**327670**	**52**	**326411**	**31**
	64	246755	246755	**245776**	**95**	**245123**	**32**
p93791	16	1821818	**1819559**	1819559	16	**1783085**	**8**
	32	946311	**919466**	**895027**	**46**	895027	46
	48	599263	**596224**	**595703**	**69**	**593039**	**27**
	64	451094	451094	**444272**	**75**	444272	75

Table 7.3: TAT Comparison for TestRail Architecture with Flexible-Length Scan Chains.

SOC	W_{max}	$f_{tam} = f_{ext}$		$f_{tam} \leq f_{ext}$		$f_{tam} \leq 2f_{ext}$	
		TR-Architect	mfTAM	$mfTAM_l$	W_{vt-l}	$mfTAM_h$	W_{vt-h}
p22810	16	431331	**428303**	**427466**	**19**	**424493**	**8**
	32	218982	**215461**	**215397**	**33**	**213733**	**19**
	48	148766	**144809**	144809	48	**143002**	**24**
	64	110922	**109755**	109755	64	**107699**	**33**
p34392	16	971816	**964488**	**962976**	**22**	**945162**	**11**
	32	499083	499083	**483402**	**43**	**481488**	**22**
	48	326400	326400	**325472**	**64**	**325255**	**36**
	64	241254	241254	241254	64	241254	64
p93791	16	1757602	1757602	1757602	16	**1755868**	**8**
	32	885071	**882713**	882713	32	**878801**	**16**
	48	587755	**587571**	587571	48	**586631**	**24**
	64	444195	**441874**	**441560**	**94**	**441357**	**32**

Table 7.4: TAT Comparison for Test Bus architecture with Flexible-Length Scan Chains.

chains for the Test Bus architecture, respectively. For each of the three SOCs, for a maximum value of TAM width W_{ttl}, we show the test application time obtained by previous approaches and the test application time obtained by *mfTAMDesign* when (1) all the internal TAMs run at the frequency $f_{tam} = f_{ext}$ (2) the TAMs can run at $f_{tam} \leq f_{ext}$ and (3) the TAMs can run at the frequency $f_{tam} \leq 2f_{ext}$ (note that TAMs can also run at a frequency lower than f_{ext} in this situation). We also give the virtual TAM width used inside the SOC

when frequency conversion is used (W_{vt-l} and W_{vt-h}). We select $layoutConstraint = 1.5$, i.e., to keep routing overhead under control the internal virtual TAM wires should not exceed the ATE channels by 50%.

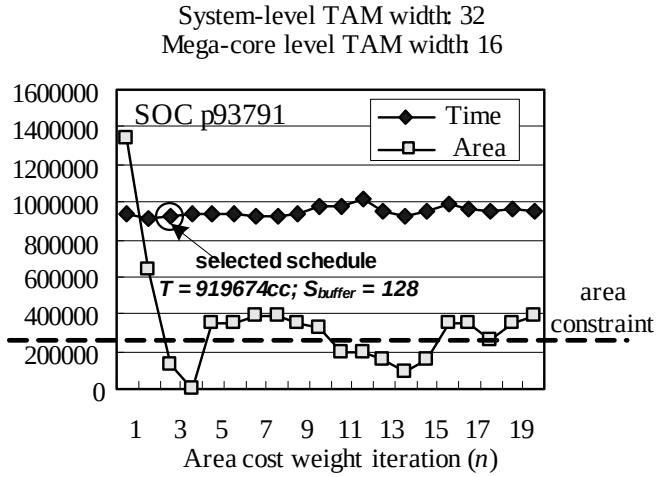
When $f_{tam} = f_{ext}$, no additional virtual TAMs are required and hence no additional hardware overhead is introduced. This means that although the algorithm starts the optimization with a large number of virtual TAM wires at a lower frequency, after merging multi-frequency TAMs and distributing the freed virtual TAM wires, all the TAMs are finally designed to operate at the same frequency (f_{ext}). The test application time is reduced in several cases, which is due to a larger solution space exploration. This is achieved with a computation time of at most a few seconds, which is unlike the ILP method [73] that requires hours for large SOCs. It is important to mention that in this case, since $f_{tam} = f_{ext}$ and no additional hardware needs to be introduced, the test application time can be improved with no penalty in area or power. It should also be noted that, although the proposed multi-frequency TAM design algorithm builds on the search engine of TR-Architect [48, 51], the basic principle used to expand the available TAM lines to a larger number of slower virtual TAMs and then merge them in order to explore a larger solution space, is also applicable to other design space exploration strategies for fixed-width TAM architectures. When $f_{tam} = f_{ext}$, as it can be observed in Table 7.2, the results from [181] are the only ones which are slightly better than the proposed solution in several cases. However, this approach uses a flexible-width Test Bus architecture, which cannot be easily extended to the proposed modular multi-frequency TAM architecture, whose benefits in further reducing test application time, when f_{ext} is lower or higher than f_{tam}, are discussed next.

When $f_{tam} \leq f_{ext}$, the result is improved in several cases, however, at the expense of additional virtual TAM wires inside the SOC ($W_{vt-l} > W_{ttl}$). When $f_{tam} \leq 2 f_{ext}$, it can be seen in most cases a lower test application time can be achieved with a small number of virtual TAM wires ($W_{vt-h} < W_{ttl}$) running at higher frequency than f_{ext}. As long as power ratings allow it, this technique is particularly attractive when one core becomes the bottleneck and increasing TAM width will not result in any improvement. For example, for SOC p34392, assuming fixed-length scan chains, when $W_{tam} \geq 32$ (for Test Bus architecture) or $W_{tam} \geq 33$ (for TestRail architecture), the test application time does not decrease any more. If the additional TAM wires are used to combine to a lower number of virtual TAM wires

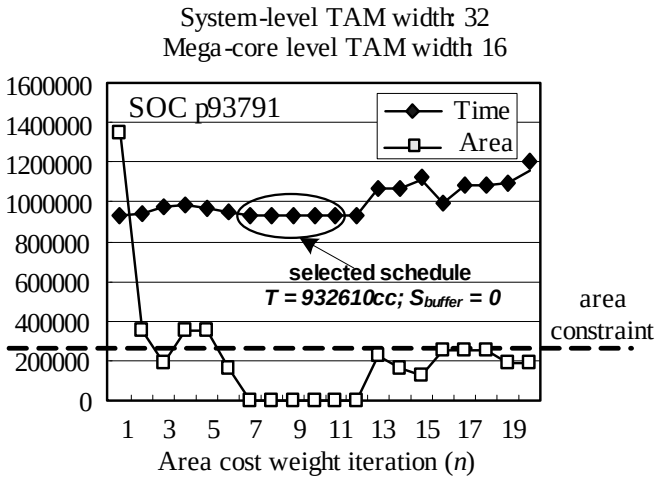
running at higher frequency (within the power rating of the SOC), the test application time can be significantly decreased for TAM widths from 33 to 64. As shown in Tables 7.1, 7.2, 7.3, and 7.4, significant savings are achieved only for the case when the TAM frequency is greater than the ATE frequency ($f_{tam} \leq 2 f_{ext}$). However, it is important to note that there are several examples where the results for $f_{tam} \leq f_{ext}$ and $f_{tam} = f_{ext}$ outperform the current methods (especially when considering the fact that, given the same constraints, the experimental results do not vary significantly for the solutions reported in Tables 7.1, 7.2 , 7.3, and 7.4).

7.4.2 Experiment 2: Testing Time for Hierarchical SOCs

In this section we show the advantages of using frequency converters for hierarchical SOCs with hard mega-cores. Before comparing the proposed multi-level TAM design method against the only existing approach [72] that tackled the same problem, we illustrate the importance of considering the DFT area overhead in the cost function used for guiding the design space exploration. Figure 7.13 shows the variation of test application time and DFT area, caused by frequency converters, for different area cost weights $costWeight = n \times p$. Based on the value of the scaling factor p, the area cost weight varies from 0 to 950 with a interval value of 50 in Figure 7.13(a); while in Figure 7.13(b), it varies from 0 to 4750 with a interval value of 250. The area is scaled as $S_{buffer} \times 1000$ to be seen clearly in the figure. We can observe that when area cost is not taken into account ($n = 0$), the added DFT area for frequency converters is quite large and hence this solution is not preferable. We can also observe an increase in SOC testing time when the area cost weight is high ($n \geq 12$ in Figure 7.13(b)), which is not desirable either. Therefore, the introduction of the area cost into the scheduling algorithm helps the system integrator to keep the added area within predefined limits, while still minimizing the test application time (whose variation is not as large as the variation in DFT area). However, *due to the nature of the heuristic approach*, the SOC testing time is not monotonic increasing and the total DFT area cost is not monotonic decreasing with the increase of the area cost weight. Nevertheless, if the system integrator provides an area constraint (shown in the figure using the dotted line), the proposed Algorithm 7.4 (shown in Figure 7.3.2) from Section 7.3.2 will select



(a) $p = 50$



(b) $p = 250$

Figure 7.13: Test Application Time and DFT area for p93791 with Different Area Cost Weights.

the solution with the minimum test application time that fulfills the given area constraint. In this experiment for SOC p93791, when the scaling factor is $p = 50$, we obtain a better result in terms of test application time when compared to $p = 250$.

In Tables 7.5 and 7.6 we compare the results of our solution with the results from [72] for non-interactive design transfer model. The reason we consider only the non-interactive

model is that the interactive model assumes that TAM design for mega-cores is flexible, which conflicts with the objective to reuse hard mega-cores. Our results indicate that as long as the system integrator accepts additional test area, savings can be achieved in test application time without any internal modification to the hard mega-cores. In this experiment, core-level TAM widths supplied to each mega-core before system-level TAM design have the same configurations as in [72]. That is, in Table 7.5 for p22810 and a586710 we have $W_l = 8$ bits, while in Table 7.6 for p34392 and p93791 we have $W_l = 16$ bits. In addition, the scaling factor is $p = 50$. T, T_ϕ, T_I and T_{II} denote the test application time from [72], the test application time obtained by using the *mfTAMdesign* algorithm, the test application time obtained by Algorithm 7.4 from Section 7.3.2 when the area constraint for frequency converters is stringent (50 for p22810 and a587610, 100 for p34392 and p93791), and when the area constraint for frequency converters is relaxed (500 for p22810 and a587610, 1000 for p34392 and p93791), respectively. The percentage change is calculated as $\Delta T_\phi = \frac{T_\phi - T}{T}$, $\Delta T_I = \frac{T_I - T_\phi}{T_\phi}$ and $\Delta T_{II} = \frac{T_{II} - T_\phi}{T_\phi}$ respectively. To provide a fair comparison and prove the benefits caused exclusively by the use of Algorithm 7.4 (*which is orthogonal to any single-level TAM design algorithm*), we report ΔT_I and ΔT_{II} with respect to T_ϕ instead of with respect to T [72].

For T_ϕ, we have the same hierarchical design flow as in [72]. It can be seen in most of the cases $T_\phi < T$, however, because the TAM width constraint for mega-cores restricts the *CreateStartSolution* step in $TR-Architect$ [48, 51] (used in *mfTAMDesign*), in a few cases the algorithm from [72] gives better results. It is worth noting that both methods cannot provide a solution for SOC p34392 and p93791 when the system-level TAM width is only 8 because the system-level TAM is not wide enough to fork out to the core-level TAM ($W_l = 16$). By introducing the frequency converter next to the wrapper of the mega-core, the system integrator can afford narrow system-level TAM design. In almost all cases the test application time is decreased when frequency converters can be used to map system-level TAM to core-level TAM, and, when the area constraint is relaxed, the test application time is further decreased (i.e., $T_{II} \leq T_I \leq T_\phi$). This improvement is due to the greater flexibility (the system integrator has more choices for the system-level TAM width assignment to the mega-core) during test scheduling and because a larger solution space can be explored. It is important to note, for SOC p93791 when $W_{tam} = 64$, $T_I = T_{II} < T_\phi$ can be acquired without

SOC	W_{tam}	T([72])	T_ϕ	ΔT_ϕ(%)	$T_I(S_{buffer} \leq 50)$	S_{buffer}	ΔT_I (%)	$T_{II}(S_{buffer} \leq 500)$	S_{buffer}	ΔT_{II} (%)
p22810	8	-	992297	-	890905	32	-10.22	870622	128	-12.26
	16	505858	496804	-1.79	474023	16	-4.59	446383	96	-10.15
	24	412682	353619	-14.31	322370	48	-8.84	317063	96	-10.34
	32	396473	280634	-29.22	280634	0	0	280634	0	0
	40	366260	280634	-23.38	280634	0	0	280634	0	0
	48	366260	280634	-23.38	280634	0	0	280634	0	0
	64	366260	280634	-23.38	280634	0	0	280634	0	0
a586710	8	-	7.95×10^7	-	7.95×10^7	0	0	7.95×10^7	0	0
	16	-	5.27×10^7	-	4.32×10^7	16	-18.03	4.21×10^7	80	-20.11
	24	3.06×10^7	3.06×10^7	0	2.87×10^7	16	-6.21	2.87×10^7	16	-6.21
	32	2.88×10^7	2.19×10^7	-23.96	2.19×10^7	0	0	2.19×10^7	0	0
	40	2.50×10^7	1.91×10^7	-23.60	1.91×10^7	0	0	1.91×10^7	0	0
	48	2.14×10^7	1.53×10^7	-28.50	1.53×10^7	0	0	1.53×10^7	0	0
	64	2.14×10^7	1.41×10^7	-34.11	1.41×10^7	0	0	1.41×10^7	0	0

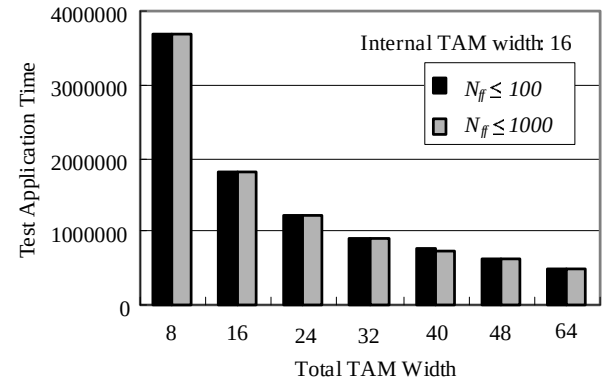
Table 7.5: Test Application Time for Hierarchical SOCs p22810 and a586710.

SOC	W_{tam}	T([72])	T_ϕ	$\Delta T_\phi(\%)$	$T_I(S_{buffer} \leq 100)$	S_{buffer}	ΔT_I (%)	$T_{II}(S_{buffer} \leq 1000)$	S_{buffer}	ΔT_{II} (%)
p34392	8	x	x	x	2281733	96	x	2224022	416	x
	16	-	1467705	-	1226208	96	-16.45	1156969	672	-21.17
	24	1347023	1467705	+8.96	834337	96	-43.15	781906	608	-46.73
	32	788873	776537	-1.56	691168	96	-10.99	626821	736	-19.28
	40	728426	776537	+6.60	606261	32	-21.93	606261	32	-21.93
	48	618597	606261	-1.99	606261	0	0	606261	0	0
	64	618597	606261	-1.99	606261	0	0	606261	0	0
p93791	8	x	x	x	3697813	64	x	3701007	576	x
	16	2044124	1865140	-8.76	1865140	0	0	1804088	256	-3.27
	24	1351710	1486628	+9.98	1240103	96	-16.58	1198247	768	-19.40
	32	1087300	1486628	+36.73	921805	96	-37.99	913078	640	-38.58
	40	839796	801271	-4.59	756160	64	-5.63	729651	896	-8.94
	48	839796	801271	-4.59	631656	64	-21.17	611859	608	-23.64
	64	839796	553775	-34.06	473503	0	-14.50	473503	0	-14.50

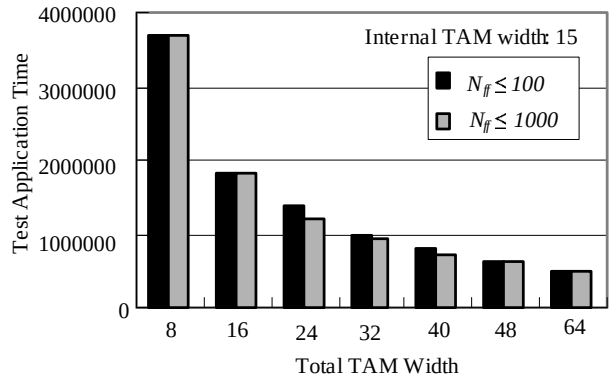
Table 7.6: Test Application Time for Hierarchical SOCs p34392 and p93791.

introducing converters. This is also because the algorithm searches a larger solution space and happens to grant all the mega-cores 16 TAM lines. Note, despite exploring a larger solution space, the computation time is still at most within seconds, which demonstrates that the proposed solution will not have any impact on test development time. For SOC p22810 when $W_{ttl} \geq 32$ and for p34392 when $W_{ttl} \geq 40$, one of the mega-cores inside the SOC becomes the bottleneck TAM on its own (mega-core 1 for p22810, mega-Core 18 for p34392). Since the system-level TAM width assigned to the mega-core already exceeds the internal core-level TAM width, the test application time for the entire SOC cannot be decreased anymore. As a result, a total system-level TAM width of 32 for p22810 and 40 for p34392 would be an effective choice for the system integrator.

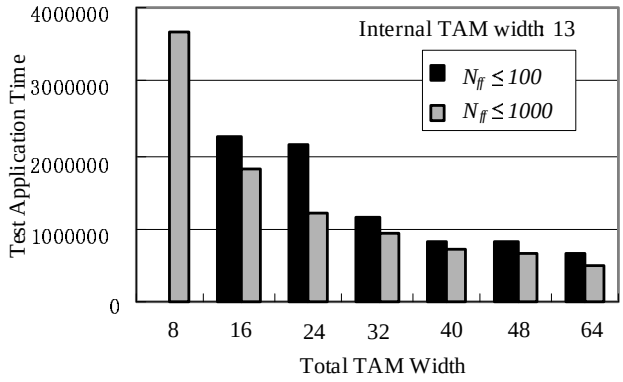
Figure 7.14 shows the influence of the internal TAM width of the mega-cores on the test schedule. In the three sub-figures, the internal TAM widths of the all the mega-cores in SOC p93791 are set as 16, 15 and 13, respectively. When the area constraint is set to 1000, type II converters can be used and each mega-core in all cases can be assigned to any arbitrary TAM width. When the area constraint is set as 100, however, because of the large size of type II converter, only the type I converter is available for matching the TAM widths. When the internal TAM width is 16, we have 5 choices for the TAM width assigned to the mega-cores (1, 2, 4, 8 and 16). When the internal TAM width is 15, we have four choices for the TAM width assigned to the mega-cores (1, 3, 5 and 15). When the internal TAM width is 13, we only have two choices for the TAM width assigned to the mega-cores (1 and 13). With the decreasing number of options for TAM widths, the solution space that the scheduling algorithm can exploit is smaller and hence it leads to solutions with longer test application time. As it can be seen in Figure 7.14, the difference between the two area constraints is much larger when the internal TAM width is set as 13. It should also be noted that we cannot even find a solution to fulfill the constraint $S_{buffer} \leq 100$ when the total TAM width is 8.



(a) Internal TAM width 16



(b) Internal TAM width 15



(c) Internal TAM width 13

Figure 7.14: Test Application Time for p93791 with Different Internal TAM Widths.

7.5 Concluding Remarks

This chapter has introduced a new method, based on the bandwidth matching technique, for designing multi-frequency TAMs for modular hierarchical SOC testing. Using experimental results it was shown that with limited area overhead we can reuse hard mega-cores and achieve reduced test application time compared to prior work. The design space exploration framework described in this chapter, can be used to rapidly explore the test application time/DFT area overhead trade-offs for hierarchical SOCs, and recommend the cost-effective solutions to the system integrator.

Chapter 8

Conclusion

This dissertation tackles several emerging test challenges for state-of-the-art core-based SOC designs, which are not addressed by prior work because of their over-simplified assumptions. An analysis of the contributions presented in this dissertation is given next.

In Chapter 4, we proposed a novel wrapper architecture for testing embedded cores with multiple clock domains. By introducing limited DFT logic, the proposed MFCW can synchronize the external low-speed tester channels with the core's internal scan chains in the shift mode, and provide at-speed test control in the capture mode, thus avoiding test data corruption caused by clock skew. ILP formulation and efficient heuristics were also presented to optimize the proposed MFCW architecture in terms of test application time under average power constraints, which enables system integrators to trade-off the testing time, routing overhead and test power consumption when selecting their test strategies.

Chapter 5 introduced a novel Producer-CUT test architecture for broadside two-pattern test of core-based SOCs. Without using enhanced WIC design, this new architecture achieves the same fault coverage by combining dedicated bus-based TAM and functional interconnects for test data transfer, thus providing full controllability of the IEEE 1500 standard WICs in the two consecutive clock cycles required by two-pattern test. Since the WOCs of producer cores become shared test resources, extra test conflicts will be introduced. To alleviate its negative impact on SOC testing time, new algorithms for test access mechanism design and test scheduling are also presented to optimize the proposed Producer-CUT test architecture.

Motivated by the fact that in practice not all embedded cores are 1500-wrapped in order to meet the design's area constraint and performance requirement, in Chapter 6, we addressed the problem of effectively and efficiently testing SOCs containing light-wrapped cores. When the number of such unwrapped logic blocks is large, we proposed a novel $Producer - CUT - Consumer$ test architecture, similar to the one presented in Chapter 5. When the number of light-wrapped cores is comparably small, we showed how to adapt the TestRail architecture to access the internal SFFs of light-wrapped cores in the Parallel ExTest mode. Efficient and effective TAM design and test scheduling algorithms have been presented for both architectures, which facilitate rapid and concurrent test of 1500-wrapped cores and light-wrapped cores.

Finally, in Chapter 7, a new framework for the design and optimization of hierarchical SOC test architectures, based on the bandwidth matching technique, was presented. We first introduced a new multi-frequency TAM design algorithm for flattened SOCs. Then, we extended the multi-frequency concepts to a multi-level TAM design algorithm for hierarchical SOCs containing hard mega-cores, by introducing two types of frequency converters used to match a higher number of core-level TAM lines to a lower number of system-level TAM lines. A new design flow has been proposed, which, in contrast to [72], enables the system integrator to trade the DFT area against savings in test application time.

There are several important topics for future work. Both test architecture optimization and TDC techniques affect the test application time of the SOC. Therefore, to further reduce the cost of SOC testing, these two tasks should be considered simultaneously. For example, a higher compression ratio of a core test will require less TAM bandwidth. If test architecture optimization is applied to the uncompressed test sets, the core will be given a higher TAM width than needed and hence an inefficient test architecture will be derived. There can be three methods to combine decompression logic (i.e., decoder) with TAM design: decoder-per-SOC, decoder-per-TAM and decoder-per-core, in which decoder-per-TAM is only applicable for fixed-width Test Bus architectures. The system integrator needs to trade-off the DFT area and test application time within these three scenarios. Since different core tests typically have different test compression ratios, the decoder-per-core strategy will achieve the best overall test data reduction, however, at the cost of the largest DFT

overhead. Decoder-per-SOC, on the contrary, will have the least DFT area overhead, however, larger test data volume. Decoder-per-TAM is in the middle of the above two strategies. Despite the intuitive analysis, how to efficiently combine the test architecture optimization with any of the three scenarios is still an open issue and needs further investigation. In addition, future challenges include also how to effectively and efficiently test high-speed interconnect logic for signal integrity and how to reuse/adapt existing DFT logic for silicon debug and diagnosis.

As a final remark, it is anticipated that the methods and findings reported in this dissertation will contribute to the future generation of SOC devices by enabling an improved defect screening process and by lowering the cost of manufacturing test.

Bibliography

[1] M. Abramovici, M. Breuer, and A. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[2] J. Aerts and E. J. Marinissen. Scan Chain Design for Test Time Reduction in Core-Based ICs. In *Proceedings IEEE International Test Conference (ITC)*, pages 448–457, Washington, DC, Oct. 1998.

[3] V. D. Agrawal. Editorial - Special Issue on Partial Scan Design. *Journal of Electronic Testing: Theory and Applications*, 7(5):5–6, Aug. 1995.

[4] P. Bardell, W. McAnney, and J. Savir. *Built-In Self Test - Pseudorandom Techniques*. John Wiley & Sons, 1986.

[5] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann. OPMISR: The Foundation for Compressed ATPG Vectors. In *Proceedings IEEE International Test Conference (ITC)*, pages 748 – 757, Nov. 2001.

[6] I. Bayraktaroglu and A. Orailoglu. Test Volume and Application Time Reduction Through Scan Chain Concealment. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 151–155, 2001.

[7] F. Beenker, K. van Eerdewijk, R. Gerritsen, F. Peacock, and M. van der Star. Macro Testing: Unifying IC and Board Test. *IEEE Design & Test of Computers*, Vol. 3(No. 4):26–32, Dec. 1986.

[8] M. Benabdenbi, W. Maroufi, and M. Marzouki. CAS-BUS: A Test Access Mechanism and a Toolbox Environment for Core-Based System Chip Testing. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):455–473, Aug. 2002.

[9] A. Benso, S. D. Carlo, P. Prinetto, and Y. Zorian. A Hierarchical Infrastructure for SoC Test Management. In *IEEE Design & Test of Computers*, pages 32–39, July 2003.

[10] S. Bhatia, T. Gheewala, and P. Varma. A Unifying Methodology for Intellectual Property and Custom Logic Testing. In *Proceedings IEEE International Test Conference (ITC)*, pages 639–648, Washington, DC, Oct. 1996.

[11] S. Bhawmik. Method and Apparatus for Built-in Self-Test with Multiple Clock Circuits. U.S. Patent 5680543, Lucent Technologies Inc., Murray Hill, N.J., Oct. 21, 1997.

[12] F. Bouwman, S. Oostdijk, R. Stans, B. Bennetts, and F. Beenker. Macro Testability; The Results of Production Device Applications. In *Proceedings IEEE International Test Conference (ITC)*, pages 232–241, Sept. 1992.

[13] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing*. Kluwer Academic Publishers, 2000.

[14] G. R. Case. Analysis of Actual Fault Mechanism in CMOS Logic Gates. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 265–270.

[15] K. Chakrabarty. Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming. *IEEE Transactions on Computer-Aided Design*, 19(10):1163–1174, Oct. 2000.

[16] K. Chakrabarty. Optimal Test Access Architectures for System-on-a-Chip. *ACM Transactions on Design Automation of Electronic Systems*, 6(1):26–49, Jan. 2001.

[17] K. Chakrabarty, V. Iyengar, and A. Chandra. *Test Resource Partitioning for System-on-a-Chip*. Kluwer Academic Publishers, 2002.

[18] K. Chakrabarty, R. Mukherjee, and A. S. Exnicios. Synthesis of Transparent Circuits for Hierarchical and System-on-a-Chip Test. In *Proceedings IEEE International Conference on VLSI Design (ICVD)*, pages 431–436, Bangalore, India, Jan. 2001.

[19] A. Chandra and K. Chakrabarty. Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 114–121, 2001.

[20] A. Chandra and K. Chakrabarty. System-on-a-chip Test Data Compression and Decompression Architectures Based on Golomb Codes. *IEEE Transactions on Computer-Aided Design*, 30(3):355–368, Mar. 2001.

[21] A. Chandra and K. Chakrabarty. Test Resource Partitioning for SOCs. *IEEE Design & Test of Computers*, pages 80–91, Sept. 2001.

[22] A. Chandra and K. Chakrabarty. Low-Power Scan Testing and Test Data Compression for System-on-a-Chip. *IEEE Transactions on Computer-Aided Design*, 21(5):597–604, May 2002.

[23] M. Chatterjee and D. K. Pradhan. A Novel Pattern Generator for Near-perfect Fault Coverage. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 417–425, 1995.

[24] K. T. Cheng and V. D. Agrawal. A Partial Scan Method for Sequential Circuits with Feedback. *IEEE Transactions on Computers*, 39(4):544–548, Apr. 1990.

[25] R. M. Chou, K. K. Saluja, and V. D. Agrawal. Scheduling Tests for VLSI Systems under Power Constraints. *IEEE Transactions on VLSI Systems*, 5(2):175–184, June 1997.

[26] A. L. Crouch. *Design-For-Test For Digital IC's and Embedded Core Systems*. Prentice-Hall, 1999.

[27] W. J. Dally and J. W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[28] S. DasGupta, R. Walther, and T. Williams. An Enhancement to LSSD and Some Aapplications of LSSD in Reliability. In *Proceedings International Fault-Tolerant Computing Symposium*, pages 32–34, June 1981.

[29] B. Dervisoglu and G. Strong. Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement. In *Proceedings IEEE International Test Conference (ITC)*, pages 365–374, Oct. 1991.

[30] R. P. Dick and N. K. Jha. MOCSYN: Multiobjective Core-Based Single-Chip System Synthesis. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 263–270, 1999.

[31] R. Dorsch and H. Wunderlich. Tailoring ATPG for Embedded Testing. In *Proceedings IEEE International Test Conference (ITC)*, pages 530–537, 2001.

[32] S. Dutta, R. Jensen, and A. Rieckmann. Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems. *IEEE Design & Test of Computers*, 18(5):21–31, Sept.-Oct. 2001.

[33] Z. S. Ebadi and A. Ivanov. Design of an Optimal Test Access Architecture Using a Genetic Algorithm. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 205–210, Kyoto, Japan, Nov. 2001.

[34] E. B. Eichelberger, E. Lindbloom, J. A. Waicukauski, and T. W. Williams. Structured logic testing. Prentice-Hall, New Jersey, 1991.

[35] M.-L. Flottes, J. Pouget, and B. Rouzeyre. Sessionless Test Scheme: Power-Constrained Test Scheduling for System-on-a-Chip. In *Proceedings IFIP International Conference on Very Large Scale Integration (VLSI-SOC)*, pages 105–110, Montpellier, France, Dec. 2001.

[36] P. Franco, S. Ma, J. Chang, C. Yi-Chin, , S. Wattal, E. McCluskey, R. Strokes, and W. Farwell. Analysis and Detection of Timing Failures in an Experimental Test Chip. In *Proceedings IEEE International Test Conference (ITC)*, pages 691 –700, 1996.

[37] S. Freeman. Test Generation for Data-Path Logic: The F-path Method. *IEEE Journal of Solid-State Circuits*, 23:421–427, Apr. 1988.

[38] H. Fujiwara. FAN: A Fanout-Oriented Test Pattern Generation Algorithm. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, pages 671–674, July 1985.

[39] H. Fujiwara and T. Shimono. On the Accelaration of Test Generation Algorithms. *IEEE Transactions on Computers*, C-32(12):1137–1144, Dec. 1983.

[40] J. Gatej, S. Lee, C. Pyron, and R. Raina. Evaluating ATE features in terms of Test Escape Rates and Other Cost of Test Culprits. In *Proceedings IEEE International Test Conference (ITC)*, pages 1040 –1049, Oct. 2002.

[41] I. Ghosh, S. Dey, and N. K. Jha. A Fast and Low-Cost Testing Technique for Core-Based System-Chips. *IEEE Transactions on Computer-Aided Design*, 19(8):863, Aug. 2000.

[42] I. Ghosh, N. K. Jha, and S. Dey. Low Overhead Design for Testability and Test Generation Technique for Core-Based Systems-on-a-Chip. *IEEE Transactions on Computer-Aided Design*, 18(11):1661, Nov. 1999.

[43] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, C-30(3):215–222, Mar. 1981.

[44] S. K. Goel. An Improved Wrapper Architecture for Parallel Testing of Hierarchical Cores. In *Proceedings IEEE European Test Symposium (ETS)*, pages 147–152, Corsica, France, May 2004.

[45] S. K. Goel, K. Chiu, E. J. Marinissen, T. Nguyen, and S. Oostdijk. Test Infrastructure Design for the Nexperia^{TM} Home Platform PNX8550 System Chip. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 108–113, Paris, France, Feb. 2004.

[46] S. K. Goel and E. J. Marinissen. A Novel Test Time Reduction Algorithm for Test Architecture Design for Core-Based System Chips. In *Proceedings IEEE European Test Workshop (ETW)*, pages 7–12, Corfu, Greece, May 2002.

[47] S. K. Goel and E. J. Marinissen. Cluster-Based Test Architecture Design for System-on-Chip. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 259–264, Monterey, CA, Apr. 2002.

[48] S. K. Goel and E. J. Marinissen. Effective and Efficient Test Architecture Design for SOCs. In *Proceedings IEEE International Test Conference (ITC)*, pages 529–538, Baltimore, MD, Oct. 2002.

[49] S. K. Goel and E. J. Marinissen. Control-Aware Test Architecture Design for Modular SOC Testing. In *Proceedings IEEE European Test Workshop (ETW)*, pages 57–62, Maastricht, The Netherlands, May 2003.

[50] S. K. Goel and E. J. Marinissen. Layout-Driven SOC Test Architecture Design for Test Time and Wire Length Minimization. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 738–743, Munich, Germany, Mar. 2003.

[51] S. K. Goel and E. J. Marinissen. SOC Test Architecture Design for Efficient Utilization of Test Bandwidth. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):399–429, 2003.

[52] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici. Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 604–611, Mar. 2002.

[53] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici. Useless Memory Allocation in System-on-a-Chip Test: Problems and Solutions. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 423–429, Monterey, CA, Apr. 2002.

[54] R. Gupta and Y. Zorian. Introducing Core-Based System Design. *IEEE Design & Test of Computers*, 14(4):15–25, Dec. 1997.

[55] A. Hales and E. J. Marinissen. IEEE P1500 Web Site. http://grouper.ieee.org/groups/1500/.

[56] I. Hamzaoglu and J. H. Patel. Compact Two-pattern Test Set Generation for Combinational and Full Scan Circuits. In *Proceedings IEEE International Test Conference (ITC)*, pages 944 –953, 1998.

[57] P. Harrod. Testing Reusable IP - A Case Study. In *Proceedings IEEE International Test Conference (ITC)*, pages 493–498, Atlantic City, NJ, Sept. 1999.

[58] D. Heidel, S. Dhong, P. Hofstee, M. Immediato, K. Nowka, J. Silberman, and K. Stawiasz. High Speed Serializing/De-Serializing Design-for-Test Method for Evaluating a 1 GHz Microprocessor. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 234 –238, 1998.

[59] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski. Logic BIST for Large Industrial Designs: Real Issues and Case Studies. In *Proceedings IEEE International Test Conference (ITC)*, pages 358–367, 1999.

[60] M. J. Howes and D. V. Morgan. Reliability and Degradation - Semiconductor Devices and Circuits. Wiley-Interscience, Chichester, UK, 1981.

[61] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, and S. M. Reddy. Static Pin Mapping and SOC Test Scheduling for Cores with Multiple Test Sets. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 99–104, Mar. 2003.

[62] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. M. Reddy. Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 265–270, Kyoto, Japan, Nov. 2001.

[63] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. M. Reddy. On Concurrent Test of Core-Based SOC Design. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):401–414, Aug. 2002.

[64] Y. Huang, N. Mukherjee, C.-C. Tsai, O. Samman, Y. Zaidan, Y. Zhang, W.-T. Cheng, and S. M. Reddy. Constraint-Driven Pin Mapping for Concurrent SOC Testing. In *Proceedings IEEE Asia South Pacific Design Automation Conference (ASP-DAC)*, Bangelore, India, Jan. 2002.

[65] Y. Huang, S. M. Reddy, W.-T. Cheng, P. Reuter, N. Mukherjee, C.-C. Tsai, O. Samman, and Y. Zaidan. Optimal Core Wrapper Width Selection and SOC Test Scheduling Based on 3-D Bin Packing Algorithm. In *Proceedings IEEE International Test Conference (ITC)*, pages 74–82, Baltimore, MD, Oct. 2002.

[66] IEEE Computer Society. *IEEE Standard Test Access Port and Boundary-Scan Architecture - IEEE Std. 1149.1-2001*. IEEE, New York, July 2001.

[67] IEEE Std. 1500. *IEEE Standard for Embedded Core Test - IEEE Std. 1500-2004*. IEEE, New York, 2004.

[68] V. Immaneni and S. Raman. Direct Access Test Scheme - Design of Block and Core Cells for Embedded ASICs. In *Proceedings IEEE International Test Conference (ITC)*, pages 488–492, Sept. 1990.

[69] International SEMATECH. *The International Technology Roadmap for Semiconductors (ITRS): 2001 Edition*. http://public.itrs.net/Files/2001ITRS/Home.htm, 2001.

[70] V. Iyengar and K. Chakrabarty. Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-Chip. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 368–374, Marina del Rey, CA, May 2001.

[71] V. Iyengar and K. Chakrabarty. Test Bus Sizing for System-on-a-Chip. *IEEE Transactions on Computers*, 51:449–459, May 2002.

[72] V. Iyengar, K. Chakrabarty, M. D. Krasniewski, and G. N. Kumar. Design and Optimization of Multi-Level TAM Architectures for Hierarchical SOCs. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 299–304, Napa, CA, Apr. 2003.

[73] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Co-Optimization of Test Wrapper and Test Access Architecture for Embedded Cores. *Journal of Electronic Testing: Theory and Applications*, 18(2):213–230, Apr. 2002.

[74] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Efficient Wrapper/TAM Co-Optimization for Large SOCs. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 491–498, Paris, France, Mar. 2002.

[75] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Integrated Wrapper/TAM Co-Optimization, Constraint-Driven Test Scheduling, and Tester Data Volume Reduction for SOCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 685–690, New Orleans, LO, June 2002.

[76] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 253–258, Monterey, CA, Apr. 2002.

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

[78] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip. *IEEE Transactions on Computers*, 52(12):1619–1632, Dec. 2003.

[79] V. Iyengar, S. K. Goel, K. Chakrabarty, and E. J. Marinissen. Test Resource Optimization for Multi-Site Testing of SOCs under ATE Memory Depth Constraints. In *Proceedings IEEE International Test Conference (ITC)*, pages 1159–1168, Baltimore, MD, Oct. 2002.

[80] V. Jain and J. Waicukauski. Scan Test Data Volume Reduction in Multi-clocked Designs with Safe Capture Technique. In *Proceedings IEEE International Test Conference (ITC)*, pages 148 –153, Oct. 2002.

[81] A. Jas and N. Touba. Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs. In *Proceedings IEEE International Test Conference (ITC)*, pages 458–464, Washington, DC, Oct. 1998.

[82] W. Jiang and B. Vinnakota. Defect-Oriented Test Scheduling. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 433 – 438, 1999.

[83] R. Kapur. *CTL for Test Information of Digital ICs*. Springer-Verlag, 2002.

[84] R. Kapur, M. Lousberg, T. Taylor, B. Keller, P. Reuter, and D. Kay. CTL – The Language for Describing Core-Based Test. In *Proceedings IEEE International Test Conference (ITC)*, pages 131–139, Baltimore, MD, Oct. 2001.

[85] R. Kapur and T. W. Williams. Manufacturing Test of SoCs. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 317–319, Tamuning, Guam, USA, Nov. 2002.

[86] M. Keating and P. Bricaud. *Reuse Methodology Manual: For System-on-a-Chip Designs*. Kluwer Academic Publishers, 1998.

[87] A. Khoche. Test Resource Partitioning for Scan Architectures Using Bandwidth Matching. In *Digest of Int. Workshop on Test Resource Partitioning*, pages 1.4.1–1.4.8, 2002.

[88] G. Kiefer, H. Vranken, E. Marinissen, and H. Wunderlich. Application of Deterministic Logic BIST on Industrial Circuits. In *Proceedings IEEE International Test Conference (ITC)*, pages 105–114, 2000.

[89] M. J. Knieser, F. G. Wolff, C. A. Papachristou, D. J. Weyer, and D. R. McIntyre. A technique for high ratio lzw compression. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 116–121, 2003.

[90] B. Koenemann. LFSR-Coded Test Patterns for Scan Designs. In *Proceedings IEEE European Test Conference (ETC)*, pages 237–242, 1991.

[91] B. Koenemann, C. Barnhart, B. Keller, O. Farnsworth, and D. Wheater. A Smart-BIST Variant with Guaranteed Encoding. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 325–330, Nov. 2001.

[92] S. Koranne. A Novel Reconfigurable Wrapper for Testing of Embedded Core-Based SOCs and its Associated Scheduling Algorithm. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):415–434, Aug. 2002.

[93] S. Koranne. Formulating SoC Test Scheduling as a Network Transportation Problem. *IEEE Transactions on Computer-Aided Design*, 21(12):1517–1525, Dec. 2002.

[94] S. Koranne. On Test Scheduling for Core-Based SOCs. In *Proceedings IEEE International Conference on VLSI Design (ICVD)*, pages 505–510, Bangelore, India, Jan. 2002.

[95] S. Koranne. Design of Reconfigurable Access Wrappers for Embedded Core Based SoC Test. *IEEE Transactions on VLSI Systems*, 11(5):955–960, Aug. 2003.

[96] S. Koranne. Solving the SoC Test Scheduling Problem Using Network Flow and Reconfigurable Wrappers. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 93–98, San Jose, CA, Mar. 2003.

[97] S. Koranne and V. Iyengar. On the Use of k-tuples for SoC Test Schedule Representation. In *Proceedings IEEE International Test Conference (ITC)*, pages 539–548, Baltimore, MD, Oct. 2002.

[98] A. Krstic and K. T. Cheng. *Delay Fault Testing for VLSI Circuits*. Kluwer Academic Publishers, 1998.

[99] L. Krundel, S. K. Goel, E. J. Marinissen, M.-L. Flottes, and B. Rouzeyre. User-Constrained Test Architecture Design for Modular SOC Testing. In *Proceedings IEEE European Test Symposium (ETS)*, pages 80–85, May 2004.

[100] E. Larsson and H. Fujiwara. Power Constrained Preemptive TAM Scheduling. In *Proceedings IEEE European Test Workshop (ETW)*, pages 119–126, Corfu, Greece, May 2002.

[101] E. Larsson and H. Fujiwara. Optimal System-on-Chip Test Scheduling. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 306–311, Nov. 2003.

[102] E. Larsson and H. Fujiwara. Test Resource Partitioning and Optimization for SOC Designs. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 319–324, Napa, CA, Apr. 2003.

[103] E. Larsson and Z. Peng. An Integrated System-on-Chip Test Framework. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 138–144, Munich, Germany, Mar. 2001.

[104] E. Larsson and Z. Peng. Test Scheduling and Scan-Chain Division under Power Constraint. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 259–264, Kyoto, Japan, Nov. 2001.

[105] E. Larsson and Z. Peng. A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling. In *Proceedings IEEE International Test Conference (ITC)*, pages 1135–1144, Charlotte, NC, Sept. 2003.

[106] E. Larsson, Z. Peng, and G. Carlsson. The Design and Optimization of SOC Test Solutions. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 523–530, San Jose, CA, Nov. 2001.

[107] J.-F. Li, H.-J. Huang, J.-B. Chen, C.-P. Su, C.-W. Wu, C. Cheng, S.-I. Chen, C.-Y. Hwang, and H.-P. Lin. A Hierarchical Test Scheme for System-on-Chip Designs. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 486–490, Munich, Germany, Feb. 2002.

[108] L. Li and K. Chakrabarty. Test Data Compression Using Dictionaries with Fixed-Length Indices. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 219–224, 2003.

[109] X. Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli. High-Frequency, At-Speed Scan Testing. *IEEE Design & Test of Computers*, 20(5):17–25, Oct. 2003.

[110] X. Lin and R. Thompson. Test Generation for Designs with Multiple Clocks. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 662 –667, June 2003.

[111] R. Madge, B. R. Benware, and W. R. Daasch. Obtaining High Defect Coverage for Frequency-Dependent Defects in Complex ASICs. *IEEE Design & Test of Computers*, 20(5):46–53, Oct. 2003.

[112] E. J. Marinissen. The Role of Test Protocols in Automated Test Generation for Embedded-Core-Based System ICs. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):435–454, Aug. 2002.

[113] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters. A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 284–293, Washington, DC, Oct. 1998.

[114] E. J. Marinissen, S. K. Goel, and M. Lousberg. Wrapper Design for Embedded Core Test. In *Proceedings IEEE International Test Conference (ITC)*, pages 911–920, Atlantic City, NJ, Oct. 2000.

[115] E. J. Marinissen, V. Iyengar, and K. Chakrabarty. ITC'02 SOC Test Benchmarks Web Site. http://www.extra.research.philips.com/itc02socbenchm/.

[116] E. J. Marinissen, V. Iyengar, and K. Chakrabarty. A Set of Benchmarks for Modular Testing of SOCs. In *Proceedings IEEE International Test Conference (ITC)*, pages 519–528, Baltimore, MD, Oct. 2002.

[117] E. J. Marinissen, R. Kapur, M. Lousberg, T. Mclaurin, M. Ricchetti, and Y. Zorian. On IEEE P1500's Standard for Embedded Core Test. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):365–383, Aug. 2002.

[118] E. J. Marinissen, R. Kapur, and Y. Zorian. On Using IEEE P1500 SECT for Test Plug-n-Play. In *Proceedings IEEE International Test Conference (ITC)*, pages 770–777, Atlantic City, NJ, Oct. 2000.

[119] E. J. Marinissen and M. Lousberg. Macro Test: A Liberal Test Approach for Embedded Reusable Cores. In *Digest of Papers of IEEE International Workshop on*

Testing Embedded Core-Based Systems (TECS), pages 1.2–1–9, Washington, DC, Nov. 1997.

[120] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel. Towards a Standard for Embedded Core Test: An Example. In *Proceedings IEEE International Test Conference (ITC)*, pages 616–627, Atlantic City, NJ, Sept. 1999.

[121] P. Maxwell, R. Aitken, K. Kollitz, and A. Brown. IDDQ and AC Scan: The War against Unmodelled Defects. In *Proceedings IEEE International Test Conference (ITC)*, pages 250 –258, 1996.

[122] S. Mitra and K. S. Kim. X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction. In *Proceedings IEEE International Test Conference (ITC)*, pages 311 – 320, 2002.

[123] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI Module Placement Based on Rectangle-Packing by The Sequence-Pair. *IEEE Transactions on Computer-Aided Design*, 15(12):1518–1524, Dec. 1996.

[124] V. Muresan, X. Wang, V. Muresan, and M. Vladutiu. A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling. In *Proceedings IEEE International Test Conference (ITC)*, pages 882–891, 2000.

[125] B. Nadeau-Dostie, D. M. Burek, and A. S. M. Hassan. ScanBist: A Multifrequency Scan-Based BIST Method. *IEEE Design & Test of Computers*, pages 7–17, Spring 1994.

[126] B. Nadeau-Dostie, A. S. Hassan, D. M. Burek, and S. K. Sunter. Mulitple Clock Rate Test Apparatus for Testing Digital Systems. U.S. Patent 5349587, Northern Telecom Limited, Montreal, Canada, Sep. 20, 1994.

[127] P. Nag, A. Gattiker, S.Wei, R. Blanton, and W. Maly. Modeling the Economics of Testing: A DFT Perspective. *IEEE Design & Test of Computers*, 19:29–41, Jan. 2002.

[128] N. Nicolici and B. M. Al-Hashimi. Multiple Scan Chains for Power Minimization During Test Application in Sequential Circuits. *IEEE Transactions on Computers*, 51(6):721–734, June 2002.

[129] N. Nicolici and B. M. Al-Hashimi. *Power-Constrained Testing of VLSI Circuits*. Kluwer Academic Publishers, 2003.

[130] A. P. Niranjan and P. Wiscombe. Islands of Synchronicity, a Design Methodology for SoC Design. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 64–69, 2004.

[131] M. Nourani and C. Papachristou. Structural Fault Testing of Embedded Cores Using Pipelining. *Journal of Electronic Testing: Theory and Applications*, 15(1):129, 1999.

[132] K. P. Parker. *The Boundary-Scan Handbook*. Springer-Verlag, 3rd edition, 2003.

[133] J. H. Patel, S. S. Lumetta, and S. M. Reddy. Application of Saluja-Karpovsky Compactors to Test Responses with Many Unknowns. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 107–112, 2003.

[134] S. Pateras. Achieving At-Speed Stuctural Test. *IEEE Design & Test of Computers*, 20(5):26–33, Oct. 2003.

[135] B. Pouya and N. Touba. Modifying User-Defined Logic for Test Access to Embedded Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 60–68, Washington, DC, Nov. 1997.

[136] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, 2nd edition, 2003.

[137] Rafey Mahmud. Techniques to Make Clock Switching Glitch Free. http://www.eetimes.com/story/OEG20030626S0035.

[138] J. Rajski, M. Kassab, N. Mukherjee, N. Tamarapalli, J. Tyszer, and J. Qian. Embedded Deterministic Test for Low-cost Manufacturing. In *IEEE Design & Test of Computers*, pages 58–66, Sept.-Oct. 2003.

[139] J. Rajski, J. Tyszer, C. Wang, and S. M. Reddy. Convolutional Compaction of Test Responses. In *Proceedings IEEE International Test Conference (ITC)*, pages 745–754, Oct. 2003.

[140] R. Rajsuman. Testing a System-on-a-Chip with Embedded Microprocessor. In *Proceedings IEEE International Test Conference (ITC)*, pages 499–508, Atlantic City, NJ, Sept. 1999.

[141] S. Ravi and N. K. Jha. Test Synthesis of System-on-a-Chip. *IEEE Transactions on Computer-Aided Design*, 21(10):1211–1217, 2002.

[142] S. Ravi, G. Lakshminarayana, and N. K. Jha. Testing of Core-Based Systems-on-a-Chip. *IEEE Transactions on Computer-Aided Design*, 20(3):426–439, Mar. 2001.

[143] C. P. Ravikumar, G. Chandra, and A. Verma. Simultaneous Module Selection and Scheduling for Power-constrained Testing of Core Based Systems. In *Proceedings IEEE International Conference on VLSI Design (ICVD)*, pages 462–467, 2000.

[144] C. P. Ravikumar, A. Verma, and G. Chandra. A Polynomial-Time Algorithm for Power Constrained Testing of Core Based Systems. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 107–112, 1999.

[145] J. Rearick. Too Much Delay Fault Coverage Is A Bad Thing. In *Proceedings IEEE International Test Conference (ITC)*, pages 624–633, Nov. 2001.

[146] R.L.Wadsack. Fault modeling and logic simulation of cmos and mos integrated circuits. *Bell Systems Technical Journal*, pages 1449–1474, May-June 1978.

[147] J. Roth. Diagnosis of Automata Failures: A calculus and a Method. *IBM Journal of Research and Development*, 10(4):278–291, July 1967.

[148] H. L. S. Hellebrand and H. Wunderlich. A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters. In *Proc. IEEE International Test Conference*, page 778C784, Oct. 2000.

[149] D. Salomon. *Data Compression: The Complete Reference*. Springer-Verlag, New York, 2000.

[150] J. Schmid and J. Knablein. Advanced Synchronous Scan Test Methodology for Multi Clock Domain ASICs. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 106 – 113, 1999.

[151] H. Schwab. Lp solve. In *http://elib.zib.de/pub/Packages/mathprog/linprog/lp-solve*, 1997.

[152] A. Sehgal and K. Chakrabarty. Efficient Modular Testing of SOCs Using Dual-Speed TAM Architectures. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 422–427, Paris, France, Feb. 2004.

[153] A. Sehgal, S. K. Goel, E. J. Marinissen, and K. Chakrabarty. IEEE P1500-Compliant Test Wrapper Design for Hierarchical Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 1203–1212, Charlotte, NC, Oct. 2004.

[154] A. Sehgal, V. Iyengar, M. D. Krasniewski, and K. Chakrabarty. Test Cost Reduction for SOCs Using Virtual TAMs and Lagrange Multipliers. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 738–743, Anaheim, CA, June 2003.

[155] C.-P. Su and C.-W. Wu. A Graph-Based Approach to Power-Constrained SOC Test Scheduling. *Journal of Electronic Testing: Theory and Applications*, 20(1):45–60, Feb. 2004.

[156] M. Sugihara, H. Date, and H. Yasuura. A Novel Test Methodology for Core-Based System LSIs and a Testing Time Minimization Problem. In *Proceedings IEEE International Test Conference (ITC)*, pages 465–472, Washington, DC, Oct. 1998.

[157] Taiwan Semiconductor Manufacturing Corporation. TSMC 0.18μ CMOS technology. http://www.tsmc.com.

[158] Technical White Paper. Designs with Multiple Clock Domains: Avoiding Clock Skew and Reducing Pattern Count Using DFT Advisor and Fast Scan. http://www.mentor.com/dft.

[159] M. Tehranipour, M. Nourani, and K. Chakrabarty. Nine-Coded Compression Technique with Application to Reduced Pin-Count Testing and Flexible on-chip Decompression. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1284–1289, Feb. 2004.

[160] N. Touba and E. McCluskey. Altering a Pseudorandom Bit Sequence for Scan-Based BIST. In *Proceedings IEEE International Test Conference (ITC)*, page 167C175, Oct. 1996.

[161] N. Touba and B. Pouya. Using Partial Isolation Rings to Test Core-Based Designs. *IEEE Design & Test of Computers*, 14(4):52–59, Dec. 1997.

[162] P. Varma and S. Bhatia. A Structured Test Re-Use Methodology for Core-Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 294–302, Washington, DC, Oct. 1998.

[163] H. Vermaak and H. Kerkhoff. Enhanced P1500 Compliant Wrapper suitable for Delay Fault Testing of Embedded Cores. In *Digest of Papers of IEEE European Test Workshop (ETW)*, pages 257–262, Maastricht, The Netherlands, May 2003.

[164] B. Vermeulen, S. Oostdijk, and F. Bouwman. Test and Debug Strategy of the PNX8525 Nexperia^{TM} Digital Video Platform System Chip. In *Proceedings IEEE International Test Conference (ITC)*, pages 121–130, Baltimore, MD, Oct. 2001.

[165] L. Whetsel. An IEEE 1149.1 Based Test Access Architecture for ICs with Embedded Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 69–78, Washington, DC, Nov. 1997.

[166] F. G. Wolff and C. Papachristou. Multiscan-Based Test Compression and Hardware Decompression Using LZ77. In *Proceedings IEEE International Test Conference (ITC)*, pages 331–339, Oct. 2002.

[167] C.-W. Wu. VLSI Testing and Design for Testability Course - Lecture on Core-Based SOC Testing. http://larc.ee.nthu.edu.tw/˜ cww/n/625/6251/13SOC0211.pdf.

[168] D. M. Wu, M. Lin, S. Mitra, K. S. Kim, A. Sabbavarapu, T. Jaber, P. Johnson, D. March, and G. Parrish. H-DFT: A Hybrid DFT Architecture for Low-Cost High Quality Structural Testing. In *Proceedings IEEE International Test Conference (ITC)*, pages 1229–1238, Oct. 2003.

[169] H. Wunderlich and G. Kiefer. Bit-Flipping BIST. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 337 –343, Nov. 1996.

[170] Q. Xu. Updated ITC'02 Benchmark SOCs with Random Functional Interconnect Information. http://www.ece.mcmaster.ca/˜ nicola/cadt.html.

[171] Q. Xu and N. Nicolici. Resource-Constrained System-on-a-Chip Test: A Survey. *IEE Proceedings, Computers and Digital Techniques*, 152(1):67–81, Jan. 2005.

[172] T. Yoneda and H. Fujiwara. Design for Consecutive Testability of System-on-a-Chip with Built-In Self Testable Cores. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):487–501, Aug. 2002.

[173] T. Yoneda and H. Fujiwara. Design for Consecutive Transparency of Cores in System-on-a-Chip. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 287–292, Napa, CA, Apr. 2003.

[174] T. Yoneda, T. Uchiyama, and H. Fujiwara. Area and Time Co-Optimization for System-on-a-Chip based on Consecutive Testability. In *Proceedings IEEE International Test Conference (ITC)*, pages 415–422, Charlotte, NC, Sept. 2003.

[175] D. Zhao and S. Upadhyaya. Power Constrained Test Scheduling with Dynamically Varied TAM. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 273–278, 2003.

[176] D. Zhao and S. Upadhyaya. A Generic Resource Distribution and Test Scheduling Scheme for Embedded Core-Based SoCs. *IEEE Transactions on Instrumentation and Measurment*, 53(2):318–329, Apr. 2004.

[177] Y. Zorian. A Distributed BIST Control Scheme for Complex VLSI Devices. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 6–11, Princeton, NJ, Apr. 1993.

[178] Y. Zorian. Test Requirements for Embedded Core-Based Systems and IEEE P1500. In *Proceedings IEEE International Test Conference (ITC)*, pages 191–199, Washington, DC, Nov. 1997.

[179] Y. Zorian, E. J. Marinissen, and S. Dey. Testing Embedded-Core Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 130–143, Washington, DC, Oct. 1998.

[180] Y. Zorian, E. J. Marinissen, and S. Dey. Testing Embedded-Core-Based System Chips. *IEEE Computer*, 32(6):52–60, June 1999.

[181] W. Zou, S. M. Reddy, I. Pomeranz, and Y. Huang. SOC Test Scheduling Using Simulated Annealing. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 325–330, Napa, CA, Apr. 2003.

Index