An Integrated Framework for Concurrent Test and Wireless Control in Complex SoCs

by

Dan Zhao

December 2003

A dissertation

submitted to the Faculty of the Graduate School

of State University of New York at Buffalo

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Department of Computer Science and Engineering

To my husband, parents and sister

to the earlier generations who gave me potential,

to my current family who supports, encourages and inspires me to pursue my passions,

and most of all, to the future generations who give me a reason to dream.

ACKNOWLEDGMENTS

Doctoral research is an intensely individual journey supported by a tremendous team effort. Without the assistance of others, a dissertation cannot flourish. I have been immensely fortunate to have a phenomenal group of people nurture me and my research over the past four years, and this dissertation would not be complete without recognizing their efforts.

I would like to express my gratitude to my advisor, Dr. Shambhu Upadhyaya, for his exceptional guidance and continuous support, for his insights and outlook on computer engineering in general. I have benefited greatly during my study at UB from the effort of Dr. Upadhyaya. He knows the importance of allowing students the freedom to find their own way, yet at the same time he is always willing and able to give advice. He is interested not only in the research I perform while in the Security, PrIvacy, DEpendability Research (SPIDER) Laboratory, but also in cultivating abilities that will inspire me in my entire career.

I also want to sincerely and gratefully thank my dissertation committee members, Dr. Ramalingam Sridhar and Dr. Nihar Mahapatra. Dr. Sridhar has shaped my research experience in many ways, most of all, the inspiration of my interest in VLSI world. The most significant contributions of Dr. Mahapatra relate to teaching. I am always grateful for his role in helping me find a passion in teaching.

I would like to acknowledge Dr. Martin Margala in University of Rochester for the thoughtful discussions we had in the last a couple of years. His insights and collaborations have dedicated to the completion of this work. My thanks also go to my dissertation outside reader, Dr. Krishnendu Chakrabarty, for his cogent comments. His never-ending enthusiasm for new and interesting ideas have impacted my doctoral experience.

I wish to thank the folks in SPIDER lab, especially, Dr. Jae Min Lee, Ramkumar, Suranjan, who

have contributed to the invaluable discussion and interactions of the SPIDER family. Over the years, I have shared great times with my friends in Buffalo. Their friendly attitude and strong dedication have made life so much more pleasant for graduate studies in CSE. I truly appreciate their efforts and more importantly, their warm friendship.

Foremost, of course, major gratitude must be extended to my husband, my parents and my sister for their constant encouragement, love and support. The greatest influence on my doctoral research and on my life is Hongyi, my husband. He is a remarkable man who, just by being himself, inspires me. Finally, thanks to those who wrote “A Fool on the Hill”, a tune which made our pedestrian tasks somewhat easier.

Contents

List of Figures

List of Tables

ABSTRACT

System-on-chip (SoC) is evolving as a new design style, where an entire system is built by reusing pre-designed, pre-verified IP (intellectual property) cores. Embedded with numerous heterogeneous and complex IP cores, an SoC can be viewed as an interconnected network of various functional modules. This new design style shortens time-to-market while meeting various design requirements, such as high performance, low power, and low cost, compared to the traditional system-on-board (SoB) design. In the meantime, however, embedded core-based SoC test becomes a challenging task due to IP protection. In particular, there are three major issues to be addressed in SoC test: (1) a test access path needs to be constructed for each core to propagate test stimulus and collect test responses, (2) one needs to partition test resources and schedule IP cores to achieve maximum parallelism, and (3) a test control network is needed to initialize different test resources used in the test application and observe the corresponding test results at appropriate instants.

In this dissertation, we develop cost-effective SoC test and diagnosis solutions from various crucial aspects, such as test time, test access architecture, and memory depth on automatic test equipment (ATE). It is the very first work that introduces radio frequency (RF) technology into SoC test control for the forthcoming billion-transistor era. We mainly address two research issues: *integrated testability design and optimization of SoC test solutions*, and *on-chip wireless test control network design*. We first develop a general test model for SoC testability analysis, test scheduling, and test diagnosis. We then propose several test scheduling algorithms with the consideration of various test constraints such as resource sharing, power dissipation, and fault coverage, and develop an integrated framework that combines wrapper design, test access mechanism (TAM) configuration, and test scheduling. More specifically, we propose a fault model oriented test set selection scheme and formulate the test scheduling as a shortest path problem with the feature of evenly balanced resource usage. We also propose a dynamic test partitioning technique based on the test compatibility graph to address the power-constrained test scheduling problem. Furthermore, we develop an integrated framework to handle constrained scheduling in a way that constructs core access paths and distributes TAM bandwidth among cores, and consequently configures the wrapper scan chains for TAM width adaptation. Using the "Radio-on-Chip" technology, we introduce a novel test control

network to transmit control signals chip-wide by RF links. We propose three types of wireless test control architectures, i.e., a miniature wireless local area network, a multihop wireless test control network, and a distributed multihop wireless test control network. The proposed architectures consist of three basic components, namely the test scheduler, the resource configurators, and the RF nodes supporting the communication between the scheduler and the IP cores. Under the multilevel tree structure, the system optimization is performed on control constrained resource partitioning and distribution. Several challenging system design issues such as RF nodes placement, clustering, and routing, are studied along with integrated resource distribution (including not only the circuit blocks to perform testing, but also the on-chip RF nodes for intra-chip communication) and test scheduling (concurrent core testing as well as interconnect testing). Cost oriented optimization technique is developed which addresses several highly interdependent design issues to achieve the minimum overall testing cost.

Chapter 1

Introduction

The evolution of nanometer technology and the increasing system complexity have given rise to the popularity of System-on-Chip (SoC) technology, where an entire system is built on a single chip using pre-designed, pre-verified complex logic blocks called embedded cores, which leverage the system by the intellectual property (IP) advantage. The system designers or integrators may use the cores which cover a wide range of functions from CPU to SRAM to DSP to analog, and integrate them into a single silicon with their own user-defined-logics (UDLs). According to ITRS'01 [5], at $65nm$ and below, design of very complex SoCs consisting of billions of transistors, operating below one volt and running at 10*GHz* will become a reality by the end of the decade. SoC design in the forthcoming billion-transistor era will involve the integration of numerous heterogeneous IP cores. The SoC technology has shown great advantage in shortening the time-to-market of a new system and meeting various design requirements such as high performance, low power, and low cost, compared to the traditional system-on-board (SoB) design.

The embedded cores are delivered at the hardware description level, i.e., soft (register-transfer level), firm (netlist), or hard (technology-dependent layout). Therefore, they are not manufactured and tested before integration. The system integrators need to test the whole system chip, i.e., not only the interconnects between the cores, but also the cores themselves. The core-level tests need to be selected from the pre-designed tests for various cores and additional tests need to be developed for user-defined-logics (UDLs) around the cores and interconnects. Such a test strategy is referred

to as the core-based SoC testing, which brings forth several new challenges.

As the system integrators have limited knowledge of core internals, the core tests, including design-for-testability (DfT) techniques, test pattern generation, core internal test requirements are often provided by core-vendors. The system integrators should consider the trade-offs between test quality and test cost, i.e., total test time, area overhead, performance overhead and power dissipation. On the other hand, various testing methods such as BIST, scan, functional and IDDQ for many kinds of design environments are provided by different core vendors. With continued scaling of microelectronics, a future SoC will see several hundreds of embedded components in a single package [6] and today's SoC will become tomorrow's IP core [7]. Embedded with numerous heterogeneous and complex IP cores, an SoC can be viewed as an interconnected network of various functional modules. Testing such high-density high-volume core-based SoCs faces three major issues: *accessing deeply embedded cores with high-speed high-efficiency low-cost interconnect structure; partitioning test resources and scheduling IP cores to achieve maximum parallelism; and developing a high-efficiency low-cost control network to execute the test application based on a predetermined schedule*.

Reuse methodologies have forced partitioning of the test data over IP blocks, which directly affects the cost of test in terms of both test application time and test data volume [8]. System level scheduling is pursued to reduce the test cost, specifically, the test application time by a certain level of parallelism while meeting the test quality. There are several constraints that must be considered in scheduling of tests. First, in a core-based SoC, not all tests can be applied at the same time due to resource conflicts. For example, several cores may share the same test generator or response evaluator, and thus cannot be tested in parallel. In addition, the power consumption must be taken into account in order to guarantee proper operating conditions. For instance, in a self-tested system, testing the cores in parallel may cause high power consumption exceeding the maximum power limit, which will result in system damage due to overheating, while the cores may not activate simultaneously in normal functional mode. Finally, certain fault coverage should be achieved when testing an SoC. There are usually a number of core-testing methods available and each of them detects different faults (e.g., BIST for detecting performance-related defects and non-modelled faults, while

external test for detecting modelled faults). More than one method may be needed to test a core in order to achieve the required fault coverage. In general, the basic idea of scheduling is to arrange the tests in parallel so that no resource conflict occurs with respect to the test access architecture, and the total power dissipation of the system does not exceed the maximum power limit at any time while minimizing the overall test application time.

Cores are deeply embedded in the SoC, and direct access to the cores is usually impossible. Thus, an efficient test access architecture is needed to access the cores, which includes three major components, test source and sink, test access mechanisms (TAMs) and test wrappers. TAMs transport the test stimuli from the source to the core-under-test (CUT) or the test responses from CUT to the sink. The TAM design involves the trade-offs between the transport capacity (bandwidth) of the mechanism and the test application cost it induces, such as test time and area overhead [1]. Several types of TAM structures such as Macro test [9], core transparency [10], multiplexed direct parallel access [11], Boundary Scan based test [12, 13], dedicated test bus [14] and TestRail [15] have been proposed for testing core-based SoCs. At the same time, IEEE P1500 [16] provides standard, but scalable and configurable test wrappers to achieve efficient test isolation and to ease test access. The wrappers may provide width adaptation by serial-parallel or parallel-serial conversion, in case of a mismatch between the width of available TAMs and the core input/output terminals.

When moving into the billion-transistor era, the core accessibility becomes essential as direct physical access is not available, and the accessibility is severely restricted in not only testing time but also test coverage, and consequently test cost and reliability. Although copper/low κ materials have been introduced for deep sub-micron interconnects, they may become insufficient as the technology goes below $100nm$. Recent studies have shown that the traditional hard-wired metal interconnect system will eventually encounter fundamental limits and may impede the advances of future ultralarge-scale integrated systems (ULSIs) [18]. In the meantime, recent advances in silicon integrated circuit technology are making possible tiny low-cost antennae, receivers and transmitters to be integrated on chip. As a result, a new radio frequency (RF)/Microwave interconnect technology has been introduced for future intra-chip communication [18, 19]. In [19], the feasibility of employing on-chip wireless interconnects for clock distribution has been investigated. By introduc-

ing a novel test data and control architecture with wireless connectivity and communication, test accessibility of deeply embedded cores from chip level pins could be significantly improved. Accordingly, a new SoC test strategy needs to be developed by using very short-range, low-power and low-cost wireless network integrated with core-level and chip-level tests. As first step, we investigate the applicability of the recently developed "Radio-on-Chip" technology on test control, which requires transmission of only single tone wireless signals chipwide.

One of the major issues in SoC test is the development of a low-cost, efficient control network that initializes different test resources used in test application and observes the corresponding test results at appropriate instants. In the current technology, the control network connects the central controller (system level controller) with local control mechanisms by wires in one of the three structures: star, bus, and multiple bus [17]. A system level controller is used to execute the test application based on a predetermined schedule. With the integration of tiny antennae and transceivers onto a single chip, the chip-based wireless radios can replace wires used in conventional control network to increase accessibility, to improve bandwidth utilization, and to eliminate delay and cross-talk noise in conventional wired interconnects.

This dissertation investigates cost-effective SoC test and diagnosis solutions from various crucial aspects such as test time, test access architecture, and memory depth on automatic test equipment (ATE). It is the very first work that introduces radio frequency (RF) technology into SoC test control for the forthcoming billion-transistor era. Specifically, we develop effective algorithmic models for optimal test scheduling and efficient test access architecture design, and establish a novel distributed multi-hop wireless test control network based on the recent development of "Radio-on-Chip" technology. In this dissertation, we mainly address two research issues:

- **Integrated Testability Design and Optimization of SoC Test Solutions**

  With numerous heterogeneous and complex intellectual property (IP) cores that perform different functions and operate at different clock frequencies integrated in a single package, well-designed test access architecture and test scheduling algorithms are important to run intra-core and inter-core tests efficiently, reducing overall test cost while meeting the test quality requirements. With this motivation, we develop a general test model for SoC testabil-

ity analysis, test scheduling, and test diagnosis. We propose several test scheduling algorithms with the consideration of various test constraints such as resource sharing, power dissipation, and fault coverage, and develop an integrated framework that combines wrapper design, test access mechanism configuration, and test scheduling. More specifically, we propose a fault model oriented test set selection scheme and formulate the test scheduling as a shortest path problem with the feature of evenly balanced resource usage. We also propose a dynamic test partitioning technique based on a test compatibility graph to address the power-constrained test scheduling problem. Furthermore, we develop an integrated framework to handle constrained scheduling in a way that constructs core access paths and distributes TAM bandwidth among cores, and consequently configures the wrapper scan chains for TAM width adaptation.

- **On-chip Wireless Test Control Network Design**

  When moving into the billion-transistor era, the wired interconnects used in conventional SoC test control models are rather restricted in not only system performance, but also signal integrity and transmission with continued scaling of the feature size. Recent advances in silicon integrated circuit technology are making possible tiny low-cost transceivers to be integrated on chip. Using the "Radio-on-Chip" technology, we introduce a novel test control network to transmit control signals chip-wide by RF links. We propose three types of wireless test control architectures, i.e., a miniature wireless local area network, a multihop wireless test control network, and a distributed multihop wireless test control network. The proposed architectures consist of three basic components, namely the test scheduler, the resource configurators, and the RF nodes supporting the communication between the scheduler and the IP cores. Several challenging system design issues, such as RF nodes placement, clustering, and routing are studied for control constrained resource partitioning and distribution. We further present the optimization technique for the integration of system resource distribution (including not only the circuit blocks to perform testing, but also the on-chip RF nodes for intra-chip communication), TAM design and test scheduling (concurrent core testing as well as interconnect testing) under power and cost constraints.

The rest of this dissertation is organized as follows. Chapter 2 introduces the background of SoC test including test scheduling, TAM design, wrapper configuration and test control. In Chapter 3, we formulate the test scheduling problem as the shortest path problem and propose a novel scheduling scheme based on effective balancing of resource usage. Furthermore, we propose a grouping scheme and all-permutation scheduling to further reduce the overall test time. Chapter 4 presents a novel adaptive scheduling algorithm in a way that dynamically partitions and allocates the tests, consequently constructs and updates a set of dynamically partitioned power constrained concurrent test sets, and ultimately reduces the test application time. Chapter 5 addresses the power constrained test scheduling with dynamically varied TAM which efficiently optimizes the core assignment on TAMs and distributes varied TAM widths to the cores according to their data bandwidth needs. Chapter 6 introduces three types of test control architectures: miniature WLAN, multihop scheme and distributed multihop scheme, and formulates the problem of RF nodes placement. In Chapter 7, we first propose an integrated framework for core test and interconnect test under wireless control. Further, we analyze and formulate SoC testing cost and present a cost oriented system optimization algorithm which addresses several highly interdependent design issues so as to achieve the minimum overall testing cost. Finally, Chapter 8 concludes this dissertation and presents the future work.

Chapter 2

Background and Related Work

2.1 Background

In this section, we provide an overview of current industrial practices as well as academic research on SoC test. We also discuss industry-wide efforts by VSIA and IEEE P1500 Standard Working Group and describe the challenges on SoC testing research.

2.1.1 Overview of Embedded Core Based SoC Test

Core-based design and reuse is emerging as a new paradigm for modern systems, where the system integrators (or designers) reuse embedded modules in building on-chip systems similar to using integrated circuits in a printed circuit board (PCB). These large system ICs are often referred to as *system on chips* (SoCs). SoC designers formed a rich library of pre-designed, pre-verified building blocks, the so-called *embedded cores* to import technology to a new system and differentiate the corresponding product by leveraging IP advantages [1]. The SoC design often contains a very wide range of functional modules from programmable CPUs to DSPs, as well as application-specific hardware, embedded memories of different types, and some analog modules. Cores sometime come in hierarchical compositions, i.e., a complex core embeds one or several simple cores. Cores are delivered at a wide range of hardware description levels, i.e., soft (register-transfer level), firm (netlist), or hard (technology-dependent layout). Each type of cores has different modelling and test requirements [20]. The SoC design process shortens time-to-market while meeting various design

requirements such as high performance, low power consumption, and low cost.

Although the design process in core-based SoCs is conceptually analogous to the traditional system on board (SoB) design, the manufacturing test processes in both cases are quite different [1]. In the SoB approach, as described in Figure 2.1(a), IC design, manufacturing, and testing are performed by the IC provider, prior to PCB assembly and test done by the system integrator. Whereas in SoC trajectory, as shown in Figure 2.1(b), the core provider only delivers a description of the core design to the system integrator, and the cores are not manufactured and tested before integration. The system integrator is responsible for not only the design and test of UDLs, interconnect logic and wiring between the cores, but also the test of cores themselves. SoC test is a single composite test, i.e., the manufacturing and test are performed for the whole system chip.

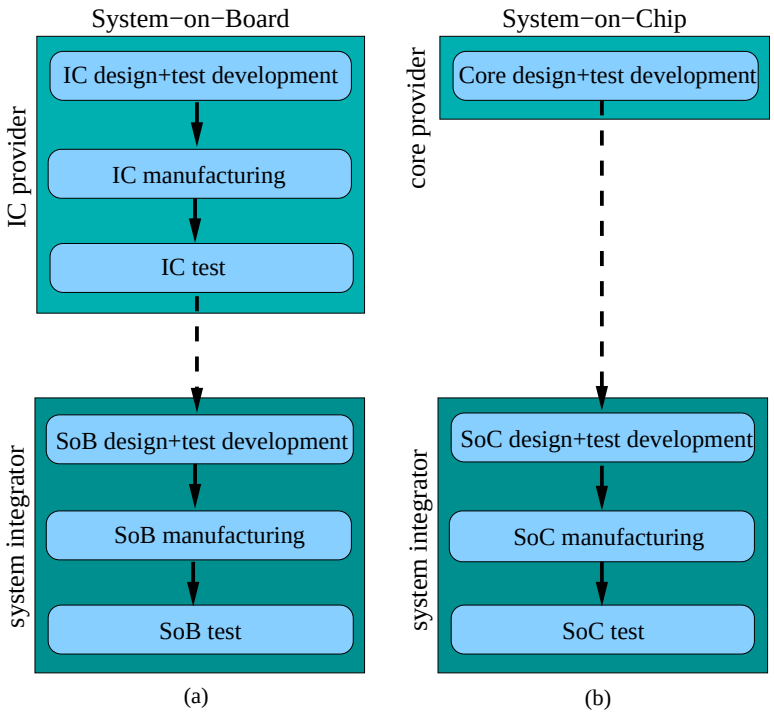


Figure 2.1: System-on-Board (a) vs. System-on-Chip (b) trajectory [1].

2.1.2 SoC Test Challenges

The core-based SoC design brings forth several new challenges, especially in the domains of manufacturing test and design validation and debug.

Core-Level Testing

Increased usage of embedded pre-designed reusable cores necessitates a core-based test strategy, in which cores are tested as separate entities. As the system integrators in most cases, have limited knowledge of the core internals (except for soft cores) and the cores usually appear as black boxes with known functionality and I/Os, the core tests including DfT technique, test pattern generation, core internal test requirements, etc., are often provided by core-vendors [1]. The system integrators need to assemble a high-quality, but inexpensive test for each core. In other words, they should consider the trade-offs between the test quality and the testing costs, i.e., total test time, area overhead, performance overhead and power dissipation. Specifically, an efficient test scheduling scheme is required to optimize above test issues.

Test Access to Embedded Cores

As cores are deeply embedded in the SoC, direct access to embedded cores is usually impossible. An efficient test access architecture (as shown in Figure 2.2) is needed to access the cores, which includes three basic components [21].

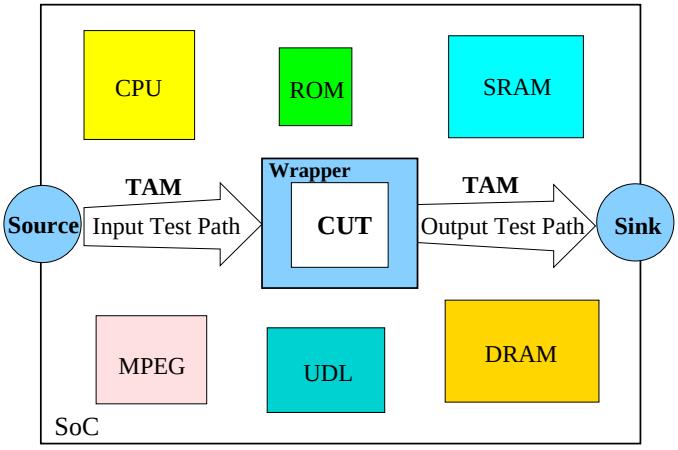


Figure 2.2: Architecture overview of embedded core-based SoC test.

The first one, namely, the *test pattern source and test pattern sink* creates test stimulus for core-under-test (CUT) and compares the responses to the expected results, respectively. Test pattern source and sink can be implemented either off-chip by external automatic test equipment (ATE), or on-chip data generators and response evaluators as used in many built-in self-test (BIST), or

as a combination of them. Off-chip ATE and on-chip BIST have their specific advantages and drawbacks. Quality and cost considerations, driven by both the circuitry type of the cores as well as IC-level optimization issues, determine the actual choice between ATE and BIST [21]. In theory all kinds of test patterns can be generated on-chip, but in practice only algorithmic patterns, such as the regular patterns for memories, functional patterns for analog modules, or pseudo random patterns for random logic can be generated on-chip without requiring an excessive amount of silicon area [22].

The second component is the *test access mechanism (TAM)* which is used to transport the test stimuli from the source to the CUT and transport responses from CUT to the sink. Many different TAM implementations exist; even on one SoC, different TAMs may coexist. Several TAMs have been proposed and currently used for testing core-based SoCs, such as Macro test [9], core transparency [10], multiplexed direct parallel access [11], Boundary Scan based test [12, 13], dedicated test bus [14] and TestRail [15], etc. The bus-based TAMs, such as TestBus and TestRail, provide the flexibility to make the trade-offs between test time and silicon area, in terms of its variable width for multiple TAMs which are connected in many different ways (see Figure 2.3). In [2], various TestRail configuration in the context of scan-testable cores are analyzed with respect to test time.

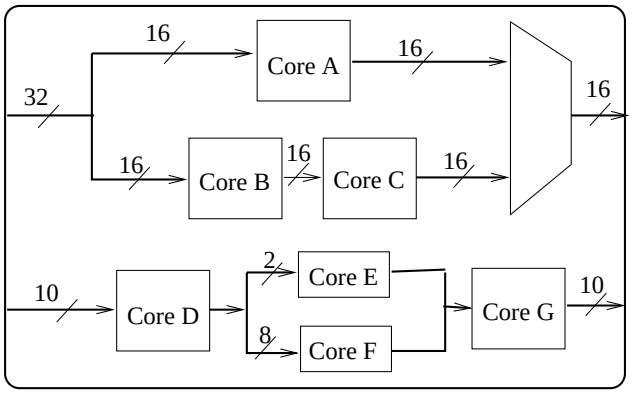


Figure 2.3: An example of Test Bus connection.

The two key parameters of any TAM are its width and length. The *width* refers to the TAM's transport capacity. An efficient TAM design involves the trade-offs between the transport capacity (bandwidth) of the mechanism and the test application costs it induces such as test time and area overhead. The TAM bandwidth is limited by the bandwidth of source and sink and the area available

for TAM wiring, while the test time is affected by the test data width of the individual cores and the TAM bandwidth [1]. More specifically, TAM width determines the maximum test data bandwidth, which in turn determines the maximum number of chip-level test vectors and the test application time. Thus, TAM configuration determines the possibilities for testing multiple cores in parallel and hence affects the outcome of test scheduling. The **TAM Configuration** problem is to obtain an optimal mix of the number and width of TAMs and the assignment of cores per TAM.

The *length* of a TAM is the physical distance it has to bridge between source and core or core and sink. By asserting core bypass, we can easily access the dedicated core. Thus an independent test path is established for the CUT by bypassing the surrounding environment. The cores on the same TAM which are not tested right now, can be put into bypass mode to shorten the access path for the CUT. Therefore, bypass increases the accessibility for the CUT, i.e., controllable and observable at its inputs and outputs respectively. Meanwhile, the interconnects are thoroughly tested since they are used to transport the test data from source to sink. Figure 2.4 shows the CUT and a test route between source and sink. As we can see, two test paths (Input/Output test path) are established between source and sink, which arises resource (source and sink) placement and TAM routing issues. The **TAM Routing** problem is to establish a shortest test path (fastest route) to carry test data for a CUT between the dedicated source and sink.

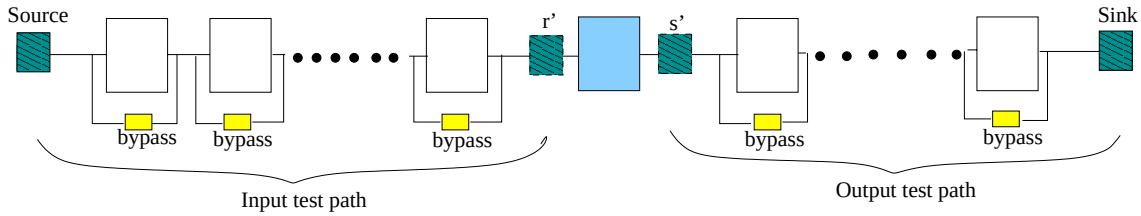


Figure 2.4: A general view of the test route [2].

The third component is the *core test wrapper* which functions as the interface between the cores and the rest of the SoC and TAM. Test conflicts can be minimized by placing the core in a wrapper. Several wrapper structures have been proposed, such as TestShell [15], TestCollar [14], IEEE P1500's wrapper [16], and analog wrapper [23]. According to the control signals from the wrapper control interface (WCI), the wrapper cells switch between *normal operation* / *core-internal*

test / *core-external test* modes. The wrappers may provide *width adaption* by serial-parallel or parallel-serial conversion, in case of a mismatch between the width of available TAM and the core input and output terminals [21]. The **Wrapper Configuration** problem is to partition a set of core-internal scan chains into m disjoint sets, one for each TAM chain, while minimizing wrapper scan chain length. The partitioning of the scan chains directly affects the test time T for a core as defined in [24],

$$T = \{1 + \max(s_i, s_o)\} \times p + \min(s_i, s_o) \tag{2.1}$$

where p denotes the number of test patterns, and s_i and s_o denote the scan-in and scan-out time for a core, respectively. Equation 2.1 can also be used to calculate the test time for non-scan-testable cores. In addition, the core bypass function is also enabled by the wrapper cells so that a test route is established for the CUT. The WCI is implemented as a state machine, which creates control signals to the wrapper cells on the basis of the global control signal, transports test data and enables test access to dedicated cores. The WCI can be enhanced with the embedded intelligence to execute the system-level test in a predetermined schedule, while the test sequencing of all the cores is embedded in a network of distributed modular controllers.

Chip-level SoC Test and Optimization

From the system integrators' point of view, an efficient test strategy is to test the system as a whole. The SoC test should cover the individual core test, the UDL test, as well as the test of their interconnects. In general, the test development in chip level should consists of expanding core-level tests to chip-level test, adding interconnect tests and chip-level test scheduling to minimize the test time. In order to select an efficient test strategy for an SoC, several performance criteria listed below need to be considered.

(1) *overall test time*. The overall test time of a testing scheme is defined as the period from the start time of the test activity to the end time when the last test task finishes. Note that, only when all test sets in parallel test queues finish their tasks, we say it's the end time of the test. In other words, the longest test queue dominates the overall test time. In addition, since the expensive testers are shared by many cores, the shorter the test time, the lower the cost is. The test time may be reduced

by using shorter test vectors or better scheduling schemes.

(2) *fault coverage*. In order to gain a high fault coverage, the individual embedded cores and UDLs should be tested thoroughly. This includes consideration of various fault models. In addition, the interconnections between different system blocks also need to be tested. Finally the system level testing should be processed to check the system functions.

(3) *area overhead*. The area overhead is the extra silicon area needed in order to perform the SoC test. The area overhead should be limited within a certain area budget, and kept as small as possible.

(4) *performance overhead*. As one undesirable side-effect of integrating test resources into the system, the power consumption of the SoC may increase while its speed may decrease. This performance overhead may vary when using different testing methods, and thus becomes a major performance criterion when evaluating various test strategies.

2.1.3 IEEE P1500 Scalable Architecture

IEEE P1500 (Standard for Embedded Core Test, SECT) is a standard under development with respect to various aspects of core-based testing, that aims at improving ease of reuse and facilitating interoperability with respect to the test of core manufacturing defects, especially when various cores of different sources are brought together in one SoC. IEEE P1500 handles two main requirements: easy integration and interoperability (plug-n-play) on the one hand, and flexibility and scalability on the other [25]. More specifically, it facilitates test reuse for embedded cores through core access and isolation mechanisms, and provides testability for SoC interconnect and logic. It also facilitates core test interoperability, with plug-n-play protocols, in order to improve the efficiency of test between core providers and core users.

Since 1997, the IEEE P1500 Working Group [26] is working towards a Standard for Embedded Core Test (SECT). IEEE P1500 focuses on standardizing a core test architecture which defines a core test interface between an embedded core and the SoC, i.e., the interface between core provider and core user. These tasks involve *core test knowledge transfer*, and *test access to embedded cores*. The corresponding two main components of the standard are a *core test language* and a *core test*

wrapper architecture. In the development of a core-based SoC test, the core providers prepare their cores with proper design-for-testability hardware and create test patterns for the cores, while the SoC integrator adds system-level design-for-testability and creates an SoC test using the core-level test as building blocks [1]. IEEE P1500 SECT does not cover the core's internal test methods or DfT, nor SoC test integration and optimization due to the fact that their requirements differ for different cores and SoCs. The IEEE P1500 SECT has two levels of compliance, *IEEE P1500 Unwrapped Core*, which does not have an IEEE P1500 wrapper, and *IEEE P1500 Wrapped Core*, which incorporates an IEEE P1500 wrapper function [25]. The motivation is to provide the flexibility required in testing core-based SoCs. Direct generation of a P1500-wrapped core provides the possibility to integrate the wrapper functionality with the core itself and hence minimizes the performance and area impact of the wrapper. By creating a 1500-unwrapped core first and then turning into a 1500-wrapped core, it allows to take the advantage of the scalability of the standardized wrapper and instantiate the wrapper with respect to the SoC system environment. An example application is illustrated in Figure 2.5 that shows the chip-level connections of the 1500-compliant cores with one serial TAM and one parallel TAM per core.

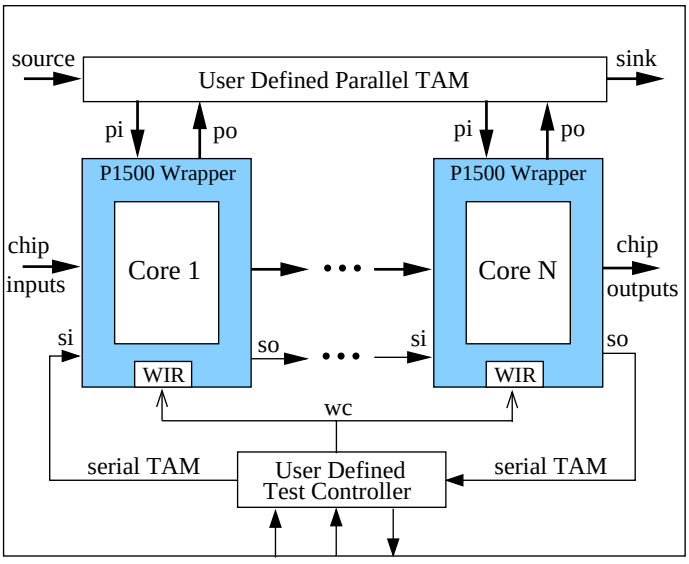


Figure 2.5: Overview of the P1500 scalable architecture [3].

Scalable Wrapper

P1500's wrapper is a thin shell around the core that provides the switching capability between the core and its various access mechanism [3]. In addition to a mode for connecting core inputs and outputs for functional operation, the wrapper has modes for connecting the core input and output terminals to a mandatory single-bit wide serial TAM, and zero or more scalable multi-bit TAMs. Figure 2.6 shows the wrapper architecture for an example core. The wrapper contains a *Wrapper Instruction Register* (WIR), controlling the operation of the wrapper, a *Wrapper Boundary Register* (WBR), consisting of multiple wrapper cells that provide controllability and observability on core terminals, a one-bit *Wrapper Bypass Register*, serving as a bypass for the serial TAM.

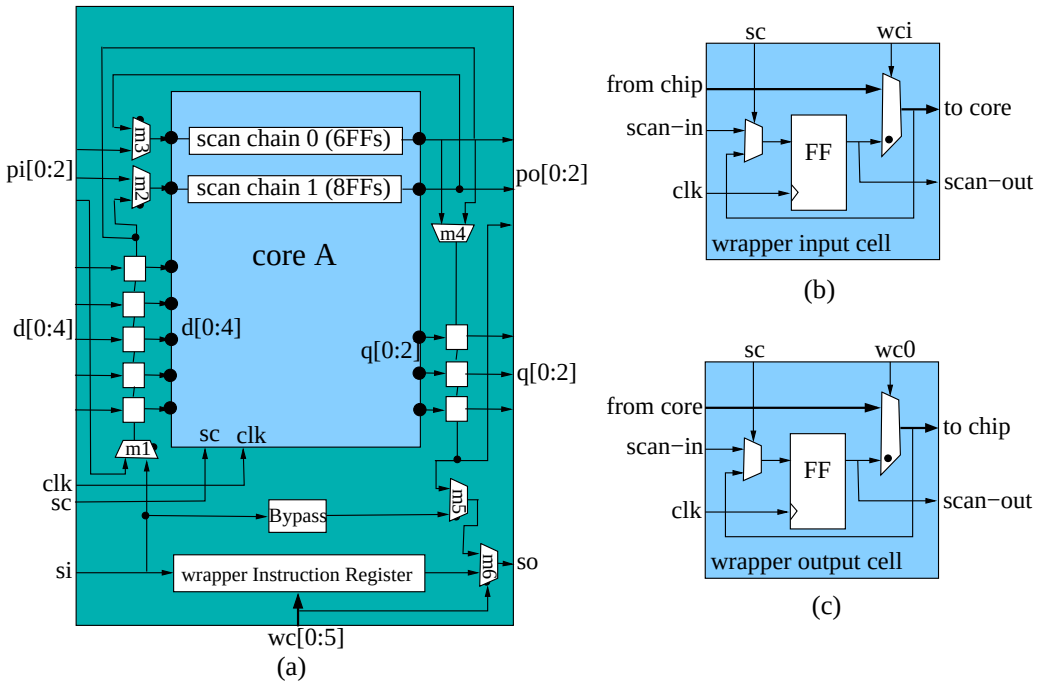


Figure 2.6: Example core A with P1500 wrapper (a) and wrapper input cell (b) and wrapper output cell (c) [3].

Core Test Language

IEEE P1500 SECT's Core Test Language (CTL) is a language for capturing and expressing test-related information for reusable cores [3]. Within CTL, one can create enough information at the boundary of the core to allow for successful instantiation of a wrapper, mapping of the core terminals

to wrapper terminals, reuse of the core test data, and testing of the user-defined logic and wiring external to the core.

2.1.4 Test Connectivity and Communication in Billion-transistor Era

According to ITRS'01 [5], at $65nm$ and below, design of very complex SoCs consisting of billions of transistors, operating below one volt and running at 10*GHz* will become a reality by the end of the decade. SoC design in the forthcoming billion-transistor era will involve the integration of numerous heterogeneous IP cores. Problems will arise from non-scalable global wire delays, failure to achieve global synchronization, errors due to signal integrity issues, bandwidth limitation, and difficulties associated with wired interconnects [27]. Although copper/low κ materials have been introduced for deep sub-micron interconnects, they may become insufficient as the technology goes below $100nm$. Recent studies have shown that the traditional hard-wired metal interconnect systems will eventually encounter fundamental limits and may impede the advances of future ultralarge-scale integrated systems (ULSIs) [18].

In the meantime, recent advances in silicon integrated circuit technology are making possible tiny low-cost transceivers to be integrated on chip. As a result, a new radio frequency (RF)/Microwave interconnect technology has been introduced for future intra-chip communication [18, 19]. In [19], the feasibility of employing on-chip wireless interconnects for clock distribution has been investigated. The tiny receivers, transmitters and on-chip zigzag antennae are implemented in $0.18\mu m$ TSMC CMOS technology with area consumption of $0.116mm^2$, $0.215\ mm^2$ and $0.15mm^2$, respectively. In particular, for a die size of $2.5cm$ microprocessor, the total area with one transmitter, 16 receiver and 17 antennae will consume about 1% area [28]. As the technology accelerates, new interconnect techniques (such as RF) and on-chip micro-networks (μNetwork) need to be introduced and developed for test connectivity and communication.

2.2 Related Work

With up to several hundreds of embedded components in a single package, the SoC manufacturing test becomes a bottleneck in the SoC design process. In this section, we briefly summarize previous

work that are related to our research.

2.2.1 Test Scheduling

System level scheduling is pursued to reduce the test cost (specifically, the test application time) by a certain level of parallelism while meeting the test quality. It may consume a significantly long period of time to test each embedded core successively, given that an SoC is embedded with hundreds of cores with extensive DfT techniques and resources. Several strategies have been proposed to shorten the test time, for example, compressing the test vector sets [29, 30], designing an appropriate test access architecture [2, 15, 31] or providing efficient system level test scheduling [32–34]. In contrast to the other approaches, test scheduling does not influence the test quality nor the SoC design, but it is still related to test cost. A good test schedule can help reduce test application time significantly.

Various approaches have been proposed for test scheduling. Some of them map the problem to shop scheduling [24, 32] or bin-packing [33, 35], some are defect-oriented [36], and the others are based on graph theory [37–41]. Among these approaches, [17] is one of the first ones to take into account the power dissipation issues in test scheduling. However, this work has mainly focused on the problem definition rather than proposing an algorithm to solve it. The first thorough analysis of the power-constrained test scheduling (PTS) at IC level has been performed in [40]. It proposes the use of a compatible test clustering technique, which is based on test compatibility to derive the power compatible set (PCS) and apply the weighted covering table minimization technique to obtain the optimum schedule. However, this work is limited to a theoretical analysis. In addition, the computation is quite excessive due to the enormous covering tables generated. Recently, an extended tree growing approach has been proposed to exploit the potential of test parallelism by expanding the compatible tree via merging the block-test intervals of compatible subcircuits [41]. Although this approach tries to fill in the idle time with shorter tests based on the compatibility relations among the tests, it has the limitation imposed by the test session boundaries, which means, no test can stretch across two continuing test sections. [42] has mapped the test scheduling problem to open shop and flow shop scheduling models, and solved it through a mixed integer linear programming (MILP) approach combined with precedence, preemption and power constraints. However,

the computation time of MILP grows exponentially with the number of cores and test resources. [35] has transformed the problem to bin-packing and adopted a heuristic Best-fit algorithm to map the pins of embedded cores to SoC I/O pins or configure TAM (test access mechanism) buses. However, their work has focused on TAM configuration to reduce the test time rather than the scheduling of test sets under various constraints. [33] has proposed a test parallelization combining scheduling scheme to minimize test time under power limitation. But the problem is quite simplified by the assumption of the linear dependence of test time and power on scan chain subdivision.

2.2.2 Test Access Mechanism Design and Test Wrapper Optimization

A test access architecture, in the form of dedicated design-for-testability hardware guarantees test access from the chip pins to the embedded cores and vice versa [15]. A number of test access strategies and TAM structures have been proposed for testing core-based SoCs, such as Macro Test [9], core transparency [10], multiplexed direct parallel access [11], Boundary Scan based test [12, 13], dedicated test bus [14] and the TestRail [15], etc. Bus-based TAMs, such as TestRail [15], appear to be the most promising, since they provide the flexibility to trade-off between the test time and silicon area in terms of its variable width connected in different ways (i.e., Muxed, bypass, merge and fork). Several types of wrappers have been proposed, such as TestShell [15], TestCollar [14], IEEE P1500's wrapper [16], and analog wrapper [23]. IEEE P1500 has standardized the core wrapper structure to ease of plug-n-play for testing, while maintaining the required flexibility to cope with different cores and SoCs [16]. Based on a specific TAM structure and wrapper architecture, TAM configuration [2, 31] and wrapper optimization problems [15] have been addressed independently.

Recently, several approaches have been proposed for the integrated framework of TAM optimization and test scheduling because of the dependency of test time calculation on TAM configuration and wrapper adaptation. We can classify these approaches into two categories, partition-based [43–46] and geometric packing [35, 47, 48]. The partition-based approaches are usually formulated into Integer Linear Programming [44], network transportation [45], etc. In these approaches, the top level TAM width W_{max} is first partitioned into M fixed width TAMs $w_1, w_2, ..., w_M$ (or further subdivided into fixed width sub-TAMs), irrespective of test data needs of individual cores and

test constraints between the cores. The major disadvantage of these approaches is the inflexibility of fixed TAM partitioning resulting in inefficient test scheduling on TAMs. Goel and Marinissen improve TAM partitioning in TR-Architect [49] by efficiently determining the number of TAMs and their widths through multi-step optimization by merging and redistributing the critical TAMs and/or the bottleneck TAMs. Geometric packing, specifically the bin packing approaches [35, 48], provides efficient and effective test scheduling by assigning flexible width TAMs to the cores when allocating individual test sets. In these approaches, there is no explicit partitioning on the total TAM width, thus eliminating the constraints on assigning some cores to a particular width TAM.

Iyengar et al. [48, 50] have proposed a rectangle-packing approach for wrapper/TAM co-optimization and test scheduling, where the rectangles, whose length and height represent the test time and TAM width of the test respectively, are allocated into a two-dimensional bin with fixed height of W_{max}, while minimizing the unbounded length, i.e., the overall test application time. However, in this approach, scheduling efficiency strongly relies on the initial calculation of the preferred TAM width for each test set. Although the TAM width of a few cores may vary when they are used to fill in the idle time, it is fixed at the preferred value for most cores, and thus results in loss of flexibility to dynamically adjust the TAM width and the test time. Huang et al. have proposed in [4] a restricted 3-dimensional bin packing approach to optimize test time under pin and power constraints. In this approach, a set of cubes with their 3 dimensions being the wrapper width (width), the peak power (length) and the test time (height) respectively are selected and packed into a 3-D bin to meet the pin count limit and the power constraint while minimizing the height, i.e., the overall test time. However, the division of sub-bins on the TAM width dimension explicitly adds the constraints on assigning some cores to particular partitioning of TAMs. In addition, the 3-D bin is divided based on the initial test times of the cores in the first level by a simplified linear relation with their peak powers. Additionally, in all the above-mentioned bin-packing approaches, the test constraints are checked only when allocating an individual test. That means, when scheduling a new incoming test T_i in parallel with scheduled tests T_j and T_{j+1}, they should check whether it meets test constraints, such as resource conflict, precedence constraint and maximum power limitation, etc. Since the test compatibility is not thoroughly utilized, the tests scheduled later have less choices to handle

constraints effectively. The more the test constraints are integrated into the system, the worse the scheduling efficiency becomes.

2.2.3 Test Control Network

The control network executes the test application based on a predetermined schedule. Most test control schemes [17, 37, 51, 52] use a hierarchical test control methodology employing distributed controllers. Standard interfaces are used for communication between the control units at different levels of the hierarchy. [37] proposes a hierarchical test control architecture, within which each module is associated with a STU (self-testable unit). A hierarchy of supervisors is used to control the test of the entire chip in a way that the top level supervisor communicates with the ATE and controls the lower level supervisors which in turn control the STUs. In [17], three types of hierarchical test controllers provide a structured division of control functions: external BIST access port, BRCs (BIST Resource Controllers) and a distributed BIST control network, which results in uniform and simplified interface protocols between three control levels. Hierarchical test models for core-based system-on-chips are introduced in [51, 52], which are capable of testing not only JTAG (IEEE 1149.1) cores and CTAG (IEEE P1500) cores, but also the hierarchical cores (cores integrated in a hierarchical fashion). Note that, a vast majority of SoC interconnect networks are using a mix of buses and various forms of point-to-point data or control links. When moving into the billion-transistor era, transmitting signals chip-wide through wires will become more difficult.

Recently, the concept of using an on-chip network as the fundamental communication architecture for a complex SoC design has been proposed in [53–55]. To surpass the fundamental limitation of conventional hard-wired interconnects, [18] has introduced an RF/wireless interconnect for future inter- and intra-chip communications, which is based on capacitive coupling, low loss and dispersion-free microwave signal transmission, and modern multiple-access algorithms. With the integration of tiny antennae, receivers and transmitters, an intra-chip wireless interconnect system is proposed in [19] for clock distribution at a chip distance of $5.6mm$. As the technology accelerates, new interconnect techniques (such as RF) and on-chip micro-networks (μNetwork) need to be introduced and developed for test connectivity and communication. Moore et al. [56] have applied

for the first time, the concept of wireless technique for on-wafer testing.

2.3 Summary

This chapter has provided the background and motivation for our research on SoC testing. Some of the ideas presented in the literature have been extended and several new proposals have been made to address the SoC testing problem as described in the remaining chapters of this dissertation.

Chapter 3

Resource Balancing Based Test Scheduling

In this chapter, we formulate the test scheduling problem for embedded core-based SoCs as a shortest path problem. We first consider a system where one test set needs to be selected for each core from a group of alternative test sets using different test resources, and propose a novel test scheduling algorithm to reduce the overall testing time. Then, we extend the algorithm to support multiple test sets selection for each core. The basic idea is to effectively construct a shortest path going through each core exactly once, while simultaneously balancing the parallel resource usage [57].

3.1 Rationale

Most of the existing test scheduling approaches assume that all of the given test sets have to be used in testing. Although test scheduling with multiple test sets has been introduced in [34] and an MILP model has been developed in [32], their work focuses on selecting a test set for each core from a set of alternatives with a varying proportion of BIST and external test patterns, which is just a special case of the problem being studied in our research. We assume that a core may be provided with several test configurations, each corresponding to a test set, or a core may consist of several functional blocks or submodules, each of them requiring a different test method in order to meet the fault coverage requirement. For example, a core may be provided by core vendors with several

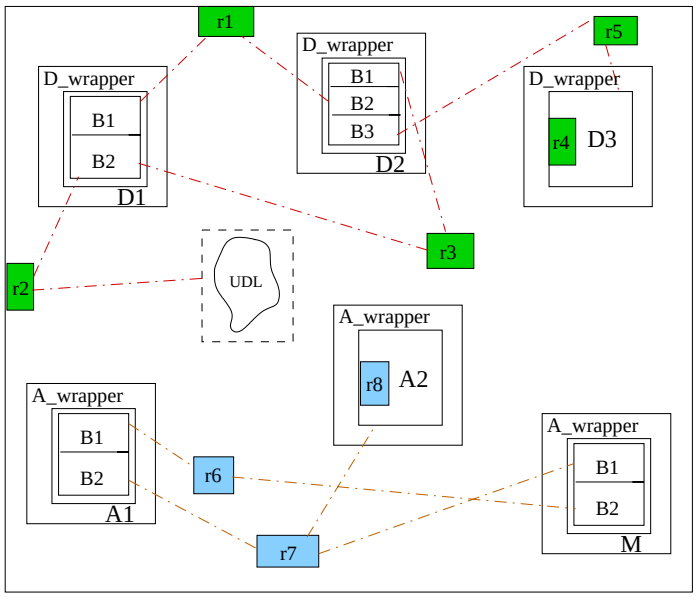
precomputed test patterns to provide flexibility for different system needs. Moreover, as system integrators can purchase cores from various core vendors, multiple core vendors may provide cores with similar functionality but different test configuration. In this case, we may consider the test sets for a core with similar functionality as a group of candidates (i.e., alternate tests). The system integrators need to select one from each group for their system.

We first propose a scheduling algorithm for the case where only one from a group of test sets may be selected for each core to perform testing, and take into consideration the test conflicts and the fault coverage requirements. Our method subsumes the problem of constrained scheduling, where some tests may not be executed concurrently due to resource conflicts. In those existing approaches, for example, the MILP formulation, additional constraints have to be added to formulate the problem thus increasing complexity. However, we map the test resources into parallel queues, and the nodes competing for the same resource will sequentially enter the particular resource queue, thus no additional constraints are necessary. The goal behind our formulation is that we expect to minimize the overall testing time by shortening the usage time for any particular test resource. Thus we view test resources as queues and the test to be scheduled as the job entering corresponding queue. The test scheduling problem is deduced to minimizing the longest queue length which represents the overall testing time. In order to solve this problem, we formulate it as a single-pair shortest path problem by representing vertices as test sets, directed edges between vertices as a segment of a schedule sequence, and the edge weight as the test time of the test set at the end of the segment. Thereby, the original problem becomes finding a shortest path from the source to the destination by going through each core exactly once.

3.2 SoC Modeling

A general SoC model is shown in Figure 3.1, which consists of digital cores (D_1, D_2 and D_3, for example), analog cores (A_1 and A_2) and mixed-signal cores (M) as well as UDLs which can be treated as cores so as to unify the formulation. In order to facilitate test reuse, a test access architecture, which consists of test wrappers, test access mechanism (TAM) and test source and sink, is constructed for individual cores embedded in the SoC so that the tests can be applied and

the responses can be observed at the chip level. The wrappers are logic structures that surround the cores to support both core isolation and test access to IP cores during test operation. The TAM works as “test data highway” which propagates test patterns from the test pattern source to the core-under-test (CUT) and test responses from the CUT to the test pattern sink, as well as the control signals to perform SoC test in a predetermined schedule. In addition, each core may include several functional modules, and each block may be tested by one or multiple test sets using one or multiple resources, thus to provide flexibility for test scheduling. As we can see, if analog, digital and mixed-signal cores do not share resources (for example, the mixed-signal tests must be executed on a special mixed signal test bus like IEEE 1149.4 test bus), they can be separated and tested in parallel using the same scheduling technique, as shown in Figure 3.2.



r1 − r8 are different test resources
D1 − D3 are digital cores
A1 − A2 are analog cores
M is a mixed−signal core
B_i is denoted as different functional blocks in a core

Figure 3.1: A general SoC model.

3.3 System Definition and Assumptions

Before introducing the system definition, we list the assumptions made.

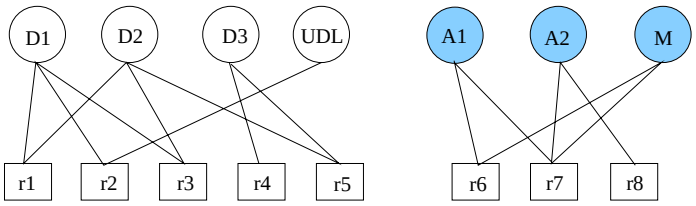


Figure 3.2: Graph representation of resource sharing.

1. It is assumed that an SoC is embedded with n testable cores with m test resources. A core may need to perform one test (or several tests) by using one resource (or several resources) to meet the required fault coverage.

2. Test resources are defined as test buses, BIST, or any specific set of circuit blocks for certain test configuration. For instance, the circuit blocks, i.e., the test control logic, TPG, compressors/analyzers, and any intervening logic, needed to execute BIST test on core c_i are grouped as test resource r_j for c_i.

3. A collision occurs when the tests sharing the same resource or the tests for the same core are performed in parallel. Therefore, a core can be tested by one test set by using certain resource at one time. A resource can be shared among several cores, but only one resource can be used by a given core at a given time.

4. Each test set includes a set of test vectors. Different test sets may have different test times by using different test resources. In other words, core vendors may have provided a set of alternative tests, and one test from each group needs to be performed to achieve the required fault coverage.

Given the test times and the required fault coverage, the goal of the scheduling technique is to efficiently determine the start times of the test sets to minimize the total test application time.

Formally, we define the SoC model as *TM*=$\{C, RSC, T, FC\}$, in which *C*=$\{c_1, c_2, ..., c_n\}$ is a finite set of cores, *RSC*=$\{r_1, r_2, ..., r_m\}$ is a finite set of resources, *FC* is the fault coverage required to test each core, and *T*=$\{T_{11}, T_{12}, ..., T_{1m}, ..., T_{n1}, T_{n2}, ..., T_{nm}\}$ is a finite set of tests, which is shown as an $n \times m$ matrix in Figure 3.1. Test set T_{ij} represents a test set for testing core c_i by

T_{ij}	r_1	r_2	r_3	. . . . .	r_m
c_1	T_{11}	T_{12}	n/a	. . . . .	n/a
c_2	n/a	T_{22}	n/a	. . . . .	T_{2m}
c_3	T_{31}	n/a	n/a	. . . . .	n/a
c_4	n/a	n/a	T_{43}	. . . . .	n/a
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	. . . . .	.
.	.	.	.	.	.
.	.	.	.	.	.
c_n	T_{n1}	T_{n2}	T_{n3}	. . . . .	T_{nm}

Table 3.1: Matrix representation of test sets.

using resource r_j, and has a test time of t_{ij}. The entries with n/a indicate that such test sets are not available.

3.4 The Resource Balancing-based Test Scheduling Algorithm

We introduce a new scheduling algorithm for SoC testing. The basic idea of this approach is to map the test sets to a directed graph with weighted edges, and apply the shortest path algorithm to obtain the best testing scheme [58].

3.4.1 Problem Definition

We consider a system discussed in Sec. 3.3 and assume that one core needs only one test set (selected from a number of candidates) to achieve the required fault coverage (however, this assumption will be nullified in the extended approach to be discussed in Sec. 3.6). According to the matrix shown in Table 3.1, we can construct a graph with $m \times n$ vertices, one for each entry in the matrix (see Figure 3.3).

- If an entry is n/a (which indicates that the test is not available), it is mapped to a dummy vertex (see the shaded circles in Figure 3.3).
- A vertex T_{ij} (i representing the core index and j representing the resource index) is connected to all vertices (except the dummy ones) in the next column (i.e., $T_{i+1,k}$, $1 \leq k \leq m$) with

directed weighted edges.

Definition 1 : *The weight of an edge connecting vertices* T_{uv} *and* T_{ij} is defined as a vector, $w(T_{uv}, T_{ij}) = (0, ..., t_{ij}, ..., 0)$ (only the jth entry, corresponding to resource r_j, has a value of the test time for T_{ij}, while other entries are zeros). The major motivation behind using the weight assignment is to allow the shortest path algorithm to consider, and moreover, balance the usage of test resources.

- In addition, two special nodes, the source s and the destination d are added. Node s connects the vertices T_{1k} ($1 \leq k \leq m$) with the weight of $w(s, T_{1k}) = (0, ..., t_{1k}, ..., 0)$. The vertices T_{nk} ($1 \leq k \leq m$) connect to node d with a weight of (0,0,...,0).

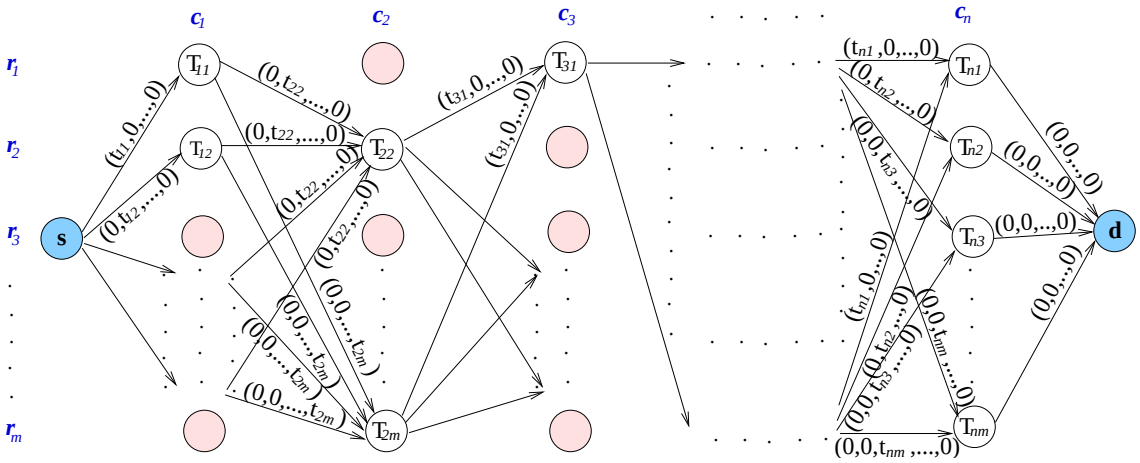


Figure 3.3: The graph constructed from the $n \times m$ matrix.

Definition 2 : For a path p from a vertex $T_{i+1,j}$ to s including the vertices $T_{1k_1}, T_{2k_2}, ..., T_{ik_i}$ ($k_1, ..., k_i$ can be any value between 1 to m), *the length of the path* $p = (s, T_{1k_1}, T_{2k_2}, ..., T_{ik_i}, T_{i+1,j})$ is denoted by the distance vector $W(p : T_{i+1,j} \rightarrow s) = (D_1^p, D_2^p, ...D_k^p, ..., D_m^p)$, where $D_k^p = \sum_{i=1}^{n} t_{ik}$ is the sum of the test time shown in the kth entry of the weights of all edges along the path p. In addition, *the predecessor of* $T_{i+1,j}$ *on the path* p *to* s is recorded in $\tau_{i+1,j}$.

We define m queues in parallel corresponding to m resources that may be used at the same time independently [59] (see Figure 3.4). The length of the queue denotes the total testing time of all test sets using the resource. For example, as we can see from Figure 3.3, the nodes on the

first row enter resource queue r_1 depending on whether the path is going through them, and their weights contribute to D_1^p of $W(p : s \rightarrow d)$. Since the longest queue length dominates the overall test time of a schedule, the absolute value of the path distance $|W(p : s \rightarrow d)|$ is defined as $\max\{D_1^p, D_2^p, ...D_k^p, ..., D_m^p\}$. Accordingly, the test scheduling problem can be converted to the problem of finding a shortest path from s to d. More specifically, the SoC test scheduling problem $STS([T_{ij}], s, d)$ can be formulated as follows.

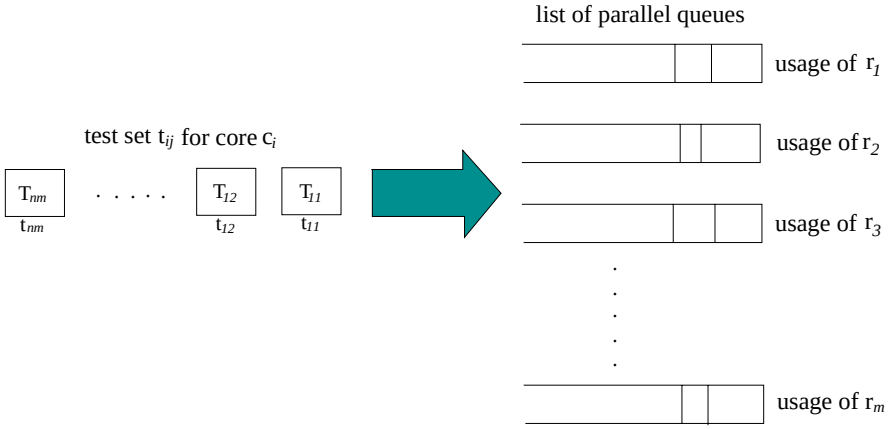


Figure 3.4: Parallel usage of test resources.

$\underline{STS([T_{ij}], s, d)}$: Given an SoC represented by an $n \times m$ matrix, construct a weighted, directed graph $G(V, E)$ (where V includes $m \times n$ vertices), with m-tuple weight function $w(T_{uv}, T_{ij}) = (0, ..., t_{ij}, ..., 0)$ for some edges. The length of a path p is the sum of the weights on corresponding resource tuples of its constituent edges. We define the shortest-path weight from source s to destination d by

$$\delta(s, d) = \begin{cases} \min\{|W(p : s \rightarrow d)|\}, & if\ exists\ path\ s \rightsquigarrow^p d \\ \infty, & otherwise \end{cases} \tag{3.1}$$

The objective is to find a path p from source s to destination d such that $W(p : s \rightarrow d) = \delta(s, d)$.

3.4.2 The Schedule with Modified Single-Pair Shortest-Path (SPSP) Algorithm

Dijkstra's algorithm [60] is a well-known approach to solve the single-source shortest path problem when all edges have nonnegative weights. A variation of this algorithm can be used to find the

shortest path from s to d of the graph shown in Figure 3.3. More specifically, each vertex T_{ij} maintains a m-tuple vector as the distance to the source s, $W(p : T_{ij} \rightarrow s) = (D_1^p, D_2^p, ..., D_m^p)$, and τ_{ij} to record its predecessor on the shortest path to s. Each vertex may be in one of the following three states:

- **state 1**: not updated; the distance vector is $(\infty, \infty, ..., \infty)$.
- **state 2**: updated; the distance vector has been updated at least once.
- **state 3**: finalized; the distance vector is the shortest distance to s, and it will not be updated in the future.

Initially, s is finalized (i.e., in state 3), with the distance vector $W(p : s \rightarrow s) = (0, 0, ..., 0)$, and all other vertices are not updated (i.e., in state 1), with the distance vector $W(p : T_{ij} \rightarrow s) = (\infty, \infty, ..., \infty)$. s will update the distance vectors of all its neighbors (T_{1k}, $1 \leq k \leq m$, in Figure 3.3). More specifically, $W(p : T_{1k} \rightarrow s) = w(s, T_{1k})$, $\tau_{1k} = s$, and the states of these vertices will be changed to *state 2*. Then, one of the vertices in state 2 with the smallest $|W(p : T_{1k} \rightarrow s)|$ will be selected and finalized. Again, it will update the distance vector of all of its neighbors. In general, when a vertex T_{ij} (with the smallest $|W(p : T_{ij} \rightarrow s)|$ among all vertices in state 2) is finalized, it will update the vertices $T_{i+1,k}$ ($1 \leq k \leq m$). If $W(p : T_{i+1,k} \rightarrow s) > W(p : T_{ij} \rightarrow s) + w(T_{ij}, T_{i+1,k})$, then $W(p : T_{i+1,k} \rightarrow s) = W(p : T_{ij} \rightarrow s) + w(T_{ij}, T_{i+1,k})$, and $\tau_{i+1,k} = T_{ij}$. This algorithm will continue until the vertex d is finalized. The pseudocode is provided in Appendix 8. It can be shown that, the worst-case time complexity of the initialization step is $\Theta(V)$, and the computation is dominated by Priority Queue operation $\Theta(E \log V)$, where V is the number of vertices and E is the number of directed edges in Graph G. Therefore, the worst case time complexity of this algorithm is $\Theta(E \log V) = \Theta(m^2 n \log(mn))$, where m is the number of resources and n is the number of cores in the system, respectively.

Figure 3.5 shows an example of applying the algorithm for a core-based system with 7 cores and 4 resources (as shown in Table 3.2). We first construct a graph (see Figure 3.5(a)) as described in Sec. 3.4.1, then we apply the modified SPSP algorithm to find the shortest path from s to d (see Figure 3.5(b)). As we can see from Figure 3.5(b), a shortest path from s to d includes tests T_{14},

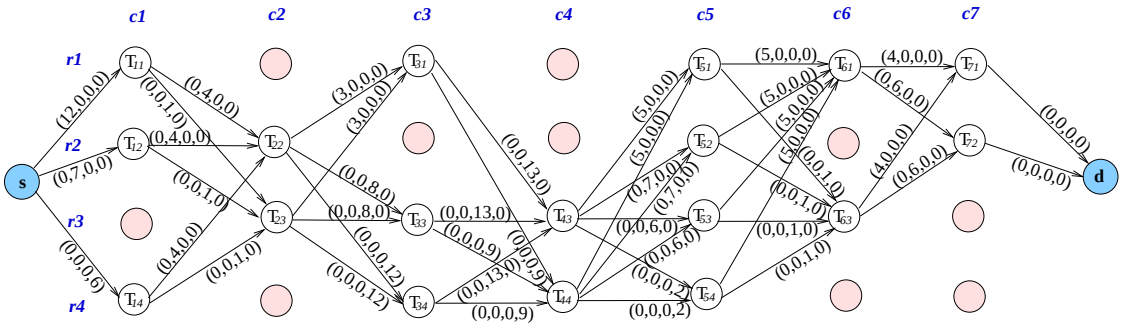
T_{ij}	r_1	r_2	r_3	r_4	P
c_1	12	7	n/a	6	3
c_2	n/a	4	1	n/a	2
c_3	3	n/a	8	12	3
c_4	n/a	n/a	13	9	2
c_5	5	7	6	2	4
c_6	5	n/a	1	n/a	2
c_7	4	6	n/a	n/a	2

Table 3.2: The matrix of test sets for an example system.

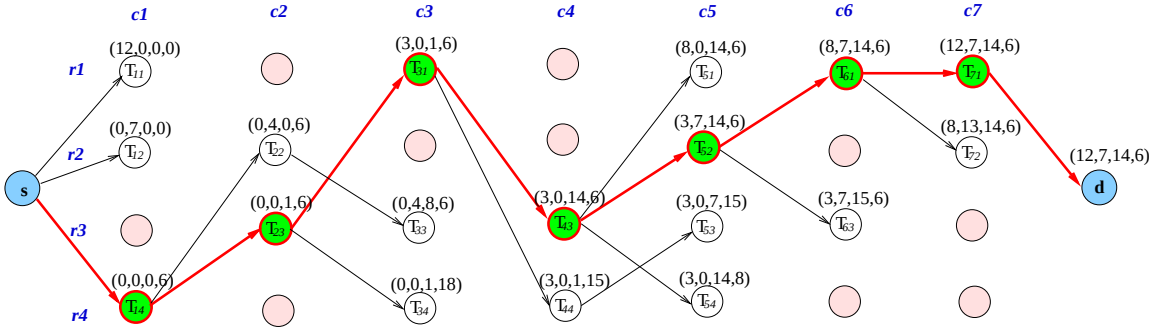
T_{23}, T_{31}, T_{43}, T_{52}, T_{61} and T_{71}, within which T_{31}, T_{61} and T_{71} are the tests sequentially entering resource queue r_1, T_{23} and T_{43} sequentially enter resource queue r_3, and T_{52} and T_{14} are the only test sets using resource r_2 and r_4, respectively. The test sets in different resource queues can be applied concurrently. The distance vector of d, $W(p : d \rightarrow s) = (12, 7, 14, 6)$, represents the total test time on the corresponding resource queue. As we can see, queue r_3 is the longest resource queue which results in an overall test time of $|W(p : d \rightarrow s)| = 14$. We convert the shortest path from s to d into a way of resource usage of the cores as shown in Figure 3.6(a). As a matter of fact, the shortest path from s to d is constructed by balancing the resource queue lengths. In addition, as we have noticed, one of the advantages of the proposed approach is that there is no idle time between successive tests (namely, *explicit dead time*) in any of the queues.

Proposition 1 *A shortest path from source s to destination d is constructed in a way that balances the resource usage queues.*

Proof : As shown in Figure 3.4, we have defined m queues in parallel corresponding to m resources. Meanwhile, the graph on which the modified shortest path algorithm is applied is constructed in a way that the rows are corresponding to the resource usage queues while the columns are corresponding to the cores in the SoC. In other words, the vertices in column i represent all the tests for core c_i and the vertices in row j are the candidate tests entering resource queue r_j. As we have discussed earlier, each vertex maintains a m-tuple vector as the distance to source

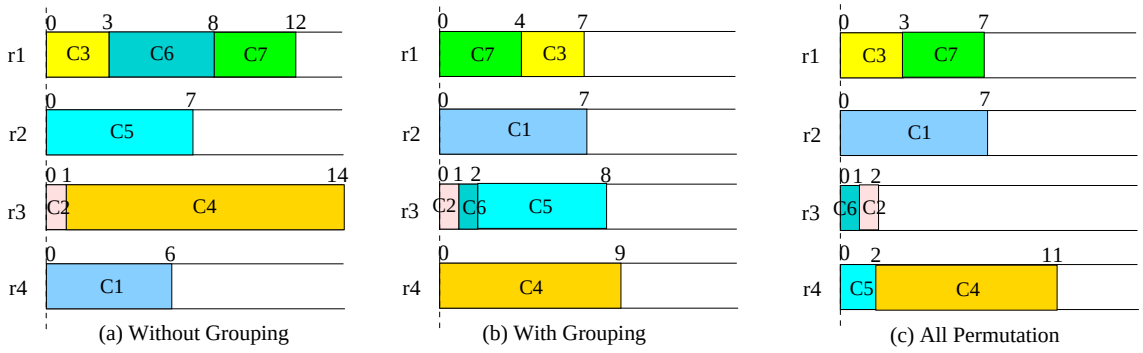


(a) The Graph of The Example System



(b) The Schedule Without Grouping

Figure 3.5: The scheduling with the modified SPSP algorithm.



(a) Without Grouping (b) With Grouping (c) All Permutation

Figure 3.6: The final schedule illustrated on parallel queues.

s, $W(p\colon T_{ij} \rightarrow s) = (D_1^p, D_2^p, ..., D_m^p)$. Each vertex T_{ij} in the graph is updated when a shorter longest queue length can be reached when going through vertex $T_{i-1,k}$, i.e., $W(p\colon T_{ij} \rightarrow s) > W(p\colon T_{i-1,k} \rightarrow s) + w(T_{i-1,k}, T_{ij})$. Vertex T_{ij} is finalized when its longest queue length $|W(p\colon T_{ij} \rightarrow s)|$ is the smallest among updated vertices. That means, a shortest path is constructed from s to T_{ij}. Thus, the shortest path from s to d is constructed by balancing the resource queues to minimize the length

of the longest queue which dominates the total test time. ∎

Proposition 2 *There is no "explicit dead time" in this resource balancing approach.*

Proof : Explicit dead time arises due to resource conflicts. There are two types of resource conflicts defined in our system. 1) Several tests compete to use the same resource; 2) Different tests for the same core are executed at the same time. Conflict of the first kind is totally overcome by the resource balancing approach, since the tests competing for a resource sequentially enter the resource queue. Although for each core a set of tests is provided, only one of them will be executed to test the core. So the conflict of the second kind is eliminated. ∎

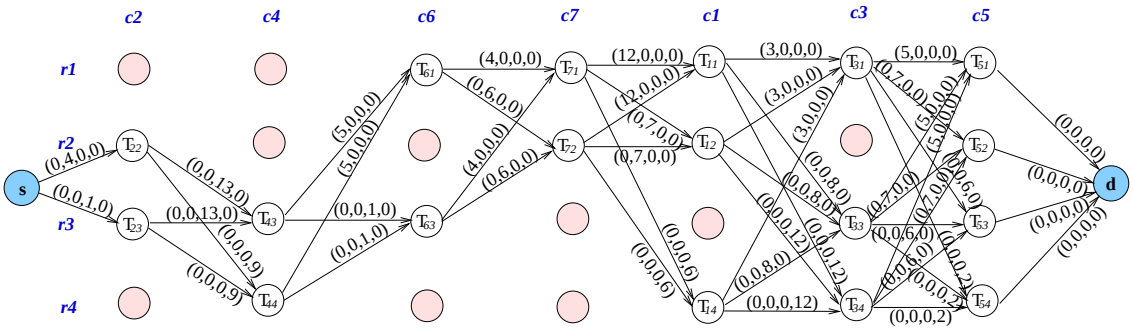
3.4.3 Grouping Scheme

As there is no explicit dead time in resource balancing, our purpose is to effectively reduce the *implicit dead time* (i.e., the idle time appearing at the beginning or the end of a schedule) at the end of the resource queues (obviously, no implicit dead time appears at the beginning in this approach). Because the shortest path from s to d is set up by going through certain test set of each core from left to right, different ordering of ready-to-schedule cores (i.e., the cores before entering the resource queues) results in different schedule of tests, and accordingly the total test time.

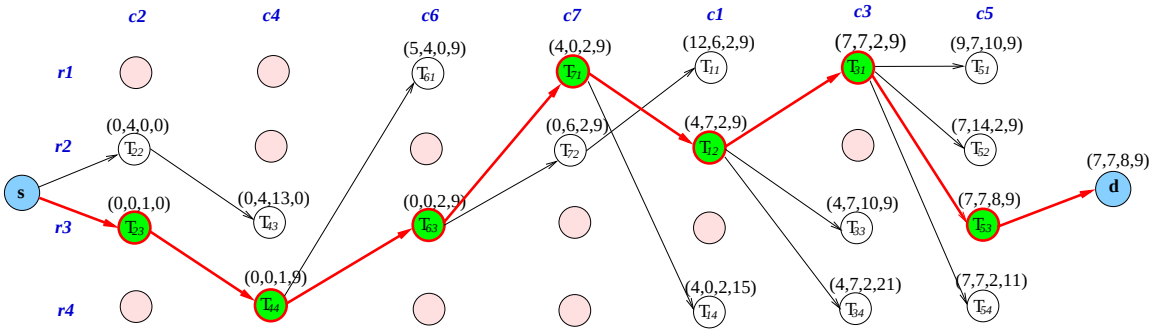
We group the cores based on the number of available tests they have, such that in a group G_p, all cores have P alternate test sets. This is a one time effort. For example, the right most column of Table 3.2 shows the P value of each core. The cores in the group with smaller P value will be scheduled earlier, because these tests have to be put into certain queues (i.e., the corresponding cores have to be tested by using certain resources). Then, we schedule the test sets in the group with larger P value to balance the lengths of the resource queues, and accordingly shorten the longest queue length. Balancing queue length ultimately results in shorter overall test time.

Figure 3.7 illustrates the execution of the algorithm with grouping for the given example. Core c_2, c_4, c_6 and c_7 have 2 alternate test sets and are scheduled first. With the P value of 3, c_1 and c_3 are the cores to be scheduled next, and finally c_5 with P of 4 is scheduled. After the graph is constructed, the shortest path algorithm described in the previous subsection can be employed here

to find the shortest path. Figure 3.6(b) shows the resource usage of the final schedule. Compared to the case without grouping, the total test time is reduced by 30% by using the grouping scheme. As we can see, grouping the cores properly before scheduling can reduce the total testing time and achieve better balancing of resource usage, while the worst case time complexity remains the same.



(a) The Graph After Rearranging The Cores



(b) The Schedule With Grouping

Figure 3.7: The scheduling with grouping scheme.

3.4.4 All Permutation Scheduling

As we have discussed above, different ordering of the cores will affect the performance of the schedule significantly. To perform test scheduling on a system embedded with n cores, there are $n!$ ways for the ordering of the cores. Thus the optimal schedule can be determined after running $n!$ times of the SPSP algorithm on all $n!$ different ordering of cores. Clearly, the computation is quite excessive (the worst case time complexity is $\Theta(n!m^2 n \log(mn))$).

One way to reduce computational complexity while eliminating the effect on core ordering is to construct a single graph with all possible permutations of the cores and running the SPSP algorithm

T_{ij}	r_1	r_2	r_3	r_4
c_1	T_{11}	T_{12}	n/a	T_{14}
c_2	n/a	T_{22}	T_{23}	n/a
c_3	T_{31}	n/a	T_{33}	T_{34}

Table 3.3: The test sets for all permutation scheduling.

once. We call this kind of scheduling *All-Permutation Scheduling*. In this model, the graph is constructed with the size of $m \times n \times n$. We list all the tests (including the dummy nodes) for the cores in one column and copy by n times, thus there are $n!$ possible ways for the ordering of these cores. Note that, when we connect a vertex in column i ($1 \leq i \leq n-1$) to a vertex in column $(i+1)$, they should not belong to the same core. We use a simple example for illustration, which includes 3 cores and 4 resources as shown in Table 3.3. We first construct the graph as shown in Figure 3.8. In this way, we consider all $3! = 6$ permutations of the three cores in one graph: (1) c_1, c_2, c_3; (2) c_1, c_3, c_2; (3) c_2, c_1, c_3; (4) c_2, c_3, c_1; (5) c_3, c_1, c_2; (6) c_3, c_2, c_1. With the graph ready, the shortest path algorithm discussed in Sec. 3.4.2 can be employed on this graph with a minor modification that a vertex T_{ij} cannot update those neighbors in the next column if the cores they belong to have already been included in the shortest path of T_{ij} to s. Therefore, for each vertex T_{ij} we maintain a record that traces the cores which consist of the shortest path from T_{ij} to s. Note that, the all-permutation scheduling does not result in an optimal schedule, because when we construct the shortest paths for the vertices to s, some among the $n!$ permutations will not be taken into consideration anymore since those finalized vertices will not contribute to the shortest path from s to d.

We apply the all-permutation scheduling (AP, for short) on the same example used by the schedules with/without grouping (WG and WOG for short, respectively), the schedule result is shown in Figure 3.6(c) (We do not show the graph of all-permutation scheduling due to its complexity). When we compare the results of the three approaches, an interesting observation is that although AP approach considers all possible permutations of the cores at one time, it does not result in a better performance than WG approach. It is because balancing the resource usage queues at the earlier stage does not guarantee the final result to be well balanced. For example, Figure 3.9 shows the

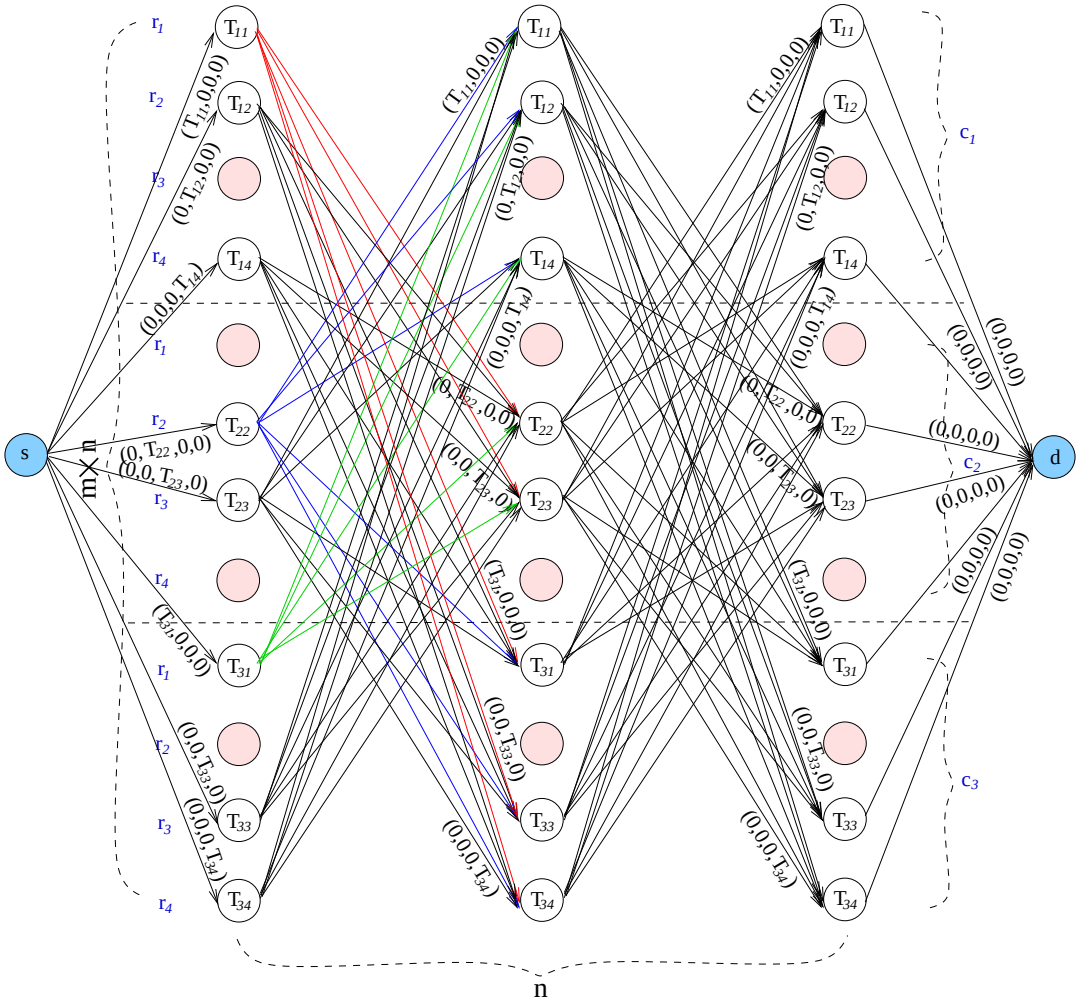


Figure 3.8: The graph constructed for all permutation scheduling.

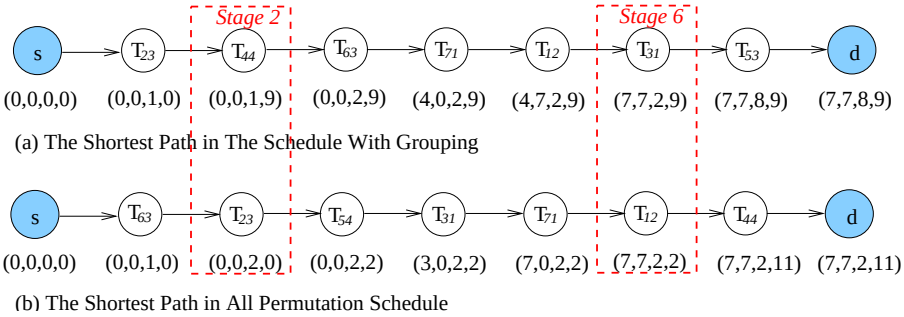


Figure 3.9: Comparing the shortest paths in the schedules with WG and AP approaches.

shortest paths constructed by WG and AP approaches on the example system. Although, AP results in more balanced queues from stage 2 to 6 (for instance, when in stage 2, the longest queue

length in AP is 2 while in WG 9), it leads to a longer longest queue length (11) than that in WG (9). As we can see, till the last stage, WG balances the queues by inserting tests into other queues than the longest queue. While in AP, it balances the resource queues well at the beginning (till stage 6, the longest queue length in AP is 7), but in the final stage, it cannot result in more balanced queues rather than increasing the length of the queue of r_4 (which results in the longest queue length finally). In addition, all-permutation scheduling results in much higher complexity, $\Theta(E \log V) = \Theta(m^2 n^3 \log(mn^2))$.

3.5 Simulation Study

We evaluate the proposed scheduling algorithms by implementing them in C and running simulations on Sun Enterprise 450 Workstation with four 450MHz UltraSPARC-II CPUs. We define the balance ratio as G as given below:

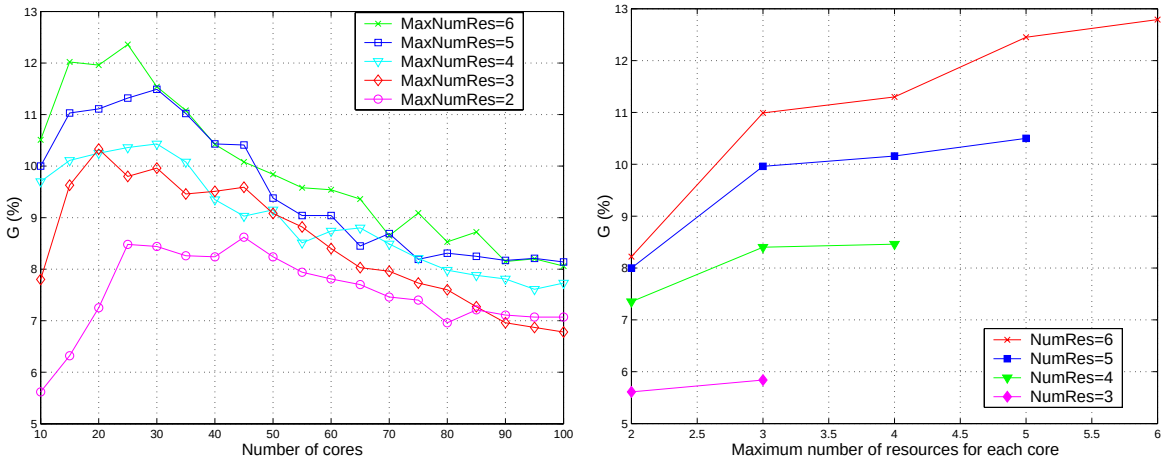
$$G = \frac{L_{wog} - L}{L_{wog}}$$

where L_{wog} is the total test time of a schedule without grouping while L can be either L_{wg} or L_{ap}, i.e., the total test time of a schedule with grouping or all-permutation scheduling.

In simulation scenario 1, we assume that there are 5 resources in the system and each core may be provided with 1 to 3 test sets using corresponding resources to meet the fault coverage requirement. Table 3.4 shows the comparison results of the performance of WG over WOG, as well as AP over WOG. The number of cores (*NumCore*) in the SoCs changes from 10 to 45. TL in WOG/WG/AP represents the total test time by using WOG/WG/AP approach and the balance ratio of WG to WOG is represented by G_{og} in percentage, while the balance ratio of AP to WOG in G_{oa}. ET in WOG/WG/AP means the corresponding CPU execution time in these approaches, represented in milli seconds. As we can see, WG and AP perform better than WOG since both G_{og} and G_{oa} are greater than zero. When Comparing WG with AP, WG achieves better performance than AP in terms of the balance ratio and the CPU execution time. G_{og} reaches as high as 9.48% when *NumCore* is 40, while G_{oa} is 4.97%. When *NumCore* is 45, AP needs $158427ms$ CPU time to execute the algorithm while WG only needs $5ms$.

# of cores	TL in WOG	TL in WG	TL in AP	G_{og} (%)	G_{oa} (%)	ET (ms) in WOG	ET (ms) in WG	ET (ms) in AP
10	136592.84	126900.89	131228.16	7.10	3.93	0	0	117
15	186448.02	170924.75	179253.49	8.33	3.86	1	1	837
20	224807.08	205515.81	216476.31	8.58	3.71	1	1	3394
25	270352.56	246014.63	258553.52	9.00	4.36	2	2	10005
30	318043.99	290777.63	307681.87	8.57	3.26	3	3	24359
35	365636.25	332672.08	350888.77	9.02	4.03	3	3	51968
40	413141.74	373992.82	392622.08	9.48	4.97	4	4	94022
45	455018.06	417752.12	434622.70	8.19	4.48	5	5	158427

Table 3.4: The comparison between WOG, WG and AP approaches.



(a) G_{og} changing with *NumCore*.

(b) G_{og} changing with *MaxNumRes*.

Figure 3.10: G_{og} changing with the resource distribution.

In scenario 2, we study the effect of the number of cores on the test time and the maximum number of resources (*MaxNumRes*) provided for each core on the test time. We first assume that the total number of resources in the system is 6. Figure 3.10(a) shows the G_{og} values with number of cores ranging from 10 to 100 and maximum number of resources ranging from 2 to 6. As we can see, with the same maximum number of resources, G_{og} increases when the number of cores

increases. After it reaches a peak, it drops slowly when the number of cores increases further. For example, when *MaxNumRes* is set at 5, G_{og} increase from 10% when *NumCore* is 10. It reaches a peak of 11.49% when *NumCore* is 30. Then it drops slowly, G_{og} decreases to 8.14% with *NumCore* 100. This is reasonable because, when there are small number of cores, the total number of tests is also small and we could not balance the resource queues more evenly due to less flexibility. As the number of cores increases, the flexibility increases, and accordingly, G_{og} increases. On the other hand, when there are a large number of tests, the benefit of grouping will be dominated by the randomness, which in turn results in the dropping of the curve.

Moreover, we choose the number of cores to be 25 (for example), and change the total number of resources in the system from 3 to 6. Figure 3.10(b) shows G_{og} with various maximum number of resources for each core. As we can see, with the same total number of resources, G_{og} increases with the maximum number of resources for each core, while with the same maximum number of resources for each core, G_{og} increases when the total number of resources increases. This is again due to the change in flexibility of choosing test resources as discussed above.

Based on our simulation, we have the following result.

Proposition 3 *Grouping always helps balance the resource usage queue lengths fast and efficient.*

3.6 Fault-Model Oriented Multiple Test Sets Scheduling

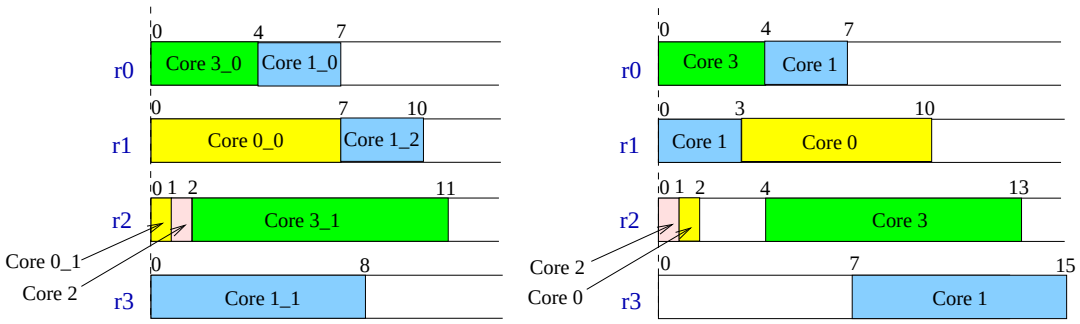
In the previous section, we assumed that one core needs only one test set. However, it is possible that a core may need multiple (say L) test sets to achieve a certain fault coverage. For example, in an embedded core-based SoC, several test methods are used to test the embedded memory. As we know, in addition to stuck-at, bridge, and open faults, memory faults include bit-pattern, transition, and cell-coupling faults. Parametric, timing faults, and sometimes, transistor stuck-on/off faults, address decoder faults, and sense-amp faults are also considered. [61] lists various test methods for embedded memory, i.e., direct access, local boundary scan or wrapper, BIST, ASIC functional test, through on-chip microprocessor, etc. Different test methods may require different test resources, use different test times, and provide different fault coverage. In this case, we can simply make

L virtual cores and convert the 1-L mapping to a 1-1 mapping. The only difference between this and the single test selection we discussed earlier is that, when choosing the shortest queue, one has to check if the selected test set conflicts with others which are for the same core and overlap the running time. Table 3.5 shows an example system, in which the tests are to be performed using the corresponding resources, for instance, test t_{10} to be applied using resource r_0, test t_{11} using resource r_2, etc. For each fault model, we need to select one test method by applying certain test from the candidates. Figure 3.11 illustrates the multiple test sets scheduling for the system, which can be performed in two steps.

Core ID	Fault Model	Candidate Test set	Resource Usage
c_0	f_{00}	$t_{00} = 12$	r_0
		$t_{01} = 7$	r_1
		$t_{02} = 6$	r_3
	f_{01}	$t_{03} = 4$	r_1
		$t_{04} = 1$	r_2
c_1	f_{10}	$t_{10} = 3$	r_0
		$t_{11} = 8$	r_2
		$t_{12} = 12$	r_3
	f_{11}	$t_{13} = 13$	r_2
		$t_{14} = 8$	r_3
	f_{12}	$t_{15} = 5$	r_0
		$t_{16} = 3$	r_1
		$t_{17} = 6$	r_2
		$t_{18} = 11$	r_3
c_2	f_{20}	$t_{20} = 5$	r_0
		$t_{21} = 1$	r_2
c_3	f_{30}	$t_{30} = 4$	r_0
		$t_{31} = 6$	r_1
	f_{31}	$t_{32} = 18$	r_1
		$t_{33} = 11$	r_3
		$t_{34} = 9$	r_2

Table 3.5: A fault model based system.

First, we create L virtual cores for each core corresponding to L fault models. For example, in Table 3.5, two virtual cores, $Core\ 0\_0$ and $Core\ 0\_1$ are generated for Core c_0 according to the two fault models, f_{00} and f_{01}, respectively. For each fault model, a group of test sets with various test times are provided for the required fault coverage. This means, each virtual core has a group of test sets available and we select one of them to perform testing. For instance, in order to cover fault f_{00},



(a) Test set selection for each fault model of cores

(b) Rescheduling to avoid conflicts

Figure 3.11: Multiple test sets scheduling.

we need to select one test set from the alternate test sets, t_{00}, t_{01} and t_{02}. Thus we map the multiple tests selection model to the single test selection case. We select the tests in a way that we balance the queues in order to avoid the situation where all the test sets will only use some of the resources and thus result in long length in these queues. In the second step, we need to reschedule the tests for the same core which overlap the running time. The shortest-task-first procedure is adopted here for rescheduling [62]. The worst case complexity is $O(r^3)$, where r is the number of virtual cores.

3.7 Summary

Optimal test scheduling for embedded core-based problem is a NP-hard problem. Our major technical contributions are as follows:

1. We have formulated test scheduling problem for SoCs as the single-pair shortest path problem by representing vertices as test sets, directed edges between vertices as a segment of a schedule sequence, and the edge weight as the test time of the test set at the end of the segment. Thereby, the problem of minimizing overall test time of a schedule has become equivalent to the problem of finding a shortest path. We have presented efficient test scheduling heuristic algorithms for its solution.

2. We have handled constrained scheduling by parallel resource usage queues. Resource conflict is the most commonly addressed constraint during scheduling, which arises due to the same DfT hardware shared among several cores. In addition, certain fault coverage should

be achieved when testing an SoC. One method or a combination of several methods may be needed to test a core in order to attain the required fault coverage. In this work, we have defined m queues in parallel corresponding to m resources, thus the test sets competing for the same resource would sequentially enter the resource usage queue.

3. Our simulation results have shown that there is no explicit dead time in our approach and we can further reduce the implicit dead time by grouping the cores and assigning higher priorities to those with smaller number of alternate test sets.

4. In our future work, we will discuss the modelling of mixed-signal SoCs for testability analysis, scheduling and diagnosis, and present efficient test scheduling algorithms to minimize the test cost.

Chapter 4

Dynamic Test Partitioning Under Power Constraints

In this chapter, we address the power-constrained test scheduling problem. We consider a system where one test set or a combination of test sets may be provided for testing each core in order to provide required fault coverage. Given a set of test sets for the cores, a set of resources, the test access architecture and the maximum power allowance, we propose a novel test scheduling scheme to minimize the overall test time by efficiently overlapping blocks of compatible tests of unequal length. Moreover, the total power consumption must not exceed the maximum power allowance at any time.

4.1 Rationale

Given a set of test sets for the cores, a set of resources, the test access architecture and the maximum power allowance, we propose a novel test scheduling scheme to minimize the overall test time by efficiently overlapping blocks of compatible tests of unequal length. There are three possible schedules for unequal length tests, i.e., nonpartitioned testing, partitioned testing with run to completion, and partitioned testing corresponding to three different test environments [37]. In an embedded core based SoC, each core has its dedicated test scheme and local control unit to process tests independently. Although a test must run to completion once started, a long test can proceed

continuously even if a shorter test in the same block finishes and a new test is initiated if it is compatible with the currently running tests. This gives us the flexibility to schedule the tests irrespective of whether all tests in a session are completed or not. For instance, in Figure 4.1, we denote the rectangle as a test set, with its length as the test time, and its width as the test power. Suppose we have scheduled tests T_1 and T_2 which constructs a test session, and now another two tests, T_3 and T_4, which are compatible with T_1 (the test compatibility will be discussed later in Sec. 4.2.3), need to be scheduled. According to our approach, both T_3 and T_4 can be allocated in this session without interrupting T_1 (see Figure 4.1(c)), while the basic clique partitioning approach [38] and the tree growing technique [41] only allow a new test to start after all tests in the previous session are completed (see Figure 4.1(a) and (b)). The basic idea of this approach is to generate a group of power-constrained concurrent test sets (PCTS's) and schedule the tests based on the compatibility relation among them [63].

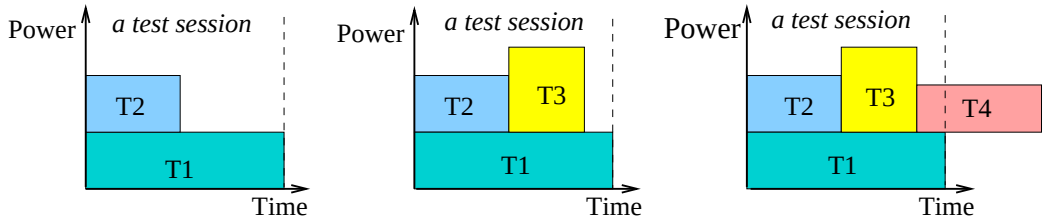


Figure 4.1: The comparison of our approach with the existing approaches.

4.2 Problem Formulation

In this section, we consider the embedded-core based SoCs, where each core may have multiple test sets using different resources. To satisfy the fault coverage requirement with minimum test application time under certain power constraints, the testing of all cores at the system level should be performed in parallel to the greatest possible extent.

4.2.1 System Definition

Without loss of generality, we assume that an SoC includes n cores, and there are m resources available for testing. A core may need one or several tests to meet the required fault coverage. Each test needs one or several resources, which can be used by one core at a time. In addition, a test is associated with an execution time and a test power while using certain test resources. A collision occurs when the tests sharing the same resource or the tests for the same core are scheduled to be applied concurrently. Moreover, the total power consumption must not exceed the maximum power allowance at any time. Given the test time, the test power and the required fault coverage, the goal of the scheduling technique is to efficiently utilize the test resources, and accordingly minimize the total test application time, while satisfying the power constraints.

More formally, we define the SoC as *TM*=$\{$*C, RSC,* P_{max}*, T, FC, S*$\}$, where, *C*=$\{c_1, c_2, ..., c_p\}$ is a finite set of cores, $RSC = \{r_1, r_2, ...r_m\}$ is a finite set of resources, P_{max} is the maximal allowed power consumption at any time, FC is the fault coverage required to test each core, and $T = \{T_1, T_2, ...T_n\}$ is a finite set of tests. A test is defined as $T_i = (t_i, p_i)$, where t_i is the test time and p_i is the power dissipated during the application of T_i. For illustration, we denote the rectangle as a test set, with its length as the test time, and its width as the test power. To simplify our discussion, we set the value of p_i to be the maximum power dissipation over all test vectors in T_i [40]. We define S_i as a set of power-constrained concurrent test sets (PCTS), with $L(S_i)$ as its length. $S = \{S_1, S_2, ...S_q\}$ is a finite set of PCTS's. The power dissipation $P(S_i)$ for a PCTS S_i is given as $P(S_i) = \sum_{j=0}^{M} p_j$ (p_j is the power of a test T_j in S_i, and M is the number of test sets in S_i), which should satisfy the power constraints, i.e.,

$$P(S_i) \leq P_{max} \tag{4.1}$$

Based on the system defined above, the power-constrained test scheduling problem or the **PTS Problem** can be formulated as: Given the SoC model TM, explore the solution space of PCTS (S_i), such that the total test application time is minimized ($\sum_{i=0}^{N} L(S_i)$, where, N is the total number of PCTS's), while each test T_j in set T will be executed exactly once.

4.2.2 Test Power Analysis

Generally speaking, the power dissipated during test mode of a system is substantially higher than the power dissipated during normal functional operation, due to the high switching activity [17]. The power dissipation in CMOS circuit can be divided into static, short circuit, leakage and dynamic power consumption, and 80% of the total power dissipation is attributed to dynamic power dissipation caused by switching of the gate outputs [64]. The power dissipation due to charging and discharging load capacitance can be characterized by:

$$P_d = C_{load} \times V_{dd}^2 \times f \times N_{switch} \tag{4.2}$$

where C_{load} is the load capacitance, V_{dd} is the supply voltage, f is the global clock frequency and N_{switch} is the switching activities (defined as the total number of gate output transitions). From Equation 4.2, the dynamic power dissipation highly depends on the switching activity, in other words, it is input-pattern dependent. In test mode, it depends on the test vector sets and testability techniques, such as scan. The value of p_i is instantaneously changing over time during testing, because the power dissipation is dominated by the dynamic power dissipation caused by switching of the gate outputs, and accordingly depends on the test vector sets in the test mode.

There exist mainly two models to estimate the power in VLSI circuits, the average power and the worst case instantaneous power [65]. For the PTS problem, [40] presented a simplification scheme, which assigns a fixed value for the power dissipation p_i, of a test T_i, that consists of a sequence of test vectors applied over time. Furthermore, p_i is defined as the maximum power dissipation over all test vectors in T_i (specified in Figure 4.2), and the power consumption is no more than p_i when the test T_i is executed. To simplify our discussion, we also use this estimation model. During testing, the test vectors are known for each core. We can use the test vectors generated by the ATPG as the inputs to a power simulation tool and then build up a table of p_i's for the tests T_i's of each core.

As shown in Figure 4.2, the power dissipation $P(S_i)$ for a concurrent test set S_i is given as:

$$P(S_i) = \sum_{j=0}^{M} P_j \tag{4.3}$$

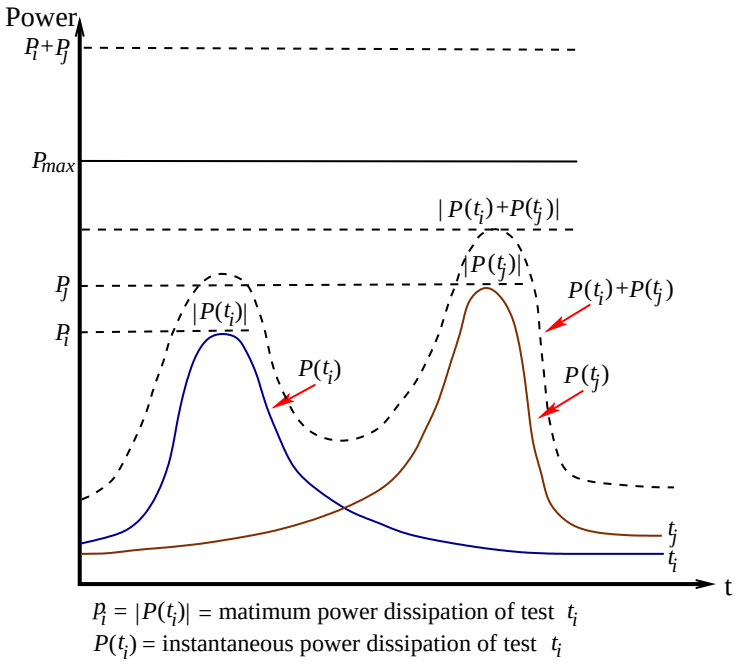
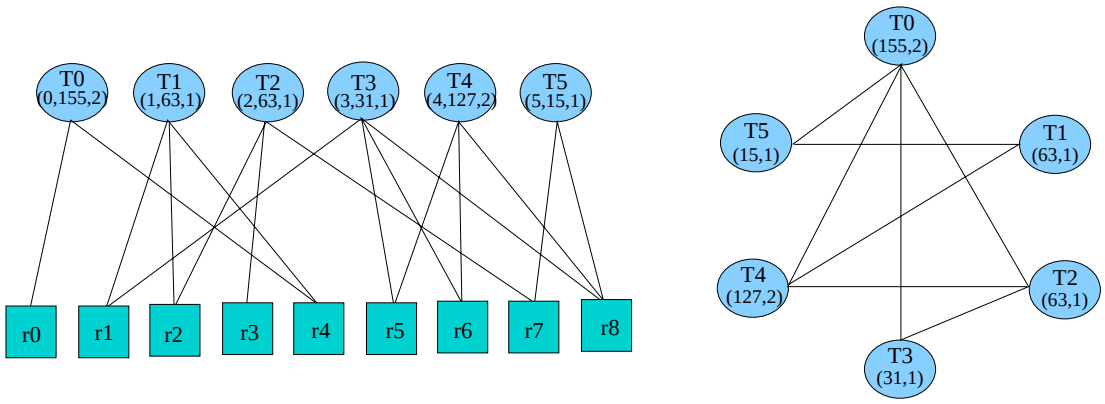


Figure 4.2: The power estimation model.

4.2.3 Test Compatibility

As proposed in [38], the resource conflict in an SoC is represented by a bipartite graph, called *resource conflict graph*. For illustration, we use an example as shown in Figure 4.3(a), which is similar to the one presented in [37–40]. The nodes on the top, labelled with its core number, test time and test power, denote the test sets for the cores, while the bottom nodes denote all test resources shared among the tests. For example, in a BIST enabled SoC, the circuit blocks which are required to perform a test (i.e., the test control logic, TPG, compressors/analyzers, and any intervening logic) are *test resources*. An edge between two nodes represents the usage of a resource by a test. A test may require one or several resources while a resource may be shared by several tests, which gives rise to resource conflicts.

Based on the resource conflict graph, a test compatible graph (TCG) can be set up by denoting the nodes as the test sets and connecting two nodes by an edge if there is no resource conflict between them. Thereby the nodes can be partitioned into a set of cliques, and within each clique the nodes are time compatible. Meanwhile, we need to consider power dissipation conflicts, since the nodes in the same clique may not be compatible from the power consumption point of view when



(a) Resource conflict graph

(b) Power−constrained test compatible graph

Figure 4.3: Obtain power-constrained TCG from resource conflict graph.

executing all the tests in the same clique at the same time might exceed the maximum power limit. Specifically, a *power-constrained TCG* (P-TCG) is established by deleting the edge connecting node T_i and T_j if the total power of the two nodes exceeds the maximum power dissipation limit. In the P-TCG, the pairs of nodes remaining connected are both time and power compatible and therefore can be executed concurrently. For example, we construct the P-TCG, as shown in Figure 4.3(b), from the resource conflict graph in Figure 4.3(a) assuming P_{max} to be 4. Our purpose is to obtain all PCTS's from the power-constrained TCG, thus we can handle the constrained scheduling problem effectively.

4.2.4 A Test Case

In order to illustrate the proposed test scheduling algorithm, and to demonstrate its effectiveness, we present a representative core-based SoC as a running example throughout this paper. It consists of 7 embedded cores from the ISCAS'85 combinational benchmarks [66] and ISCAS'89 sequential benchmark circuits [67]. Each core is provided with a test set, represented by a set of parameters, the number of inputs and outputs, the number of scan chains and their lengths, the number of test patterns, the number of capacitance nodes and peak switching frequency (PSF). Table 4.1 outlines the characteristics of these circuits. The test patterns for these cores are obtained from [68], while the related test power information is taken from [69, 70]. In addition, we assume that each core is provided with one test set, and the number of resources is 14 and P_{max} is $900mW$.

core	# of PIs	# of POs	# of scan chains	scan chain length max	scan chain length min	# of DFFs	# of test patterns	# of test cycles	# of cap nodes	PSF
s9234	36	39	4	54	52	211	105	5723	8221	1.229
s38417	28	106	32	55	51	1636	68	3656	33988	0.737
s5378	35	49	4	46	44	179	97	4507	4440	1.249
c5315	178	123	0	-	-	0	37	37	4509	1.714
s35932	35	320	32	54	54	1728	12	714	30317	1.410
c7552	207	108	0	-	-	0	73	73	6252	2.020
s15850	77	150	16	34	33	534	95	3359	14343	0.722

Table 4.1: Test data for an SoC embedded with cores from ISCAS benchmarks.

4.3 Basic Definitions

Definition 1 : A *maximum power-constrained concurrent test set (max-PCTS)* is a PCTS, in which no compatible tests can be added without exceeding the maximum power dissipation limit. For example, in Figure 4.4, nodes 1, 3, 6 form a clique and the total power dissipation within the clique satisfies the maximum power allowance. So $\{T_1, T_3, T_6\}$ is a max-PCTS.

Definition 2 : A *seed* is a candidate node, whose degree is the smallest among unscheduled tests (if several candidates have the same degree, the one with the lowest index is selected), where the *degree* is defined as the frequency of appearance of a node in unscheduled max-PCTS's. The basic idea of selecting the seed is to first schedule the tests which have the lowest compatibility. This intuitively helps reduce the probability that only one (or very few) test is performed at a given period, and accordingly reduce the overall test time.

Definition 3 : As to be discussed later, we will construct the *Dynamically Partitioned PCTS* (DP-PCTS) during scheduling. A DP-PCTS includes a number of power constrained concurrent test sets. Each DP-PCTS has a start time and a stop time.

Definition 4 : A DP-PCTS is *full* if it reaches a max-PCTS. A DP-PCTS, $D_j(T_i)$ (where j is its index in the current set of DP-PCTS's), is called the *DP-PCTS of the node* T_i if it can contain T_i. Note that, the reason why we specify $D_j(T_i)$ is to dynamically mark all the DP-PCTS's which can

contain the new node T_i to ease the scheduling later. A node may have long enough test time and has to be contained in several continuing DP-PCTS's, called a *DP-PCTS sequence*.

4.4 Power-constrained Concurrent Test Scheduling Algorithm

In our approach, the PTS problem is mapped into a graph theoretic problem by dynamically constructing a set of PCTS's, where a long test can be uninterrupted while initiating new tests. This relaxation will explore the restrictive space defined by concurrent block test scheduling [40, 41] so that further reduction of the total test time can be achieved. The proposed PCTS approach is in two steps. The first step searches for all maximum power-constrained concurrent test sets by constructing power-constrained TCG from the resource conflict graph. The second step performs the scheduling of the tests.

4.4.1 Generating max-PCTS

In Step 1, we generate all max-PCTS's, in which the tests meet all the constraints, i.e., resource sharing and power constraint. We first construct the power-constrained TCG (see Figure 4.4 for the test case in Sec. 4.2.4) as discussed earlier in Sec. 4.2.3. Then, we obtain all the cliques, and check whether the total power dissipation within each clique exceeds the maximum power allowance. If a clique with k nodes does not satisfy the power constraints, it will be sub-divided into k CTS sets, and each of them includes $k-1$ nodes. If any of them are a subset of a max-PCTS, it will be deleted to reduce redundancy. If any of the remaining CTS's are not a PCTS, they will be divided further into smaller sets by repeating the above operations until we get the max-PCTS's.

For the example in Figure 4.4, all maximum PCTS's are: $S_0 = \{T_0, T_1, T_6\}$, $S_1 = \{T_1, T_3, T_6\}$, $S_2 = \{T_2, T_3, T_5\}$, $S_3 = \{T_2, T_4\}$, $S_4 = \{T_3, T_5, T_6\}$, $S_5 = \{T_4, T_6\}$.

4.4.2 Dynamic Test Partitioning and Allocation

After obtaining the max-PCTS's, the next step is to schedule the test sets. It basically can be divided into three main sub-steps, namely, obtaining seed, adaptive allocation of tests and dynamically constructing DP-PCTS's. These steps are formally presented in the algorithm of Figure 4.5.

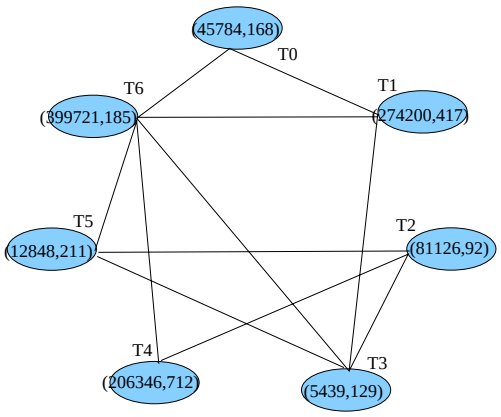


Figure 4.4: Obtaining power-constrained TCG.

Procedure PCTS( )

```
1  obtain the power−constrained test compatible graph;
2  derive the maximum power−constrained concurrent test sets;
3  determine degree for each node;
4  While (scheduled tests<total number of tests)
5       obtain seed;
6       allocate the seed;          /* According to one of three cases */
7       construct newly created DP−PCTS;
8       update the existing DP−PCTS's;
9       delete all full DP−PCTS's;
10      for every other test within the same max−PCTS with the seed
11            allocate the node;     /* According to one of three cases */
12            construct newly created DP−PCTS;
13            update the existing DP−PCTS's;
14            delete all full DP−PCTS's;
15      update the degree of all the tests;
```

Figure 4.5: The new power-constrained concurrent test scheduling algorithm.

Step 1: We first select the seed according to Definition 2 and allocate it, then allocate the test sets within the same max-PCTS of the seed and construct DP-PCTS's. After we schedule the first seed and other test sets within the same max-PCTS, we need to update the degree of the nodes and search for the next seed.

Step 2: A new node T_i can be allocated in either of the following three ways:

- **Case 1:** If its test length, t_i, is shorter than the length of one of its DP-PCTS, we may allocate it in this DP-PCTS by specifying its start time as the start time of the DP-PCTS and the stop time as the sum of its start time and its length (refer to Figure 4.6(a1)). If there are multiple

such DP-PCTS's, we choose the one with the shortest length. In other words, we allocate the test into the DP-PCTS whose length is closest to and larger than t_i. In case, its DP-PCTS happens to be the end of the current schedule, t_i may exceed the length of its DP-PCTS.

- **Case 2:** If its length is larger than any of its DP-PCTS's but shorter than the total length of its DP-PCTS sequence, we may allocate it in this DP-PCTS sequence by specifying its start time as the start time of the first DP-PCTS and the stop time as the sum of its start time and its length. Note that, if the last DP-PCTS is the end of the current schedule, the length of T_i may exceed the length of the DP-PCTS sequence (refer to Figure 4.6(b1)). Again, if there are multiple such DP-PCTS sequences, we choose the shortest one.

- **Case 3:** If we could not find either a DP-PCTS or a DP-PCTS sequence as discussed in the above two cases, then we may allocate it at the end of the schedule by specifying its start time as the stop time of the last DP-PCTS and the stop time as the sum of its start time and its length (refer to Figure 4.6(c1)).

Step 3: After T_i has been scheduled, new DP-PCTS's need to be constructed and some existing DP-PCTS's need to be updated according to the above three allocation cases:

- In case 1, the DP-PCTS of T_i is split into two DP-PCTS's, say D_j and D_{j+1}. The stop time of T_i is the stop time of D_j, which is the start time of D_{j+1} (See Figure 4.6(a2)). In addition, we need to check if they are full or not. If it is full, it will be deleted. In other words, no additional tests will be allocated into it.

- In case 2, if the stop time of T_i exceeds the existing schedule, a new DP-PCTS is set up at the end of the current schedule with its length equal to the stop time of T_i minus the stop time of the last DP-PCTS. Otherwise, similar to case 1, a DP-PCTS will be split into two. Meanwhile, the remaining DP-PCTS's in the sequence should be updated, and the full DP-PCTS's will be deleted (See Figure 4.6(b2)).

- Case 3 is quite simple since the node is added at the end of the schedule alone, a new DP-PCTS is created with the same start time and the stop time as those of T_i (See Figure 4.6(c2)).

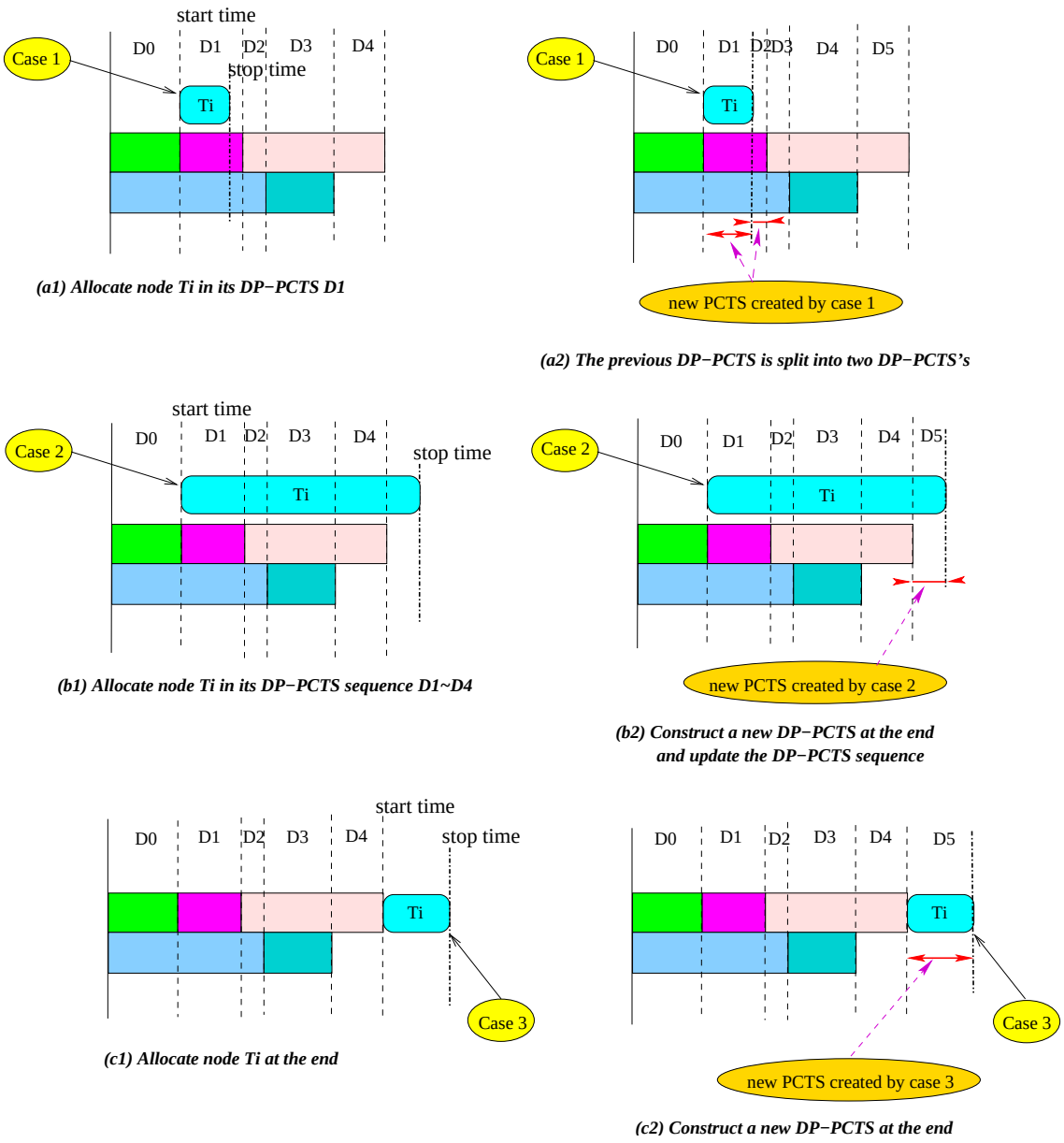


Figure 4.6: The three ways to allocate a new coming node.

The following is an illustration of the steps of the PCTS algorithm applied to the set of max-PCTS's generated in Sec. 4.4.1 to find the best possible test scheduling. We select T_0 as the first seed, and allocate it at the beginning (i.e., with a start time of zero). In the meantime, we construct the first DP-PCTS $D_0 = \{T_0\}$. The start time of D_0 is 0, and the stop time is t_0. Then we allocate T_1 and T_6 since they are within the same max-PCTS as T_0 and consequently construct three DP-PCTS's as shown in Figure 4.7(1). As the updated $D_0 = \{T_0, T_1, T_6\}$ is full, because it is equal to

S_0, we will delete D_0. Then we choose T_3 as the next seed. There are two DP-PCTS's available $D_1 = \{T_1, T_6\}$, $D_2 = \{T_6\}$. Since both of them can contain T_3, we choose the shortest one and allocate T_3 in $D_2(T_3)$ (To simplify, we will use the symbol D_j to mean $D_j(T_i)$ in the remainder of the example), and the previous D_2 is split into two DP-PCTS's, $D_2 = \{T_3, T_6\}$ and $D_3 = \{T_6\}$ as shown in Figure 4.7(2). Then we allocate other tests, T_5 and T_2, within the same max-PCTS of T_3. Since T_5 is compatible with the DP-PCTS sequence of D_2, D_3, it is allocated into it and split previous D_3 into $D_3 = \{T_5, T_6\}$ and $D_4 = \{T_6\}$. Since the updated $D_2 = \{T_3, T_5, T_6\}$ reaches full, it is deleted. When we schedule test T_2, as there is no DP-PCTS to contain it, it is allocated at the end of the schedule and generate a new DP-PCTS $D_5 = \{T_2\}$ with length of t_2 (see Figure 4.7(3)). The last test to be scheduled is T_4. Since D_4 and D_5 form its DP-PCTS sequence, it is contained into this sequence. As the length of T_4 exceeds the length of the sequence, a new DP-PCTS $D_6 = \{T_4\}$ is created at the end. The final schedule is shown in Figure 4.7(4).

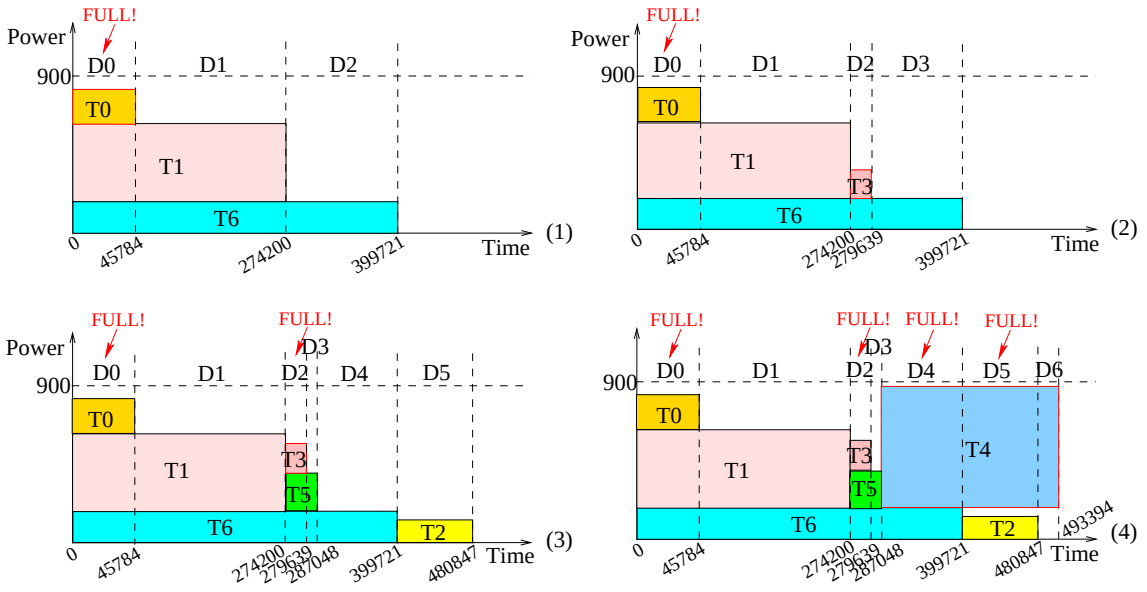


Figure 4.7: The scheduling steps of the example system.

As discussed above, a test is not necessarily to be partitioned in our approach unless it is contained in a sequence of DP-PCTS's. In this case, the test is partitioned after it is allocated. In other words, the test is scheduled as a whole and the partitioning is only used to provide the test compatibility information for the remaining unscheduled tests. This partitioning property is different from

the test partitioning with run to completion proposed in [37], in which each test is divided into a set of equal length sub-tests, and scheduled via a simplified approach for the equal length test problem. Hereafter, we refer to our approach as the **dynamic test partitioning**, which can adapt to various system configurations and resource allocation and sharing, and make the best fit for individual tests in a resource and power constrained schedule.

4.5 Discussion and Results

We evaluate the proposed scheduling algorithm via simulation. In our simulation model, we use randomly generated test sets and resources. We assume that each core may be provided with one or several test sets, and one test set may use at least one test resource to meet the fault coverage requirement. The performance of our approach is evaluated by comparing with other scheduling techniques.

4.5.1 Discussion of the Comparable Approaches

In order to evaluate the performance of the new algorithm, we compared it with two existing approaches which also address the PTS problem, the block-test scheduling [40] (BT, for short) and the greedy best-fit algorithm [33] (GD, for short). The first method is based on the concept of block-test, which assumes that the tests cannot be partitioned so that the scheduling problem is deduced to find a minimum cover of a set of power compatible sets. In addition, as we discussed earlier, the computation is quite excessive due to the enormous covering tables generated. The second method implements a fast heuristic algorithm, whose worst case complexity is $O(T^2)$, where T is the number of test sets. Note that, since our purpose is to evaluate the scheduling performance according to the test application time, we will not consider improving parallel degree by integrating the scan chain division technique as proposed in [33]. Although this approach allows test overlapping between each other, it greedily searches through all the scheduled tests when allocating an individual test, and the test compatibility is checked only by then. Obviously, it does not utilize the compatibility information effectively. While in our approach, we allow a long test to be contained in a group of continuing DP-PCTS's and the test sets are dynamically partitioned to provide effective

information for efficient allocation of tests in idle time. This relaxation explores a way to reduce the explicit dead time introduced by the concurrent block test scheduling approaches. Meanwhile, it efficiently utilizes the test compatibility relation between tests by constructing and updating a set of DP-PCTS's, so that further reduction of the total test time can be achieved.

The worst case complexity of the proposed PCTS algorithm (Figure 4.5) is $O(T^2)$, where T is the number of test sets. From implementation point of view, since the identification of all the cliques in a graph is a NP-complete problem [71], it dominates the time complexity of our approach. Therefore, heuristic algorithms must be employed to obtain practical and near-optimal solutions [72] for the clique identification step. We believe that if a suitable practical clique finding algorithm is well implemented, our approach can provide a fast and efficient test scheduling.

We show the comparative results of the proposed PCTS algorithm with BT and GD. Dealing with tests of unequal length for the same example system in Figure 4.4, we can see from Figure 4.7(4) and Figure 4.8 that our approach has a shorter overall test time (493394) than that of the others (618915 for BT and 561672 for GD, respectively). As we know, there are three possible schedules for unequal length tests, i.e., nonpartitioned testing, partitioned testing with run to completion, and partitioned testing according to different test environments [37]. Generally speaking, BT belongs to nonpartitioned testing, while GD and PCTS are grouped to partitioned testing with run to completion. In an embedded core based SoC, each core has its dedicated test scheme and local control unit to process tests independently. Therefore, partitioned test schedules with run to completion are usually applied to provide the flexibility to schedule the tests irrespective of whether all tests in a session are completed or not, thus saving the overall test time significantly.

4.5.2 Experiment Results

In order to further evaluate the proposed PCTS approach, we implement GD and PCTS, and run the experiments on 5 hypothetical but nontrivial SoCs consisting of ISCAS'85 benchmarks and ISCAS'89 benchmark circuits. In our experiments, we assumed a single top level TAM of width 32 bits. For each SoC we generated a collection of graphs and the overall test time was averaged over such a collection. Moreover, we assumed that each core may use up to 3 test resources to meet

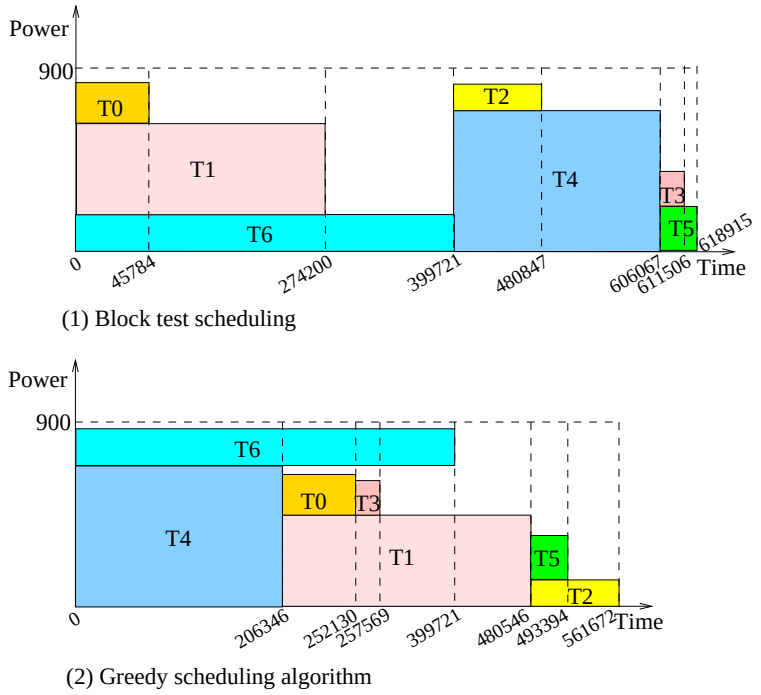


Figure 4.8: The comparison with two existing approaches.

the fault coverage requirement, and P_{max} is 900mW. The test data related to each core in SoCs are presented in Table 4.2 and the comparison of the SoCs is listed in Table 4.3.

As we can see, the PCTS approach achieves better performance than GD from the test time point of view. The test time is reduced at least by 8.6% (e.g., when the number of cores is 12, and the total number of resources shared among cores is 24), and it reaches 24.1% when the number of cores is 30 (with 50 test resources).

4.6 Summary

In this chapter, we have presented a novel adaptive scheduling algorithm for testing embedded core-based SoCs under power constraints. We have mapped the PTS problem into a graph theoretic problem by constructing the P-TCG graph and efficiently utilizing the test compatibility between the tests. We have performed the scheduling in a way that dynamically partitions and allocates the tests such that a long test can be uninterrupted when initiating new tests. We have consequently constructed and updated a set of dynamically partitioned power constrained concurrent test sets to

core	# of PIs	# of POs	# of scan chains	# of DFFs	# of test patterns	# of test cycles	SoC 1	SoC 2	SoC 3	SoC 4	SoC 5
c880	60	26	0	0	16	16	1	1	0	0	0
c1355	41	32	0	0	84	84	1	0	0	0	0
c1908	33	25	0	0	106	106	0	1	2	2	3
c2670	233	140	0	0	44	44	0	1	2	2	0
c3540	50	22	0	0	84	84	1	0	2	2	3
c5315	178	123	0	0	37	37	1	0	2	2	0
c6288	32	32	0	0	12	12	1	0	0	0	0
c7552	207	108	0	0	73	73	1	1	2	3	0
s838	35	2	1	32	75	2507	0	1	0	2	3
s1423	17	5	2	74	20	797	0	1	0	0	3
s5378	35	49	4	179	97	4507	0	1	0	2	3
s9234	36	39	4	211	105	5723	1	1	2	2	3
s13207	62	152	16	638	233	9593	0	1	2	2	3
s15850	77	150	16	534	95	3359	0	1	1	2	3
s35932	35	320	32	1728	12	714	1	1	1	2	3
s38417	28	106	32	1636	68	3656	1	1	1	2	3

Table 4.2: Test data for cores in SoC 1 to 5.

SoC	# of cores	# of test resources	Test time in PCTS	Test time in GD	Time saved (%)
1	10	20	480558	526342	8.7
2	12	24	1481378	1620940	8.6
3	20	34	1687083	1985890	15.1
4	25	40	1841535	2601573	29.2
5	30	50	2234102	2943915	24.1

Table 4.3: Comparison of PCTS approach with GD.

reduce the explicit dead time, and ultimately reduces the test application time. Simulation study have shown the productivity gained by the new approach. Further research on test scheduling is not only limited to the scheduling of the test sets provided by the core vendors, but also involves the activities to develop the DfT techniques, to design the test controller IPs, and to efficiently partition and distribute the test resources, which will in turn result in more efficient test scheduling.

Chapter 5

Constrained Scheduling with Wrapper/TAM Co-optimization

In this chapter, we present a novel scheduling algorithm for testing embedded core-based SoCs. Given test conflicts, power consumption limitation and top level TAM constraint, we handle the constrained scheduling in a unique way that adaptively assigns the cores in parallel to the TAMs with variable width and concurrently executes the test sets by dynamic test partitioning, thus reducing the test cost in terms of the overall test time.

5.1 Problem Statement

In this section, we consider the embedded-core based SoCs, where each core may have multiple test sets using different resources to provide required fault coverage. Given a set of test sets for the cores, a set of resources, the test access architecture, the TAM bandwidth limitation and the maximum power allowance, we describe the test scheduling problem based on wrapper scan chain configuration and rectangle test set transformation such that the testing of all cores at the system level is performed in parallel to the greatest possible extent. Therefore, the test scheduling is not only limited to the ordering of the test sets provided by the core vendors, but also involves the issues of core access path construction and TAM width distribution. We assume that multiple bus-based

TAMs, such as TestBus or TestRail[1] in the system operate independently, and the cores on the same TAM are tested sequentially. In addition, TAMs can fork and merge between cores to improve utilization of TAM wires.

For an SoC, the chip level test ports and core input/output terminals are given, thus the total TAM width is known. Without loss of generality, we assume that an SoC includes n cores sharing m test resources, and a total of W_{max} width TAMs are used for test data transportation. A core may need one or several tests to meet the required fault coverage, and each test needs one or several test resources involved in the test. On the other hand, a resource may be shared by several tests that gives rise to resource conflicts. A test set, which is associated with a test power, assigned TAM width and an execution time, is represented as a cube as shown in Figure 5.1. Note that, when several tests are executed concurrently, their total TAM width and their total power consumption must not exceed the maximum available TAM bandwidth and the maximum power allowance, respectively.

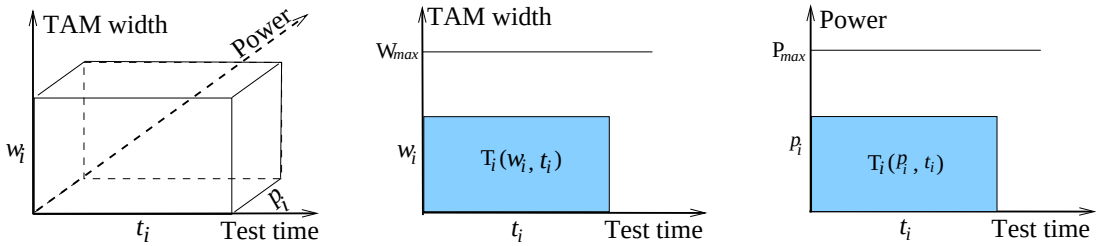


Figure 5.1: Representing a test set as a cube.

More formally, we define the SoC as TM=$\{C,\ RSC,\ P_{max},\ T,\ W_{max},\ S\}$, where, $C = \{c_1, c_2, ...c_n\}$ is a finite set of cores, $RSC = \{r_1, r_2, ...r_m\}$ is a finite set of resources, P_{max} is the maximal allowed power consumption at any time, and $T = \{T_1, T_2, ...T_n\}$ is a finite set of tests. A test is expressed as a three-tuple $T_i = (w_i, t_i(w_i), p_i)$, where w_i is the TAM width assigned to core c_i, $t_i(w_i)$ is the corresponding test time, p_i is the power dissipated during the application of T_i. To simplify the formulation, we use the peak power dissipated over all test vectors in T_i as the value of p_i [40]. In addition, we assume that the peak power p_i of a core c_i does not vary with

[1]TestRail outperforms TestBus in terms of testing the wiring and circuitry in between the cores [49]. As we consider core testing only in this chapter, either of them is used as the test access architecture.

the configuration of the wrapper scan chains. The test application time t_i is decided by the test data volume of the individual cores and the TAM width. W_{max} is the top level TAM width of the system, which is constrained due to pin count limitation.

We define S_i as a set of power-constrained concurrent test sets (PCTS), with $L(S_i)$ as its length. $S = \{S_1, S_2, ...S_q\}$ is a finite set of PCTS's. The power dissipation $P(S_i)$ for a PCTS S_i, which is given by $P(S_i) = \sum_{j=0}^{M} p_j$ (where p_j is the power of a test T_j in S_i and M is the total number of tests in S_i), should satisfy the power constraints, i.e.,

$$P(S_i) \leq P_{max} \tag{5.1}$$

In the meantime, the total TAM width of S_i should meet the TAM bandwidth limit, i.e.,

$$W(S_i) = \sum_{j=0}^{M} w_j \leq W_{max} \tag{5.2}$$

where w_j is the TAM width of T_j.

Thus the constrained test scheduling problem on TAMs (namely, *the CTST problem*) can be stated as follows: Given the SoC model TM, we schedule n test sets on a total of W_{max} width TAMs in a way that W_{max} is dynamically partitioned during test scheduling while meeting the top level TAM width constraint and maximum power limitation, and the total test time is minimized by concurrently executing constrained test sets.

5.2 Wrapper Configuration

Test width adaptation is usually performed by serially connecting core I/Os with internal scan chains through test wrappers such as IEEE P1500 when there is a mismatch between test data width of individual cores and the TAM bandwidth. Because of the stressed need for balanced wrapper scan chains [2], the test times of scan-testable cores vary with TAM width as a "staircase" function [44],

$$t_i(T_i, w_i) = (1 + \max\{L_i(w_i)\}) \times P_i + \min\{L_i(w_i)\} \tag{5.3}$$

where, $L_i(w_i)$ is the length of the wrapper scan chain when the TAM width is w_i, and P_i is the number of test patterns for testing T_i.

The scan chain partitioning problem has been mapped to well-known Multiprocessor Scheduling and Bin Design problem in [2, 44], and fast heuristic algorithms (such as FFD, BFD, etc.) have been proposed to solve it. Here, we use *Best-Fit Decreasing* approach to design wrapper scan chains with the varying of TAM bandwidth and calculate the candidate rectangle set $R_i(w_i, t_i(w_i))$. Thus for each test set, we actually supply a set of candidate rectangles R_i, as shown in Figure 5.2, when assigning different TAM width resulting in different test time. Furthermore, we define the maximum TAM width assigned to core c_i as the bitwidth of c_i, $\phi_i = \max\{w_i\}$. When $w_i = \phi_i$, we obtain the minimum test time for T_i, $t_i(\phi_i) = \min\{t_i(w_i)\}$. In other words, even if we increase TAM width further, the test time cannot be reduced any more.

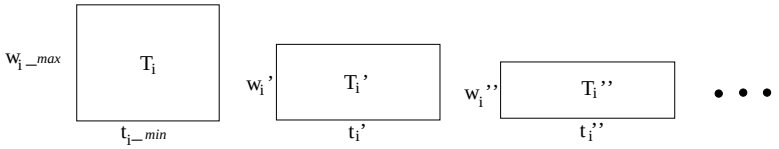


Figure 5.2: Candidate set of rectangles of test T_i.

5.3 The CTST Scheduling Algorithm

The proposed scheduling algorithm includes four major steps, namely, obtaining max-PCTS, deriving seed, adaptively assigning TAM width, and dynamic test partitioning (see Figure 5.3) [73]. The worst case time complexity of the CTST algorithm is $O(T^2)$, where T is the number of test sets.

5.3.1 Obtaining max-PCTS

As we discussed earlier, bin-packing approach is not very efficient since it checks the constraints separately when scheduling each individual core, thus the test compatibility information is not effectively utilized, resulting in non-optimal scheduling for the remaining tests. In our approach, we construct P-TCG and obtain power-constrained concurrent test sets, with resource conflicts and power constraints of all tests taken into consideration at the same time.

Theorem 1 *A maximal clique or a subset of maximal clique S ($\subseteq V$) where no nodes can be added without exceeding the maximum power limit in graph P-TCG $G = (V, E)$ is a max-PCTS.*

```
Procedure CTST( )
0  Initialization;
     i)  randomly generate test resource distribution among the cores;
     ii) obtain the power−constrained TCG and the conflict graph;
     iii) obtain candidate rectangle set for each test;
1  derive the max−PCTS's from P−TCG;
2  derive the conflict sets from the conflict graph;
3  determine the first seed set;
4  while (scheduled tests<total number of tests)
5      for each seed in the seed set
6          for each node in the max−PCTS with the seed
7              select a suitable rectangle test from the candidate set such that
8                  their total TAM ⩽ W_max, and
9                  their test times are close to each other;
10         for every test within the same max−PCTS
11             allocate the node;
               /* according to one of three cases as described in Sec.4.4 */
12             construct newly created DP−PCTS;
13             update the existing DP−PCTS's;
14             delete all full DP−PCTS's;
15     search for the next seed set;
```

Figure 5.3: The CTST test scheduling algorithm.

Proof : As discussed in Sec. 4.2.3, a P-TCG $G = (V, E)$ is constructed in a way that the vertices in V are connected by edges in E if they are both time compatible or power compatible. Thus, the tests in a clique (i.e., a complete subgraph within which any pair of nodes are adjacent [71, 74]) are free of resource conflicts. A maximal clique (S) is a clique including a set of vertices so that any set $H \supset S$ is not a clique. When the total power of the test sets in S exceeds the maximum power limit, S is further divided into a set of K (smaller) cliques (S_i, $1 \leq i \leq K$ and $\bigcup S_i = S$) so that the total power dissipation in S_i meets the power constraint and no nodes can be added without exceeding the maximum power allowance. Clearly, each such set is a max-PCTS. ∎

For an example SoC system, whose test data is shown in Table 4.1, we first construct the power-constrained TCG as discussed earlier in Sec. 4.2.3. Then, we use a branch and bound method proposed in [72] to obtain all maximal cliques from the P-TCG, and check if the total power dissipation within each maximal clique exceeds the maximum power allowance. If a maximal clique does not satisfy the power constraints, it will be divided further into smaller sets as described in Sec. 4.4.1 until the power constraint is satisfied, resulting in max-PCTS's. For example, for a P-TCG as shown in Figure 5.4, we obtain the max-PCTS's as follows: $S_0 = \{T_0, T_1\}$, $S_1 = \{T_0, T_2\}$,

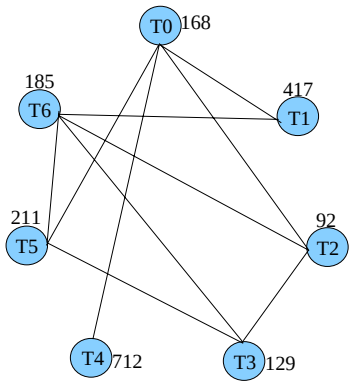


Figure 5.4: Power-constrained TCG.

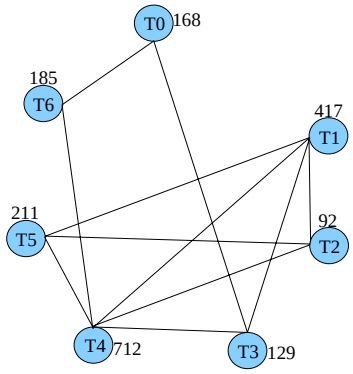


Figure 5.5: The corresponding conflict graph.

$S_2 = \{T_0, T_4\}$, $S_3 = \{T_0, T_5\}$, $S_4 = \{T_3, T_6, T_5\}$, $S_5 = \{T_3, T_6, T_2\}$ and $S_6 = \{T_6, T_1\}$.

As we can see, a test set has multiple choices to be allocated in different PCTS, which provides the flexibility to overlap PCTS's in a way that a long test can be contained in several continuing PCTS's (see Sec. 5.3.4).

5.3.2 Obtaining Seed Set

In order to facilitate an efficient schedule, we start with scheduling seeds, i.e., the candidate nodes. We first obtain the conflict sets that contain the maximal disconnected vertices in P-TCG, within which no tests can be overlapped.

Theorem 2 *A maximal clique F ($\subseteq V$) in the complementary graph of P-TCG $\widetilde{G} = (V, \widetilde{E})$ is a conflict set.*

Proof : In P-TCG, the adjacent vertices are concurrent test sets, while the disconnected nodes are the conflicting tests. Thus an empty subgraph of P-TCG contains the set F of vertices that cannot be executed in parallel, i.e., the set $F \subset V$ satisfies $F \bigcap E(F) = \Phi$ (1). In contrast to a set of conflicting nodes with size k where no two vertices are adjacent in G, the same set of vertices in complementary graph $\widetilde{G}$ are all connected to each other, i.e. a clique of size k. It is quite obvious, therefore, that a maximal clique F ($\subseteq V$) that satisfies (1) together with $H \bigcap E(H) \neq \Phi, \forall H \supset S$ is a conflict set. That means, a maximal clique of $\widetilde{G}$ corresponds to the conflict set of G and vice versa. ∎

For instance, the complementary graph of Figure 5.4 (also dubbed as *conflict graph*) is shown in Figure 5.5. We obtain the following conflict sets: $F_0 = \{T_4, T_1, T_2, T_5\}$, $F_1 = \{T_4, T_1, T_3\}$, $F_2 = \{T_4, T_6\}$, $F_3 = \{T_0, T_3\}$ and $F_4 = \{T_0, T_6\}$.

Since the tests in a conflict set cannot be executed in parallel, it is likely that the largest conflict set (in terms of the total test time) will dominate the overall test time of SoC testing. So we choose the largest conflict set as the seed set and resolve this first. More specifically, after we obtain $t_i(\phi_i)$ for each core, we calculate the minimum total test time of each conflict set F_j,

$$|F_j| = \sum_{i=0}^{B-1} t_i(\phi_i) \tag{5.4}$$

where B is the number of conflicting test sets in F_j. The conflict set with maximum total test time is chosen to be the seed set ε_i such that

$$|\varepsilon_i| = \max\{|F_j|\} \tag{5.5}$$

and $|\varepsilon_i|$ is denoted as the *size of seed set* ε_i. For example, we choose $\varepsilon_0 = \{T_4, T_1, T_2, T_5\}$ as the first seed set for the tests shown in Figure 5.5 ($|F_0|$ = $t_4(\phi_4 = 32)$ + $t_1(\phi_1 = 32)$ + $t_2(\phi_2 = 5)$ + $t_5(\phi_5 = 30)$ = 792+3876+4656+584 = 9908).

Thus, in step 2 (lines 3&15 in Figure 5.3), we choose the seed set as described above and start the TAM assignment and test scheduling with the max-PCTS of the seed. If more than one max-PCTS contains the seed, the one that has the largest number of nodes is selected. If they have the same size, one of them is randomly picked. After we allocate the seeds and other tests within the same max-PCTS of the seeds, we need to update the status of scheduled nodes and exclude them from the sets to search for the next seed set.

5.3.3 Adaptive TAM Assignment

In order to obtain a suitable TAM assignment among the tests, we use a adaptive assignment scheme here so that we can avoid the drawbacks induced by fixed TAM partitioning or preferred TAM width initialization. In this scheme, we distribute the top level TAM width W_{max} among the nodes in a max-PCTS in a way that adaptively adjusts the TAM width for each node from its bitwidth so that their test lengths are close to each other as much as possible. More specifically, we assign bitwidth

to each node in the same max-PCTS at first. For example, when we schedule the test sets within a max-PCTS $S_i = \{T_u, T_v, T_x, T_y\}$, we first assign the rectangles with the bitwidths (with their test times $t_x > t_y > t_v > t_u$) to all the tests. If their total TAM width $total\_TAM = \phi_u + \phi_v + \phi_x + \phi_y$ exceeds W_{max}, we reduce the TAM width of node T_u, whose test time is the minimum among them, to the next candidate set $T'_u(w'_u, t'_u)$. Then we check the total TAM width $total\_TAM$ again and repeat the above operations until meeting the top level TAM width requirement ($total\_TAM \leq W$). In this way, we always assign the most suitable TAM width to the node and its test time is most possibly minimized.

Based on the adaptive assigning scheme, we dynamically partition the top level TAM width among the max-PCTS of the seed in step 3 (lines 6-9 in Figure 5.3) before allocating these nodes. Therefore, the nodes within the max-PCTS meet all the constraints and suitable rectangle tests are chosen from their candidate sets.

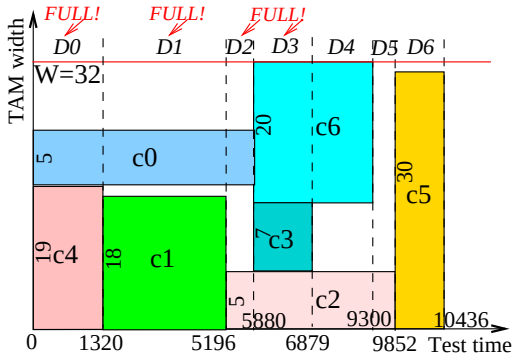
5.3.4 Dynamic Test Partitioning

After assigning suitable TAM width to each core, the last step (lines 10-14 in Figure 5.3) is to schedule the tests. We first allocate the seed from the seed set with the lowest compatibility, and then allocate the test sets within the same max-PCTS of the seed and construct DP-PCTS's. After we schedule the first seed set and other test sets within the same max-PCTS's of the seeds, we repeat these operations for the next seed set until all test sets are scheduled. Mainly, the scheduling is performed in two subsequent steps, adaptive allocation of tests and dynamically constructing DP-PCTS's as described in Sec. 4.4.2.

For example, we schedule the tests for the SoC shown in Table 4.1 with a top level TAM width of 32 bits. We select $F_0 = \{T_4, T_1, T_2, T_5\}$ to be the seed set and T_4 to be the first scheduled test which has the lowest compatibility. We assign the bitwidth $\phi_0 = 5$ to c_0 and the 2^{nd} maximum width $w_4 = 19$ to c_4,[2] and allocate them at the beginning. We next construct two DP-PCTS's $D_0 = \{T_0, T_4\}$ and $D_1 = \{T_0\}$. As D_0 equals to S_2, it is full. That means, it will not be considered

[2] After we assign 5 bit TAM width to c_0, the available TAM width for c_4 in parallel with c_0 is 27 bits. If we assign the bitwidth ϕ_4=32 to c_4, their total TAM width exceeds the TAM limit. According to Pareto optimization [44], we choose the next pareto optimal point for c_4.

to allocate a new node. Similarly, we choose the 2^{nd} maximum width $w_1 = 18$ to c_1 and schedule it in parallel with c_0. Previous D_1 is split into two DP-PCTS's $D_1 = \{T_0, T_1\}$ and $D_2 = \{T_0\}$. As to the third seed T_2, we adaptively partition the total TAM width among nodes 3, 6, 2 in the max-PCTS of S_5, resulting in the assignment of bitwidth $\phi_6 = 20$, $\phi_2 = 5$ to c_6 and c_2, respectively. Since T_2 is compatible with T_0, it is contained in D_2. A new DP-PCTS $D_3 = \{T_2\}$ is generated, and previous D_2 is updated to $D_2 = \{T_0, T_2\}$. Then we allocate T_3 and T_6 which are in the same max-PCTS of T_2, and construct new DP-PCTS's of $D_3 = \{T_2, T_3, T_6\}$, $D_4 = \{T_2, T_6\}$ and $D_5 = \{T_2\}$. For the final test T_5, since we cannot find a DP-PCTS to contain it, it is allocated at the end and uses up the total TAM width as much as it can and results in a total test time of 10436 as shown in Figure 5.6.



core # c_i	*bitwidth* ϕ_i	*assigned TAM width* w_i
c0	5	5 ✳
c1	32	18 △
c2	5	5 ✳
c3	30	7
c4	32	19 △
c5	30	30 ✳
c6	20	20 ✳

✳ labels the cores which use up the bitwidth
△ labels the cores which use the 2nd max width

Figure 5.6: Schedule result of the example SoC.

5.3.5 Lower Bound

As we can see, a seed set is chosen from the conflict sets with maximum total test time, i.e., the longest sequential test time. After allocating the seeds in current seed set and other tests within the same max-PCTS of the seeds, we update the conflict sets and search for the next seed set. Therefore, a **lower bound** τ of the overall testing time is found to be the sum of the size of the seed sets,

$$\tau = \sum_{i=0}^{\epsilon-1} |\varepsilon_i| \tag{5.6}$$

where ϵ is the total number of seed sets. For this example, only one seed set $\varepsilon_0 = \{T_4, T_1, T_2, T_5\}$ is needed to finish the scheduling, thus the lower bound is $\tau = |\varepsilon_0| = 9908$.

Theorem 3 *For an SoC with n cores and a total of W_{max} TAM width, a lower bound on the overall testing time τ is given by*

$$\tau \geq \sum_{k=0}^{\epsilon-1} \max_{j} \{ \sum_{i=0}^{B-1} t_i(\phi_i) \} \tag{5.7}$$

Proof : The testing time for core c_i depends on the TAM width assigned to it. Clearly, the testing time for c_i is at least $t_i(\phi_i)$. For an SoC, the resource conflicts and power constraint among the cores are represented by P-TCG, $G = (V, E)$. A conflict set of vertices, i.e., a maximal subset $V' \subseteq V$ such that no two vertices in V' are joined by an edge in E, are the test sets which can only be tested in sequence. Since the overall system testing time is constrained by the longest test times of sequential tests, the largest conflicting sequence with the size of $\max_j \{\sum_{i=0}^{B-1} t_i(\phi_i)\}$ will dominate the overall testing time, i.e.,

$$\tau \geq \sum_{k=0}^{\epsilon-1} \max_{j} \{ \sum_{i=0}^{B-1} t_i(\phi_i) \}$$

This value is intuitively the sum of the test times of all seeds when assigned to the maximum TAM width. ∎

5.4 Simulation and Comparison

We evaluate the proposed scheduling algorithm via simulation running on *750MHz* SunBlade*1000* with *512MB* memory. In our simulation model, each core is provided with one test set, and may use up to 3 test resources to meet the fault coverage requirement, and P_{max} is *900mW*. Given the number of cores and the number of resources, we randomly generate resource distribution among the cores. Moreover, for each SoC we generate a collection of graphs and the overall test time is chosen to be the minimum over such a collection. We run the experiments on 4 hypothetical but nontrivial SoCs consisting of ISCAS'85 [66] and ISCAS'89 [67] benchmark circuits. The test data related to each core in SoCs are presented in Table 5.1. We also extend the experiments on two ITC'02 SoC test benchmarks [75], d695 and h953 (which is the only benchmark that contains the power consumption information), for some valuable results. The performance of our approach is evaluated by comparing with the lower bound and various bin packing approaches in [4, 50] (which take power consumption limitation into consideration).

core	# of PIs	# of POs	# of scan chains	scan chain length max	scan chain length min	# of test patterns	SoC 1 (d695)	SoC 2	SoC 3	SoC 4
c5315	178	123	-	-	-	37	-	1	-	2
c6288	32	32	-	-	-	12	1	1	-	2
c7552	207	108	-	-	-	73	1	1	-	2
s838	35	2	1	32	32	75	1	1	3	2
s1423	17	5	2	37	37	20	-	1	2	2
s5378	35	49	4	46	44	97	1	2	3	2
s9234	36	39	4	54	52	105	1	2	2	2
s13207	62	152	16	41	39	233	1	1	3	2
s15850	77	150	16	34	33	95	1	1	3	2
s35932	35	320	32	54	54	12	1	1	1	2
s38417	28	106	32	55	51	68	1	2	1	2
s38584	38	304	32	45	44	110	1	1	2	3

Table 5.1: Test data for cores in SoC 1 to 4.

Table 5.2 shows the schedule results of the CTST approach on the 4 SoCs when the top level TAM equals to 64 or 32. Both the lower bound and the overall test time are listed for comparison. More experiments are performed on d695 and h953 when W_{max} and (or) P_{max} vary (see tables 5.3, 5.4 and 5.5). We also compare the results of d695 with the rectangle packing approach proposed in [50], and show the percentage of reduced test application time. Meanwhile, we compare the total test application time of d695 with the 3-D bin packing approach proposed in [4] for a given number of TAM width limit and the peak power constraint. ΔT presents the percentage change in testing time over [4]. From the simulation data, we have the following observation:

(1) Our approach performs well, the results are reasonably close to the lower bound. For d695 from Table 5.3, the test time is 23% more above the lower bound when W_{max}=16, this rate is reduced to 16% when W_{max} is increased to 64. In addition, the lower bound from [49] is also listed in Table 5.3 for comparison. As we have noticed, when $W_{max} \leq 24$, our lower bound is lower than what reported in [49]. That is because we use the minimum test time when optimally assigning

the bitwidth to each seed during the calculation of lower bound on the seed set. Most cores cannot reach this optimal value when the top level TAM width is low ($W_{max} \leq 24$), thus resulting in a loose lower bound in this case.

SoC	# of cores	# of test resources	W_{max}=64		W_{max}=32	
			Lower bound	Test time	Lower bound	Test time
1	10	18	10836	12954	22602	25612
2	15	24	18286	18880	29741	31736
3	20	34	25590	26578	49987	50947
4	25	40	30184	30193	55230	57535

Table 5.2: CTST test scheduling results for SoC 1 to 4.

Top level TAM W_{max}	Lower bound in CTST	Lower bound in [49]	Test time in CTST P_{max}+RSC	Test time in [50] preemptive+P_{max}	Time saved ΔT (%)
16	33454	40951	43619	47574	8.31
24	24982	27305	29699	-	-
32	22602	20482	25612	29039	11.80
40	17319	16388	18909	-	-
48	16159	13659	18186	28441	36.06
56	14192	11709	14402	-	-
64	10836	10247	12954	20004	35.24

Table 5.3: Comparison of CTST algorithm with rectangle packing approach (d695).

(2) For a given SoC, the overall test time is reduced further when increasing the top level TAM bandwidth. That is because the bitwidth of the cores increases accordingly and results in shorter test time. For any SoC in Table 5.2, both the lower bound and the test time when W_{max}=64 are lower than those when W_{max}=32.

(3) Our approach achieves a better performance than the rectangle packing approach by adaptively assigning the TAM width and concurrently executing tests by dynamic test partitioning. As we can see from Table 5.3, for a fixed TAM bandwidth, we can reduce the test time further. The test application time reduced by applying our approach is at least by 8.31% and it reaches as high

as 36.06%. On the other hand, for a fixed test time, we use less TAM bandwidth. For example, for a test time of 18909, we use TAM width of 40 for SoC 1 while the approach in [50] needs 64 TAM widths to reach 20004.

(4) When increasing either the top level TAM width or the total power allowance, the overall test time can be further reduced. This reduction in testing time has been confirmed in [4]. However, except the power and pin count constraints, [4] does not take the resource conflicts into consideration. For example, several cores may share the same test generator or response evaluator, and thus cannot be tested in parallel. Our approach provides the solution to more comprehensive constrained scheduling problem which considers various constraints including not only the power and TAM constraints but also the resource conflicts between the cores. The schedule results for d695 are shown in Appendix 8 with the related power and resource conflicting information in P-TCG when W_{max}=32 and P_{max} changes from 1500 to 2500. When comparing with [4], further reduction is obtained in the overall testing time as shown in Table 5.4, which implies that the CTST algorithm achieves better performance over [4].

P_{max}		W_{max}=32	W_{max}=48	W_{max}=64	W_{max}=80	W_{max}=96	W_{max}=112	W_{max}=128
1500	CTST	26495	20432	15510	15510	15510	15510	15510
	3-D	45560	31028	27573	20914	20914	16841	16841
	ΔT (%)	41.84	34.15	43.75	25.84	25.84	7.90	7.90
1800	CTST	26158	18824	15270	15184	13242	13242	13169
	3-D	44341	29919	24454	20467	18077	14974	14899
	ΔT (%)	41.01	37.08	37.56	25.81	26.75	11.57	11.61
2000	CTST	26027	17996	14267	13737	13737	13737	13737
	3-D	43221	29419	24171	19206	17825	14128	14128
	ΔT (%)	39.78	38.83	40.97	28.48	22.93	2.77	2.77
2500	CTST	25669	17779	12696	10572	10572	10572	10572
	3-D	43221	29023	23721	19206	15847	14128	12993
	ΔT (%)	40.61	38.74	46.48	44.95	33.29	25.17	18.63

Table 5.4: Comparison of CTST algorithm with 3-D bin packing approach [4] (d695).

(5) By applying our approach, we may optimally determine the top level TAM needs while minimizing the total test application time. For instance, SoC h953 has 12 inputs and 41 outputs. As we can see from Table 5.5, the minimum test application time is achieved for $P_{max} = 6 \times 10^9$ when the top level TAM is settled at 11.

Top level TAM W_{max}	$P_{max} = 6 \times 10^9$		$P_{max} = 7 \times 10^9$		$P_{max} = 8 \times 10^9$	
	CTST (P_{max}+RSC)	3-D (P_{max})	CTST (P_{max}+RSC)	3-D (P_{max})	CTST (P_{max}+RSC)	3-D (P_{max})
8	132821	-	132821	-	132821	-
9	123045	-	**119357**	-	**119357**	-
10	123045	-	119357	-	119357	-
11	**122457**	-	119357	-	119357	-
12	122457	-	119357	-	119357	-
16	122457	-	119357	-	119357	-
32	122457	122636	119357	119357	119357	119357
48	122457	122636	119357	119357	119357	119357
64	122457	122636	119357	119357	119357	119357
80	122457	122636	119357	119357	119357	119357
96	122457	122636	119357	119357	119357	119357
112	122457	122636	119357	119357	119357	119357
128	122457	122636	119357	119357	119357	119357

Table 5.5: Determine the top level TAM needs (h953).

5.5 Summary

We have presented a novel TAM scheduling algorithm based on a graph-theoretic formulation. Our major technical contributions are as follows:

1. We have formulated the test scheduling problem for SoCs as a graph theoretic problem by constructing the power-constrained test compatibility graph and conflict graph. From the conflict graph, we chose suitable candidate sets to initiate the scheduling. Based on the test

compatibility graph, we obtained the power-constrained test sets for concurrent scheduling.

2. We have scheduled constrained tests on TAMs. The test scheduling problem has been deduced to provide test access to the core level test terminals from the system level pins and efficiently scheduled the tests so as to reduce the total test application time.

3. We have studied the trade-off between the assigned TAM width and the corresponding test time of each core. The TAM width for each test was properly selected according to the power-constrained concurrent test sets (PCTS). Specifically, since a group of tests in a PCTS might be scheduled concurrently and utilized all available TAM bandwidth (which is a known parameter), we partitioned the bandwidth among tests in a way that the tests in the same PCTS have their test lengths close to each other. Thus the test time of each core was most possibly minimized while meeting test data bandwidth needs.

4. We have reduced the explicit dead time. The explicit dead time, i.e., the idle time between test sets, was reduced by the *dynamic test partitioning* scheme. The test sets in continuing PCTS's overlapped with each other in a way that a new test can be initiated immediately when a shorter test in a test session finishes.

5. Further research is needed to bring forth more SoC design and test features into test scheduling, such as test resource partitioning, design for testability and design for reconfiguration, etc.

Chapter 6

Wireless Test Control Architecture

When moving into the billion-transistor era, the direct or bus interconnects used in conventional SoC test control models are rather restricted in not only system performance, but also signal integrity and transmission with continued scaling of feature size. On the other hand, recent advances in silicon integrated circuit technology are making possible tiny low-cost transceivers to be integrated on chip. Based on the recent development in "Radio-on-Chip" technology, a new distributed multihop wireless test control network is proposed. In this chapter, we first introduce the basic network components. Then we present three proposed test control architectures, i.e., miniature wireless local area network, multihop wireless test control network, and distributed multihop wireless test control network.

6.1 Network Components

Three basic components are used in the proposed test control architectures: the test scheduler, the resource configurators and the RF (radio frequency) nodes dispersed on the SoC supporting the communication between the scheduler and the IP cores. The test scheduler is employed as a central controller, it (1) carries out the chip level test procedure, including the testing of the interconnects between the cores, the testing of the user-defined logics around the cores and the core testing, (2) communicates with the resource configurators and also with the chip external, such that no conflict arises during resource utilization and test application, (3) configures the routing of the test

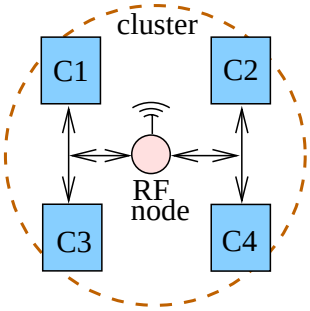


Figure 6.1: A RF node in a cluster of cores.

control path for each individual core, and (4) provides proper test control signals to carry out the test procedure of the selected core. The function of the resource configurator is to configure the test resources required for testing a particular core on command of the scheduler. A set of test resources (i.e., the circuit blocks required to perform testing) is distributed in the system for testing the cores. At any particular control step, each resource is configured into its appropriate operating mode by the control signals. In case when more than one test shares common test resources, the resource configurators are activated such that no conflicts result in the use of resources. The RF node works as a wireless test access node that has a radio-frequency interface (i.e., tiny low cost on-chip transceiver) for (two-way) communication between the scheduler and IP cores. Particularly, one RF node is dedicated to the scheduler. The distribution of RF nodes chip-wide provides the coverage of the entire on-chip wireless communication. While it is possible to equip each core with a dedicated RF node, it is more feasible to assign one RF node to each cluster of cores in order to reduce the cost due to the area and power overhead. IP cores are hard-wired to the RF node of its cluster, which has the RF interface. For example, as shown in Figure 6.1, cores c_1, c_2, c_3 and c_4 are organized into one cluster and are wired to the RF node. In addition, the IP cores in the system are organized into clusters and each has the IEEE P1500 wrapper interface to switch between different modes according to the control signals received. Note that, the wireless test circuitry is used only for testing purpose and will be deactivated or isolated during normal system operation, thus its impact on system performance is negligible.

6.2 Miniature Wireless LAN

Our first proposal is a miniature wireless LAN (local area network) that works as the intra-chip test control network for system-on-chips, where the scheduler broadcasts control signals through the attached RF node as shown in Figure 6.2. A single wireless channel is shared by all RF nodes in the chip and the control signals sent from the scheduler will be received by all RF nodes. Each RF node has a unique ID and each control signal is attached with an ID field to specify the intended recipient. Upon receiving a signal, a node checks the ID field through its local decoder. If the signal is intended for the receiving node, the node processes the control signal, otherwise, it is just ignored. By specifically assigning the ID (for example, reserving one bit to indicate multicasting while the remaining bits are to hold a group number), we can also support multicasting to a subset of RF nodes and consequently a subset of cores can be tested concurrently.

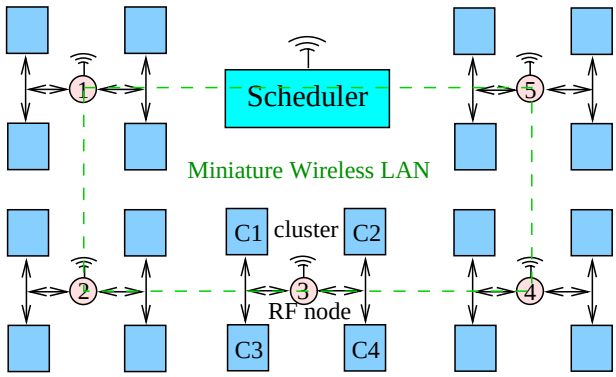


Figure 6.2: The illustration of miniature wireless LAN.

When a core finishes testing, the related RF node needs to notify the scheduler its completion. Since the schedule of the tests is predetermined, each RF node is given exclusive access to the network in a predetermined order. Permission to transmit signals to the scheduler is passed from one RF node to another using a special message called a poll and the polling order is maintained by the scheduler according to the schedule result. When the scheduler receives the completion signal from the RF node which holds the poll, it then forwards the poll to the next node in the polling sequence. This centralized polling scheme has its unique features as compared to the conventional polling network, which divides time into alternating types of intervals: polling intervals, during

which the poll is transferred between stations, and transmission intervals, during which the station with the poll transmits packets. Our scheme is quite simplified due to the fact that the scheduler knows in advance the completion time of each test and the transmission time is quite short. Thus it is not necessary to maintain the polling and transmission intervals. By using the polling scheme, no collision occurs even when multiple tests finish testing at the same time.

6.3 Multihop Wireless Test Control Network

With the simple design of miniature WLAN, all RF nodes should be within the transmission range of the controller, and the interference between RF nodes need to be carefully concerned. Since the transmission power grows with the transmission range to the power of 2 to 4, relaying signal between RF nodes may result in lower transmission power than communicating over large distance. In addition, the heat dissipated by higher power transmission may damage the surrounding circuits. Therefore, we propose a new low-power, high efficiency multihop scheme, where some RF nodes communicate through multiple "hop" routing.

6.3.1 Architecture Overview

Due to the limited transmission range of wireless network interfaces, multiple network "hops" may be needed for one RF node to exchange data with another across the network, we name this kind of network as **Multihop Wireless Test Control Network (MTCNet)**. In MTCNet, the IPs are placed at the leaves and RF nodes, i.e., wireless communication interface, placed at the vertices with the scheduler as the root of a tree structure. Figure 6.3 illustrates a MTCNet with 16 IPs. Each RF node is shared by a cluster of cores which are hard-wired to it so as to reduce routing cost and area overhead. For instance, as shown in Figure 6.3, since only RF nodes 1 and 2 are within the direct wireless transmission range of the scheduler, the transmission of the control signals between node 3 (or 4) and the scheduler is through the node 1 (or 2). Clearly, some nodes (for instance, node 1 or 2) operate not only as a host but also as a router, forwarding signals to other clusters in the network.

The routing cost in MTCNet involves wireless links between RF nodes and the scheduler, and hard-wiring between RF nodes and dedicated cores. In MTCNet, the location of the cores and re-

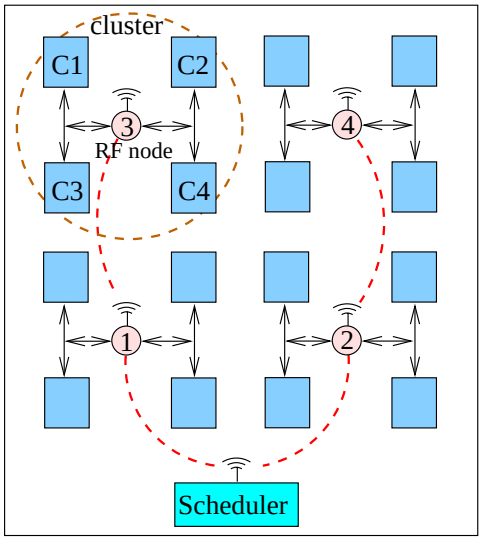


Figure 6.3: The illustration of MTCNet.

sources are fixed, and the placement of the RF nodes is predetermined. In such a static network, two issues need to be addressed with regards to routing. First, the cores need to be properly clustered such that the cost for hard-wiring a core to the RF node within its cluster is minimized. Core clustering depends on core functionality and resource sharing, and is determined before test scheduling. Second, an efficient topology needs to be formed such that the degree of each RF node (the number of neighbors with direct wireless links) should be small. Here, the topology is defined as the set of RF links between node pairs used explicitly by a routing mechanism.

6.3.2 Wireless Routing Algorithm

In MTCNet, a wireless routing protocol is needed to route the control signals over several hops to its destination. The existing routing protocols such as distance vector and link state for static infrastructure networks, or dynamic source routing (DSR) and ad hoc on-demand distance vector (AODV) for wireless ad hoc networks [76] cannot be directly used here, because all of them rely on powerful and complex hardware support, while an RF node is only a tiny wireless interface. A simplified but efficient routing protocol is needed for effective transmission in MTCNet.

We propose a packet-based source routing approach for communication between the RF nodes, where the scheduler specifies the route that a control signal should take through the network. Each

packet consists of a header, an IP address field which specifies the address of the destination IP, and a data field which contains the control signal. Each RF node is assigned a unique ID to specify the intended recipient, and each packet carries in its header the complete ordered list of node IDs through which the signal must pass. More specifically, the scheduler maintains a routing table, recording the shortest routes between any node and the scheduler, and works as the central node to make routing decisions. When sending a control signal, the scheduler puts the entire route into the header. Upon receiving a signal, a node checks the header through its local decoder. If the signal is intended for the receiving node, the node processes the control signal, otherwise, the intermediate node forwards the signal to the next hop accordingly. A shortest path algorithm (for example, Dijkstra's algorithm [77]) is needed to find the shortest route to any RF node in the network from the scheduler. The idea is to build a graph of the network, with each vertex representing an RF node and each edge between two nodes representing a wireless link, and the RF node attached with the scheduler is defined as the source. The shortest route between any destination and the source is found by running the shortest path algorithm on the graph. To facilitate parallel testing, the control signal sent from the scheduler may be directed to multiple destination nodes simultaneously, i.e., *multicasting*.

6.4 Distributed Multihop Wireless Control Network

In order to improve parallel test control processing, we propose an advanced hierarchical multihop scheme. In this architecture as shown in Figure 6.4, the scheduler is the system controller controlling a set of subsystem controllers which are distributed within the transmission range of the scheduler. Each subsystem includes a number of clusters and has a similar architecture as the basic network as shown in Figure 6.3. In this multilevel tree structure, the system controller will send the control information to the subsystem controllers which in turn control their subnetwork, such that efficient parallel communication is achievable. However, introducing a hierarchical level of controllers increases the test control overhead. In order to well balance the tradeoff between test application time and test control cost, the number of subsystem controllers usually equals to the maximum number of tests in a concurrent test set (within which all tests can be executed in parallel). Thus, the tests

that are simultaneously interrupted are processed by different subsystem controllers.

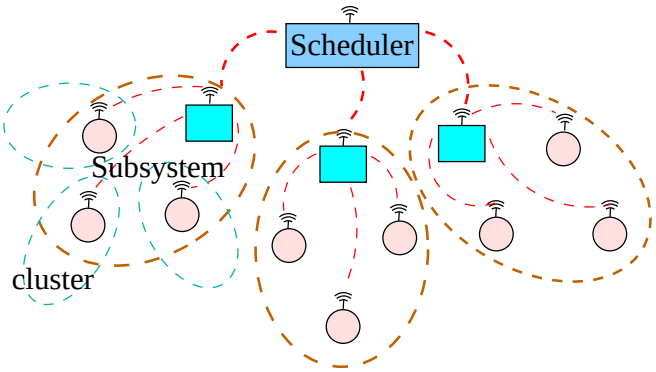


Figure 6.4: The distributed multihop architecture.

6.5 Test Control Overhead and Resource Partitioning

The test control cost in the system mainly includes the number of subsystem controllers and their complexity, the distribution of resources (including not only the circuitry to perform testing but also the RF nodes) and routing cost (including wireless routing as well as hard-wiring among clusters). The impact of test control overhead on the overall testing cost is briefly discussed below.

In order to minimize the overall testing time, nonconflicting test sets (i.e., there is no resource conflict between them and their total power meets the maximum power limit) are executed in parallel. A schedule for a system is made up of a set of test sessions, each consisting of a set of power-constrained concurrent test sets (PCTS) [78]. Corresponding to each PCTS, a set of controllers is needed to issue a set of control signals to parallel-process the controlling and these control signals are routed along different paths. Each control signal drives different test resources required for dedicated cores in the same PCTS. This directs the partitioning of the test resources, where each partitioning would be driven by the same set of control signals. Thus the test resources driven by the same control signal would be physically adjacent to each other.

Due to parallel controlling, the number of controllers relies on the maximum number of tests (size of a PCTS) in the PCTS's. Different scheduling will result in different size of PCTS's and in turn requires different number of controllers, and vice versa. Moreover, the complexity of each

controller depends on the number of control signals it needs to generate. The number of control signals required to execute a test may vary from one to another. Hence, the lower the disparity in the number of control signals that are necessary for different test sets in the same PCTS, the more cost-efficient the controller becomes.

The wiring cost can be significantly reduced if the cores sharing the same resources are physically adjacent. Thus the clustering of cores is performed according to their physical locations and the sharing of resources. Further, the clustering also depends on the placement of RF nodes (which will be discussed in the next subsection) as each RF node covers the communication of a cluster of cores. Note that different partitioning of test resources and RF nodes may result in different clustering of the cores which affects the concurrency relation between the cores, and accordingly different test schedules. Thus the reduction in the test application time does not necessarily imply the reduction in overall testing cost. In addition, as the cores within a cluster are adjacent to each other, the wiring cost is properly reduced, and the routing cost is mainly determined by the number of RF nodes for constructing an efficient topology. In summary, reduction in test control overhead should aim at proper partitioning of overall resources (including not only the testing resources but also the communication resources) and the concurrent scheduling of tests.

6.6 The Placement of RF Nodes

Given an SoC embedded with n cores and m test resources, the test sets parameters to perform testing, and a tentative floor plan of the cores, where a core c_i has the coordinates (x_i, y_i). Each RF node has a maximum assistant distance R, i.e., the maximum range for connecting a core to the RF node. The *RF nodes distribution problem* (P_{RD}) is deduced to finding the minimum number of RF nodes needed to cover the communication of all cores within the chip and their placement.

6.6.1 System Modeling

The P_{RD} problem can be formulated into *geometric disk covering* [79, 80], where a (clustered) wireless control network can be abstracted as a set of disks, each centered at an RF node with a radius of R, that covers a set of embedded IP cores (wireless clients) in the chip plane. It is known

that this problem is strongly NP-complete [79]. A graph $G(V,E)$ is used to represent the system. V is a set of vertices, each representing an embedded IP core. E is a set of edges, each connecting two vertices within a distance of $2R$. Assuming an optimally placed RF node can assist at least two IP cores, i.e., a disk covers at least two vertices (it becomes trivial if one RF node can assist only one core), we can always move the RF node such that there are two vertices on the circumference of the disk, while the disk covers the same set of vertices (see Figure 6.5(a)) [80]. For each edge, an RF node can be placed in two ways such that the two vertices connected by the edge are on the circumference of the disk (see Figure 6.5(b)). Thus there are maximum $2|E|$ possible disk placements need to be considered. The position of each disk is specified by its center, i.e., each RF node placement $C_i(X_i, Y_i)$ $(1 \leq i \leq 2|E|)$, which can be computed according to the coordinates of the two given points as illustrated in Figure 6.5(b). By formulating the RF nodes distribution into disk covering, we can apply existing heuristic and approximation algorithms to solve it [77, 80–82].

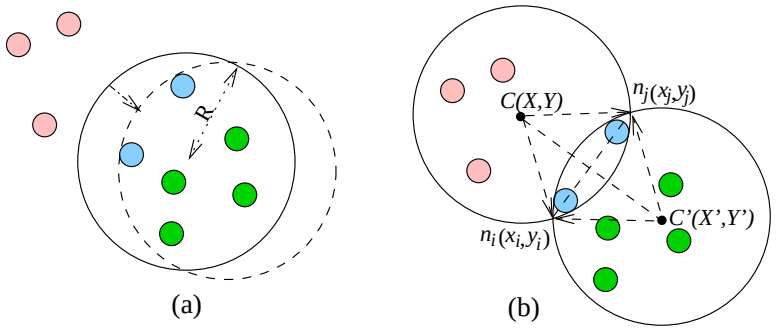


Figure 6.5: The illustration of disk covering.

6.6.2 Greedy Set Covering Scheme

In [83], we have discussed a greedy heuristic which selects at each iteration the disk that covers the largest number of IP cores which have not been selected. The basic idea is that when we obtain $M(\leq 2|E|)$ possible RF node placements, each corresponding disk covers a set of vertices, denoted as set S_i. Therefore, the original problem is deduced to choosing a minimum $K(\leq M)$ sets from the M sets to cover all IP cores in the chip with each set assisted by one RF node, i.e., to keep the coverage of the K sets equal to $|V|$. This is a typical set-covering problem, which is NP-hard [77]

and efficient heuristic algorithms [77, 81] exist to solve it. For example, a simple greedy algorithm presented in [77] can be applied (e.g., it picks at each iteration the set that covers the maximal number of uncovered points) with a polynomial time and ($ln|M| + 1$) approximation ratio.

6.6.3 Grid Disk Covering Scheme

Hochbaum and Maass have presented in [80] a polynomial approximation scheme that applies a shifting technique in the context of planar graphs. More recently, Franceschetti et al. [82] have proposed a grid strategy to find a covering of the points by placing disks only at the vertices of a mesh. We further present a grid disk covering scheme using a divide-and-conquer method that combines the grid strategy with the shifting technique.

We assume the shift parameter to be l. Two nested application of the shift strategy is used. First, the chip plane is cut into vertical strips of width $l \times D$ (groups of l consecutive strips of width D=2R are considered). By repeating the shift of all groups l-1 times over length D, there are l different ways of partitioning of the plane into strips of $l \times D$ wide, each partitioning denoted as S_i ($1 \leq i \leq l$). Then, in order to cover the points in such a strip, the shifting strategy is applied in the other dimension. Thus, the considered strip is cut into squares of side length $l \times D$. The optimal covering of points in such a square is found by applying the grid strategy. As we can see that with $\lceil l\sqrt{2} \rceil^2$ disks of diameter of D, the square of side length $l \times D$ can be covered compactly, i.e., the number of disks to cover points in the square does not exceed $\lceil l\sqrt{2} \rceil^2$. Thus, the number of possible disk positions is finite. By checking all possible arrangement of maximum $\lceil l\sqrt{2} \rceil^2$ disks, an optimal covering is found within the square. In addition, the coverage of an RF node C_i is represented by a set of IP cores covered by the disk, denoted by P_i. Let A be the grid covering algorithm that provides optimal covering within each square. For a given partition S_i, let $A(S_i)$ be the divide-and-conquer approach that applies A to each square and outputs the union of all disks used (i.e., $\cup_{i=1}^{K} P_i$=$|V|$). The minimum cardinality of the l partitioning is chosen to be the final best covering. The proposed algorithm has a polynomial time approximation complexity of $O(l^2 \lceil l\sqrt{2} \rceil^2 (2N)^{2\lceil l\sqrt{2} \rceil^2 + 1})$ with performance ratio $\leq (1 + 1/l)^2$.

6.6.4 Clustering Option

Further enhancement is employed by considering the core clustering and the workload balancing during the placement of RF nodes. We first specify the clustering options due to resource constraint. For example, some cores, having similar functionality, or competing for the same resource, or physically adjacent with precedence constraint, can be grouped into the same cluster and share the same RF node. We use a virtual core c_{g_i} to represent each such group of cores labelled with the workload of the number of cores in the group. We also need to specify the hierarchical cores and use a virtual core c_{h_i} to represent a hierarchy of cores. Moreover, in order to overcome the situation where most of the cores are assigned to one or a few of RF nodes, we add the maximal load requirement. In other words, we bound the maximum number of IP cores assigned to any particular RF node by $|S_i| \leq L$ ($|S_i|$ is the size of set S_i which is covered by the RF node, naturally B has to be large enough, i.e., $B \geq N/k$, where N is the number of cores and k is the number of RF nodes). In order to incorporate the maximal load requirement, we split a set S_i of size $|S_i|$ into the number $C^B_{|S_i|}$ of subsets with size of B.

6.6.5 Simulation Study

We evaluate the proposed algorithm via simulation. We assume the chip size of 100×100. The number of IP cores in the system is denoted by N, the maximum assistant distance of RF nodes by R, and the load bound by B. We assume that the chip plane represents the universal set, each IP core placement is viewed as a single point with (x, y)-coordinates, and thus the N points to cover represent the elements of the sets. We study the effect of N, R and B on the number of RF nodes. The simulation result is shown in tables 6.1, 6.2. As we can see, when increasing the number of IP cores in an SoC, the number of RF nodes required to provide the wireless coverage chip-wide increases. For the same number of IP cores, the number of RF nodes decreases generally when B or (and) R increases. As we know, the number of RF nodes determines the wireless routing cost while the maximum RF node assistant distance determines the hard-wiring cost between the RF nodes and the clusters of the cores. The wireless routing cost reduces at the expense of hard-wiring cost, and vice versa. In addition, the number of RF nodes implicitly influences the concurrency testing

between the cores, as the cores using different RF nodes can be tested in parallel if meeting all conflicts and constraints. The higher the number of RF nodes which increases the routing cost, the higher the possibility of test concurrency which reduces the total testing time. Therefore, efficient optimization techniques need to be developed to minimize the overall testing cost.

N	B=5	B=10
10	7	7
20	10	10
30	14	14
40	18	17
50	18	18
60	19	18
80	25	19
100	28	22

Table 6.1: Number of RF nodes with the changing of N and B when R=10.

N	R=10	R=15	R=20	R=25
10	7	5	5	3
20	10	8	6	5
30	14	11	8	5
40	17	12	8	6
50	18	12	10	8

Table 6.2: Number of RF nodes with the changing of N and R when B=10.

6.7 Summary

In this chapter, we have proposed a novel distributed wireless test control network using the "Radio-on-Chip" technology for future high-density, high-volume embedded cores-based SoCs. Three types of test control architectures, i.e., miniature Wireless LAN, multihop wireless test control network, and distributed multihop wireless test control network have been presented and the system

optimization has been performed on RF nodes placement. Simulations using randomly generated test sets have been carried out for evaluation and verification of the proposed test optimization algorithm. Several system optimization issues such as RF nodes placement, clustering, and routing will be further addressed in the following chapter. Techniques need to be presented for the integration of test resource distribution and system optimization among TAM design, test scheduling under power and cost constraints.

Chapter 7

Cost Oriented Resource Distribution and System Optimization

With the application of "Radio-on-Chip" technology in test control, several system optimization issues, such as RF nodes distribution, core clustering, and control routing (wireless routing as well as hard-wiring between the IP cores and the RF nodes), are brought forward for the control constrained resource partitioning and distribution. One major problem in wireless test control architecture is to design a test procedure to ensure quality testing of all IP cores while minimizing the overall testing cost. In this chapter, we present the optimization technique for the integration of test resource distribution (specifically, the test access mechanism routing between pairs of test source and sink) and wireless test control distribution (i.e., to build the network of RF links for intra-chip communication). We further propose a system optimization scheme to minimize the combined cost of overall testing time and test control under power constraint and resource conflict. We assume that a multihop wireless test control architecture is used to reduce transmission power, to avoid chip overheating, and to achieve the most efficient control processing.

7.1 An Integrated Test Model for System Resource Distribution

With the introduction of on-chip wireless test control, the system resources in an SoC consist of two parts, the circuit blocks required to perform a test (the test resources) and RF nodes in the intra-chip

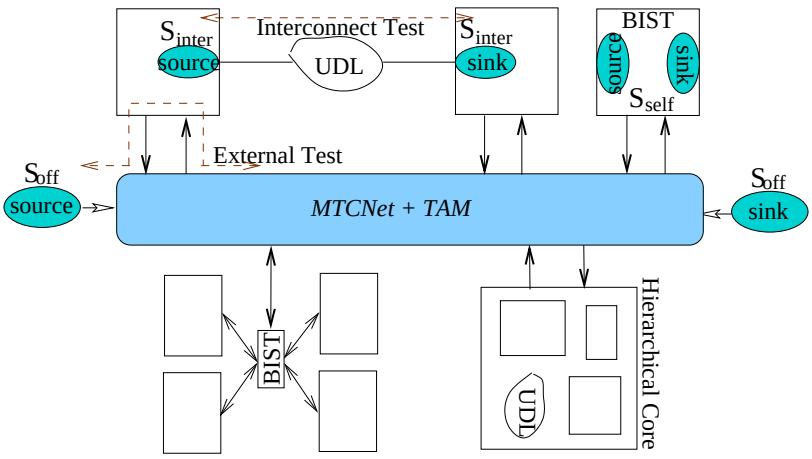


Figure 7.1: An integrated system framework.

wireless test control network. We have addressed the distribution of RF nodes in [84], and the test resource distribution mainly focuses on the optimal routing of test access mechanism (TAM) from a dedicated test source to the core-under-test (CUT) and from the CUT to a dedicated test sink. In this section, we present a test model (in Figure 7.1) for the integration of system resources in concurrent testing of core internals and externals under wireless test control.

There are three types of test pattern source and sink used in this model, S_{off}, S_{self}, and S_{inter} as shown in Figure 7.1. S_{off} is implemented off-chip by using external ATE, and S_{self} and S_{inter} are implemented inside a chip (i.e., on-chip). S_{self} is used for testing the core itself (such as in BIST-enabled core), while S_{inter} is used for testing interconnects. We assume that each individual core is testable by either BIST or an external test or a combination of them. Meanwhile, we take into consideration core testing (IEEE P1500 wrapped cores) as well as interconnect testing (which includes the testing of interconnect logics, UDLs and wiring). We assume that the IP cores in an SoC have the IEEE P1500 wrapper interface which switches between different modes, internal test mode, external test mode and normal function mode, according to the control signals received. TestRail [15] is adopted as the test access mechanism as it facilitates interconnect test as well as core test. During external test, the core requires a high data bandwidth, while in BIST the requirement is low (it only needs one bit TAM for initialization thus the BIST can be executed autonomously and after that need one bit TAM to shift out the signature for comparison).

Various test conflicts may appear during core test and interconnect test. For example, special care should be taken during the testing of a UDL and the two cores c_i and c_j connecting to the UDL. Specifically, when testing c_i, the wrapper surrounding the core is put into internal test mode and test stimuli is transported from the required test source to c_i and the test response is transported from c_i to the test sink through a set of TAM wires. When testing the UDL, the wrappers of c_i and c_j are put into external test mode and the test stimuli is transported from the test source via c_i to the UDL and the test response is transported from the UDL via c_j to the test sink. Obviously, the testing of the UDL and cores c_i and c_j cannot be carried out at the same time due to wrapper usage conflict. In order to unify the problem formulation, we treat interconnects as unwrapped cores with the consideration of wrapper usage conflict. A test conflict also occurs when the common test resources are shared among a set of cores or the same core is tested by several test sets. Thus the test sets for the same core cannot be executed at the same time and the same resource can only be used by one test set at one time.

7.2 SoC Testing Cost Optimization

Various test scheduling and wrapper/TAM optimization algorithms have been proposed in the literature to reduce test cost in terms of test application time. However, less attention is paid to test control cost which constitutes a major part of the total test overhead. In this section, we analyze and formulate SoC testing cost, more specifically, the cost of test control distribution and test resource distribution.

7.2.1 Cost of Test Control Distribution

In MTCNet, the neighboring IP cores are clustered in such a way that one on-chip RF node is shared by a cluster of IPs (see Figure 7.2). Assuming that the maximum assistant distance of an RF node is R (i.e., the maximum range for connecting an IP to the RF node), it may result in different clustering of IPs by varying the value of R, and accordingly, different test control routing cost. The cost of routing a control signal to its destination IP A includes the wireless communication cost that is represented by the number of RF nodes needed to forward the signal from the scheduler to A, and

the hard-wiring cost that is represented by the total wire length between A and the associated RF node. Specifically, the total *test control routing cost* of A is represented by:

$$C_{control\_a} = C_{wire} \times l_{wire\_a} + C_{rf} \times n_{rf\_a} \tag{7.1}$$

where C_{wire} is the unit hard-wiring cost and C_{rf} is the cost of an RF node. l_{wire_a} is the total wiring length between core A and the RF node within the cluster. We estimate the length of wires between a core and its RF node using Manhattan distance function, $l_{wire\_a} = |x_a - x_c| + |y_a - y_c|$, where the center-coordinates of core A and its RF node C are (x_a, y_a) and (x_c, y_c), respectively. n_{rf_a} is the total number of RF nodes needed to route the control signal to A. The maximum RF node assistant distance determines both the number of RF nodes needed and the hard-wiring between the RF nodes and the cores in their clusters. Clearly, the wireless communication cost reduces at the expense of increased hard-wiring cost, and vice versa.

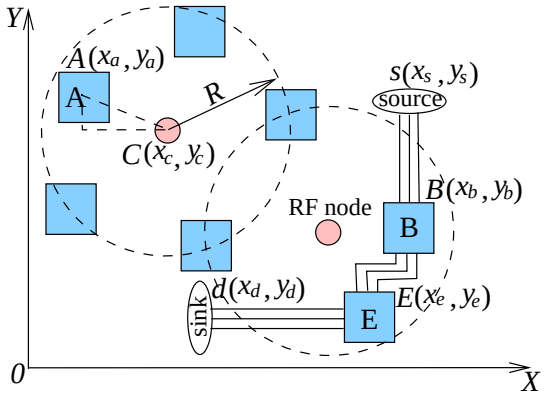


Figure 7.2: Clusters of IPs each sharing one RF node.

In order to reduce the overall test control routing cost, we determine the optimal distribution of RF nodes on-chip, i.e., the optimization of *RF node placement*. We have mapped the problem of RF node placement to disk covering problem in [83], where a set of disks, each centered at an RF node with a radius of R covers all IP cores in the chip. In order to optimize the test control distribution, we efficiently distribute RF nodes on chip and minimize the associated test control cost by the control routing cost function.

7.2.2 Cost of Test Resource Distribution

The distribution of on-chip test resources mainly addresses the routing of IP cores to a set of test sources and sinks and assigning their data transportation bandwidth accordingly, i.e., the *TAM routing*. Each core may have its dedicated test source and test sink or may share either test source or test sink with other IPs or may share both test source and test sink. Given the placement of test source s and test sink d, the total cost of routing an IP A on a TAM with bandwidth of W_a is represented by:

$$C_{TAM\_a} = C_{wire} \times W_a \times l_{wire}(s \rightsquigarrow a \rightsquigarrow d) \tag{7.2}$$

Similarly, the Manhattan distance function is used to calculate the TAM wire length from s to A to d. For example, the TAM wire length connecting cores B and E (with center-coordinates as shown in Figure 7.2) is $l_{wire}(s \rightsquigarrow b \rightsquigarrow e \rightsquigarrow d) = dist(s,b) + dist(b,e) + dist(e,d) = |y_s - y_b| + (|x_b - x_e| + |y_b - y_e|) + |x_d - x_e|$.

Given core clustering, there are three cases to route the TAMs according to resource conflicts among the cores.

- **Case 1**: The cores connected on the same TAM belong to different clusters, as shown in Figure 7.3(a). For concurrent testing of cores A and B in the same cluster, unicast the control signal shared by A and B to RF node 1 in their cluster, which is further forwarded to A and B along hard-wires separately.

- **Case 2**: The cores connected on the same TAM belong to the same cluster, as shown in Figure 7.3(b). For concurrent testing of A and B in two separate clusters, multicast the same control signal to their dedicated RF nodes 1 and 2 along two separate wireless routing paths. Compared with case 1, case 2 has less TAM routing (as neighboring IPs are connected on the same TAM) but more control routing (due to multicasting).

- **Case 3**: Combination of case 1 and 2, as shown in Figure 7.3(c).

Moreover, the TAM bandwidth affects the test application time of a test significantly due to various wrapper scan chain configurations. According to Pareto optimization [44], the test times of

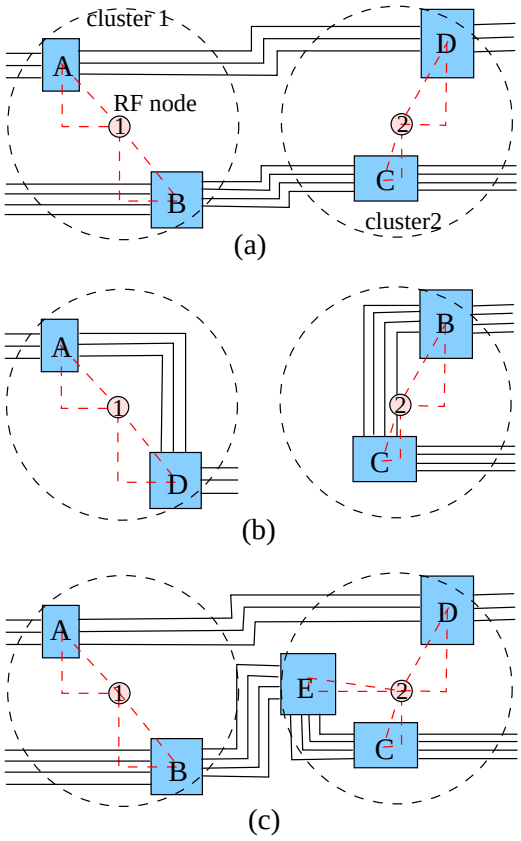


Figure 7.3: Three cases of TAM routing.

a core A vary with TAM width as a "staircase" function:

$$t_a(T_a, W_a) = (1 + \max\{L_a(W_a)\}) \times P_a + \min\{L_a(W_a)\} \tag{7.3}$$

where, $L_a(W_a)$ is the length of the wrapper scan chain when the TAM width is W_a, and P_a is the number of test patterns for testing T_a. In order to reduce the overall test application time, an efficient test scheduling algorithm needs to be applied. It connects the IPs in the SoC to the test sources and sinks in a way that compatible tests are routed on parallel TAMs while the cores competing for the same test resource are connected sequentially on the same TAM. In the meantime, it reduces the cost of overall system resource distribution in terms of TAM routing as well as test control cost $C_{res} = C_{control} + C_{TAM}$.

7.3 Cost Oriented Resource Distribution

The goal of designing a test procedure is to minimize the overall testing cost including both the test application time and the hardware cost associated with the testing process. In this section, we propose an integrated optimization technique to efficiently distribute system resources on-chip and minimize the overall test application cost. The problem of cost oriented system resource distribution can be formally stated as follows.

Problem ⟨*System_Resource_Distribution*⟩

Given is an SoC embedded with N cores and M test resources, and a tentative floor plan with center-coordination system. Also given is the test set parameters to perform testing on each core T_i ($1 \leq i \leq N$), such as the number of test patterns P_i, the number of functional inputs/outputs I_i/O_i, the number of scan chains s_i and their lengths $l_{i,k=1 \sim s_i}$. Furthermore, given is the top level TAM constraint W_{max} (due to pin count limit). Determine the optimal scheduling of N IPs on the total width of W_{max} TAMs while meeting various constraints with the objective of minimizing the overall testing cost $C_{all} = \alpha \times T_{appl} + \beta \times C_{res}$ with user specified relative weight α/β (where C_{all} is the overall testing cost of an SoC, T_{appl} is the overall test application time, and C_{res} is the overall cost on system resource distribution).

We introduce a cost oriented system optimization algorithm. The basic idea is to specify four highly interdependent design items, i.e., the rectangle test set (with different configuration of wrapper scan chains) selection, the control resource floorplanning (including RF nodes placement and core clustering), the TAM design (i.e., TAM routing and TAM bandwidth distribution), and the system level test scheduling (to achieve maximum parallelism). The overall test application time can be minimized by executing IP cores in parallel, but the possibility of concurrent testing depends on the TAM bandwidth between any particular pair of test source and sink and the TAM routing through the IPs. The placement of RF nodes has a direct impact on core clustering which influences concurrent testing and TAM routing. Finally, the partitioning over wrapper scan chains impacts the TAM design. The pseudo-code of the algorithm is given in Figure 7.4, which includes three main steps inside an iteration of the changing of maximum RF node assistant distance R.

Procedure System_Resource_Distribution()

input: N /* # of IP cores embedded in an SoC */
M /* # of sources and sinks */
$n_i(x_i, y_i)$ /* floorplan of all nodes on chip */
R /* max assistant dist of RF node */

Preprocessing:
1 obtain a set of candidate rectangle set of each core;
2 specify constraint list;
3 estimate the initial rectangle test set of each core;

1 initialize min(C_{all});
2 construct a directed weighted graph;
3 for($l_{min} <= R <= l_{max}$)
/* RF node placement */
4 create_disk_cover();
5 greedy_set_cover();
6 place RF node with load balancing;
7 cluster cores by hardwiring to RF node;
8 calculate $C_{control}(R)$;
/* TAM routing */
9 while($num\_scheduled < N$)
10 multisource_shortest_path();
11 get the shortest path among all pairs of source−sink;
12 record the shortest path;
13 record the length of the shortest path;
14 remove IPs constructing the shortest path from the graph;
15 update $num\_scheduled$;
16 adaptive TAM width adjustment;
17 calculate $C_{all}(R)$;
18 if $C_{all}(R) <$ min(C_{all})
19 min$(C_{all}) = C_{all}(R)$;

output: min(C_{all});
minimum # of RF nodes and their placement;
clustering of cores;
parallel TAM routing between pairs of sources−sink;

Figure 7.4: The system resource distribution algorithm.

7.3.1 A Disk Covering Algorithm for RF Distribution

The first step of this algorithm is to find the minimum number of RF nodes needed to cover the communication of all cores within the chip and their placement, and obtain the minimum overall test control cost $C_{control} = C_{wire} \times L_{wire} + C_{rf} \times N_{rf}$. In order to optimally distribute control resources, we formulate the problem into geometric disk covering, and solve it with greedy set covering as described in Sec. 7.2.1. For any particular R varying from l_{min} (the shortest distance between any two cores) to l_{max} (half of the chip side-length). We list the number of $|F| \leq 2|E|$ possible disk placement which represents the sets to choose from (as shown in Figure 7.5). Then,

the greedy heuristic (in Figure 7.6) is pursued that selects at each iteration the set that covers the maximal number of uncovered points. Furthermore, we incorporate a control cost function and the optimization of control distribution is oriented by minimizing the overall control cost (see lines4-8 in Figure 7.4). Further enhancement is employed by proper core clustering. For example, workload balancing is pursued to overcome the communication congestion problem when most IPs are assigned to some of RF nodes. In addition, if a core is settled at the overlap region of several clusters, it is hard-wired to only one of the RF nodes that results in the shortest hard-wire length and balanced workloads.

Procedure create_disk_cover( )

```
for n_i(x_i, y_i) (1 ≤ i ≤ N−1)
    for n_j(x_j, y_j) (2 ≤ j ≤ N)
        if ( dist (n_i, n_j) ≤ 2R)
            obtain two disk placements
            /* with n_i, n_j on the border */
            calculate c(x, y) and c'(x', y')
            /* coordinates of the centers of the disks */
            for n_i(x_i, y_i) (1 ≤ i ≤ N)
                if ( dist (n_i, c) ≤ R)
                    S ← n_i    /* assisted by RF node c */
                if ( dist (n_i, c') ≤ R)
                    S' ← n_i   /* assisted by RF node c' */
            F ← { S, S'}
```

Figure 7.5: Listing all possible RF nodes placement.

Procedure greedy_set_cover( )

```
U ← {n_i}
/* set U including all uncovered nodes */
C ← ∅
/* set C includes the result */
while (U ≠ ∅)
    select set S ∈ F that maximizes |S ∩ F|
        U ← U − S
        C ← C ∪ S
return C
```

Figure 7.6: Greedy set covering algorithm.

7.3.2 A Shortest Path Algorithm for TAM Routing

The next step is to schedule the tests on parallel TAMs which is performed upon the routing of TAMs from sources to sinks with the optimization of weighted cost of test application time and TAM routing, i.e., $C_{total_i} = C_{appl} \times T_{TAM_i} + C_{wire} \times C_{TAM_i}$ (where, $i \in \Upsilon$, a set of TAMs).

A directed weighted graph $G(V, E)$ is constructed to represent the system as follows:

- Each pair of test source and sink functioning as the source s_i ($1 \leq i \leq M$) and destination d_i is connected to/from all vertices of cores with directed weighted edges. A source is connected to any vertex with the weight of $wt(s_i \rightarrow T_j) = C_{appl} \times t_j + C_{wire} \times W_j \times dist(s_i \rightsquigarrow T_j)$. Meanwhile, any vertex is connected to a sink by $wt(T_j \rightarrow d_i) = C_{wire} \times W_j \times dist(T_j \rightsquigarrow d_i)$.

- Any two vertices of cores are connected to each other by bidirectional edges. The weight of an edge from T_i to T_j is represented by $wt(T_i \rightarrow T_j) = C_{appl} \times t_j + C_{wire} \times W_j \times dist(T_i \rightsquigarrow T_j)$, and similarly, the opposite edge has a weight of $wt(T_j \rightarrow T_i) = C_{appl} \times t_i + C_{wire} \times W_i \times dist(T_j \rightsquigarrow T_i)$. Obviously, $dist(T_i \rightsquigarrow T_j) = dist(T_j \rightsquigarrow T_i)$.

- Each node v_i (including cores and sources and sinks) in the system maintains a shortest distance to the source (denoted by v_i_dist), and a predecessor (denoted by v_i_pred. Each node will be in one of the three states: *state 1*, initial, $v_i\_pred = NIL$ and $v_i\_dist = \infty$; *state 2*, updated, the distance and the predecessor have been updated at least once; *state 3*, finalized, the distance is the shortest distance to the source, and it will not be updated in the future.

The major feature of this approach is to run a cost oriented *multisource_shortest_path* algorithm on a reduced graph (see lines9-15 in Figure 7.4). Each time a shortest path is selected after running the single source shortest path algorithm for all dedicated pairs of source-sink. The covered vertices of IPs are deleted from the graph, and the above operation is repeated again until all IPs are routed. Each selection of a shortest path specifies the IPs routed sequentially on the same TAM which results in the minimum total cost among all given pairs of source-sink. The *multisource_shortest_path* algorithm (in Figure 7.7) resolves an multiple-pairs shortest path problem by running a single source shortest path algorithm (e.g., Dijkstra's algorithm) k times, once for each dedicated pair of test source and sink. As we can see, for each pair of source-sink, a shortest (least-weight) path is

constructed where the weight of a path is the sum of the weights of its constituent edges. Thereby, the distance of a finalized sink indicates the minimum overall cost accumulated along a path going through the source and then a subset of IPs and lastly the sink.

Procedure multisource_shortest_path( )

```
/* initialize all sources */
for 1<=i<=M+N        /* V(G), |V|=M+N */
    v[i].dist <- INFTY
    v[i].pred <- NIL
for 1<=i<= m         /* m is the # of sources */
    v[i].dist <- 0
for 1<=k<= m
/* for each pair of source−sink, run Dijkstra's algo */
    S <- ∅
    num_pq <- N+2        /* the # of nodes in priority queue */
    Q[1]<- s[k]          /* priority queue */
    Q[N+2]<- d[k]
    for 1<=i<= N
        Q[i+1]<- v[i+M]
    while(num_pq >0)
        u <- Q[1]
        Q[1] <- Q[num_pq ]
        Dequeue(1, Q, num_pq )  /* downheap priority queue */
        S <- S ∪ {u}
        /* relax(u, v[i], wt(u->v[i]))*/
        for v[i]∈Adj [u]
            if v[i].dist > u.dist + wt(u->v[i])
                v[i].dist = u.dist + wt(u->v[i])
                v[i].pred = u
                Enqueue( i , num_pq ) /* upheap priority queue */
```

Figure 7.7: Multisource shortest path algorithm.

Note that, we start TAM routing with the estimated rectangle test set for each core. For core T_i, we assume that the area defined by its test time multiplied by its TAM width does not change due to balanced wrapper scan chain configuration, i.e., $1 \times t_i(1) = W_i \times t_i(W_i)$, thus,

$$t_i(W_i) = \frac{t_i(1)}{W_i} \tag{7.4}$$

As the shortest path is constructed by optimizing the weighted cost of test application time and TAM routing cost, $C_{total_i} = C_{appl} \times t_i(W_i) + C_{wire} \times W_i \times l_{wire}(s \rightsquigarrow T_i \rightsquigarrow d)$, we derivate the cost function with respect to W_i and get,

$$\frac{dC_{total_i}}{dW_i} = -C_{appl} \times \frac{t_i(1)}{W_i^2} + C_{wire} \times l_{wire}(s \rightsquigarrow T_i \rightsquigarrow d) \tag{7.5}$$

and by letting C'_{total_i}=0 we calculate,

$$W_i = \sqrt{\frac{C_{appl} \times t_i(1)}{C_{wire} \times l_{wire}(s \rightsquigarrow T_i \rightsquigarrow d)}} \tag{7.6}$$

After getting a second derivative, we obtain $C''_{total_i} > 0$, i.e., a minimum. For estimation, we set the initial TAM width $W_{i_{init}}$ of core i to the closest pareto optimal point such that $|W_{i_{init}} - W_i|$ is minimum, and we obtain the corresponding test time $t_i(W_{i_{init}})$.

7.3.3 Adaptive TAM Redistribution

After we route the TAMs between pairs of source-sink, each TAM sequentially connecting to a set of rectangle IP cores with the length of estimated test time and the height of associated TAM width respectively, further optimization need to be carried out in following cases:

- As we use the estimated rectangle test set as the initial value for each core, the cores on the same TAM may be assigned different TAM width. We may adjust the TAM width of some cores to fully utilized the bandwidth $W_{TAM_i} = max\{W_j\}$ ($j \in \gamma$, a subset of cores on TAM i), and accordingly reduce the total cost C_{total_i} on this TAM, where $T_{TAM_i} = \sum_{j=1}^{\gamma} t_j$, and $C_{TAM_i} = W_{TAM_i} \times L_{TAM_i}$, $L_{TAM_i} = dist(s_i \rightsquigarrow T_j \rightsquigarrow T_{j+1} \rightsquigarrow ... \rightsquigarrow d_i)$.

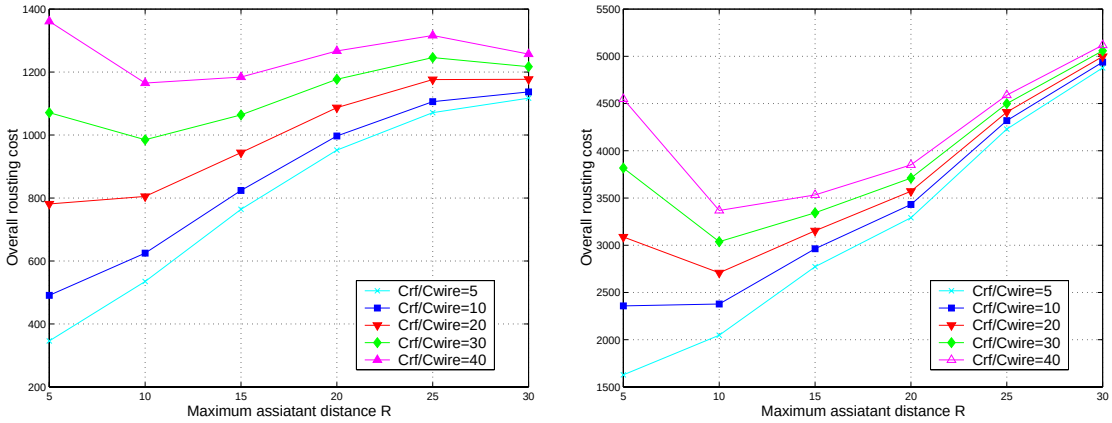
- If the total TAM width $W_{total} > W_{max}$ ($W_{total} = \sum_{i=1}^{\Upsilon} W_{TAM_i}$), we reduce the width of TAM i by 1, which results in the minimum total cost $C_{total} = C_{appl} \times T_{appl} + C_{wire} \times C_{TAM}$ (where $T_{appl} = \max\{T_{TAM_i}\}$, $C_{TAM} = \sum_{i=1}^{\Upsilon} C_{TAM_i}$), and then check W_{total} again, so on and so forth, until $W_{total} \leq W_{max}$.

- On the other hand, if $W_{total} < W_{max}$, we distribute the remaining bandwidth to a TAM i by 1 each time, which results in the minimum total cost C_{total}.

7.3.4 Simulation Study

In simulation scenario 1, the overall routing cost with R ranging from 5 to 30 is shown in Figure 7.8(a) and 7.8(b) with 50 and 200 cores, respectively. As we can see, if the cost of an RF node is relatively low ($C_{rf}/C_{wire} < 20$), one may deploy as many RF nodes as possible to minimize

the overall cost. On the other hand, when an RF node is expensive ($C_{rf}/C_{wire} > 20$), the optimal value of R (and accordingly the number of RF nodes) can be determined based on the lowest overall routing cost. For example, $R = 10$ is optimal as shown in Figure 7.8.



(a) C_{route} changing over R when N=50.

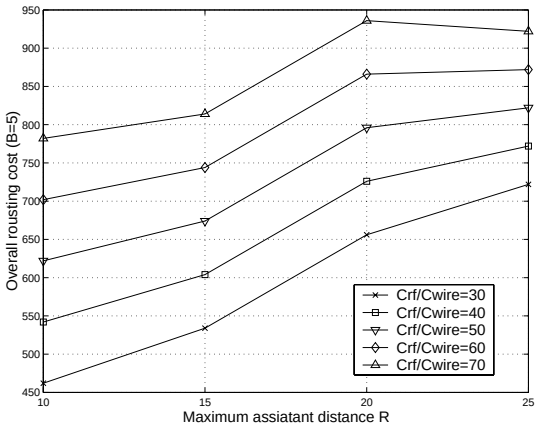
(b) C_{route} changing over R when N=200.

Figure 7.8: Illustration of the overall routing cost optimization.

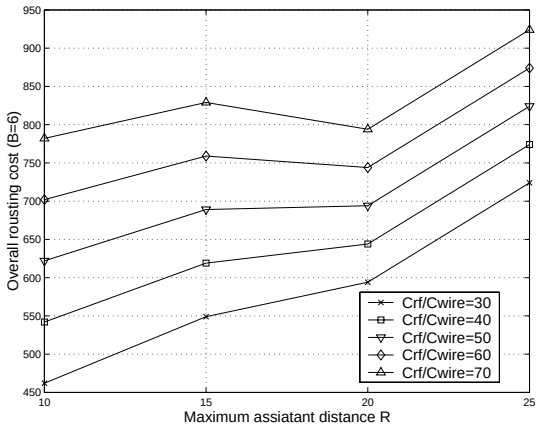
In simulation scenario 2, we study the effect of the workload balancing on the overall routing cost optimization and determine the minimum test control cost assuming an example SoC with 25 cores distributed on the chip with size of 50×50. We run the routing cost optimization algorithm on the SoC when R changing from 10 to 25. As shown in Figure 7.9, we obtain the optimally placed RF nodes and accordingly the overall routing cost. As we can see, when B is 5, the overall routing cost increases as R increases, and we obtain the lowest C_{route} when R=10. That means, we use as many RF nodes as we can to reduce the routing cost. When B increases, the C_{route} curve drops at first and then increases. When B=8, the lowest C_{route} is settled at R=20 ($C_{rf}/C_{wire} > 50$), i.e., the optimal number of RF nodes is shifted to 20.

In simulation scenario 3, we present a sample of the experimental results for the proposed cost oriented *System_resource_distribution*. We use SoC *d695* from ITC'02 SOC Test Benchmarks [75] for illustration. The floorplan of the cores embedded on-chip is randomly generated, as the original benchmarks do not provide any related information.

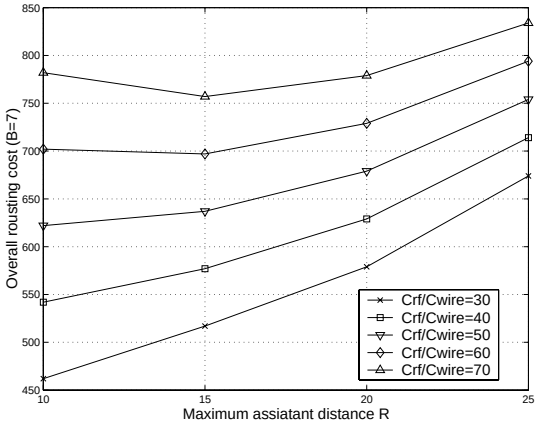
As shown in Figure 7.10, we obtain the minimum overall control routing cost when R=20 (at



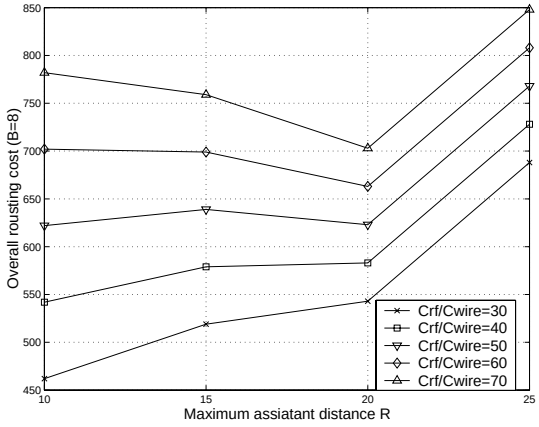
(a) C_{route} changing over R when B=5.



(b) C_{route} changing over R when B=6.



(c) C_{route} changing over R when B=7.



(d) C_{route} changing over R when B=8.

Figure 7.9: The overall test control cost optimization.

$C_{rf}/C_{wire}>35$) which requires 4 RF nodes to be distributed chipwide and to be shared by 10 IPs.

Table 7.1 shows the results for a range of W_{max} when specifying the weights as $C_{appl}/C_{wire}/C_{rf}$ = 1/5/200. As we can see, the overall test application time reduces when increasing W_{max}, while the total TAM routing cost increases. As a result, we obtain the minimum overall cost at W_{max}=32, it is because when W_{max} is small, the test application time dominates the overall cost; when W_{max} increases further, instead, the TAM routing cost dominates.

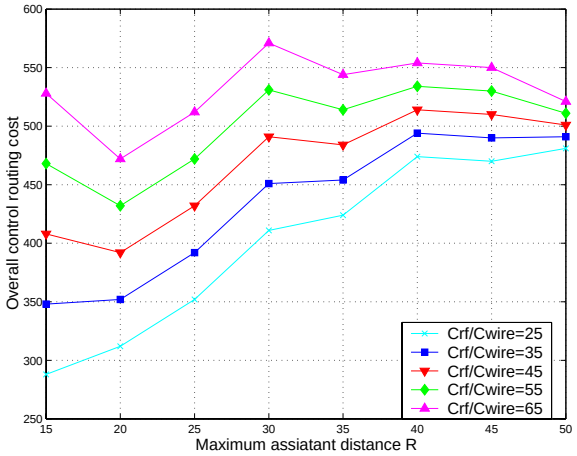


Figure 7.10: Control routing cost of SoC *d695*.

W_{max}	T_{appl}	C_{TAM}	$C_{control}$	C_{all}
16	46071	2295	1860	59406
24	33342	3430	1860	51073
32	23475	4590	1860	48285
40	19821	5785	1860	48746
48	19698	6905	1860	54223
56	16438	8020	1860	56538
64	15418	9140	1860	61118

Table 7.1: Experiment results for SoC *d695*.

7.4 Summary

One of the objectives of SoC testing is to minimize the combined cost of test application time and the associated hardware cost in resource distribution. We have proposed a multihop wireless test control network using radio-frequency nodes for control signal distribution. In this chapter, we have studied several system optimization issues such as RF nodes placement, core clustering and routing for the control constrained resource partitioning and distribution. We have presented the system optimization technique for the integration of resource distribution including not only the RF links for intra-chip communication but also the TAM routing with the minimization of overall testing

cost. In the future work, we will further address the system optimization problem and evaluate the impact of wireless control on the system testing solution.

Chapter 8

Conclusion and Future Work

This research work aims at testing of heterogeneous core-based System-on-Chips (SoCs), collaborating the state-of-the-art design and test techniques with algorithmic model development, system architecture design, and wireless applications. We have addressed not only the development of system design-for-test methodology and optimization techniques but also the application of wireless technology to on-chip test connectivity and communication for future complex SoCs. The motivation is to provide cost-effective test solutions to the rapidly evolving SoC design process. The major contributions of this dissertation are as follows:

1. We have formulated the SoC test scheduling to the single-pair shortest path problem, and presented efficient test scheduling heuristic algorithms for embedded core-based SoCs. With the flexibility of selecting a test set from a set of alternatives, we have proposed to schedule the tests for a given system in a way that balances the resource usage queue as evenly as possible, thus reducing the overall test time. Moreover, we have presented a grouping scheme and all permutation scheduling to optimize the schedule and evaluated the proposed approaches via simulation. Our simulation results have shown that there is no explicit dead time in our approach and we can further reduce the implicit dead time by proper grouping. We have also extended the algorithm to allow multiple test sets selection from a set of fault model based alternatives. We expect that the proposed approach can be properly extended for testing the mixed-signal SoCs as well.

2. We have mapped the power-constrained test scheduling problem into a graph theoretic problem and presented a novel PCTS algorithm for embedded core-based SoCs, which minimizes the overall test time by overlapping blocks of power constrained concurrent test sets. We efficiently utilized the test compatibility among the tests for a given system by obtaining a set of PCTS's from the power-constrained TCG. Furthermore, we have proposed to schedule the tests in a way that reduces the explicit dead time by applying the dynamic test partitioning technique. The simulation results showed that the PCTS approach achieved better performance than the existing comparable scheduling algorithms. Further research on scheduling is not only limited to the scheduling of the test sets provided by the core vendors, but also involves the activities to develop the DfT techniques, test controller IP and efficient test resource configuration and sharing, which will in turn result in more efficient test scheduling.

3. We have presented a novel TAM scheduling algorithm based on a graph-theoretic formulation. Given an SoC integrated with a set of cores and a set of test resources, the power consumption limitation and the top level TAM width constraint, we have constructed a set of PCTS's from a P-TCG graph to utilize the test compatibility. We have derived seed sets from the test conflict graph to facilitate efficient scheduling. Furthermore, we have handled the constrained scheduling in an unique way that adaptively assigns the cores in parallel to the TAMs with variable widths and efficiently utilizes the TAM bandwidth such that the tests in the same PCTS have their lengths close to each other. Then we have reduced the explicit dead time by the dynamic test partitioning scheme. Through simulation, we have shown that up to 40% of SoC testing time reduction can be achieved by using our approach. Further research is needed to bring forth more SoC design and test features into test scheduling, such as test resource partitioning, design for testability and design for reconfiguration, etc.

4. We have proposed a novel distributed wireless test control network using the "Radio-on-Chip" technology for future high-density, high-volume embedded SoCs. Three types of control architectures, i.e., miniature wireless LAN, multihop wireless test control network

and distributed multihop wireless test control network have been presented. The proposed architectures consist of three basic components, the test scheduler, the resource configurators, and the RF nodes which support the communication between the test scheduler and clusters of cores. Moreover, the wireless test control cost has been analyzed and the system optimization has been performed on control constrained test resource partitioning and distribution. The system optimization has been performed on RF nodes placement, the optimal number of RF nodes and core clustering under the multilevel tree structure.

5. We have addressed several challenging system optimization issues such as RF nodes placement, core clustering and routing for the control constrained resource partitioning and distribution assuming the use of multihop wireless test control network for test control distribution. We have presented the system optimization technique for the integration of resource distribution including not only the RF links for intra-chip communication but also the TAM routing with the objective of minimizing the overall testing cost. Four highly interdependent design items, i.e., wrapper configuration, control resource distribution, TAM routing, and system-level scheduling, have been specified to optimize the SoC test solution. In the future work, we will further address the system optimization problem and evaluate the impact of wireless test control on the system testing solution.

Our research provides interesting opportunities for developing methodologies, algorithms, techniques and tools for design and test of deep sub-micron (DSM) system-on-chip devices. In the future we intend not only to enhance the applicability and usefulness of the initial results, but also to extend research into related areas to find novel applications for the theoretical development. More specifically:

1. According to ITRS, test and diagnosis of SoCs is the most important one among the major hurdles to be overcome in the next decade. Core-based SoC design strategies with automated high-level synthesis is a potential means of overcoming the VLSI design complexities. High-level testability designs are essential so as to minimize test complexity without affecting area or performance. Automated generation of optimized test architecture for

core-based SoCs is also an essential means of shortening time-to-market while reducing design and test cost. Because of the strategic importance of SoC/IP design in the years ahead, this research will focus on the development of design for testability methodologies and tools to facilitate testing of complex SoCs consisting of heterogeneous IPs. We would like to develop a system view of design for testability so that problems related to testing in production, operation and maintenance phases can be dealt with in a systematic way and be solved by formal and efficient methods.

2. SoC design in the forthcoming billion-transistor era will involve the integration of numerous heterogeneous IP cores. Some of the main problems arise from non-scalable global wire delays, failure to achieve global synchronization, errors due to signal integrity issues, bandwidth limitation, and difficulties associated with wired interconnects. Meanwhile, the evolutionary changes on design and testing of billion-transistor chips will be driven by several factors, such as reusing IP cores in a plug-and-play fashion, on chip interconnections limiting the performance and energy consumption, design with the requirement of quality of service (QoS). Micronetworks based on new interconnect technology such as RF need to be brought forward for intra-chip test connectivity and data communication. Thus an SoC can be viewed as a micronetwork of embedded cores, where the communication among the cores can be done through intra-chip wireless links. More research is needed to investigate the applicability of a distributed test access architecture based on multihop wireless technology.

3. By introducing wireless test data and control communication in SoC testing strategy, one important research issue is to investigate the feasibility and the limitation of the proposed wireless test access mechanism, especially the area consumption and noise management. With the distribution of on-chip RF nodes, the area cost by utilizing wireless test network should be compatible with the saved interconnect area. In the current wired test access models, the interconnect area mainly includes the total area of a particular TAM structure in terms of dedicated DfT hardware and a hierarchy of test control architecture for parallel test processing. When testing a billion-transistor SoC, the complexity of the supporting

interconnect infrastructure would become a major challenge. Therefore, we introduce the idea of using wireless radios to transmit test data and control signals to resolve the acerbated core accessibility problem in testing GHz SoCs. The major area consumption is contributed from the tiny low-power and low-cost RF interfaces and local control mechanisms. Further, the test data transmission over wireless medium on-chip will present the real challenge in the near future. Data communication will rely on more complex hardware implementation than single tone wireless signal transmission. For the purpose of area cost estimation and evaluation, we will develop area estimation models for both the proposed wireless test network and the conventional wired test access architectures. Noise management is another critical concern. Electrical noise introduces any unwanted energy, which tends to interfere with the proper reception and reproduction of transmitted signals. The internal noise generated by the receiver itself is random and difficult to treat on an individual basis but can be described statistically. Random noise power is proportional to the bandwidth over which it is measured. The external noise is accumulated from the surrounding circuitry environment and the switching operation among various cores. We may use Gaussian noise to identify the accumulative effect of all random noise generated both external and internal to the wireless test network and average over a period of time with the consideration of all frequencies. Noise figure, i.e., the signal to noise ratio (SNR) is defined as the ratio of the desired signal power to the noise power. SNR estimation indicates the reliability of the link between the transmitter and receiver. In addition, the interference within the network of RF nodes may count on the RF transmission range and transmission frequency. Employing single frequency implementation will simplify the hardware design of RF nodes but would rely on complex MAC protocol to handle the interference. The interference can be most possiblly reduced by implementing each dedicated RF node in different transmission frequency. However, it may complicate the hardware design. The collision within the network needs to be carefully considered during the transmission of different signals to the same node, which can be avoided by a predetermined test schedule and an efficient routing protocol. We will develop simulation models according to differ-

ent scenarios and measure these effects with the use of RF interconnects for intra-chip test communication.

Appendix A

The Pseudocode of Modified SPSP Algorithm

structure CORE: id; /* core id */

num_t; /* the num of test sets */

$t[m]$; /* the test time of each test set */

$r[m]$; /* the corresponding test resources */

structure NODE: $dist[m]$; /* a vector of distance to s */

$wt[m]$; /* the weight on the incident edge */

$pred$; /* the predecessor */

max_dist; /* the max among $dist[m]$ */

Modified SPSP (int m, int n, struct CORE $core[n]$, struct NODE $vertex[mn]$)

begin

/* *initialize* $V[G]$ */

for each vertex $v \in V[1..mn]$ /* m *is the num of resources,* n *is the num of cores* */

for the distance on each resource queue $j \in r[1..m]$

$vertex[v].dist[j] = \infty$;

for the weight of the incident edge on each resource queue $j \in r[1..m]$

$vertex[v].wt[j] = vertex[v].t[j]$;

$vertex[v].pred = NIL$;

$vertex[v].max\_dist = \infty$;

/* *initialize source* s */

for the distance on each resource queue $j \in r[1..m]$

$vertex[s].dist[j] = 0$;

$vertex[s].pred = NIL$;

$vertex[s].max\_dist = 0$;

/* *initialize destination* d */

for the distance on each resource queue $j \in r[1..m]$

$vertex[d].dist[j] = \infty$;

for the weight of the incident edge on each resource queue $j \in r[1..m]$

$vertex[d].wt[j] = 0$;

$vertex[s].pred = NIL$;

$vertex[s].max\_dist = \infty$;

$S \leftarrow \{\phi\}$;

Enqueue all vertex $v \in V[G, s, d]$ into priority queue $Q[1..mn + 3]$;

while $Q \neq \phi$

/* $u \leftarrow$ *Extract-Min(Q)* */

Dequeue(u, Q, m, n); /* *remove node* u *from* Q *with minimum* max_dist *value* */

$S \leftarrow S \bigcup \{u\}$;

if (node $u == s$)

for each node $v \in Adj[s]$

for the distance on each resource queue $j \in r[1..m]$

$Q[v].dist[j] = Q[v].wt[j]$;

update $Q[v].max\_dist \leftarrow max\{Q[v].dist[j]\}$;

$Q[v].pred \leftarrow s$;

Enqueue(v, Q, m, n); /* *add node* v *into* Q */

else if (node $u == d$)

print minimum distance vector;

print path from $s \rightarrow d$;

break; /* *a shortest path from s to d is found* */

else

for each node $v \in Adj[u]$

/* *relaxation* */

for the distance on each resource queue $j \in r[1..m]$

$A[j] = Q[v].dist[j]$;

for the distance on each resource queue $j \in r[1..m]$

$B[j] = Q[u].dist[j] + Q[v].wt[j]$;

if ($getmin(A, B, m) == 1$) /* *if $A > B$ return 1; else return -1* */

for the distance on each resource queue $j \in r[1..m]$

$Q[v].dist[j] = Q[u].dist[j] + Q[v].wt[j]$;

update $Q[v].max\_dist \leftarrow max\{Q[v].dist[j]\}$;

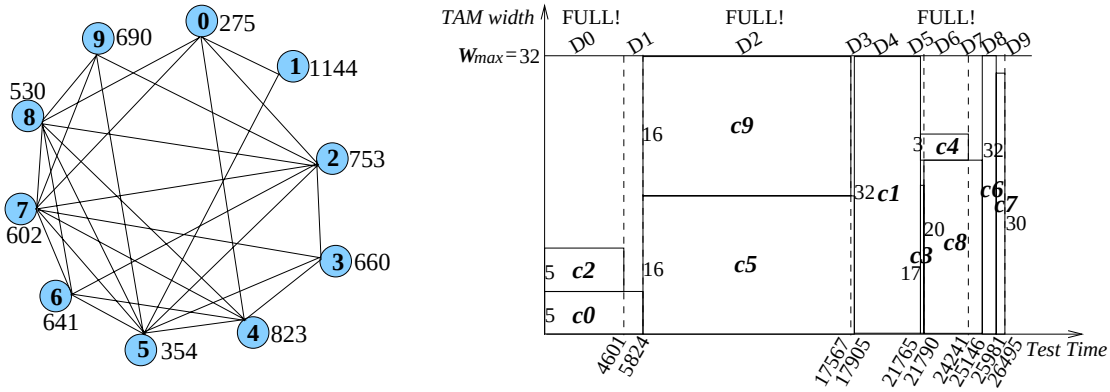
$Q[v].pred \leftarrow u$;

Enqueue(v, Q, m, n); /* *add node v into Q* */

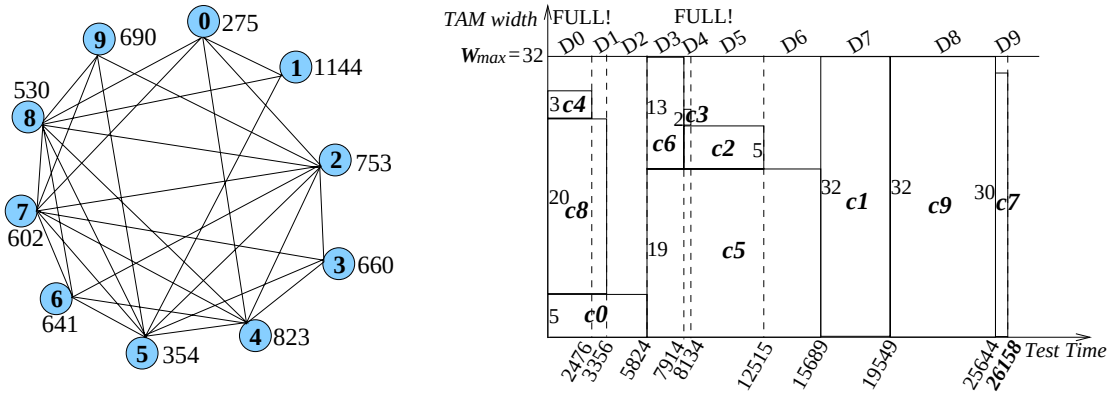
end

Appendix B

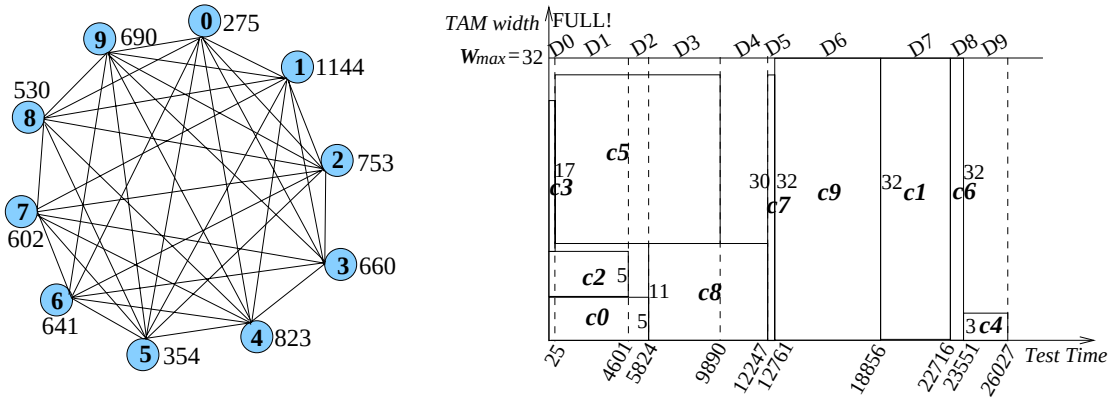
The Schedule Results of d695 with W_{max}=32



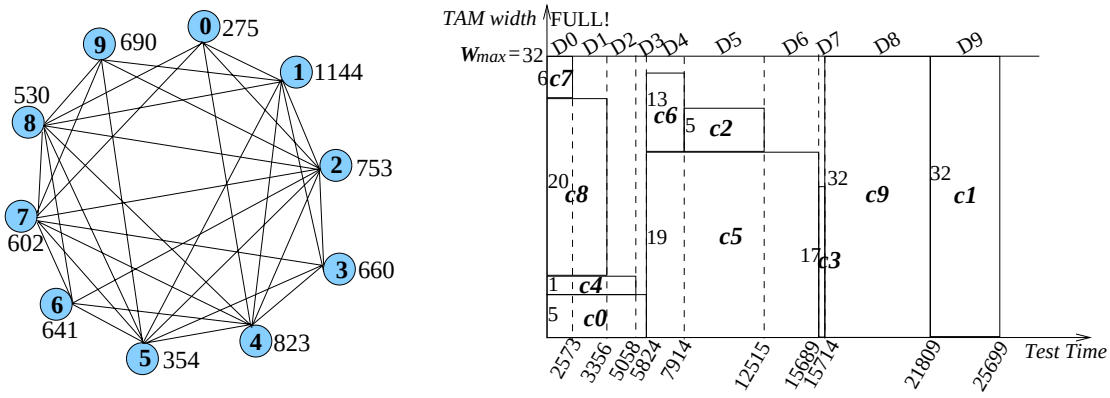
(a) The schedule with P_{max}=1500



(b) The Schedule with P_{max}=1800



(c) The schedule with P_{max}=2000



(d) The schedule with P_{max}=2500

ACKNOWLEDGMENTS FOR FUNDING PROVIDERS

This research was in part supported by a NYSTAR grant from the Microelectronics Design Center through University of Rochester.

Bibliography

[1] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," *IEEE Computer*, pp. 52–60, June 1999.

[2] J. Aerts and E. J. Marinissen, "Scan chain design for test time reduction in core-based ICs," in *Proc. of Int'l Test Conf.*, pp. 448–457, October 1998.

[3] R. Rajsuman, *System-On-A-Chip: Design and Test*. Artech House, Incorporated, 2000.

[4] 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 *The Proc. of ITC*, pp. 74–82, 2002.

[5] International Technology Roadmap for Semiconductors, 2001 Edition.

[6] A. Allan, D. Edenfeld, W. H. Joyner, A. B. Kahng, M. Rodgers, and Y. Zorian, "2001 technology roadmap for semiconductors," *IEEE Computer*, vol. 35, pp. 42–53, January 2002.

[7] P. Gallagher, V. Chickermane, S. Gregor, and T. S. Pierre, "A building block BIST methodology for SOC designs: A case study," in *Proceedings of International Test Conference*, October 2001.

[8] R. Kapur, R. Chandramouli, and T. W. Williams, "Strategies for low-cost test," *IEEE Design and Test of Computers*, vol. 18, no. 6, pp. 47–54, 2001.

[9] E. J. Marinissen and M. Lousberg, "Macro test: A liberal test approach for embedded reusable cores," in *IEEE Int'l Workshop on Testing Embedded Cores-based Systems*, pp. 1–9, November 1997.

[10] I. Ghosh, S. Dey, and N. K. Jha, “A fast and low cost testing technique for core-based system-on-chip,” in *Proc. of DAC*, pp. 542–547, 1998.

[11] V. Immaneni and S. Raman, “Direct access test scheme - design of block and core cells for embedded ASICs,” in *Proc. of ITC*, pp. 488–492, September 1990.

[12] L. Whetsel, “An IEEE 1149.1 based test access architecture for ICs with embedded cores,” in *Proc. of ITC*, pp. 69–78, 1997.

[13] N. A. Touba and B. Pouya, “Using partial isolation rings to test core-based designs,” *IEEE Designs and Test of Computers*, vol. 14, no. 4, pp. 52–59, 1997.

[14] P. Varma and S. Bhatia, “A structured test re-use methodology for core-based system chips,” in *Proc. of ITC*, pp. 294–302, 1998.

[15] 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 *Proc. of ITC*, pp. 284–293, 1998.

[16] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, “Towards a standard for embedded core test: An example,” in *Proc. of ITC*, pp. 616–627, 1999.

[17] Y. Zorian, “A distributed BIST control scheme for complex VLSI devices,” in *Proc. IEEE VLSI Test Symposium*, pp. 4–9, April 1993.

[18] M.-C. F. Chang, V. Roychowdhury, L. Zhang, H. Shin, and Y. Qian, “RF/wireless interconnect for inter- and intra-chip communications,” *Proc. of The IEEE*, vol. 89, pp. 456–466, April 2001.

[19] B. A. Floyd, C.-M. Hung, and K. K. O, “Intra-chip wireless interconnect for clock distribution implemented with integrated antennas, receivers, and transmitters,” *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 543–552, May 2002.

[20] R. K. Gupta and Y. Zorian, “Introducing core-based system design,” *IEEE Design & Test of Computers*, vol. 14, pp. 15–25, December 1997.

[21] E. J. Marinissen and Y. Zorian, “Challenges in testing core-based system ics,” *IEEE Communication magazine*, pp. 104–108, June 1999.

[22] V. Agrawal, C. Lin, P. Rutkowski, S. Wu, and Y. Zorian, “Built-in self-test for digital integrated circuits,” *AT&T Technical Journal*, vol. 73, March 1994.

[23] M. Stancic, L. Fang, and H. G. Kerkhoff, “Testing of parametric faults in embedded analog cores of system-on-chip,” in *Proc. of ProRISC*, pp. 515–518, 2000.

[24] E. J. Marinissen and J. Aerts, “Test protocol scheduling for embedded-core based system ICs,” in *2nd IEEE Int’l. Workshop on TECS*, pp. 5.3–1–9, October 1998.

[25] E. J. Marinissen, R. Kapur, and Y. Zorian, “On using IEEE P1500 SECT for test plug-n-play,” in *Proc. of ITC*, pp. 770–777, October 2000.

[26] IEEE P1500 web site. http://grouper.ieee.org/groups/1500/.

[27] L. Benini and G. D. Micheli, “Networks on chips: a new soc paradigm,” *IEEE Computer*, vol. 35, pp. 70–78, 2002.

[28] B. Floyd, “A CMOS wireless interconnect system for multigigahertz clock distribution,” 2001. PhD Dissertation, University of Florida.

[29] A. Jas and N. A. Touba, “Test vector decompression via cyclical scan chains and its application to testing core-based design,” in *Proc. of Int’l Test Conf.*, pp. 458–464, 1998.

[30] A. Chandra and K. Chakrabarty, “System-on-a-chip test data compression and decompression architectures based on golomb codes,” *IEEE Trans. on Computer-Aided Design*, vol. 20, no. 3, pp. 355–368, 2001.

[31] K. Chakrabarty, “Design of system-on-a-chip test access architectures using integer linear programming,” in *Proc. of IEEE VLSI Test Symp.*, pp. 127–134, 2000.

[32] K. Chakrabarty, “Test scheduling for core-based systems using mixed-integer linear programming,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1163–1174, October 2000.

[33] E. Larsson and Z. Peng, “Test scheduling and scan-chain division under power constraint,” in *The Proc. 10th Asian Test Symposium*, pp. 259–264, October 2001.

[34] M. Sugihara, H. Date, and H. Yasuura, “Analysis and minimization of test time in a combined BIST and external test approach,” in *Design, Automation and Test in Europe Conf.*, pp. 134–140, March 2000.

[35] 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 *The 10th Asian Test Symposium*, pp. 265 –270, October 2001.

[36] W. Jiang and B. Vinnakota, “Defect-oriented test scheduling,” *IEEE Trans. on VLSI Systems*, vol. 9, pp. 427–438, June 2001.

[37] G. L. Craig, C. R. Kime, and K. K. Saluja, “Test scheduling and control for VLSI built-in self-test,” *IEEE Trans. on Computers*, vol. 37, pp. 1099–1109, September 1998.

[38] C. R. Kime and K. K. Saluja, “Test scheduling in testable VLSI circuits,” in *Proc. of 12th Int’l. Symp. Fault-Tolerant Computing*, pp. 406–412, 1982.

[39] C.-I. H. Chen, “Graph partitioning for concurrent test scheduling in VLSI circuit,” in *Proc. of DAC*, pp. 287–290, 1991.

[40] R. Chou, K. Saluja, and V. Agrawal, “Scheduling tests for VLSI systems under power constraints,” *IEEE Trans. on VLSI Systems*, vol. 5, pp. 175–185, June 1997.

[41] V. Muresan, X. Wang, V. Muresan, and M. Vladutin, “Mixed classical scheduling algorithms and tree growing techniques in block-test scheduling under power constraints,” in *12th International Workshop on Rapid System Prototyping*, pp. 162–167, 2001.

[42] V. Iyengar and K. Chakrabarty, “Precedence-based, preemptive, and power-constrained test scheduling for system-on-a-chip,” in *19th IEEE Proceedings on VLSI Test Symp.*, pp. 368–374, 2001.

[43] M. Nourani and C. Papachristou, “An ILP formulation to optimize test access mechanism in system-on-chip testing,” in *Proc. of ITC*, pp. 902–910, October 2000.

[44] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “Test wrapper and test access mechanism co-optimization for system-on-a-chip,” in *Proc. of ITC*, pp. 1023–1032, 2001.

[45] S. Koranne and V. S. Choudhary, “Formulation of SoC test scheduling as a network transportation problem,” in *Design, Automation and Test in Europe Conf.*, p. 1125, March 2002.

[46] D. Bagchi, D. RoyChowdhury, J. Mukherjee, and S. Chattopadhyay, “A novel strategy to test core based designs,” in *Proc. of 14th Int’l Conference on VLSI Design*, pp. 122–127, January 2001.

[47] E. Larsson and Z. Peng, “An integrated system-on-chip test framework,” in *Design, Automation and Test in Europe Conf.*, pp. 138–144, March 2001.

[48] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “On using rectangle packing for wrapper/TAM co-optimization,” in *Proc. IEEE VLSI Test Symp.*, pp. 253–258, 2002.

[49] S. K. Goel and E. J. Marinissen, “Effective and efficient test architecture design for SOCs,” in *Proc. of ITC*, pp. 529–538, 2002.

[50] 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 *Proc. of IEEE/ACM Design Automation Conf.*, pp. 685–690, 2002.

[51] K.-J. Lee and C.-I. Huang, “A hierarchical test control architecture for core based design,” in *Asian Test Symp.*, pp. 248–253, December 2000.

[52] 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 *Proc. Design, Automation and Test in Europe*, pp. 486–490, March 2002.

[53] P. Guerrier and A. Greiner, “A scalable architecture for system-on-chip interconnections,” in *Proc. of the Sophia Antipolis Forum on MicroElectronics*, pp. 90–93, October 1999.

[54] K. Lahiri, G. Lakshminarayana, A. Raghunathan, and S. Dey, "Communications architecture tuners: A methodology for the design of high-performance communication architectures for system-on-chip," in *Proc. of DAC*, pp. 513–518, June 2000.

[55] D. Wingard, "Micronetwork-based integration for SOCs," in *Proc. of DAC*, pp. 673–677, June 2001.

[56] B. Moore, D. Garvey, M. Margala, and C. Backhouse, "Wireless testing techniques and circuits for deep submicron VLSI circuits," in *Proc. of the 12th International Conference on Wireless Communications*, pp. 224–235, July 2000.

[57] D. Zhao and S. Upadhyaya, "A generic resource distribution and test scheduling scheme for embedded core-based SoCs," *IEEE Transactions on Instrumentation and Measurement*, April 2004.

[58] D. Zhao and S. Upadhyaya, "A resource balancing approach to soc test scheduling," in *Proc. of IEEE International Symposium on Circuits and Systems*, May 2003.

[59] D. Zhao, S. Upadhyaya, and M. Margala, "Minimizing concurrent test time in SoCs by balancing resource usage," in *Proc. of the 12th ACM Great Lakes Symposium on VLSI*, pp. 77–82, April 2002.

[60] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[61] R. Rajsuman, "Design and test of large embedded memories: An overview," *IEEE Design and Test of Computers*, vol. 18, pp. 16–27, May-June 2001.

[62] R. Conway, W. Maxwell, and L. Miller, *Theory of Scheduling*. Addison-Wesley, 1967.

[63] D. Zhao and S. Upadhyaya, "Dynamically partitioned test scheduling for SoCs under power constraints," in *Proc. of IEEE North Atlantic Test Workshop*, pp. 72–81, May 2002.

[64] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[65] F. N. Najm, "Estimating power dissipation in VLSI circuits," in *IEEE Circutis and Devices Magazine*, 1994.

[66] F. Brglez and H. Fujiwara, "A neutral netlist of ten combinational benchmark circuits and a target simulator in fortran," in *Intl. Symp. on Circuits and Systems*, pp. 695–698, 1985.

[67] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Intl. Symp. on Circuits and Systems*, pp. 1929–1934, 1989.

[68] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," in *Proc. of Intl. Conference on Computer Aided Design*, pp. 283–289, 1998.

[69] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "K2: An estimator for peak sustainable power of VLSI circuits," in *International Symposium on Low Power Electronics and Design*, pp. 178–183, 1997.

[70] M. S. Hsiao, "Peak power estimation using genetic spot optimization for large VLSI circuits," in *Proceedings of the Design, Automation and Test in Europe Conference*, pp. 175–179, 1999.

[71] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. American Elsevier Publishing Co., 1997.

[72] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph," *Comm. ACM*, vol. 16, pp. 575–577, 1973.

[73] D. Zhao and S. Upadhyaya, "Power constrained test scheduling with dynamically varied TAM," in *Proc. of IEEE VLSI Test Symposium*, April 27-May 1 2003.

[74] G. Chartrand and O. R. Oellermann, *Applied and Algorithmic Graph Theory*. McGraw-Hill International Editions, 1993.

[75] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *Proceedings of IEEE Intl. Test Conference*, pp. 519–528, 2002.

[76] A. S. Tanenbaum, *Computer Networks*. Prentice Hall Inc., 1996.

[77] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[78] D. Zhao and S. Upadhyaya, "Adaptive test scheduling in SoCs by dynamic partitioning," in *Proc. of IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 334–342, November 2002.

[79] M. Franceschetti, M. Cook, and J. Bruck, "A geometric theorem for approximate disk covering algorithms," in *Paradise Technical Report, ETR035*, January 2001.

[80] D. Hochbaum and W. Maass, "Approximation schemes for covering and packing problems in image processing and VLSI," *Journal of the ACM*, vol. 32, no. 1, pp. 130–136, 1985.

[81] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233–235, 1979.

[82] M. Franceschetti, M. Cook, and J. Bruck, "A geometric theorem for wireless network design optimization," in *The Lee Center for Advanced Network Workshop*, October 2002.

[83] D. Zhao, S. Upadhyaya, and M. Margala, "A new distributed test control architecture with multihop wireless test connectivity and communication for gigahertz system-on-chips," in *Proc. of IEEE North Atlantic Test Workshop*, May 2003.

[84] D. Zhao, S. Upadhyaya, and M. Margala, "Control constrained resource partitioning for complex SoCs," in *Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, November 2003.