BLOCK-LEVEL TEST SCHEDULING UNDER POWER DISSIPATION CONSTRAINTS

by

Valentin Murcşan

B Eng , "Politehnica" University of Timişoara

A THESIS SUBMITTED IN FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in the School

of

Electronic Engineering

DUBLIN CITY UNIVERSITY

December 2001

APPROVAL

Name	Valentin Mureşan
Degree	Doctor of Philosophy
Title of thesis	Block-Level Test Scheduling Under Power Dissipation Constraints
Examining Committee	Dr Ronan Scaife, Dublin City University Chair
	Dr Martin Burke, Lecturer, Trinity College Dublin, External Examiner
	Dr Xiaojun Wang, Lecturer, Dublin City University, Supervisor
	Dr Sean Marlow, Senior Lecturer, Dublin City University, Internal Examiner
Date Approved	

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work

Signed _____ ID No 95159631

Date 31/07/2002

To my family and Cristina

"Because there is light at the end of the tunnel"

Acknowledgments

I think that so far in my life I have done nothing more than follow my fate The good thing about this is that fate has been generous to me It has brought in my way, as if on a "blind date", various people who have helped me in everything I have done Ironically enough, I would often forget the names of those being introduced to me, only to see them acquire later on a special resonance in my mind or in my heart

When closing an important chapter in one's life, one looks back and savours once more the moments of satisfaction, which are all the more rewarding if achieved through effort Each such moment brings to one's mind some key sequences and each sequence has its key people And when the time comes to move on, one feels his heart fill with gratitude towards all these people, and with the desire to reciprocate their generosity

In such pleasant moments one would like to thank everybody, but there is always a risk of forgetting someone This is why I would like to start by thanking all those whom I have met during my PhD studies They have all contributed, directly or indirectly, to these lines

The first to come to mind are my family Something I would like to mention here, something that I hope I will never forget, that they have kept my faith alive that one day I will go back home, to be with them

Talking about fate, it must be said that it has not worked by itself A few people have plotted with it to make the satisfaction of this final moment possible Since this final moment coincides with this thesis, the first to come to mind is my supervisor, Dr Xiaojun Wang There are many things to thank him for, but they are too numerous to mention them here All I will say is that I thank him for believing in me and for supervising me patiently up to the very last lap of this thesis I am also very grateful to Prof Dr Ing Mircea Vladutiu of the Technical University of Timisoara, for training me as a researcher Moreover, I would have never written these lines if he had not given me a certain chance, at a certain moment

At this point I should also express my gratitude for having met Cristian Ciressan, who not only has been a great friend, but also made this position in DCU available and persuaded me to take it I will never forget the talk I had with him and Doru Todinca, when they convinced me to come to Dublin

I would further like to thank the staff in DCU and the Technical University of Timisoara Special thanks to Dr Ronan Scaife, who showed patience and understanding for the language difficulties I had at the beginning, and to my "soccer coach", Dr Sean Marlow, who has always praised me for being "altruistic" in attack

My grateful thoughts also go towards all my friends, who turned these long years of PhD work into a series of pleasant moments Many thanks to Leon Costinasi and his family, for welcoming me into their house and treating me as one of their own Their warmth has often made up for my homesickness

Thanks as well to all my friends from around the N110 lab, whose different backgrounds have showed me how great and true the idea of "one world" is This is where it is most difficult to list everyone, because there have been so many of them Gabi Muntean and Prince Anandarajah are probably the ones who could probably tell the funniest stories about Val

There is one person whom I do not know how to thank She is one of the people to whom I dedicate this thesis, as being her work as well

Finally, I would be wrong not to thank God, for bringing all these wonderful people into my life, who have helped me to understand that patience will finally bear fruit

Abstract

As device technologies such as VLSI and Multichip Module (MCM) become mature, and larger and denser memory ICs are implemented for high-performance digital systems, *power dissipation* becomes a critical factor and can no longer be ignored either in normal operation of the system or under *test conditions* One of the major considerations in test scheduling is the fact that heat dissipated during test application is significantly higher than during normal operation (sometimes 100 - 200% higher) Therefore, this is one of the recent major considerations in *test scheduling* Test scheduling is strongly related to test concurrency *Test concurrency* is a design property which strongly impacts *testability* and *power dissipation* To satisfy high fault coverage goals with *reduced test application time* under certain *power dissipation constraints*, the testing of all components on the system should be performed in parallel to the greatest extent possible

Some theoretical analysis of this problem has been carried out, but only at IC level The problem was basically described as a compatible test clustering, where the compatibility among tests was given by test resource and power dissipation conflicts at the same time From an implementation point of view this problem was identified as an Non-Polynomial (NP) complete problem In this thesis, an efficient scheme for overlaying the block-tests, called the *extended tree growing technique*, is proposed together with classical scheduling algorithms to search for *power*-constrained block-*test scheduling* (PTS) profiles in a *polynomial time* Classical algorithms like list-based scheduling and distribution-graph based scheduling are employed to tackle at high level the PTS problem This approach exploits test parallelism under power constraints This is achieved by overlaying the block-test intervals of compatible subcircuits to test as many of them as possible concurrently so that the maximum accumulated power dissipation is balanced and does not exceed the given limit The test scheduling discipline assumed here is the *partitioned testing with run to completion* A *constant additive model* is employed for power dissipation analysis and estimation throughout the algorithm

Contents

List of Tables

List of Figures

List of Acronyms

ALAP	As Late As Possible
ALU	Arithmetical Logic Unit
APD	Average Power Dissipation
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BILBO	Built-In Logic-Block Observation
BIST	Built-In Self-Test
BIT	Built-in Test
BS	Boundary Scan
BTS	Block-Test Set
BUT	Block Under Test
CBILBO	Concurrent Built-In Logic-Block Observation
CDFG	Control/Data-Flow Graph
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CPUT	Central Processing Unit Time
CTS	Concurrent Test Set
CUT	Circuit Under Test
DFC	Dynamic Frequency Clocking
DFT	Design for Testability
DG	Distribution Graphs
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor

DV	Distribution Variance
ECT	Extended Compatibility Tree
ETP	Expandable Tree Path
FC	Forbidden Conflict
FDS	Force-Directed Scheduling
FPGA	Field Programmable Gate Array
FU	Functional Unit
GA	Genetic Algorithm
HC	Hard Conflict
HLLPS	High Level Low-Power Synthesis
HLS	High-Level Synthesis
HLTS	High-Level Test Synthesis
IC	Integrated Circuits
ILP	Integer Linear Programming
INSITU	In-Situ
JTAG	Joint Test Action Group
LEA	Left-Edge Algorithm
LFSR	Linear Feedback Shift Register
LRU	Least Recently Used
LS	List Scheduling
MCM	Multichip Module
MISR	Multiple Input Signature Register
MPD	Maximal Accumulated Power Dissipation
MRU	Most Recently Used
MSE	Mean Square Error
MVS	Multiple Voltage Scheduling
NOP	No Operation
NP	Non-Polynomial
ORD	Ordered
PCDG	Power-Concurrency Distribution Graphs
PCS	Power Compatible Set

PDD	Power Dissipation Dispersion
PRPG	Pseudo-Random Pattern Generator
PTS	Power-Constrained Block-Test Scheduling
PTS-DV	Distribution Variance Based PTS Approach
PTS-FDS	Force-Directed Scheduling Based PTS Algorithm
PTS-LEA	Left-Edge Algorithm Based PTS Approach
PTS-LEADV	Mixed LEA - DV Based PTS Approach
PTS-LEAFDS	Mixed LEA - FDS Based PTS Approach
PTS-LS	List Scheduling Based PTS Algorithm
PTS-LSDV	Mixed LS - DV Based PTS Approach
PTS-LSFDS	Mixed LS - FDS Based PTS Approach
PTS-LS^2	Squared List Scheduling PTS Algorithm
RAND	Random
RISC	Reduced Instruction Set Computer
RMS	Root Mean Square
RT	Register Transfer
RTL	Register Transfer Level
SA	Signature Analyzer
SAN	Simulated Annealing
SC	Soft Conflict
SD	Standard Deviation
SOC	System on Chip
SRSG	Shift-Register Sequence Generator
STUMPS	Self-Testing Using MISR and Parallel SRSG
TCG	Test Compatibility Graph
TDM	Testable Design Methodology
TIG	Test Incompatibility Graph
TL	Test Length
TPG	Test Pattern Generator
TS	Tabu Search
TSP	Travelling Sales Person
VLSI	Very Large Scale Integration
WSA	Weighted Switching Activity

Chapter 1

Introduction

1.1 Thesis Scope

1.1.1 Digital Testing

Testing is an activity which aims at finding design errors and physical faults. Design errors are introduced by designers during the product development process. The testing of design errors, sometimes called design verification, tries to find discrepancies between the design specification and its implementation. Physical faults comprise fabrication errors, fabrication defects and physical failures [ABF94].

Production testing is twofold: *test pattern generation* and *test application*. In the test pattern generation phase, test vectors are generated while in the test application phase, the test vectors are applied to the circuit under test. The test application phase is firstly dedicated to production testing, but is later launched again in field or depot testing and, therefore, it is important that it be as short as possible. The test pattern generation task can be carried out manually by the designer or automatically. Automatic Test Pattern Generation (ATPG) is either done by an ATPG tool or by built-in circuitry. The latter technique is called Built-In Self-Test (BIST).

1.1.2 Low-Power Design for Test

In general, Design for Testability (DFT) circuits operate in two modes: *normal mode* and *test mode*. The circuit's registers, when activated during test mode, can be in states that are not reachable in normal mode. As a result, state transitions

that are not possible during normal mode are possible during test mode Therefore, during testing, sequences that could lead to much larger power dissipation may be applied to the circuit compared with sequences that are applied during normal mode Even though designs might be optimized for low-power dissipation during normal mode, the test mode could still damage the chip, unless the testing was optimized for low power dissipation as well Therefore, *low-power design for test* has become a promising new topic for research and is the wider scope of this thesis

1 1 3 High-Level Test and Low-Power Synthesis

Test synthesis is a design automation technique which is motivated by the high complexity of current designs and large testing costs Test synthesis is meant to optimize a circuit for testability while keeping within reasonable limits or even improving performance, area or power characteristics Test related activities, such as test generation and test application, usually represent a relatively big share of the total design and development cost Thus, the main idea of test synthesis is to improve testability of the design during early synthesis steps, which is expected to reduce the testing costs Test synthesis can be performed at different levels but the early synthesis steps are assumed to be employed at high levels High-level test optimization can be performed in both behavioural and structural domains

The excessive heat dissipation is tied to trends such as circuit miniaturization and deep-submicron technologies and is a serious problem for portable products Very Large Scale Integration (VLSI) circuits running in test mode may consume more power than when running in normal mode [Zor93] Therefore, the heat dissipated during test application influences the test design methodology for practical circuits and has to be taken into consideration at early design stages, i e at high levels

Lately, the major concerns of VLSI designers as area, performance, cost, reliability, and power dissipation are usually optimized by High-Level Synthesis (HLS) separately However, this has begun to change and, for example, power is being optimized with respect to a design's performance or testability, and vice-versa Several factors have contributed to this trend but the driving factor has been the remarkable growth of the class of mobile computing devices and wireless communications systems which demand robust solutions, high-speed computation, and complex functionality with low power consumption [Ped96]

1 1 4 Test Parallelism vs Power Constraints

In a power conscious VLSI/System on Chip (SOC) design, under normal system operation, only a few blocks or modules are simultaneously activated, while other blocks are in the power-down mode to minimize the power dissipation Under test conditions, however, in order to test the system in the shortest possible time, it is desirable to concurrently activate as many blocks as possible provided that the power dissipation limit of the system is not exceeded A good example in the testing of a MCM is the multi-level integrated BIST strategy [ZB97] In this attractive approach for testing MCMs, as many BIST blocks as possible execute tests in parallel Under normal operation, blocks are not simultaneously activated and hence, the inactive blocks do not contribute much to power dissipation However, a concurrent execution of BIST in many blocks will result in high power dissipation which might exceed the maximum power dissipation limit Therefore, an intelligent way of scheduling these blocks to run in parallel to the highest extent possible (so that the power limit is not exceeded) is desirable

1 1 5 Power-Constrained Test Scheduling

Test application time and *power dissipation* are nowadays two very important issues that have to be taken into consideration during high-level synthesis and optimization Otherwise the test cost, test design time and the cost involved in the cooling systems for the power dissipated during test can become higher than the cost of circuit design and manufacturing without test considerations On the other hand, more and more products are designed for mobile applications These products are designed to be low-power for their normal functions, but their testing may exceed specified power limits The avoidance of this dangerous condition implies the need for test optimization and test scheduling in order to decrease the *test cost* and *test application time* having in mind low-power solutions The complexity of this problem turns out to be Non-Polynomial (NP) and thus sub-optimal search solutions have to be proposed

The components which are required to perform a test (test control logic, test buses, test pattern generators, signature analyzers, Block Under Test (BUT), and any intervening logic) are known as *test resources* and they may be shared among BUTs Each activity or ensemble of activities requiring a clock period during the

test mode and occurring in the same clock period, can be considered as a *test step*. A *block test* is the sequence of test steps that correspond to a specific part of hardware (block). The testing of a VLSI/SOC system can be viewed as the execution of a collection of block tests. The steps in a step sequence belonging to the same block test can be pipelined and steps from different block tests can be concurrently executed, if there are no resource conflicts between the steps. Two major types of test parallelism approaches have been identified in the literature thus far:

- *block-test scheduling*, which deals with *tests for blocks of logic*. These potentially consist of many test vectors and are regarded as indivisible entities for test scheduling. It deals with more abstract descriptions (from Register Transfer (RT) to system levels),
- *test pipelining*, which deals with the *test steps* that need to be applied and the resources to be utilized in a specific temporal order. It is applied at lower levels of abstraction, where the structure of the datapath is known in detail (logic or RT level),

Block tests and *test steps* have their *resource sets* used to build up their test plans. Depending on the test design methodology selected, once a *resource set* is compiled for each test t_i, then it is possible to determine whether they could run in parallel without any resource conflict. A pair of tests that cannot be run concurrently is said to be *incompatible*. Each application of time compatible tests is called a *test session*, and the time required for a test session is named *test length*. From this point of view, circuits fall, in general, into two classes: circuits in which all tests are equal in length and circuits in which the tests are unequal in length.

Test scheduling fixes the order in which tests are applied during test execution. Test schedules must be implemented by additional circuitry for test control, test pattern generation and test response evaluation. Test schedules affect test execution time, fault coverage, hardware overhead cost and power dissipation. Initially, test scheduling methods were proposed to minimize test application time with given fault coverage requirements. Later, test scheduling methods that trade off test execution time over test application hardware have been defined. More recently, the power dissipation issue within the test scheduling problem has been raised and proposed as a promising research topic [CSA97].

If $p_i(t)$ is the instantaneous power dissipation during test t_i and $p_j(t)$ is the instantaneous power dissipation during test t_j, then the power dissipation of a test session consisting of just these two tests is approximately $p_i(t) + p_j(t)$. Usually this instantaneous power is constrained so as not to exceed a power dissipation limit, P_{max}, if they were meant to be executed in the same test session. The power dissipation $p(s_j)$ for a test session s_j can be defined as: $p(s_j) = \sum_{t_i \in s_j} p_i(t)$, while the power constraint in test scheduling is defined as: $p(s_j) \leq P_{max}\ \forall j$. At high level a fixed value P_i is used to estimate the power dissipation of a test [AB86]. In figure 1.1 the power dissipation and test length parameters of a test example are given.

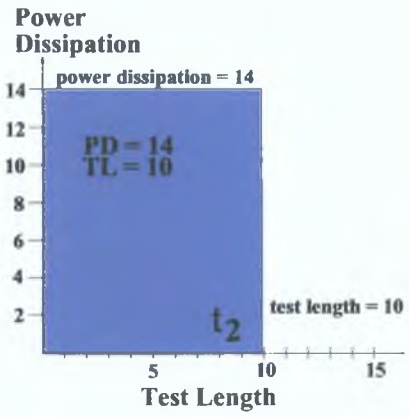
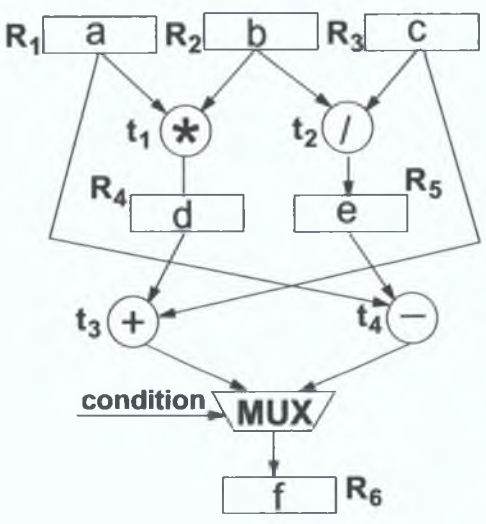
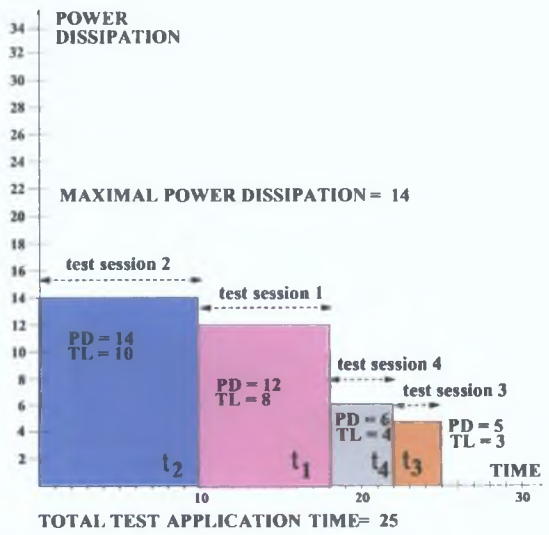


Figure 1.1: Block-Test Example



(a) Datapath Example



(b) Fully Sequential Test Schedule

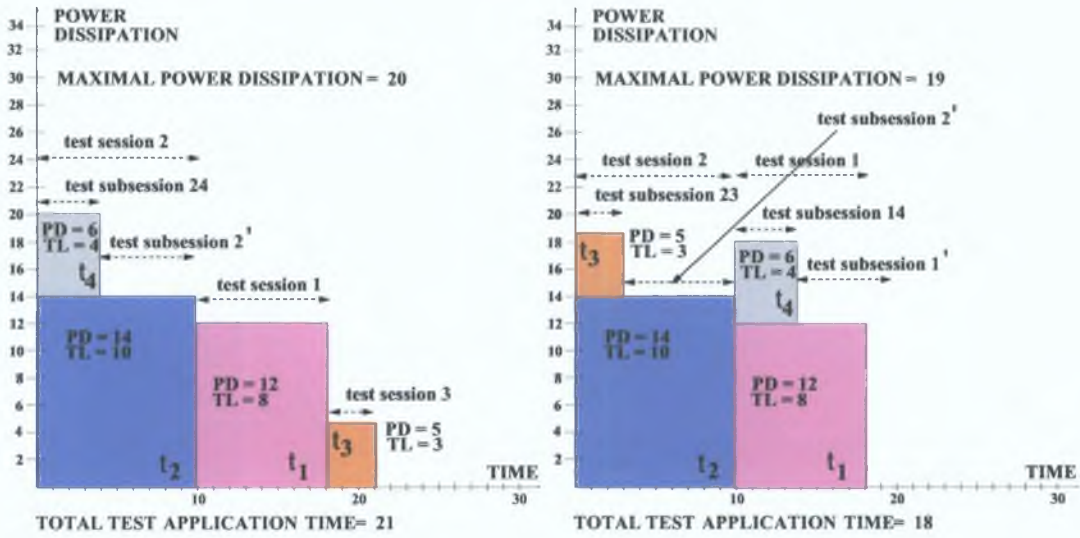
Figure 1.2: Test Scheduling Example

In order to have a better understanding of the above mentioned definitions a short example is given next. Say that the datapath from figure 1.2(a) has been synthesized from the following code: `IF condition THEN f = a * b + c ELSE f`

= a - b/c The allocated resources are registers R_1, , R_6 and a multiplexer MUX If the functional units are to be tested, then four tests have to be run t_1, t_2, t_3, and t_4 The test resource sets for each test are $t_1 = \{R_1, R_2, R_4\}$, $t_2 = \{R_2, R_3, R_5\}$, $t_3 = \{R_3, R_4, MUX, R_6\}$, and $t_4 = \{R_1, R_5, MUX, R_6\}$ In figure 1 2(b) the power dissipation and test length parameters of tests t_1, , t_4 are given By looking at their test resource sets, the test resource compatibility relationship between tests can be compiled t_1 is incompatible with any other tests because R_2 is a conflicting resource for t_1 and t_2, R_4 is a conflicting resource for t_1 and t_3, and R_1 is a conflicting resource for t_1 and t_4 In the same way t_2, t_3, and t_4 are incompatible with any other tests from the test set

Different power-test scheduling solutions are given for this short test set in figures 1 2(b), 1 3(a), and 1 3(b) Figure 1 2(b) depicts the pessimistic case when all the tests have to be sequentially executed in different test sessions because of the above given test resource incompatibility Figure 1 3(a) depicts an improved power-test scheduling solution where t_2 and t_4 can be run in parallel if R_5 is a Concurrent Built-In Logic-Block Observation (CBILBO) register and, thus, it would not be a conflicting test resource anymore The total test application time of the test schedule has decreased from 25 in figure 1 2(b) to 21 in figure 1 3(a), while the maximal power dissipation has increased from 14 to 20 A shorter test application time (18) would be obtained for a lower maximal power dissipation (19) for the power-test scheduling solution given in figure 1 3(b), if tests t_2 and t_3 were simultaneously stimulated by register R_3 as a Pseudo-Random Pattern Generator (PRPG), while tests t_1 and t_4 were simultaneously stimulated by register R_1 as a PRPG The test application time improvement is achieved by running tests t_2 and t_3 in parallel in one test session, and tests t_1 and t_4 in parallel in another test session

In order to achieve this test parallelism, different test subsessions are generated in the same test session in figures 1 3(a) and 1 3(b) For example the *test subsession*$_{24}$ and *test subsession*$_{2'}$ (gap) run inside *test session*$_2$ in figure 1 3(a) In figure 1 3(b) test subsessions called *test subsession*$_{23}$ and *test subsession*$_{14}$ are generated to run test t_3 in parallel with t_2, and test t_4 in parallel with t_1, respectively Gaps *test subsession*$_{2'}$ and *test subsession*$_{1'}$ can also be seen in figure 1 3(b) after the insertion of tests t_3 and t_4 in parallel with tests t_2 and t_1, respectively These gap test subsessions could be exploited by executing other tests compatible with t_2 and t_1 and with test length shorter than the gap's length



(a) Power-Test Schedule (b) Improved Power-Test Schedule

Figure 1.3: Power-Test Schedule Examples

1.2 Thesis Structure

The structure of the thesis is briefly described here. The second chapter gives a theoretical background for the topics dealt with in the thesis. Special attention is given to the test scheduling problem which is discussed in detail. Then, the third chapter formulates and models the Power-Constrained Block-Test Scheduling (PTS) problem. Definitions and explanations of the terms used in the thesis are given in this chapter. Chapter four describes the first set of algorithms proposed as greedy solutions in the context of the PTS problem. They are the so-called list scheduling approaches. The fifth chapter carries on with the distribution-graph based algorithms, which are more intelligent approaches. Chapter six focuses on the experimental side of the implemented algorithms. A comparison between the different proposed approaches is given as a basis for the conclusions debated in the last chapter. Chapter seven gives, based on the experiments in chapter six, a list of advantages and disadvantages of the algorithms proposed in the thesis. The disadvantages can be a starting point for further work. Other topics adjacent with the problem discussed in the thesis are also envisaged and proposed as a basis for future work.

Chapter 2

Theoretical Background

This chapter is meant to give the reader an idea of the test scheduling models, methodologies, techniques and terminology relevant to this thesis. First of all, the bigger scope is described by enumerating in the first two sections the High-Level Test Synthesis (HLTS) and High Level Low-Power Synthesis (HLLPS) techniques proposed so far in the literature. Next, the scope is shrunk by giving a few solutions for test parallelism and test application time improvement. Eventually, the topic of this thesis is given. *Test scheduling* is one of the most used techniques for test parallelism improvement and it will be described in deeper detail here along with its previous work.

2.1 High-level Test Synthesis Techniques

HLS tries to find at high-level a good trade-off among design's testability, performance, area and power consumption. High-Level Test Synthesis (HLTS) is usually carried out by DFT specific transformations together with traditional HLS methods. It uses high-level testability measures as one factor of the cost function to guide the synthesis process. There are two main trends in HLTS methodology. One of these implicitly addresses test synthesis in HLS systems and testability constraints are used together with other constraints (timing, area, power), while the objective function has a testability component. The other carries out test synthesis by test-improvement transformations based on a testability analysis scheme. These transformations can be achieved in both, behavioural and structural domains, at both algorithmic and Register Transfer Level (RTL).

Different approaches have been proposed so far both in the behavioural and structural domains, at algorithmic or RT levels A survey on HLTS was given in [WD96] The following approaches have been reported for HLTS structural test-point insertion [Gu96, DP94b] or its behavioural equivalent, test-statement insertion [CKS94] Test-register minimization is another approach achieved either by HLS techniques or by RTL transformations [Avr91, AM93, HP93, PCH91] Loop detection and elimination is another technique achievable through HLS, usually during the allocation stage I-path (unaltered data transfer path, see subsection 2 5 2) detection and utilization [AB85b, DP94a] is usually applied after scheduling, through allocation and binding Test application control enhancement is another approach and *test scheduling* is one of the known techniques (see section 2 5) to accomplish that Test session minimization is an approach related to test scheduling being employed as one of the above test control enhancement techniques By partitioning for testability [GKP95, Gu96] the test synthesis can be optimized as well Input/output variable spreading over as many registers as possible [LWJA92] is another HLTS solution Test behaviour addition [PCH91] is a promising new approach Two structural techniques have been so extensively used in literature that they have become mature The full-scan approach of chaining all the test registers and the partial-scan approach of selecting an optimum set of test registers so that the wanted testability/area-overhead ratio is achieved [MTOM96, GB95]

Test path insertion is necessary in order to observe a component, and minimization of conflicts between tests is important as it directly affects test concurrency, and therefore test time These two criteria are considered at each stage of behavioural and structural synthesis in [OH97] The behavioural and structural test synthesis problem was decomposed in [OH97] into the following five components as shown in figure 2 1 *Dataflow Scheduling for Testability* where HLS scheduling defines interconnect to test registers by placing scheduling cuts between operations while testability

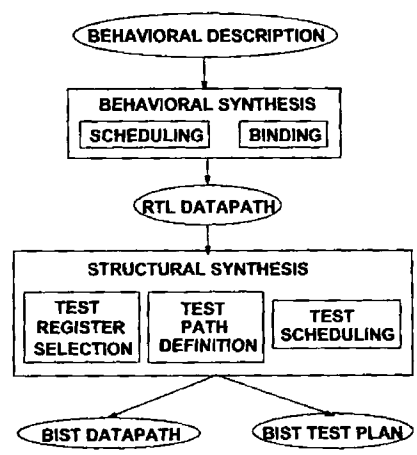


Figure 2 1 Test Synthesis Approach

is taken into consideration *Binding for Testability* which is performed to facilitate easy accessibility of chip components from registers and I/O pins by defining RTL interconnect *Test Register Selection* where a subset of datapath registers used to test the datapath modules is selected for use as test registers *Test Path Definition* stage generates the paths through which test data propagate in test mode are defined by determining the multiplexor configurations during testing Finally, the *test Scheduling* stage schedules each test operation (test step for test pipelining or block-test for block-test scheduling) to a particular test control step in order to generate a test plan to run on the test scheme of the hardware structure The hardware structure can be an RTL structure for test pipelining (see subsection 2 5 3) or any level in the test hierarchy for block-test scheduling (see subsection 2 5 4)

The approach in [OH97] employs a HLS solution to generate RTL circuits which guarantee concurrent controllability and observability of all hardware components from test registers Thus, the difficulty of test scheduling is shifted to higher levels where the amount of information to be dealt with is sensibly smaller, while testability metrics are used to estimate the impact of HLS design decisions on the test concurrency problem The goal was to limit the test scheduling problem to a structural solutions like pipeline or non-pipeline BIST microarchitectures (see figure 2 10 in section 2 5 3) A slightly different approach was theoretically formulated in [BKH97], where the idea was to evaluate the RTL test scheduling solutions, to identify the bottleneck spots in the RTL datapath and backannotate them to a transformational HLS system in order to repeat the synthesis process to get rid of possible test scheduling bottlenecks Thus, the first approach [OH97] synthesizes based on estimations, while the second approach [BKH97] is based on more precise evaluations, but may have a longer synthesis cycle

Among the HLTS techniques enumerated above, the test scheduling techniques are studied in this thesis Test scheduling deals with the optimization of test session sequences in order to achieve a short test application time, a simplified test control or an optimum power dissipation A test session can be based on any means of generating and applying a test The BIST methodology is the most widely used ATPG solution, but it is not the only one dealing with the test session concept [MKRT95] Test paths are sought to carry out the test on a piece of hardware to be tested during a test session Test paths usually share some hardware (registers, Arithmetical Logic Unit (ALU)s, multiplexors, buses), thus creating conflicts and

forcing the need for multiple test sessions The above HLTS methodologies have been proposed to synthesize data paths to avoid the test conflicts between the test sessions Unfortunately, this is not an ultimate and sufficient solution for overall design optimization Total test application time and power dissipation throughout it can be optimized by test scheduling techniques

2.2 High-level Low Power Synthesis Techniques

Lately, plenty of solutions for power-efficient synthesis have been reported in the literature [Ped96, MK96] The range of solutions also span the behavioural and structural domains between the algorithmic and RT levels A survey of these techniques can be found in [Ped96] Among the most widely used low-power design techniques the following can be enumerated low-power module selection, power-conscious storage allocation and data mapping solutions, low-power module sharing, parallel operators on time-critical paths, pipelining, algebraic transformations for low power, number representation selection, operator shutdown, low-supply voltage

The above techniques have recently been used at high-level for low-power design However, they are beyond the scope of this thesis The approach adopted in this thesis will require from the low-power design results only the high-level power estimation values for the test sessions to be scheduled A more detailed explanation of this approach will be given in chapter 3

2.3 Test Methodology and Terminology

Two basic test philosophies have been proposed so far *functional test* and *structural test* Functional testing attempts to exercise a chip as it would be used during the execution of its normal functionality On the other hand, structural testing uses a chip logic model with a fault model to hypothesize the behaviour of defective circuits, which enables the creation of automatically-generated test patterns [WNML94] Judging by the way testing is applied, testing techniques can be classified into two categories *online testing* and *off-line testing* Online testing is run during the *normal* functional operating conditions [AAMH98] Two different modes of online testing have been defined *Concurrent online testing* takes place simultaneously with normal system operation *Nonconcurrent online testing* takes

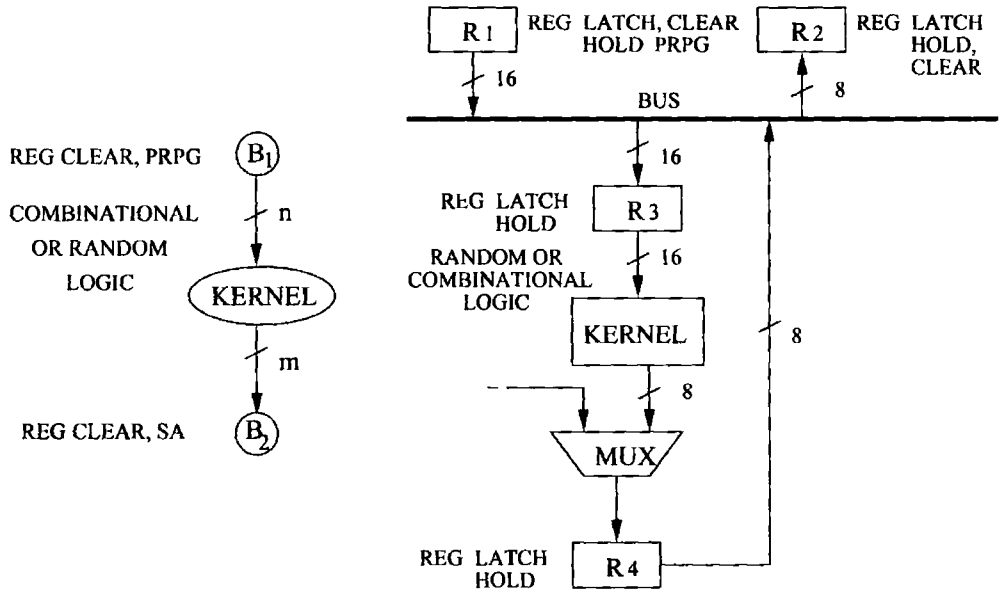
place while normal operation is temporarily suspended (idle state) On the other hand off-line testing deals with testing a system when it is not carrying out its normal functions and it is applicable at the manufacturing, field or depot testing *Functional off-line testing* deals with the execution of a test based on a functional description of the Circuit Under Test (CUT) *Structural off-line testing* deals with the execution of a test based on the structure of the CUT and an explicit structural fault model may be used

In this thesis, test scheduling approaches are proposed at high-level and do not constrain the type of testing which has to be employed for the tests to be scheduled The test scheduling here is supposed to be applied to tests running in the test mode Only test sessions are supposed to be scheduled by this approach Therefore, throughout this thesis, an off-line testing approach is employed as test methodology for the designs the actual test scheduling approach is meant for

Addressing the test parallelism problem assumes that a Testable Design Methodology (TDM) [AB86] is adopted beforehand A TDM deals with the complete process of adopting one or more different testing architectures mapped on different blocks of the design It also has to avail of an easily testable structure optimized by HLTS The known ways of testing the structure up to date are using external and/or built-in test hardware

The lowest level considered in this thesis for test scheduling application is RTL Examples of well-known TDMs at RTL are scan path, Built-In Logic-Block Observation (BILBO), syndrome testing, and autonomous testing The structural aspects of a TDM can be best modelled by a template like the one given in [AB86] and depicted in figure 2 2(a) A TDM template conveys information about the type of structure, referred to as kernel, to which the TDM is applicable The template also indicates the required Built-in Test (BIT) structures and the connection paths which must exist between them and the kernel As an example a methodology for BILBO is given in figure 2 2 In addition to the structural aspect of a TDM, there is also an operational one A *test scheme* together with a *test plan* specify how a test methodology is to be executed [AB86] The main aspects specified by a test scheme are the generation of test vectors, the transfer of the test vectors to the structure (kernel) to be tested, the propagation of the test vectors through the kernel, the transfer of response data to some response analysis circuit, and the processing of the response data The test plan is a stepwise activation of the operations enumerated

m a test scheme



(a) Test Template (b) Structure Under Test

Figure 2 2 Testable Design Methodology

Test schemes and test plans consist of three sections a head, a body, and a tail The body of a test scheme describes the on-chip actions that constitute the life cycle of a single test vector from the time it is generated until its effects on the kernel have been captured and processed In addition to the actions in the body of a scheme which must be executed once for every test vector, a test scheme often has a head (tail) section in which initialization (closing) actions are specified

Two schemes to apply the tests were defined in [AS98] *test per clock* scheme (e g , BILBO, circular BIST [EW91]) and *test per scan* scheme (scan-based test like Self-Testing Using MISR and Parallel SRSG (STUMPS) [SM97]) In a test-per-scan scheme, all or part of the registers are set-up as test registers Thus, in test mode they form a chain called a scan path The test registers are fed with test vectors by means of a scan path When each test register has received its test vector, the whole circuit is evaluated during one clock when the circuit is running in the normal mode During this clock, test vectors are applied to the logic and the response is captured in the same test registers Then the response is shifted out by means of the same scan path, while new test vectors could be shifted in at the same time

On the other hand, in a test-per-clock scheme, some test registers are enhanced such that in special test modes they autonomously generate patterns or compact

test responses The most widely used TDM in this scheme is BIST A test-per-clock scheme has advantages with respect to test application time, but it often requires a higher hardware overhead than the test-per-scan scheme For example, a CBILBO register can simultaneously generate test patterns and compact test responses, but it contains two registers instead of one As a result, test scheduling is simplified, while test application time is reduced, but the test registers have doubled at least

A test plan has the same general form as a test scheme but it is usually rendered by pipeline functionality A test plan is the sequence of actions that have to be run in order to implement a given test scheme These actions can be naturally organized into a number of steps, such that the actions in one step can all be executed in one clock period Each activity or the ensemble of activities requiring a clock period and occurring in the same clock period can be considered as a *test step* which is considered the basic element of test pipeline modelling [SK92] The test of hardware consists of the repetitive execution of the same sequences of test steps with different data values and through different hardware stages (resources) The execution starts at one or more test sources and terminates at one or more test sinks A *test function* is the sequence of test steps that correspond to a specific part of hardware (a sequence of hardware stages) and the control configuration established during the execution of one iteration of a test

Testing of a VLSI system can be viewed as the execution of a collection of test functions In order to increase their parallelism and, thus, decrease the test time, three approaches can be considered Firstly, an overlap in the *time domain* available via pipelining, which permits concurrent execution of test functions Secondly, different test functions can also be concurrently executed in the *space domain* due to parallel functional units, i e , if their execution utilizes separate hardware structures Finally, further reduction in testing time can be achieved if different test functions can be scheduled to utilize shared hardware at different times during their execution

2.4 Test Parallelism and Test Time Reduction

Approaches to reduce the test application time by restructuring the test sequence can be divided into the following two classes Firstly, the static test sequence reduction approaches which do not increase the complexity of test generation They

put together the tests generated by a test generator in such an order that the overall application time is reduced Basically, there are two approaches rearranging the test strings [Diw91, FM91] or reordering the flip-flops (registers) [GB91, MM91] Secondly, the dynamic test sequence reduction approaches that try to reduce the number of test vectors by carefully assigning the unspecified input signal values to binary constants and thus reduce the number of test vectors [PS92, LS92] There are also other ATPG techniques to reduce the test application time by optimizing the test vector sets, but they are beyond the scope of this thesis

2.4 1 Single Scan Path Reconfiguration

For self-testable (e g , BIST) circuits the first test (sub)session lasts until the subcircuit with the smallest test length has been completely tested Afterwards, test patterns are applied only to the remaining subcircuits [NNB92, GB91] proposed techniques to order the registers included in a single scan path such that the total test application time is minimized

A reconfiguration approach was given in [NB95, NB93] for single scan paths in order to minimize the shifting time m applying test patterns on a device The main idea was to employ multiplexers to bypass registers that are not accessed frequently in the test process and hence reduce the overall test application time

In [LKL93] the reduction of test application time for the general scan designed circuits was studied Given a scan path, the test application time can be reduced by exploiting and eliminating unnecessary scan operations The reduction problem was investigated from three aspects test generation, selective scan, and rearrangement of scan path A two phase test generation strategy was proposed the scanless phase for easy-to-detect faults and the scan phase for hard-to-detect faults Selective scan methods were then proposed because it is not always necessary to control and observe every scanned element in every test vector application Finally, a technique for the rearrangement of scan path was proposed based on a scan elements ordering heuristic

2 4 2 Multiple Scan Path Reconfiguration

A more comprehensive and integrated method was proposed in [OBT91] to reduce the test application time by reducing the number of test vectors and shift-operations

Firstly, a scan path organization was carried out by ordering the scan registers in a scan path and by distributing the scan registers in multiple scan paths Then a scan test scheduling was done by overlaying shift operations of different tests and by parallel scheduling of different tests

Multiple-scan chain restructuring techniques were also described in [LCH98] It was noticed there that multiple scan chain architectures require a large pin overhead and they are not supported by boundary scan The solution was to allow a single input line to support multiple scan chains In [NGB92, NGB93] a different length multiple scan chain approach was proposed in order to assign those scan elements that are more frequently used to the shortest scan chains

The STUMPS architecture is another solution to the pin overhead problem In [KW98] this idea was used in order to synthesize a pattern generator to simultaneously stimulate all scan chains and decrease test application time

2 4 3 Test Structure Insertion and Optimization

There are many attempts to reduce the area overhead at the same time with the optimization of test scheduling by sharing the hardware elements Unfortunately, reduction of area overhead and testing time turn out to be conflicting objectives In [KTH88] the authors addressed the issue of BILBO minimization using the minimal set cover technique The area optimization, thus obtained, is followed by the determination of a test schedule using the graph colouring approach in [KS82] In [BWBM92] the same problem was formulated as an Integer Linear Programming (ILP) as well as a graph search problem with a heuristic cost function

A systematic approach has been developed in [LNB91] to provide designers with a set of testable versions for a given design, ranging from minimal test time solution to the minimal area overhead solution An expert selection system is employed to operate as an intelligent BIST design advisor

A large amount of work was carried out by Stroele and Wunderlich in trading-off the test application time and test area overhead by test register insertion, configuration and test scheduling A test structure insertion and optimization technique is initially given in [SW95] for the test-per-clock scheme, emphasizing the advantages of CBILBO methodology A unified method for assembling all the single tests to a global schedule is then presented in [SW94] In order to reduce the BIST hardware overhead, the number of evaluated signatures was minimized there It was also

observed in [SW86] that test schedules have an impact on the number of signatures that have to be evaluated These last two approaches utilize the mutual influence of signatures to construct test schedules such that only the signatures at the primary outputs must be evaluated The relationship between the structure of the test schedules and the hardware costs is then investigated in [Str92] and the knowledge about it is used to guide the search for an optimal schedule from both, time and cost point of view Based on the same concepts a gate-level to RTL test scheduling approach is given in [SW92] Here, 1-bit test cells are inserted at gate level, and initial test schedules are constructed Based on the information in these schedules, test cells that can be controlled in the same way are assembled to test registers Finally, a test schedule at RTL is constructed, to reduce both BIST hardware and test application time, while a minimal set of test control signals is determined

2.5 Test Scheduling

2 5 1 Test Scheduling Goal

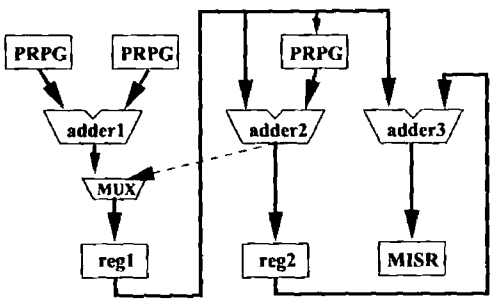


Figure 2 3 BIST Datapath Example

Test concurrency is a design property which strongly impacts on testability To satisfy high fault coverage goals with reduced test application time, the testing of all components on the chip should be performed in parallel to the greatest extent possible The structure of the circuit (number and characteristics of test paths), however, may not permit concurrent observation of all components For example, two components whose results are observed by the same test register cannot be concurrently observed In figure 2 3 ([OH97]) the test path is depicted in solid line, while the *MUX* multiplexer could be a possible testing bottleneck if both adders, $adder_1$ and $adder_2$ try to set up test paths through the multiplexer The test concurrency problem is magnified when DFT area overhead is an issue, because removing test registers from a datapath increases the number of components which must be observed by each remaining test register Therefore, HLS is used to generate datapaths whose components are concurrently testable to

the highest extent possible [OH97] Then test schemes or plans must be defined to execute the concurrent testing of each component However, even if the above solution is employed to achieve a high test concurrency, it cannot be a 100% success

In Very Large Scale Integration (VLSI) circuits, a large device count exists with a relatively few input/output pins Moreover, BIST, which is the most used testing technique, is minimized as much as possible to keep under control its area overhead This results in complex structures for which test generation is difficult, consisting of long tests with high input/output traffic during testing In the VLSI environment, desirable goals for BIST are to eliminate as much test generation as possible, to permit a fairly general class of failure modes, to permit easy circuit initialization and observation, to reduce input/output pin signal traffic, to *reduce test length*

Although BIST techniques manage to achieve a number of the goals listed above, for very large circuits with extensive BIST resources, the testing time can still be quite long if the tests for the various parts of the circuits are executed sequentially In such cases, in order to reduce testing time and fully exploit the power of the BIST resources, it is essential to control the testing process so that full use is made of the potential parallelism available

Typically, physical paths must be established from the test pattern generators to the inputs of the BUT and from the BUT to the response compressors/analyzers The blocks which are required to perform a test (test control logic, Test Pattern Generator (TPG), compressors/analyzers, BUT, and any intervening logic) are known as *test resources* Test resources may be shared among BUTs For example, testing schemes exist, like CBILBO, in which the Signature Analyzer (SA) for one BUT can be used as an input stimulus (TPG), for another BUT Also, for those blocks which lie on the periphery of a chip, a portion of the test resources may he off-chip

Two major types of test parallelism approaches have been identified in the literature thus far One approach, named *block test scheduling* (macro-test scheduling or block-level test scheduling) [CKS88, KS82], deals with tests for blocks of logic, which potentially consist of many test vectors and are regarded as indivisible entities for test scheduling Furthermore, there is no temporal relationship between the test vectors in different block tests other than that defined by the usually rare conflicting use of resources The other approach [AB85a, AB86], called *test pipelining* (micro-test scheduling or test-step level scheduling), deals with *test steps* which have to be applied and resources utilized in a specific temporal order Intuitively, test-step

level scheduling is applied at lower levels of abstraction, where the structure of the datapath is known in detail (logic or RT level) while block-level test scheduling deals with higher levels, namely a more abstract description (RT or system levels) Most of the approaches formulated so far for test pipelining were given in the structural domain at lower levels (mainly logic level) Lately, though, a trend to behavioural and register transfer level approaches [CSA97, BKH97, OH97] has been noticed In the case of block test scheduling a system level structure is taken into consideration, where the blocks represent processors (datapaths and controllers), memories and buses

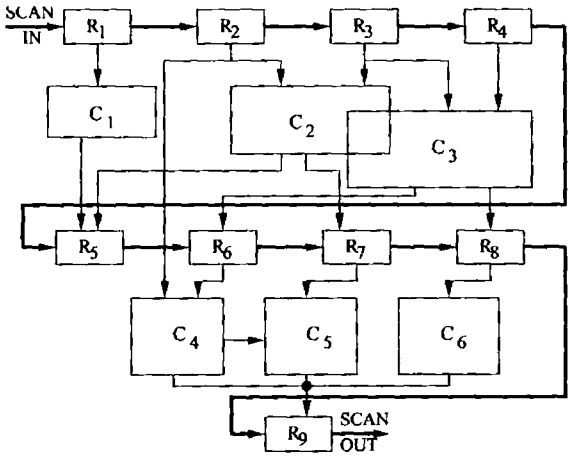


Figure 2 4 Circuit Under Test Example

To summarize, an efficient *test scheduling model* has to provide sufficient information about the contention over test resources and to exploit the deterministic and repetitiveness characteristics of testing All information required can be summarized by the following generic set of components for each test function the number of tests to be applied, the accessibility requirements in terms of test and/or system resources, and additionally, in the case of test pipelining, the execution picture for one iteration of the test step sequence

2 5 2 Test Scheduling Model

The steps in a test step sequence (test plan) belonging to a test function can be pipelined and, steps from different test functions (test blocks) can be concurrently executed if there are no resource conflicts In both, *block test scheduling* and *test pipelining* approaches, *block-tests* and *test steps*, respectively, have their *resource sets* used to build up test schemes or paths Depending on the TDM selected, once a *resource set* is compiled for each test t_i, then it is possible to define a *compatibility relation* between tests [CY92]

In a resource allocation graph, a resource node that is connected to more than one test indicates possible contention between the tests for use of that resource

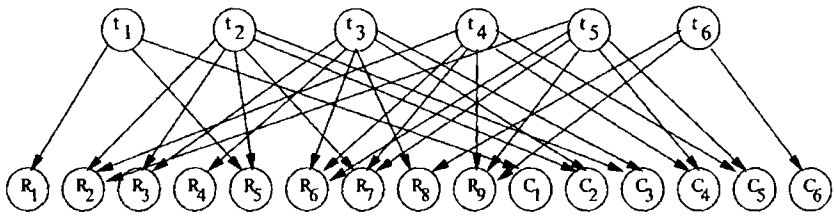


Figure 2 5 Resource Allocation Graph

For the design example taken from [CY92] and depicted in figure 2 4 the resource allocation graph is depicted in figure 2 5 It consists of combinational logic denoted by C_i and registers denoted by R_i A BILBO TDM is assumed and a serial scan path is available for initializing TPGs and for observing the resulting signatures in SAs The use of the resources in this structure can be illustrated, for example, by test t_4 of C_4, which uses as inputs the registers R_2 and R_6 and the register R_9 as output

Since the test path for C_4 has an output branch through C_5, there is in the resource allocation graph an edge between t_4 and C_5 as well Thus the resource set for test t_4 is R_2, R_6, R_9, C_4, C_5, and R_7 because of C_5 The resource set for test t_5 is R_2, R_6, C_4, R_7, C_5 and R_9 Resources R_2 and R_6 are input resources to C_4, which is one of the input resources for C_5 during test t_5 Furthermore, during the test t_2 of C_2, one of the resources is C_3 which overlaps C_2 C_2 is also in the resource set of test t_3

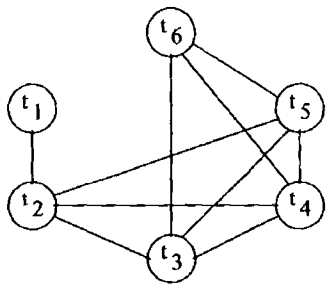


Figure 2 6 Test Incompatibility Graph

A pair of tests that cannot be run concurrently, will be said to be incompatible Otherwise, they are compatible Each application of time compatible tests is called a *test session*, and the time required for a test session is referred to as *test length* Such relations can be represented by Test Compatibility Graph (TCG) or by the complementary Test Incompatibility Graph (TIG) (figure 2 6) In a Test Compatibility Graph (TCG) an edge is drawn between node t_i and node t_j if test t_i and test t_j are compatible A TCG or TIG can be used as a basis for scheduling the tests so that the total testing time is minimized In general, circuits fall into two classes circuits in which all tests are equal in length

and circuits in which the tests are unequal in length Based on this classification, the following two problems may be stated Firstly, to find a test schedule such that the total time to run all tests is minimum provided that each test t_i takes T units of time to run completely Secondly, to find a test schedule so that the total time to run all tests is minimum provided that a test t_i takes T_i units of time to run completely

A *clique* is a maximal subgraph of a TCG and represents a maximal set of tests which can be run concurrently When power consumption is also considered in the scheduling of tests, the clique solution is not sufficient The nodes, or equivalently the tests, in the same clique are time compatible only with respect to the resource constraints They may not be compatible from the power consumption point of view as executing all tests in the same clique might exceed the maximum power limit imposed by the technology In such a case, they must not be scheduled in the same test session

Tests t_1, t_2, , t_p can run concurrently if and only if they form a complete subgraph G_i in the TCG [CKS88] A Concurrent Test Set (CTS) is a set of tests which may be run concurrently improving the test parallelism Intuitively, in order to obtain an optimal schedule, the number of CTSs should be as small as possible The order in which the CTSs associated with different cliques G_i are run is not important from testing point of view, but might be important from power dissipation point of view (see section 3 2 2) For tests which appear in more than one CTS, it is possible to delete all but one occurrence of such tests The decision to eliminate redundant executions of a particular test is usually dependent on the test control implementation It is possible to exploit these duplicate test executions to reduce aliasing in Signature Analyzers (SAs) [CKS88] For each execution of a test, a different configuration of the SA corresponding to a different polynomial can be used This has been shown to reduce the probability of aliasing [WI95]

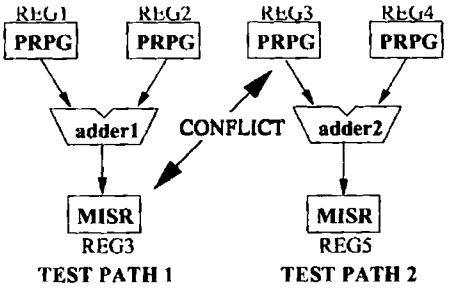


Figure 2 7 Hard Conflict Example

A higher-level approach can be found in [LNB91, AB85b, HO94] where an I-path based test methodology was adopted A K-port was defined as an input/output port of a kernel An I-path was defined here as a data path from a primary input

or a register to a K-port or from a K-port to a primary output or a register so that data can be transferred unaltered. A *test session* was defined as the execution of a set of tests for some kernels. For each test session a *test plan* is required. This plan specifies how to initialize test hardware, perform tests for kernels, and observe final signatures. A K-port is said to be *testable* if it is covered by at least one I-path. A kernel is said to be testable if each of its K-ports is testable and no conflicts exist between the I-paths of its K-ports. Two I-paths are said to have a Forbidden Conflict (FC) if they cannot co-exist in a testable design. Two I-paths are said to have a Hard Conflict (HC) if they cannot be operated in the same test session. Two I-paths are said to have a Soft Conflict (SC) if they can be operated in the same test session but restrictions on their schedule exist.

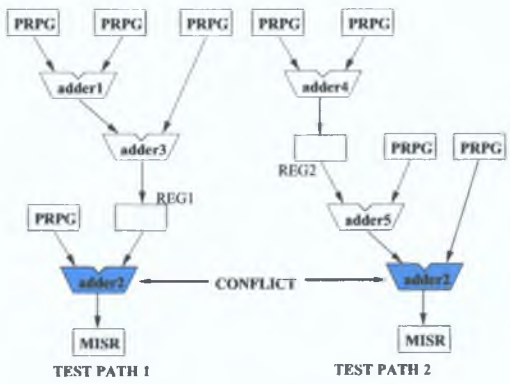


Figure 2.8: Soft Conflict Example

In [OH97] HC occurs when one test path uses a register as an Multiple Input Signature Register (MISR), while another test path uses the same register as a PRPG (see figure 2.7). The FCs are considered HCs in [OH97] and are no longer forbidden by using the CBILBO registers. Then HCs between test paths are structurally eliminated by using either a BILBO or CBILBO register, which acts as both Linear Feedback Shift Register (LFSR) and MISR. In the same way a SC appears when two test paths share intermediate registers, multiplexors, buses, or functional units at the same control step (see figure 2.8). SCs are avoided by using test scheduling to exclude the concurrent execution of the hardware-sharing operations. The use of test scheduling to avoid SCs results in reduced testing throughput which can be alleviated through pipelined testing. Partial-intrusion BIST given in [OH97] is such a test pipelining technique and is described in the last paragraph of subsection 2.5.3.

2.5.3 Test Pipelining

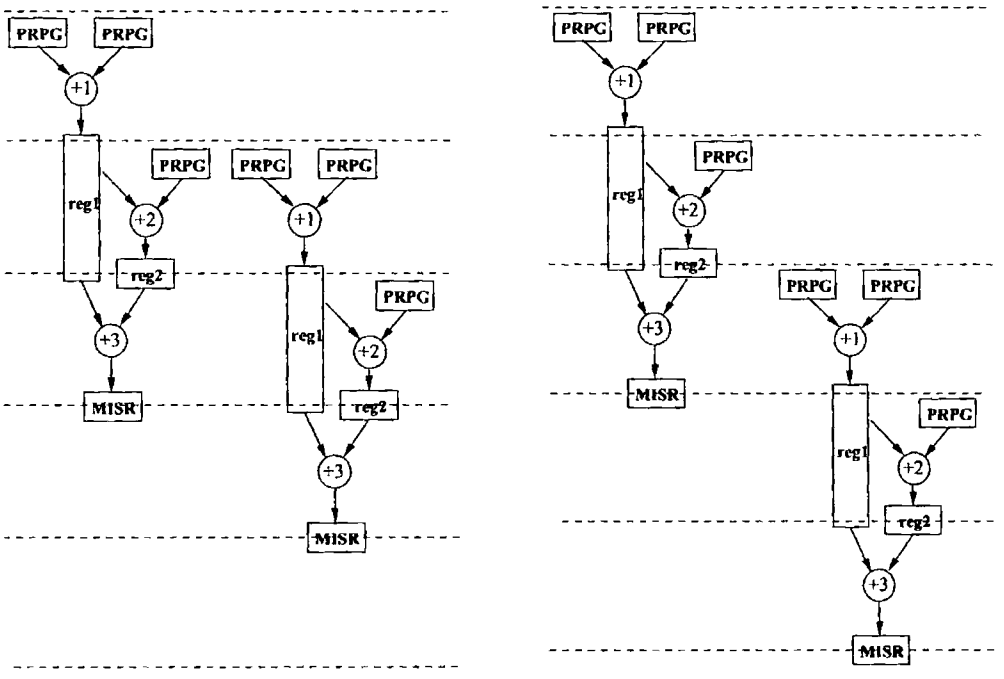
In most cases, the processing of test vectors according to the test plan can be pipelined through a circuit, thus reducing the total test time by increasing the

throughput The manner of overlapping the processing of test vectors leads to the concept of *pipelined test scheduling* or *test pipelining* There are two approaches known in the literature for test pipelining The first one is to schedule test steps for already pipelined hardware architectures The second one is to pipeline test steps for non-pipelined architectures, such that no test resource conflicts are met

Scheduling tests for parallel and pipelined units differs from traditional multiprocessing and multipipeline scheduling in the following ways Firstly, the data dependencies between consecutive executions of test functions are nonexistent in test pipeline scheduling For example, there is no data dependency between test vectors Therefore, there is no concern with regard to precedence and order of execution Secondly, the application of test functions is very repetitive, focusing on scheduling repetitive actions instead of transient actions Thirdly, a test function must be executed on the specific hardware block to be tested, whereas in traditional pipeline scheduling the pipelined operations can be executed for different resource allocation solutions Therefore, allocation to parallel resources in test pipelining is quite constrained However, if some resource allocation is required, it is assumed to have been done prior to scheduling Finally, traditional static pipeline reservation tables and the associated state diagrams used to allocate hardware units for single pipelines become impractical when dealing with multiple pipeline configurations The conclusion of these differences is that traditional pipeline scheduling approaches are not suitable for test pipeline scheduling As a consequence, approaches specifically for test pipeline scheduling have appeared scheduling tests for already pipelined units [SK88, SK89, SK92] and test step pipelining [AB85a, AB86]

The approach proposed in [SK92] generates a test schedule for testing already pipelined structures In order to achieve a minimum overall testing time for multiple test functions with multiple test steps, while retaining simple test control, the idea was to partition the test functions into sets The test functions' steps in a given set were repetitively issued with different test values until all test data had been applied A conflict occurred in the test schedule when multiple initiations of the same or different test functions attempted to simultaneously utilize any one resource, i e , in the same clock period

In [AB85a, AB86] test step pipelining techniques were given that focused on generating test functions with improved throughput The throughput improvement was attempted by inserting idle steps (No Operations NOPs) into a test function



(a) Higher Throughput (b) Lower Throughput

Figure 2 9 Pipelined BIST Test Schedules

Minimizing the overall test time was achieved however at logic level because the test function application was rather detailed

Because pipelined testing enables multiple iterations of a test path to be executed in parallel, it is possible for a conflict to occur when two different iterations of the same test path share hardware at the same control step However, a conflict is not caused when two iterations share the use of a non-test register, as long as the flow of test data through the physical hardware components is not altered For example, the test schedule for the BIST structure in figure 2 3 is given in figure 2 9 and it depicts the temporal sequence of test data propagation From figure 2 9 it can be seen that, since the two register storage operations are actually identical operations in two different iterations of the same test path, the flow of test data through the physical hardware components remains the same in both schedules [OH97] demonstrated that a conflict is not caused when two iterations share the use of a non-test register, as long as the flow of test data through the physical hardware components is not altered That is, the test vector in the aforementioned register has to be constant This is the case in figure 2 9(a) where, in order to have a higher test throughput, reg_1 must have the same test vector value throughout the high

throughput test In this case, because the value of reg_1 does not change, the two *PRPG* registers to feed its inputs are useless (their content has to be constant), while the adder +1 is not tested in this mode (only adders +2 and +3 are tested) In order to also test adder +1, a lower throughput mode has to be employed (see figure 2 9(b)), in which reg_1 is used as an un-conflicting test resource Thus, its content can be changed at each step permitting for the test of adder +1

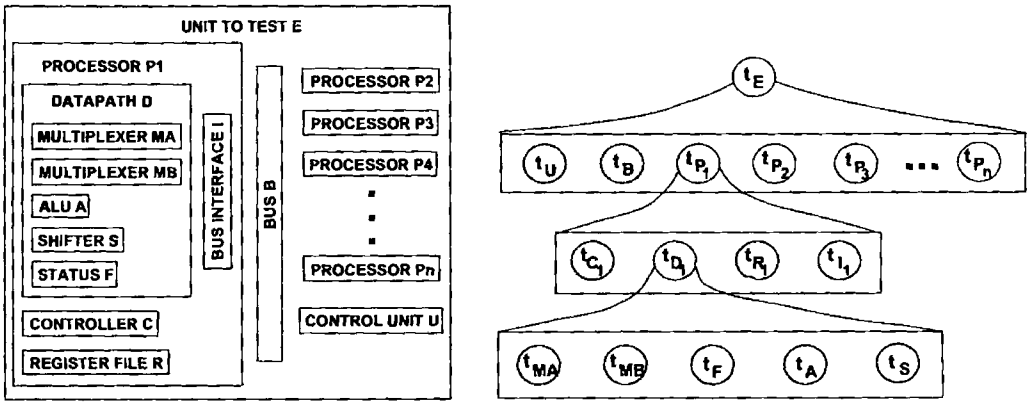
In [OH97, HO94], pipelined testing was made possible by the use of *partial-intrusion BIST* approach as illustrated in figure 2 10(a) This is a widely used HLTS technique (see section 2 1), which assumes an intelligent selection and transformation of a certain number of registers into BIST registers to increase design testability The rest of the logic is tested without reducing test throughput by using the non-test registers to propagate test data and pipeline it this way The selective BIST intrusion is driven by the fault-coverage/area-overhead ratio initially given to be achieved By using simple registers to store intermediate test values, test pipelining was enabled because each test path could be segmented into time-discrete stages In the case of figure 2 10(a) partial-intrusion BIST also reduced the need for test register insertion because the stimuli for a test stage could be supplied by the previous stage rather than a test register A pruning approach was employed to perform a gradual scheduling, incrementally removing scheduling options The algorithm in [HO94] was a repetition of three steps conflict evaluation, conflict avoidance, and unfeasible test option pruning They defined a *legitimate BILBO embedding* (see figure 2 10(b)) as a structure consisting of a kernel, a driving path for each input port (kernel port or K-port) and a receiving path for each output port of the kernel The conflict was modelled here like in [LNB91, AB85b] (see subsection

(a) Pipelined Testing Through REG (b) BILBO Non-pipelined Testing

Figure 2 10 Partial-Intrusion BIST Approach

2 5 2) Two embeddings were considered *fully compatible* if the I-paths between them did not have conflicts They could be executed in the same test session to reduce the total test time When the I-paths between two different embeddings had SCs, reservation tables for their test plans had to be analyzed to obtain the test time for executing them in one test session, and the test time for executing them in sequence If the former was shorter, these two embeddings were called *partially compatible* Only fully and partially compatible embeddings were considered for concurrent execution

2 5 4 Block-Test Scheduling



(a) Block Diagram of the Unit (b) Partial Testing Hierarchy

Figure 2 11 Block Test Hierarchy to Schedule

As a parenthesis, one can say that BIST scheduling is applicable to both test pipelining and block-test scheduling, where blocks are considered BILBO sandwiches and are the lowest level blocks in the test hierarchy RTL is the lowest level considered for block-test scheduling in a hierarchical test model In figure 2 11 a general perspective of the overall parallelism and the corresponding hierarchical model for block-test scheduling is presented The block diagram of the structure to be tested may include bus elements (bus B), processors (P_1, P_2, P_3, , P_n) and an overall control unit as in figure 2 11(a) Moreover, every processor can contain a datapath, a local controller, a bus interface and memory (register file) Furthermore, the datapath can consist of functional elements (ALU, shifters, coders/decoders), multiplexors, status registers The test schedule of the entire unit is the result of parallelism investigations carried out for each block of the test hierarchy given in

figure 2 11(b) Each test t_X in figure 2 11(b) represents the test of block X from the blockwise hierarchy depicted in figure 2 11(a) Some of the known approaches proposed for the block-test scheduling problem are presented next

In [KS82, CKS88, CY92] compatibility relations between complete test sequences for different hardware blocks were derived based on the total absence of resource conflicts in the execution of the tests Heuristic-based algorithms using classical graph colouring were employed to generate test schedules The work presented in [SW94] assumes that during HLS or during top-down design, test circuitry has already been added at the RT level, and the circuit has been segmented into subcircuits that are surrounded by test registers Block-tests are considered as test units to be clustered in test sessions if they are compatible Compared with other similar approaches, the block-test approach given in [SW94] takes into consideration the precedence relations (precedence tree) between block-tests In order to minimize the number of test sessions, the set of test units must be clustered into a minimal number of subsets with pairwise compatible test units A similar approach was found in [OH93], where a HLS system was extended to include BIST test path generation for test application time minimization The approach minimizes the test application time, firstly by minimizing the test application time of each individual module to be tested, and, secondly, by maximizing the concurrency between the testing of different paths Another block-level approach was formulated in [BKH97] to deal with the scheduling problem for equal and unequal length tests A clique partitioning method was described there to schedule the tests in the datapath having the control part logic synthesized already

In the block-level approaches mentioned above, the solution to the test scheduling problem is more or less based only on theoretical analysis From an implementation point of view, several parts of these test scheduling algorithms, especially, the identification of all cliques in a graph and the covering table minimization technique, belong to the class of NP-complete problems Therefore, heuristic-driven algorithms must be employed to obtain practical and near-optimal solutions Even though the schedule in [CSA97] is proved to be the optimum test schedule, no suitable practical algorithm for the test scheduling problem has yet been implemented Moreover, the approach theoretically given in [BKH97] does not suggest any possible practical solutions for the proposed algorithms, while the level of abstraction of the considered intermediate circuit design is a mixed RTL/gate level one mixed

RTL/gate level datapath and gate-level control

2 5 5 Test Scheduling Heuristics

In principle, the search for an optimal test schedule requires a complete scan for all possible sequential and concurrent combinations of test functions in different test sessions The outcome of the scan is obviously limited between serializing the issuing of all test functions and finding the best possible parallel issuing that saturates the system bottlenecks, i e , fully utilizes one or more of the most constraining bottlenecks of the system at the effective test application time Therefore, the sum of the application times of all the tests provides a loose upper bound on the total test application time, while the maximum of the sums of the application times for tests using a common resource provides a loose lower bound for the effective test application time The test scheduling problem for test application time minimization with or without power constraints is surely an NP-complete task Different heuristics have been proposed, based on the optimization approaches presented next The heuristics proposed early in literature to generate solutions for the test scheduling problem have mainly been applied at gate-level [SK88, SK92, AB86] or for mixed RTL/gate-level [BKH97] designs Lately, trends to higher-levels, RTL and algorithmic level, have emerged [OH97, CSA97, CKS88]

Two stepwise-improvement heuristic-based algorithms were employed as greedy approaches in [SK92] The first one was applied to the equal test length scheduling problem for test time minimization in pipeline structures [SK92] Some steps of this heuristic are based on the partitioning approach The second stepwise-improvement heuristic-based algorithm was applied to the unequal test length scheduling problem Even though this heuristic algorithm does not feature any hill-climbing mechanism a *scheduling history* is kept for *tabu* labels on unsuccessful motions Reference [NA99] used the tabu search-based heuristic to seek for BIST testable hardware solutions with a primary goal of minimizing test application time and to minimize BIST hardware overhead as the secondary goal Then a graph partitioning algorithm was employed to group modules of the same type that can be tested in parallel with the same test resources An integer linear programming solution was generated in [BWBM92] for the test scheduling problem This one was used to partition in sets the test plans such that the overall time overhead and area overhead were simultaneously minimized

Test compatibility partitioning was by far the most commonly used technique for test scheduling In [CSA97] this approach was used firstly to schedule equal length tests under power constraints by resource and power compatibility compliance The test application time minimization was carried out by a covering table minimization technique Secondly, in [CSA97, CY92], the scheduling of unequal length tests under power constraints was achieved again taking into account the resource and power compatibility and by employing the covering table minimization technique

A compatibility technique was proposed in [CKS88] to schedule equal length tests for test time minimization A resource compatibility graph was constructed and a graph colouring technique was used for test application time minimization Then the same approach for scheduling unequal length tests was studied in order to minimize the test application time The unequal length tests set was transformed beforehand into a larger set of equal length tests [CKS88]

The problem of scheduling equal length tests was approached in [BKH97] by the clique partitioning technique which was induced by the resource conflicts between tests Then the scheduling of unequal length tests was tackled in [BKH97, SW94] by clique (or chromatic) partitioning also induced by the resource conflicts between tests and test application moments Conflicts between tests can also be induced by the conflicts between the control signals that are used to drive the RTL test paths [BKH97] Two approaches were proposed in [CY92] The first one, called clustered partitioning test scheduling was similar to the unequal-length test scheduling described above The second one, called non-clustered partitioning test scheduling made use of the idle times left in the clustered test-compatible sessions, when the long compatible tests were still running while the short ones had completed The same problem of wasting idle times within the clustered test-compatible sessions was noticed in [Xia94], thus a test subsession partitioning was proposed there The idea behind the subsession level granularity was to get a finer grained conflict model between test resources

A clique partitioning approach, based again on test resource compatibility, was proposed in [AB86] as a solution to the test pipelining problem A weighted cluster partitioning approach was used in [Che91] to cluster the block-tests in test sessions by assigning a weight to each edge in the compatibility graph The value of the weight was used to select the best choice from a set of equally viable candidates The weight was given by the benefit of combining two block-tests in a test session

Compatibility partitioning, based on maximum clique partitioning or minimum covering partitioning (graph colouring) algorithms, is an NP-complete problem The graph colouring technique tries to colour each node in a graph such that no two adjacent nodes (nodes with a common edge) have the same colour The problem of finding a minimum cover of the Test Incompatibility Graph (TIG) is equivalent to finding its minimum colouring The set of nodes having the same colour in a minimum colouring of a TIG is analogous to the CTS described in subsection 2 5 2 In [CKS88, KS82], this problem has been solved using a suboptimal covering algorithm Studies were performed for random test graphs These studies showed that even for a reasonable number of tests the required computation became excessive This was primarily due to the very large number of cliques generated, which produced enormous covering tables These results prompted the need to find a heuristic that can generate a good solution without enumerating all of the cliques of the TCG In the following chapters a different model of the test scheduling problem is proposed to enable the development of heuristic algorithms Moreover, the heuristical algorithms always keep the power dissipation under a given limit

Scheduling Equal Length Tests

The procedure employed to carry out the search for scheduling equal length tests was presented in [KS82, CKS88] as consisting of the following steps

1 construct the TCG of the circuit,

2 find G, the set of all cliques of the TCG Let $G = \{G_1, G_2, \quad , G_r\}$, where each G_i is a clique of the TCG

3 by using a covering table, find a minimal subset S of G such that $\bigcup_S G_i = \{t_1, t_2, \quad , t_q\}$, the set of all tests in the TCG,

4 schedule all the tests in each G_i from S to run concurrently The total testing time is $\mid S \mid * T$, where $\mid S \mid$ denotes the size of the set S

Scheduling equal length tests has been formulated then at RT level [BKH97], but it actually was applied at a mixed RT - gate level However, it is very similar to the approach detailed above The BIST methodology has been presumed as well

Scheduling Unequal Length Tests

In the unequal length test problem, a test t_i requires T_i units of time to run completely. Moreover, if two tests t_1 and t_2 can run concurrently and $T_1 > T_2$, then both t_1 and t_2 are initiated simultaneously and t_2 will finish before t_1. Consider now the TCG shown in figure 2.12 which is taken from [CKS88, KS82] as an example. Suppose that the time for the completion of each of the tests is $T_1 = T_3 = T_5 = 2T$, $T_2 = T_4 = T_6 = T$. Three possible schedules for this example are depicted next.

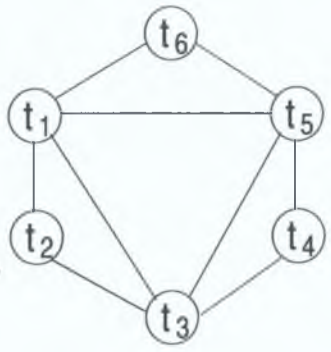


Figure 2.12: Unequal Length TCG

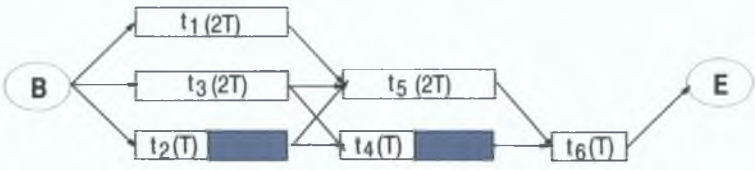


Figure 2.13: Nonpartitioned Testing

For the first case, suppose test t_2 can be modified to take $2T$ units of time to run completely or alternatively t_2 can be stopped and its results saved. Thus, only on completion of all tests (t_1, t_2, t_3) can the analysis of all tests' results be internally performed by a local controller or externally performed after the results have been accessed. This approach is called *nonpartitioned testing* in [CKS88, KS82] and is depicted in figure 2.13.

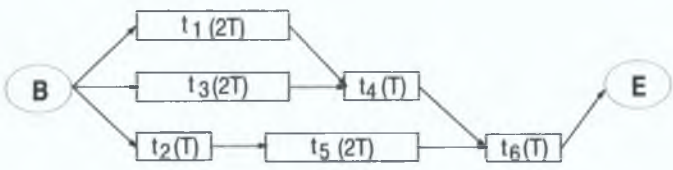


Figure 2.14: Partitioned Testing with Run to Completion

If local test analysis hardware exists internally, then local control can be provided to process each test independently. Therefore, in such an environment, t_1 proceeds uninterrupted while a new test, say t_5, which is compatible with t_1 (t_5 is incompatible

with t_2) can be initiated However, it is important to note that each test once initiated must run to completion This approach is called *partitioned testing with run to completion* in [CKS88] and is given in figure 2 14

The third possibility exists if a mechanism which permits storing and restoring of test state is available Tests t_1 and t_3 are interrupted upon completion of test t_2, the results of test t_2 along with the status of tests t_1 and t_3 are saved The results of t_2 can be analyzed and a new set of tests can be started Figure 2 15 depicts the case when test t_1 needs only be restarted from its interrupted state after restoring its status at interruption while the unfinished segment of t_3 is completed at a later time This approach is called *partitioned testing* in [CKS88]

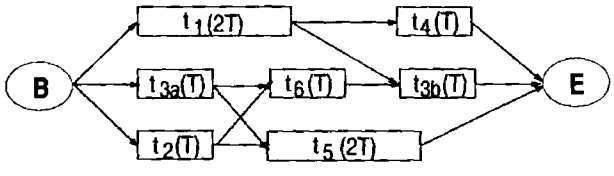
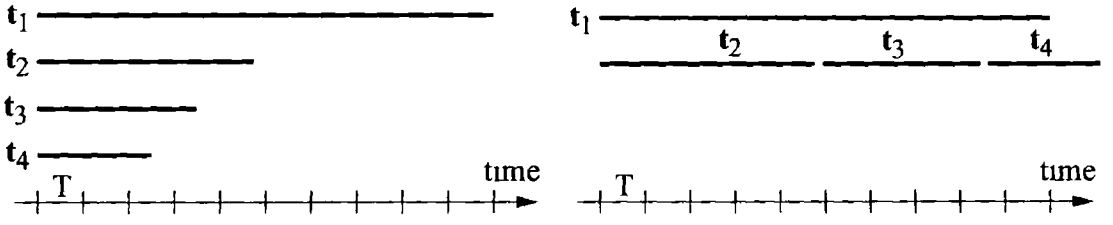


Figure 2 15 Partitioned Testing

In the above cases it was assumed that the time spent in saving and restoring of the partial state and results of a test and the time spent in the comparison of results is negligibly small compared to the duration of any segment of the test If the time overhead due to interruption is not small, then such time should also be added to the total time for testing [CKS88] Moreover, in the previous sections most of the testing was considered to be applied to kernels Kernels were usually considered functional logic However, an additional set of tests must be applied to registers and switching units (multiplexors) [OH97]

The theoretical approach enunciated in [BKH97] to give a solution to the scheduling problem with unequal length tests, was to introduce a coefficient $k < 1$ and break the longest length test t_{max} of a compatible clique into smaller length periods $t_k = k * t_{max}$ and spread the other compatible tests over these time slots (new test sessions) Figure 2 16(a) depicts an example of a test scheduling result generated by scheduling in a classic manner unequal length tests With the new heuristic, the result was that shorter length tests were sequentially executed in parallel with longer length test (see figure 2 16(b)) The benefit resulted is a more balanced schedule in terms of power consumption with less idle time This is the goal of the work presented in this thesis, but a power dissipation constraint is taken into

consideration as well However, the scheduling algorithms are based on a process of trade-off between tests' power/concurrency and the total test application time As it can be noticed in figure 2 16(b), tests t_2, t_3 and t_4 are run in parallel with test t_1, but the Total Test Application Time (TAT) has increased from $TAT = T_1$ in figure 2 16(a) to $TAT = T_2 + T_3 + T_4$ in figure 2 16(b)



(a) Unbalanced Test Schedule (b) Balanced Test Schedule

Figure 2 16 RTL Test Schedule Solutions

Chapter 3

PTS Problem Scope

The VLSI/SOC test problem is very complex and not practical to be dealt with at low levels *Test application time* is one aspect of any test strategy which deserves a sustained focus since it obviously affects the turnaround time (because the production test time increases along with the *test application time*) The heat dissipated during test application has started to heavily influence the design of test methodologies for large circuits It was reported in [Zor93] that one of the major considerations in test scheduling was the fact that heat dissipated during test application can be significantly higher than that during circuits' normal operation This is because one of the goals of testing optimization is to minimize test application time, thus the test session parallelism is high There is a high probability that high power consumption spikes of several tests could overlap and, therefore, power-conscious ways of scheduling them are to be sought Test application sequences that minimize the total test time while keeping the power dissipation under a limit should be searched for

The approaches proposed in this thesis solve the so-called *unequal-length block-test scheduling* problem [CKS88], because it deals with tests for blocks of logic, which do not have equal test lengths It is meant to be part of a system-level block-test approach to be applied on a modular view of a test hierarchy The modular elements of this hierarchy could be given in any HLS domain, between the system and RT levels (see figure 3 1) subsystems, backplanes, boards, MCMs, Integrated Circuits (IC)s (dies), macro blocks and RTL transfer blocks The lowest level block the test hierarchy accepts is RTL, and at this level it is assumed that a test-step level scheduling (see subsection 2 5 3) has already been carried out Generally speaking

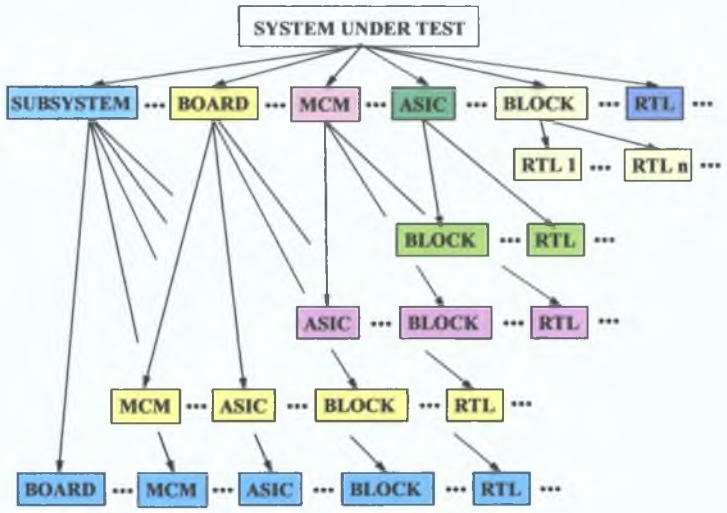


Figure 3.1: Example of System Under Test

any node in the hierarchy (apart from the leaves) has different subnodes. Every test node t_i is characterized by a few parameters, which are determined after the test scheduling optimization has been applied on the node. These features are given in figure 3.2: test application time T_i or Test Length (TL), power dissipation P_i, and test resource set R_i. This approach assumes a bottom-up traversing of the test hierarchy within a *divide and conquer* strategy. Thus, in this thesis it is assumed that the parameters of the RT level block-tests have already been optimized by test-step level scheduling approaches. Furthermore, the parameters of higher-level blocks are optimized by the algorithms proposed in this thesis by making use of the parameters of their sub-blocks. An *expanded tree growing technique* is employed here to generate the block-test schedule profile at any node level above the RTL one. This is carried out in order to minimize the overall test application time while analyzing and optimizing as much as possible the characteristics of power dissipation during testing.

The first section of this chapter describes the problems of *system testing* and the approaches currently employed to solve them. Emphasis is placed on the core testing techniques and core related scheduling approaches. Then, the second section gives a brief survey of the *power-conscious test parallelism* techniques. Here, a closer picture of the previous work on power-constrained test scheduling algorithms is taken. Finally, the model proposed for the aforementioned system testing problem under power constraints is described. The proposed power-test model is an efficient straight-forward description of the hierarchical system testing in terms of power,

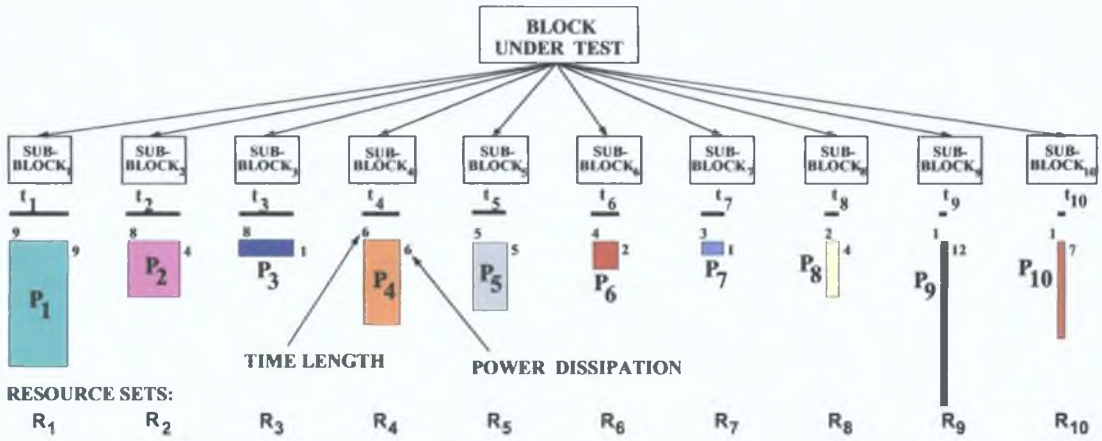


Figure 3.2: First Example of Node Under Test

test length and test resources. Efficient algorithms are then proposed in the next two chapters based on this power-test model.

3.1 System Testing

Since the size of today's digital SOC designs are well beyond the capabilities of most design teams, the effective use of such large capacities suggests, even demands, the ability to reuse existing designs [And97]. Nowadays the industry community provides reusable cores to their customers. A core is typically a hardware description language model of today's standard ICs, e.g., Digital Signal Processor (DSP), Reduced Instruction Set Computer (RISC) processor, Dynamic Random Access Memory (DRAM). Such cores may be available in synthesizable RTL (soft) form, gate-level netlist form (synthesized RTL or firm form), or layout-level "hard macro" form [Zor97]. Thus, the core synthesis and its testing are two topics under sustained research lately.

In system testing a core should be well characterized with respect to the test, power dissipation during test, and fault coverage. The core internal test prepared by a core creator needs to be adequately described, portable and ready for plug and play, i.e. for interoperability, with the SOC test. In order to have an interoperable internal test of a core, it needs to be described in a commonly accepted standard format. Such a standard is currently being developed by the IEEE P1500 [Zor97]. In addition to the test integration and interdependence issues, the SOC composite test requires adequate test scheduling. The scheduling is needed to meet a number of system level requirements, such as total test time, power dissipation limit and

area overhead Also, test scheduling is necessary to run intra-core and inter-core tests in a certain order, so as not to impact on the testing contents of individual cores or modules The intra-core testing follows in principle the well-known testing methodologies (e g, BIST) proposed so far

Interest in MCMs has lately grown rapidly as the benefits in miniaturization, higher performance and reliability continue to be demonstrated in a wide range of commercial and military electronic products The *production test* and *field test* of huge MCM designs could suffer from *test application time* and *power dissipation* (during test mode) points of view, unless they have been optimized during test development Thus, the complexity and the dimensions of such electronic systems like MCMs determine the need for additional *test application time optimization with power dissipation constraints*

Most of today's electronic systems are a big challenge for test designers For example, taking the case of the highly integrated switching system given in [Hug97], which has up to 77 dense logic boards containing hundreds of high pin count VLSI devices, the thought of testing that system can be compared to a bad dream This kind of system can be thought of as a hierarchical structure like in figure 3 1 starting at highest level with backplanes, continuing downwards with boards, complex VLSI circuits, MCMs, Application Specific Integrated Circuit (ASIC)s, embedded cores, blocks, down to RTL transfer blocks Within the hierarchy, the most difficult elements to test are the complex VLSI and MCM elements which are a testing bottleneck for big systems like the one mentioned above This is due to the lack of efficient solutions for interfacing the large BIST structures inside these elements and the Boundary Scan (BS) structures around them

3 1 1 Core Testing

Cores may come with a wide range of functionalities and in hierarchical compositions like the structure from figure 3 3 described in [Zor97] Because today's VLSI chips use multiple cores from different providers the testing problems and the power dissipation during it are exacerbated The interconnection of cores within the chip and the ability to perform an effective test on the final device are the two key issues that all core users must face The *test concurrency* of a core-based system's testing depends largely upon the core supplier's *application interface*, that is the set of signals and resources by which the core connects to the rest of the chip

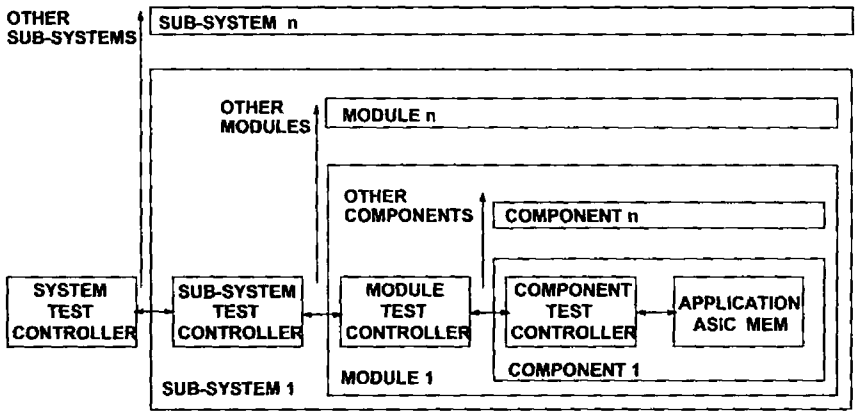


Figure 3 3 A General Hierarchical Test Structure

Furthermore, in system testing, the failure mechanisms differ from those in production and the test time and power requirements (e g for BIST) may be different Hence, a core should be well characterized with respect to test, fault coverage and power consumption (in both normal and test modes) Ideally, the internal test should be a composite one, containing modular sections with different characteristics This is fully achievable in a system consisting only of soft cores, where an overall test strategy is possible after optimizing the tests at every level of the system in a bottom-up manner (RTL transfers, block level, IC level, chip level, MCM level, SOC level) The most common methods supporting core test were given in [And97] This was implemented, firstly, by providing parallel multiplexed access from chip pins to promote core functional tests to run in the complete chip The *test concurrency* was decreased when there were more core I/O than chip pins or when routing was complex The next step was to encapsulate cores in a Joint Test Action Group (JTAG)-like boundary scan ring to run in parallel isolated core tests with little need for external support Finally, scan or BIST techniques were used to test each core and to provide internal controllability and observability, and connecting core test paths together (possibly serially) Unless multiple cores can be tested in parallel, system test time can be unacceptably long Fortunately, BIST is an autonomous testing method and, therefore, was considered ideal for modular-based systems [Zor97]

The core test scheme has to provide as well a modular interface to allow integration of a hierarchical test control scheme and allow potential sharing of test resources at higher levels [Zor97] In order to achieve high fault coverage with low

area overhead and low power dissipation for complex SOCs, a specific BIST technique is needed for each module [Zor98] There are BIST strategies like the one in [Zor90] which tried to solve this problem by utilizing a divide and conquer approach to enhance the overall controllability and observability However, this strategy was still reported as having pending problems in isolating and accessing the boundaries of the modules on one hand, and in automating the process of assembling the set of inter-module and intra-module tests in the complete chip, on the other hand [Ben97]

Macro Test is an approach for testing embedded modules as stand-alone units, and hence is very suited to core-based testing From this point of view it is very suitable for hierarchical approaches It supports every type of test access, including parallel direct access, serial scan and BIST, as long as the access is sufficient for the particular module under consideration Macro Test is based on the following concepts [ML99] A test is broken into a *test protocol* and *test patterns*, where the test protocol describes how to apply the test patterns to the inputs and observe the outputs of the macro under consideration Then a translation of macro-level tests to IC-level tests is done by *test protocol expansion* Thus, Macro Test and test protocol expansion are designed to support multiple levels of hierarchy in a design This concept is useful for the approach presented in this thesis The various core tests, once expanded to chip level, can either be applied in a simple sequential order, or scheduled by the *test protocol scheduler* [MA98] The test protocol scheduler tries to execute as many as possible of the various core tests in parallel in order to reduce test time However, the power dissipation increase while reducing the test application time was not taken into account

The test methodologies for SOC designs have selected mostly hierarchical approaches to tackle the modular structures The approach proposed in this thesis works with any core test standard because the cores are particular cases of design blocks

3 1 2 Core Test Scheduling

For a core-based design, with a given set of cores, and given corresponding expanded test protocols and sets of test patterns, test protocol scheduling (TPS) tries to schedule the various expanded test protocols such that the total test vector set of the IC is minimized [MA98] It was also proved there that the test scheduling at

test protocol level offers a good trade-off between test vector set reduction and the computational effort required to achieve this

[Cha99] proposed an optimal solution approach for the test scheduling problem for core-based systems It was based on a mixed-integer linear programming model However, this approach featured a non-polynomial complexity In order to handle such systems, a shortest-task-first heuristic scheduling algorithm was proposed instead Given a set of tasks (test sets for the cores), a set of test resources and a test access architecture, the test scheduling solution in [Cha99] referred to the problem of determining start times for the tasks such that the total test application time was minimized Other approaches like [Mea98, ZMD98] addressed the core testing at system level by focusing only on the design of efficient test access architectures

[SDY98] addressed the problem of selecting a test set for each core from a set of test sets provided by the core vendor and scheduling these tests in order to minimize the testing time Each test set consisted of a subset of patterns for BIST and a subset of patterns for external testing This approach requires the core vendor to provide multiple test sets for each core, with the test sets containing varying proportions of patterns for BIST and external testing The scheduling problem was formulated as a combinatorial optimization problem and solved heuristically Two restrictive assumptions were made in this approach every core had its own BIST logic, i e , the BIST components of the test sets for any two cores could be assigned identical starting times, and external testing could be carried out for only one core at a time, i e , there was only one test access bus at the system level

A simple greedy heuristic was also proposed for core test scheduling under power constraints in [LP99] Moreover, the same work was developed in [LP00b, LP00a], as a technique for test scheduling and design of test bus infrastructure at the same time Test application time and test bus length and width were minimized while constraints on power consumption and test resources were considered That approach had two stages firstly, a greedy heuristic was repetitively used to select at a low computational cost a non-optimal solution, then a simulated annealing approach was used to optimize the solution These algorithms took into account the power dissipation during idle and active states of the cores during test application However, the disadvantage of these core test scheduling approaches was that they were aimed only at the particular case of core-only based designs

3.2 Power-Conscious Test Parallelism

Power dissipation during test has not yet been thoroughly researched, though, in recent years, portable and mobile communications have attracted the researchers attention in this direction For example, in the technological domain the proposed solutions were to test with reduced operation frequency and to oversize power supply, package and cooling to stand the increased current during testing A few structural domain approaches tackled the power dissipation problem during test application at logic level such as switching activity conscious ATPG, scan latch reordering, test vector reordering, and test vector inhibiting Unfortunately, the above approaches are not efficient at high levels The only efficient solution known at system-level is to partition the system under test and propose an appropriate test planning and scheduling for it [Zor93] (see subsection 3 2 2) Nevertheless, not many practical solutions have been found that are able to solve the test scheduling problem under power constraints at high level This thesis proposes a feasible solution in the next chapters to sort out this problem Before that, this section describes the power dissipation problem during test application and surveys the main techniques known to have been applied in solving this problem Then, the focus will be on the previous work on power-constrained test scheduling techniques

3 2 1 Power Minimization During Test Application

The correlation between consecutive test vectors generated by an ATPG is very low, since a test is generated for a given target fault without any consideration of the previous vector in the test sequence It was observed that the ordering of both scan flip-flops and the test patterns influences power and energy [DCPR98, CD94c, DC94, CD94a, CD94b, WG99, WG97b, WG97a, WG98, WG94, CFN+98, CFN+99] Some of the above techniques [WG98, WG97a, CD94d] were proposed for deterministic test patterns, the rest were more or less aimed at BIST methodologies Apart from the techniques proposed at a higher level (by test scheduling), most of the above techniques were given at logic level and can be classified into those that apply to test-per-scan BIST schemes and those that apply to test-per-clock BIST schemes

In test-per-scan BIST, a test pattern is applied to the CUT (via a scan chain) every $m + 1$ clock cycles, where m is the number of flip-flops in the scan chain The response is captured into the scan chain and scanned out during the next m

clock cycles while the next test vector can be scanned in simultaneously Several low power testing strategies were proposed for scan-based BIST In [HW98], the authors proposed a modification of the scan cell design in such a way that the CUT inputs remained unchanged during shift operation This novel design for scan path elements allows significant energy savings compared to a standard scan-based BIST scheme In [WG99] a low-transition random pattern generation technique was proposed to reduce signal activity in the scan chain A k-input AND gate and a T latch were used to generate high correlation between neighboring bits in the scan chain, thus reducing the number of transitions and hence the average power In [GW99] it was proposed to combine the toggle suppression technique from [HW98] with the use of additional logic in the scan path design for suppressing random patterns which did not contribute to the increase of fault coverage Weighted Switching Activity (WSA) was used in [GW99] as a metric for energy consumption since in static Complementary Metal-Oxide Semiconductor (CMOS) circuits the switching contributes to over 90% of the total energy consumption [DM95]

A post-ATPG phase was proposed to reduce power dissipation for full-scan [CD94c, CD94a, CD94b] and for pure combinational [DC94] circuits [DCPR98] used a *transition graph* for low power consumption in scan circuits and combinational circuits In the full-scan case, first of all, a fixed scan-latch ordering was assumed and a test-vector ordering was computed, by a greedy heuristic, in order to minimize the power dissipation during test application Then two heuristics (random ordering heuristic and simulated annealing) were used to minimize the power dissipation by both, the scan-latch ordering and test-vector ordering Finally, switching activity was inhibited in the embedded combinational logic by circuit disabling methods while the test values were scanned-in and scanned-out

In test-per-clock BIST, the outputs of a test pattern generator are directly connected to the inputs of the CUT At each clock cycle a new test pattern is applied and the response is loaded into the response analyzer Switching activity in the CUT can be reduced by generating test vectors from TPGs that cause less transitions at circuit inputs In this sense, [WG97b] proposed a BIST strategy based on two different speed LFSRs Their objective is to decrease the overall internal activity of the circuit by connecting inputs with elevated transition density to the slow LFSR This approach reduces the average power consumption with no loss of fault coverage, but not the same is guaranteed to be happening with the peak power

consumption In order to minimize the energy dissipation during test, [GGLP99a] proposed a reseeding scheme together with a vector inhibiting technique to tackle the increased activity during test operation of hard-to-test circuits that contain pseudo-random resistant faults An enhancement of this technique is proposed in [MGF+99] where the filtering action is extended to all the non-detecting vectors of the pseudo-random test sequence However, these techniques do not prevent the circuit from an excessive peak power consumption either A different technique was proposed in [GGL+99] in order to reduce the energy consumption by cleverly selecting the parameters (seed and polynomial) of the LFSR TPG The work in [ZRB99] modifies the LFSR by adding weight sets to tune the signal probabilities of the pseudo-random vectors in order to decrease the energy consumption and increase the fault coverage Unfortunately, the area overhead imposed by the weighted TPG may represent a severe limitation for the use of this approach In [GGLP99a], a test-per-clock BIST that can reduce overall energy consumption during entire test application time was proposed This technique also fails to consider power dissipation spikes which can damage the CUT in a short period of time Furthermore, it is not applicable to test-per-scan BIST

The term *adjacency test* was first introduced in the context of pseudo-exhaustive BIST [CK85] It means that only a single transition is applied at the primary inputs of the CUT at each clock cycle of the test session In other words, it means that the Hamming distance between vectors of the test sequence is always one [CFN+98, CFN+99] It was proved in [GLPS98] that there is a strong correlation between the Hamming distance of vectors of a given test sequence and the switching activity induced in the CUT

A BIST pseudo-random pattern generator for test-per-clock was designed in [GGLP00] to reduce average power and peak power It consisted of an adjacency-based TPG plus a conventional pseudo-random TPG (i e a LFSR) Each test pattern generated by the mixed TPG was thus composed of two parts in one part only one bit was changed and, in the second part bits were randomly generated from the LFSR The idea behind the use of such a structure is to reduce the number of transitions on primary inputs for each clock cycle of the test application session Thus, only for the primary inputs needing a high fault coverage a LFSR was still used

[GGLP99b] proposed a test vector reordering technique for a given test sequence

in order to minimize the average and peak power during test operation The technique considered combinational or full scan sequential circuits and did not modify the initial fault coverage The technique proceeded as follows Firstly, an initial deterministic test sequence for the considered CUT was derived from an ATPG program Next, from the signal transition probability on primary inputs calculated from the initial test sequence, the transition density was computed at each input

ATPG for the minimization of heat dissipation during test application was proposed for combinational [WG98, WG97a] and scanned [WG94] circuitry Test vectors generated by the proposed ATPG were shown to decrease heat dissipation during test application by a significant factor Three cost functions, namely, transition controllability cost, transition observability cost, and transition test generation cost were defined to be employed by the ATPG proposed for combinational circuitry The ATPG proposed in [CFN+98, CFN+99, WG94] reduced heat dissipation during testing of sequential circuits with full-scan by exploiting all don't cares that occurred during scan shifting, test application, and response capture in order to minimize switching activity in the CUT For the completely specified test patterns, papers [CFN+98, CFN+99] modelled the solution space as a Euclidian Travelling Sales Person (TSP) which is NP-hard, but several existing efficient polynomial-time approximation algorithms could be applied On the other hand, the approach in [WG94] used an iterative improvement bipartitioning algorithm to assign values to the don't cares so as to maximize the blocking of unwanted transitions

A gate-level low-energy BIST strategy based on circuit partitioning was proposed in [GGLP99c] The strategy could be applied to both test-per-clock and test-per-scan BIST schemes and consisted of partitioning the original circuits into two structural subcircuits so that each subcircuit could be successively tested through two different BIST sessions By partitioning the circuit and planning the test session, the switching activity in a time interval (i e the average power dissipation) as well as the peak power consumption were minimized Moreover, the total energy consumption during BIST was also reduced since the test length required to test the two subcircuits was roughly the same as the test length for the original circuit

3 2 2 Power-Constrained Test Scheduling

The absence of resource conflict for a pair of tests does not mean these two tests can be applied concurrently, because the total power consumption must not exceed

the maximum power limit in order to guarantee proper operating conditions during test and thereafter For example, memories are organized into blocks of many fixed sizes Under normal system operation, exactly one block is activated per memory access while other blocks are in the power-down mode to minimize the power consumption In the testing environment, however, in order to test the memory system in the shortest possible time, it is desirable to concurrently activate as many blocks as possible provided that the power consumption limit of the system is not exceeded Another example is the testing of MCMs An attractive approach for testing MCMs is to use BIST blocks executing in parallel Under normal operation, blocks are not simultaneously activated and hence, the inactive blocks do not contribute significantly to the total power dissipation However, a concurrent execution of BIST in many blocks will result in high power dissipation which might exceed the maximum power dissipation limit To ensure the reliability of the system, execution of the self-test blocks must be scheduled in such a way that the maximum power dissipation limit is not exceeded at all times during test

The power dissipation requirements can be satisfied during test application in several ways For the test scheduling problem at least two solutions exist [CSA97] clock(s) can be slowed down to reduce the average dynamic power dissipation, or the tests can be executed in sequential order such that no two tests are overlapped in time These methods are just two extremes in the attempt to reduce the power dissipation during test application, both at the expense of the total test time A better way of scheduling the tests would be to minimize the total test time by maximizing test parallelism while satisfying the power constraints

Most of the proposed block-test scheduling techniques address only logic-level blocks in order to schedule for test time minimization by using parallelism, or to schedule for area overhead optimization by sharing test resources in data path blocks [CKS88] These techniques are certainly valid for logic-level or, at most, RT level blocks However, they cannot, for example, schedule BIST of blocks in parallel for a complex VLSI device, due to their power and noise dissipation impact during BIST execution

The BIST scheduling approach given in [Zor93] was one of the first to take into account power dissipation during block-test scheduling It performed global optimization considering also other factors such as block type, adjacency of blocks (device floor plan), but the latter are usually unknown at high-level Moreover, in

complex VLSI circuit designs, the block-test set is large and varies in test lengths Therefore, this approach failed to provide any polynomial complexity algorithm to solve this problem, focusing only on defining and analyzing it

The problem of minimizing power dissipation during test application was addressed at higher levels in [CSA97, CSA94] In [CSA97] the scheduling problem of equal length tests with power constraints was formulated The objective was to find a power-constrained test schedule that covered every test in at least one test session such that the total time required for testing was minimum The solution was obtained in two steps identifying the solution space and then searching the solution space for an optimum solution For solution space identification the following definitions were given *Power Compatible Set (PCS)* is a set of tests that can be executed concurrently, i e , they are time compatible and satisfy the power constraints *Maximum PCS* is a PCS to which no compatible tests can be added without exceeding the maximum power consumption limit For the unequal length test case the maximum PCS, dealt with in the equal length test case (see subsection 2 5 5), was first expanded in order to enlarge the solution space to include all the possible optimum solutions for the unequal length test problem under power constraints

3.3 PTS Problem Modelling

The Power-Constrained Block-Test Scheduling (PTS) problem was first analyzed in [CSA97] at IC level, but this was only a theoretical analysis It is, basically a compatible test clustering problem, where the compatibility among tests is given by test resource and power dissipation conflicts at the same time However, the identification of all cliques in the graph of compatible block-tests belongs to the class of NP-complete problems, thus it is not practical An extended tree growing heuristic is proposed instead in this thesis together with classical scheduling algorithms to deal with the PTS problem The approach has a polynomial complexity, which is very important for the efficiency of the system-level test scheduling A *constant additive model* is employed for power dissipation analysis and estimation throughout the approach and it will be described next The proposed approach belongs to the so-called *unequal-length block-test scheduling* class, because it deals with tests for blocks of logic, which do not have equal test lengths The order of the tests within

the test sets of the various modules in the circuit is not considered in this approach The test scheduling discipline assumed here is the *partitioned testing with run to completion* defined in [CKS88] (see subsection 2 5 5)

3 3 1 High-Level Power Dissipation Estimation

Five parameters are important to evaluate the power properties of a BIST architecture Most of them are enumerated in [GW99] The *consumed power* (energy) directly corresponds to the switching activity and has an impact on battery lifetime and junction temperature during testing It is obtained by integrating power on a cycle by cycle basis over the test application time The *time-averaged power* (average power) is the consumed power divided by the test time This parameter is important as hot spots and reliability problems may be caused by constantly high power consumption The *maximum power* (peak power) corresponds to the highest power consumption value during the test time If the maximum power exceeds certain limits the functionality of the circuit may be affected temporarily or even permanently This parameter is used to size power buses to meet worst case noise margins The *Root Mean Square (RMS) power* is used for circuit sizing to meet electromigration guidelines The *instantaneous power* is used to determine the allowable parasitic inductance presented in the device These values are used for the minimization of ground bounce effects

The instantaneous power, $p(t)$, is the power dissipation at any time instant t $p(t) = i(t) \times v(t)$, where $i(t)$ and $v(t)$ are the instantaneous current and voltage in the circuit The voltage in general does not vary and is equal to the power supply, i e $v(t) = V_{dd}$ If $p_i(t)$ is the instantaneous power dissipation of test t_i and $p_j(t)$ is the instantaneous power dissipation of test t_j, then the power dissipation of a test session consisting of just these two tests being overlapped is the sum of the instantaneous power of test t_i and t_j as can be seen in figure 3 4 depicted in [CSA97] Usually this instantaneous power is constrained so as not to exceed a maximum power dissipation limit, P_{max} However, in reality the instantaneous power for each test vector is hard to obtain since it depends in a CMOS circuit on the number of zero-to-one and one-to-zero transitions, which in turn could be dependent on the order of execution of test vectors (see subsection 3 2 1) Consequently, different test schedules will result in different instantaneous power dissipation profiles for the same test In [CSA97], in order to simplify the analysis, a fixed power value P_i was

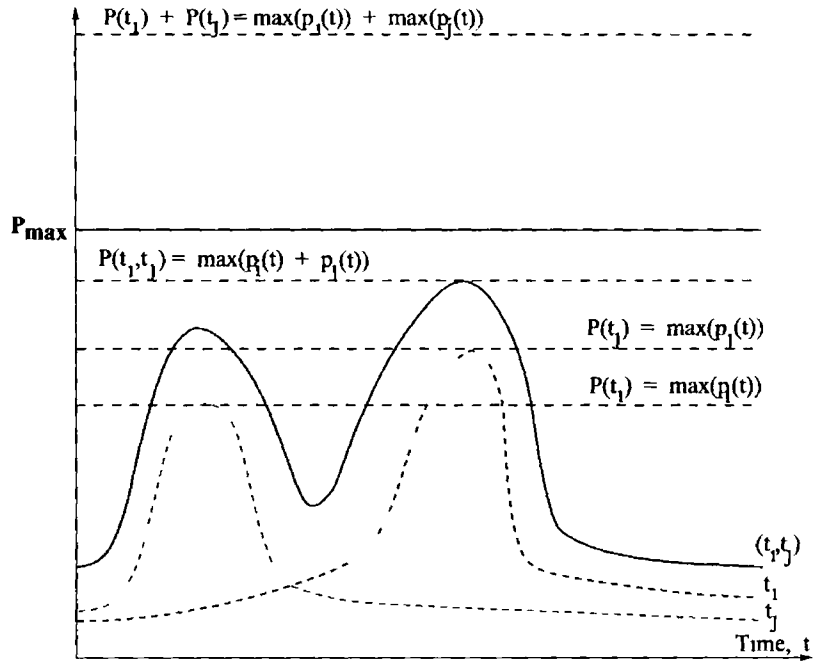


Figure 3 4 Power Dissipation as a Function of Time

assigned to all test vectors in t_i such that at any time instant when the test t_i is in progress the power dissipation was no higher than P_i

In this thesis the above described approach for power analysis is adapted to work with the proposed PTS algorithms At high-level, accurate power evaluation is impossible and the only solution is power estimation Thus, a *constant additive model* is employed for power estimation A constant power dissipation value is associated with each block-test t_i The total power dissipation at a certain moment of the test schedule is computed by simply summing the power dissipation of the running block-tests The power dissipation P_i of a test t_i could be estimated at a high-level in three ways Firstly, P_i can be defined as the *average power dissipation* over all test vectors in t_i This definition might be overly optimistic in the analysis of power dissipation when many test vectors are simultaneously applied, since the average value cannot reflect the instantaneous power dissipation of each test vector Hence, it might lead to an undesirable test schedule which exceeds the power dissipation limit of the device at some time instants Secondly, P_i can be defined as the *maximum power dissipation* (peak power) over all test vectors in t_i This is the upper bound power dissipation in t_i and its definition is pessimistic in this case since it disallows two tests t_i and t_j, whose peak power occur at different time instants, from being scheduled in the same test session as shown in figure 3 4 However, the test

schedule obtained with this definition guarantees the maximum power dissipation limit of the device to be observed at all times Thirdly, an *RMS power dissipation* can be employed when the instantaneous power dissipation is prone to power spikes and a more accurate estimation is sought The RMS power formula is given in equation 3 1 The RMS power value is located between the average and maximum power values and it is sometimes called the *effective power* value The higher the number of spikes in the power distribution, the smaller the difference between the RMS and maximum power values Though, as it will be seen in chapter 6, the RMS power value is usually closer to the average power value

$$P_{RMS} = \frac{1}{N}\sqrt{\sum_{i=0}^{N-1} P_i^2}, \tag{3 1}$$

3.3.2 PTS Problem Formulation

In the test environment, the difference between the above different power estimation values for each test is often small [CSA97] since the objective is to maximize the circuit activity so that the circuit can be thoroughly tested in the shortest possible time This aspect is true especially with the approach proposed in this thesis where the lowest level block considered in the test hierarchy is the RTL and, at this level it is assumed that a test-step level scheduling has already been applied Moreover, using the approach proposed here to optimize the blocks in the test hierarchy from the lowest level (RTL) to the top level (system level), the differences between the power values could be further ignored That is because at each level, after the PTS algorithm is applied, it is believed that the circuit activity (power consumption) is maximized and balanced

Therefore, it is reasonable to define P_i as the maximum power dissipation over all test vectors in t_i [CSA97] Hence, in the subsequent analysis, P_i is assumed to be the maximum power dissipation of test t_i However, the statement of the problem and the constraints given are independent of the method of assigning values to P_i for a test t_i Thus, the power dissipation P_{s_j} for a test session s_j can be defined as $P_{s_j} = \sum_{t_i \in s_j} P_i$, while the power constraint in test scheduling as defined in [CSA97] is $P_{s_j} \leq P_{max}\ \forall j$ To ensure the reliability of the system, the PTS problem is formulated as the approach used to schedule tests in such a way that the power dissipation limit is not exceeded at all times during test Basically, by test scheduling

all the time compatible block-tests are found and, their overlaying and/or sequencing is carried out with the goal of determining a sequence exhibiting a minimum total test application time When power dissipation is also considered in the scheduling of tests, the block-tests scheduled in the same test session may not be compatible from the power dissipation point of view, because, by executing all tests in the same test subsession, the accumulated power dissipation might exceed the maximum power limit imposed by the technology To overcome this problem an efficient scheme for overlaying the block-tests, called *extended tree growing technique*, is proposed in the next subsection to work together with some classical scheduling algorithms adapted in the next two chapters to find near-optimal solutions in a polynomial time *From this point, throughout the thesis, by test (when used) is meant block-test for simplicity*

3 3 3 Tree Growing Technique

Due to the wide range of test lengths exhibited by the block-test set applied to a complex VLSI circuit, it is possible to schedule some short tests to begin when subcircuits with shorter testing time have finished testing, while other subcircuits with longer testing time have not (if they are compatible) The *tree growing technique* given in [JPP89] is very productive from this point of view That is because it is used to exploit the potential of test parallelism by merging and constructing the Concurrent Test Set (CTS) This is achieved by means of a *binary tree structure* (not necessarily complete), called *compatibility tree*, which is based on the compatibility relations between tests

Nevertheless, a big drawback in [JPP89] is that the compatibility tree is a binary one This limits the number of children test nodes that could be overlapped to the parent test node to only two In reality the number of children test nodes can be much larger, as in the example depicted in figure 3 5 Therefore an Extended Compatibility Tree (ECT), given by means of a *generalized tree*, is proposed here to overcome this problem The sequence of nodes contained in the same tree path of an ECT represents an expansion of the CTS Given a partial scheduling chart of a CTS, a test t can be merged in this CTS if and only if there is at least one tree path P in the corresponding compatibility tree of the CTS, such that every test contained in the nodes of P is compatible with t The compatibility relationship has three components Firstly, tests have to be compatible from a resources point

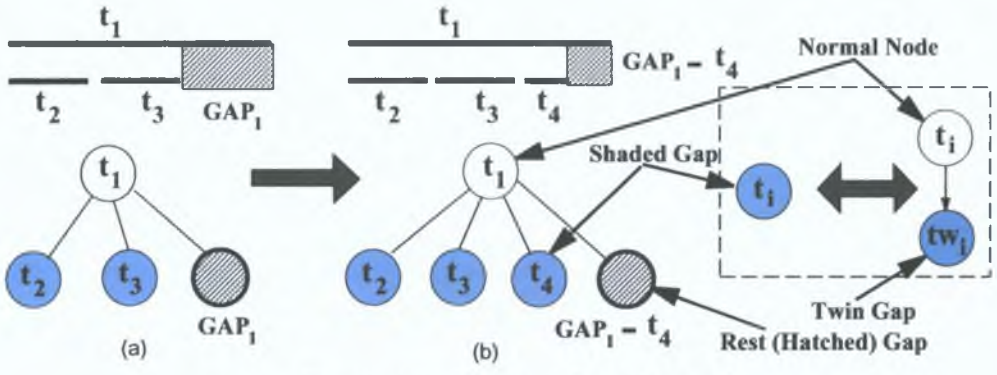


Figure 3.5: Merging Step Example

of view. Appendix A gives a comprehensive analysis of the test resource compatibility problem. Secondly, the test length of the nodes in a tree path have to be monotonously decreasing from root to leaf. Thirdly, the power dissipation accumulated on the above tree path should be less than or equal to the power dissipation constraint P_{max}.

A *merging step* example is given in figure 3.5. The partial test schedule chart is given at the top, while the partially grown compatibility tree is given at the bottom. Let us assume that tests t_2, t_3 and t_4 are compatible with t_1, while they are not compatible with each other. Also assume that T_1, T_2, T_3 and T_4 are, respectively, the test lengths of tests t_1, t_2, t_3 and t_4, and say $T_2 + T_3 < T_1$. Finally, assume that test t_4 has to be scheduled in the partial test schedule depicted in figure 3.5(a). As can be seen, there is a gap GAP_1 given by the following test length difference: $GAP_1 = T_1 - (T_2 + T_3)$. Thus, a merging step can be achieved, if $T_4 \leq GAP_1$, by inserting t_4 in the partial test schedule and its associated ECT as in figure 3.5(b).

The process of constructing CTSs is implemented by growing the ECT from the roots to their leaf nodes. The root nodes are considered test sessions, while the expanded tree paths are considered their test subsessions. When a new test has to be merged to the CTS, the algorithm should avail of all possible paths in the ECT. In order to keep track of the available tree paths and to avoid the complexity of the generalized tree travel problem, a list of potentially Expandable Tree Path (ETP) is kept. This list is kept by means of special nodes that are inserted as leaf nodes in each ETP of the expanded compatibility tree. These leaf nodes are called *gaps* and are depicted as hatched or shaded nodes in figure 3.5. There are two types of gaps. The first set of gaps (hatched) are those "rest gaps" left behind each merging

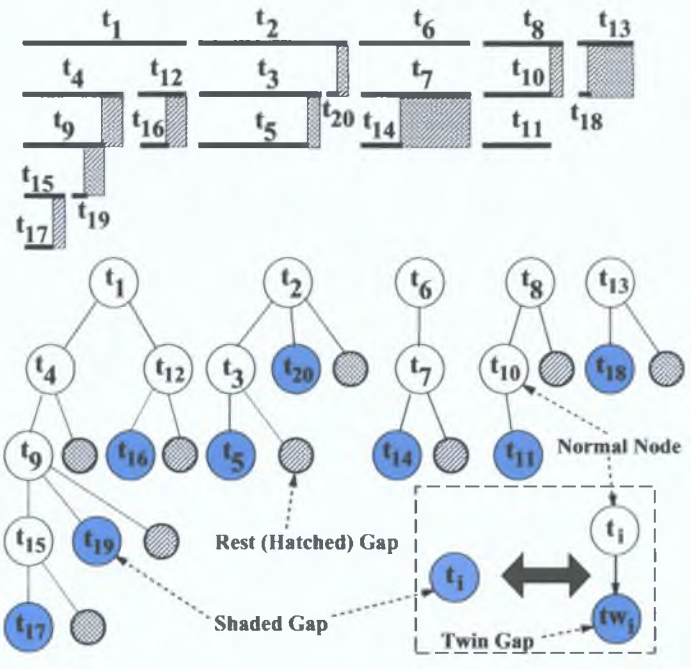


Figure 3.6: Test Scheduling Chart and ECT Example

step, as in the cases of GAP_1 and $GAP_1 - t_4$ in the above example. They are similar to the incomplete branches of the binary tree from [JPP89]. The second set of gaps (shaded), are actually bogus gaps generated as the superposition of the leaf nodes and their twins as in the equivalence given at the right in figure 3.5. They are generated in order to keep track of "non-saturated" tree paths, which are also potential ETPs. By "non-saturated" tree path is meant any ETP with accumulated power dissipation still under the given power dissipation limit. The root nodes (test sessions) are considered by default "shaded" gaps before being expanded.

3.3.4 Power-Test Scheduling Chart

The test scheduling chart example given in figure 3.6 is the test schedule solution generated by the List Scheduling Based PTS Algorithm (PTS-LS) (detailed in chapter 4) on its application to the 20 Block-Test Set (BTS) depicted in figure 3.7 and given in subsection 4.1.3. In order to have a more intuitive representation, the test scheduling chart in figure 3.6 can be easily translated into a PTS chart in figure 3.8. This representation gives a clear view of the power dissipation distribution over the test application time. The power-test characteristics of the schedule solution

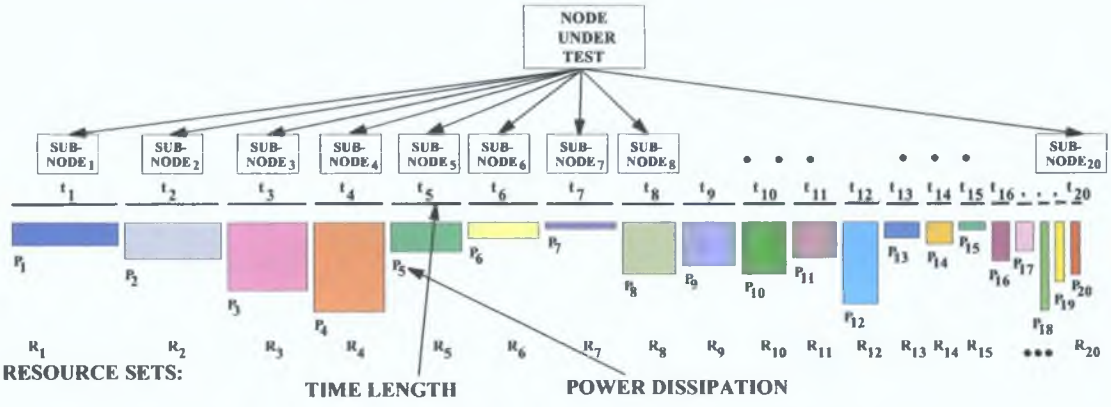


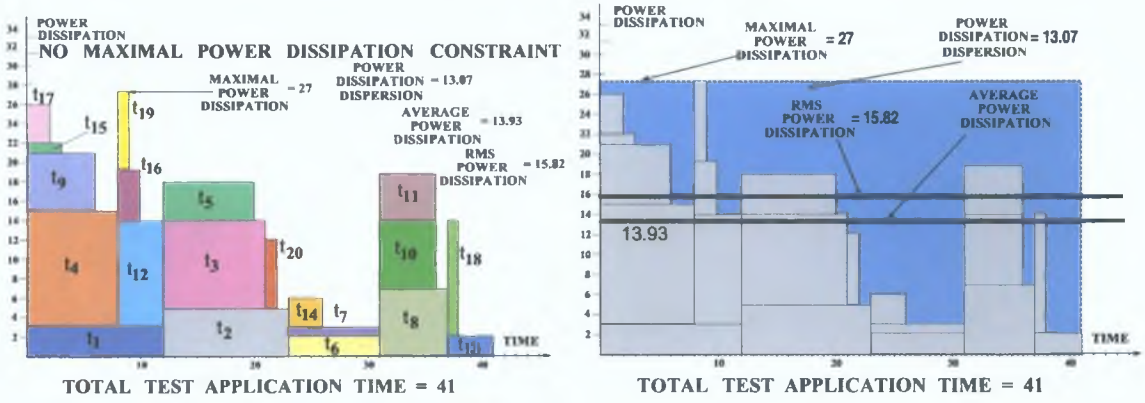
Figure 3.7: Second Example of Node Under Test

in figure 3.8(a) are given in figure 3.8(b). These characteristics are defined and explained next.

3.3.5 Power-Test Scheduling Chart Characteristics

The characteristics of a power-test scheduling chart (e.g., figure 3.8(a)) are defined as follows (see figure 3.8(b)): Test Length (TL), Maximal Accumulated Power Dissipation (MPD), Average Power Dissipation (APD), Power Dissipation Dispersion (PDD), and RMS power dissipation (RMS). TL represents the total test application time of the test scheduling solution. MPD is the maximum power dissipation over the final power-test scheduling solution. APD is considered the ideal MPD when all the ETPs exhibit the same accumulated power dissipation, that is, the power dissipation is fully balanced over the power-test scheduling chart. It is calculated with formula: $APD = (\sum_{i=0}^{N-1} P_i * T_i)/TL$. The rectangle given by APD and TL would be the ideal power-test scheduling chart and, therefore, the ideal test scheduling profile. PDD is directly proportional to the accumulated power dissipation dispersion over the power-test scheduling chart, which is considered to be given by the area left unused inside the power-test rectangle having MPD and TL as sides. PDD is calculated as the difference between MPD and APD. RMS gives the root mean square value for the power dissipation distribution of a scheduling chart (see formula 3.1). All these power-test characteristics are used by PTS algorithms (see chapters 4 and 5) to analyze and improve their scheduling solutions.

$$PDD = \frac{MPD * TL - \sum_{i=0}^{N-1} P_i * T_i}{TL} = MPD - \frac{\sum_{i=0}^{N-1} P_i * T_i}{TL} = MPD - APD$$



(a) Power-Test Scheduling Chart (b) Characteristics of PTS Charts

Figure 3.8: PTS Charts of PTS-LS Approach Solution

3.3.6 Adapted Classical Scheduling Algorithms

A clear parallel between the HLS scheduling problem and the power-constrained test scheduling (PTS) problem can be noticed by looking at the similarities between the c-steps in HLS and the test sessions (subsessions) in PTS, between operations (HLS) and block-tests (PTS), and between hardware resource constraints (HLS) and power dissipation constraints (PTS). Therefore, there is an obvious coincidence between the process of assigning operations to c-steps (HLS scheduling) and the process of assigning block-tests to test (sub)sessions (PTS). The biggest achievement of the tree growing technique is that proven efficient HLS algorithms can be easily applied to the PTS problem modelled as an extended tree growing process.

A classical HLS register allocation algorithm such as the Left-Edge Algorithm (LEA) is firstly adapted in chapter 4. The approach, consisting of three algorithms, is named Left-Edge Algorithm Based PTS Approach (PTS-LEA). The HLS List Scheduling (LS) algorithm is employed in chapter 4 as a greedy List Scheduling Based PTS Algorithm (PTS-LS). In PTS-LS, the next test session expansion is carried out using a local priority function. The local priority function is given by a system of two lists. The first one is the list of block-tests left at a certain moment to be scheduled, which are ordered by the block-test mobility. The second one is the list of test (sub)sessions to be expanded that are ordered by their accumulated power dissipation. Local priority functions do not render all the time optimal solutions. Therefore, global priority functions are preferable. The main

difference between List Scheduling (LS) and Force-Directed Scheduling (FDS) approaches is the forecasting ability of their priority functions A *Force* function is employed in Force-Directed Scheduling Based PTS Algorithm (PTS-FDS) (see chapter 5) to steer the test scheduling so that the final solution has a more balanced test power-dissipation The Distribution Variance Based PTS Approach (PTS-DV), given in chapter 5, is aimed at achieving a balanced schedule by merely assessing the Power-Concurrency Distribution Graphs (PCDG) and the effect of block-test/test-subsession assignments Unlike the PTS-FDS approach, the time consuming stage of *Forces* calculations is avoided by using the Distribution Variance (DV) function, resulting in a more computationally efficient solution Finally, a mixed classical scheduling approach is also proposed in chapter 5 in order to improve the test concurrency having assigned power dissipation limits It is called Mixed LS - FDS Based PTS Approach (PTS-LSFDS) In this case a sequence of list and distribution-graph based scheduling algorithms, mentioned above, is adapted to tackle the PTS problem Firstly, a PTS-LS algorithm is run in order to rapidly achieve a test scheduling solution with a near-optimal test application time Then the power dissipation distribution of this solution is balanced by applying a PTS-FDS algorithm

In conclusion the work described in this thesis comes as an answer to the call for proposals enunciated in [CSA97] In the next two chapters algorithmic solutions are given to solve the PTS problem modelled in this chapter

Chapter 4

List Scheduling Based Approaches

In this chapter two list scheduling based PTS approaches are proposed and detailed The first approach is based on a left-edge algorithm and has three different implementations Then a list scheduling algorithm is implemented to solve the same PTS problem

4.1 Left-Edge Algorithm Based Approach

The PTS problem stated in [CSA97] is by far an NP-complete problem The goal in this section is to project the Left-Edge Algorithm (LEA) [HS71, KP87] onto the block-level test scheduling problem as a greedy approach The algorithm is supposed to be applied at the node level of the modular test hierarchy described in chapter 3 The LEA algorithm is well known for its application in channel-routing tools for physical-design automation [HS71] The goal of the channel routing problem was to minimize the number of tracks used to connect points on the channel boundary Two points on the channel boundary are connected with one horizontal (i e , parallel to the channel) and two vertical (i e , orthogonal to the channel) wire segments Since the channel width depends on the number of horizontal tracks used, the LEA algorithm tries to pack the horizontal segments into as few tracks as possible The LEA algorithm was first applied in the field of high-level synthesis to solve the register-allocation problem [KP87], in which variable lifetime intervals correspond to horizontal wire segments and registers to wiring tracks The input to the LEA algorithm, given in [KP87], was a list of variables to be allocated with registers A lifetime interval, with start time and end time, was associated with each variable

The list of variables was sorted on two keys the start time of the variables as the primary key to sort them in ascending order, and the end time as the secondary key to sort in descending order the variables with the same start time The algorithm had to run through the list of variables several times until all variables had been assigned to registers

The high level of similarities between the register allocation task and the kind of block-test scheduling problem tackled in this thesis led to the application of LEA algorithm to the Power-Constrained Block-Test Scheduling (PTS) problem In the PTS version of LEA algorithm (PTS-LEA) block-tests take the place of variables, while the test sessions (subsessions) are the former registers Thus, the input to the PTS-LEA algorithm is a list of tests to be allocated to different test sessions (subsessions) with the goal to minimize the total test application time, while keeping the power dissipation within the given limits The test resource compatibility is compiled for each test entry in the BTS table before the algorithm is run as in appendix 7 3 3

The "variables list" in this algorithm is a list of tests sorted by the following two keys their test application time (test length) is used as the primary key to sort the list in a descending order, and their estimated power dissipation is used as the secondary key to sort the tests with the same test application time in a descending order as well During each run through the list, tests are assigned to test sessions (subsessions) by generating other test subsessions in order to obtain better packing density Throughout the algorithm, the power dissipation accumulated along each test session (subsession) has to comply with the given power dissipation constraint There are three travel approaches that can be followed through the test session list The first one assumes a travelling down through the test list once for every generated gap (hatched or shaded, see subsection 3 3 3) until further merging cannot be performed anymore within the existent gaps (test subsessions) This approach uses exactly the list travelling approach from the LEA algorithm Every newly generated gap is considered a newly "allocated register" The second approach assumes the allocation of a new test session anytime a test from the list cannot be assigned to the existent gaps The new test session consists of the test that can not be accommodated within the existent gaps Every newly generated gap is considered a newly "allocated register" as well The third approach is a mixture of the two mentioned above The idea is to consider only the test sessions as "allocated

registers" That is, once a new test session has been allocated, a run through the full test list is carried out to check for the remaining tests that could be accommodated in the current set of gaps A new test session is allocated only when there are no more tests in the list compatible with the gaps existent in the current test session

4 1 1 Algorithm Pseudocode

PSEUDOCODE 1

◇sort all the tests by their mobility in two steps (test length, power dissipation),
◇initialize the *GrowingTree* **and the** *GapsList*,
◇while there are unscheduled tests { /* *BlockTestList* **is not empty*/**

- **if (***GapsList* **is empty) then {**
  - *CurTest* **= head of** *BlockTestList*,
  - **insert** *CurTest* **as the tail of** *GrowingTree* **roots, /*new test session*/**
  - **make** *CurTest* **"used",**
  - **remove** *CurTest* **from** *BlockTestList*,
  - **generate a** *TwinGap* **gap as the twin of** *CurTest*,
  - **insert** *TwinGap* **into** *GapsList*, **}/*if*/**
- **else {**
  - *CurGap* **= head of** *GapsList*,
  - *CurTest* **= head of** *Comp List*$_{CurGap}$,
  - **while** *CurGap* **is the head of** *GapsList* **AND** *CurTest* **did not reach the end of** *Comp List*$_{CurGap}$ **{**
    - **if (**$T_{CurTest} \leq T_{CurGap}$ **AND** $PD_{CurGap} + PD_{CurTest} \leq PD_{MAX}$ **AND** *CurTest* **NOT "used") then {**
      SCHEDULE(*CurTest,CurGap,GrowingTree,GapsList,* *BlockTestList***), /*schedules** *CurTest* **into the power-test scheduling chart and inserts it into the** *GrowingTree*, **marks** *CurTest* **"used"*/**
      break,}
    - **else** $CurTest = CurTest \longrightarrow next$, **/*next in the** *Comp List*$_{CurGap}$***/**
  - **} /*while*/**
  - **if (***CurGap* **is still the head of** *GapsList***) then**
    /*it means there are no compatible tests left for *CurGap* ***/**
    - **remove** *CurGap* **from the** *GapsList*,
- **} /*else*/ } /*while*/**

The pseudocodes of all three approaches are given below in this subsection The data structures used to implement them are the following the *GrowingTree* to model the ECT, the *GapsList* to model the list of potentially expandable gaps (shaded and hatched gaps), and the *BlockTestList* to keep the ordered but not yet merged tests *CurTest* is the test to be merged at each iteration *CurGap* is the gap under focus at each iteration in order to see whether it is expandable (compatible) with the *CurTest* In the pseudocode the term "used" means that the test has already been merged in the ECT *TwinGap* is the newly generated shaded gap at every iteration It will not be inserted in the *GapsList* anymore after its generation if its resulting compatibility list is null, i e it will not be an ETP *RestGap* is meant to keep the hatched gap generated at every iteration if it is not null, i e *CurTest* does not cover completely *CurGap*, that is $T_{CurGap} > T_{CurTest}$ Additionally, T_{node}, PD_{node} and $Comp\ List_{node}$ are, respectively, the test length, the power dissipation and the compatibility list of the node, which can be either a test or a gap If a new gap (test subsession) is generated inside the current one, the new one replaces the current gap in the *GapsList* and *GrowingTree*, and the procedure is repeated having a new *GapsList* The first approach of the PTS-LEA approach is given above

As can be figured out from the pseudocode itself, the algorithm is repeated until all the tests in the initial *BlockTestList* are scheduled in the ECT If the list of currently available gaps (*GapsList*) is empty then a new test session (and indirectly a new gap) is generated with the current test which is removed from the *BlockTestList* If the *GapsList* is not empty then the first gap in the list is taken for further expansion Its compatibility list is spanned starting with the test exhibiting the lowest mobility (long test length and high power dissipation) The first unscheduled yet test in the *BlockTestList* which turns out to be compatible with the current gap is scheduled in the *Growing Tree* generating two new gaps (twin and rest) *BlockTestList* and *GapsList* structures are updated then as well If the current gap turns out to be unexpandable it is removed from the *GapsList* and the process is repeated for the next gap in the list

The second approach of the PTS-LEA algorithm is following below Its algorithm is proven in chapter 6, by experimental results, to be the most different to the first approach out of the PTS-LEA algorithms If in the first approach the algorithm was trying to find a block-test in the *BlockTestList* to be accommodated in

the current gap, in the second pseudocode it is the other way around That is, for the current test the algorithm tries to find a suitable gap to accommodate it out of the already existent ones in the *GapsList* The algorithm is repeated for each test left in the *BlockTestList* The *SCHEDULE* procedure is the same with the one invoked in the first pseudocode and will be detailed after the third pseudocode

PSEUDOCODE 2

◇**sort all the tests by their mobility in two steps (test length, power dissipation),**
◇**initialize the** *GrowingTree* **and the** *GapsList*,
◇*CurTest* = **head of** *BlockTestList*
◇**while there are unscheduled tests {** /**BlockTestList* **is not empty***/

- *CurGap* = **head of** *GapsList*,
- **while** *CurGap* **did not reach the end of** *GapsList* **AND** *CurTest* **NOT "used" {**
  - **if** ($T_{CurTest} \leq T_{CurGap}$ **AND** *CurTest* **COMPATIBLE** $comp\ list_{CurGap}$ **AND** $PD_{CurGap} + PD_{CurTest} \leq PD_{max}$) **then {**
    - **SCHEDULE**(*CurTest*,*CurGap*,*GrowingTree*,*GapsList*, *BlockTestList*), /***schedules** *CurTest* **into the power-test scheduling chart and inserts it into the** *GrowingTree*, **marks** *CurTest* **"used"***/
    - **break,}**
  - **else** *CurGap* = *CurGap* $\longrightarrow$ *next*, /* **next in the** *GapsList**/
- } /***while***/
- **if** (*CurTest* **NOT "used") then {**
  - **insert** *CurTest* **as the tail of** *GrowingTree* **roots,** /***new test session***/
  - **make** *CurTest* **"used",**
  - **remove** *CurTest* **from** *BlockTestList*,
  - **generate a** *NewGap* **gap as the twin of** *CurTest*,
  - **insert** *NewGap* **as the tail of** *GapsList*, }
- *CurTest* = **the new head of** *BlockTestList*,

◇} /***while***/

The third pseudocode approach, detailed below, is a hybrid of the first two Here, both lists, *BlockTestList* and *GapsList*, are run through at the same time

For each test in the *BlockTestList* a gap is sought in the *GapsList* to be expanded with it

PSEUDOCODE 3

◇sort all the tests by their mobility in two steps (test length, power dissipation),
◇initialize the *GrowingTree* **and the** *GapsList*,
◇while there are unscheduled tests { /* *BlockTestList* **is not empty*/**

- **if (***GapsList* **is empty (initialized)) then {**
  - *CurTest* **= head of** *BlockTestList*,
  - **insert** *CurTest* **as the tail of** *GrowingTree* **roots, /*new test session*/**
  - **make** *CurTest* **"used",**
  - **remove** *CurTest* **from** *BlockTestList*,
  - **generate a** *TwinGap* **gap as the twin of** *CurTest*,
  - **insert** *TwinGap* **into** *GapsList*, **} /*if*/**
- **for all the tests left in the** *BlockTestList* **{**
  - *CurTest* **= head of** *BlockTestList*,
  - *CurGap* **= head of** *GapsList*,
  - **while** *CurGap* **did not reach the end of** *GapsList* **AND CurTest NOT "used" {**
    - **if (**$T_{CurTest} \leq T_{CurGap}$ **AND** *CurTest* **COMPATIBLE** *Comp List*$_{CurGap}$ **AND** $PD_{CurGap} + PD_{CurTest} \leq PD_{max}$**) then {**
      SCHEDULE(*CurTest*,*CurGap*,*GrowingTree*,*GapsList*, *BlockTestList*), **/*schedules** *CurTest* **into the power-test scheduling chart and inserts it into the** *GrowingTree*, **marks** *CurTest* **"used"*/ break,}**
    - **else** *CurGap* = *CurGap* $\longrightarrow$ *next*, **/*next in the** *GapsList*/*
  - **} /*while*/**
  - *CurTest* = *CurTest* $\longrightarrow$ *next*, **/*next in the** *BlockTestList*/*
- **} /*for*/**
- **update the** *GapsList*,

◇} /*while*/

The difference between the first and the third pseudocode is small, which will also be proved experimentally In the third pseudocode a new test session is generated only when there is no test in the *BlockTestList* to be accommodated in any of the available gaps (test subsessions) of the current test session to be expanded On the other hand, in the first pseudocode a new test session is generated only when there is no test to be accommodated in any of the available gaps (possibly from different test sessions) in the *GapsList* at each iteration However, the experiments prove in chapter 6 the similarity between these two approaches, which also proves the fact that usually most of the gaps available for expansion at each iteration belong to one test session This is not the case in the second pseudocode, which is experimentally proved to generate most of the time better results than the other two approaches This could be explained by the fact that the second approach can select the next test-to-test subsession assignment from a wider range of options (gaps belonging to different test sessions)

SCHEDULE(*CurTest*, *CurGap*, *GrowingTree*, *GapsList*, *BlockTestList*) {

- **generate** $RestGap = CurGap - CurTest$ **only if resulting** $T_{RestGap} \neq 0$,
  - $T_{RestGap} = T_{CurGap} - T_{CurTest}$,
  - $PD_{RestGap} = PD_{CurGap}$,
  - $Comp\ List_{RestGap} = Comp\ List_{CurGap}$,
- **generate** *TwinGap* **as the twin gap of** *CurTest*,
  - $T_{TwinGap} = T_{CurTest}$,
  - $PD_{TwinGap} = PD_{CurTest} + PD_{CurGap}$,
  - $Comp\ List_{TwinGap} = Comp\ List_{CurTest} \cap Comp\ List_{CurGap}$,
- **remove** *CurGap* **from the** *GrowingTree*,
- **insert** *CurTest* **and** *RestGap* **in place of** *CurGap*, /***if** $T_{RestGap}$ **is not zero***/
- **insert** *TwinGap* **into the** *GrowingTree* **as the unique offspring of** *CurTest*,
- **remove** *CurGap* **from the** *GapsList*,
- **INSERT**(*TwinGap*,*RestGap*,*GapsList*), /***inserts the newly generated gaps into the** *GapsList**/
- **make** *CurTest* **"used" (merged)**,
- **remove** *CurTest* **from** *BlockTestList*},

The pseudocode of the *SCHEDULE* is given above The *SCHEDULE* procedure schedules (merges) the current test into the *GrowingTree*, and subsequently removes it from the *BlockTestList* As can be seen in figure 3 5 from section 3 3 3, the merging step implies the generation of shaded and hatched gaps The hatched gap in the *SCHEDULE* procedure is called *RestGap* It represents the space (gap) left behind each merging step Therefore it inherits the accumulated power dissipation and compatibility list data from the former test subsession, but, on the other hand, the test length is the rest of test time left after the merging step If the test time left is null, there is no *RestGap* generated *TwinGap* represents the twin of the just merged test and, therefore, their test length is the same *TwinGap*'s power dissipation value is the sum of the power dissipation of the test scheduled in the former gap and the power dissipation of the gap *TwinGap*'s compatibility list is the intersection between the compatibility list of the previous ETP (gap) and the compatibility list of the just merged test If the resulting compatibility list is null it means that the newly generated *TwinGap* is born "saturated" and there is no point to take it into account for further expansion Thus, it is inserted into the *GrowingTree*, but not into the *GapsList* Another way of avoiding the useless assignation of *TwinGap* to the *GapsList* is to check the difference between the power dissipation constraint and *TwinGap*'s accumulated power dissipation The tests left unscheduled yet which have power dissipation characteristics higher than this difference are removed from *TwinGap*'s compatibility list If its compatibility list becomes empty after this step then the *TwinGap* is not inserted in the *GapsList* anymore

The *INSERT* procedure mentioned in the above pseudocode is meant to update the *GapsList* anytime another gap is generated and inserted There are five different approaches which can be used to insert the newly generated gaps into the *GapsList*

- the Most Recently Used (MRU) insertion - the newly generated gaps are inserted at the beginning (head) of the *GapsList* so that they would be the first to be processed at the subsequent iterations,
- the Least Recently Used (LRU) insertion - the newly generated gaps are inserted at the end (tail) of the *GapsList* so that they would be the last to be processed at the subsequent iterations,
- the In-Situ (INSITU) insertion - the newly generated gaps are inserted right

in the place of the gap expanded with the current test For example, if the current gap is at the i^{th} position in the *GapsList* then the newly generated gaps (rest and twin) take up the i^{th} and $i+1^{th}$ positions in the list,

- the Random (RAND) insertion - the newly generated gaps are randomly inserted in the *GapsList* This approach would resemble a simulated annealing approach if a smart search engine is employed,
- the ORD insertion - the newly generated gaps are kept ordered in the *GapsList* by their power dissipation

However it should be mentioned here that, basically, different power-test scheduling charts are obtained running the same algorithm by choosing different insertion approaches for the *GapsList* This is due to the fact that the *GapsList* gives the sequence of processing the gaps within the algorithm The MRU approach of the *INSERT* procedure is detailed next

INSERT(*TwinGap*, *RestGap*, *GapsList*) { /***inserts the newly generated gaps into the** *GapsList**/

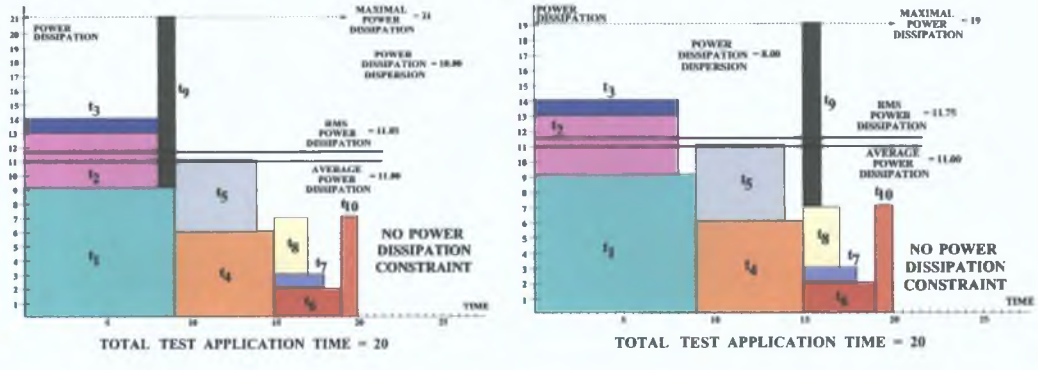
- **insert** *RestGap* **as the head of** *GapsList*, /***if** $T_{RestGap}$ **is not null***/
- **insert** *TwinGap* **as the head of** *GapsList*,
  /***if** *Comp List*$_{TwinGap}$ **is not null OR is not consisted of tests** t_i**, such that** $PD_{t_i} > PD_{max} - PD_{TwinGap}, \forall t_i \in$ *Comp List*$_{TwinGap}$***/**}

4 1 2 Algorithm Complexity

The complexities of all three LEA approaches is given next The complexity of the first pseudocode is $O(N^2)$ This is given by the two nested *while* loops, one to run through the *GapsList* and another one to run through the *BlockTestList* The number of tests in the *BlockTestList* is initially N, but it decreases each step by one Theoretically, the number of gaps in the *GapsList* can be at most N for all three pseudocodes, when either all the tests are run sequentially or they are run all concurrently and their test lengths are all different These two extremes hardly ever happen during the real test sets However, by the time the tests are scheduled the *BlockTestList* is empty Therefore the order of computational complexity of $O(N^2)$ is too pessimistic for this algorithm The complexity of the second pseudocode is $O(N^2)$ as well In this case this is given by the two nested *while* loops, one to travel

inside the *BlockTestList* and the other one to travel inside the *GapsList*. The arguments given above about the dimensions of the *BlockTestList* and *GapsList* are true here as well. The complexity of the third pseudocode is $O(N^3)$ because it contains three nested loops and each of them has a maximum length of N. N is the number of initial tests to be scheduled. The external *while* loop is used to repeat the solution search until all the tests are scheduled. The middle *for* loop travels through the *BlockTestList* for every newly generated test session. The inner most *while* loop travels through the *GapsList*. However, the dimension of *BlockTestList* is decreasing at each step and therefore $O(N^3)$ is a pessimistic order of computational complexity as well.

4.1.3 Test Scheduling Example



(a) First Approach - MRU Insertion (b) Second Approach - MRU Insertion

Figure 4.1: PTS Charts Without Power Constraints - 10 BTS

The following example should provide a deeper insight into the working and the results of these three algorithms. Figure 4.1 depicts comparatively the power-test scheduling results of the first and second pseudocodes generated without any power dissipation constraint for the BTS given next. Suppose the following ten tests (10 BTS) are to be scheduled under an average power constraint (PDC = 12) with the second algorithm using a MRU insertion approach and that their parameters are specified in the order: power dissipation, test length and their compatibility list.

$test_i(power\ dissipation, test\ length, \{compatibility\ list\})$

For simplicity reasons, the tests listed below are already ordered by test length

and power dissipation keys as depicted in figure 3 2 from chapter 3

$t_1(9, 9, \{t_2, t_3, t_5, t_6, t_8, t_9\})$
$t_2(4, 8, \{t_1, t_3, t_7, t_8\})$
$t_3(1, 8, \{t_1, t_2, t_4, t_7, t_9, t_{10}\})$
$t_4(6, 6, \{t_3, t_5, t_7, t_8\})$
$t_5(5, 5, \{t_1, t_4, t_9, t_{10}\})$
$t_6(2, 4, \{t_1, t_7, t_8, t_9\})$
$t_7(1, 3, \{t_2, t_3, t_4, t_6, t_8, t_9\})$
$t_8(4, 2, \{t_1, t_2, t_4, t_6, t_7, t_9, t_{10}\})$
$t_9(12, 1, \{t_1, t_3, t_5, t_6, t_7, t_8, t_{10}\})$
$t_{10}(7, 1, \{t_3, t_5, t_8, t_9\})$

The initial values for the data structures used inside the algorithm are $GrowingTree(GT) = 0$, $GapsList(GL) = 0$, $BlockTestList(BTL) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$, $CurrentTest(ct) = 0$, $CurrentGap(cg) = 0$, $TwinGap(tw) = 0$, $RestGap(rg) = 0$, while $PD_{max} = 12$ is the power dissipation constraint Since the number of tests to be scheduled is ten, there are ten main steps all together, which are depicted in figure 4 2

Step 1 The first test is selected from BTL ($ct = t_1$) in order to merge it to the GT but, since GL is initially empty, the first test session is generated (see the first step from figure 4 2) A twin gap tw_{t_1} is generated and inserted in GL so that $GL = \{tw_{t_1}\}$, while the t_1 node inserted into GT is shaded

Step 2 At the beginning of the second step $BTL = \{t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ and $GL = \{tw_{t_1}\}$ Thus, $ct = t_2$ and $cg = tw_{t_1}$ Even though ct and cg are compatible from the test length and the resource point of view, the accumulate power dissipation would be $PD_{t_2} + PD_{tw_{t_1}} = 13$, which is higher than the PD_{max} constraint Therefore, t_1 and t_2 cannot run in parallel and the solution is sequential as in the second step of figure 4 2 After this step $BTL = \{t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ and $GL = \{tw_{t_2}, tw_{t_1}\}$

Step 3 The next test to be scheduled is $ct = t_3$, while the head of GL is $cg = tw_{t_2}$ Because ct and cg are compatible from all points of view, they can be scheduled in parallel A rest gap rg is not generated here because $T_{t_3} - T_{tw_{t_2}} = 0$,

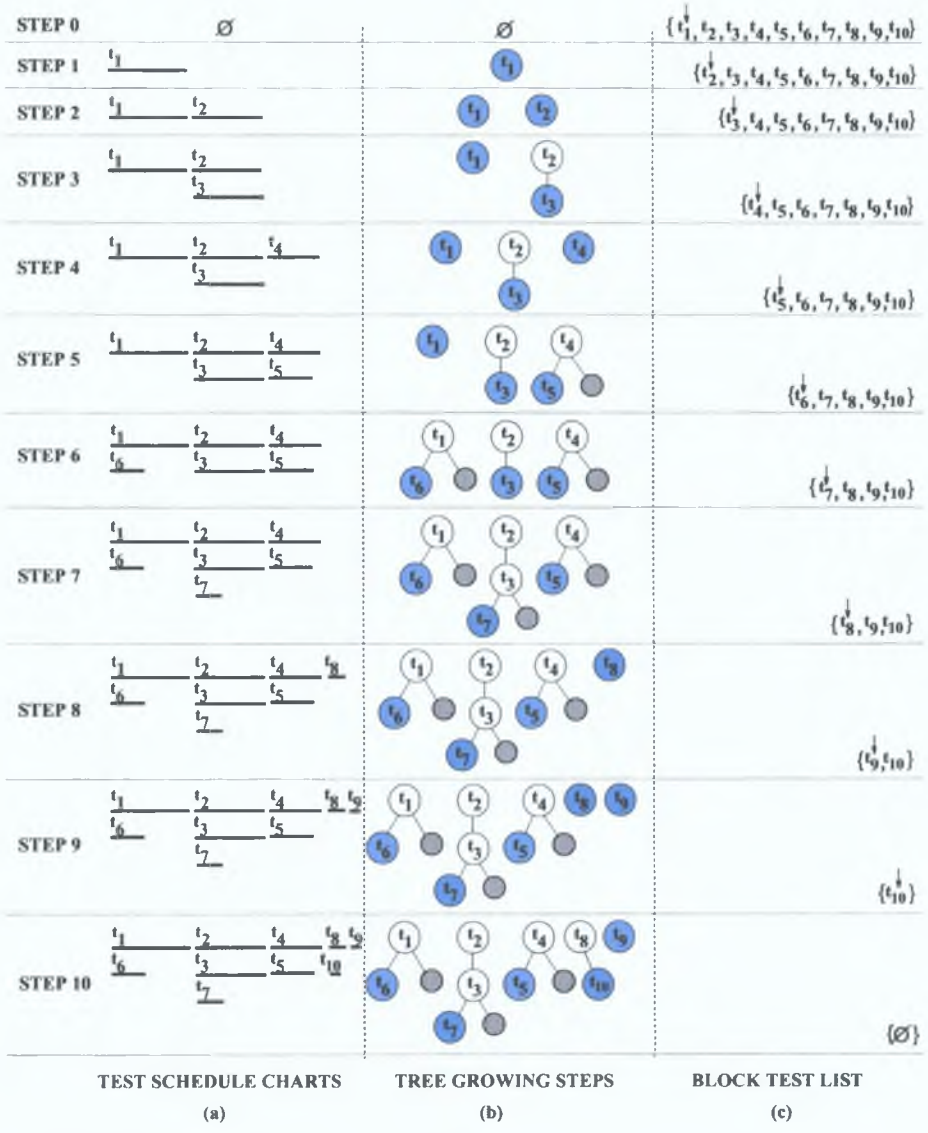


Figure 4.2: Tree Growing Steps Example (Second Pseudocode with MRU Insertion)

thus $t_2(tw_{t_2})$ and t_3 overlap completely. A twin gap $tw = tw_{t_{23}}$ is generated though with the following parameters: $T_{tw_{t_{23}}} = T_{t_3}$, $PD_{tw_{t_{23}}} = PD_{tw_{t_2}} + PD_{t_3} = 5$ and $Comp.List_{tw_{t_{23}}} = Comp.List_{tw_{t_2}} \cap Comp.List_{t_3} = \{t_1, t_7\}$. The new *GapsList* is $GL = \{tw_{t_{23}}, tw_{t_1}\}$, while the test list is $BTL = \{t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$.

Step 4. During the 4^{th} step, the test $ct = t_4$ has to be scheduled. Initially, $cg = tw_{t_{23}}$ is checked for compatibility with $ct = t_4$, but they are not compatible because $Comp.List_{tw_{t_{23}}} = \{t_1, t_7\}$, and $t_4 \notin Comp.List_{tw_{t_{23}}}$. Thus, the algorithm proceeds to the next gap in GL, that is $cg = tw_{t_1}$, but t_4 is not compatible with t_1 either. Therefore, a new test session is generated for t_4 and, consequently, a twin gap tw_{t_4} is

also generated, updating $GL = \{tw_{t_4}, tw_{t_{23}}, tw_{t_1}\}$ and $BTL = \{t_5, t_6, t_7, t_8, t_9, t_{10}\}$

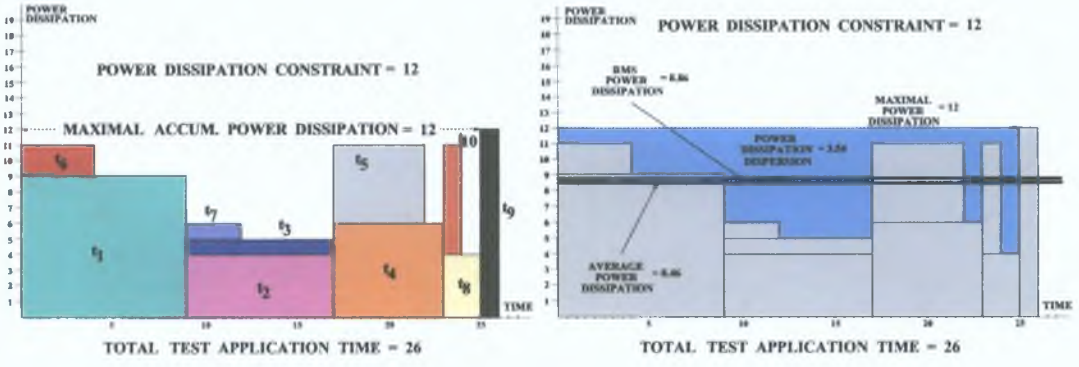
Step 5 For this step $ct = t_5$ and $cg = tw_{t_4}$, and they are compatible from all points of view Thus, a *RestGap* and a *TwinGap* have to be subsequently generated and then inserted into the *GapsList* and *GrowingTree* structures The *RestGap* rg_{t_4} has the following parameters $T_{rg_{t_4}} = T_{t_4} - T_{t_5} = 1$, $PD_{rg_{t_4}} = PD_{t_4} = 6$ and $Comp\ List_{rg_{t_4}} = Comp\ List_{t_4} = \{t_3, t_5, t_7, t_8\}$ The *TwinGap* $tw_{t_{45}}$ has the following parameters $T_{tw_{t_{45}}} = T_{t_5} = 5$, $PD_{tw_{t_{45}}} = PD_{tw_{t_4}} + PD_{t_5} = 11$ and $Comp\ List_{tw_{t_{45}}} = Comp\ List_{t_4} \cap Comp\ List_{t_5} = \{\oslash\}$, and ,therefore, it will not be inserted into the *GapsList* anymore Thus, after this step $GL = \{rg_{t_4}, tw_{t_{23}}, tw_{t_1}\}$ and $BTL = \{t_6, t_7, t_8, t_9, t_{10}\}$

Step 6 During this step the test $ct = t_6$ has to be scheduled The algorithm goes through the *GapsList* starting with $cg = rg_{t_4}$ (not compatible from the resource point of view), then $cg = tw_{t_{23}}$ (not compatible from the resource point of view), and ending with $cg = tw_{t_1}$ The last gap, $cg = tw_{t_1}$, is compatible with $ct = t_6$ A *RestGap* $rg = rg_{t_1}$ is generated having the following parameters $T_{rg_{t_1}} = T_{t_1} - T_{t_6} = 5$, $PD_{rg_{t_1}} = PD_{t_1} = 9$ and $Comp\ List_{rg_{t_1}} = Comp\ List_{t_1} = \{t_2, t_3, t_5, t_6, t_8, t_9\}$ The *TwinGap* $tw_{t_{16}}$ is generated with the following parameters $T_{tw_{t_{16}}} = T_{t_6} = 4$, $PD_{tw_{t_{16}}} = PD_{tw_{t_1}} + PD_{t_6} = 11$ and $Comp\ List_{tw_{t_{16}}} = Comp\ List_{tw_{t_1}} \cap Comp\ List_{t_6} = \{t_8, t_9\}$ Then both gaps will be inserted into the *GapsList* $GL = \{tw_{t_{16}}, rg_{t_1}, rg_{t_4}, tw_{t_{23}}\}$, while $BTL = \{t_7, t_8, t_9, t_{10}\}$

Step 7 In order to schedule $ct = t_7$, the algorithm has to find firstly a gap compatible with it t_7 is incompatible with $cg = tw_{t_{16}}$ and $cg = rg_{t_1}$ from test resources point of view t_7 is also incompatible with $cg = rg_{t_4}$ because the gap's test length is shorter than the test length of t_7 However, t_7 is compatible with $cg = tw_{t_{23}}$ Therefore, a *RestGap* $rg = rg_{t_{23}}$ is generated having the following parameters $T_{rg_{t_{23}}} = T_{tw_{t_{23}}} - T_{t_7} = 5$, $PD_{rg_{t_{23}}} = PD_{tw_{t_{23}}} = 5$ and $Comp\ List_{rg_{t_{23}}} = Comp\ List_{tw_{t_{23}}} = \{t_1, t_7\}$ Because both t_1 and t_7 have already been scheduled at this stage, and $rg_{t_{23}}$ is not compatible with any other tests, it would be pointless to insert this gap into the *GapsList* The *TwinGap* $tw_{t_{237}}$ has the following parameters $T_{tw_{t_{237}}} = T_{tw_{t_7}} = 3$, $PD_{tw_{t_{237}}} = PD_{tw_{t_{23}}} + PD_{t_7} = 6$ and $Comp\ List_{tw_{t_{237}}} = Comp\ List_{tw_{t_{23}}} \cap Comp\ List_{t_7} = \{\oslash\}$ Because its compatibility list is empty, it will not be inserted into the *GapsList* either After this step $GL = \{tw_{t_{16}}, rg_{t_1}, rg_{t_4}\}$, while $BTL = \{t_8, t_9, t_{10}\}$

Step 8. The $ct = t_8$ test cannot be scheduled in $cg = tw_{t_{16}}$ because the accumulated power dissipation would overflow, it cannot be merged with $cg = rg_{t_1}$ for the same reason, and cannot be scheduled in $cg = rg_{t_4}$ because the test length left $T_{rg_{t_4}} = 1$ is not enough for $T_{t_8} = 2$. Thus, a new test session t_8 is generated together with its twin gap tw_{t_8} ($PD_{tw_{t_8}} = PD_{t_8} = 4$). Consequently, $GL = \{tw_{t_8}, tw_{t_{16}}, rg_{t_1}, rg_{t_4}\}$ and $BTL = \{t_9, t_{10}\}$.

Step 9. Virtually the same happens during this step because the power dissipation of $ct = t_9$ is $PD_{t_9} = 12$, which is already equal to PD_{max} so that $ct = t_9$ could not be power dissipation compatible with any of the existing gaps: $tw_{t_8}, tw_{t_{16}}, rg_{t_1}, rg_{t_4}$. Therefore, a new test session t_9 is generated together with its twin gap tw_{t_9}. Consequently, $GL = \{tw_{t_9}, tw_{t_8}, tw_{t_{16}}, rg_{t_1}, rg_{t_4}\}$ and $BTL = \{t_{10}\}$.



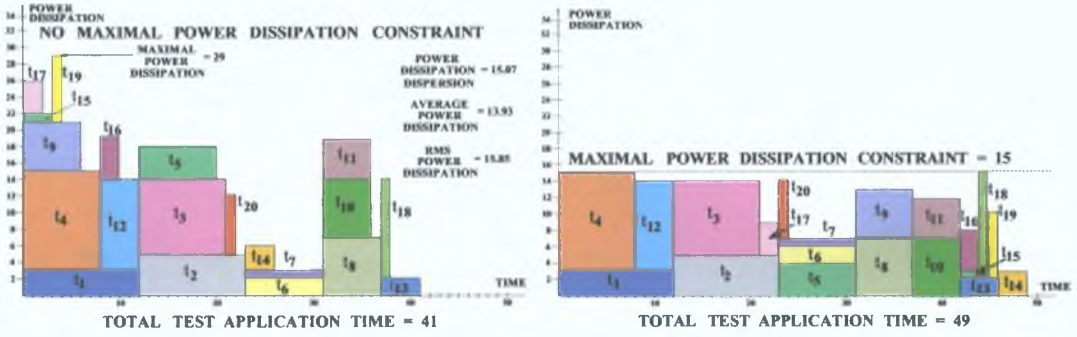
(a) Power-Test Scheduling Chart (b) Characteristics of the PTS Chart

Figure 4.3: PTS Charts With Power Constraints (PDC = 12) - 10 BTS

Step 10. During the last step $ct = t_{10}$ is scheduled in gap $cg = tw_{t_8}$, because it is not compatible with $cg = tw_{t_9}$ for the same power dissipation reasons. A *RestGap* $rg = rg_{t_8}$ is generated having the following parameters: $T_{rg_{t_8}} = T_{t_8} - T_{t_{10}} = 1$, $PD_{rg_{t_8}} = PD_{t_8} = 4$ and $Comp.List_{rg_{t_8}} = Comp.List_{t_8} = \{t_1, t_2, t_4, t_6, t_7, t_9, t_{10}\}$. Since all tests in the compatibility list have already been scheduled it would be pointless to insert this *RestGap* into the *GapsList*. The *TwinGap* $tw_{t_8\ 10}$ has the following parameters: $T_{tw_{t_8\ 10}} = T_{tw_{t_{10}}} = 1$, $PD_{tw_{t_8\ 10}} = PD_{tw_{t_8}} + PD_{t_{10}} = 11$ and $Comp.List_{tw_{t_8\ 10}} = Comp.List_{tw_{t_8}} \cap Comp.List_{t_{10}} = \{t_9\}$.

The final power-test scheduling chart for this example is given in figure 4.3. Figure 4.3(a) depicts the final power-test scheduling chart for the stepwise test scheduling example described above. Figure 4.3(b) depicts the power-test characteristics of the test scheduling solution given in figure 4.3(a).

Another example is given below where the number of tests is doubled. For this



(a) Without Power Constraints (b) With Power Constraints PDC = 15

Figure 4.4: PTS Charts of First PTS-LEA Approach - 20 BTS

example only the power-test charts of the final test scheduling solutions are given. This example is depicted in figure 3.7. The BTS is given below:

$test_i(power\ dissipation, test\ length, \{compatibility\ list\})$

$t_1(3, 12, \{t_4, t_5, t_8, t_9, t_{10}, t_{12}, t_{15}, t_{16}, t_{17}, t_{19}, t_{20}\})$

$t_2(5, 11, \{t_3, t_4, t_5, t_9, t_{12}, t_{13}, t_{14}, t_{17}, t_{19}, t_{20}\})$

$t_3(9, 9, \{t_2, t_5, t_7, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{17}, t_{18}\})$

$t_4(12, 8, \{t_1, t_2, t_7, t_9, t_{11}, t_{14}, t_{15}, t_{17}, t_{19}\})$

$t_5(4, 8, \{t_1, t_2, t_3, t_6, t_7, t_8, t_{12}, t_{15}, t_{17}, t_{18}, t_{20}\})$

$t_6(2, 8, \{t_5, t_7, t_9, t_{11}, t_{14}, t_{17}, t_{20}\})$

$t_7(1, 8, \{t_3, t_4, t_5, t_6, t_9, t_{12}, t_{14}, t_{15}, t_{16}, t_{18}, t_{19}, t_{20}\})$

$t_8(7, 6, \{t_1, t_5, t_9, t_{10}, t_{11}, t_{14}, t_{16}, t_{17}, t_{19}, t_{20}\})$

$t_9(6, 6, \{t_1, t_2, t_4, t_6, t_7, t_8, t_{11}, t_{12}, t_{15}, t_{17}, t_{19}\})$

$t_{10}(7, 5, \{t_1, t_3, t_8, t_{11}, t_{15}, t_{16}, t_{17}, t_{18}\})$

$t_{11}(5, 5, \{t_3, t_4, t_6, t_8, t_9, t_{10}, t_{14}, t_{16}, t_{18}, t_{20}\})$

$t_{12}(11, 4, \{t_1, t_2, t_3, t_5, t_7, t_9, t_{13}, t_{14}, t_{16}, t_{19}\})$

$t_{13}(2, 4, \{t_2, t_3, t_{12}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}\})$

$t_{14}(3, 3, \{t_2, t_3, t_4, t_6, t_7, t_8, t_{11}, t_{12}, t_{16}, t_{18}, t_{20}\})$

$t_{15}(1, 3, \{t_1, t_4, t_5, t_7, t_9, t_{10}, t_{13}, t_{16}, t_{17}, t_{18}\})$

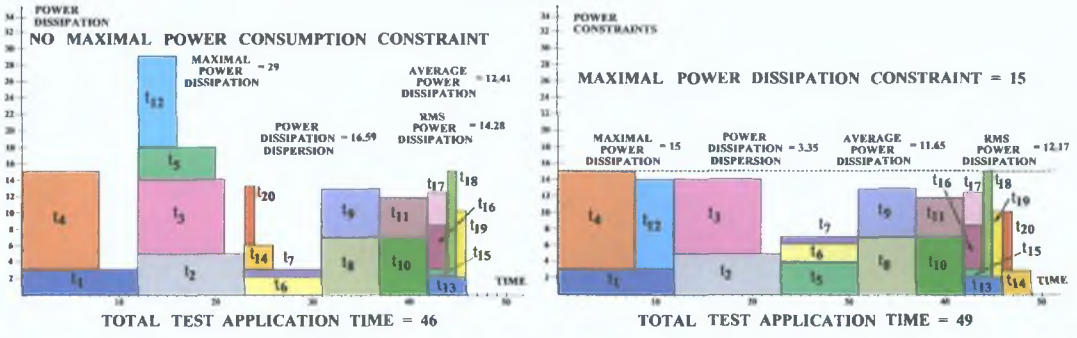
$t_{16}(5, 2, \{t_1, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{17}, t_{19}, t_{20}\})$

$t_{17}(4, 2, \{t_1, t_2, t_3, t_4, t_5, t_6, t_8, t_9, t_{10}, t_{13}, t_{15}, t_{16}, t_{18}, t_{19}, t_{20}\})$

$t_{18}(12, 1, \{t_3, t_5, t_7, t_{10}, t_{11}, t_{13}, t_{14}, t_{15}, t_{17}, t_{19}, t_{20}\})$

$t_{19}(8, 1, \{t_1, t_2, t_4, t_7, t_8, t_9, t_{12}, t_{13}, t_{16}, t_{17}, t_{18}, t_{20}\})$

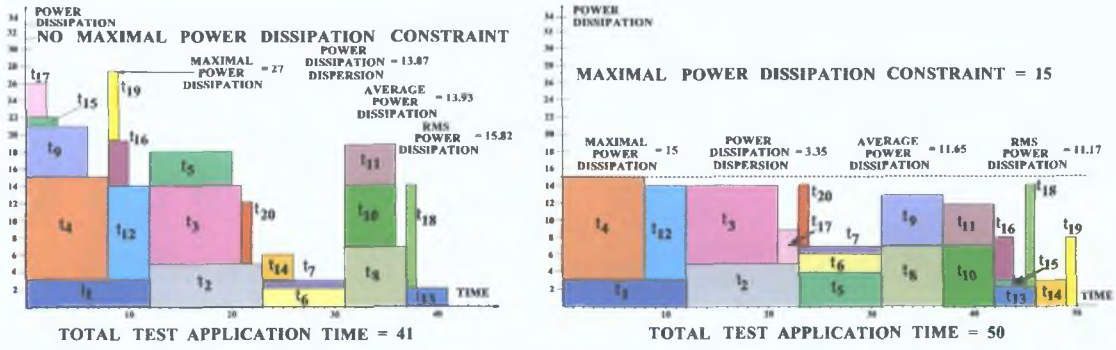
$t_{20}(7, 1, \{t_1, t_2, t_5, t_6, t_7, t_8, t_{11}, t_{14}, t_{16}, t_{17}, t_{18}, t_{19}\})$



(a) Without Power Constraints (b) With Power Constraints PDC = 15

Figure 4.5: PTS Charts of Second PTS-LEA Approach - 20 BTS

Figures 4.4, 4.5 and 4.6 are the scheduling results of the 20 BTS example using the first, the second and the third approaches, respectively, and employing the MRU gap insertion. Figures 4.4(a), 4.5(a), and 4.6(a) are the scheduling results without power constraints. Figures 4.4(b), 4.5(b), and 4.6(b) are the scheduling results with a maximum power constraints of 15. It can be seen that a power dissipation constraint forces the power-test scheduling to a more balanced power dissipation throughout the test application time, while obvious power dissipation spikes could be seen in figures 4.4(a), 4.5(a) and 4.6(a) due to the lack of power constraints. However, when there are power constraints, the total test application time increases as in figures 4.4(b), 4.5(b) and 4.6(b). Therefore, it is obvious that the power constraint is the only mechanism for cutting the spikes from the power-test scheduling charts, but this mechanism also increases the total test application time. Even though the test scheduling solutions of the first and third pseudocodes are not the same in these examples, chapter 6 will experimentally prove their similarity.



(a) Without Power Constraints (b) With Power Constraints PDC = 15

Figure 4.6: PTS Charts of Third PTS-LEA Approach - 20 BTS

4.2 List Scheduling Approach

This section describes the projection of the HLS version of the LS algorithm on the PTS problem This approach is therefore called PTS-LS In the List Scheduling (LS) algorithm [Dav81] a hardware constraint was specified and the algorithm attempted to minimize the total execution time by using a local priority function to defer operations when resource conflicts occurred The operations were sorted in topological order (top to bottom) using the precedences dictated by data and control dependencies in the Control/Data-Flow Graph (CDFG) The sorted operations were then iteratively scheduled into control steps The set of operations that could be placed in a control step (c-step) were then evaluated These operations were called *ready* operations If the number of ready operations of a single type exceeded the number of hardware modules available to perform them, then one or more operations had to be deferred The selection of the deferred operation was determined by a *local priority function* which depended on all operations that could be scheduled in the current control step The priority function was called *mobility* or *urgency* in [Dav81]

In terms of running through the list structures, the LS approach is similar to the PTS-LEA approach given in section 4 1 where an iterative tree growing technique was proposed to minimize the total test application time by deferring the tests when the power dissipation was exceeded during merging a new test to one of the Expandable Tree Path (ETP)s Actually, the first PTS-LEA pseudocode turned out to be almost the same as the PTS-LS algorithm In the PTS-LS algorithm, the tests are initially also ordered by their test length as the first key and then by power dissipation as the second key The sorted tests are then iteratively scheduled into test (sub)sessions (ETPs) and when the power dissipation is exceeded the tests to be currently scheduled are deferred to another test (sub)session (ETP)

Applying the LS approach on the PTS problem, a power dissipation constraint is also specified The local priority function is considered to be the test mobility modelled in the next section, while the compatibility relation has three components as well (power dissipation, test length and resource conflicts) A test is ready for scheduling into a test (sub)session if it is not in conflict (i e , it is compatible) with all the tests already scheduled in the aforementioned test (sub)session The algorithm attempts also to minimize the total test application time by using a local priority function to defer ready tests when compatibility conflicts occur as a result

of scheduling one of them In contrast to the HLS version of the LS approach, the surplus of ready tests in each iteration in the PTS-LS approach is not deferred directly They are indirectly deferred when they cannot be scheduled to the currently expanding test subsession due to the newly accumulated parameters of the new ETP after the scheduling of one of them Conflicts can arise when one has to schedule the remaining of the previously ready tests Firstly, the accumulated power dissipation in the new test subsession is getting close to the power dissipation constraint and, therefore, the previously ready tests might not be able to be accommodated anymore in the power dissipation space left after scheduling one ready test The power dissipation space left after scheduling the last test is the difference between the power dissipation constraint and the newly accumulated power dissipation in the expanded test subsessions Secondly, the test length of the newly generated test subsessions may be shorter than the test length of the previously ready tests Finally, the compatibility lists of the test subsessions generated after a test assignment may not still contain the previous ready tests It was observed that some ready tests could become incompatible (i e , become not ready) to the resulting test subsessions (ETPs) after the scheduling of one of them Thus, the fastest and most efficient way is to firstly schedule during each iteration in the current gap (ETP) the test "ready" for that gap with the lowest mobility Then, after the new ETPs are generated, new sets of ready tests are compiled for them And most probably some of the previously ready tests become not ready anymore Thus, the initially ready tests that could not be scheduled anymore in the new resulting ETPs are considered indirectly deferred during each iteration

Test Mobility Function

A parallel can be drawn between the test scheduling under power dissipation constraints and the operation scheduling under hardware resource constraints A mobility function can be defined for the power-constrained test scheduling problem as well Since every test t_i has a test length T_i and a power dissipation P_i, a local priority function called *test mobility* TM_i can be defined for each test t_i as the inverse of the product of the test length T_i and their power dissipation P_i $TM_i = \frac{1}{T_i * P_i}$ Intuitively, the probability of scheduling a test into a test subsession is higher the higher the test mobility TM_i is The mobility of a test t_i is inversely proportional to its dimensions In figure 3 2 the dimension of a test t_i is the area of the rectangle

having its test length T_i and its power dissipation P_i as sides The bigger this area the smaller the mobility By simply employing this test mobility function model the experimental results proved that it is unproductive This can be explained by the fact that in the tree growing approach the tests' length in an ETP is monotonously decreasing from root to leaf A test can not be scheduled in an ETP, where the leaf's test length is shorter than the test length of the test This is due to the fact that tests have to be scheduled in an ETP in the order of their test lengths Therefore, the runaround solution for the tree growing approach was to split test mobility into two, its test length component and its power dissipation component Thus, like in PTS-LEA, in PTS-LS tests are sorted in topological order by using their test length as primary key to order in descending order, and their power dissipation as secondary key, to order tests having the same test length in a descending order as well Therefore, the PTS-LS approach is similar to the PTS-LEA approach, the main difference being the ordering of gaps list by gaps' power dissipation

4 2 1 Algorithm Pseudocode

The data structures used in the pseudocode are the same as the ones used in the PTS-LEA approach The *Growing Tree* to model the ECT, *GapsList* to model the ordered list of potentially expandable gaps (shaded and hatched gaps), *BlockTestList* to keep the ordered tests *CurTest* is the test to be scheduled (merged) at a certain iteration *CurGap* is the gap under focus at a certain iteration to see whether it is expandable (compatible) with the *CurTest* In the pseudocode "used" means that the test has already been merged in the ECT *TwinGap* is the newly generated shaded gap at every iteration and it will not be inserted in the *GapsList* anymore after its generation, if its resulting compatibility list is null or the accumulated power dissipation is not less than the maximum power dissipation constraint minus the total power dissipation of all tests *RestGap* is meant to keep the hatched gap generated at every iteration if it is not null, i e *CurTest* does not cover completely *CurGap*

In the PTS-LS pseudocode the tests are initially sorted by their mobility and then stored in the *BlockTestList* At the same time the growing tree structure and the gaps list are initialized as well Two versions of the PTS algorithm can be proposed Firstly, the PTS-LS approach does not have an initial set of test sessions (roots in the growing tree) to start with and builds it during the algorithm's

execution In this case the growing tree and the gaps list are initially set to the first (longest test length) test from the already sorted *BlockTestList* Secondly, the PTS-LS approach can be applied on a power-test schedule generated by a first run of PTS-LS In this case only the set of roots from the above mentioned schedule is used as the set of initial test sessions on which further power-test expansion optimization will be carried out Thus, the growing tree and the gaps list are set to this roots list This approach is called Squared List Scheduling PTS Algorithm (PTS-LS^2) because the PTS-LS algorithm is run twice on the BTS

Throughout the algorithm the *GapsList* is kept ordered in ascending order by the gap's (test subsessions) accumulated power dissipation as the first key, and also in ascending order, by their test length, for the gaps which have the same accumulated power dissipation This is carried out in order to select for further expansion at each iteration the test subsessions which consume less power Thus, there is a higher probability of decreasing the power dissipation difference between the test subsessions (ETPs) of the growing tree Consequently, the power dissipation would be more balanced

The algorithm is iterative Every iteration looks for the test with the longest test length and the lowest mobility to be scheduled in the test subsession (ETP) with the lowest accumulated power dissipation *CurGap* is the first gap from the *GapsList*, which is the one with the lowest accumulated power dissipation Next, *CurGap*'s induced compatibility list is run through from the test with the lowest mobility to the one with the highest mobility The first test (*CurTest*) which is assignable (compatible from all points of view) to the *CurGap* is scheduled in it Another pair of gaps (twin and rest) is generated and they have to be inserted in the *GapsList* so that the list is still ordered by gaps' accumulated power dissipation If no test in the *CurGap*'s compatibility list can be scheduled then the *CurGap* is removed from the *GapsList* and the algorithm continues with the next gap from the list, which is actually the new head of the list When all the gaps from the *GapsList* are removed a new test session is generated exactly like at the beginning of the algorithm during the initialization of the *GapsList* Namely, the *GapsList* is set to *TwinGap* of the first (the one with the lowest test mobility among those tests with the longest test length) test still in the sorted *BlockTestList* The *GrowingTree* expands with a new test session generated to run the same test The SCHEDULE procedure in the above pseudocode is similar to the one used by the PTS approach, but in this

implementation the gaps are kept ordered in the *GapsList* by their accumulated power dissipation

PSEUDOCODE of PTS-LS ALGORITHM

◇**sort all the tests by their mobility in two steps (test length, power dissipation),**
◇**initialize the** *GrowingTree*, **the** *BlockTestList* **and the** *GapsList*,
◇**initialize the time constraint to the initial number of test subsessions,**
-*GapsList* **is ordered by gaps' power dissipation,**
-**while (there are unscheduled tests) do** { /**BlockTestList* **is not empty***/

- **if (***GapsList* **is empty) then {**
  - *CurTest* = **head of** *BlockTestList*,
  - **insert** *CurTest* **as the tail of** *GrowingTree* **roots, /*new test session*/**
  - **make** *CurTest* **"used",**
  - **remove** *CurTest* **from** *BlockTestList*,
  - **generate a** *TwinGap* **gap as the twin of** *CurTest*,
  - **insert** *TwinGap* **into** *GapsList*, } /***if***/
- **else {**
  - *CurGap* = **the head of** *GapsList*,
  - *CurTest* = **the head of** *Comp List*$_{CurGap}$,
  - **while(**$T_{CurTest} > T_{CurGap}$ **OR** $PD_{CurGap} + PD_{CurTest} > PD_{max}$ **OR** *CurTest* **"used") do**
    - *CurTest* = *CurTest* $\longrightarrow$ *next*, /***next in the** *Comp List*$_{CurGap}$*/
  - **if (a** *CurTest* **was found AND** $T_{CurTest} \leq T_{CurGap}$ **AND** $PD_{CurGap} + PD_{CurTest} \leq PD_{max}$ **AND** *CurTest* **NOT "used") then {**
    - **SCHEDULE(***CurTest*, *CurGap*, *GrowingTree*, *GapsList*, *BlockTestList***),**
    - **break, }**
  - **else remove** *CurGap* **from** *GapsList*,

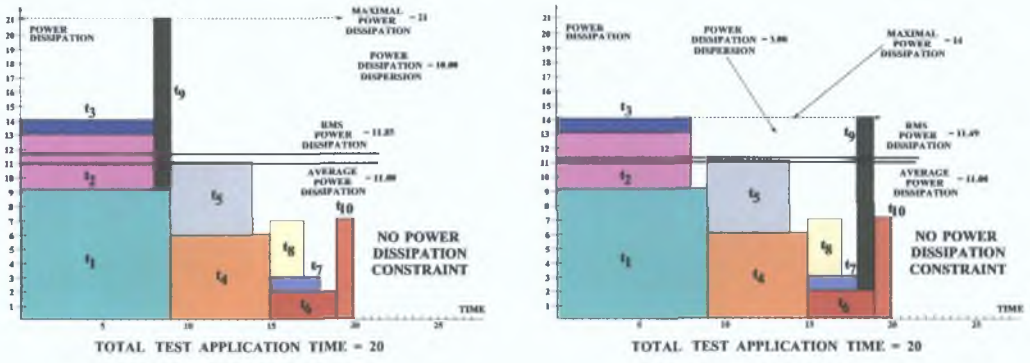
◇**}/*while*/**

4 2 2 Algorithm Complexity

The complexity of this approach is $O(N^2)$ because it is structured on two nested while loops (one to run through the *BlockTestList* and another one to run through the *GapsList*) which are dependant on the initial number of tests in the

BlockTestList. However, as for the PTS-LEA algorithms described in the previous section, the above order of computational complexity is rather pessimistic since the number of gaps in the *GapsList* virtually never gets to a degree of N, which is the initial number of tests (see subsection 4.1.2).

4.2.3 Test Scheduling Example

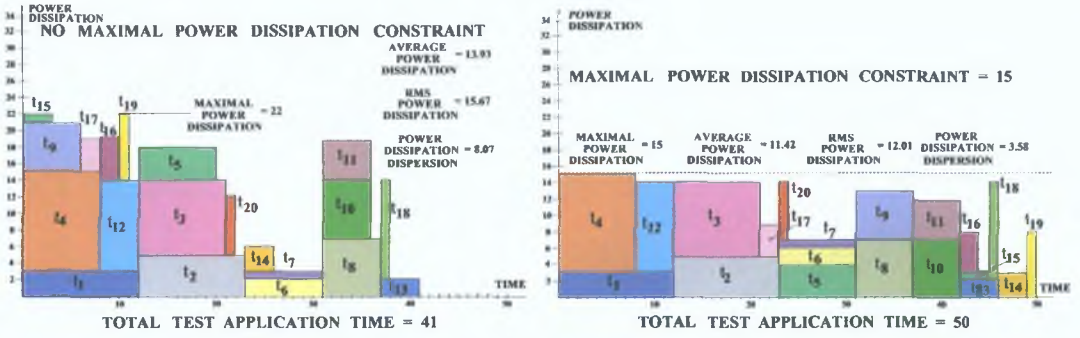


(a) PTS-LS Algorithm Chart (b) PTS-LS^2 Algorithm Chart

Figure 4.7: PTS Charts of LS-based Algs. No Power Constraints - 10 BTS

The two examples in subsection 4.1.3 are experimented in this subsection with the LS-based approaches. For the first example (10 BTS example) figure 4.7(a) depicts the PTS solution given by the PTS-LS algorithm. It is similar to the ones given by the first and third PTS-LEA pseudocodes. Figure 4.7(b) depicts the improved solution after the second run of the PTS-LS algorithm. It can be seen that test t_9 is shifted and accommodated in the test session running test t_6. This change generates a power-test scheduling chart which exhibits balanced power dissipation characteristics for the same test application time. Both power-test charts in figure 4.7 are generated without power constraints. As for the solutions generated with power dissipation constraints, this very small BTS example is not the best example because the solutions for a small number of tests (thus a small solution space) are similar under power dissipation constraints. The second example is run for the 20 BTS initially given in subsection 4.1.3, and is commented on in order to point out the advantages and disadvantages of the PTS-LS and PTS-LS^2 approaches.

This second BTS example is meant to provide the reader with a better understanding of the results generated by the PTS-LS-based algorithms. In figures



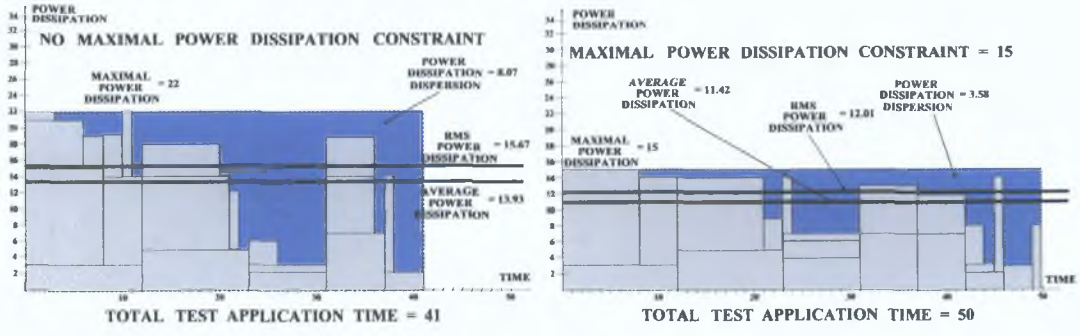
(a) Without Power Constraints (b) With Power Constraints PDC = 15

Figure 4.8: PTS Charts of PTS-LS Approach - 20 BTS

4.8 and 4.9 the results of the PTS-LS algorithm are given, both, without (figures 4.8(a), 4.9(a)) and with (figures 4.8(b), 4.9(b)) power dissipation constraints ($PD_{max} = 15$). Figure 4.8 depicts the power-test scheduling charts, while figure 4.9 exhibits the power-test characteristics of the aforementioned charts.

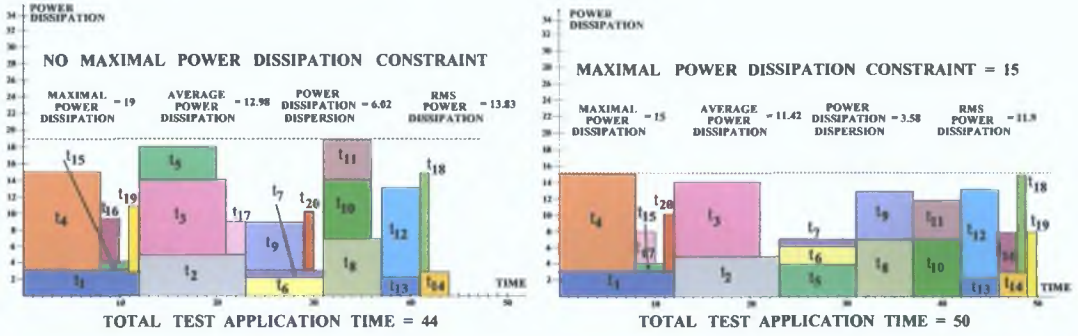
It can be seen in figures 4.8(b) and 4.9(b) that a tighter power dissipation constraint forces the power-test scheduling results generated by the PTS-LS algorithm to exhibit a more balanced power dissipation characteristic throughout the test application time. At the same time obvious power dissipation spikes could be seen in figures 4.8(a), 4.9(a) due to the lack of power dissipation constraints. This means the power dissipation is less balanced when the PTS-LS algorithm is loosely constrained. This is a big disadvantage of the PTS solutions generated by the PTS-LS algorithm. Intuitively, when there are tighter power dissipation constraints the total test application time increases.

On the other hand, it can be seen that the PTS charts given in figure 4.10 are



(a) Without Power Constraints (b) With Power Constraints PDC = 15

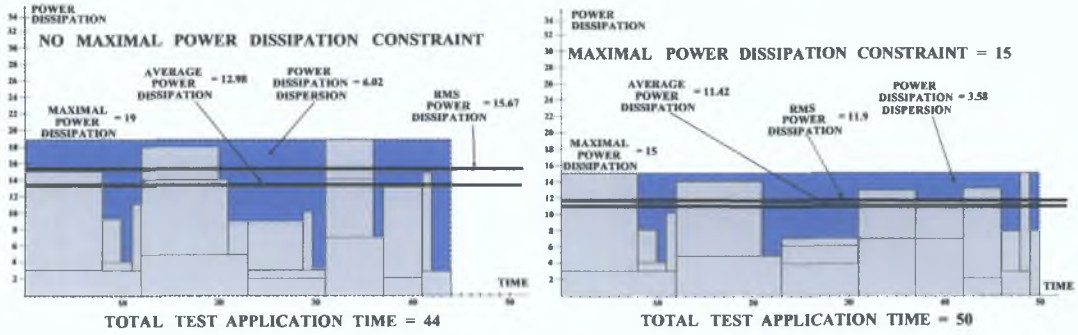
Figure 4.9: PTS Charts' Characteristics of PTS-LS Algorithm - 20 BTS



(a) Without Power Constraints (b) With Power Constraints PDC = 15

Figure 4.10: PTS Charts of PTS-LS^2 Approach - 20 BTS

improved by a second run of the PTS-LS algorithm. Thus, the PTS-LS^2 approach gives a more balanced power-test scheduling chart even without power dissipation constraints. However, the disadvantage of it is that the total test application time also increases even for loose power dissipation constraints. This is due to the greedy characteristic of the PTS-LS algorithm. This characteristic is rendered by the local priority function which steers these algorithms. The PTS approaches proposed in the next chapter can overcome this problem by employing a global priority function.



(a) Without Power Constraints (b) With Power Constraints PDC = 15

Figure 4.11: PTS Charts' Characteristics of PTS-LS^2 Approach - 20 BTS

Chapter 5

Distribution-Graph Based Approaches

A distribution-graph based approach is adapted in this chapter to tackle the problem of unequal-length block-test scheduling under power dissipation constraints As in the previous chapter, the extended tree growing technique is also used in combination with classical HLS scheduling algorithms in order to improve the test concurrency having assigned power dissipation limits The goal is to achieve a balanced test power dissipation by employing a distribution-graph based global priority function like *force* [PK89] or *distribution variance* [KAHA97]

As presented in the previous chapter, with the LS-based algorithms the tests are initially ordered by their *test mobility* before being scheduled The sorted tests are then iteratively scheduled into the available test (sub)sessions The next test (sub)session expansion is carried out using the test mobility as a *local priority function* Local priority functions do not render all the time optimal solutions Therefore, global priority functions are preferable The main difference between the LS algorithm and the Force-Directed Scheduling (FDS) algorithm is the forecasting ability of their priority functions For example, the FDS algorithm given in [PK89] uses a *global priority function* called *Force* The time consuming stage of *Force* calculations can be avoided by using a Distribution Variance (DV) function, resulting in a more computationally efficient solution The so-called Mean Square Error (MSE) function employed in [KAHA97] is also a distribution-graph based global priority function

Efficient distribution-graph based algorithms have been presented in [PK87,

PK89, KAHA97] In these algorithms, a global time constraint was specified and the algorithms attempted to minimize the number of resources required to meet that constraint The time constraint was typically given in terms of the number of control steps allowed for the execution of a specified behaviour The main strength of the algorithm is the use of a global measure of concurrency to guide the scheduling process In this chapter, the *Force* function is firstly employed to steer the assignment of tests to test (sub)sessions The selection of a test subsession in which a selected test will be placed is based on the objective to achieve a balanced distribution of power dissipation and test concurrency Then, in the following section the DV function is employed to achieve a balanced schedule merely by assessing the Power-Concurrency Distribution Graphs (PCDG) and the effect of test/test-subsession assignments Finally, the aforementioned classical scheduling algorithms are combined in order to further improve the test concurrency Actually, for this last case a sequence of list and distribution-graph based scheduling algorithms is adapted to tackle the PTS problem

5.1 Force-directed Scheduling Approach

The intent of the FDS algorithm in [PK89] was to reduce the number of required functional units (registers and buses) by *balancing the concurrency of the operations assigned to them*, but without lengthening the total execution time In contrast to other constructive approaches like the LS-based ones, the FDS algorithm does global analysis of the operations and control steps when selecting the next operation to be scheduled and therefore it is more computationally expensive but renders better results

Concurrency balancing helps to achieve high utilization - or low idle time - of structural units, which in turn minimizes the number of units required This idea is adapted here to carry out a power-constrained test scheduling The objectives here are to achieve a test concurrency and power dissipation balance along with test application time minimization, given certain power dissipation limits Thus, the operations to be scheduled in High-Level Synthesis (HLS) are equivalent to the tests to be scheduled in Power-Constrained Block-Test Scheduling (PTS), the control steps (or c-steps in HLS scheduling) are similar to test (sub)sessions (PTS),

and the hardware resource constraints (HLS) are similar to power dissipation constraints (PTS) A tree growing technique [JPP89] is also used together with the FDS algorithm in order to generate and keep track of the test subsessions of the Extended Compatibility Tree (ECT) It can be noticed that the set of Expandable Tree Paths ETPs in the growing tree changes throughout the algorithm's execution because they are expanded and then replaced with new test subsessions (hatched and shaded nodes) Consequently, the set of test subsessions is dynamic for the PTS-FDS approach, while their equivalent in FDS (the set of c-steps) is static This is one of the three main differences that exist between FDS and PTS-FDS

The PTS-FDS algorithm is iterative, with one test scheduled in each iteration It has been seen in section 4 2 when it came to talk about the *test mobility* function, that the order of test assignments affects the scheduling process That is due to the fact that a test can not be scheduled in an ETP, where the test subsession's test length is shorter than test's test length The following approach is adapted to obtain optimum results First, all tests are ordered by their test mobility as in section 4 2 At each iteration the PTS-FDS algorithm schedules the first test in the sorted list The selection of the test subsession in which this test will be placed is based on the objective of achieving a balanced distribution of power dissipation and test concurrency in each test subsession This is achieved by employing a *Total Force* global priority function This function is obtained by using the three step algorithm proposed with the FDS approach [PK89] Next, the adapted FDS algorithm to tackle the PTS problem is presented by analogy with the original HLS version of FDS algorithm

Determination of time frames (test subsession sets)

In the FDS approach, the first step consists of determining the time frames of each operation by evaluating the As Soon As Possible (ASAP) and As Late As Possible (ALAP) schedules The time frames are contiguous in FDS and represent the sequence of c-steps where operations could be scheduled On the other hand, in the PTS-FDS approach the time frame of a test is the set of test subsessions (ETPs) where the test can be placed The ETPs expandable at a certain moment with a test do not have to be adjacent and, therefore, the time frame of a test in PTS-FDS is not or does not have to be contiguous This is the second outstanding difference between the FDS and PTS-FDS approaches

The objective of FDS is to achieve a balanced operation concurrency In FDS the probability of assigning an operation to any of the c-steps in its time frame is assumed to be equal For example, if an operation's time frame contains 3 c-steps, then the probability of assigning this operation to any of the 3 c-steps is 1/3 On the other hand, the goal of PTS-FDS is mainly to balance the power dissipation and, indirectly, the test concurrency, while keeping the test application time as tight as possible In order to balance the power dissipation, the heuristic should avail at each iteration of a means of predicting the power dissipation distribution along the test application time in the next iteration Each test t_i to subsession ts_j assignment affects the power dissipation distribution in a different way Therefore, at each iteration this prediction is mandatory in order to select the test-to-test subsession assignment that exhibits the most balanced prediction of power dissipation distribution This will help the algorithm to choose the assignment which will most probably balance the power dissipation distribution This prediction is called in this thesis Power-Concurrency Distribution Graphs (PCDG) because it is meant to give a measure of both power dissipation and, indirectly, test concurrency distribution

The probable power dissipation distribution of the new schedule after assigning test t_i to subsession ts_j is obtained by adding the probable power dissipation contribution of test t_i to the power dissipation already accumulated in test subsession ts_j A simple uniform probability was used by FDS to give the probability of assigning an operation to a c-step In PTS-FDS this is replaced by the probability of t_i's power dissipation that would be added to the power dissipation accumulated in test subsession ts_j, if test t_i was assigned to subsession ts_j Intuitively, this probability has to be proportional to t_i's power dissipation and the probability of assigning test t_i to subsession ts_j Therefore, it is calculated as the product between t_i's power dissipation and its assignment probability $Assign_{Prob}(t_i, ts_j)$

$$PCDG(t_i, ts_j) = P_{t_i} * Assign_{Prob}(t_i, ts_j), \tag{5 1}$$

The assignment probability $Assign_{Prob}(t_i, ts_j)$ is proportional to the uniform probability of assigning test t_i to one of the test subsessions from its current time frame At the same time the assignment probability should have a component to give a prediction of the compatibility between test t_i and other tests in the compatibility list of test subsession ts_j Unfortunately, the test clustering related problems are proven to be NP-complete (see subsection 2 5 5) Therefore, only a probabilistic

measure of the test compatibility between tests can be employed To model comprehensively even this probabilistic measure is a difficult task in itself [MD, JY00] It is considered a general problem of random graph theory and combinatorics This conclusion was drawn in collaboration with a group of mathematicians The task of modelling probabilistically the test compatibility has proven to be very complex, even though eventually some formulas have been generated However, these formulas are very complex themselves with a lot of factorial (computationally heavy) components Instead, a very simplified version of these formulas is used in the approach proposed in this chapter, and it is described below Having this prediction problem sorted out by probabilistic means, the assignment probability $Assign_{Prob}(t_i, ts_j)$ of a test t_i to a test subsession ts_j is defined as the product of

- compatibility probability $Comp_{Prob}(t_i, ts_j)$ between t_i and other tests assignable to the same test subsession ts_j (tests in the compatibility list of ts_j),
- uniform probability $Prob(t_i)$ of assigning test t_i to one of its time frame's subsessions This probability is the same as the one employed in FDS

$$Assign_{Prob}(t_i, ts_j) = Comp_{Prob}(t_i, ts_j) * Prob(t_i) \tag{5 2}$$

The compatibility probability between test t_i and subsession ts_j is simplified to the probability that test t_i would be compatible with other tests in ts_j's compatibility list For instance, take the partial schedule in figure 5 1 as an example It is the partial schedule obtained in figure 3 5 from subsection 3 3 3 after the merging step Suppose now that test t_i is next to be scheduled Say it exists in the compatibility lists of test subsessions tw_{12}, tw_{13} and rg_1, but not in tw_{14}'s compatibility list The question is which test subsession out of tw_{12}, tw_{13} and rg_1 is the best to be further considered for expansion with t_i? The more

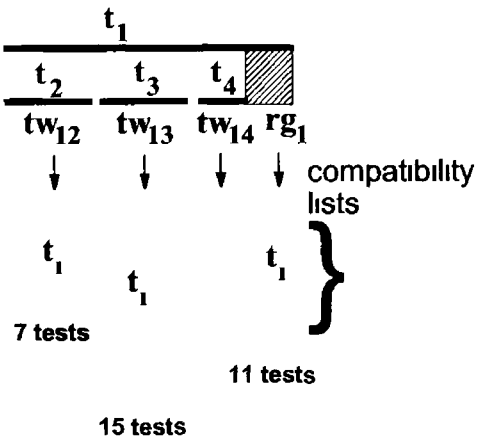


Figure 5 1 Partial Test Schedule

tests in the compatibility lists, the harder it is to answer this question Intuitively, in order to know exactly which test subsession ts_j should be expanded at this stage with t_i, all the test compatibility cliques including t_i should be generated for each test subsession compatibility list This way it would be known which of the other tests from ts_j's compatibility list could be further scheduled in the twin gap of t_i and the rest gap of ts_j after t_i's scheduling in ts_j Unfortunately, the clique partitioning problem is NP-complete and this algorithmic option is out of the question Moreover, one scheduling decision taken at each iteration based on the set of partitioned cliques would not guarantee the final optimal solution On the other hand, the goal of PTS is to minimize the total test application time while keeping the power dissipation under the given limit Therefore, intuitively, the goal would be here, that by scheduling t_i in any of the available test subsessions, to leave as many as possible open options for further expansion in the existing test subsessions Otherwise, if test subsessions are "saturated" prematurely with the tests scheduled at each iteration, then the rest of the tests in the compatibility list will be forced to generate new test sessions This way the number of test sessions (roots) that would be generated increases and so does the total test application time This leads to the conclusion that test t_i should be scheduled to the test subsession that has in test subsessions' compatibility list the highest number of tests compatible with t_i

For example, suppose now that in figure 5 1 tw_{12} contains 7 tests, tw_{13} contains 15 tests, and rg_1 contains 11 tests Intuitively, it can be stated that there is a higher probability of finding tests compatible with t_i in tw_{13} because tw_{13} has the highest number of tests in the compatibility list The following formula is employed as an approximation to measure the probability that t_i is compatible with other tests in ts_j's compatibility list

$$Comp_{Prob}(t_i, ts_j) = \begin{cases} 0 & if\ t_i\ INCOMPATIBLE\ TO\ ts_j, \\ 1 - \frac{C_{N\ -\ N_{incomp}}^{K-1}}{C_{N-1}^{K-1}} & if\ K \neq 1\ AND\ N_{incomp} \geq K-1, \\ 1 & if\ K = 1\ OR\ N_{incomp} < K-1 \end{cases} \quad (5\ 3)$$

where N_{incomp} is the number of tests from t_i's *incompatibility* list, N is the total number of tests left unscheduled and K is the number of tests compatible with test subsession ts_j The incompatibility list of t_i includes the other tests incompatible with it from all points of view, that is power dissipation, test length and test resources The way the above formula is given is equivalent to calculating the improbability that test t_i would be incompatible with all the other tests in ts_j's

compatibility list The reasoning behind formula 5 3 is simple Firstly, the probability that t_i would be incompatible with all the other tests in ts_j's compatibility list is calculated This probability corresponds to $\frac{C_{N_{incomp}}^{K-1}}{C_{N-1}^{K-1}}$ in formula 5 3 Then, the inverse of this probability is calculated to get the probabilistic formula of test compatibility This inverse gives the probability that t_i would be compatible with at least one test in ts_j's compatibility list

At each iteration it is known how many unscheduled tests are incompatible with t_i (N_{incomp}), and how many tests are in the compatibility list of subsession ts_j (K) The question is, what is the probability that the rest of $K-1$ tests from ts_j's compatibility list are all incompatible with t_i? To answer this question, two other questions have to be answered Firstly, in how many ways $K-1$ tests can be chosen out of the $N-1$ (without test t_i) unscheduled tests? The answer is in $(K-1)$-combinations of $N-1$ tests Secondly, in how many ways $K-1$ tests can be chosen to be all incompatible with test t_i? The answer is in $(K-1)$-combinations of N_{incomp} tests known to be incompatible with t_i Knowing the answer to the latter two questions, the probability that test t_i would be incompatible with all the other tests in ts_j's compatibility list can be calculated by $\frac{C_{N_{incomp}}^{K-1}}{C_{N-1}^{K-1}}$ Then, the improbability of this event is easily calculated ($improbability_{event} = 1 - probability_{event}$)

Intuitively, the probability in formula 5 3 is 1 when t_i is the only test in ts_j's compatibility list, that is, $K = 1$ This is also equivalent with saying that t_i has the highest chance of being scheduled to ts_j without affecting the time frame of any other test The probability is also 1 when $N_{incomp} < K - 1$ That is, t_i is known to be incompatible with a number of tests (N_{incomp}) and this number is less than the number of other tests in ts_j's compatibility list Thus, t_i can not be incompatible with all other tests in ts_j's compatibility list

It is important to say that mathematically speaking, the above assumptions are an oversimplification of the compatibility problem described here That is, in the presented compatibility problem, the N_{incomp} values above are not independent outcomes and the compatibility probability problem becomes a lot more complex Mathematically speaking, if an event must result in one of the mutually exclusive (independent) outcomes $O_1, O_2, \quad , O_n$ with probabilities $P_1, P_2, \quad , P_n$, respectively, than the probability of this event is $P_{event} = P_1 * P_2 * \quad * P_n$ This is not the case in the test compatibility problem described in this thesis For example,

say there are $t_1, t_2, ..., t_k$ unscheduled tests at a certain iteration. Say the probability of the outcome that t_u is incompatible with all the other tests is P_u, and the probability of the outcome that t_v is incompatible with all the other tests is P_v. For the test compatibility problem described in this thesis, the probability P_{uv} of the outcome that t_u and t_v are incompatible with all the other tests is not equal to the product of P_u and P_v probabilities. Therefore, formula 5.3 is far from being exact when the incompatibility relations between all the tests in the Block-Test Set (BTS) are simultaneously taken into account. However, the higher the number of assumptions/relations included in the probability formula, the more complex it gets. This work proves that formula 5.3 is a good trade-off between its precision and its computation complexity and this will be seen in chapter 6.

Consider the example from figure 5.2 where the block-tests are the same as those given in chapter 1. Suppose we have the following test resource compatibilities: t_1 is compatible with t_3 (R_4 = CBILBO, $R_2 \neq$ PRPG), t_2 is compatible with t_3 (R_3 = PRPG) and t_4 (R_5 = CBILBO), t_3 is compatible with t_1 and t_2, and t_4 is compatible with t_2. Figure 5.3 depicts the test compatibility graph built for the example described above. Using formula 5.3 the following compatibility probabilities are determined and explained in table 5.1.

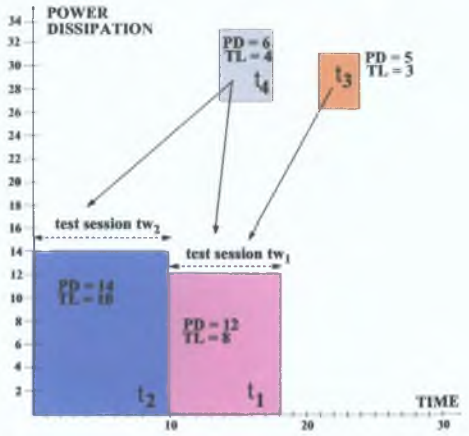


Figure 5.2: PTS Example I

Creation of distribution graphs

The next step in the FDS algorithm is to take the sum of the same type operations' scheduling probabilities for each c-step of the Control/Data-Flow Graph (CDFG). The resulting Distribution Graphs (DGs) indicate in FDS the concurrency of similar operations. For each Distribution Graphs (DG), the distribution in c-step i is given by: $DG(i) = \sum_{Opntype} Prob(Opn, i)$, where the sum

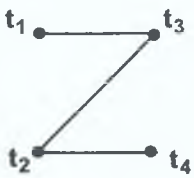


Figure 5.3: Compatibility Graph

$t_i \leftrightarrow ts_j$ *Probability of Compatibility*	$Comp_{Prob}(t_i, ts_j)$ *values*	*Reasons* ($N = 2$ *in all cases*)	
		$K, N_{incomp_{t_i}}$ *values*	*because*
$t_3 \leftrightarrow tw_1$	$Comp_{Prob}(t_3, tw_1) = 1$	$K = 1$	$CompList_{tw_1} = \{t_3\}$
		$N_{incomp_{t_3}} = 1$	$IncompList_{t_3} = \{t_4\}$
$t_3 \leftrightarrow tw_2$	$Comp_{Prob}(t_3, tw_2) =$ $= 1 - C_1^1/C_1^1 = 0$	$K = 2$	$CompList_{tw_2} = \{t_3, t_4\}$
		$N_{incomp_{t_4}} = 1$	$IncompList_{t_3} = \{t_4\}$
$t_4 \leftrightarrow tw_2$	$Comp_{Prob}(t_4, tw_2) =$ $= 1 - C_1^1/C_1^1 = 0$	$K = 2$	$CompList_{tw_2} = \{t_3, t_4\}$
		$N_{incomp_{t_4}} = 1$	$IncompList_{t_4} = \{t_3\}$

Table 5.1: Compatibility Probabilities for Figure 5.2 Example

is taken over all operations of a given type and $Prob(Opn, i)$ is the probability of an operation to be assigned in the i-th c-step.

On the other hand, the next step in PTS-FDS is to take the sum of the power-test probabilities for all feasible test/test subsession assignments and add them on top of the power dissipation accumulated already in the partial power-test chart. The resulting Power-Concurrency Distribution Graphs (PCDG) indicate the power dissipation expectations and, indirectly, the possible test concurrency distribution of the future test scheduling solution. PCDG's formula in each test subsession ts_j is:

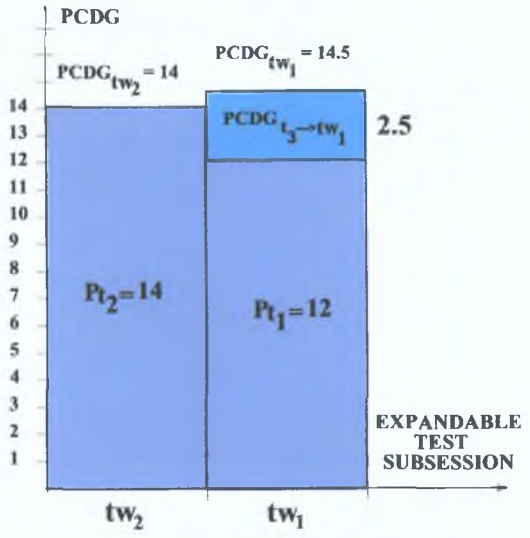


Figure 5.4: PCDG Example

$$PCDG(ts_j) = P_{ts_j} + \sum_{t_k} PCDG_{t_k}(ts_j), where \tag{5.4}$$

$$PCDG_{t_k}(ts_j) = P_{t_k} * Comp_{Prob}(t_k, ts_j) * Prob(t_k, ts_j), \tag{5.5}$$

with t_k being the set of tests assignable to ts_j.

For the above example, the PCDG values for tw_1 and tw_2 are calculated next based on formulas 5.4 and 5.5 and depicted in figure 5.4.

$$
\begin{aligned}
PCDG(tw_1) &= P_{tw_1} + PCDG_{t_3}(tw_1) \\
&= P_{tw_1} + P_{t_3} * Comp_{Prob}(t_3, tw_1) * Prob(t_3) \\
&= 12 + 5 * 1 * 0\,5 = 14\,5, \\
PCDG(tw_2) &= P_{tw_2} + PCDG_{t_3}(tw_2) + PCDG_{t_4}(tw_2) \\
&= P_{tw_2} + P_{t_3} * Comp_{Prob}(t_3, tw_2) * Prob(t_3) + P_{t_4} * Comp_{Prob}(t_4, tw_2) * Prob(t_4) \\
&= 14 + 5 * 0 * 0\,5 + 6 * 0 * 1 = 14
\end{aligned}
$$

An interesting aspect that has to be emphasized here is that $PCDG_{t_4}(tw_2) = P_{tw_2}$ That is, there are no other $PCDG_{t_3}(tw_2)$ and $PCDG_{t_4}(tw_2)$ values to be added to P_{tw_2} This is due to the fact $Comp_{Prob}(t_4, tw_2) = Comp_{Prob}(t_3, tw_2) = 0$, because t_3 and t_4 are the only tests in tw_2's compatibility list and they are incompatible with each other However, it can be seen in figure 5 2 that t_3 and t_4 could be executed sequentially in tw_2, even though they are not test resource compatible Probably a non-null compatibility probability would be more adequate For the above small example this test length compatibility between t_3 and t_4 is obvious and easy to check, but in general it would be harder to check for cases when ts_j's compatibility list is large However, instead of predicting this case by the PCDG formula, the algorithm checks this by temporarily assigning a test to a test subsession and recalculating all the probabilities

Another intuitive aspect of the PCDG formula given in 5 5 is that it should also include a time component The test time length component has not been employed in this algorithm for several reasons For example, if the $P_{t_k} * T_{t_k}$ component is used in formula 5 5 instead of the power P_{t_k} component, then PCDG turns into an energy distribution graph On the other hand, the PTS algorithms proposed in this thesis are meant to balance the power dissipation by reducing the power spikes Moreover, the main goal of the hierarchical approach proposed here is to optimize the power and test characteristics of the test hierarchy by parsing the modular test hierarchy in a bottom-up *fashion* (see subsection 3 3 2) Now an overall optimized power dissipation can be obtained throughout the system's testing mainly by balancing the power dissipation at each level of the system's modular test hierarchy Thus, at any level in the test hierarchy the power dissipation is considered balanced if the differences between the maximum, average and RMS power dissipation values (see subsection 3 3 1) are minimized Therefore, since the PCDG formula is a prediction component in the cost function (*Force*) in order to achieve minimized

power differences it should predict the power dissipation distribution rather than the energy consumption distribution On the other hand, speaking in terms of technology, power dissipation spikes could be sometimes neglected if they are isolated In this case the distribution graph has to look at the overall energy consumption rather than at the difference between maximum and average values of the power dissipation thorughout the test application time

The PTS approach proposed here does not yet tackle any technological peculiarity Such a technological case necessitates a more complex formulation to both predict the energy consumption and ignore the power dissipation spikes Moreover, a complex fine tuning would have to be carried out on the formulation dedicated to the aforementioned particular technological cases These approaches are not covered in this thesis and are left for future research

Calculation of Forces

The final step in FDS is to calculate the force associated with scheduling unscheduled operations to every possible c-steps The final step in PTS-FDS is to calculate the force associated with scheduling the first test from the ordered list to each compatible test subsession In PTS-FDS, for a given test t_i the force in test subsession ts_j is given by

$$Force_{t_i}(ts_j) = PCDG(ts_j) * x_{ts_j}(t_i), \tag{5 6}$$

where $PCDG(ts_j)$ is the predicted power dissipation value in ts_j and $x_{ts_j}(t_i)$ is the increase (or decrease) of t_i's probability in ts_j after assignment The assignment is done by temporarily reducing t_i's time frame (expandable test subsession set) to the test subsession selected for expansion

The *Self Force* in FDS is a quantity that reflects the effect of an operation-to-c-step assignment on the overall operation concurrency In PTS-FDS, the *Self Force* is a quantity that reflects the effect of a test-to-test subsession assignment on the overall power dissipation distribution and, indirectly, on the test concurrency distribution This *Self Force* is positive if the assignment causes an increase of power dissipation dissimilarity, and is negative for a decrease The *Self Force* associated with the assignment of test t_i to one of the test subsessions ts_j is

$$Self\ Force_{t_i \to ts_j} = \sum_{ts_j \in SETS_{t_i}} Force_{t_i}(ts_j), \tag{5 7}$$

where $ts_j \in SETS_{t_i}$ is the Set Of Expandable Test Subsessions ts_j with which t_i is compatible For the example given above the following forces are calculated

$$\begin{aligned}
Self\ Force_{t_3 \to tw_1} &= Force_{t_3 \to tw_1} + Force_{t_3 \nrightarrow tw_2} \\
&= PCDG(tw_1) * x_{tw_1}(t_3) + PCDG(tw_2) * x_{tw_2}(t_3) \\
&= 14\ 5 * (5 * 1 * 1 - 5 * 1 * 0\ 5) + 14 * (5 * 0 * 0 - 5 * 0 * 0\ 5) \\
&= 14\ 5 * 5 * 1 * 0\ 5 = 36\ 25, \\
Self\ Force_{t_3 \to tw_2} &= Force_{t_3 \to tw_2} + Force_{t_3 \nrightarrow tw_1} \\
&= PCDG(tw_2) * x_{tw_2}(t_3) + PCDG(tw_1) * x_{tw_1}(t_3) \\
&= 14 * (5 * 1 * 1 - 5 * 0 * 0\ 5) + 14\ 5 * (5 * 0 * 0 - 5 * 1 * 0\ 5) \\
&= 14 * 5 + 14\ 5 * (-2\ 5) = 70 - 36\ 25 = 33\ 75, \\
Self\ Force_{t_4 \to tw_2} &= Force_{t_4 \to tw_2} \\
&= PCDG(tw_2) * x_{tw_2}(t_4) \\
&= 14 * (6 * 1 * 1 - 6 * 0 * 1) = 14 * 6 = 84
\end{aligned}$$

where, for example, the term $PCDG(tw_1) * x_{tw_1}(t_3)$ is arrived at as below

$$\begin{aligned}
PCDG(tw_1) * x_{tw_1}(t_3) &= PCDG(tw_1) * (PCDG'_{t_3}(tw_1) - PCDG_{t_3}(tw_1)) \\
&= PCDG(tw_1) * (PCDG_{t_3 \to tw_1} - PCDG_{t_3}(tw_1)) \\
&= PCDG(tw_1) * (P_{t_3} * Comp'_{Prob}(t_3, tw_1) * Prob'(t_3) - \\
&\quad - P_{t_3} * Comp_{Prob}(t_3, tw_1) * Prob(t_3)) \\
&= 14\ 5 * (5 * 1 * 1 - 5 * 1 * 0\ 5) = 36\ 25,
\end{aligned}$$

where $PCDG'_{t_3}(tw_1) = PCDG_{t_3 \to tw_1}$ is the $PCDG_{t_3}(tw_1)$ value after t_3 has been assigned to tw_1 In this case both $Comp'_{Prob}(t_3, tw_1)$ and $Prob'(t_3)$ values become 1 ($Comp_{Prob}(t_3, tw_1)$ was 1 before the $t_3 \to tw_1$ assignment as well)

The *Self Force* was demonstrated in the FDS approach to be equal to the difference between the average distribution for the c-steps bounded by the new time frame, and the average for the c-steps of the initial one [PK89] By analogy, the

same formulation stands for PTS-FDS, as can be seen in the formula 5 8 below Using the FDS formulation, the force associated in PTS-FDS with the reduction of an initial time frame (bounded by the initial set of expandable test subsessions) to a new time frame (bounded by the final set of expandable test subsessions) is given by the following equation

$$SelfForce_{t_i \rightarrow ts_j} = \sum_{ts_j \in FS_{ts}} \frac{PCDG'(ts_j)}{NFS_{ts}} - \sum_{ts_j \in IS_{ts}} \frac{PCDG(ts_j)}{NIS_{ts}}, \qquad (5\ 8)$$

where FS_{ts} and IS_{ts} are, respectively, the final and the initial set of test subsessions expandable with t_i, having, respectively, NFS_{ts} and NIS_{ts} elements

Incompatibility Forces

In order to optimize the power dissipation (test concurrency) throughout the test application time, it is necessary to assign tests to test subsessions such that the power-dissipation/test-concurrency distribution is balanced However, assigning a test to a specific gap (expandable test subsession) often affects the time frames (i e , the set of test subsessions to where they can be assigned) of the other initially "ready" tests, which may become incompatible with the test subsessions (twin gap or shaded gap) newly generated after the assignment This can happen because scheduling a test, say t_i, is equivalent to reducing its test subsession set to one test subsession, say ts_j This modification could propagate to the test subsession set of the other tests (initially assignable to ts_j) which may become unassignable to subsession ts_j (actually the test subsessions generated inside ts_j after the insertion of t_i to it) hereafter

The above mentioned tests (initially assignable to ts_j) could become incompatible with the newly generated test subsessions from different points of view Firstly, the accumulated power dissipation has almost reached the limit and consequently no other tests can be scheduled in the new test subsession (twin gap) Secondly, the test length of the newly generated test subsessions (rest gap and twin gap) is smaller than the test length of the tests left to be scheduled Thirdly, they are simply incompatible with the test just assigned, from a test resource utilization point of view, and implicitly they are incompatible with the test subsession (twin gap) resulting from the assignment

Thus, a test-to-test subsession assignment usually creates additional forces that can reduce or even counter the globally intended improvement. Therefore it is important that they are accounted for. The force calculation must be performed for all tests which become incompatible. These forces are named *incompatibility forces* in this approach and they are calculated like the normal forces and added to the *Self Force*. After forces of all incompatible tests have been calculated and added to the *Self Force*, the test subsession exhibiting the lowest *Total Force* is selected for test assignment. It has been seen in section 4.2, when it came to talk about the *test mobility* function, that initially all tests are ordered by their mobility. Then, at each iteration the PTS-FDS algorithm schedules the first unscheduled test from the sorted list to the test subsession, giving the lowest *Total Force*.

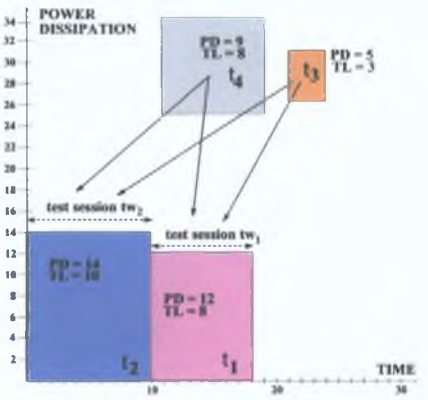


Figure 5.5: PTS Example II

The partial PTS schedule from figure 5.5 is taken as an example to show how *Total Forces* are calculated. The difference between the partial PTS schedule from figure 5.5 and the partial PTS schedule from figure 5.2 is test t_4. In figure 5.5, test t_4 has a lower mobility, that is $T_{t_4} = 8$, $P_{t_4} = 9$, and $CompList_{t_4} = \{t_1, t_2\}$. At this stage of the partial PTS schedule, test t_4 has to be scheduled. It can be scheduled to either tw_1 or tw_2. The unscheduled tests are t_4 and t_3, and their time frames are 2 for both, i.e. $\{tw_2, tw_1\}$. Their compatibility probabilities are given in table 5.2.

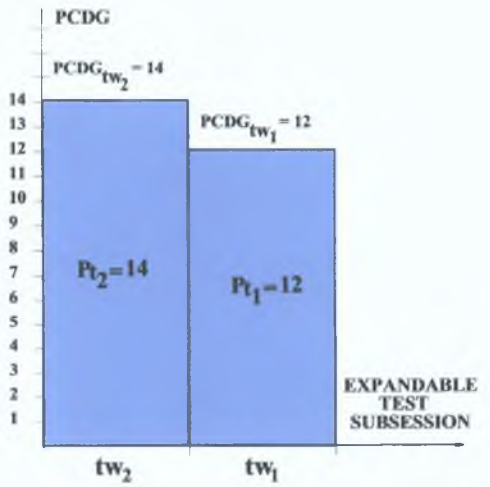
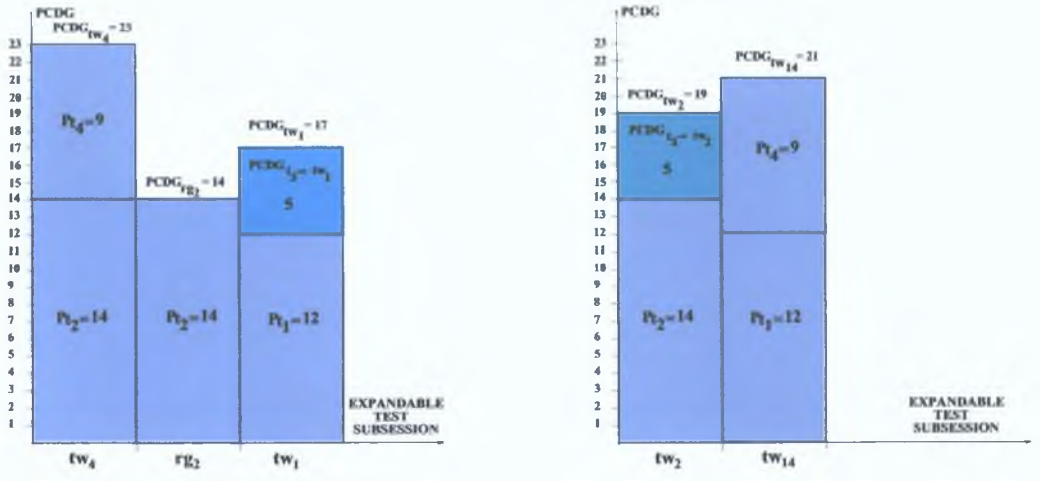


Figure 5.6: PCDG Before Scheduling

$t_i \leftrightarrow ts_j$ *Probability of Compatibility*	$Comp_{Prob}(t_i, ts_j)$ *values*	*Reasons* ($N = 2$ *in all cases*)	
		$K, N_{incomp_{t_i}}$ *values*	*because*
$t_3 \leftrightarrow tw_1$	$Comp_{Prob}(t_3, tw_1) =$	$K = 2$	$CompList_{tw_1} = \{t_3, t_4\}$
	$= 1 - C_1^1/C_1^1 = 0$	$N_{incomp_{t_3}} = 1$	$IncompList_{t_3} = \{t_4\}$
$t_4 \leftrightarrow tw_1$	$Comp_{Prob}(t_4, tw_1) =$	$K = 2$	$CompList_{tw_1} = \{t_3, t_4\}$
	$= 1 - C_1^1/C_1^1 = 0$	$N_{incomp_{t_4}} = 1$	$IncompList_{t_4} = \{t_3\}$
$t_3 \leftrightarrow tw_2$	$Comp_{Prob}(t_3, tw_2) =$	$K = 2$	$CompList_{tw_2} = \{t_3, t_4\}$
	$= 1 - C_1^1/C_1^1 = 0$	$N_{incomp_{t_3}} = 1$	$IncompList_{t_3} = \{t_4\}$
$t_4 \leftrightarrow tw_2$	$Comp_{Prob}(t_4, tw_2) =$	$K = 2$	$CompList_{tw_2} = \{t_3, t_4\}$
	$= 1 - C_1^1/C_1^1 = 0$	$N_{incomp_{t_4}} = 1$	$IncompList_{t_4} = \{t_3\}$

Table 5.2: Compatibility Probabilities for Figure 5.5 Example

The PCDG before scheduling t_4 is depicted in figure 5.6. If t_4 is scheduled in tw_2, then two test gaps are generated: tw_{24} ($T_{tw_{24}} = 8$, $P_{tw_{24}} = 14 + 9 = 23$, $CompList_{tw_{24}} = \{\varnothing\}$) and rg_2 ($T_{rg_2} = 10 - 8 = 2$, $P_{rg_2} = 14$, $CompList_{rg_2} = \{t_3\}$). t_3 can be scheduled to neither tw_{24} nor rg_2. If t_4 is scheduled then $Comp_{Prob}(t_3, tw_1)$ becomes 1 because $K = 1$ for tw_1's compatibility list and $N_{incomp_{t_3}}$ becomes 0. Then t_3 could be scheduled only to tw_1 and the new compatibility probability would be $Comp_{Prob}(t_3, tw_1) = 1$. The new PCDG is depicted in figure 5.7(a). Then the *Total Force* is given in the equation below by t_4's *Self Force* (first two components) and t_3's *Incompatibility Force* generated by the fact that t_3 can not be scheduled anymore in tw_2.



(a) If t_4 is Scheduled in tw_2

(b) If t_4 is Scheduled in tw_1

Figure 5.7: PCDGs After t_4's Scheduling

$$\begin{aligned} Total\ Force_{t_4 \to tw_2} &= Force_{t_4 \to tw_2} + Force_{t_4 \nrightarrow tw_1} + IncompF_{t_3 \nrightarrow tw_2} + IncompF_{t_3 \to tw_1} \\ &= 14 * (9 * 1 * 1 - 9 * 0 * 0\,5) + 12 * (9 * 0 * 0 - 9 * 0 * 0\,5) + \\ &\quad +14 * (5 * 0 * 0 - 5 * 0 * 0\,5) + 12 * (5 * 1 * 1 - 5 * 0 * 0\,5) \\ &= 126 + 0 + 0 + 60 = 186 \end{aligned}$$

where, for example, the term $Force_{t_4 \to tw_2}$ is calculated as presented next

$$\begin{aligned} Force_{t_4 \to tw_2} &= PCDG(tw_2) * (PCDG'_{t_4}(tw_2) - PCDG_{t_4}(tw_2)) \\ &= PCDG(tw_2) * (PCDG_{t_4 \to tw_2} - PCDG_{t_4}(tw_2)) \\ &= PCDG(tw_2) * (P_{t_4} * Comp'_{Prob}(t_4, tw_2) * Prob'(t_4) - \\ &\quad -P_{t_4} * Comp_{Prob}(t_4, tw_2) * Prob(t_4)) \\ &= 14 * (9 * 1 * 1 - 9 * 0 * 0\,5) = 126, \end{aligned}$$

If t_4 is scheduled in tw_1, then the test gap is tw_{14} ($T_{tw_{14}} = 8$, $P_{tw_{14}} = 12+9 = 21$, $CompList_{tw_{14}} = \{\varnothing\}$) Rest gap rg_1 is not generated anymore because ($T_{rg_1} = 8-8 = 0$) t_3 can not be scheduled to tw_{14} If t_4 is scheduled then $Comp_{Prob}(t_3, tw_2)$ becomes 1 because $K = 1$ for tw_2's compatibility list and $N_{incompt_3}$ becomes 0 Then t_3 could be scheduled only to tw_2 and the new compatibility probability would be $Comp_{Prob}(t_3, tw_2) = 1$ The new PCDG is depicted in figure 5 7(b) In this case the *Total Force* is given below by t_4's *Self Force* and t_3's *Incompatibility Force* generated by the fact that t_3 can not be scheduled anymore in tw_1

$$\begin{aligned} Total\ Force_{t_4 \to tw_1} &= Force_{t_4 \to tw_1} + Force_{t_4 \nrightarrow tw_2} + IncompF_{t_3 \nrightarrow tw_1} + IncompF_{t_3 \to tw_2} \\ &= 12 * (9 * 1 * 1 - 9 * 0 * 0\,5) + 14 * (9 * 0 * 0 - 9 * 0 * 0\,5) + \\ &\quad +12 * (5 * 0 * 0 - 5 * 0 * 0\,5) + 14 * (5 * 1 * 1 - 5 * 0 * 0\,5) \\ &= 108 + 0 + 0 + 70 = 178 \end{aligned}$$

$Total\ Force_{t_4 \to tw_1}$ is smaller than $Total\ Force_{t_4 \to tw_2}$, thus t_4 will be scheduled in tw_1

5.1 1 Algorithm Pseudocode

To make it clear it is emphasized here that there is an important conceptual difference between the original FDS algorithms and the PTS-FDS algorithm This is

considered to be the third difference between the FDS and PTS-FDS algorithms Basically, this difference is given by the fact that in the former algorithm the *Total Forces* for each feasible test-to-test subsession assignment are calculated and the one with the lowest force is taken for scheduling On the other hand, in PTS-FDS, only the first test in the ordered test mobility list is considered for scheduling at each iteration The *Total Forces* of all its feasible test subsession assignments are calculated and the lowest force dictates the choice of the test subsession to be expanded next

PSEUDOCODE of PTS-FDS ALGORITHM

◇sort all the tests by their mobility in two steps (test length, power dissipation),
◇initialize the *GrowingTree*, **the** *BlockTestList* **and the** *GapsList*,
◇while (there are unscheduled tests) do { /* *BlockTestList* **is not empty*/**

- **evaluate time frames for all tests,**
- **while (there are tests having null time frames) do{**
  - *CurTest* **= the first out of** *BlockTestList* **having null time frame**
  - **insert** *CurTest* **as the tail of** *GrowingTree* **roots /*new test session*/ and make** *CurTest* **"used",**
  - **remove** *CurTest* **from** *BlockTestList*,
  - **generate a** *TwinGap* **gap as the twin of** *CurTest*,
  - **insert** *TwinGap* **into** *GapsList*,
  - **evaluate time frames for all tests, } /*while*/**
- *CurTest* **= the head of** *BlockTestList*,
- **update power/concurrency distribution graphs,**
- **if (there are more than one test subsessions in** *CurTest's* **time frame) do{**
  - **calculate** *CurTest's* **Self Forces for every feasible test subsession assignment,**
  - **add incompatibility forces to Self Forces,**
  - **SCHEDULE** *CurTest* **to the test subsession exhibiting the lowest Total Force at assignment, } /*if*/**
- **else SCHEDULE** *CurTest* **to the test subsession,**

◇} /*while*/

The PTS-FDS algorithm is iterative, with one test scheduled in each iteration The data structures used in it are the same as in the LS-based approaches (see subsection 4 1 1) Its pseudocode is given above Tests are initially ordered before being scheduled The sorted tests are then iteratively scheduled into the available test (sub)sessions (ETPs) The first stage of the algorithm detects the test having null time frames These tests will be considered as test sessions and inserted as roots into the *Growing Tree* In the second stage, *CurTest* is assigned each time with the first test from the *BlockTestList*, which is the one with the longest test length (and then highest power dissipation) Then, the time frames of all the tests are updated and they are further used to update the *PCDG* The latter is then used to calculate the *Forces* for every feasible *CurTest-CurGap* assignment, where *CurGap* is iteratively assigned with all the gaps from *GapsList* The *CurGap*, that exhibits the lowest *Total Force* for the assignment with *CurTest* test, is finally expanded with *CurTest*, which is then removed from *BlockTestList*

The *SCHEDULE* procedure is the same as the one implemented in the LS-based approaches in the previous chapter It carries out the scheduling of a test into the *GrowingTree* and its subsequent removal from the *BlockTestList* As it can be seen in figure 3 5, the merging step implies the generation of shaded and hatched gaps The hatched gap in the *SCHEDULE* procedure is called *RestGap* It represents the space (gap) left behind each merging step Therefore it inherits the accumulated power dissipation and test resource compatibility list data from the former test subsession, but, on the other hand, the test length is the test time left after the merging step Obviously, if the test time left is null, there is no *RestGap* generated *TwinGap* represents the twin of the just merged test and, therefore, its test length is the same as the newly scheduled test Its power dissipation is the sum of the test subsession's power dissipation before assignment and the power dissipation of the test just scheduled Its compatibility list is the intersection of the previous ETP's compatibility list with the compatibility list of the just merged test If the resulting compatibility list is null it means that the newly generated *TwinGap* is born "saturated", and there is no point in taking it into account for further expansion Thus, it is inserted into the *GrowingTree*, but not into the *GapsList*

5 1 2 Algorithm Complexity

The complexity of the PTS-FDS algorithm can be derived in the same way as the complexity of FDS Firstly, each iteration of the algorithm schedules at least one test This implies there can be at most N iterations, where N is the initial number of tests Secondly, within each iteration, for a test to be scheduled, there are at most N test subsession (gaps) for which forces must be calculated This assumption is a very conservative upper bound because it is only in the worst case, where all tests are totally incompatible, that the maximum possible number of test subsessions making up the test's time frame is equal to N (total number of tests) Finally, for each tentative test-to-test subsession assignment, there may be at most *N-1* tests incompatible with the current one to be affected, and their incompatibility force must also be calculated The combined effect of the above three considerations yields the combined $O(N^3)$ complexity For the PTS-FDS, one method can be applied in order to reduce substantially the complexity and is similar to the one presented in [PK89] In [PK89] the complexity was reduced by performing a preliminary reduction of all time frames which exceed a constant maximum allowable height H Forces are then calculated in the usual fashion, and all long time frames are reduced simply by removing from the time frame the c-steps exhibiting the highest forces Similarly in the PTS, approach the test subsessions sets can be limited to a certain number H and the scheduling complexity would then be reduced to $O(HN^2)$, where H is a predefined constant Forces are then calculated in the usual fashion as well That is, all big test subsession sets are reduced by removing from the test subsession set, the test subsessions exhibiting the highest forces

5.1 3 Test Scheduling Example

The same two BTS examples given in subsection 4 1 3 are used here to provide a deeper insight into the workings and the results of this algorithm The 10 BTS is repeated below For this example, the initial values for the data structures are $GT = 0, GL = 0, BTL = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ The algorithm is repeated until all the tests are scheduled Each iteration starts with the scheduling of the unassigned tests that have null time frames Thus, the lowest mobility tests with null time frames are made roots of the GT and the time frames are then updated This

repeats until every unused test from BTL has a non-zero time frame Then, tests are scheduled according to their mobility Since the number of tests to be scheduled is ten, there are ten main steps all together, which are depicted in figure 5 8 Following this the scheduling steps from figure 5 8 executed by the first algorithm of the PTS-LEA approach are detailed

$test_i(power\ dissipation, test\ length, \{compatibility\ list\})$
$t_1(9, 9, \{t_2, t_3, t_5, t_6, t_8, t_9\})$
$t_2(4, 8, \{t_1, t_3, t_7, t_8\})$
$t_3(1, 8, \{t_1, t_2, t_4, t_7, t_9, t_{10}\})$
$t_4(6, 6, \{t_3, t_5, t_7, t_8\})$
$t_5(5, 5, \{t_1, t_4, t_9, t_{10}\})$
$t_6(2, 4, \{t_1, t_7, t_8, t_9\})$
$t_7(1, 3, \{t_2, t_3, t_4, t_6, t_8, t_9\})$
$t_8(4, 2, \{t_1, t_2, t_4, t_6, t_7, t_9, t_{10}\})$
$t_9(12, 1, \{t_1, t_3, t_5, t_6, t_7, t_8, t_{10}\})$
$t_{10}(7, 1, \{t_3, t_5, t_8, t_9\})$

BTL	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
TF	0	0	0	0	0	0	0	0	0	0
GL	$\{\emptyset\}$									
GT	$\{\emptyset\}$									

Table 5 3 Data Structures (Step 0)

Step 1 Time frames of all tests in BTL are null because there are no test (sub)sessions in GL Therefore, the first test (t_1) is selected from BTL for scheduling It is merged to GT as root A twin gap tw_1 is generated and inserted in GL so that $GL = \{tw_1\}$, while node t_1, inserted into GT, is shaded Then t_1 is removed from BTL

BTL	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
TF	1	1	0	1	1	0	1	1	0
GL	$\{tw_1\}$								
GT	t_1 is merged as root								

Table 5 4 Data Structures (After Step 1)

Step 2 The time frames of the unused tests from BTL are updated Their new time frames are $TF_{t_2} = 1, TF_{t_3} = 1, TF_{t_4} = 0, TF_{t_5} = 1, TF_{t_6} = 1, TF_{t_7} = 0, TF_{t_8} = 1, TF_{t_9} = 1, TF_{t_{10}} = 0$ t_4 is the next test in the sorted BTL having null time frame Consequently, it is selected for schedule during this step The structures become $GL = \{tw_1, tw_4\}$, while BTL consists of the tests left with the following updated time frames $TF_{t_2} = 1, TF_{t_3} = 1, TF_{t_5} = 2, TF_{t_6} =$

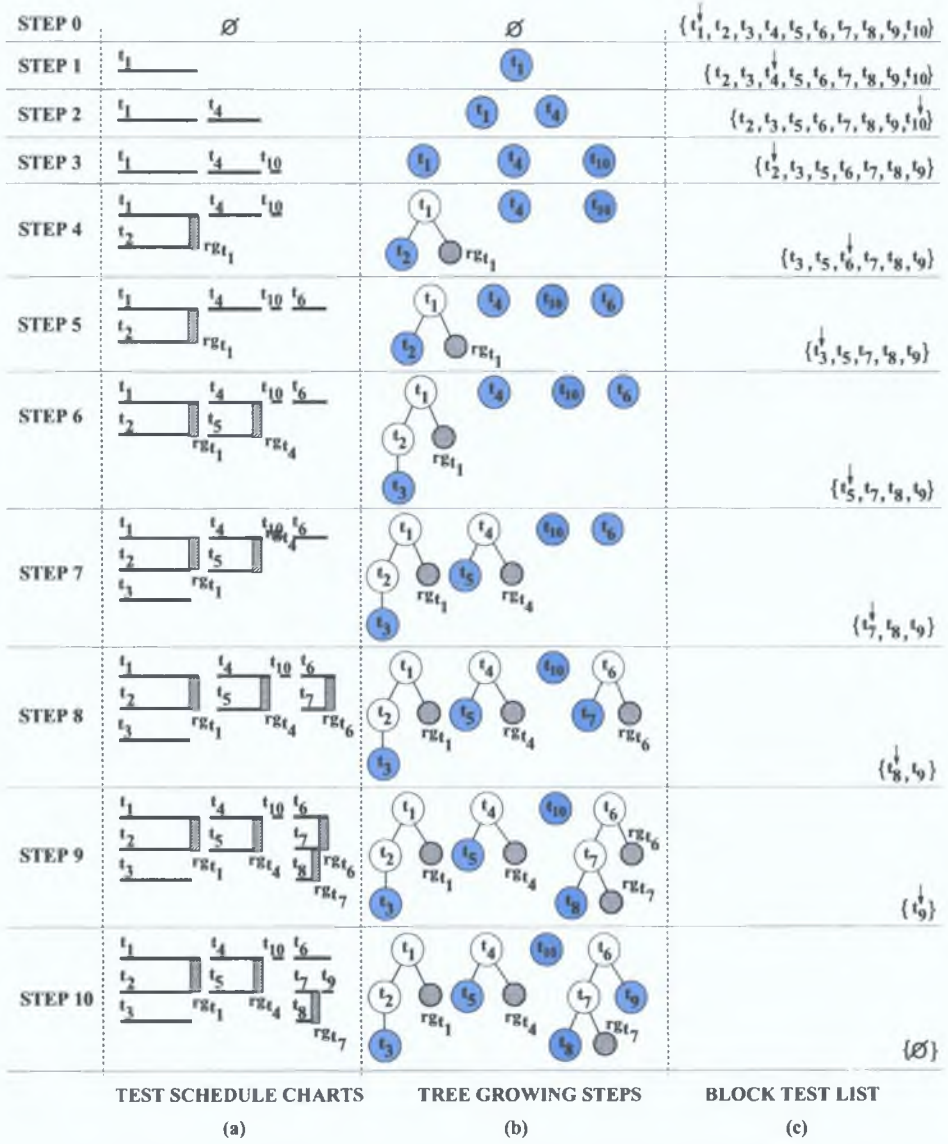


Figure 5.8: Tree Growing Steps Example (PTS-FDS Approach)

$1, TF_{t_7} = 1, TF_{t_8} = 2, TF_{t_9} = 1, TF_{t_{10}} = 0.$

BTL	t_2	t_3	t_5	t_6	t_7	t_8	t_9	t_{10}
TF	1	1	2	1	1	2	1	0
GL	$\{tw_1, tw_4\}$							
GT	t_4 is merged as root							

Table 5.5: Data Structures (After Step 2)

Step 3. t_{10} is scheduled next because it is the only test left having null time frame. The structures are now: $GL = \{tw_1, tw_4, tw_{10}\}$, while BTL consists of the tests left, with the following updated time frames: $TF_{t_2} = 1, TF_{t_3} = 1, TF_{t_5} = 2, TF_{t_6} = 1, TF_{t_7} = 1, TF_{t_8} = 2, TF_{t_9} = 2.$

BTL	t_2	t_3	t_5	t_6	t_7	t_8	t_9
TF	1	1	2	1	1	2	2
GL	$\{tw_1\ tw_4\ tw_{10}\}$						
GT	t_{10} is merged as root						

Table 5 6 Data Structures (After Step 3)

Step 4 As can be seen above there are no tests left with null time frames Thus t_2 is to be scheduled next because it is the next in the sorted *BTL*

Force values are to be calculated for its assignments to each test subsession (gap) from its time frame, and the one with the lowest *Total Force* value gives the test subsession where t_2 will be scheduled The *PCDG* is updated during each step using formula 5 4 Because its time frame is equal to one test subsession (tw_1), there is no need to update the *PCDG* and subsequently, there is no need to calculate the *Total Force* of assigning t_2 to tw_1 The structures become $GL = \{rg_1, tw_2, tw_4, tw_{10}\}$, while tests left in *BTL* have the following updated time frames $TF_{t_3} = 1, TF_{t_5} = 1, TF_{t_6} = 0, TF_{t_7} = 1, TF_{t_8} = 2, TF_{t_9} = 2$

BTL	t_3	t_5	t_6	t_7	t_8	t_9
TF	1	1	0	1	2	2
GL	$\{rg_1\ tw_2\ tw_4\ tw_{10}\}$					
GT	t_2 is scheduled in tw_1					

Table 5 7 Data Structures (After Step 4)

Step 5 Test t_6 has now a null time frame Therefore, it will be scheduled as root in *GT* and removed from *BTL* *GL* becomes $GL = \{tw_6, rg_1, tw_2, tw_4, tw_{10}\}$, while *BTL*'s tests have the following time frames $TF_{t_3} = 1, TF_{t_5} = 1, TF_{t_7} = 2, TF_{t_8} = 3, TF_{t_9} = 3$

BTL	t_3	t_5	t_7	t_8	t_9
TF	1	1	2	3	3
GL	$\{rg_1\ tw_2\ tw_4\ tw_{10}\ tw_6\}$				
GT	t_6 is merged as root				

Table 5 8 Data Structures (After Step 5)

Step 6 Test t_3 is to be scheduled next Its time frame is one (tw_2) After its scheduling to tw_2, *GL* becomes $\{tw_6, rg_1, tw_4, tw_{10}\}$ tw_3 is not generated because its resulting compatibility list is empty Then, rg_2 is not generated either because its test length is zero, t_2 and t_3 having the same test length Now, *BTL*'s tests have the following time frames $TF_{t_5} = 1, TF_{t_7} = 2, TF_{t_8} = 2, TF_{t_9} = 3$ As can be seen, there is no need to calculate the *Total Force* in this step either However, it is going to be calculated here in order to give an example of how the *Total Force* and its components (Self and Incompatibility Forces) are generated

In order to calculate any force, the PCDG has to be calculated first For the PCDG, the compatibility probability is needed The calculations are made according to formulas 5 3, 5 4, 5 6, and 5 7 and are given below The first set of calculations are made on the $Comp_{Prob}(t_i, ts_j)$ values The first set represents

the $Comp_{Prob}(t_i, ts_j)$ values before the scheduling of t_3 in tw_2, while the second set of $Comp_{Prob}(t_i, ts_j)$s are calculated after the scheduling of t_3 in tw_2 The $Comp_{Prob}(t_i, ts_j)$s before scheduling t_3 to tw_2 are calculated as in table 5 9

$t_i \leftrightarrow ts_j$ *Prob of Comp*	$Comp_{Prob}(t_i, ts_j)$ *values*	*Reasons* ($N = 5$ *in all cases*)	
		$K, N_{incomp_{t_i}}$ *values*	*because*
$t_5 \leftrightarrow tw_4$	$Comp_{Prob}(t_5, tw_4) =$ $= 1 - C_3^2/C_4^2 = 0\,5$	$K = 3$	$CompList_{tw_4} = \{t_5, t_7, t_8\}$
		$N_{incomp_{t_5}} = 3$	$IncompList_{t_5} = \{t_3, t_7, t_8\}$
$t_7 \leftrightarrow tw_4$	$Comp_{Prob}(t_7, tw_4) = 1$	$K = 3$	$CompList_{tw_4} = \{t_5, t_7, t_8\}$
		$N_{incomp_{t_7}} = 1$	$IncompList_{t_7} = \{t_5\}$
$t_8 \leftrightarrow tw_4$	$Comp_{Prob}(t_8, tw_4) =$ $= 1 - C_2^2/C_4^2 = 0\,83$	$K = 3$	$CompList_{tw_4} = \{t_5, t_7, t_8\}$
		$N_{incomp_{t_8}} = 2$	$IncompList_{t_8} = \{t_3, t_5\}$
$t_9 \leftrightarrow tw_{10}$	$Comp_{Prob}(t_9, tw_{10}) = 1$	$K = 1$	$CompList_{tw_{10}} = \{t_9\}$
		$N_{incomp_{t_9}} = 0$	$IncompList_{t_9} = \{\emptyset\}$
$t_3 \leftrightarrow tw_2$	$Comp_{Prob}(t_3, tw_2) =$ $= 1 - C_2^1/C_4^1 = 0\,5$	$K = 2$	$CompList_{tw_2} = \{t_3, t_8\}$
		$N_{incomp_{t_3}} = 2$	$IncompList_{t_3} = \{t_5, t_8\}$
$t_8 \leftrightarrow tw_2$	$Comp_{Prob}(t_8, tw_2) =$ $= 1 - C_2^1/C_4^1 = 0\,5$	$K = 2$	$CompList_{tw_2} = \{t_3, t_8\}$
		$N_{incomp_{t_8}} = 2$	$IncompList_{t_8} = \{t_3, t_5\}$
$t_9 \leftrightarrow rg_1$	$Comp_{Prob}(t_9, rg_1) = 1$	$K = 1$	$CompList_{rg_1} = \{t_9\}$
		$N_{incomp_{t_9}} = 0$	$IncompList_{t_9} = \{\emptyset\}$
$t_7 \leftrightarrow tw_6$	$Comp_{Prob}(t_7, tw_6) = 1$	$K = 3$	$CompList_{tw_6} = \{t_7, t_8, t_9\}$
		$N_{incomp_{t_7}} = 1$	$IncompList_{t_7} = \{t_5\}$
$t_8 \leftrightarrow tw_6$	$Comp_{Prob}(t_8, tw_6) =$ $= 1 - C_2^2/C_4^2 = 0\,83$	$K = 3$	$CompList_{tw_6} = \{t_7, t_8, t_9\}$
		$N_{incomp_{t_8}} = 2$	$IncompList_{t_8} = \{t_3, t_5\}$
$t_9 \leftrightarrow tw_6$	$Comp_{Prob}(t_9, tw_6) = 1$	$K = 3$	$CompList_{tw_6} = \{t_7, t_8, t_9\}$
		$N_{incomp_{t_9}} = 0$	$IncompList_{t_9} = \{\emptyset\}$

Table 5 9 Compatibility Probabilities Before Scheduling t_3 to tw_2

The below figure gives by means of two tables an overview of the change of $Comp_{Prob}(t_i, ts_j)$ values before and after scheduling t_3 to tw_2 After scheduling t_3 to tw_2, the $Comp_{Prob}(t_i, ts_j)$s are calculated as in table 5 10

$Comp_{Prob}$ Tests	Test Subsessions				
	tw_4	tw_{10}	tw_2	rg_1	tw_6
t_3			0 5		
t_5	0 5				
t_7	1				1
t_8	0 83		0 5		1
t_9		1		1	1

(a) $Comp_{Prob}$ Before t_3's Scheduling

$Comp_{Prob}$ Tests	Test Subsessions					
	tw_4	tw_{10}	tw_3	rg_2	rg_1	tw_6
t_5	0 67					
t_7	1					1
t_8	1					1
t_9		1			1	1

(b) $Comp_{Prob}$ After t_3's Scheduling

Figure 5 9 $Comp_{Prob}$ Values in Step 6

In figure 5 10 the PCDG values before the scheduling of test t_3 into test subsession tw_2 are depicted Figure 5 11 depicts the PCDG values after the scheduling of test t_3 into test subsession tw_2 rg_2 is not generated because its test length is null ($T_{rg_2} = T_{tw_2} - T_{t_3} = 8 - 8 = 0$)

$t_i \leftrightarrow ts_j$ Prob. of Comp.	$Comp_{Prob}(t_i, ts_j)$ values	Reasons ($N = 4$ in all cases)	
		$K, N_{incompt_i}$ values	because
$t_8 \leftrightarrow tw_3$	$Comp_{Prob}(t_8, tw_3) = 0$	$K = 0$	$CompList_{twin} = \{\oslash\}$
		$N_{incompt_8} = 1$	$IncompList_{t_8} = \{t_5\}$
$t_8 \leftrightarrow rg_2$	$Comp_{Prob}(t_8, rg_2) = 0$	$T_{rest} = 0$	
$t_5 \leftrightarrow tw_4$	$Comp_{Prob}(t_5, tw_4) =$ $= 1 - C_2^2/C_3^2 = 0.67$	$K = 3$	$CompList_{tw_4} = \{t_5, t_7, t_8\}$
		$N_{incompt_5} = 2$	$IncompList_{t_5} = \{t_7, t_8\}$
$t_7 \leftrightarrow tw_4$	$Comp_{Prob}(t_7, tw_4) = 1$	$K = 3$	$CompList_{tw_4} = \{t_5, t_7, t_8\}$
		$N_{incompt_7} = 1$	$IncompList_{t_7} = \{t_5\}$
$t_8 \leftrightarrow tw_4$	$Comp_{Prob}(t_8, tw_4) = 1$	$K = 3$	$CompList_{tw_4} = \{t_5, t_7, t_8\}$
		$N_{incompt_8} = 1$	$IncompList_{t_8} = \{t_5\}$
$t_9 \leftrightarrow tw_{10}$	$Comp_{Prob}(t_9, tw_{10}) = 1$	$K = 1$	$CompList_{tw_{10}} = \{t_9\}$
		$N_{incompt_9} = 0$	$IncompList_{t_9} = \{\oslash\}$
$t_9 \leftrightarrow rg_1$	$Comp_{Prob}(t_9, rg_1) = 1$	$K = 1$	$CompList_{rg_1} = \{t_9\}$
		$N_{incompt_9} = 0$	$IncompList_{t_9} = \{\oslash\}$
$t_7 \leftrightarrow tw_6$	$Comp_{Prob}(t_7, tw_6) = 1$	$K = 3$	$CompList_{tw_6} = \{t_7, t_8, t_9\}$
		$N_{incompt_7} = 1$	$IncompList_{t_7} = \{t_5\}$
$t_8 \leftrightarrow tw_6$	$Comp_{Prob}(t_8, tw_6) = 1$	$K = 3$	$CompList_{tw_6} = \{t_7, t_8, t_9\}$
		$N_{incompt_8} = 1$	$IncompList_{t_8} = \{t_5\}$
$t_9 \leftrightarrow tw_6$	$Comp_{Prob}(t_9, tw_6) = 1$	$K = 3$	$CompList_{tw_6} = \{t_7, t_8, t_9\}$
		$N_{incompt_9} = 0$	$IncompList_{t_9} = \{\oslash\}$

Table 5.10: Compatibility Probabilities After Scheduling t_3 to tw_2

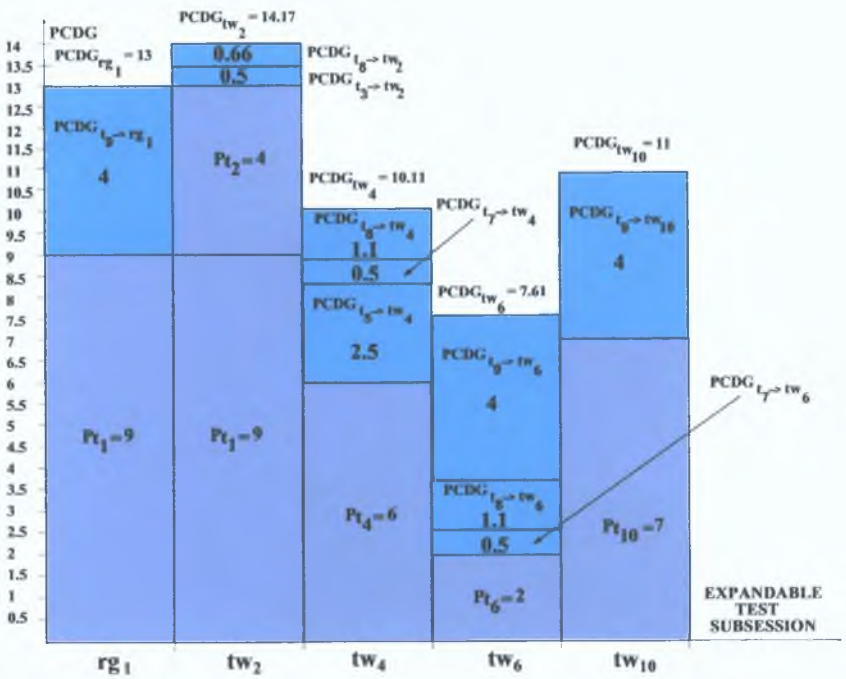


Figure 5.10: PCDG Before t_3's Scheduling - Step 6

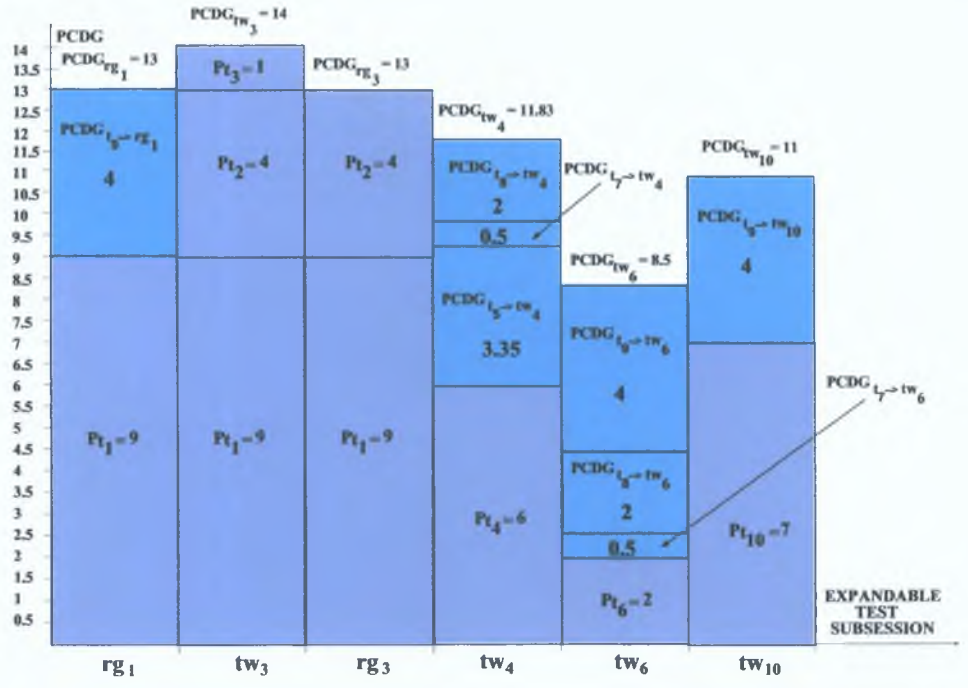


Figure 5.11: PCDG After t_3's Scheduling - Step 6

Then the *Total Force* of assigning t_3 to tw_2 is given below. It is the sum of t_3's Self Force of assigning it to the test subsession tw_2 and the Incompatibility Forces. One Incompatibility Force is given by the fact that scheduling test t_3 to test subsession tw_2, means that test t_8 can not be scheduled in this test subsession anymore. That is due to the fact that the new twin gap tw_3 is not compatible with t_8, because t_3 is not compatible with t_8, and the rest gap rg_{23} has a null test length ($T_{t_2} = T_{t_3}$). Another Incompatibility Force is given by the possible assignment of test t_8 to the other two test subsession (tw_4 and tw_6).

$$
\begin{aligned}
TF_{t_3 \to tw_2} &= F_{t_3 \to tw_2} + IncompF_{t_8 \to tw_4} + IncompF_{t_8 \nrightarrow tw_2} + IncompF_{t_8 \to tw_6} \\
&= PCDG(tw_2) * x_{t_3 \to tw_2} + PCDG(tw_4) * x_{t_8 \to tw_4} + PCDG(tw_2) * x_{t_8 \nrightarrow tw_2} + \\
&= +PCDG(t_8) * x_{t_8 \to tw_6} \\
&= 14.17 * (1 * 1 * 1 - 1 * 0.5 * 1) + 10.11 * (4 * 1 * 0.5 - 4 * 0.8(3) * 0.33) + \\
&\quad +14.17 * (4 * 0 * 0.5 - 4 * 0.5 * 0.33) + 7.61 * (4 * 1 * 0.5 - 4 * 0.83 * 0.33) \\
&= 7.08 + 8.99 - 9.4 + 6.77 = 13.39.
\end{aligned}
$$

The value of $TF_{t_3} \to tw_2$ is positive, that is t_3's assignment to tw_2 is not an attractive choice. However, this is the only choice for test t_3 to be scheduled in the actual test schedule without increasing the total test application time. This is the

PTS-FDS algorithm's peculiarity that tests are scheduled by their order in the test mobility list

BTL	t_5	t_7	t_8	t_9
TF	1	2	2	3
GL	$\{rg_1\ tw_4\ tw_{10}\ tw_6\}$			
GT	t_3 is merged in tw_2			

Table 5 11 Data Structures (After Step 6)

Step 7 During this step, test t_5 is scheduled to the only test subsession (tw_4) of its time frame After its scheduling to tw_4, GL becomes $\{rg_1, tw_{10}, tw_6\}$ tw_5 is not generated because its resulting compatibility list is empty rg_4 is not generated either because its test length $TL_{rg_4} = 1$ could accommodate only test t_9, but the latter is not test resource compatible to rg_4 BTL's tests have the following time frames $TF_{t_7} = 1, TF_{t_8} = 1, TF_{t_9} = 3$

BTL	t_7	t_8	t_9
TF	1	1	3
GL	$\{rg_1\ tw_{10}\ tw_6\}$		
GT	t_5 is merged in tw_4		

Table 5 12 Data Structures (After Step 7)

Step 8 After this step, test t_7 is also scheduled to the only test subsession (tw_6) of its time frame GL becomes $GL = \{rg_1, rg_6, tw_7, tw_{10}\}$ BTL's tests have the following time frames $TF_{t_8} = 1, TF_{t_9} = 4$

BTL	t_8	t_9
TF	1	4
GL	$\{rg_1\ rg_6\ tw_7\ tw_{10}\}$	
GT	t_7 is merged in tw_6	

Table 5 13 Data Structures (After Step 8)

Step 9 Now, test t_8 is being scheduled to the only test subsession (tw_7) of its time frame GL becomes $\{rg_1, rg_6, rg_7, tw_8, tw_{10}\}$ The only test left has the following time frame $TF_{t_9} = 5$

BTL	t_9
TF	5
GL	$\{rg_1\ rg_6\ rg_7\ tw_8\ tw_{10}\}$
GT	t_8 is merged in tw_7

Table 5 14 Data Structures (After Step 9)

Step 10 In the last step, test t_9 could be scheduled to one of $GL's$ test subsessions ($GL = \{rg_1, rg_6, rg_7, tw_8, tw_{10}\}$) Thus, the first step is to update the $PCDG$ Using formula 5 3 the following values have been calculated

$Comp_{Prob}(t_9, tw_{10}) = 1\ 00$, while $Prob(t_9, tw_{10}) = 0\ 2$,
$Comp_{Prob}(t_9, rg_1) = 1\ 00$, while $Prob(t_9, rg_1) = 0\ 2$,
$Comp_{Prob}(t_9, rg_6) = 1\ 00$, while $Prob(t_9, rg_6) = 0\ 2$,
$Comp_{Prob}(t_9, rg_7) = 1\ 00$, while $Prob(t_9, rg_7) = 0\ 2$,
$Comp_{Prob}(t_9, tw_8) = 1\ 00$, while $Prob(t_9, tw_8) = 0\ 2$

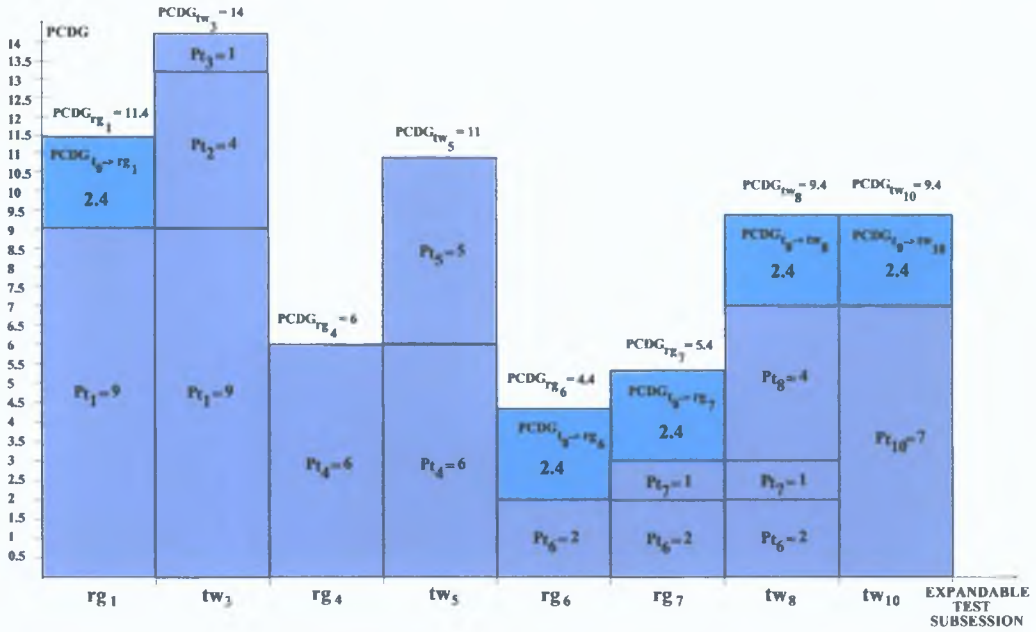
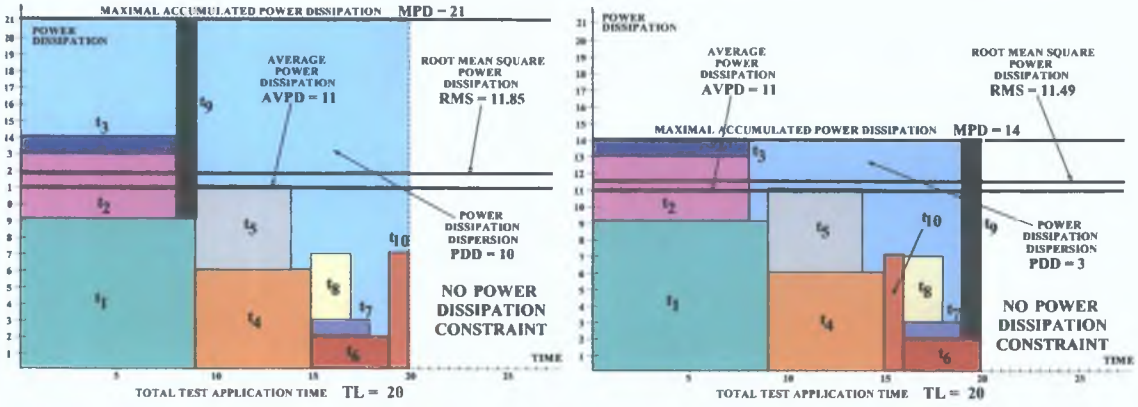


Figure 5.12: PCDG - Step 10 (PTS-FDS Approach)

$$
\begin{aligned}
SF_{t_9 \to tw_{10}} &= F_{t_9 \to tw_{10}} + F_{t_9 \nrightarrow rg_1} + F_{t_9 \nrightarrow rg_6} + F_{t_9 \nrightarrow rg_7} + F_{t_9 \nrightarrow tw_8} \\
&= 9.4(12 * 1.00 * 1.00 - 12 * 1.00 * 0.2) + 11.4(0 - 12 * 1.00 * 0.2) + \\
&\quad +4.4(0 - 12 * 1.00 * 0.2) + 5.4(0 - 12 * 1.00 * 0.2) + 9.4(0 - 12 * 1.00 * 0.2) \\
&= 90.24 - 27.36 - 10.56 - 12.96 - 22.56 = 16.80; \\
SF_{t_9 \to rg_1} &= 40.80; \\
SF_{t_9 \to rg_6} &= -43.20; \\
SF_{t_9 \to rg_7} &= -22.56; \\
SF_{t_9 \to tw_8} &= 16.80.
\end{aligned}
$$

Using formula 5.4, $PCDG$ is generated for test t_9 and depicted in figure 5.12. Because t_9 is the last test-block to be scheduled, its compatibility probabilities are equal (1.00) for all five gaps. The same happens with the uniform probability of assigning t_9 to the five gaps, which is 0.2. Thus, the probable distribution calculated with the sum from formula 5.4 is $12 * 1.00 * 0.2 = 2.4$. It can be seen that on top of the already accumulated power dissipation of the gaps, where t_9 can be assigned, the probable distribution, which is 2.4 for all gaps, is added. Intuitively, in order to balance the power dissipation over all the gaps, by assigning t_9, the assignment should be done to rg_6. This is demonstrated by calculating above with formula 5.7 the *Self Forces* (SF) associated with the assignment of t_9 to each of the five gaps.

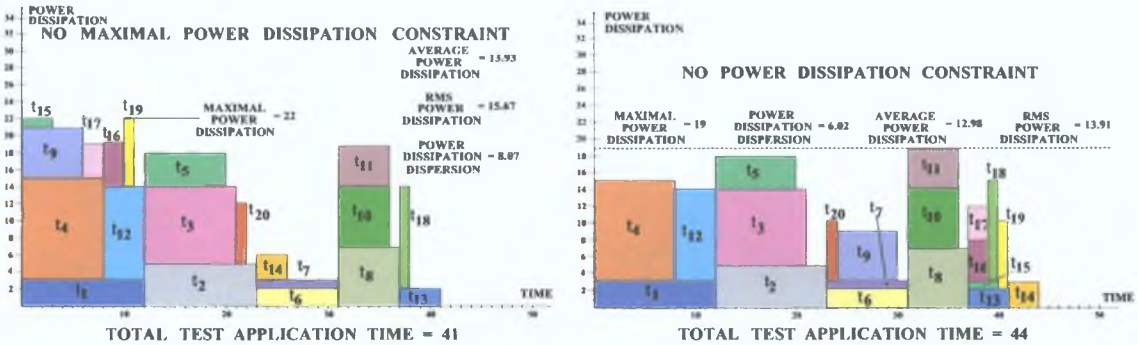


(a) PTS-LEA (MRU) Solution (b) PTS-FDS Solution

Figure 5.13: PTS Charts Without Power Constraints - 10 BTS

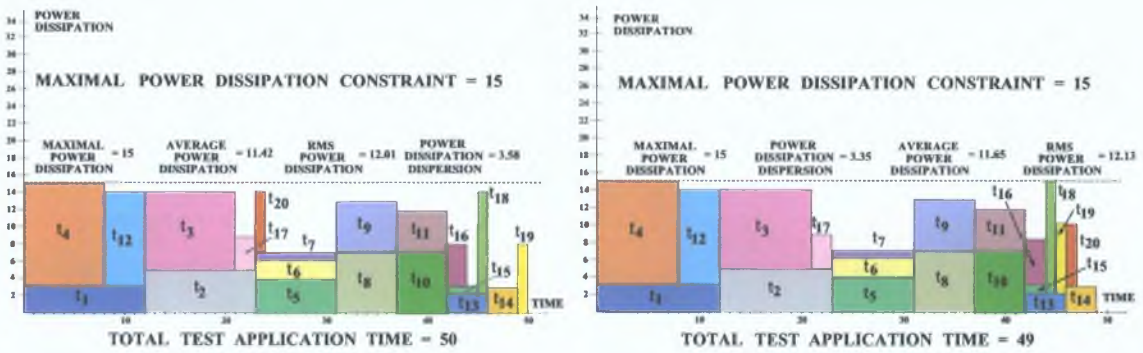
Test t_9 is the last unscheduled test. Consequently, it's *Total Force* does not have incompatibility components. Thus, the assignment of t_9 to rg_6 gives the lowest *Total Force* and, therefore, it is the assignment carried out in the last step of this example. The final power-test scheduling chart of this example is given in figure 5.13(b) and is compared in figure 5.13(a) with the solution generated by the first pseudocode of the PTS-LEA algorithm. It can be observed how power dissipation characteristics can be improved by scheduling test t_9 into the test session for test t_6 instead of scheduling it into the test session for test t_1. Comparing with the PTS-LS algorithm, this is possible in the PTS-FDS algorithm by employing a force-directed priority function in the gap-selection process of the tree growing technique.

Figure 5.14 gives a comparison between the PTS solution generated by the PTS-LS algorithm (the best solution generated in chapter 4 for the 20 BTS example)



(a) PTS-LS Solution (b) PTS-FDS Solution

Figure 5.14: PTS Charts Without Power Constraints - 20 BTS



(a) PTS-LS Solution (b) PTS-FDS Solution

Figure 5.15: PTS Charts With Power Constraints (PDC = 15) - 20 BTS

and the one generated by the PTS-FDS algorithm for the 20 BTS example, without power constraints. It can be noticed that the PTS-FDS solution in 5.14(b) exhibits a more balanced power dissipation than the PTS-LS solution given in figure 5.14(a), even though the total test application time has increased slightly as well. It will be seen in chapter 6 that for very loose power constraints, normally, the PTS-FDS algorithm dramatically improves the power dissipation characteristics (comparing to PTS-LS and PTS-LEA) usually without increasing the total test application time. On the other hand, for tighter power constraints as in the case of figure 5.15(b), it will be seen in chapter 6 that the PTS-FDS algorithm generates solutions with balanced power dissipation but with longer test application time (comparing again to the PTS-LS and PTS-LEA approaches). However, the example given in figure 5.15 is again an exception because the total test application time is not increased by applying the PTS-FDS algorithm comparing to the PTS-LS algorithm).

5.2 Distribution Variance Based Approach

The HLS FDS approach from [KAHA97] was also used the distribution graph concept where a so-called Mean Square Error (MSE) function was used as a priority function to schedule operations to c-steps, resulting in a computationally efficient solution. The schedule of this approach was constructed iteratively, each individual operation was considered, assessed and finally scheduled into the most suitable c-step. This was unlike the previous HLS FDS algorithm where the influence of all unscheduled operations on the schedule was evaluated before the most suitable operation to c-step assignment.

It was proved in [KAHA97] that higher mobility operations have smaller effect on the overall operation distribution because of their long time frames, which result in lower probability values This means that it is unnecessary to schedule high mobility operations early Therefore, the operations were classified in [KAHA97] into a sorted list according to operation mobility (from low to high), increasing ASAP time, and decreasing number of succeeding nodes This list was used to schedule all operations one after another without the need to re-sort the list, thus preserving the algorithm's low complexity On the other hand, it has been explained in section 4 2, that in the PTS problem, the test mobility function has to be employed in order to sort the initial test list The other two sorting keys mentioned above are not applicable to the PTS problem because the order of test application is not important at test session level

The algorithm in [KAHA97] had the following steps The first unscheduled operation op_i was taken out from the sorted list To establish the optimal c-step into which this operation was to be scheduled, the operation was tentatively assigned to all valid c-steps within its time frame Scheduling an operation into a c-step might affect the time frames of its preceding and succeeding operations As a result, the probability values of these operations might vary and, therefore, modified distribution graph values $DG'_j(type, i)$ had to be determined for each tentative c-step j assignment, where *type* represented the type of operation op_i To investigate the effect of different c-step assignments on the operation distribution, the temporary DG'_j was assessed knowing that a good schedule has a balanced DG Let c-steps$_{type}$ be the set of c-steps into which operations of this type could be scheduled Let M_{type} be the number of such c-steps Then the average DG value of this type of operation over their possible c-step assignment is

$$AVG_{type} = \frac{1}{M_{type}} \sum_{i \in c-steps_{type}} DG(type, i), \tag{5 9}$$

Let $DG'_j(type, i)$ be the distribution graph if operation op_i was assigned to c-step j Then the so-called Mean Square Error (MSE) of $DG'_j(type, i)$ is an indication of the balance of the distribution graph of this operation type

$$MSE(j, type) = \sqrt{\frac{1}{M_{type}} \sum_{i \in c-steps_{type}} (DG'_j(type, i) - AVG_{type})^2}, \tag{5 10}$$

There was one MSE value for each operation type and, in order to find an overall rate, all MSE values for all types were added

$$MSE(j) = \sum_{type} C_{type} \quad MSE(j, type), \tag{5 11}$$

where C_{type} was an optional constant reflecting the cost of the particular hardware resource Having determined the MSE values for all valid c-steps, operation op_i was finally scheduled into the c-step j which resulted in the lowest MSE value This was followed by adjusting the time frames of its preceding and succeeding operations and updating the DG values This procedure was repeated until all operations were scheduled

An aside has to be made here with regard to the mathematical definition of the MSE function mentioned above First of all, the formula given in equation 5 10 is not a Mean Square Error (MSE) but resembles the Standard Deviation (SD) definition, which is the square root of the average squared deviations from the mean On the other hand, MSE is the average of squared deviations of the current population from the values of the previous population (see equation 5 12) The approach in [KAHA97] was to calculate the deviation of a new population (i e , DG' values) from the average of the previous population (i e , DG values) because it was aiming at a balanced distribution graph This deviation is calculated with the Standard Deviation (SD) formula On the other hand, the MSE formula calculates the deviation of the new population's values from the values of the previous population In terms of distribution graph, by using the MSE formula the algorithm would aim at minimizing the difference between the shape of the previous distribution graph and the new shape of the distribution graph

$$MSE(j, type) = \sqrt{\frac{1}{M_{type}} \sum_{i \in c-steps_{type}} (DG'_j(type, i) - DG_j(type, i))^2}, \tag{5 12}$$

Moreover, mathematically speaking the Average Value (AVG) in Standard Deviation (SD)'s formula should be constant, which is not the case in the approach employed in formula 5 10 Therefore, the term defined in formula 5 10 should be called a Deviation from a Mean Target, where the mean target (AVG) changes at each iteration

The same idea from [KAHA97] is employed for the PTS algorithm proposed in this section with some amendments First of all, Standard Deviation (SD) formula was considered, but it turns out that the square root function is superfluous This was due to the fact that the SD values had to be compared, and the ordering relationship (i e , $<, >$) between two values is the same for the relationship between

their square or square root values (convex functions) Therefore, the square of the so-called SD from target (AVG) function is eventually implemented which coincides with the mathematical definition of Distribution Variance (DV) from Target (AVG)

Inspired by the approach given in [KAHA97] a Distribution Variance Based PTS Approach (PTS-DV) is proposed in this section together with the extended tree growing technique to deal with the problem of unequal-length block-test scheduling under power dissipation constraints It aims at achieving a balanced power dissipation throughout the test application time merely by assessing the same Power-Concurrency Distribution Graphs (PCDG) proposed in the previous section PCDG gives the effect of test/test-subsession assignments Unlike the PTS-FDS approach given in the previous section, the time consuming *Force* calculations are avoided here by using the DV function, resulting in a more computationally efficient solution This is achieved using only the two steps summarized below

Determination of time frames

The first step consists again of determining the time frames of each test by evaluating the set of test subsessions (ETPs) where the test can be merged The ETPs expandable at a certain moment with a test do not have to be adjacent and, therefore, a test's time frame in PTS-DV, as in PTS-FDS, is not or does not have to be contiguous The goal of PTS-DV is also to balance the power dissipation and, indirectly, the test concurrency, while keeping the test application time as tight as possible Therefore, the same *power dissipation probability* from the PTS-FDS approach is to be employed

For the 4 BTS example given in section 5 1 and depicted in figure 5 5, both tests t_3 and t_4 have two test subsessions (tw_1 and tw_2) in which they can be scheduled That is, the time frames of both tests t_3 and t_4 consist of the test subsessions tw_1 and tw_2

Creation of distribution graphs

The next step is to take the sum of $PCDG_{t_i}(ts_j)$ values for each possible test/test subsession assignment and add them on top of the power dissipation accumulated so far in the partial power-test chart The resulting PCDG indicates the power dissipation expectations PCDG's formula is the same one as formula 5 4 given in section 5 1

First, all tests are ordered by their mobilities Scheduling a test t_i into a certain test subsession ts_j might affect the time frames of other tests As a result, the probability values of these tests may vary and modified distribution graph values $PCDG'(ts_j)$ should be determined for each possible $t_i \rightarrow ts_j$ assignment To investigate the effect of different test subsession assignments on the tests' distribution, the temporary $PCDG'(ts_j)$ is assessed knowing that a good schedule has a balanced $PCDG$ The difference between the $PCDG$ values and the average value AVG provides an indication of the distribution balance as in the [KAHA97] approach The average value can be obtained from the original $PCDG$ using the following formula

$$AVG = \frac{1}{N_{TS}} \sum_{i=0}^{N_{TS}-1} PCDG(i), \tag{5 13}$$

where N_{TS} is the number of test subsessions in the schedule The *distribution variance* of the temporary $PCDG'$ from the AVG value gives an indication of the balance of the power dissipation distribution The DV function to be calculated for each tentative assignment of a test t_i to a test subsessions ts_j is defined below

$$DV_{t_i \rightarrow ts_j} = \frac{1}{N_{TS}} \left(\sum_{j=0}^{N_{TS}-1} (PCDG'(ts_j) - AVG)^2 \right), \tag{5 14}$$

where $PCDG'(ts_j)$ are the values of the modified power-concurrency distribution graph (in each test subsession ts_j) for a tentative $t_i \rightarrow ts_j$ assignment As in [KAHA97], having determined the $DV_{t_i \rightarrow ts_j}$ values for all valid test subsessions ts_j (to which test t_i can be assigned), the test t_i is finally scheduled into the test subsession exhibiting the lowest $DV_{t_i \rightarrow ts_j}$ value This is followed by adjusting the time frames of the *remaining tests* in ts_j's compatibility list and updating the $PCDG$ values The above steps are repeated until all tests are scheduled

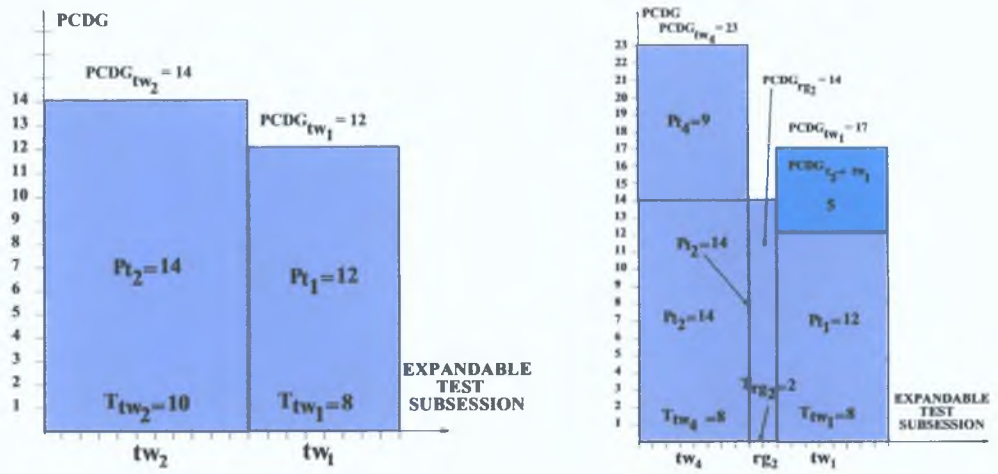
Thus far a uniform time length (one time unit) has been considered for all the test subsessions in which the PCDG has been calculated Recall that in the PTS-FDS approach, the time length has not been employed in the calculation of the *Force* function The reason behind this is the fact that the PTS algorithms deal with the idealistic technological case, where each power spike has to be minimized On the other hand, in the PTS-DV approach test subsessions' time length can be employed for the calculation of AVG and $DV_{t_i \rightarrow ts_j}$ values in order to better discretise the PCDG over the test application time

$$AVG = \frac{1}{TL} \sum_{i=0}^{N_{TS}-1} T_{ts_i} * PCDG(i), \tag{5.15}$$

Thus, an improvement of formulas 5.13 and 5.14 to also take into account the time component of test subsessions is given, respectively, in formulas 5.15 and 5.16. The AVG value of the PCDG was calculated before in formula 5.13 by uniformly (by one time unit) sampling the PCDG values for each test subsession (in total N_{TS} test subsessions). On the other hand, in formula 5.15 the AVG value of the PCDG is calculated this time by sampling the predicted PCDG values for each time unit of the total application time or total test length TL (where $TL = \sum_{j=0}^{N_{TS}-1} T_{ts_j}$). That is, the predicted PCDG value of test subsession ts_j will be sampled T_{ts_j} times and will have the same constant value $PCDG(ts_j)$. The $DV_{t_i \to ts_j}$ values are calculated in the same way as in formula 5.16. The distribution is now discretised TL times. Therefore, its population has TL values. Within each test subsession ts_j the new PCDG values are constant ($PCDG'(ts_j)$) for T_{ts_j} time units.

$$DV_{t_i \to ts_j} = \frac{1}{TL} \left(\sum_{j=0}^{N_{TS}-1} T_{ts_j} * (PCDG'(ts_j) - AVG)^2 \right), \tag{5.16}$$

In figure 5.16 are given the PCDG charts for the example given in section 5.1 and depicted in figure 5.5. Figures 5.16(a) and 5.16(b) are the correspondents, respectively, of figures 5.6 and 5.7(a) after formulas 5.15 and 5.16 have been employed.



(a) Before t_4 is Scheduled in tw_2 (b) After t_4 is Scheduled in tw_2

Figure 5.16: PCDGs with Test Length Component

Thus, the PCDG before tentatively scheduling test t_4 to any of the expandable

test subsessions (tw_2 or tw_1) is depicted in figure 5 16(a) The average value of PCDG is now $AVG = (T_{tw_2} * PCDG(tw_2) + T_{tw_1} * PCDG(tw_1))/(T_{tw_2} + T_{tw_1}) = (140 + 96)/(10 + 8) = 13\ 11$ The PCDG after the tentative scheduling of test t_4 in test subsession tw_2 is given in figure 5 16(b) The DV values generated by tentatively scheduling test t_4 to tw_2 and tw_1 are calculated below From these DV values it can be concluded that the smallest deviation is given by the $t_4 \rightarrow tw_1$ assignment That is, the power distribution deviation $DV_{t_4 \rightarrow tw_2} = 50\ 28$ from the average value $AVG = 13\ 11$ of the previous PCDG given by the assignment of $t_4 \rightarrow tw_2$ is bigger than the deviation $DV_{t_4 \rightarrow tw_1} = 46\ 94$ generated in the PCDG by the $t_4 \rightarrow tw_1$ assignment

$$
\begin{aligned}
DV_{t_4 \rightarrow tw_2} &= \frac{1}{T_{tw_4} + T_{rg_2} + T_{tw_1}} (T_{tw_4} * (PCDG'(tw_4) - AVG)^2 + \\
&\quad + T_{rg_2} * (PCDG'(rg_2) - AVG)^2 + T_{tw_1} * (PCDG'(tw_1) - AVG)^2) \\
&= \frac{1}{18} (8 * (23 - 13\ 11)^2 + 2 * (14 - 13\ 11)^2 + 8 * (17 - 13\ 11)^2) \\
&= \frac{782\ 5 + 1\ 58 + 121\ 06}{18} = \frac{905\ 14}{18} = 50\ 28, \\
DV_{t_4 \rightarrow tw_1} &= \frac{1}{T_{tw_2} + T_{tw_4} + T_{rg_1}} (T_{tw_2} * (PCDG'(tw_2) - AVG)^2 + \\
&\quad + T_{rg_1} * (PCDG')(rg_1) - AVG)^2 + T_{tw_4} * (PCDG'(tw_4) - AVG)^2) \\
&= \frac{1}{18} (10 * (19 - 13\ 11)^2 + 0 * (12 - 13\ 11)^2 + 8 * (21 - 13\ 11)^2) \\
&= \frac{346\ 92 + 498\ 01}{18} = 46\ 94
\end{aligned}
$$

5.2 1 Algorithm Pseudocode

The pseudocode of the PTS-DV algorithm is given below The data structures used in it are the same as in all previous PTS approaches (see subsection 4 1 1) As it can be seen in the pseudocode, tests are firstly sorted by their test mobility The schedule is iteratively developed while individual tests are considered in order, assessed and finally scheduled into the most suitable test subsessions At each iteration one test is scheduled The first stage of each iteration detects (as in the PTS-FDS approach) the tests having null time frames These tests will be considered as test sessions and inserted as roots into the *Growing Tree* When all the tests left unscheduled have time frames different than zero, the first unscheduled test is picked up from the sorted list To establish the optimal test subsession into which it will be scheduled, test t_i is tentatively assigned to each expandable test subsession within its time

frame Subsequently, time frames are updated for all tests affected by the tentative $t_i \longrightarrow ts_j$ assignment and the PCDG is constructed Eventually, the result of each tentative assignment is a DV value At each iteration, t_i is scheduled into the test subsession ts_j for which the lowest DV value was found Then the time frames of the unscheduled tests are consequently updated together with the distribution graph PCDG

PSEUDOCODE of PTS-DV Algorithm

◇initialize and sort all tests according to their test mobility with two keys (test length and power dissipation),

◇initialize the *GrowingTree*, the *BlockTestList* and the *GapsList*,

◇while there are unscheduled tests do { /*$BlockTestList$ is not empty*/

- **take the next test out from the sorted list,**
- **evaluate time frames for all tests,**
- **while (there are tests having null time frames) do{**
  - ***CurTest* = the first out of *BlockTestList* having null time frame**
  - **insert *CurTest* as the tail of *GrowingTree* roots /*new test session*/ and make *CurTest* "used",**
  - **remove *CurTest* from *BlockTestList*,**
  - **generate a *TwinGap* gap as the twin of *CurTest*,**
  - **insert *TwinGap* into *GapsList*,**
  - **evaluate time frames for all tests, } /*while*/**
- **for each test subsession into which the test could be scheduled do {**
  - **assign test t_i tentatively to test subsession ts_j,**
  - **update time frames of remaining tests in ts_j's compatibility list,**
  - **calculate distribution graph for the modified growing tree,**
  - **evaluate the distribution variance (DV) function,**
- **} /*for*/**
- **Schedule test into the test subsession for which the lowest DV value was found,**
- **update time frames of remaining tests in ts_j's compatibility list,**
- **update distribution graph,**

◇} /*while*/

5.2.2 Algorithm Complexity

The complexity of the PTS-DV algorithm can be derived in the following way. Firstly, each iteration of the algorithm schedules one test. This implies there are N iterations (N is the initial number of tests). Secondly, within each iteration, for a test to be scheduled, there are at most N test subsessions (gaps) for which $PCDG$ must be calculated. Finally, for each tentative test-to-test subsession assignment, there may be at most N test subsessions for which the new $PCDG$' values have to be calculated. This assumption is a conservative upper bound. The combined effect of the above three considerations yields the combined $O(N^3)$ complexity.

5.2.3 Test Scheduling Example

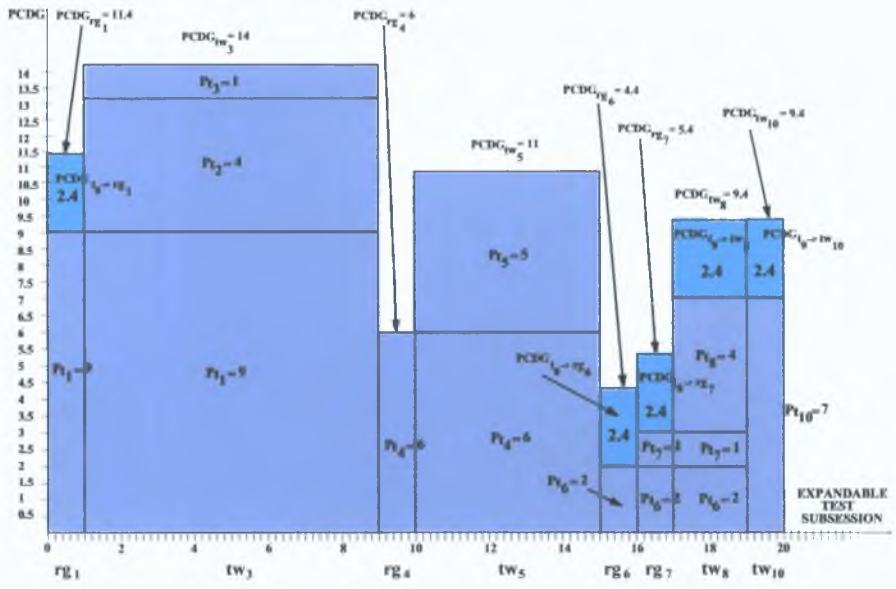


Figure 5.17: PCDG - Step 10 (PTS-DV Approach)

The PTS solution generated by the PTS-DV algorithm for the 10 BTS example given in subsection 4.1.3 is the same as the solution generated by the PTS-FDS algorithm in figure 5.13(b). This is the same for both algorithms because the BTS size is too small and both algorithms converge this time to the same solution inside a small solution space. Test scheduling steps of this example happen to be exactly the same as in figure 5.8. That is mainly due to the fact that for this BTS example only in the last step are taken calculations-based decisions. However, the decision in step 10 is not based on *Forces* anymore, but on DV calculations. First of all, the

$PCDG$ is constructed as in figure 5 17, which is an update of figure 5 12 Then, its average power dissipation is calculated according to formula 5 15 as below

$$TL = T_{rg_1} + T_{tw_3} + T_{tw_5} + T_{rg_4} + T_{rg_6} + T_{rg_7} + T_{tw_8} + T_{tw_{10}},$$

$$\begin{aligned}\sum_{i=0}^{N_{TS}-1} T_{ts} * PCDG(i) &= T_{rg_1} * PCDG(rg_1) + T_{tw_3} * PCDG(tw_3) + T_{tw_5} * PCDG(tw_5) + \\ &+T_{rg_4} * PCDG(rg_4) + T_{rg_6} * PCDG(rg_6) + T_{rg_7} * PCDG(rg_7) + \\ &+T_{tw_8} * PCDG(tw_8) + T_{tw_{10}} * PCDG(tw_{10}),\end{aligned}$$

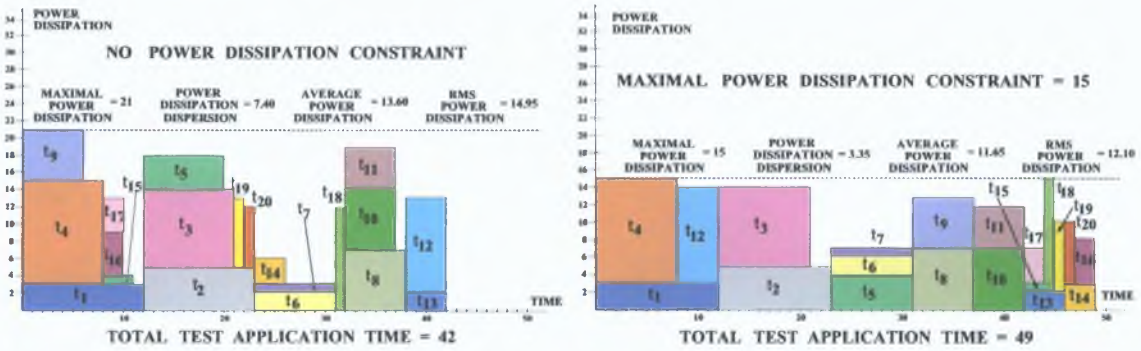
$$\begin{aligned}AVG &= \frac{1}{TL}\sum_{i=0}^{8-1} T_{ts_i} * PCDG(i) \\ &= \frac{1*11\,4 + 8*14 + 5*11 + 1*6 + 1*4\,4 + 1*5\,4 + 2*9\,4 + 1*9\,4}{1+8+5+1+1+1+2+1} \\ &= \frac{222\,4}{20} = 11\,12\end{aligned}$$

Then the DV values are calculated for each tentative $t_9 \rightarrow ts_j$ assignment using formula 5 14 Test t_9 is the last to be scheduled The set of test subsessions at this stage is $tw_{10}, rg_1, rg_6, tw_8, rg_7$ After any of the tentative assignments the number of gaps increases because a test subsession is replaced with two test subsessions (i e, a twin gap and a rest gap) For example, the assignment of t_9 to tw_{10} exchanges the latter with rg_{10} and tw_9 as in the below calculations of $DV_{t_9 \rightarrow tw_{10}}$ The least DV value is given by the $t_9 \rightarrow rg_6$ assignment (see below)

$$\sum_{j=0}^{N_{TS}-1} T_{ts_j} * (PCDG'(ts_j) - AVG)^2 =$$

$$\begin{aligned}= &T_{rg_{10}} * (PCDG(rg_{10}) - AVG)^2 + T_{tw_9} * (PCDG(tw_9) - AVG)^2 + \\ &+T_{rg_1} * (PCDG(rg_1) - AVG)^2 + T_{tw_3} * (PCDG(tw_3) - AVG)^2 + \\ &+T_{tw} * (PCDG(tw_5) - AVG)^2 + T_{rg_4} * (PCDG(rg_4) - AVG)^2 + \\ &+T_{rg_6} * (PCDG(rg_6) - AVG)^2 + T_{tw_8} * (PCDG(tw_8) - AVG)^2 + \\ &+T_{rg_7} * (PCDG(rg_7) - AVG)^2,\end{aligned}$$

$$\begin{aligned}DV_{t_9 \rightarrow tw_{10}} = &\frac{1}{20}(0 * (7 - 11\,12)^2 + 1 * (19 - 8\,875)^2 + 1 * (9 - 11\,12)^2 + \\ &+8 * (14 - 11\,12)^2 + 5 * (11 - 11\,12)^2 + 1 * (6 - 11\,12)^2 + \\ &+1 * (2 - 11\,12)^2 + 2 * (7 - 11\,12)^2 + 1 * (3 - 11\,12)^2) = \frac{342\,29}{20} = 17\,11,\end{aligned}$$



(a) Without Power Constraints (b) With Power Constraints = 15

Figure 5.18: PTS Charts of PTS-DV Approach - 20 BTS

$$
\begin{aligned}
DV_{t_9 \rightarrow rg_1} &= 19.51; \\
DV_{t_9 \rightarrow rg_6} &= 11.11; \\
DV_{t_9 \rightarrow tw_8} &= 17.11; \\
DV_{t_9 \rightarrow rg_7} &= 12.31.
\end{aligned}
$$

In figure 5.18 are depicted the PTS chart solutions given by the PTS-DV algorithm for the 20 BTS example without (figure 5.18(a)) and with (figure 5.18(b)) power constraints ($PD_{max} = 15$). Comparing the PTS charts from figures 5.18, 5.14, and 5.15 it can be noticed for the 20 BTS that the PTS-DV approach outputs more or less the same power-test scheduling charts as generated by the PTS-FDS algorithm. This is also due to the fact that the 20 BTS example is still small to be able to fully differentiate the characteristics of PTS-FDS and DV algorithms. This can be done later on with the examples given in chapter 6.

5.3 Mixed List and Force-directed Approach

Mixed classical scheduling algorithms are proposed here to further improve the test concurrency having assigned loose power dissipation limits. A sequence of list and distribution-graph based scheduling algorithms is adapted to tackle the power-test scheduling problem. The extended tree growing technique is again the background technique for these algorithms in order to model the PTS problem. This mixed scheduling approach contains mainly two steps. Firstly, a list scheduling-based algorithm (PTS-LEA, PTS-LS) is run in order to rapidly achieve a power-test scheduling solution with a near-optimal test application time. Then the power

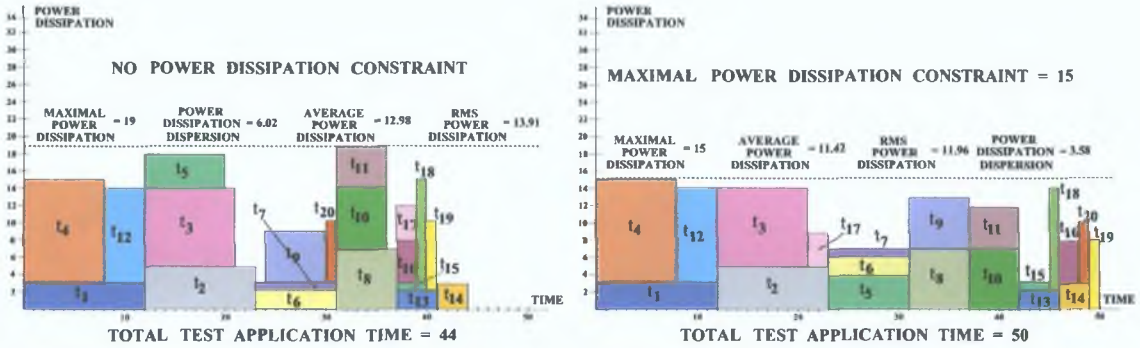
dissipation distribution of this solution is balanced by applying a distribution-graph based scheduling algorithm (PTS-FDS, PTS-DV)

It has been noticed in [MWMV00] (see chapter 6) that the list scheduling-based approaches (PTS-LEA and PTS-LS) rapidly generate results which usually exhibit very good test application times The shortcoming of their solutions is that their power dissipation characteristics are poor That is because the power dissipation distribution is not balanced These characteristics can be improved by the distribution-graph based approaches (PTS-FDS and PTS-DV) The distribution-graph based approaches take more computational time on calculating the global priority function (Forces or Distribution Variance (DV)) but render solutions without power spikes This is the goal of the two-step approach proposed in this section Firstly, to generate a solution exhibiting a good test application time using list scheduling-based approaches And secondly, to balance the power dissipation distribution of this solution by running on it distribution-graph based approaches With the algorithms available in this thesis four versions of this approach can be named

- Mixed LEA - FDS Based PTS Approach (PTS-LEAFDS) - PTS-LEA to accomplish the first step and PTS-FDS for the second step,
- Mixed LEA - DV Based PTS Approach (PTS-LEADV) - PTS-LEA for the first step and PTS-DV for the second step,
- Mixed LS - FDS Based PTS Approach (PTS-LSFDS) - PTS-LS for the first step an d PTS-FDS for the second step,
- Mixed LS - DV Based PTS Approach (PTS-LSDV) - PTS-LS for the first step and PTS-DV for the second step

5 3 1 Algorithm Pseudocode

The pseudocode of PTS-LEAFDS (PTS-LSFDS, PTS-LEADV, PTS-LSDV) algorithms is very simple and, basically, assumes the sequential execution of the PTS-LEA (PTS-LS) and PTS-FDS (PTS-DV) algorithms Below is given the simple pseudocode of the PTS-LSFDS algorithm, the pseudocode of the other mixed approaches are very similar



(a) Without Power Constraints (b) With Power Constraints = 15

Figure 5.19: PTS Charts of PTS-LSFDS Approach - 20 BTS

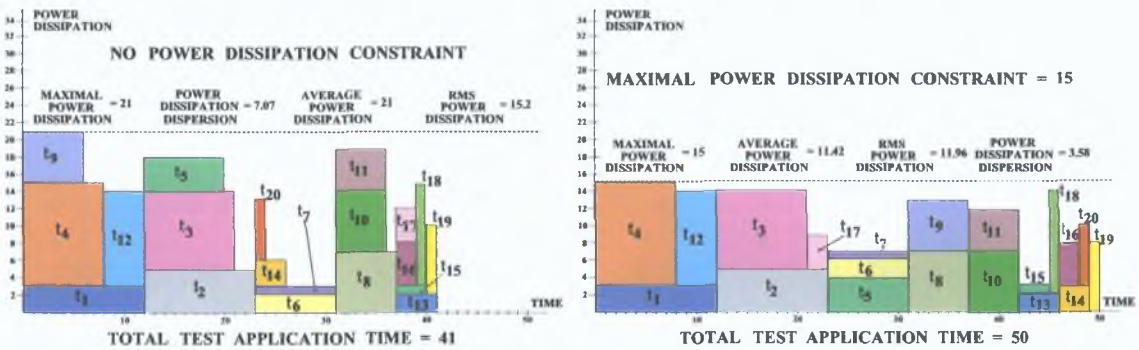
PSEUDOCODE of PTS-LSFDS Algorithm:
-execute the PTS-LS algorithm on the initial *BlockTestList*;
-initialize the *GrowingTree* **keeping only its roots in the structure;**
-execute the PTS-FDS algorithm on the updated (initialized) *GrowingTree*;

In the pseudocode, the intermediate step between the two main scheduling steps assumes the updating of the growing tree. The test application time of a power-test scheduling solution is equal to the sum of the test lengths given by growing tree's roots. In order to keep this time characteristic unchanged, the distribution-graph based algorithm starts re-shuffling the tests leaving the root tests (test session set) unchanged in the growing tree. By using the global priority function the tests are re-accommodated in the newly updated growing tree (in the roots test sessions) by generating new test subsessions.

5.3.2 Algorithm Complexity

The complexity of the PTS-LS approach is $O(N^2)$, while the complexity of PTS-FDS is $O(N^3)$. Since this approach runs PTS-LS and PTS-FDS sequentially, its complexity is $O(N^3)$. However, it can be noticed in chapter 6 that even though, intuitively, the Central Processing Unit (CPU) time of the mixed scheduling approach should be approximately the sum of the CPU times taken by the sequenced list and distribution-graph based algorithms, this is not the case. The CPU time taken by the mixed scheduling algorithms is much longer than the sum of list and distribution-graph based scheduling algorithms and the discrepancy grows along with the dimension of the BTS. This increase in CPU time is due to the increase

of the CPU time component given by the distribution-graph algorithm. This fact proves the idea that after the first step, by keeping the roots of the growing tree as an initial (partial) solution for the second step algorithm, the search space has increased. Intuitively, this can be explained by the fact that keeping the roots in the growing tree unchanged makes the *GapsList* bigger from the beginning. Thus, this new dimension of the *GapsList* results in more branches during solution searching.



(a) Without Power Constraints (b) With Power Constraints = 15

Figure 5.20: PTS Charts of PTS-LSDV Approach - 20 BTS

5.3.3 Test Scheduling Example

For the same reasons given in subsection 5.2.3 (similarity among the scheduling charts generated by PTS-FDS, PTS-DV, and PTS-LSFDS approaches for the 10 BTS example), here are given only the PTS charts generated for the 20 BTS example. Figure 5.19 depicts the results of running the PTS-LSFDS algorithm without and with power constraints, while figure 5.20 gives the results for running the PTS-LSDV algorithm for the same power constraints conditions.

It can be noticed comparing figures 5.14, 5.15, 5.18, 5.19, and 5.20 that for the 20 BTS example the distribution-graph based approaches exhibit more or less the same features and it is hard to differentiate them with a relatively small BTS example. Therefore, in the next chapter more experiments are carried out in order to find out the differences between them in terms of advantages and disadvantages. However, one can anticipate one of the conclusions of chapter 6. That is, the PTS-LSFDS and the PTS-LSDV approaches improve the power dissipation distribution while keeping constant the total test application time only for loose power constraints.

Chapter 6

Experimental Results

6.1 Implementation

For the implementation of the algorithms proposed in this thesis and for their application on the testbenches, Visual C++ was chosen as the development environment. The reason for using the Visual C++ environment was the ease of developing a simple graphical interface. Thus, with the graphical interface designed, it is easy to handle, stepwise or otherwise, PTS algorithms, testbenches and experimental results. The graphical interface is given in figure 6.1. It has basically four split-windows which are handy when the user tries to follow step by step how the algorithm works. A user can visualize the test set in the top split-window with all its block-tests and their characteristics (test length, power dissipation, test compatibility list). Then, in the second split-window, the growing tree's content and structure can be browsed, expanded or suppressed. In the third split-window are visualized characteristics of the node (one of the growing-tree's nodes) selected by the user in the previous split-window. The characteristics of the gaps left for expansion while the algorithm is run are listed in the last split-window.

The user is provided in the dialog bar with buttons to run algorithms step by step, to run them at once, to generate a test set, and to simulate the results of all algorithms while power dissipation constraints are ranged from loose to tight. Edit-type dialog fields are provided to give the user the facility of setting the limits between which power dissipation constraint is decreased during simulation: maximal power dissipation constraint, minimal power dissipation constraint, and the decrement step size. Then, parameters used in the random generation of new testbenches

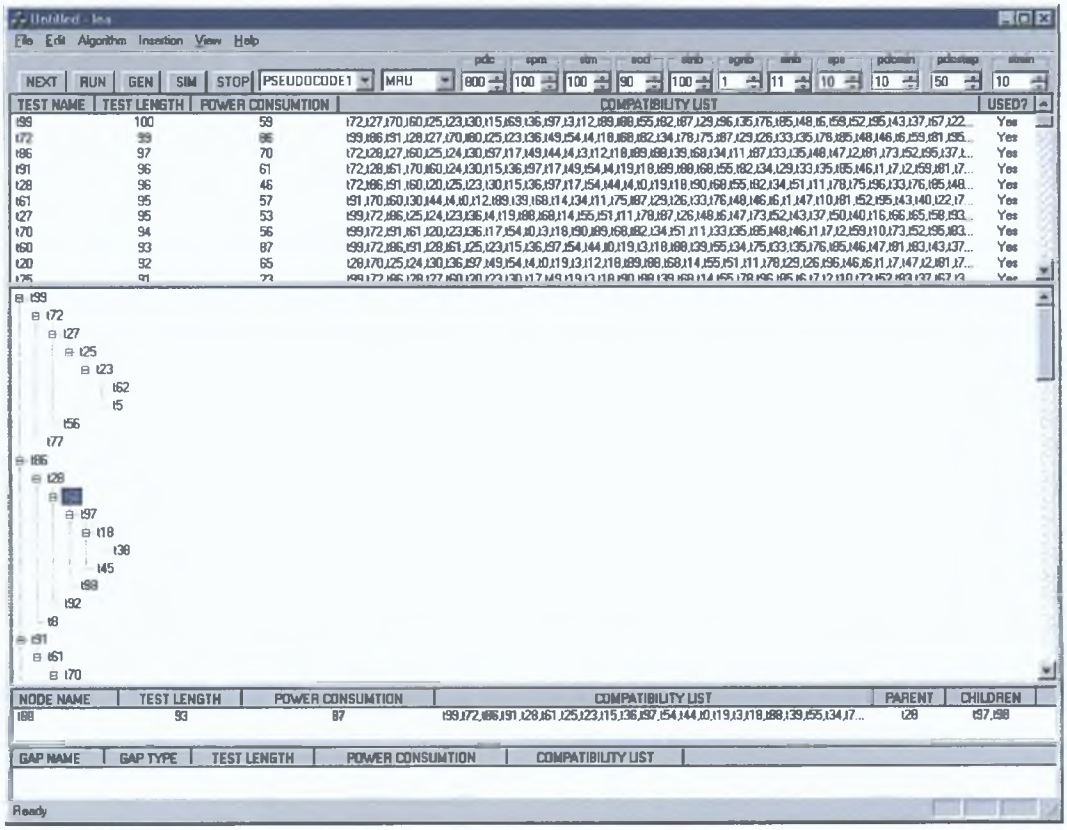


Figure 6.1: GUI interface

can be set in the dialog bar as well: maximal power dissipation characteristic value, minimal power dissipation characteristic value, maximal test length value, minimal test length value and test resource compatibility degree (percentage). If the compatibility degree is left at zero, then the generator assigns to it a random value between 0% and 100%.

Other fields from the dialog bar are the following: number of tests in the BTS to be generated; number of testbenches generated for the same set of tests, where only the compatibility degree between tests is changed for each of the testbenches. This facility is useful when the user wants to see the behaviour of proposed algorithms over a range of test resource compatibility degrees. Different random testbenches can be developed within seconds. However, testbenches can also be manually generated by means of dialog bar and menu commands, or already generated testbenches can be modified by the same means.

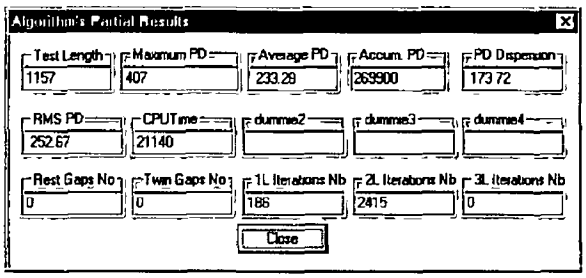


Figure 6 2 Results Window

The user can also see at each step the partial/final power-test characteristics of the generated schedules by enabling a *results* window like the one depicted in figure 6 2 Also, the whole set of algorithms can be run automatically on the chosen testbenches so that the user does not have to interact with the interface The PTS solutions can be saved in two formats

- one format to save the structure of the extended growing tree for the currently generated test schedule,
- another format to save the PTS chart This format is saved as a series of data importable and further processable in Excel or any other tool similar to it Thus, PTS charts can be visualized and processed by Excel-like utilities The results saved in this format are used to depict pictures of the PTS charts

6.2 Experimental Results

In practical circuits (e g MCMs) only a few blocks or modules are activated at a certain moment, under normal system operation, while other blocks are in the power-down mode to minimize the power dissipation However, in order to test the system in the shortest possible time, it is desirable to concurrently activate as many blocks as possible provided that the power dissipation limit of the system is not exceeded For low compatibility degree BTSs, the PTS algorithms exhibit a very short run time However, the complexity of the solution space and the time allocated for optimal solution search increase a lot with the increase of compatibility degree between tests This convinced the author to focus on the results given by the testbenches with a high resource compatibility degree between the test resources Actually, this set of testbenches or real-life cases is what the PTS algorithms proposed in this thesis is aimed at

Because in complex VLSI circuit designs the BTS is large and varies in test length, it is possible to schedule some short tests to begin when subcircuits with shorter testing time have finished testing, while other subcircuits with longer testing time have not (obviously, if they are resource compatible) The author wants to prove in this chapter by experiments that the *extended tree growing heuristic* proposed in chapter 3 is very productive from this point of view Random testbenches in which the test length of the BTS is ranged are generated The experiments prove indeed the efficiency of the proposed algorithms for BTSs with a high resource compatibility degree

When PTS algorithms are employed for PTS optimization at higher levels in the modular test hierarchy, there will be a smaller BTS Intuitively, this is due to longer test length blocks at higher levels At system level the number of blocks usually does not exceed a dozen The difficulty of the PTS process is then believed to decrease almost exponentially Therefore, the big battle for PTS improvements is given at RTL level

The experiments have been run on a Pentium II machine of 600MHz All testbenches apart from the last set were randomly generated The structure of these testbenches is given in appendix A This chapter is structured on six main sets of testbenches The content of the testbenches processed in this chapter is described below

- the first experiment is run on a 50 BTS randomly generated The degree of resource compatibility varies for the same BTS within a range from low to high The maximum power dissipation characteristic value of the tests from the BTS is 20 and the their maximum test length characteristic value is 20 The minimum power dissipation value of the tests from the BTS is 1 and their minimum test length value is 1 The tests' power and time length values are generated without regard to their similarity as will be the case in subsequent experiments,

- the second experiment is run on a 50 BTS randomly generated as well This time the degree of resource compatibility varies for the same BTS within a range from low to high, but with a finer increment, namely 10% Thus, 9 testbenches have been generated for the same BTS, with the compatibility degree ranging from 10% up to 90% The maximum power dissipation value of the tests from the BTS is 100 and their maximum test length value is 100

The minimum power dissipation value of the tests from the BTS is 10 and their minimum test length value is 10,

- for the third experiment 9 testbenches consisting also of 50 tests are generated The resource compatibility amongst the tests is kept constant at 90%, the maximum test length value of the tests is set at 100, and their minimum test length value is set at 10 This time a spectrum of tests' power dissipation values is tried out The maximum power dissipation values of the test from the BTS are kept at 100 all the time, but their minimum power dissipation values are increased stepwise by 10 from 10 to 90 These experiments test the impact of BTSs with tests exhibiting, on one hand, very different power dissipation values and, on the other hand, very similar power dissipation values,

- the fourth experiment consists of 9 testbenches with 50 tests each The tests' resource compatibility is kept constant at 90%, and also constant is kept the BTS's power dissipation spectrum This time the minimum test length value of the test from the BTS is increased stepwise by 10 from 10 to 90 Thus, the experiments test the impact of BTSs with tests having, on one hand, very different test length and, on the other hand, very similar test length,

- the fifth experiment is the most complex one 27 testbenches are generated for varied values of power dissipation, test length and test resource compatibility degree Each characteristic is set to one of the three spectra below

  - broad range spectrum, in which the minimum value of the test can range between 10% and 100% of the maximum value,
  - average range spectrum, in which the minimum value of the test can range between 50% and 100% of the maximum value,
  - narrow range spectrum, in which minimum value of the test can range between 90% and 100% of the maximum value

- the sixth experiment is based on a practical case test set which is an extension [LP00b] of the ASIC Z design given in [Zor93] The test set has 27 tests spread over 9 cores, employing 11 test resources The BTS has a repetitive structure and, intuitively, could be clustered in independent sets

The following simplified notations and acronyms will be used throughout this chapter

- for PTS algorithms
  - P1 - first version of PTS-LEA algorithm,
  - P2 - second version of PTS-LEA algorithm,
  - P3 - third version of PTS-LEA algorithm,
  - LS - PTS-LS algorithm,
  - LS2 - PTS-LS^2 algorithm,
  - FDS - PTS-FDS algorithm,
  - DV - PTS-DV algorithm,
  - P1FDS - PTS-LEAFDS algorithm firstly running P1 as the PTS-LEA algorithm,
  - P2FDS - PTS-LEAFDS algorithm firstly running P2 as the PTS-LEA algorithm,
  - P3FDS - PTS-LEAFDS algorithm firstly running P3 as the PTS-LEA algorithm
  - P1DV - PTS-LEADV algorithm firstly running P1 as the PTS-LEA algorithm,
  - P2DV - PTS-LEADV algorithm firstly running P2 as the PTS-LEA algorithm,
  - P3DV - PTS-LEADV algorithm firstly running P3 as the PTS-LEA algorithm,
  - LEAFDS - general case of PTS-LEAFDS algorithms,
  - LEADV - general case of PTS-LEADV algorithms,
  - LSFDS - PTS-LSFDS algorithm,
  - LSDV - PTS-LSDV algorithm,
- for the gap insertion approach employed by the list scheduling based algorithms (i e , PTS-LEA, PTS-LS)
  - MRU - Most Recently Used Insertion,
  - LRU - Least Recently Used Insertion,
  - INSITU - In-Situ Insertion,

- RAND - Random Insertion,
- ORD - Ordered Insertion,
- combinations of acronyms are used, e g P2LRU means the P2 algorithm run with the LRU insertion approach for gaps,

- for the power-test characteristics
  - TL - Test Length,
  - MPD - Maximal Accumulated Power Dissipation,
  - APD - Average Power Dissipation,
  - PDD - Power Dissipation Dispersion,
  - RMS - Root Mean Square Power Dissipation,
  - CPU - Central Processing Unit time taken by the algorithm (**given in milliseconds**),
- PDC - power dissipation constraint value

6 2 1 First Experiment

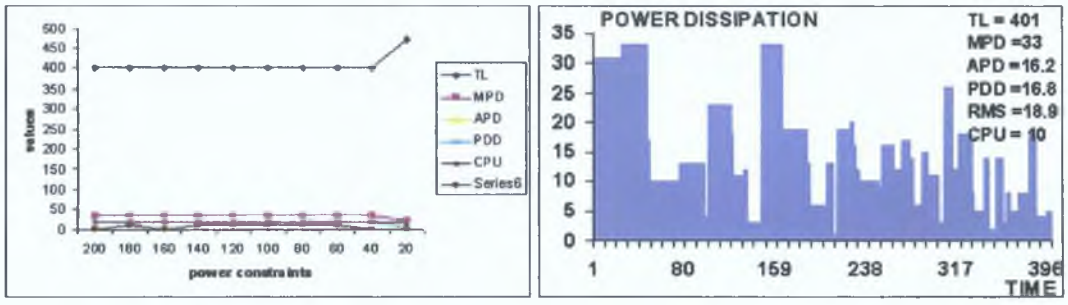
- tests' resource compatibility degree is ranged from low to high,
- tests' power and test length values are randomly generated without regard to their similarity

power cons	P1MRU					
	TL	MPD	APD	PDD	RMS	CPU
200	401	33	16	17	18 5	< 1
180	401	33	16	17	18 5	10
160	401	33	16	17	18 5	< 1
140	401	33	16	17	18 5	10
120	401	33	16	17	18 5	10
100	401	33	16	17	18 5	10
80	401	33	16	17	18 5	10
60	401	33	16	17	18 5	10
40	401	33	16	17	18 5	< 1
20	472	20	13 6	6 4	14 5	< 1

Table 6 1 P1MRU Results (L case)

The first example gives the scheduling results of the proposed PTS algorithms for a randomly generated 50 BTS Their degrees of resource compatibility are increased within a range from low to high low (L) 10%, average-low (A-L) 30%, average (AV) 50%, average-high (A-H) 70% and high (H) 90% The values of the other two characteristics of the tests from the BTS are randomly generated without regard to their values' similarity Section 3 3 5 gives the definitions of the power-test characteristics generated by the PTS algorithms

Case L: Low (L) resource compatibility case



(a) Power-Test Characteristics (b) PTS Chart (PDC = 200)

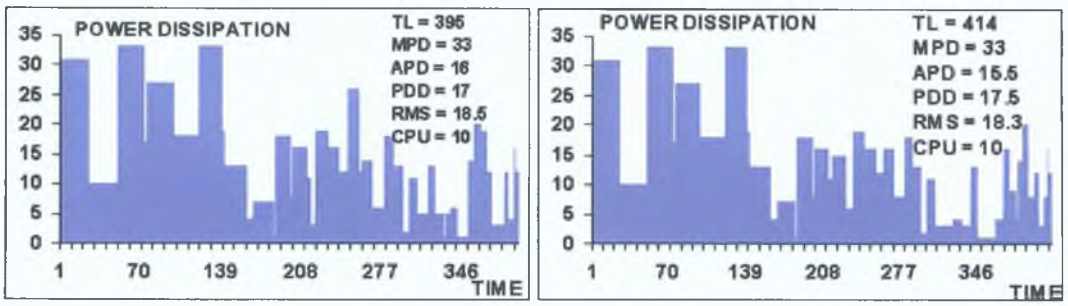
Figure 6.3: Typical Results Generated with P1MRU (L case)

For the low resource compatibility case, the power-test characteristics (see figure 6.3(a)) are almost the same for all list-scheduling based algorithms (i.e., all PTS-LEA versions and PTS-LS). The characteristics' values of all the list-scheduling based algorithms are almost constant for any given power limit, apart from the P2RAND algorithm. For P2MRU, P2INS, P2ORD the values are somehow different from the other list-scheduling based algorithms, but within a very small range of 2-3%. The typical results of P1MRU over a range of power dissipation constraints from loose to tight are given in table 6.1. Their power-test characteristics over the same range of power dissipation constraints are plotted in figure 6.3(a). The curves given by the power-test characteristics' values over the power constraints range are almost flat, that is they are almost constant. These results prove that the PTS algorithms' effect is diminished by the low degree of test resource compatibility. That is the scheduling solution space is reduced by a low resource compatibility between tests, and the schedules exhibit similar power dissipation distributions over similar test application time values.

The power-test chart of the test schedule obtained by the same algorithm (i.e. P1MRU) for a very loose power dissipation constraint (PDC = 200) is given as an example in figure 6.3(b). It can be noticed that the power dissipation distribution is remotely unbalanced resulting in rather frequent power spikes.

The best results for this test set are obtained for the case without power constraints (PDC = 200) and are generated with the P2RAND algorithm. It has to be emphasized here that this algorithm gives randomly different results at each iteration. The aforementioned best schedule randomly generated after several iterations

has the following power-test characteristics which are not very far from the ones presented in figure 6.3: TL = 395, MPD = 33, APD = 16.2, PDD = 16.8, RMS = 18.9, CPU = 10 ms. Its PTS chart is depicted in figure 6.4(a). As it can be seen in this schedule, there is a big difference between the power constraint value (PDC = 200) and the MPD value (MPD = 33). That proves the fact that power constraints would take effect (i.e., would start constraint) for the low compatibility degree case only, when it is set below the MPD = 33 value. The difference between MPD and the other power characteristics (i.e. APD, PDD, and RMS) is around 50%, that is the charts are very unbalanced.



(a) Best Result (PDC = 200) (b) Worst Result (PDC = 180)

Figure 6.4: PTS Charts Generated with P2RAND (L case)

The worst results are generated by the same P2RAND algorithm for a power dissipation constraint value of PDC = 180: TL = 414, MPD = 33, APD = 15.5, PDD = 17.5, RMS = 18.3, CPU < 1 ms. Its PTS chart is given in figure 6.4(b). As it can be seen the characteristics' trend of the schedules generated with the P2RAND algorithm is not monotonous. However, comparing the charts in figure 6.4, it can be noticed that both the best and the worst generated power-test schedules are unbalanced for the case of low-degree compatibility. Moreover, the PTS charts in figures 6.3(b) and 6.4 are similar in terms of power distribution and power-test characteristics. For example, the total test application time characteristic (TL - test length), which ranges the most between the previously depicted three PTS charts, varies with only 3% between the best and the worst PTS solutions generated by the list-scheduling based algorithms.

The experimental results given by the distribution-graph based PTS algorithms for the low resource compatibility test set are discussed below. In subsection 5.1.1 it has been theoretically discussed and demonstrated that there was an important

conceptual difference between the original FDS algorithm and the PTS-FDS algorithm It is further emphasized and experimentally proven here that the exact implementation of the FDS algorithm does not give good results for the PTS problem For example, the results generated by this exact implementation for a PDC = 200 (but actually for any power constraint) are far from the ones given by the other (list-scheduling and distribution-graph based) algorithms TL = 426, MPD = 33, APD = 15 1, PDD = 17 9, RMS = 17 8, CPU = 300 ms

power cons	DV					
	TL	MPD	APD	PDD	RMS	CPU
200	397	33	16 2	16 8	18 8	271
180	397	33	16 2	16 8	18 8	270
160	397	33	16 2	16 8	18 8	270
140	397	33	16 2	16 8	18 8	270
120	397	33	16 2	16 8	18 8	260
100	397	33	16 2	16 8	18 8	271
80	397	33	16 2	16 8	18 8	271
60	397	33	16 2	16 8	18 8	281
40	397	33	16 2	16 8	18 8	291
20	472	20	13 6	6 4	14 4	250

Table 6 2 DV Results (L case)

For the low compatibility degree testbenches, all the distribution-graph based algorithms (i e , PTS-FDS, PTS-DV) have the same results over the PDC range Only the CPU varies slightly (see table 6 2) The PTS-FDS algorithm generated the following schedule characteristics for loose power constraints (PDC = 200) TL = 395, MPD = 33, APD = 16 2, PDD = 16 8, RMS = 18 8, CPU = 301 ms

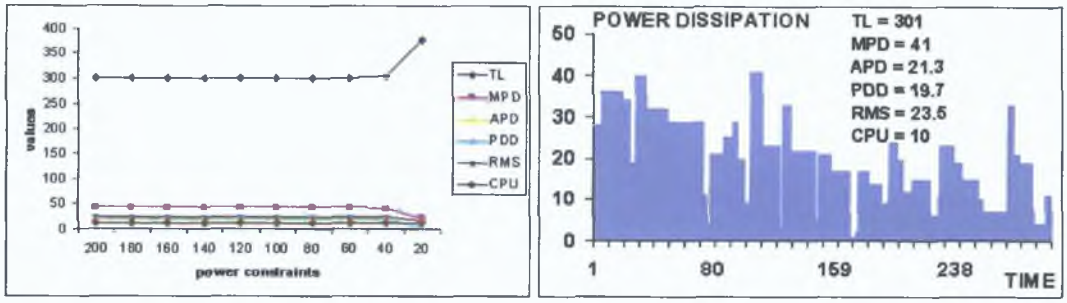
power cons	P1MRU FDS					
	TL	MPD	APD	PDD	RMS	CPU
200	401	33	16	17	18 5	411
180	401	33	16	17	18 5	393
160	401	33	16	17	18 5	398
140	401	33	16	17	18 5	402
120	401	33	16	17	18 5	405
100	401	33	16	17	18 5	389
80	401	33	16	17	18 5	407
60	401	33	16	17	18 5	412
40	401	33	16	17	18 5	409
20	483	20	13 3	6 7	15 6	410

Table 6 3 P1MRU-FDS Results (L case)

Again, for the low resource compatibility testbenches, the further application of the FDS (DV) algorithms on the schedules generated by the LS algorithms, i e the use of the LSFDS (LSDV) algorithms, does not help at all However, the typical LSFDS (LSDV) results are given above in table 6 3 for the sequence of P1MRU - FDS algorithms

Case AL Average-low (AL) resource compatibility case

As it was experienced in the previous example, the characteristics are almost the same for all list-scheduling algorithms This time though, comparing figures 6 3(b) and 6 5(b), it can be noticed that the power distribution in the latter is slightly less spiky than the former The results generated for this test set are very similar to those of the previous test set Firstly, the characteristics' curves over the



(a) Characteristics of P1MRU Results (b) Best Result (PDC = 200)

Figure 6.5: Typical Results Generated with P1MRU (A-L test set)

range of power dissipation constraints are almost constant. Secondly, the power-test characteristics' values again are slightly different for the P2MRU, P2INS, P2ORD algorithms comparing with the other list-scheduling based algorithms. However, the differences also remain within a 2-3% range.

power const.	P2RAND					
	TL	MPD	APD	PDD	RMS	CPU
200	303	46	21.2	24.8	22.8	10
180	320	40	20	20	21.7	10
160	309	46	20.8	25.2	22.7	10
140	320	43	20	23	21.8	10
120	323	43	19.9	23.1	21.6	10
100	320	46	20	26	21.9	10
80	303	48	21.2	26.8	23	10
60	322	40	19.9	20.1	21.8	10
40	320	35	20	15	21.6	10
20	394	20	16.3	3.7	16.7	10

Table 6.4: P2RAND Results (AV case)

The typical (apart from the P2 algorithms mentioned above) result is again the one given by P1MRU. For a power constraint of 200 the characteristics are: TL = 301, MPD = 41, APD = 21.3, PDD = 19.7, RMS = 23.5, CPU = 10 ms. As for the previous test set, the exceptions arise for the P2 algorithms. For example, the results given by P2MRU (P2INS, P2ORD) for a power constraint of 200 are: TL = 312, MPD = 40, APD = 20.6, PDD = 19.4, RMS = 22.1, CPU = 10 ms. An expected exception are the P2RAND results given in table 6.4.

Again, the results generated by the distribution-graph based algorithms exhibit more or less the same behaviour as the list-scheduling: the results' values are almost constant over a range of power dissipation constraints. For example, for a power constraint of 200 the characteristics are, for both FDS and DV algorithms: TL = 301, MPD = 41, APD = 21.3, PDD = 19.7, RMS = 23.1, CPU = 1522 ms.

With the LSFDS algorithm, the FDS algorithms do not improve the results/characteristics of the charts given by the LS algorithms, again because of the low degree of compatibility amongst test resources.

A difference worth mentioning is the CPU time used by the list-scheduling based algorithms and the distribution-graph based ones This CPU time difference increases with the increase of resource compatibility as can be seen in the following testbenches This is because higher resource compatibility corresponds to more relaxed constraints and, therefore, it takes a longer time to search through the bigger solution space

Case AV Average (AV) resource compatibility case

power const	P1MRU					
	TL	MPD	APD	PDD	RMS	CPU
200	204	79	31 4	47 6	37 4	10
180	204	79	31 4	47 6	37 4	10
160	204	79	31 4	47 6	37 4	20
140	204	79	31 4	47 6	37 4	20
120	204	79	31 4	47 6	37 4	10
100	204	79	31 4	47 6	37 4	20
80	204	79	31 4	47 6	37 4	10
60	209	58	30 7	27 3	35	10
40	242	40	36 5	13 5	28 5	10
20	371	20	17 3	2 7	17 6	10

Table 6 5 P1MRU Results (AV case)

For the first time the power-test characteristic curves are not flat This trend is going to be emphasized with the next two testbenches of this example Beginning roughly with this level of resource compatibility degree (average - 50%), the PTS algorithms start showing their effects for tight power constraints That is, starting with an average level of resource compatibility, the tight power constraints are a stronger factor than the compatibility degree that drives the PTS algorithm's solution space search Thus far the resource compatibility constraints seemed to limit the solution space more than the power constraints Moreover, the CPU time invested by these algorithms is still around 10 ms for list scheduling-like algorithms, but for the distribution-graph based algorithms the CPU time has increased from dozens of milliseconds to following values FDS = 16 s, DV = 2 5 s (for PDC = 200) The LSFDS algorithms take, for the same example, CPU times of up to 30 s

This time, the typical results generated by P1MRU are given in table 6 5 However, exceptions again are the P2 (e g , P2ORD) algorithms which exhibit as in the previous cases a longer (11% for loose PDC) TL, but lower valued power characteristics (less than 3 % for loose PDC)

The distribution-based algorithms still do not generate better results than the list-scheduling based algorithms The power dissipation constraint is still less significant to PTS algorithms' search engine than the low compatibility (high conflict) between the test resources The LSFDS algorithms do not improve the LS results, but rather they worsen the results at this stage

Case AH Average-high (AH) resource compatibility case

power const	P1MRU					
	TL	MPD	APD	PDD	RMS	CPU
200	154	94	41 6	52 4	51 7	20
180	154	94	41 6	52 4	51 7	20
160	154	94	41 6	52 4	51 7	20
140	154	94	41 6	52 4	51 7	20
120	154	94	41 6	52 4	51 7	20
100	154	94	41 6	52 4	51 7	20
80	156	41 1	79	37 9	48 3	30
60	159	60	40 3	19 7	44	30
40	193	40	33 2	6 8	34 7	30
20	348	20	18 4	1 6	18 6	30

Table 6 6 P3MRU Results (AH case)

power const	DV					
	TL	MPD	APD	PDD	RMS	CPU
200	130	85	49 3	35 7	55 1	3084
180	130	85	49 3	35 7	55 1	3085
160	130	85	49 3	35 7	55 1	3094
140	130	85	49 3	35 7	55 1	3094
120	130	85	49 3	35 7	55 1	3094
100	130	85	49 3	35 7	55 1	3104
80	136	80	47 2	32 8	53 1	2944
60	148	59	43 3	15 7	46 6	2954
40	187	40	34 3	5 7	35 3	3205
20	356	20	18	2	18 2	3485

Table 6 7 DV Results (AH case)

The first obvious thing is the CPU time increase from previous example (average compatibility degree) because the search space is increased with higher compatibility degree of test resource sets The results are the same for all five insertion types, for P1, P3 and list scheduling algorithms The typical results are given in table 6 6

Again, the second pseudocode algorithm generates slightly better results (8% shorter TL and 1% lower MPD) For example, the RAND insertion gives this time constantly the best results The distribution-based algorithms give for the first time constantly better results than the list-scheduling based ones The FDS and DV algorithms give almost the same results For example, table 6 7 gives the results generated by both FDS and DV algorithms for PDC = 200 Thus, the distribution-graph based algorithm improved for this example the TL characteristic by 16% (from 154 to 130) and the power characteristics by $\sim$ 10% (e g MPD by 9 57%, from 94 to 85)

As for the LSFDS type of algorithms, a comparison of the P2MRU and P2MRU-DV algorithms is given here, as an example, for a power constraint PDC of 200 The initial result given by P2MRU TL = 151, MPD = 93, APD = 42 5, PDD = 50 5, RMS = 46 9, CPU = 20 ms has been improved with the LSDV algorithm to TL = 151, MPD = 67, APD = 42 5, PDD = 24 5, RMS = 44 9, CPU = 31946 ms Thus, while the TL is kept constant, the results' power characteristics have been improved by 28% at the expense of a 32 seconds longer CPU time

Case H High (H) resource compatibility case

power const	P3MRU					
	TL	MPD	APD	PDD	RMS	CPU
200	99	197	64 8	132 2	90 9	20
180	99	180	64 8	115 2	87 7	30
160	99	160	64 8	95 2	84	20
140	99	139	64 8	74 2	80 6	20
120	100	120	64 1	55 9	76 7	20
100	108	100	59 4	40 6	69 3	30
80	109	80	58 8	21 2	64 6	20
60	139	60	46 1	13 9	49 7	30
40	183	40	35	5	36 4	30
20	345	20	18 6	1 4	18 8	30

Table 6 8 P3MRU Results (H case)

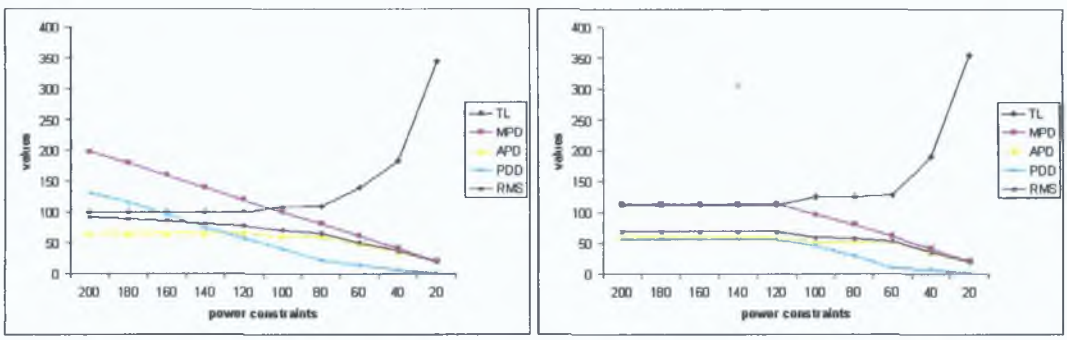
Next the PTS algorithms are run for a 50 randomly chosen BTS, where the degree of test resource compatibility is high (around 90%) Previous examples have shown that solution space's dimensions are directly proportional to the degree of resource compatibility between tests The higher the resource compatibility degree, the larger the solution space is expected to be Moreover, it is expected that the features of the PTS algorithms could be better distinguished for bigger solution spaces The most important thing to be emphasized here after assessing the results is that the PTS-LS and PTS-LEA algorithms could be classified into two groups The first one is comprised of P1, P3, and LS algorithms and the second one is made up by the P2 algorithms

Now, the P1, P3, and LS algorithms are firstly examined in order to analyze the characteristics of their results which are chosen from a big solution space The typical results generated by these algorithms for a high resource compatibility test set are given in table 6 8 The other algorithms' (apart from P2 s) results range as usual around the aforementioned values On the other hand, the P2 algorithms generate invariably different results for each test set and for each different insertion strategy For example, in table 6 9, the results of P2MRU and P2ORD algorithms show how different the characteristics of P2 algorithm's results for different insertion strategies are This behaviour has been noticed starting with the previous test set Namely, for the low, average-low and average resource compatibility testbenches,

power cons	P2MRU						P2ORD					
	TL	MPD	APD	PDD	RMS	CPU	TL	MPD	APD	PDD	RMS	CPU
200	111	113	57 8	55 2	66 8	30	85	120	75 4	44 6	81	30
180	111	113	57 8	55 2	66 8	30	85	120	75 4	44 6	81	30
160	111	113	57 8	55 2	66 8	30	85	120	75 4	44 6	81	30
140	111	113	57 8	55 2	66 8	31	85	120	75 4	44 6	81	30
120	111	113	57 8	55 2	66 8	30	85	120	75 4	44 6	81	20
100	125	95	51 5	43 7	58 2	30	94	92	68 2	23 8	72 5	30
80	125	79	51 3	27 7	56 7	30	121	80	53	27	58	30
60	127	60	50 5	9 5	52 8	30	134	60	47 9	12 1	49 6	30
40	189	40	33 9	6 1	34 8	30	190	40	33 8	6 2	34 7	30
20	356	20	18	2	18 2	20	365	20	17 6	2 4	17 7	20

Table 6 9 PTS Characteristics of P2 Algorithms' Results (H case)

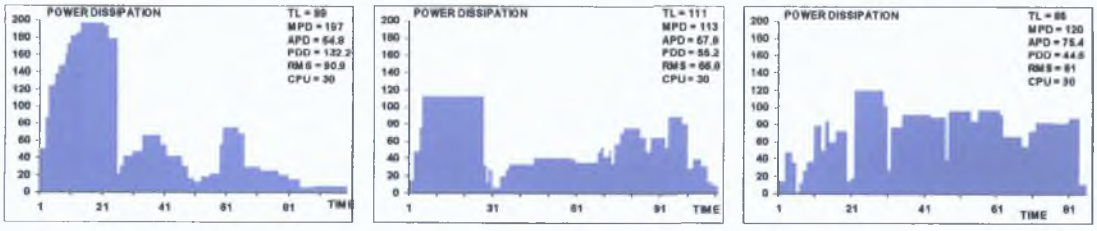
the P2 algorithms's results were exhibiting longer TL characteristics, but lower power characteristics. For the last two testbenches (average-high and high), P2 algorithms generate more balanced power distribution results. Some of their results have even the TL characteristics improved compared with the results given by the first set of list-scheduling based algorithms (i.e., P1, P3, and LS). In comparison to the typical P3MRU results from table 6.8, it can be noticed that the MRU insertion strategy of the P2 algorithm generates 10% longer TL characteristic results with a 43% improvement of the MPD characteristic for loose power constraints. For the ORD insertion strategy, the P2 algorithm generates a 14% shorter TL characteristic results improving this time by only 39% the MPD characteristic.



(a) P1MRU Results (b) P2MRU Results

Figure 6.6: PTS Characteristics' Curves Comparison (P1 and P2 Algs., H Case)

The curves of the PTS characteristics featured by the results generated with the list-scheduling based algorithms, over a range of power constraints from loose to tight, are depicted in figure 6.6. Comparing the two sets of curves from figure 6.6(a) and figure 6.6(b), there is one interesting aspect. Namely, while the APD and RMS curves have more or less the same trends in both graphs, it is not the case of the MPD and PDD curves. In figure 6.6(a) the MPD and PDD curves are rather strictly decreasing for a more stringent power constraints. On the other hand, in figure 6.6(b) these curves have a strictly decreasing trend only for the second part (i.e., for tighter power constraints), whereas in the first part they are flat. The explanation of this behaviour is that the P2 (for example, P2MRU in figure 6.6(b)) algorithm, with its several insertion strategies, generates for loose power constraints results that exhibit more balanced power distribution charts. Therefore, the P2 algorithm is a list-scheduling based algorithm that behaves in alike manner to the distribution-based algorithms. This is the reason why P2 was exhibiting



(a) P1MRU chart (b) P2MRU chart (c) P2ORD chart

Figure 6.7: PTS Charts of LS-based Algorithms (PDC = 200, H case)

throughout this first experiment slightly better power characteristic results than the other list-scheduling based algorithms. Unfortunately, the more balanced power characteristics of P2 algorithm's results did not preserve for all previous testbenches the short TL characteristics exhibited by the results of the other list-scheduling based algorithms.

In order to further emphasize the difference between the P2 algorithm and other list-scheduling based algorithms, a comparison of PTS chart examples is given in figures 6.7 and 6.8. Figure 6.7 depicts the PTS charts generated by P1MRU, P2MRU and P2ORD algorithms for loose power constraints (PDC = 200). Figure 6.8 depicts the PTS charts generated by P1MRU, P2MRU and P2ORD algorithms for tighter power constraints (PDC = 140). It can be noticed that figures 6.7(b) and 6.7(c) are, respectively, identical to figures 6.8(b) and 6.8(c). This is explained by the fact that P2 algorithms generate more balanced power distribution charts than the other list-scheduling based algorithms. This observation is reinforced in table 6.9, which proves that the power-test characteristics of the P2 algorithms' results are invariantly identical for power constraints ranging from loose (PDC = 200) to average (PDC = 120). Therefore, it can be stated that the power constraint is not the most important factor which drives P2 algorithm's search engine. This is not

(a) P1MRU chart (b) P2MRU chart (c) P2ORD chart

Figure 6.8: PTS Charts of LS-based Algorithms (PDC = 140, H case)

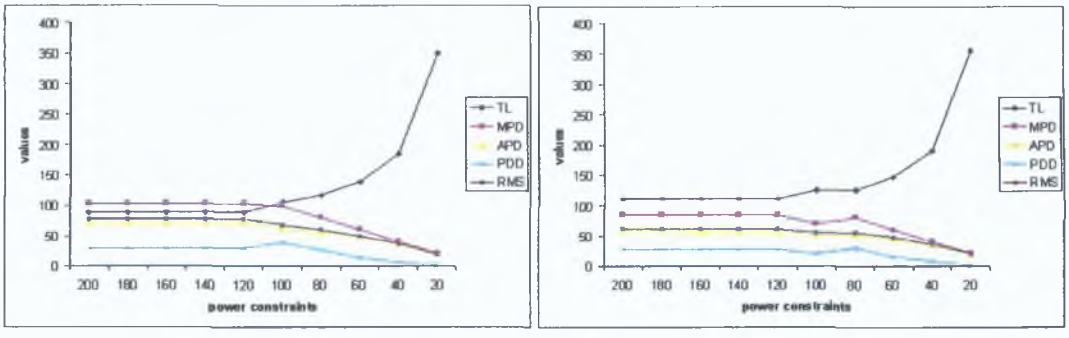
the case for the other list-scheduling based algorithms, where the power constraint is the most important driving factor. In figure 6.7(a), the power distribution of the results generated by the P1MRU algorithm is remotely unbalanced because of the loose power constraint. Figure 6.8(a) shows how the power constraint limits the maximum power (MPD) characteristic of the chart, bringing it close to the average power (APD) characteristic.

power cons.	FDS						DV					
	TL	MPD	APD	PDD	RMS	CPU	TL	MPD	APD	PDD	RMS	CPU
200	88	103	72.9	30.1	77.4	5858	95	102	67.5	34.5	73.3	4326
180	88	103	72.9	30.1	77.4	5798	95	102	67.5	34.5	73.3	4366
160	88	103	72.9	30.1	77.4	5808	95	102	67.5	34.5	73.3	4326
140	88	103	72.9	30.1	77.4	5818	95	102	67.5	34.5	73.3	4356
120	88	103	72.9	30.1	77.4	5809	95	102	67.5	34.5	73.3	4347
100	105	99	61.1	37.9	67.5	4978	108	97	59.4	37.6	67.1	4096
80	117	80	54.8	25.2	*60*	*3896*	*114*	78	56.3	21.7	59.6	4186
60	138	60	46.5	13.5	49.2	22973	138	60	46.5	13.5	49.2	3956
40	185	40	34.7	5.3	35.5	7381	185	40	34.7	5.3	35.4	4376
20	351	20	18.3	1.7	18.4	14601	349	20	18.4	1.6	18.5	4346

Table 6.10: PTS Characteristics' Comparison (Distribution-based Algs., H case)

Amongst the distribution-based PTS algorithms (i.e., FDS and DV) the best results are given by the FDS algorithm (as can be seen in the time-power characteristics in table 6.10), but at an expense of computation time (see CPU time characteristic in the same table). The shape of their result characteristic curves drawn over a range of power dissipation constraints are at first sight somewhat similar to the ones generated with the PTS-LS and PTS-LEA algorithms. Only the characteristic values seem to be different (see figure 6.9). A closer comparison of all the results generated by the PTS algorithms is given in the next conclusive subsection.

Table 6.11 proves that the LSFDS (e.g., P3MRU-FDS, P1MRU-FDS) algorithms manage to improve the power characteristics of the initial LS (e.g., P3MRU) results.



(a) FDS Results (b) P2MRU-FDS Results

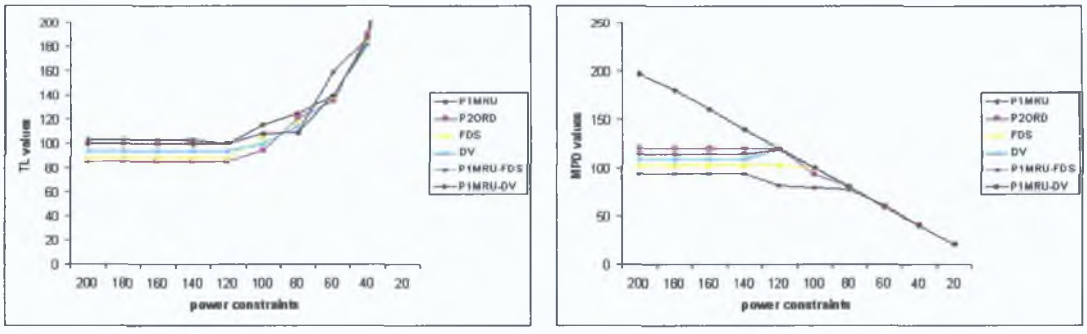
Figure 6.9: PTS Characteristics' Curves Comparison (FDS and LSFDS Algs.)

power cons.	P3MRU-FDS						P3MRU-DV					
	TL	MPD	APD	PDD	RMS	CPU	TL	MPD	APD	PDD	RMS	CPU
200	103	93	62.3	30.7	67.1	15913	99	114	64.8	49.2	73.1	5668
180	103	93	62.3	30.7	67.1	15563	99	114	64.8	49.2	73.1	5563
160	103	93	62.3	30.7	67.1	15642	99	114	64.8	49.2	73.1	5738
140	103	93	62.3	30.7	67.1	15553	99	114	64.8	49.2	73.1	5798
120	100	81	64.1	16.9	66.5	17215	100	119	64.1	54.9	73.8	5127
100	108	79	59.4	19.6	62.4	16264	115	100	55.8	44.2	64.7	5187
80	109	77	58.8	18.2	61.3	16264	124	80	51.7	28.3	58.9	5628
60	159	60	40.3	19.7	43.2	17825	139	59	46.1	12.9	48.9	5158
40	187	40	34.3	5.7	35.4	15257	187	40	34.3	5.7	35.7	4577
20	345	20	18.6	1.4	18.8	16874	345	20	18.6	1.4	18.8	3064

Table 6.11: PTS Characteristics' Comparison (LSFDS Algorithms, H case)

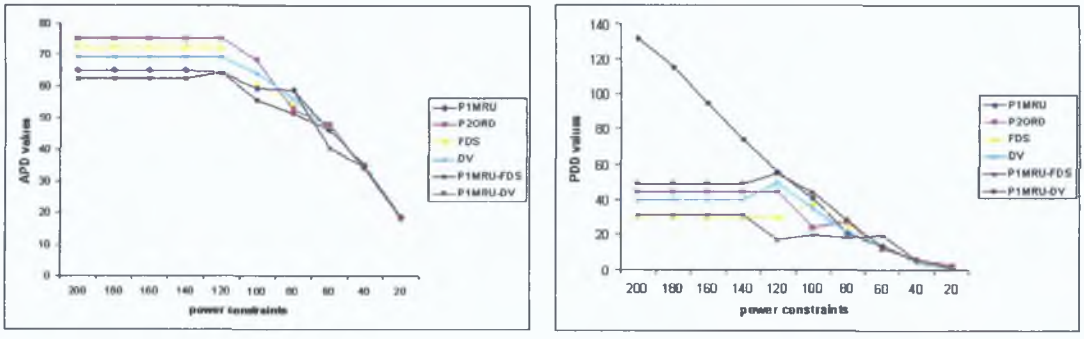
For loose power dissipation constraints, the P3MRU-FDS (or P1MRU-FDS) algorithms balance in most of cases the power dissipation characteristics keeping the TL characteristic unchanged (i.e., no increase). However, there are cases when, even though the TL time happens to get slightly increased, it pays off to have a much better power characteristic as it is the case of P3MRU-FDS results in table 6.11. This point is arguable though in cases when the power constraint is loose and MPD is below the constraint because a decreased PDD might not offer a real advantage or improvement. In this event the solution which gives the lowest TL might be preferable regardless of other power dissipation balance improvement. It is up to the user to decide over this TL/PDD trade-off.

The P3MRU-FDS algorithm does not improve the TL characteristic of the initial P3MRU solution due to the quality of the set of roots taken over by the FDS algorithm from the solution given by the P3MRU. This is a drawback of LSFDS algorithms because there are even cases (rare though)when the FDS solution already exhibits better TL values than the initial LS solutions. In these cases it is a waste of time to generate the initial solution with the LS algorithm. On the other hand the P3MRU-DV algorithm improves the power dissipation characteristics without increasing the TL.



(a) TL Curves (b) MPD Curves

Figure 6.10: PTS Characteristics' Curves Comparison (H case) I

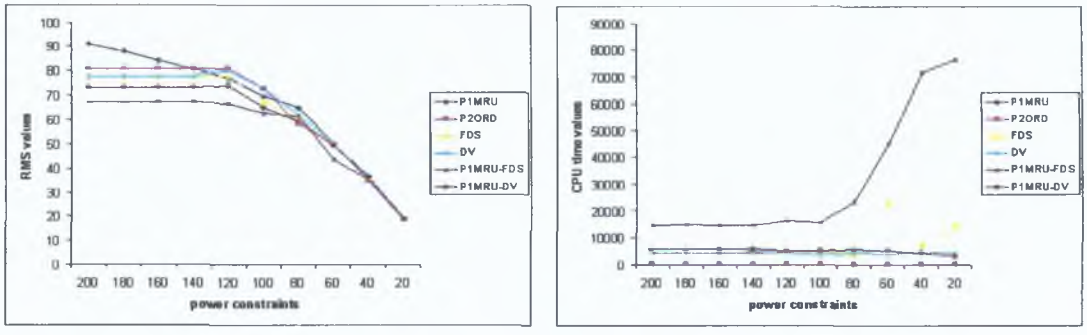


(a) APD Curves (b) PDD Curves

Figure 6.11: PTS Characteristics' Curves Comparison (H case) II

Conclusions

The shapes of the PTS characteristics' curves over the range of power constraints are somewhat similar for all the PTS algorithms, as could be seen in figures 6.6 and 6.9. Figures 6.10, 6.11, 6.12 further emphasize these similarities by depicting and comparing the PTS characteristics' curves for all of the PTS algorithms proposed in this thesis. If the power constraints are reasonably loose, with the increase of test resource compatibility degree, more tests can be accommodated in parallel. For the first experiment, these two factors proved to be the key constraining elements. Therefore, from low (e.g., see figure 6.3(a)) to high (e.g., see figure 6.6(a)) degree resource compatibility, the total test application time TL values decrease (see figure 6.42(a)). And for high degree resource compatibility in figure 6.10(a), the total test application time TL increases at the same time with the increase of (e.g., tightening) power constraints. A notable difference between the PTS algorithms' results are the



(a) RMS Curves (b) CPU time Curves

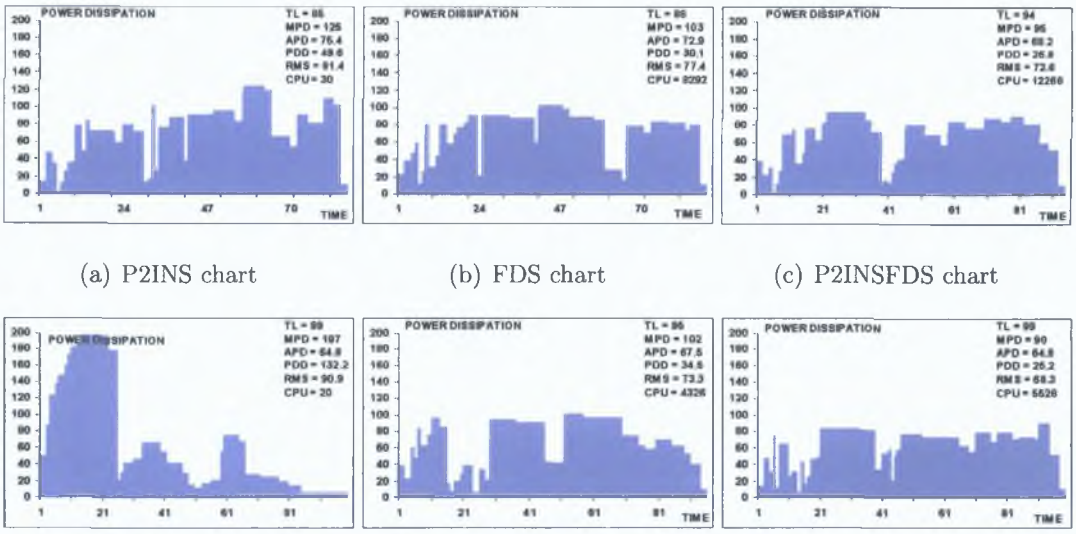
Figure 6.12: PTS Characteristics' Curves Comparison (H case) III

curves of the MPD and PDD characteristics (see figures 6 10(b) and 6 11(b)) of the P2 and all distribution-graph based algorithms, on the one hand, and the rest of list-scheduling based algorithms on the other hand Namely, in figures 6 10(b) and 6 11(b) the curves of the MPD and PDD characteristics of the P1MRU algorithm are overall strictly decreasing throughout the power constraint's increase, which is not the case for the curves of the MPD and PDD characteristics generated by the other kinds of PTS algorithms for loose power constraints

The APD curves are depicted in figure 6 11(a) They are almost constant for loose power constraints This is due to the fact that the TL curves are almost constant for loose power constraints and APD is the ratio between the area taken by the PTS charts and their total test time TL (see subsection 3 3 5) The TL characteristics generated by the list-scheduling based algorithms are almost constant for loose power constraints even though the MPD or PDD characteristics are not This is due to the fact that for loose power constraints the list-scheduling based algorithms (apart from the P2 algorithm) generate unbalanced solutions (see figure 6 7(a)) That is, the power constraint being the only factor that can be used to get more balanced power distribution results with the same total application time as it is the case of the charts generated by P1MRU in figures 6 7(a) and 6 8(a) In these figures, for the same total application time, a better power distribution solution was obtained by running the P1MRU algorithm with a tighter power constraint (PDC = 140) This was in contrast with the other kind of PTS algorithms (i e , P2 and distribution-graph based algorithms) which gave balanced power distribution charts without imposing a tighter power constraint

Compared with the aspects discussed in the previous paragraph, the RMS characteristics' curves in figure 6 12(a) for the P1MRU algorithm are somewhere in between the MPD characteristic curve and the APD one This is explained by the fact that the RMS values of the PTS results should be between the MPD value and the APD one This is because they are independently calculated using formula 3 1 in subsection 3 3 1 giving a value that is proportional to the quantity of spikes in the power distribution The less spikes, the closer the RMS value to the APD one Otherwise, the RMS value increases towards the MPD one

For the first experiment the CPU time of the PTS algorithms range from the L case to the H case within the following values all the list-scheduling based algorithms (including the P2 algorithms) take from less than 10 ms (L case) to less than



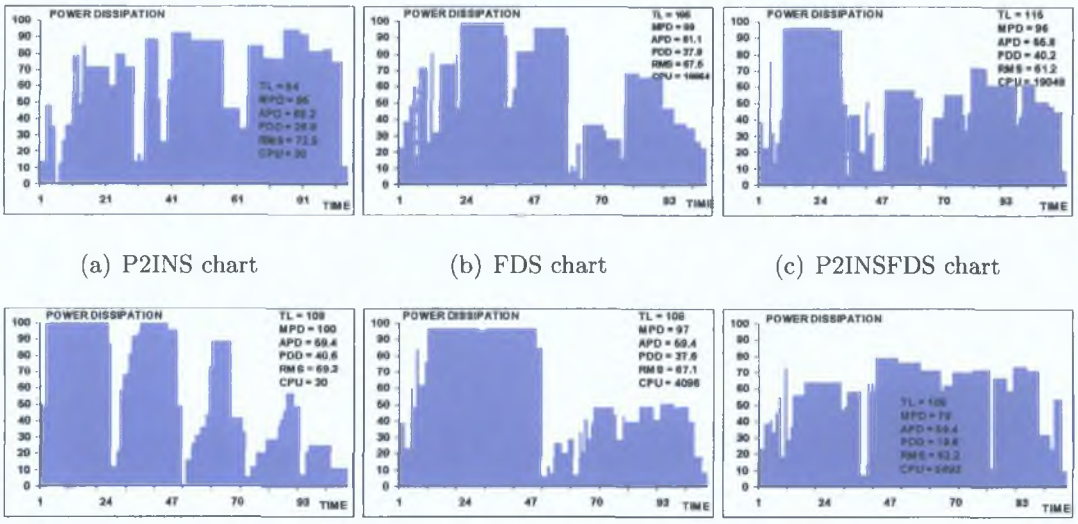
(a) P2INS chart (b) FDS chart (c) P2INSFDS chart

(d) P3MRU chart (e) DV chart (f) P3MRUDV chart

Figure 6.13: PTS Charts (PDC = 200, H case)

35 ms (H case); the FDS algorithms' CPU time values range between 600 ms (L case) to 6 - 20 s (H case); the DV algorithms take CPU time range between 400 ms (L case) to 4 - 5 s (H case); the LSFDS algorithms take between 300-400 ms (L case) to 15 - 40 s (H case); the LSDV algorithms take between 200-300 ms (L case) to 5 - 6 s (H case). The CPU time for the H case over a range of power constraints from loose to tight is depicted in figure 6.12(b). The FDS-based algorithms are the only exception from the constant curves exhibited by all the other PTS algorithms. This exception could be explained by the higher computational complexity of the *force* priority function. Intuitively, it was expected that the total CPU time required by a LSFDS or LSDV algorithm would be the sum of the time spent by the LS algorithm and the time spent to run the FDS algorithm, plus the reset (initialization) stage between the algorithms. In reality, the CPU time of the LSFDS (LSDV) algorithms is longer than the expected accumulated CPU time, especially for the LSFDS ones. This is due to the fact that the solution search space of the LS and FDS algorithms is smaller than the one the LSFDS deals with. The computation complexity of the PTS algorithms is given by the number of tests and the number of gaps at each step. The LS and FDS algorithms start from scratch without any gap in the list, and at each step it deals only with a certain number of gaps, which proves to be smaller than the number of gaps corresponding to the number of roots of the growing tree generated by the LS algorithm. This increases the computation time of the FDS

(DV) algorithm and, therefore, the computation time of the overall LSFDS (LSDV) algorithm as well.



(a) P2INS chart (b) FDS chart (c) P2INSFDS chart

(d) P3MRU chart (e) DV chart (f) P3MRUDV chart

Figure 6.14: PTS Charts (PDC = 100, H case)

One of the expected conclusions is that the distribution-graph based algorithms generate, for reasonably loose power constraints, PTS profiles that exhibit more balanced power dissipation distributions as shown by the examples in figures 6.13 and 6.14. That is, the solutions given by distribution-graph based algorithms have smaller PDD and MPD values, but they are more computationally expensive than the list-scheduling based algorithms. Moreover, while the FDS algorithm is more computationally expensive than the DV one, the solutions given by the former are generally more balanced. At the same time, all PTS algorithms exhibit almost the same scheduling solutions for tight power dissipation limits or low resource compatibility degrees, when the solution space is actually narrowed.

To conclude this experiment and recapitulate the features of the results given by the PTS algorithms, let's compare the PTS charts in figures 6.13 and 6.14. The first set of PTS solutions (figure 6.13) are generated without power constraints, whereas the second set (figure 6.14) are generated for an average-tight power constraint. It can be seen in figure 6.13 that the charts of LSFDS (figure 6.13(c)) and LSDV (figure 6.13(f)) algorithms are better in terms of PTS characteristics than the results given by all the list-scheduling based algorithms (see figure 6.13(d)). Only the P2 algorithms (see figure 6.13(a)) make an exception from the above stated rule. It can

be noticed that there is also a significant difference, in terms of the PTS characteristics' quality, between the list-scheduling based algorithms (apart from P2) and the distribution-graph based algorithms (figures 6 13(b) and 6 13(e)) However, the differences between the distribution-graph based algorithms (i e , FDS and DV) and the combined algorithms LSFDS (LSDV) are not significant, even though a small improvement is noticed with the combined algorithms But the differences between the list-scheduling based algorithms' (apart from P2) results and the LSFDS (LSDV) ones are significant This qualitative differences decrease with tighter power constraints as can be seen in figure 6 14 For tighter power constraints, the LSFDS (LSDV) algorithms struggle to both balance the power dissipation distribution of the initial PTS charts given by the LS algorithms, and to comply with the power dissipation constraints, also having this time a time constraint The time constraint is given by the total application time of the tests in the roots of the resulting growing tree generated by the LS algorithm When the power dissipation constraint becomes tight, the FDS step of the LSFDS algorithm fails to comply anymore with the additional test time constraint given by the solution of the LS step This could happen, but rarely, even for loose power dissipation constraints For example, compare the results generated by the P3MRU and P3MRU-FDS algorithms from tables 6 8 and 6 11, respectively In conclusion, the LSFDS algorithms are efficient only for loose power dissipation constraints and high degrees of test resource compatibility Therefore, from now on, the LSFDS algorithms are discussed under loose power constraints, which is the context of their applicability

Another set of PTS algorithms, which were proposed in subsection 4 2 1, are the PTS-LS^2 algorithms However, they proved to generate worse results even than the ones obtained by running the initial PTS-LS algorithms This is explained by the inability of the local priority functions employed in this kind of algorithms to render optimal solutions Therefore, these algorithms will not be under focus in this chapter at all

Since the characteristic differences between the PTS results decrease with the tightening of power dissipation constraints, for tight power constraints it is advisable to employ the list-scheduling based algorithms rather than the distribution-graph based ones, because otherwise the CPU time difference between the two types of algorithms is unjustified

6 2.2 Second Experiment

- tests' resource compatibility degree is ranged from low to high with a finer granularity than in the first experiment,
- tests' power and test length values are randomly generated within a wide range (different test length and power values)

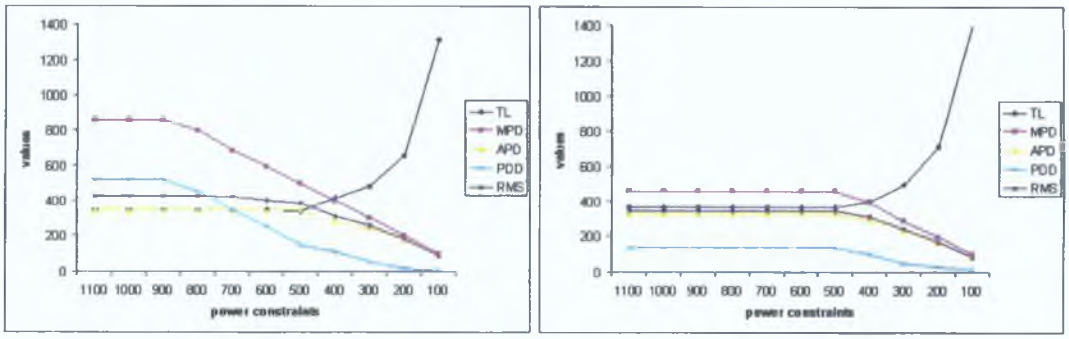
The second experiment is similar to the first one The BTS is different and the test resource compatibility degree is ranged this time from 10% to 90%, with a higher resolution, that is, with an increment step of 10% Thus, the experiments are run in the second example on 9 testbenches The results are similar There are only a few notable isolated exceptions from the results obtained in the first example and they will be discussed next

For a 20% compatibility degree between the test resources the LSFDS algorithm succeeds in improving the power characteristics for the P2 results by 24%, which was not the case in the first example On the other hand, for a compatibility degree of 30%, the P2 algorithm gives results for the first time visibly worse than the other LS algorithms(8% on the TL characteristic and an average 13% on the power characteristics) For the same case the FDS algorithms have more or less the same results as the LS ones For the 40% compatibility degree case it was noticed that the LSFDS algorithms start improving the power characteristics at the same time with a lengthening of TL The FDS algorithms also start generating better results than the LS ones

For the 50% compatibility degree case, the FDS algorithms manage to generate results exhibiting an average 35% better power dissipation characteristics than the LS ones, while the TL characteristic ranges within a 4% increase and a 1% decrease margins The same features are noticed with the LSFDS results

For the 60% compatibility degree case, another exception of the P2 algorithms is noticed in comparison with the LS ones Here the TL characteristic is increased by 10% in order to get an improvement of 27% with the power characteristics No major changes of the results generated by the FDS and LSFDS compared with the ones given by the respective algorithms in the previous (50% compatibility degree) case are present

For the 70% compatibility degree case, the exception of the P2 algorithms mentioned in the previous paragraph is repeated, but this time TL is increased up to



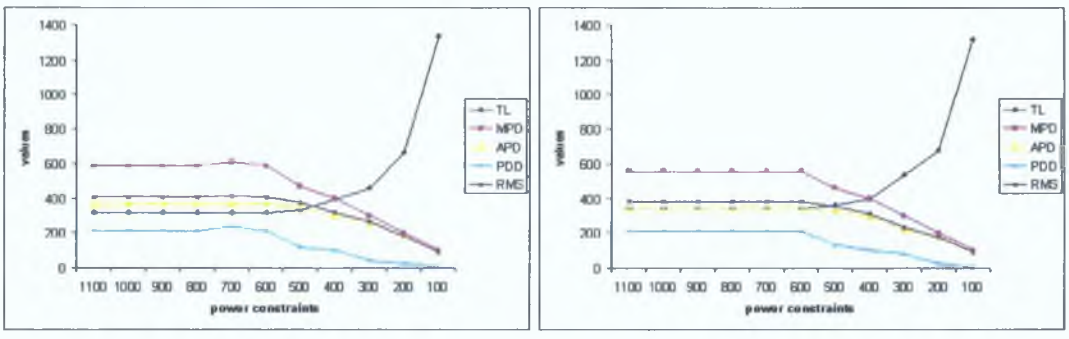
(a) P1MRU Results (b) P2MRU Results

Figure 6.15: PTS Characteristics' Curves (Second Experiment, 90% case) I

10%, while the power characteristics are improved by 46%. The FDS and LSFDS algorithms do not overrule the trend of the previous cases.

For an 80% test resource compatibility degree, it has been noticed that the FDS and LSFDS algorithms improve the characteristics as expected, but some of them happen to increase very slightly the test length (TL).

For the case of 90% test resource compatibility degree the same conclusions can be drawn as for the previous experiment (i.e., the first experiment). Figure 6.15 depicts the characteristics of the results given by the P1 algorithm, as an exponent of the list-scheduling based algorithms, and the P2 algorithm. The P2 algorithm generates solutions similar to the distribution-graph based algorithms even though it belongs to the list-scheduling based algorithms' category. Figure 6.16 depicts the characteristics of the results given by the FDS and DV algorithms, as exponents of the distribution-graph based algorithms. The only notable differences between the above mentioned figures and the ones depicted for the H case of the previous



(a) FDS Results (b) DV Results

Figure 6.16: PTS Characteristics' Curves (Second Experiment, 90% case) II

experiment (i.e., the first one) are the MPD and PDD characteristics. This is due to the fact that the PDD values are the arithmetic difference between the MPD values and the APD values. And the APD characteristics have usually the same shape and similar values (see figures 6.11(a), 6.21(a) and 6.21(b)) for all the results generated by different PTS algorithms for the same experimental case. The RMS characteristics are most of the time similar and slightly higher than the APD characteristics (compare figures 6.12(a), 6.22(a) and 6.22(b), respectively, with figures 6.11(a), 6.21(a) and 6.21(b)). Thus, the PDD characteristics are considered to reflect in the usual cases the behaviour of the MPD characteristics (compare figures 6.11(b), 6.20(a) and 6.20(b), respectively, with figures 6.10(b), 6.19(a) and 6.19(b)). Therefore, from now on, unless exceptions are encountered in the PDD characteristics, only the MPD characteristics will be employed to discuss the differences between the results generated by different PTS algorithms.

Comparing the curve of the MPD characteristics generated by P1MRU in figure 6.15(a) with the one generated by the same algorithm for the H case in the previous experiment (figure 6.10(b)), it can be noticed that the former starts with a flat region. This is due to the fact that above a power constraint value of PDC = 950 (see figure 6.15(a)) it can be considered that there are no power dissipation limits (for the 90% test resource compatible BTS example given here).

Conclusions

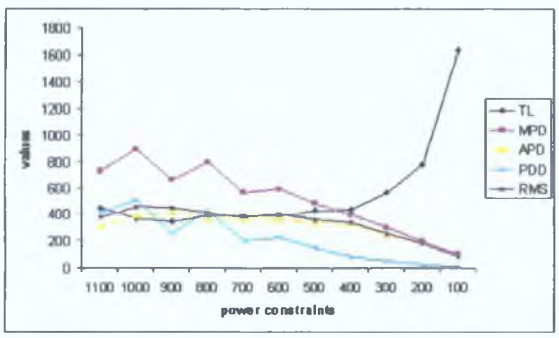


Figure 6.17: P2RAND Results (Second Exp.)

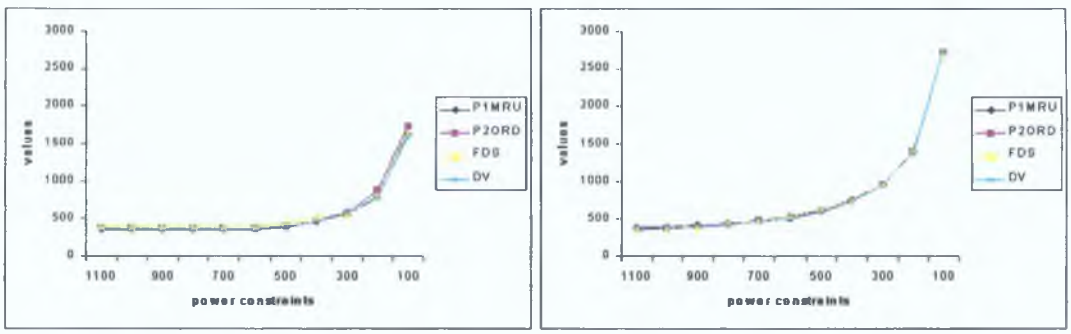
Overall, it can be stated here that the results of the previous experiment and the current one lead to the same conclusions, as their results are similar. However, it should be emphasized here that the P2RAND algorithm constantly maintains its already known behaviour of generating from the worst to the best results throughout the experimental cases (see their results' characteristics in figure 6.17). This way it manages quite often to reach the near-optimal

solutions by avoiding local minima. This fact supports the idea of employing near-optimal algorithms to the PTS problem. These algorithms are known to be able to jump out from the local minima during the search and continue to seek for the near-optimal solutions. The difference between the P2RAND algorithm and a near-optimal one is the lack of an intelligent mechanism to steer the search process. An intelligent mechanism would have the feature of weighing the best solutions at each iteration in a global manner (global priority function).

6.2.3 Third Experiment

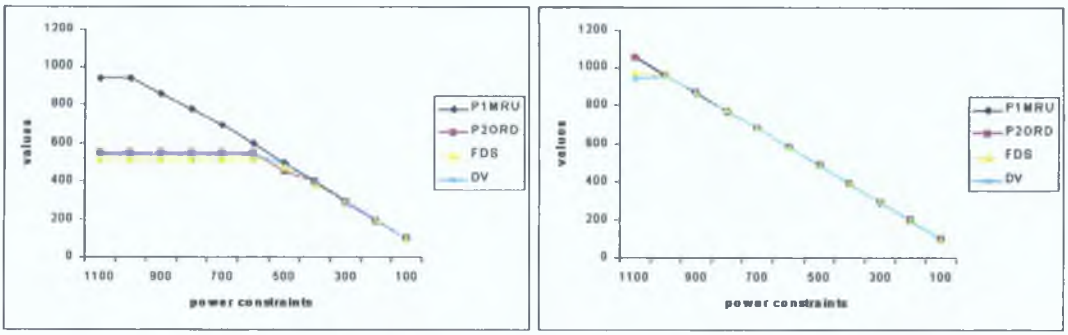
- tests' power dissipation parameters are ranged from similar to different values;
- tests' resource compatibility degree and test length values are randomly generated within a wide range (i.e., different values).

In this experiment the BTSs are generated by ranging their tests' power dissipation values (see section 3.1 in chapter 3) from very similar values to very different ones. Figures 6.18, 6.19, 6.20, 6.21, and 6.22 depict the curves of the characteristics (TL, MPD, PDD, APD, and RMS) generated by the selected algorithms (P1MRU, P2ORD, FDS, DV). However, only the extreme cases are compared because the intermediate ones exhibit results within the metamorphoses of the characteristics' curves between these two extremes. The curves in figures 6.18(a), 6.19(a), 6.20(a), 6.21(a), and 6.22(a) are given for the case where the BTS consists of tests with a high resource compatibility degree, very different TL and very different PD values. On the other extreme, figures 6.18(b), 6.19(b), 6.20(b), 6.21(b), and 6.22(b) are the characteristics generated by the same algorithms as above for a BTS with different



(a) Different Power Values (b) Similar Power Values

Figure 6.18: TL Characteristics' Curves Comparison (Third Experiment)

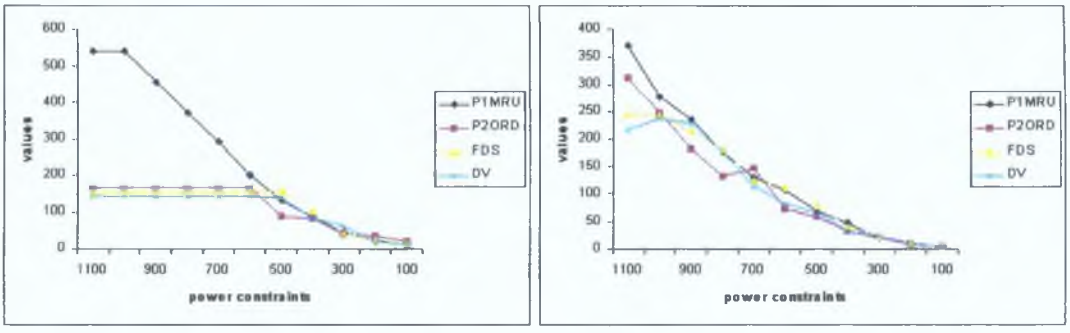


(a) Different Power Values (b) Similar Power Values

Figure 6.19: MPD Characteristics' Curves Comparison (Third Experiment)

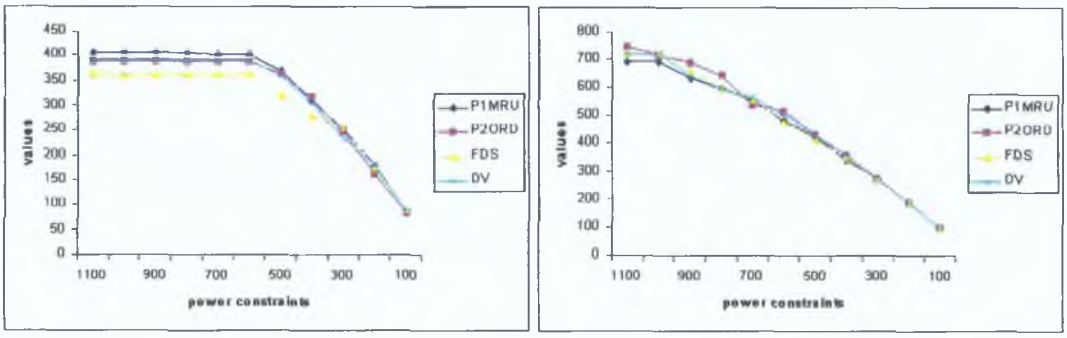
TL and similar PD values. That is, the algorithms were run in the second set of figures for the case where the BTSs have tests with high compatibility degree, and totally different TL and PD values.

In general, these curves demonstrate the previous statement that the TL and MPD curves characterize almost entirely the results. That is, the curves of the TL and MPD characteristics can be considered representative for the whole set of characteristics (TL, MPD, APD, PDD, RMS). For example, the curve of the MPD characteristic resembles the PDD one (see figures 6.19(a) and 6.20(a)), while the curve of APD and RMS are similar as well (see figures 6.21 and 6.22). Moreover, the APD (figure 6.21) and RMS (figure 6.22(a)) curves have the same kind of shape for all PTS algorithms run on the same BTS, as it is also the case of the TL curves in figure 6.18. Only the MPD characteristics' curves in figure 6.19(b) and the PDD characteristics' curves in figure 6.20(b) make up a slightly separate case. This emphasizes again the particular case of having similar power value BTS. Namely, for



(a) Different Power Values (b) Similar Power Values

Figure 6.20: PDD Characteristics' Curves Comparison (Third Experiment)

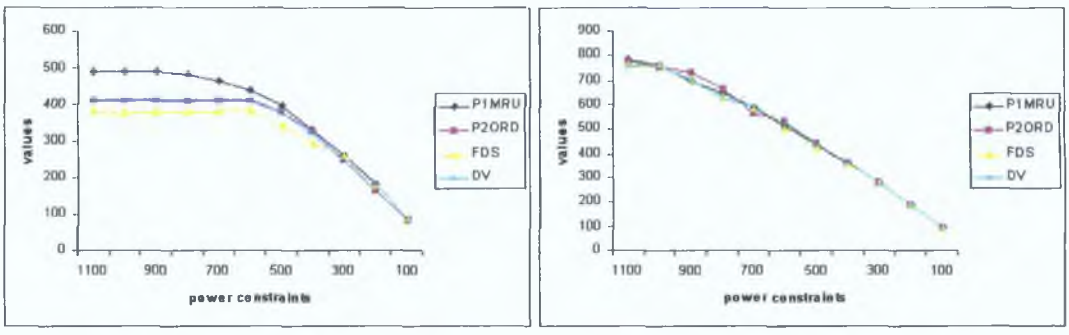


(a) Different Power Values (b) Similar Power Values

Figure 6.21: APD Characteristics' Curves Comparison (Third Experiment)

the second case presented in this experiment (similar power value tests) the curves of characteristics generated by all types of PTS algorithms are the same. Therefore, the general conclusion that can be drawn in this experiment is that for a set of tests (BTS) with similar power values the (fast but greedy) list-scheduling based PTS algorithms are a sufficient approach.

By looking at the aforementioned figures it can be noticed that the PTS algorithms generate almost the same results under any power constraint for BTS cases where tests have similar power values. For example, in figure 6.19(b) the MPD curves are almost identical for any PTS algorithm, which is not the case in figure 6.19(a). This is further emphasized by the P2RAND algorithm where the zig-zag curves in figure 6.17 (for the case of tests with different power values) become linear in figure 6.23 (for the case of tests with similar power dissipation values). Moreover, the power characteristics become linear for all types of algorithms (see figures 6.19(b), 6.21(b), and 6.22(b)).



(a) Different Power Values (b) Similar Power Values

Figure 6.22: RMS Characteristics' Curves Comparison (Third Experiment)

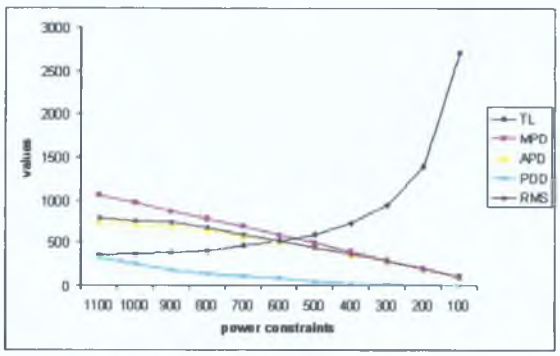
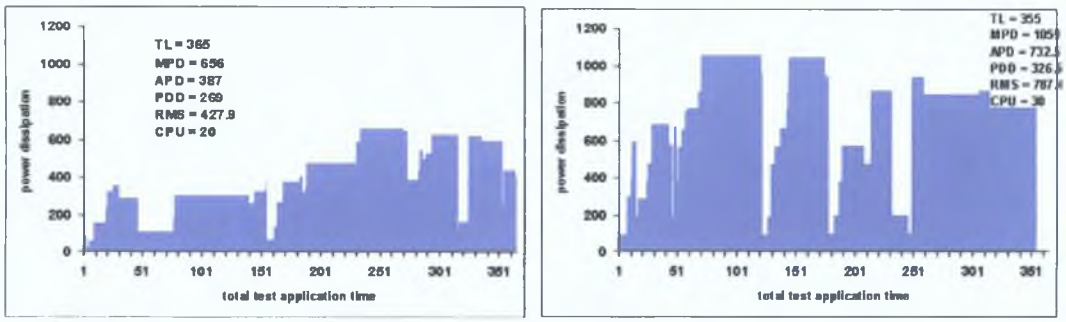


Figure 6.23: P2RAND Results (Third Exp.)

The PDD characteristics' curves (figure 6.20(b)) are different because it is zoomed closer than the other characteristics (see power axe scale values). Only for very loose power constraints do the distribution-graph based algorithms generate slightly different (better) results than all (even the P2 ones) list-scheduling based algorithms. Compared with the results generated by the list-scheduling based algorithms with loose power constraints, the power dissipation characteristics of the results generated by the FDS algorithm are up to 9% better, and the TL characteristics are also improved (decreased) by 4%. Under the same conditions (very loose power constraints), the DV algorithm achieves a 9% improvement of MPD at the same time with a 4% improvement of TL.



(a) Different Power Dissipation Values (b) Similar Power Dissipation Values

Figure 6.24: PTS Charts' Comparison (Third Experiment)

The charts from figures 6.24(a) and 6.24(b) are generated by the P2MRU algorithms with loose power constraints. It can be seen that the total test application time for loose power constraints does not change much. Actually, the total test application time of all the results generated for the third experiment vary within a $\pm 5\%$ limit. Exceptions are the results generated for very tight power constraints, where the power constraint value is almost equal to the tests' power dissipation parameter. It is an expected result because in this case the schedule is a pure sequential application of all the tests in the BTS. This is due to the fact that no tests

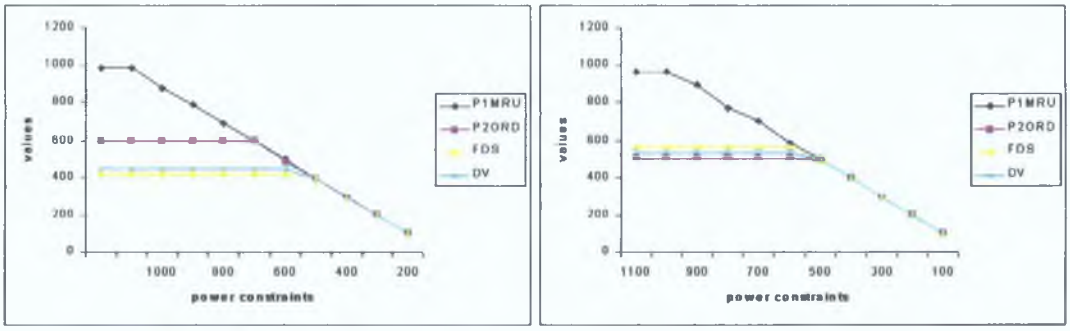
can be run in parallel under the given power limit which is almost equal to the power dissipation values of all the tests in the BTS.

An important conclusion can be drawn by looking at figures 6.18(b) and 6.19(b). Namely, the characteristics of the results generated by all PTS algorithms are almost the same when tests in the BTS exhibit a high resource compatibility degree, different TL values, and similar PD values. In these cases the application of list-scheduling based algorithms is sufficient to achieve good results.

The LSFDS-like algorithms preserve their features drawn in the previous experiments. For example, for the first case, when the BTS has tests with different power values, the LSFDS algorithms improve the MPD characteristic by 33%, while keeping the TL characteristic constant. The LSDV algorithms improve by up to 53% the MPD characteristic, by deteriorating the TL characteristic with 16%. For the second case, when the BTS has tests with similar power values, the figures are the same for both kinds (i.e., LSFDS and LSDV) of algorithms. Namely, they improve by 10% the MPD characteristic, while keeping the TL constant. The fact that the results are similar proves again that the solution searching process was slightly constrained by the similarity between the power values of BTS's tests.

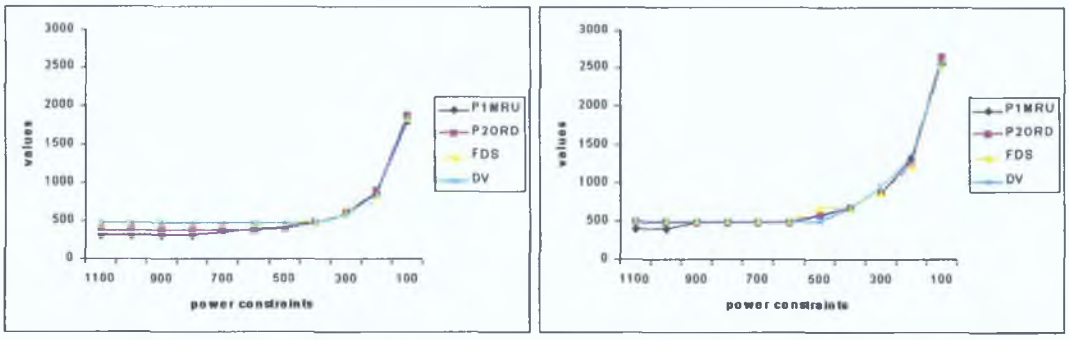
6.2.4 Fourth Experiment

- tests' application time (test length) parameters exhibit values within a wide range, from similar to different values;
- tests' resource compatibility degree and power values are randomly generated within a wide range (i.e., different values).



(a) Different Test Length Values (b) Similar Test Length Values

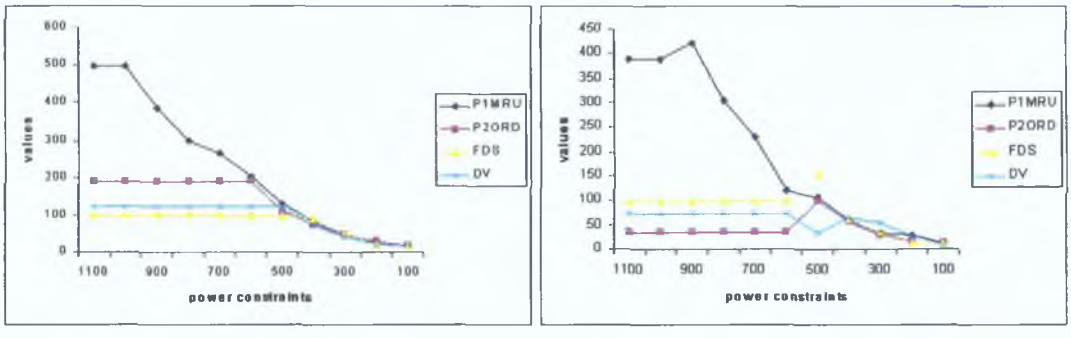
Figure 6.25: MPD Characteristics' Curves Comparison (Fourth Experiment)



(a) Different Test Length Values (b) Similar Test Length Values

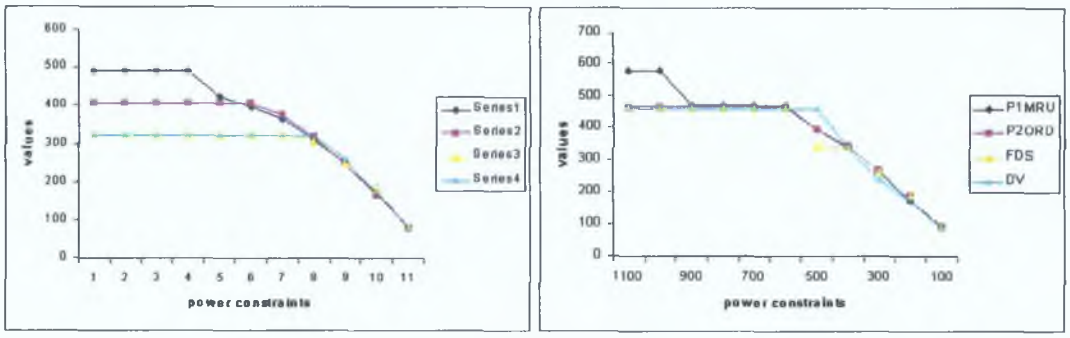
Figure 6.26: TL Characteristics' Curves Comparison (Fourth Experiment)

The fourth experiment is meant to bring under focus the behaviour of the results generated by the PTS algorithms for BTS cases where, this time, the tests' TL values vary from different to similar. Figures 6.25, 6.26, 6.27, and 6.28 draw, respectively, the MPD, TL, PDD, and APD characteristics' curves for the same PTS algorithms employed in the previous experiment. The RMS characteristics' curves are not depicted here because they have similar shape as the APD ones. Comparing the curves in figures 6.19(a), 6.25(a), and 6.25(b), it could be concluded that results in all three cases have similar features. Moreover, it could be stated that no matter whether the TL values of the tests in the BTS are different or similar, the results behave the same for high resource compatibility and different PD values. However, there are a few differences between the curves in figures 6.19(a), 6.25(a), and 6.25(b). Namely, in figure 6.25(a), the P2ORD algorithm's MPD curve is, for rather loose power constraints (i.e., the upper-half range), somewhere half way between the P1MRU's curve and the other algorithms' MPD curves, whereas in figure 6.25(b),



(a) Different Test Length Values (b) Similar Test Length Values

Figure 6.27: PDD Characteristics' Curves Comparison (Fourth Experiment)



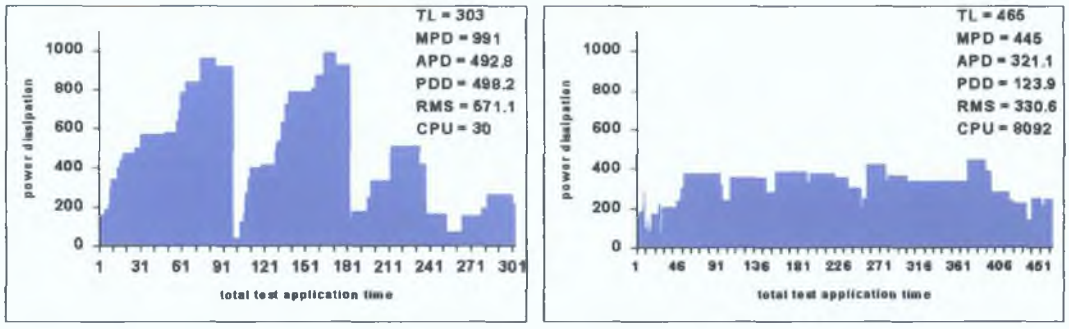
(a) Different Test Length Values (b) Similar Test Length Values

Figure 6.28: APD Characteristics' Curves Comparison (Fourth Experiment)

surprisingly, the P2ORD algorithm exhibits the best results. However, the difference between the P2ORD algorithm's results and the ones generated by the distribution-graph based algorithms (FDS and DV) are not major.

The PDD curves follow the same behaviour as the MPD ones, as can be seen in figures 6.27(a) and 6.27(b). As opposed to the curves of the MPD characteristics, the TL curves in figure 6.26 exhibit the features already experienced in the previous experiments. However, the exceptional aspect of the TL characteristics' curves in this example, compared with the previous ones, is that the list-scheduling algorithms (i.e., P1MRU and P2ORD) generate results in figure 6.26(a) (apart from the case of tight power constraints) with obviously shorter total test application times.

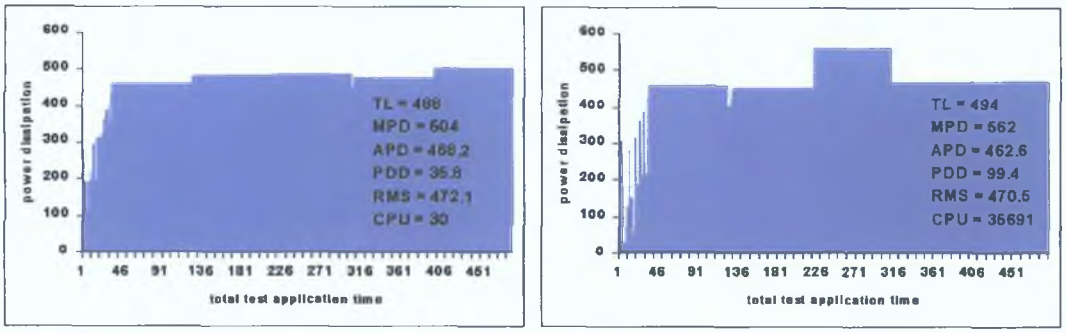
The PTS chart solutions generated by the P1MRU and DV algorithms under very loose power constraint for a BTS example where the tests' TL values are different (first case of this experiment) are compared in figure 6.29. The most obvious thing after drawing the charts in figures 6.29(a) and 6.29(b) is the power balance



(a) P1MRU chart (b) DV chart

Figure 6.29: PTS Charts' Comparison (Fourth Experiment) I

difference between them. Even though the P1MRU algorithm has shorter total test application time, its power dissipation distribution is extremely unbalanced, whereas it is the opposite case for the DV algorithm.



(a) P2ORD chart (b) FDS chart

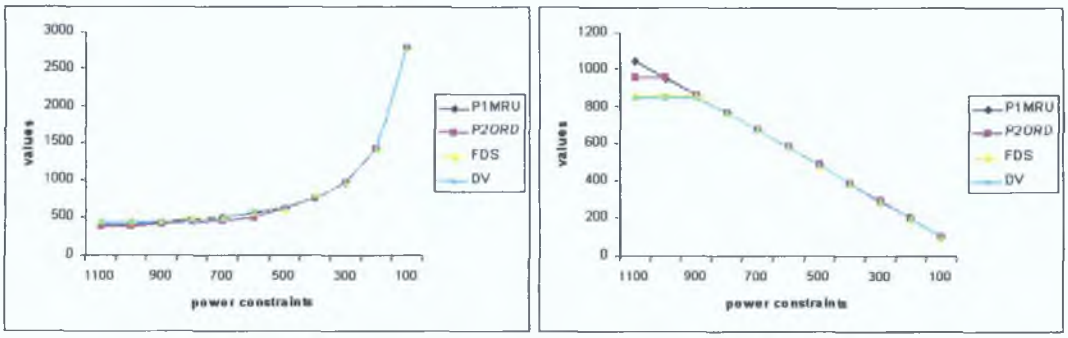
Figure 6.30: PTS Charts' Comparison (Fourth Experiment) II

In figure 6.30 the PTS chart solutions generated by the P2ORD and FDS algorithms for a power constraint of PDC = 700 are compared, for a BTS example where the tests' TL values are similar. The PTS chart generated by P2ORD in figure 6.30(a) is surprisingly almost perfect. This is further emphasized by the very low PDD characteristic value comparing with the MPD one.

The MPD and TL characteristics exhibited by the results generated by the LSFDS types of algorithms are presented next. Only the results for loose power constraints are given, as in the previous experiments. When the BTS has tests with different test length values, the LSFDS algorithms (e.g., P1MRUFDS) improve the MPD characteristic by 35% at the same time with the lengthening of the TL characteristics by 13%. On the other hand, the LSDV algorithms improve by up to 30% the MPD characteristic, by lengthening the TL characteristic with only 8%. When the BTS is consisted of tests with similar test length values, the LSFDS algorithms decrease the power (e.g., MPD) characteristics of the results generated by the LS approaches by 36%, while keeping the TL one constant. The LSDV algorithms decrease the power characteristics by 34% keeping TL constant again.

6.2.5 Fifth Experiment

- runs experiments on a large number of BTSs, in which all tests' parameters (i.e, test length, power dissipation and test resource compatibility) are simultaneously ranged and given values from similar to different with each BTS;



(a) TL Curves (b) MPD Curves

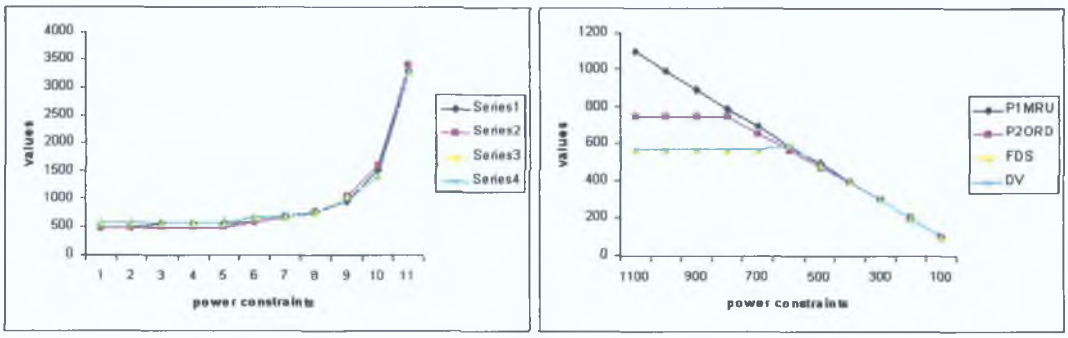
Figure 6.31: PTS Characteristics' Curves Comparison (Fifth Experiment) I

The fifth experiment, which is the largest, is a mixture of all the previous experiments, that is, all three parameters of BTS's tests (i.e., test length, power dissipation and test resource compatibility degree) are varied at the same time. The characteristics of the generated results are not far from the expected combination of the ones discussed in the previous experiments. Therefore only a few cases will be discussed in this section. Those experimental cases that previously generated rather exceptional results are discussed first in order to see if they conform with the previous conclusions. Then, the following three cases that have not been considered before are going to be studied:

- two BTS cases where the minimal values of their TL and PD parameters are half of the maximal ones;
- a BTS case where the tests' resource compatibility degree is high, and their TL and PD parameters have similar values.

Figure 6.31 depicts the TL (figure 6.31(a)) and MPD (figure 6.31(b)) characteristics of the results generated by the representative PTS algorithms, which were run on a BTS similar to the second case discussed in the third experiment (tests with high resource compatibility degree, different test length values and *similar* power values). Comparing pairwise figure 6.31(a) with figure 6.18(b) and figure 6.31(b) with figure 6.19(b), it can be noticed that they are of similar shape. The only remarkable difference is that this time the FDS and DV algorithms give better power characteristics than the P2ORD one (by 18% - 19%) a lengthening of the test application time (by 12% - 14%). Again, both the LSFDS and LSDV types of algorithms generate the same results improving by 17% the MPD characteristic of

the list-scheduling based algorithms for an increase by 7% of the TL characteristic.

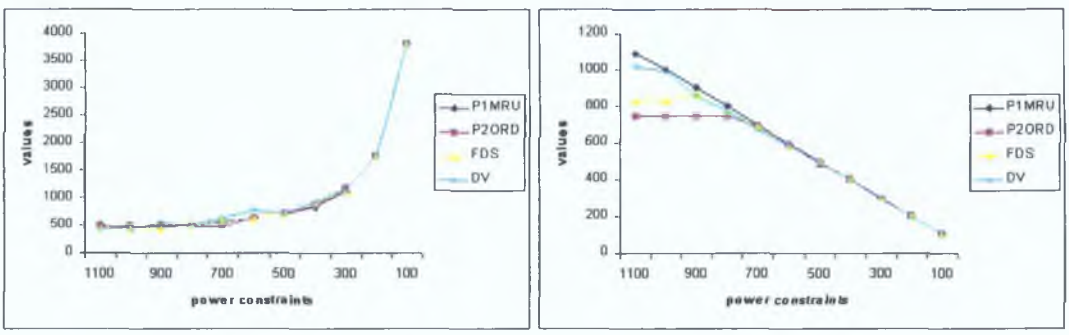


(a) TL Curves (b) MPD Curves

Figure 6.32: PTS Characteristics' Curves Comparison (Fifth Experiment) II

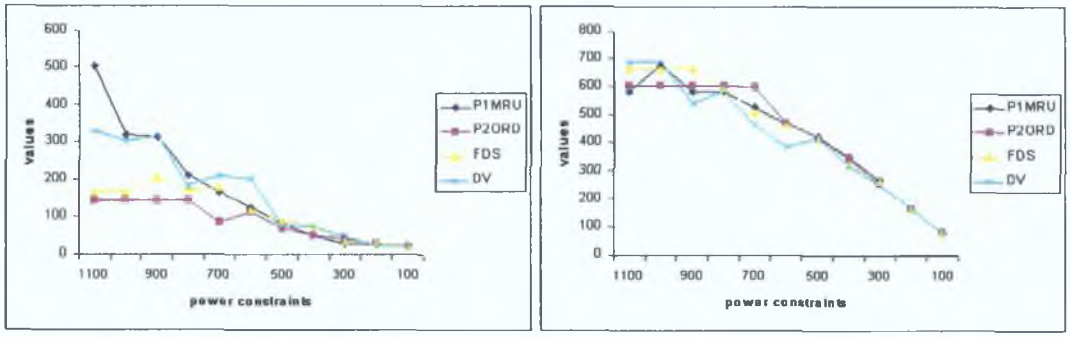
Figure 6.32 depicts the TL (figure 6.32(a)) and MPD (figure 6.32(b)) characteristics of the results generated by the same PTS algorithms, run on a BTS similar to the second case discussed in the fourth experiment (tests with high resource compatibility degree, *similar* test length and different power values). Comparing pairwise figure 6.32(a) with figure 6.26(b) and figure 6.32(b) with figure 6.25(b), it can be noticed again that they are of similar shape. However, this time the results of the P2ORD algorithm do not exhibit better power-test properties than the distribution-based algorithms as in the fourth experiment. The LSFDS and LSDV types of algorithms generate again exactly the same results for the same BTS reducing by 47% the MPD characteristic of the list-scheduling based algorithms for an 16% increase of the TL characteristic.

Figures 6.33 and 6.34 depict the TL (figure 6.33(a)), MPD (figure 6.33(b)), PDD (figure 6.34(a)), and APD (figure 6.34(b)) characteristics of the results generated



(a) TL Curves (b) MPD Curves

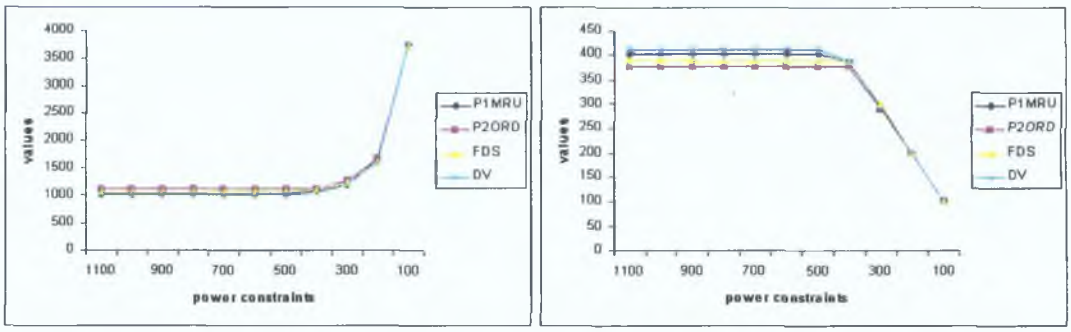
Figure 6.33: PTS Characteristics' Curves Comparison (Fifth Experiment) III



(a) PDD Curves (b) APD Curves

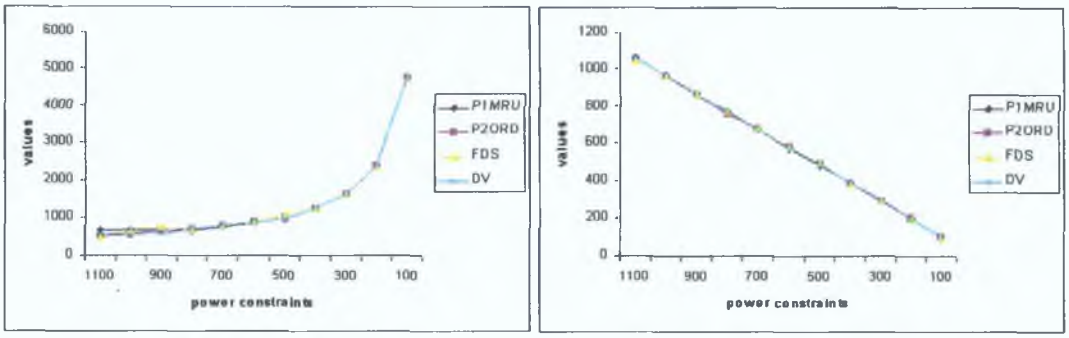
Figure 6.34: PTS Characteristics' Curves Comparison (Fifth Experiment) IV

by the same PTS algorithms for the BTS described next. In this BTS, the tests' resource compatibility degree is high, and the minimal values of their TL and PD parameters are half of the maximal ones. This is a rather exceptional case, because the results of the P2ORD algorithm do exhibit better power-test properties than the distribution-based algorithms as in the fourth experiment. Figure 6.34(a) proves that the results of P2ORD are more balanced than the others, because the PDD characteristics' values are below the ones exhibited by the other PTS algorithms. However, these unpredictable exceptions fade away with the decrease of test resource compatibility degree as could be noticed in figure 6.35. Thus, in this figure the curves of the characteristics start to get flattened over the range of power constraints when BTS's resource compatibility reach an average degree. The BTS in this case is consisted of tests with an average degree of test resource compatibility, and with the minimal values of their TL and PD parameters being half of the maximal ones. For the initial case discussed in this paragraph, it can be added that the LSFDS



(a) TL Curves (b) MPD Curves

Figure 6.35: PTS Characteristics' Curves Comparison (Fifth Experiment) V

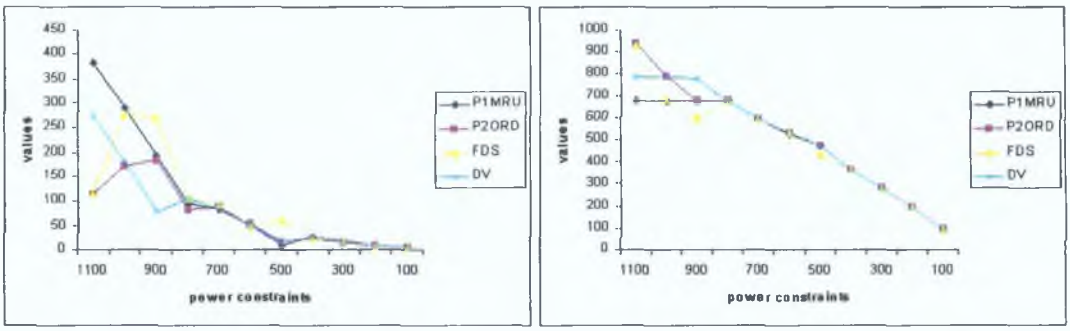


(a) TL Curves (b) MPD Curves

Figure 6.36: PTS Characteristics' Curves Comparison (Fifth Experiment) VI

and LSDV types of algorithms also generate the same results for this BTS case improving by 36% the MPD characteristic of the list-scheduling based algorithms keeping the TL characteristic constant.

Figures 6.36 and 6.37 depict the TL (figure 6.36(a)), MPD (figure 6.36(b)), PDD (figure 6.37(a)), and APD (figure 6.37(b)) characteristics of the results generated by the same PTS algorithms for the final BTS case discussed in this experiment. In this BTS case, the tests' resource compatibility degree is high, and their TL and PD parameters are both very similar. If the TL (figure 6.36(a)) and MPD (figure 6.36(b)) characteristics do not forecast anything special, the PDD (figure 6.37(a)) and APD (figure 6.37(b)) characteristics show the reality behind. Figure 6.37(a) proves that the results of P2ORD compete again with the results of the distribution-graph based algorithms. This kind of BTSs, where the tests' test length and, especially, power dissipation values are similar, proved to generate quite often unpredictable and exceptional results. The only element which neutralizes this effect



(a) PDD Curves (b) APD Curves

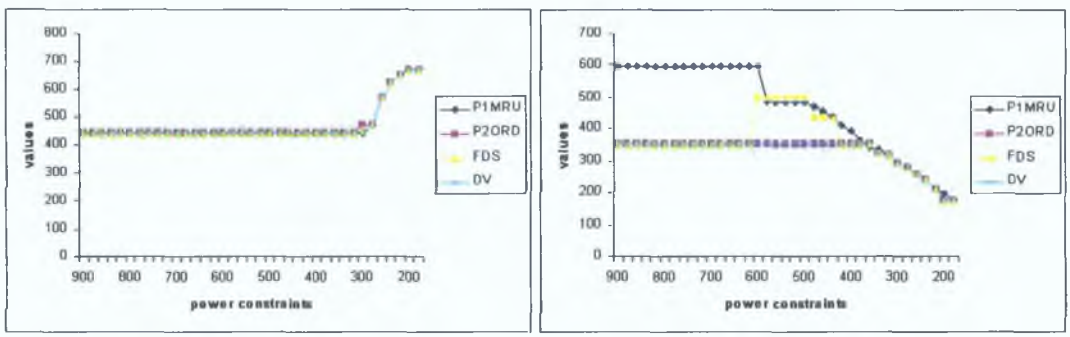
Figure 6.37: PTS Characteristics' Curves Comparison (Fifth Experiment) VII

is the tests' resource compatibility degree. Its decrease flattens the characteristics' curves regardless of the other tests' parameters.

Finally, it can be added here that the LSFDS and LSDV types of algorithms seem to generate results with the same behaviour for any kind of BTS cases. In this particular case, both types of algorithms improve by 19% the MPD characteristic of the list-scheduling based algorithms keeping the TL characteristic constant. Comparing the results generated by all the LSFDS (LSDV) types of algorithms it seems that the MPD characteristics are improved more for BTSs with similar test time values than the cases with similar power values. However, this decrease of MPD values is very often at the cost of a slight increase of the total test application time, and only for loose power constraints.

6.2.6 Sixth Experiment

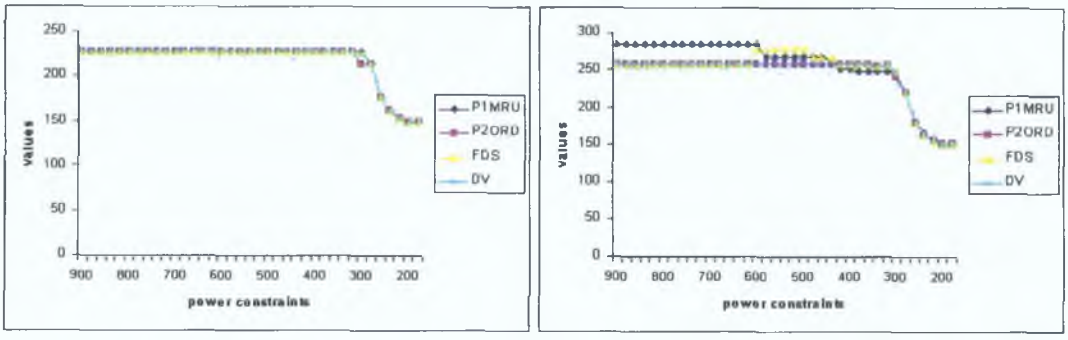
- real case example - extended ASIC Z design.



(a) TL Curves (b) MPD Curves

Figure 6.38: PTS Characteristics' Curves Comparison (Sixth Experiment) I

For the sixth example a real case is considered. An extended case [LP00b] of the ASIC Z design given in [Zor93] is experimented with the PTS algorithms. The test set has 27 tests spread over 9 cores. The whole testbench is listed in appendix A. Characteristics' curves results are depicted in figures 6.38 and 6.39 over a range of power dissipation constraints. Unfortunately, the results of the experiments run here cannot be compared with the ones given for the ASIC Z case in [Zor93, CSA97]. That is due to the fact that the test scheduling discipline assumed in [Zor93, CSA97] is the *nonpartitioned testing* defined in [CKS88], whereas the one assumed in this thesis is the *partitioned testing with run to completion*. The nonpartitioned testing

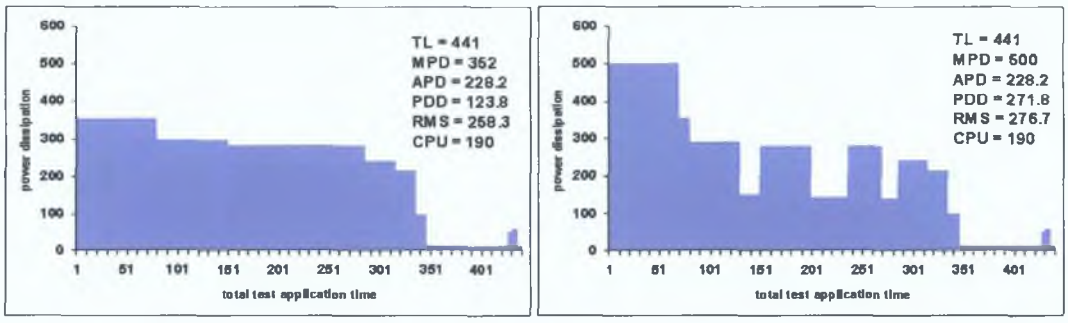


(a) APD Curves (b) RMS Curves

Figure 6.39: PTS Characteristics' Curves Comparison (Sixth Experiment) II

assumes that no tests can be started until all tests in the previous session are completed, which is the opposite of the assumption in this thesis.

The test resource compatibility degree for the ASIC-Z case ranges from 30% for the subset of the tests t_{1i}, t_{2i}, t_{3i}, $\cdots$, t_{9i}, 34% for the subset of tests t_{1e}, t_{2e}, t_{3e}, $\cdots$, t_{9e}, 85% for t_{2b}, t_{4b}, t_{5b}, t_{7b}, 93% for t_{1b}, t_{3b}, and up to 96% for the subset t_{6b}, t_{8b}, t_{9b}. This is an interesting case of mixture between the behaviour of results generated for subsets with low compatibility degree and others with very high compatibility degree. Therefore the solution space has two parts, one where the search space is big (for high resource compatibility degree between tests) and another where the search space is small (for low resource compatibility degree between tests). From the graphs depicted in figures 6.38 and 6.39 it can be noticed that the power constraint of 900mW (aimed at in [Zor93]) is too high for a test scheduling discipline with run to completion for the BTS given in [LP00b] The difference comes from the different test scheduling disciplines employed in both, our and their, approaches. For the test



(a) 800 Power Constraint (b) 540 Power Constraint

Figure 6.40: FDS Charts' Comparison (ASIC-Z, Sixth Experiment)

scheduling discipline adopted in this thesis the maximum power constraint should be around 600mW

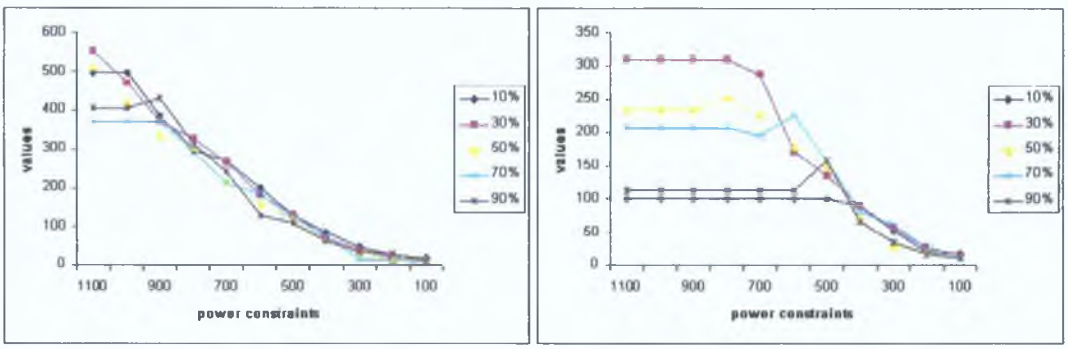
The results are the expected ones, judging by the experiments run beforehand However, an exceptional shape in the characteristics' curves can be noticed for the results generated by the FDS algorithm The suboptimal results generated between 600 and 400 can be blamed on the fact that probably the priority function (i e, force) failed to make the right decision at a certain stage of the iterative solution search process Figure 6 40 compares the power-test charts of two solutions generated by the FDS algorithm before and in the middle of the exceptional part mentioned above As it can be seen the FDS algorithm fails to accommodate some of the blocks along the chart and piles them up at the beginning of the chart This kind of sub-optimal decision could be pruned by employing more sophisticated priority functions, to take into account such particular experimental cases

The LSFDS and LSDV types of algorithms generate again exactly the same results for the ASIC-Z BTS improving by 47% the MPD characteristic of results generated by the list-scheduling based algorithms, keeping the TL characteristic constant

6 2 7 Experimental Conclusions

This chapter ends with a few overall main conclusions over the set of PTS algorithms proposed in this thesis First of all it is recalled that the testbenches used in most of the experiments have been randomly generated The values (resource compatibility degree, test length, power dissipation) assigned to BTS's tests have been generated with a uniform distribution The proposed PTS algorithms in this work have been experimented taking into account the conclusions and suggestions of previous work That is, it has been stated in [CSA97] that in the test environment the difference between the power estimation values (e g , MPD, APD, RMS) for each test is expected to be small since the objective is to maximize the circuit activity so that the circuit can be thoroughly tested in the shortest possible time Therefore, the PTS algorithms try to minimize the differences between the power estimation values, such that the accumulation of these differences is minimized in the bottom-up traversing of the test hierarchy (see chapter 3) Thus, PDD is the power characteristic that was under focus in the experiments while the difference between the MPD and APD values was used to judge the quality of similar PTS

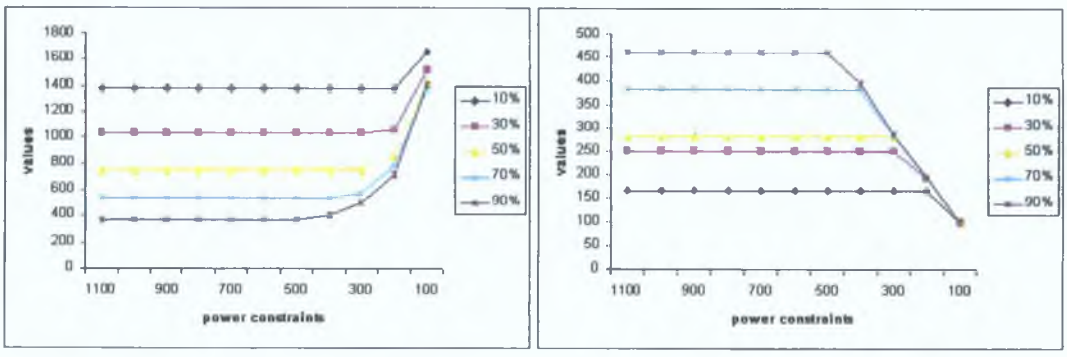
results. The smaller the difference the more balanced the power dissipation distribution. [JPP89] estimated but did not prove that the larger the distribution range ($TL_{max} - TL_{min}$) among the test lengths of the individual tests, the better should be the performance that could be achieved in terms of balancing the power dissipation distribution and achieving good overall test application time. This hypothesis is experimentally proven to be wrong in this thesis (see the fourth and fifth experiments). To further prove this, figure 6.41 shows how the PDD characteristics of the results generated by the PTS algorithms do not necessarily improve for the variance from small (90% similarity between TL values) to large (10% similarity between TL values) distribution ranges of $TL_{max} - TL_{min}$. Moreover, to prove the contrary, figure 6.41(b), which depicts the PDD characteristics of the results generated by the FDS algorithms, shows that the extremes (i.e., smallest and largest $TL_{max} - TL_{min}$ distribution ranges) generally exhibit from the PDD characteristic's point of view, the poorest results, i.e. the least balanced.



(a) P1MRU Algorithm (b) FDS Algorithm

Figure 6.41: PDD Characteristics' Curves for Various Test Length Ranges

The experimental results prove that there are several constraints that implicitly steer the PTS search process. Moreover, there is a certain order or priority of their impact or influence on the PTS algorithms' search engines. Results demonstrate that the power constraints and the test resource compatibility have more impact on the PTS solutions, while the tests' length (TL) and the power dissipation (PD) have less impact. Figure 6.42 depicts the TL and MPD characteristics' curves of the PTS results generated for a variance from small (10%) to large (90%) distribution ranges of the test resource compatibility degree. It can be noticed here that the power constraints have the highest influence on the PTS results, but this influence weakens with the decrease of test resource compatibility degree. The test resource

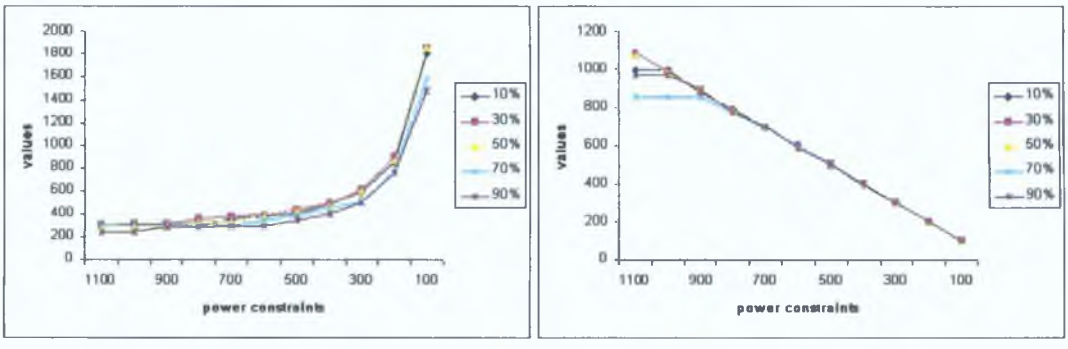


(a) TL Characteristics' Curves (b) MPD Characteristics' Curves

Figure 6.42: Curves for Various Resource Compatibility Ranges (P2ORD)

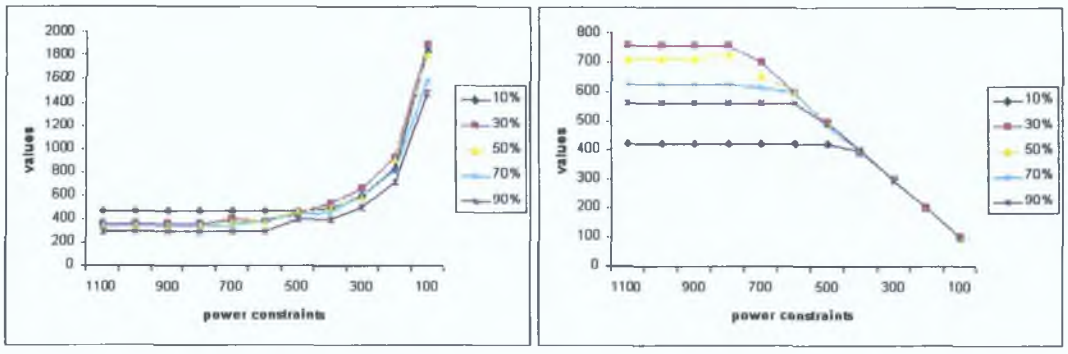
compatibility degree is overall the second most important constraint and the first most important test parameter within a BTS judging by the priorities stated above. The TL and PD test parameters do not have a large impact on the characteristics. However, they do not exhibit a consistent influence on the behaviour of the PTS results, as can be seen in figures 6.43, 6.44, 6.45, and 6.46. They show the TL and MPD characteristics for the results generated for the P1MRU (figures 6.43 and 6.45) and FDS (figures 6.44 and 6.46) algorithms. This paragraph leads to an important design recommendation that every effort be made in hardware to maintain the test parallelism and to insure a high test resource compatibility amongst block-tests.

Two main categories of PTS algorithms have been proposed in this thesis. The first one is the so-called list-scheduling based PTS algorithms (i.e., PTS-LEA and PTS-LS) described in detail in chapter 4. The second set of algorithms are the distribution-graph based PTS algorithms (i.e., PTS-FDS and PTS-DV) and are



(a) TL Characteristics' Curves (b) MPD Characteristics' Curves

Figure 6.43: Curves for Various Test Length Ranges (P1MRU) I

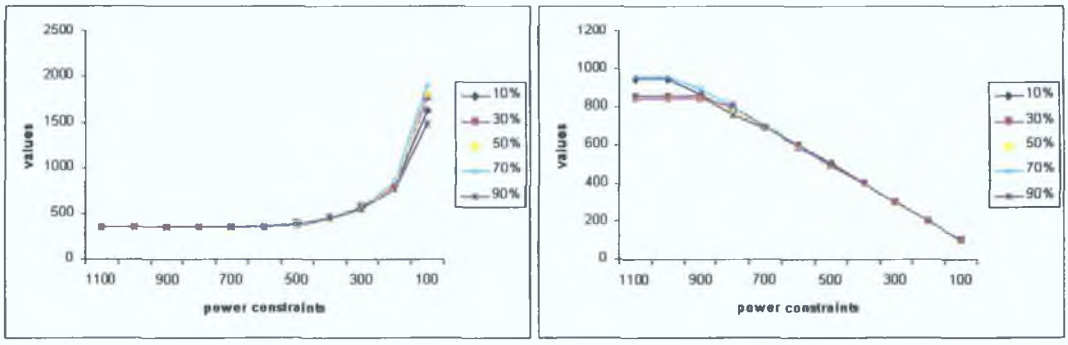


(a) TL Characteristics' Curves (b) MPD Characteristics' Curves

Figure 6.44: Curves for Various Test Length Ranges (FDS) II

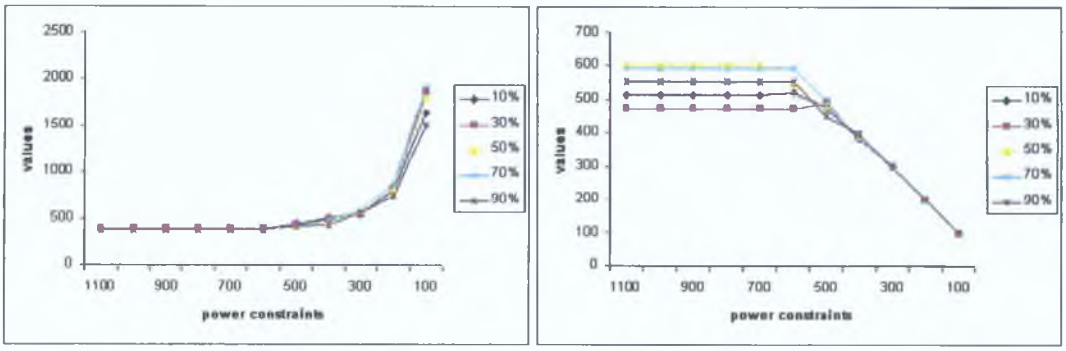
thoroughly described in chapter 5. The main difference between these two categories of PTS algorithms is the priority function used to steer the solution searching process. The list-scheduling based algorithms are considered greedy heuristics where a local (list) priority function is used. The results generated by these algorithms were expected to be not as good as the ones generated by the distribution-graph based algorithms which employ a global priority function.

The experiments proved that even the list-scheduling algorithms can be partitioned into two categories. The first one consists of the first and third approaches of the PTS-LEA algorithm, plus the PTS-LS algorithms itself. The power dissipation is less balanced when these algorithms are loosely constrained. On the other hand when there are tighter power dissipation constraints the total test application time increases, but the power dissipation characteristics are forced to improve. The second set of list-scheduling algorithms are the different insertion strategies of



(a) TL Characteristics' Curves (b) MPD Characteristics' Curves

Figure 6.45: Curves for Various Power Ranges (P1MRU) I



(a) TL Characteristics' Curves (b) MPD Characteristics' Curves

Figure 6.46: Curves for Various Power Ranges (FDS) II

the second approach of the PTS-LEA algorithm. These algorithms can occasionally generate results comparable with that of distribution-graph based algorithms. However, most of the time their results are qualitatively somewhere between the results generated by the first category of list-scheduling algorithms and the distribution-graph based algorithms. Moreover, there are even cases (e.g., those BTSs which consisted of tests' with a high degree of resource compatibility, different time length and similar power values) when these algorithms (P2ORD in this chapter) generate the best results. Otherwise, the PTS-DV algorithms generate more balanced power distribution results, but take longer time. The PTS-FDS algorithms seem to give better (balanced) results but with even higher CPU time. However, there are times when these power balancing achievements are obtained at the expense of a slightly longer total test application time.

In general, the PTS-LSFDS algorithms (i.e., LSFDS and LSDV) give PTS solutions exhibiting better power characteristics than both list-scheduling and distribution-graph based algorithms. Therefore, this algorithm is the most beneficial when the test application time of the solution by list-scheduling based algorithms are shorter than the test application time of the solution by distribution-graph based algorithms. This benefit would justify the longer run time of the PTS-LSFDS (LSFDS) and PTS-LSDV (LSDV) type of algorithms, which is longer than the sum of the time needed to run the list-scheduling and distribution-graph based algorithms sequentially.

A special category of PTS algorithms is the list-scheduling based algorithms

which employ the random (RAND) insertion strategy Their random insertion feature helps them at times to jump out from the local minima and generate surprisingly good results Unfortunately, these algorithms generate bad results with the same frequency as they generate good results Thus, it seems that an intelligent solution search engine has to be used so that it will always generate near-optimal solutions This fact leads to the conclusion that near-optimal algorithms can be successfully employed to generate the best results

Since the difference between the characteristics of the results generated by different PTS algorithms reduces with the tightening of power dissipation constraints, it is advisable for tight power constraints to use list-scheduling based algorithms rather than distribution-graph based ones The list-scheduling based algorithms take considerably shorter CPU times than the other types of PTS algorithms Moreover, it is noticed that the CPU times used even increase sometimes (especially for the distribution-graph based algorithms) with the increase of power constraints Since the differences between their results decreases with the tightening of power constraints, it is advisable to avoid any waste of time The same advice can be given when PTS algorithms are to be applied on testbenches similar to the second testbench of the third experiment (i e, tests with high degree of test resource compatibility, different test length values and *similar* power values)

When the CPU time invested in running the PTS algorithms to achieve a near-optimal power-test schedule is not an issue, a comprehensive approach can be employed This approach would assume the extensive application of all the PTS algorithms in order to generate all the possible results, under any power constraint (from very loose to very tight) Then, the user would only need to select the PTS solution which suits the design needs best according to the suitable MPD, APD, RMS, PDD, and TL characteristics (which normally should be given within some maximal and minimal limits) Therefore, the generated results are supposed to be studied and weighed by the user in order to find the best suitable power-test compromise The power dissipation constraint range could also be spanned with a user-selectable increment of step size depending on the level of complexity the user is ready to pursue for the solution search process

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The work described in this thesis proposes a polynomial-time solution to the NP-complete Power-Constrained Block-Test Scheduling (PTS) problem stated in [CSA97] It is actually the first approach proposed as a solution to the aforementioned problem It is based on classical List Scheduling (LS) and Force-Directed Scheduling (FDS) algorithms in High-Level Synthesis (HLS) adapted to an *extended tree growing modelling* of the PTS problem

This work focuses only on the *high-level* PTS problem The proposed algorithms are meant to be part of a system-level block-test approach to be applied on a modular view of a test hierarchy The modular elements of this hierarchy could be subsystems, backplanes, boards, Multichip Modules MCMs, ICs (dies), macro-blocks and Register Transfer Level (RTL) blocks This approach assumes a bottom-up traversing of the hierarchical test model within a *divide and conquer* optimization style

The algorithms given in the thesis deal with tests for blocks of logic, which do not have equal test length Thus, they are *unequal-length block-test scheduling* algorithms The test order within the test sets of various modules in a circuit is not considered in the algorithm The test scheduling discipline assumed here is the *partitioned testing with run to completion* as defined in [CKS88] A *constant additive model* is employed for power dissipation analysis and estimation throughout the approach

The algorithms implemented here are a projection of the classical LS and FDS

algorithms on the extended tree growing modelling of the PTS problem The first category of scheduling algorithms are the LS-based ones, including Left-Edge Algorithm (LEA) and List Scheduling (LS) algorithm, as described in chapter 4 They have been successfully experimented on the PTS problem These algorithms are fast but the obtained schedules lack the balanced power dissipation property desirable for the PTS solutions The second set of algorithms are the distribution-graph based PTS algorithms, given in chapter 5 These algorithms use a *global priority function* to balance the power dissipation distribution of the PTS solutions The PTS-FDS algorithm gives overall balanced power-test schedules even for tighter power constraints The Distribution Variance (DV)-based PTS approach is another algorithm from the distribution-graph based category It is less time consuming than the FDS-based algorithm and gives more balanced power-test schedules than the LS-based algorithms The mixed PTS approaches given in section 5 3 generate the best power-test schedules, but only for very loose power constraints However, they are the most time consuming approaches

All the PTS algorithms proposed in this thesis use greedy heuristics that are able to generate good schedules in a polynomial time This is very important for the rapid system prototyping of today's VLSI/SOC designs They feature a polynomial complexity but do not guarantee the optimal solutions Therefore, for more refined schedules, the near-optimal algorithms will have to be sought and will be more expensive in terms of computation time

7.2 Contributions

The main contribution of this thesis is that it proposes for the first time a polynomial complexity approach to tackle the NP-complete problem of power-constrained block-test scheduling [CSA97] The extended tree growing technique developed in this thesis combined with the adapted classical scheduling algorithms can quickly generate good schedules for the PTS problem Moreover, the modular test hierarchy proposed to tackle the PTS problem at system level is a flexible and general approach that can be applied to any kind of system test hierarchy

In order to achieve all the above mentioned features of the PTS approach defined in this thesis, a survey of high-level test scheduling and low-power design is carried out and systematically presented in chapter 2 This chapter gave a general view of

today's test design problems having in mind power dissipation constraints Then, a deeper and more detailed survey is presented in chapter 3, where, eventually, the topic of this thesis is identified, formulated and modelled This topic is chosen from the field of low-power test design surveyed in section 3 2 Previous work in the field of power-constrained block-test scheduling (see subsection 3 2 2) proved to be insufficient for the requirements of current technological trends Therefore, the work in this thesis is the first practical solution to the problem analyzed and described in [CSA97]

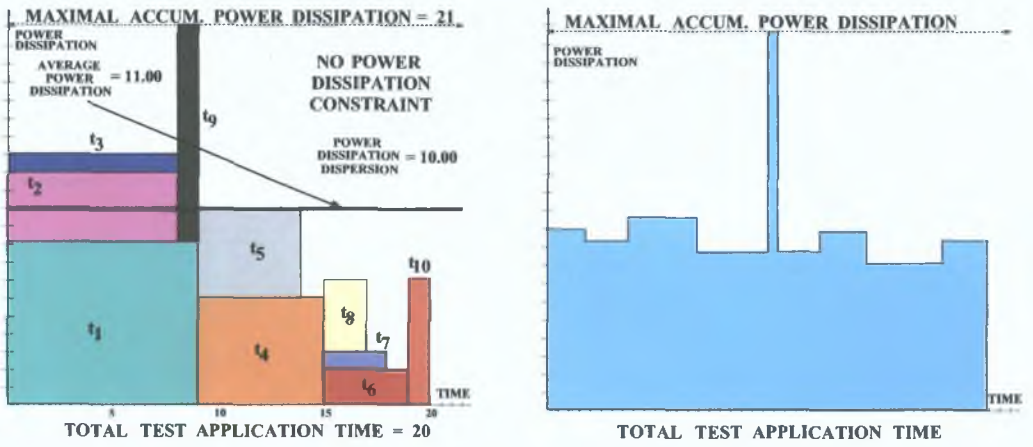
An efficient approach is then formulated in chapter 4 and chapter 5 that adapt classical scheduling algorithms to work with an extended tree growing technique for generating good schedules for the PTS problem These schedules are rapidly generated but they are not guaranteed to be the optimum PTS solutions *Near-optimal algorithms* that can generate optimal or near-optimal solutions for the PTS problem are proposed in the next section They can be the basis of future research work towards finding efficient solutions to other scheduling problems in the field of system-level low-power testing design like *dynamic power management during test*

7.3 Future Work

Future work will be in three main directions One is to further research the power-constrained block-test scheduling field Firstly, a finer tuning of the PTS algorithms proposed in this thesis has to be carried out to take into account real technological aspects of the PTS problem The second research direction is to apply near-optimal search algorithms to the PTS solution space In order to give full freedom to these near-optimal search algorithms a different modelling of the power-test solution space has to be sought The third direction is to investigate the applicability of the PTS algorithms to other power-test related scheduling fields like power-test scheduling with dynamic frequency, multiple voltage supply scheduling or burn-in power-test scheduling

7 3 1 Technological Aspects of PTS Problem

The distribution-graph based PTS algorithms proposed in chapter 5 are aimed at balancing power dissipation during test application time without taking into account the technological aspects of particular power dissipation cases For example, in



(a) PTS-LEA (P1MRU) Schedule (b) Another Schedule Example

Figure 7.1: Power-Test Scheduling Charts

figure 7.1 the power dissipation spike given by test t_9 in figure 7.1(a) and the obvious power spike in figure 7.1(b) could be ignored if they do not last for a long period of time. The mathematical formulation of the Power-Concurrency Distribution Graphs (PCDG) in chapter 5 is not meant to deal with these particular cases. Therefore, further research work is needed to tackle the particular technological aspects of power dissipation during test application time. Moreover, finer tuned mathematical formulations of the PCDG might be found in this context to emphasize more the energy consumed during test application time than the power dissipation peaks at a certain moment. That is, thus far the MPD and PDD values were the characteristic values used to drive the search for the best solutions. But if the total energy over test application time is to be more important, then characteristic values like PDD and RMS should be of more interest.

The PTS algorithms proposed in this thesis were tackling the problem of power-constrained test scheduling, where the solution search was aiming for the shortest total test application time, making sure that the power dissipation barrier is not crossed. Thus, it was a power-constrained total test application time minimization algorithm. However, there might be cases when the total test application time might be seen as a constraint and the goal could be to minimize the power dissipation over the aforementioned application time. This is another research direction to be pursued in future.

The PTS algorithms proposed in this thesis try to optimize the power-test characteristics of the design in the test mode However, these techniques will be researched in future as possible approaches for tackling the power-test optimization in the normal mode as well Thus, an extrapolation of the power-constrained test scheduling techniques proposed here will be applied to the problem of scheduling operations in the normal functional mode In order to make this extrapolation, the PTS algorithms will have to deal with another level of constraint, which will be the precedence relationship between the operations to be scheduled

7 3 2 Near-Optimal PTS Approaches

The biggest drawback of the tree growing heuristic is that the solution space is shrunk by the fact that the power-test scheduling charts should have low-mobility tests as roots (or close to roots), while high-mobility tests are leaves (or close to leaves) This is due to the fact that the test length of the nodes (test) in a tree path (ETP) have to be monotonously decreasing from root to leaf (see subsection 3 3 3) Thus, having the solution space diminished, the heuristics proposed in this thesis, even though they are fast, they can not guarantee the optimum power-test schedules This would not happen if the tests had the freedom to be placed anywhere in the power-test schedule Therefore, as future work, it is proposed here to model the map of power-test schedules as a puzzle game, where each puzzle piece (block-test) has to find its own place in the optimum power-test schedule Thus, the compatibility relationship between the block-tests will have one component less (see subsection 3 3 3) Namely, the test length of the nodes (block-tests) in the power-test schedule *do not* have to be monotonously decreasing from root to leaf Actually, the root and leaf terms disappear for the puzzle model Therefore, the idea of tree path becomes useless and the compatibility relationship reduces to the two classic power-test compatibility conditions (see subsection 3 3 2) Firstly, in order to run tests in parallel, they have to be compatible from a conflicting resources point of view Secondly, the power dissipation accumulated at any moment during parallel test application should be less than or equal to the power dissipation constraint PD_{max}

Moreover, the RAND insertion version of the second PTS-LEA pseudocode has constantly exhibited surprises throughout the experiments in chapter 6 by generating results from the worst to the best This was due to the fact that the PTS

algorithms proposed in this thesis do not have a mechanism to steer through the randomly generated search space Therefore, application of near-optimal algorithms like simulated annealing, tabu search or genetic algorithms to the PTS problem is a promising research topic to be investigated

The above described puzzle game resembles the two dimensional block constrained placement problem [SV97] or the bin packing technique employed for local repacking/rearrangement during Field Programmable Gate Array (FPGA) reconfiguration [DE97] The former approach has to solve the two constrained placement problems during the generation of VLSI macro-cell layouts The latter is used during FPGA reconfiguration, when partial rearrangement is required to alleviate the fragmentation of free logic elements that occurs on space-shared run-time reconfigurable FPGA systems

Simulated Annealing

A Simulated Annealing (SAn) [KGV83] algorithm is a neighborhood search algorithm where the neighbourhood is sampled at random It differs from the greedy algorithms in the fact that a neighbour giving rise to an increase in the cost function may be accepted This acceptance will depend on a control parameter (called temperature) and the magnitude of the increase By allowing uphill moves in a controlled manner, Simulated Annealing (SAn) provides a mechanism to allow the algorithm to escape from local optima The algorithm starts with an initial solution A neighbor of this solution is then randomly selected If the selected solution is better than the current solution, it will always be accepted and become the next solution If the selected solution is worse, it will be accepted with a probability factor Therefore, at the beginning when the temperature t is high, the probability of accepting a worse neighbour and making an uphill move is high With the reduction of temperature, this probability decreases

Tabu Search

Tabu Search (TS) [Hal96] is a neighbourhood-search method which employs intelligent search and flexible memory technique to avoid being trapped at local optima As in the case of simulated annealing, Tabu Search (TS) is a high-level heuristic procedure used to guide other methods towards an optimal solution TS is based on the assumption that intelligent search should be based on more systematic forms of

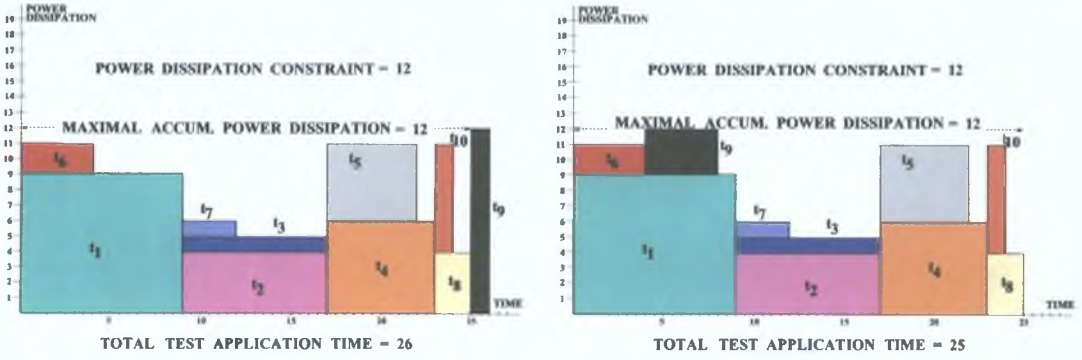
guidance rather than random selection It also exploits flexible memory to control the search process The main mechanism for exploiting memory is to classify a subset of the neighborhood moves as forbidden (called tabu) TS maintains a selective history of the states encountered and/or the moves executed during the search

Genetic Algorithms

A Genetic Algorithm (GA) [SP94, SV97] is a high-level algorithm which performs a multi-directional search by maintaining a population of potential solutions The population undergoes a simulated evolution from one generation to another at each generation the relatively good solutions reproduce, while the relatively bad solutions die The goodness or badness of the solutions are defined by a cost function In the Genetic Algorithm (GA) an optimization problem is mapped into the problem of finding the most fit individual within a population during an evolution process Fitness is measured by a fitness function, which is related to the objective function of the optimization problem A GA starts with a set of initial solutions Each solution is encoded as a chromosome which is represented as a string of bits from a binary alphabet To generate new solutions there are two typical operations which are performed on the solutions of the present generation crossover and mutation For the crossover operation, two solutions S_1 and S_2 of the current generation are selected and the chromosome corresponding to a new solution is produced The new chromosome is the result of mixing a part of the chromosome of S_1, with a part of that corresponding to S_2 This means that the new solution inherits certain features of its two parent solutions The mutation operator, on the other hand, produces a small, random perturbation to a given solution (chromosome)

7 3 3 Dynamic Power Management During Test

Dynamic power management is a system-level low power design technique aiming at controlling performance and power levels of digital circuits and systems, by exploiting the idleness or the activity of their components It can be seen in figure 7 2 that for the same set of block-tests, different solutions can be generated if stretching techniques are applied to block-tests In this case test t_9 from figure 7 2(a) is stretched and rescheduled in figure 7 2(b) The stretching technique is to expand the test application time (test length) of the block-tests with the goal of making full use of the remaining power dissipation below the constraint in the scheduling charts



(a) Without Stretched Block-Tests (b) With Stretched Block-Tests

Figure 7.2: Test Scheduling Example with Block-Test Stretching

The result of stretching can decrease the power dissipation by either voltage scaling or dynamic frequency clocking or it can increase the fault coverage by increasing the number of test vectors. In the latter case the power dissipation distribution is expected to be approximately the same. That is, even though the test application time will be increased to apply further test vectors, the power dissipation values are considered, during their application, to be around the same constant power dissipation value associated with the block-test by the high-level power estimation process (see subsection 3.3.1).

Power-Test Scheduling with Voltage Scaling

Most techniques to lower power consumption of ICs assume static behaviour. That is, circuit and system parameters are chosen at design time to minimize power dissipation. Power-down techniques can be used to make power dissipation directly proportional to the computational workload [MK96]. Power dissipation can be reduced if a variable power supply is used in conjunction with a variable clock-speed processor. The basic idea is to lower supply voltage and slow the clock during reduced workload periods instead of working at a fixed speed and idling.

Datapath scheduling techniques and behavioural synthesis techniques with multiple supply voltages were recently proposed [CP97, JR97, LHW97, RS95] as power optimization techniques. The proposed scheduling techniques refer to the assignment of a supply voltage to each operation in a data flow graph so as to minimize the average energy consumption for given computation time or throughput constraints or both. In the past few years, low power techniques by dynamic voltage

scaling have been studied [GC97, NNSaB94, WH96] and efficient scheduling algorithms for these techniques are being sought For example, [NNSaB94] presents a technique that combines self-timed circuitry with a mechanism that adaptively adjusts the supply voltage to the minimum possible, taking into account process variations, operating conditions and data dependent computation times The approach in [NNSaB94] finds the optimal voltage based on adaptive methods, while the approach in [IY98] is based on static scheduling technique which treats dynamically variable voltage [IY97] The PTS algorithms proposed in this thesis and the near-optimal algorithms mentioned in subsection 7 3 2 can be linked with the dynamic voltage scaling techniques to efficiently search the solution space to find good power-test profiles

Power-Test Scheduling with Dynamic Frequency

In [RVB98, KRV99, RVB96] dynamic frequency techniques are presented as a solution to the power minimization problem In the dynamic frequency scheme all units are driven by a single clock line which changes frequency at run time depending on the functional unit active at that time as in figure 7 3 Dynamic Frequency Clocking (DFC) utilizes the fact that different (e g , adders, multipliers etc) can be clocked at a different frequency based on their critical path delay The idea in [KRV99] proposes a time and resource constrained scheduling algorithm which utilizes the concepts of DFC and Multiple Voltage Scheduling (MVS) Only the frequency is changed in [KRV99] dynamically, while the supply voltage for each Functional Unit (FU) is fixed from one of the available levels (5 0V, 3 3V, 2 4V) Self-timed systems have also been suggested to take advantage of data dependencies (workload) [GC97]

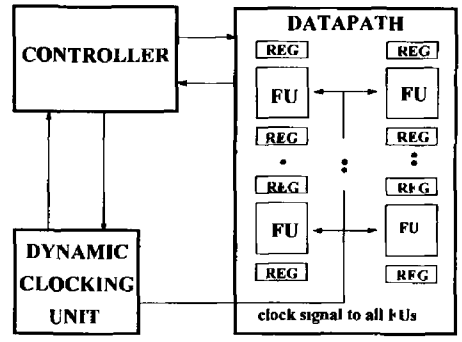


Figure 7 3 DFC Architecture

Nobody proposed so far, though, a way to mix these techniques with the power-constrained test scheduling problem These dynamic-frequency technological solutions can be efficiently combined with the power-test scheduling techniques described at the beginning of this subsection (7 3 3)

Test Scheduling for Monitored Burn-In

The growing size and complexity of VLSI/SOC designs and the reduction in feature sizes makes production of reliable chips a challenging task *Stress testing* is an effective method to improve product reliability In figure 7 4 failure probability is plotted against time This curve is known as the bath-tub curve [DC96] It can be seen that failure probability is high in the early period of product life This is known as *infant mortality* It is explained by the presence of "weak ICs" in the production lots Such ICs contain imperfections which are a consequence of manufacturing defects "Weak ICs" fail soon after passing production testing Therefore, during stress testing ICs are subjected to stress conditions and the "weak ICs" fail and are not shipped out improving product reliability

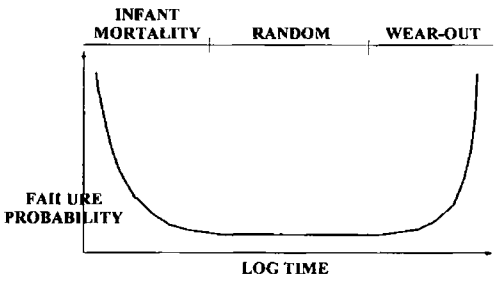


Figure 7 4 Bath Tube Curve

ICs are stressed in a variety of ways such as burn-in, power cycling, temperature cycling, voltage variations, clock variations Burn-in, the standard method used in industry so far for stressing ICs, is to subject ICs to high temperature and high voltage for an extended period [Hna84] Burn-in induces cumulative stress failures During burn-in, cyclic sequences are applied over an extended period of time such that the switching activity in the circuit is maximized Typically burn-in is performed for an extended period, usually several hours So, while selecting vectors for burn-in, it would be of more concern that the ability of the resulting burn-in test sequence to dissipate power rather than its application length

There are different kinds of burn-in (static burn-in [Hna84], high-voltage cell stress testing [Hna84], dynamic burn-in [DC96, DCNP95]), but only the monitored burn-in is of interest to us for future research

Burn-in is used to eliminate *infant mortality* in VLSI/SOC designs because it induces cumulative stress failures There are three kinds of burn-in *Static burn-in* applies a DC bias to the device at an elevated temperature (normally 125°C) [Hna84] Then, in *high-voltage cell stress* tests are created to rapidly test memories and involves cycling through all memory addresses [Hna84] The *dynamic burn-in* applies a random sequence on all the clocks and address lines [DC96, DCNP95] Finally, *monitored burn-in* assumes testing during dynamic burn-in and is becoming widespread because of long electrical test times associated with large circuits [DCNP95]

Only the monitored burn-in is of interest to us for future research This burn-in is a combination of dynamic burn-in, high voltage cell stress testing and electrical functional pattern testing Monitored burn-in is a technique in which devices are operated at an elevated temperature (125°C) and voltage (8-8 5V) for an extended period of time while subjecting all devices under test to functional testing using complex test patterns This is followed by a short duration at a lower temperature (-85°C) and voltage (5 5V) during which parametric testing is performed This cycle is performed in a few hours range Since the time to burn-in the devices has already been committed, the functional testing when performed in this parallel manner is essentially for free In the traditional manner, the functional testing is performed after burn-in to detect failures Because this approach is a serial process it is expensive and, thus, the test time can be reduced by overlapping these test steps With VLSI/SOC designs, out of necessity, test time must be increased to effectively validate good parts due to their sheer complexity Thus monitored burn-in solves those problems and allows long test time at reasonable cost to thoroughly assess the inherent quality of the VLSI/SOC designs There are some advantages in using the monitored burn-in against dynamic burn-in [DCNP95] It utilizes the "dead time" during burn-in testing *Carefully ordered* test vectors can stress circuit nodes to their maximum During dynamic burn-in, if the input vectors are not carefully chosen, switching activity in some parts may not be sustained at a higher level If a junction breaks down during burn-in, monitored burn-in will make it easier to detect the location of the fault

The drawbacks of the traditional monitored burn-in were defined in [DCNP95] In dynamic burn-in, selection of vectors is not restricted to a test set, and higher switching activity can be generated than for monitored burn-in The latter requires

a tester throughout the burn-in process and the cost of tying down a tester for a long period could be prohibitively high Also, in monitored burn-in the output of test application has to be latched back to scan registers A solution to these drawbacks is the BIST monitored burn-in Thus, a BIST-based test methodology can be used to carry out the functional test during burn-in itself At the same time the first drawback mentioned above would be avoided by the pseudo-random nature of BIST test methodology The second drawback is also avoided with the BIST monitored burn-in approach

All the monitored burn-in approaches are given at logic or test vector level One possible future research direction would be to increase the switching activity at higher levels, where the logic-level transition maximization can be replaced with block-level switching maximization For each high-level node in a system-test hierarchy (given as a set of subnodes with their block-tests) the task of a monitored burn-in test scheduling algorithm would be to generate an effective block-test schedule to maximize the stressing of the node By maximizing the stress at any node level, the stress would hierarchically accumulate at system level Thus, to extrapolate the work proposed in this thesis, the stressing maximization at system level can be achieved by maximizing the power-test characteristics at node-level in the modular test hierarchy described in chapter 3

Bibliography

[AAMH98] H Al-Assad, B T Murray, and J P Hayes Online BIST for Embedded Systems *IEEE Design and Test of Computers*, 15(4) 17–24, Oct-Dec 1998

[AB85a] M Abadir and M Breuer Constructing Optimal Test Schedules for VLSI Circuits Having Built-in Test Hardware In *Proceedings of The International Symposium of Fault-Tolerant Computers*, pages 165–170, Jun 1985

[AB85b] M S Abadir and M A Breuer A Knowledge-Based System for Designing Testable VLSI Chips *IEEE Design and Test of Computers*, pages 56–68, Aug 1985

[AB86] M Abadir and M Breuer Test Schedules for VLSI Circuits Having Built-in Test Hardware *IEEE Transactions on Computers*, C-35(4) 361–368, Apr 1986

[ABF94] M Abramovici, M A Brewer, and A D Friedman *Digital Systems Testing and Testable Design* IEEE Press, 1994

[AM93] L Avra and E J McCluskey Synthesizing for Scan Dependence in Built-In Self-Testable Designs In *Proceedings of The International Conference of Computer-Aided Design*, pages 30–35, 1993

[And97] T L Anderson Thoughts on Core Integration and Test In *Proceedings of The International Test Conference, ITC'97*, pages 1039–1039, 1997

[AS98] H J Wunderlich A P Stroele Hardware-Optimal Test Register Insertion *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(6) 531–539, Jun 1998

[Avr91] L Avra Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths In *Proceedings of the 1991 International Test Conference*, pages 463–472, Oct 1991

[Ben97] B Bennetts A Design Strategy for System-on-a-Chip Testing *Electronic Products*, pages 57–59, Jun 1997

[BKH97] J Blatny, Z Kotasek, and J Hlavicka RT Level Test Scheduling *Computers and Artificial Intelligence*, 16(1) 13–29, Jun 1997

[BWBM92] A Basu, T C Wilson, D K Banerji, and J C Majithia An Approach to Minimize Testability Overhead for BILBO based Built-In Self-Test In *Proceedings of The IEEE VLSI Test Symposium*, pages 55–59, 1992

[CD94a] S Chakravarty and V P Dabholkar Minimizing Power Dissipation in Scan Circuits During Test Application In *Technical Report No 94-06, Dept of Computer Science, SUNY at Bufallo*, 1994

[CD94b] S Chakravarty and V P Dabholkar Minimizing Power Dissipation in Scan Circuits During Test Application In *Proceedings of the IEEE International Workshop on Low Power Design*, pages 51–56, 1994

[CD94c] S Chakravarty and V P Dabholkar Two Techniques for Minimizing Power Dissipation in Scan Circuits During Test Application In *Proceedings of The 3rd Asian Test Symposium*, pages 324–329, 1994

[CD94d] S Chakravarty and V P Danholkar Two Techniques for Minimizing Power Dissipation in Scan Circuits During Test Application In *Proceedings of The 3rd Asian Test Symposium*, 1994

[CFN+98] J C Costa, P F Flores, H C Neto, J C Monteiro, and J P Marques Silva Power Reduction in BIST by Exploiting Don't Cares in Test Patterns In *Technical Report, Instituto Superior Tecnico, Cadence European Laboratories/INESC, Lisboa, Portugal*, 1998

[CFN+99] J C Costa, P F Flores, H C Neto, J C Monteiro, and J P Marques Silva Assignment and Reordering of Incompletely Specified Pattern Sequences Targeting Minimum Power Dissipation In *Proceedings of the 12th International Conference on VLSI Design*, 1999

[Cha99] K Chakrabarty Test Scheduling for Core-Based Systems In *Proceedings of The IEEE International Conference on Computer-Aided Design*, pages 391–394, 1999

[Che91] C-I H Chen Graph Partitioning for Concurrent test Scheduling in VLSI Circuit In *Proceedings of IEEE Design Automation Conference*, pages 287–291, 1991

[CK85] G L Craig and C R Kime Pseudo-Exhaustive Adjacency Testing A BIST Approach for Stuck-Open Faults In *Proceedings of The IEEE International Test Conference*, pages 126–137, 1985

[CKS88] G L Craig, C R Kime, and K K Saluja Test Scheduling and Control for VLSI Built-In Self-Test *IEEE Transactions on Computer*, 37(9) 1099–1109, Sep 1988

[CKS94] C -H Chen, T Karnik, and D G Saab Structural and Behavioral Synthesis for Testability Techniques *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(6) 777–785, Jun 1994

[CP97] J M Chang and M Pedram Energy Minimization Using Multiple Supply Voltages *IEEE Transactions on VLSI Systems*, 5(4) 436–443, Dec 1997

[CSA94] R M Chou, K K Saluja, and V D Agrawal Power Constraint Scheduling of Tests In *Proceedings of The 7th International Conference on VLSI Design*, pages 271–274, Jan 1994

[CSA97] R M Chou, K K Saluja, and V D Agrawal Scheduling Tests for VLSI Systems Under Power Constraints *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(2) 175–185, Jun 1997

[CY92] C H Chen and J T Yuen Concurrent Test Scheduling in Built-In Self-Test Environment In *Proceedings of The IEEE International Conference on Computer Design*, pages 256–259, 1992

[Dav81] S Davidson Some Experiments in Local Microcode Compaction for Horizontal Machines *IEEE Transactions on Computers*, C-30(7) 460–477, Jul 1981

[DC94] V P Dabholkar and S Chakravarty Minimizing Power Dissipation in Combinational Circuits During Test Application In *Technical Report No 94-10, Dept of Computer Science, SUNY at Bufallo*, 1994

[DC96] V P Dabholkar and S Chakravarty Dynamic Stress Tests for "Narrow Metal Imperfections" in Full Scan Circuits In *Technical Report 96-06, Dept of Computer Science, SUNY at Bufallo*, 1996

[DCNP95] V P Dabholkar, S Chakravarty, F Najm, and J Patel Cyclic Stress Tests for Full Scan Circuits In *Proceedings of The IEEE VLSI Test Symposium*, 1995

[DCPR98] V P Dabholkar, S Chakravarty, I Pomeranz, and S Reddy Techniques for Minimizing Power Dissipation in Scan and Combinational Circuits During Test Application *IEEE Transactions on Computers*, 17(12) 1325–1333, Dec 1998

[DE97] O Diessel and H ElGindy Partial FPGA Rearrangement by Local Repacking In *Technical Report 97-08*, pages 1–31 University of Newcastle, Australia, Sep 1997

[Diw91] A A Diwan An Algorithm for Minimizing the Number of Test Cycles In *Proceedings of The 4th IEEE International Symposium on VLSI Design*, pages 154–156, 1991

[DM95] S Devadas and S Malik A Survey of Optimization Techniques Targeting Low Power VLSI Circuits In *Proceedings of The 32nd Design Automation Conference*, pages 242–247, 1995

[DP94a] S Dey and M Potkonjak Transforming Behavioral Specifications to Facilitate Synthesis of Testable Designs In *Proceedings of The International Test Conference, ITC'94*, pages 184–193, 1994

[DP94b] S Dey and M Potkonjak Non-Scan Design-For-Testability of RTL Data Paths In *Proceedings of The International Conference of Computer-Aided Design*, pages 640–645, 1994

[EW91] B Eschermann and H J Wunderlich Parallel Self-Test and the Synthesis of Control Units In *Proceedings of The IEEE European Test Conference, ETC'91*, pages 73–82, Apr 1991

[FM91] S Feng and Y K Malaiya Optimization of Test Parallelism with Limited Hardware Overhead *Microelectronics Reliability*, 31(2/3) 271–276, 1991

[GB91] R Gupta and M A Breuer Ordering Storage Elements in a Single Scan Chain In *Proceedings of The International Conference on Computer-Aided Design*, pages 408–411, 1991

[GB95] R Gupta and M A Breuer Partial Scan Design of Register-Transfer Level Circuits *Journal of Electronic Testing Theory and Applications*, 7(1-2) 25–46, Aug-Oct 1995

[GC97] V Gutnik and A P Chandrakasan Embedded Power Supply for Low-Power DSP *IEEE Transactions on VLSI Systems*, 5(4) 425–435, Dec 1997

[GGL+99] P Girard, L Guiller, C Landrault, S Pravossoudovitch, J Figueras, S Manich, P Teixeira, and M Santos Low Energy BIST Design Impact of the LFSR TPG Parameters on the Weighted Switching Activity In *Proceedings of The IEEE International Symposium on Circuits and Systems, CD-ROM proceedings*, 1999

[GGLP99a] P Girard, L Guiller, C Landrault and S Pravossoudovitch A Test Vector Inhibiting Technique for Low Energy BIST Design In *Proceedings of The VLSI Test Symposium*, pages 407–412, 1999

[GGLP99b] P Girard, L Guiller, C Landrault, and S Pravossoudovitch A Test Vector Ordering Technique for Switching Activity Reduction during Test Operation In *Proceedings of The IEEE Great Lakes Symposium on VLSI*, pages 24–27, 1999

[GGLP99c] P Girard, L Guiller, C Landrault, and S Pravossoudovitch Circuit Partitioning for Low Power BIST Design with Minimized Peak Power Consumption In *Proceedings of The IEEE Asian Test Symposium*, pages 89–94, 1999

[GGLP00] P Girard, L Guiller, C Landrault, and S Pravossoudovitch An Adjacency-Based Test Pattern Generator for Low Power BIST Design In *Proceedings of The IEEE Asian Test Symposium (submitted)*, 2000

[GKP95] X Gu, K Kuchcinski, and Z Peng An Efficient and Economic Partitioning Approach for Testability In *Proceedings of The International Test Conference, ITC'95, Washington D C*, pages 680–685, October 1995

[GLPS98] P Girard, C Landrault, S Pravossoudovitch, and D Severac Reducing Power Consumption during Test Application by Test Vector Ordering In *Proceedings of The IEEE International Symposium on Circuits and Systems, CD-ROM proceedings*, 1998

[Gu96] X Gu *RT Level Testability Improvement by Testability Analysis and Transformations* PhD thesis, Linkooping University, 1996

[GW99] S Gerstendorfer and H J Wunderlich Minimized Power Consumption for Scan-Based BIST In *Proceedings of The IEEE International Test Conference*, pages 77–84, 1999

[Hal96] J Hallberg High-Level Synthesis under Local Timing Constraints Master's thesis, Linkooping University, 1996

[Hna84] E R Hnatek Thoughts on VLSI Burn-In In *Proceedings of The International Test Conference*, pages 531–534, 1984

[HO94] I G Harris and A Orailoglu Fine-Grained Concurrency in Test Scheduling for Partial-Intrusion BIST In *Proceedings of The European Design & Test Conference*, pages 119–123, 1994

[HP93] H Harmanani and C Papachristou An Improved Method for RTL Synthesis with Testability Tradeoffs In *Proceedings of The International Conference of Computer-Aided Design*, pages 30–35, 1993

[HS71] A Hashimoto and J Stevens Wire Routing by Optimizing Channel Assignment within Large Apertures In *Proceedings of The 8th Design Automation Conference Workshop*, pages 155–169, 1971

[Hug97] W J HughesIII System Level Boundary Scan in a Highly Integrated Switch In *Proceedings of The IEEE International Test Conference, ITC'97*, pages 636–639, 1997

[HW98] A Hertwig and H J Wunderlich Low Power Serial Built-In Self-Test In *Proceedings of The IEEE European Test Workshop*, pages 49–53, 1998

[IY97] T Ishihara and H Yasuura Optimization of Supply Voltage Assignment for Power Reduction on Processor-Based Systems In *Proceedings of The 7th Workshop on Synthesis and System Integration of Mixed Technology*, pages 51–58, 1997

[IY98] T Ishihara and H Yasuura Voltage Scheduling Problem for Dynamically Variable Voltage In *Proceedings of The International Symposium on Low-Power Electronics and Design*, pages 197–202, 1998

[JPP89] W B Jone, C Papachristou, and M Pereira A Scheme for Overlaying Concurrent Testing of VLSI Circuits In *Proceedings of Design Automation Conference*, pages 531–536, Jun 1989

[JR97] M C Johnson and K Roy Datapath Scheduling with Multiple Supply Voltages and Level Converters *ACM Transactions on Design Automation of Electronic Systems*, 2(3) 227–248, Jul 1997

[JY00] J Janssen and R Yassawi Private Discussion (Dalhousie University & Trent University, Canada), 1999-2000

[KAHA97] P Kollig, B M Al-Hashimi, and K M Abbott Efficient Scheduling of Behavioral Descriptions in High-Level Synthesis *IEE Proceedings-Computers And Digital Techniques*, 144(2) 75–82, Mar 1997

[KGV83] S Kirpatrick, C D Gelatt, and M P Vecchi Optimization by Simulated Annealing *Science*, 220(4598) 671–680, 1983

[KP87] F J Kurdahi and A C Parker REAL A Program for Register Allocation In *Proceedings of The 24th Design Automation Conference*, pages 210–215, 1987

[KRV99] V Krishna, N Ranganathan, and N Vijaykrishnan Energy Efficient Datapath Synthesis Using Dynamic Frequency Clocking and Multiple Voltages In *Proceedings of the 12th International Conference on VLSI Design*, pages 440–445, 1999

[KS82] C R Kime and K K Saluja Test Scheduling in Testable VLSI Circuits In *Proceedings of The International Symposium of Fault-Tolerant Computers*, pages 406–412, Jun 1982

[KTH88] K Kim, J G Tront, and D D Ha Automatic Insertion of BIST Hardware Using VHDL In *Proceedings of The 25th IEEE Design Automation Conference*, pages 9–15, 1988

[KW98] G Kiefer and H-J Wunderlich Deterministic BIST with Multiple Scan Chains In *Proceedings of the IEEE International Test Conference*, pages 1057–1064, 1998

[LCH98] K-J Lee, J-J Chen, and C-K Huang Using a Single Input to Support Multiple Scan Chains In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 74–78, 1998

[LHW97] Y R Lin, T T Hwang, and A C H Wu Scheduling Techniques for Variable Voltage Low Power Designs *ACM Transactions on Design Automation of Electronic Systems*, 2(2) 81–97, Apr 1997

[LKL93] W J Lai, C P Kung, and C S Lin Test Time Reduction in Scan Designed Circuits In *Proceedings of EuroAsic Conference*, pages 489–493, 1993

[LNB91] S P Lin, C A Njinda, and M A Breuer A Systematic Approach for Designing Testable VLSI Circuits In *Proceedings of The IEEE Conference of Computer Aided Design, ICCAD'91*, pages 496–499, 1991

[LP99] E Larsson and Z Peng An Estimation-based Technique for Test Scheduling In *Proceedings of The International Conference on Electronic Circuits and Systems*, 1999

[LP00a] E Larsson and Z Peng A Technique for Test Infrastructure Design and Test Scheduling In *Proceedings of The Design and Diagnostics of Electronic Circuits and Systems Workshop*, 2000

[LP00b] E Larsson and Z Peng Test Infrastructure Design and Test Scheduling Optimization In *Proceedings of The IEEE European Test Conference*, 2000

[LS92] S Y Lee and K K Saluja An Algorithm to Reduce Test Application Time in Full Scan Designs In *Proceedings of The IEEE International Conference on Computer-Aided Design*, pages 17–20, 1992

[LWJA92] T-C Lee, W H Wolf, N K Jha, and J M Acken Behavioral Synthesis for Easy Testability in Data Path Allocation In *Proceedings of the International Conference on Computer Design*, pages 29–32, Oct 1992

[MA98] E J Marinissen and J Aerts Test Protocol Scheduling for Embedded-Core Based System ICs In *Proceedings of The IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 5 3–1 – 5 3–9, 1998

[MD] Discrete Mathematics and Algorithms Network (Dmanet) http //www zpr uni-koeln de/dmanet

[Mca98] E J Marinissen and et al A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores In *Proceedings of The International Test Conference*, 1998

[MGF+99] S Manich, A Gabarro, J Figueras, P Girard, L Guiller, C Landrault, S Pravossoudovitch, P Teixeira, and M Santos Low Power BIST by Filtering Non-Detecting Vectors In *Proceedings of The IEEE European Test Workshop*, pages 165–170, 1999

[MK96] R S Martin and J P Knight Optimizing Power In ASIC Behavioral Synthesis *IEEE Design and Test of Computers*, 13(2) 58–71, Summer 1996

[MKRT95] N Mukherjee, M Kassab, J Rajski, and J Tsyzer Arithmetic Built-In Self Test for High-Level Synthesis In *Proceedings of The 13th IEEE VLSI Test Symposium, VTS'95*, 1995

[ML99] E J Marinissen and M Lousberg The Role of Test Protocols in Testing Embedded-Core-Based Systems ICs In *Proceedings of The IEEE European Test Workshop*, 1999

[MM91] S P Morley and R A Marlett Selectable Length Partial Scan A Method to Reduce Vector Length In *Proceedings of The IEEE International Test Conference*, pages 408–411, 1991

[MTOM96] A Motohara, S Takeoka, M Ohta, and M Muraoka A Partial Scan Design Approach Based on Register-Transfer Level Testability Analysis *IEICE Transactions on Information and Systems*, E79-D(10) 1436–1442, Oct 1996

[MWMV00] V Muresan, X Wang, V Muresan, and M Vladutiu A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling In *Proceedings of IEEE Test Conference (ITC)*, pages 882–891, 2000

[NA99] N Nicolici and B M AlHashimi Efficient BIST Hardware Insertion with Low Test Application Time for Synthesized Data Paths In *Proceedings of The Design, Automation and Test in Europe Conference and Exhibition DATE'99*, page article 60, 1999

[NB93] S Narayanan and M A Breuer Reconfigurable Scan Chain A Novel Approach to Reduce Test Application Time In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 710–715, 1993

[NB95] S Narayanan and M A Breuer Reconfiguration Techniques for a Single Scan Chain *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(6) 750–765, 1995

[NGB92] S Narayanan, R Gupta, and M Breuer Configuring Multiple Scan Chains for Minimum Test Time In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 4–8, 1992

[NGB93] S Narayanan, R Gupta, and M A Breuer Optimal Configuring of Multiple Scan Chains *IEEE Transactions on Computers*, 42(9) 1121–1131, 1993

[NNB92] S Narayanan, C Njinda, and M Breuer Optimal Sequencing of Scan Registers In *Proceedings of The International Test Conference*, pages 293–302, 1992

[NNSaB94] L S Nielsen, C Niessen, J Sparso, and K an Berkel Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage *IEEE Transactions on VLSI Systems*, 2(4) 391–397, Dec 1994

[OBT91] S Oostdijk, F Beenker, and L Thijssen A Model for Test-Time Reduction of Scan Testable Circuits In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 243–252, 1991

[OH93] A Orailoglu and I G Harris Test Pattern Generation and Test Scheduling for Self-Testable Designs In *Proceedings of The International Conference on Computer Design*, pages 528–531, 1993

[OH97] A Orailoglu and I G Harris Microarchitectural synthesis for rapid BIST testing *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(6) 573–586, Jun 1997

[PCH91] C Papachristou, S Chiu, and H Harmanani SYNTEST A Method for High-level Synthesis with Self-Testability In *Proceedings of The International Conference on Computer Design, ICCD*, pages 45–62, 1991

[Ped96] M Pedram 'Tutorial and Survey Paper' Power Minimization in IC Design Principles and Applications *ACM Transactions on Design Automation of Electronic Systems*, 1(1) 3–56, Jan 1996

[PK87] P G Paulin and J P Knight Force-Directed Scheduling in Automatic Data Path Synthesis In *Proceedings of 24th Design Automation Conference*, pages 195–202, 1987

[PK89] P G Paulin and J P Knight Force-Directed Scheduling for the Behavioral Synthesis of ASICs *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6) 661–679, Jun 1989

[PS92] D K Pradhan and J Saxena A Design for Testability Scheme to Reduce Test Application Time in Full Scan In *Proceedings of The IEEE VLSI Test Symposium*, pages 55–59, 1992

[RS95] S Raje and M Sarrafzadeh Variable Voltage Scheduling In *Proceedings of The International Symposium on Low-Power Electronics and Design*, pages 9–14, 1995

[RVB96] N Ranganathan, N Vijaykrishnan, and N Bhavanishankar A VLSI Array Architecture with Dynamic Frequency Clocking In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 137–140, 1996

[RVB98] N Ranganathan, N Vijaykrishnan, and N Bhavanishankar A Linear Array Processor with Dynamic Frequency Clocking for Image Processing Applications *IEEE Transactions on Circuits and Systems for Video Technology*, 8(4) 435–445, 1998

[SDY98] M Sugihara, H Date, and H Yasuura A Novel Test Methodology for Core-based System LSIs and a Testing Time Minimization Problem In *Proceedings of The IEEE International Test Conference*, pages 465–472, 1998

[SK88] J Sayah and C R Kime Test Scheduling for High Performance VLSI System Implementations In *Proceedings of The International Test Conference*, pages 421–430, Oct 1988

[SK89] J Sayah and C R Kime Scheduling Unequal Length Tests in High Performance VLSI System Implementations In *Proceedings of The International Conference on Computer Design*, pages 566–570, Oct 1989

[SK92] J Sayah and C R Kime Test Scheduling in High Performance VLSI System Implementations *IEEE Transactions on Computers*, 41(1) 52–67, Jan 1992

[SM97] T M Storey and B McWilliam A Test Methodology for High Performance MCMs *Journal of Electronic Testing Theory and Applications*, 10 109–118, 1997

[SP94] M Srinivas and L M Patnaik Genetic Algorithms A survey *Computer*, 27(6) 17–26, 1994

[Str92] A P Stroele Self-Test Scheduling with Bounded Test Execution Time In *Proceedings of The International Test Conference*, pages 130–139, 1992

[SV97] V Schnecke and O Vornberger Hybrid Genetic Algorithms for Constrained Placement Problems *IEEE Transactions on Evolutionary Computation*, 1(4) 266–277, Nov 1997

[SW86] A P Stroele and H J Wunderlich Signature Analysis and Test Scheduling for Self-Testable Circuits In *Proceedings of The International Symposium on Circuits and Systems*, pages 1054–1057, 1986

[SW92] A P Stroele and H J Wunderlich Configuring Flip-Flops to BIST Registers In *Proceedings of The International Test Conference*, pages 130–139, 1992

[SW94] A P Stroele and H J Wunderlich A Unified Method for Assembling Global Test Schedules In *Proceedings of The Third Asian Test Symposium*, pages 268–273, 1994

[SW95] A P Stroele and H J Wunderlich Test Register Insertion with Minimum Hardware Cost In *Proceedings of The International Conference on Computer-Aided Design*, pages 95–101, 1995

[WD96] K D Wagner and S Dey High-Level Synthesis for Testability A Survey and Perspective In *Proceedings of The 33rd Design Automation Conference*, pages 131–136, 1996

[WG94] S Wang and S K Gupta ATPG for Heat Dissipation Minimization During Test Application In *Proceedings of The International Test Conference*, pages 250–258, 1994

[WG97a] S Wang and S K Gupta ATPG for Heat Dissipation Minimization During Scan Testing In *Proceedings of The 34th Design Automation Conference*, pages 614–619, 1997

[WG97b] S Wang and S K Gupta DS-LFSR A New BIST TPG for Low Heat Dissipation In *Proceedings of The IEEE International Test Conference*, pages 848–857, 1997

[WG98] S Wang and S K Gupta ATPG for Heat Dissipation Minimization During Test Application *IEEE Transactions on Computers*, 47(2) 256–262, Feb 1998

[WG99] S Wang and S K Gupta LT-RTPG A New Test-Per-Scan BIST TPG for Low Heat Dissipation In *Proceedings of The IEEE International Test Conference*, pages 85–94, 1999

[WH96] G Y Wei and M Horowitz A Low Power Switching Power Supply for Self-Clocked Systems In *Proceedings of The International Symposium on Low-Power Electronics and Design*, pages 313–317, 1996

[WI95] Y J Wu and A Ivanov Single-Reference Multiple Intermediate Signature (SREMIS) Analysis for BIST *IEEE Transactions on Computers*, 44(6) 817–825, Jun 1995

[WNML94] D L Wheater, P Nigh, J T Mechler, and L Lacroix ASIC Test Cost Strategy Trade-offs In *Proceedings of The IEEE International Test Conference*, 1994

[Xia94] D Xiang Test Scheduling Using Test Subsession Partitioning In *Proceedings of The International Conference on Computer Design*, pages 63–69, 1994

[ZB97] Y Zorian and H Bederr An Effective Multi-Chip BIST Scheme *Journal of Electronic Testing Theory and Applications*, 10 87–95, 1997

[ZMD98] Y Zorian, E J Marinissen, and S Dey Testing Embedded core-based system chips In *Proceedings of The IEEE International Test Conference*, 1998

[Zor90] Y Zorian A Structured Approach to Macrocell Testing using Built-In Self-Test In *Proceedings of the IEEE Conference on Custom Circuits Conference*, pages 28 3 1–28 3 4, 1990

[Zor93] Y Zorian A Distributed BIST Control Scheme for Complex VLSI Devices In *Proceedings of The 11th IEEE VLSI Test Symposium*, pages 4–9, Apr 1993

[Zor97] Y Zorian Test Requirements for Embedded Core-Based Systems and IEEE P1500 In *Proceedings of The IEEE International Test Conference, ITC'97*, pages 191–199, 1997

[Zor98] Y Zorian System-Chip Test Strategies In *Proceedings of the IEEE Design Automation Conference*, pages 752–757, 1998

[ZRB99] X Zhang, K Roy, and S Bhawmik POWERTEST A Tool for Energy Conscious Weighted Random Pattern Testing In *Proceedings of The IEEE International Conference on VLSI Design*, 1999

Appendix A Testbench Example

In this appendix two test set examples are detailed in order to give an idea of how the testbenches experimented in this thesis are structured Because the size of the file hosting a testable increases with the test resource compatibility degree, the first testbench example chosen is the low compatibility degree (first) experiment The second testbench example is taken from the literature The structures of the above mentioned testbenches are given below

The first testbench starts with general information about its parameters like the maximal and minimal possible power dissipation value for any of the block-tests, the maximal and minimal possible test length for a block-test, the compatibility degree percentage, the number of block-tests, their names, and the maximal and minimal power dissipation constraint values of the simulation range A step value is also given for the decremental step of the power dissipation constraint range After all this information the actual testbench content follows It has five fields for each block-test record the name of the block-test, its estimated power dissipation, its test application time (test length), the number of the block-tests compatible with it, and finally the list of other block-tests compatible with it A record is given for each block-test and they are saved in the file being sorted by two keys their test application time as the primary key to sort the list in a descending order, and their estimated power dissipation as the secondary key to sort the block tests with the same test application time in a descending order as well

//Low Compatibility Degree Testbench
//Generated with the following parameters
//PDmax = 20 (maximum power dissipation of a block test)
//PDmin = 1 (minimum power dissipation of a block test)
//TLmax = 25 (maximum test length of a block test)
//TLmin = 1 (minimum test length of a block test)
//CD = 10% (compatibility degree with the other block tests in the list)
//Number of block tests 50
//Block tests (BT) names
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20 t21 t22 t23 t24 t25
t26 t27 t28 t29 t30 t31 t32 t33 t34 t35 t36 t37 t38 t39 t40 t41 t42 t43 t44 t45 t46 t47 t48 t49 t50
//The Maximal Power Constraint (starting the simulation with) 200
//The Minimal Power Constraint (ending the simulation with) 20
//The Power Constraint Step 20

BT Name	BT PD	BT TL	Comp BT Nb	BT Compatibility List
t1	20	25	4	t3 t13 t31 t34
t2	17	25	3	t8 t38 t44
t3	11	25	6	t1 t7 t12 t19 t27 t48
t4	8	25	4	t6 t22 t38 t41
t5	4	25	6	t9 t13 t17 t29 t34 t47
t6	2	25	4	t4 t23 t37 t44
t7	18	23	5	t3 t9 t13 t17 t50
t8	16	23	6	t2 t22 t30 t34 t40 t43
t9	9	23	5	t5 t7 t11 t19 t27
t10	3	23	2	t37 t48
t11	19	22	5	t9 t16 t24 t30 t43
t12	13	22	6	t3 t13 t18 t28 t38 t48
t13	5	22	7	t1 t5 t7 t12 t14 t23 t33
t14	1	22	4	t13 t27 t40 t44
t15	12	21	3	t30 t46 t50
t16	14	20	4	t11 t17 t32 t37
t17	9	19	6	t5 t7 t16 t19 t23 t42
t18	6	19	5	t12 t29 t34 t45 t48
t19	1	18	6	t3 t9 t17 t21 t25 t39
t20	12	17	2	t32 t50
t21	6	17	3	t19 t36 t49
t22	11	16	5	t4 t8 t27 t42 t46
t23	3	16	8	t6 t13 t17 t29 t33 t39 t44 t48
t24	8	15	4	t11 t25 t36 t42
t25	10	14	2	t19 t24
t26	2	14	4	t33 t38 t45 t48
t27	5	13	5	t3 t9 t14 t22 t29
t28	2	13	5	t12 t36 t39 t43 t45
t29	9	12	6	t5 t18 t23 t27 t33 t50
t30	7	11	5	t8 t11 t15 t31 t42
t31	5	11	4	t1 t30 t43 t47
t32	4	10	3	t16 t20 t38
t33	14	9	7	t13 t23 t26 t29 t35 t43 t48
t34	8	9	4	t1 t5 t8 t18
t35	18	8	3	t33 t37 t46
t36	11	8	5	t21 t24 t28 t40 t47
t37	8	8	6	t6 t10 t16 t35 t44 t50
t38	3	8	6	t2 t4 t12 t26 t32 t48
t39	1	8	5	t19 t23 t28 t41 t43
t40	12	7	4	t8 t14 t36 t46
t41	4	7	2	t4 t39
t42	4	7	6	t17 t22 t24 t30 t44 t49
t43	11	6	7	t8 t11 t28 t31 t33 t39 t47
t44	5	6	7	t2 t6 t14 t23 t37 t42 t50
t45	12	5	4	t18 t26 t28 t46
t46	8	5	6	t1 t22 t35 t40 t45 t48
t47	3	5	5	t5 t31 t36 t43 t49
t48	9	4	10	t3 t10 t12 t18 t23 t26 t33 t38 t46 t50
t49	8	3	3	t21 t42 t47
t50	4	2	7	t7 t15 t20 t29 t37 t44 t48

The ASIC-Z testbench is also given below This testbench has been written and used for the first time in [LP00b]

```
// Modified ASIC Z design
// Each core is tested by an
// external test a BIST test
// There is also one interconnection
// test to a neighbor for each core
//
// Max power(mW) 900
```

core	placement x	placement y	test name	idle power	test power	test time	test gen	test analy	core constr
TG1	0	0	NONE						
ROM1	0	10	t1i	0	10	10	TC	TC	ROM2
ROM1	0	10	t1e	11	140	50	TC	TC	NONE
ROM1	0	10	t1b	11	140	50	TG1	SA1	NONE
RAM2	0	20	t2i	0	10	10	TC	TC	ROM1
RAM2	0	20	t2e	9	120	30	TC	TC	NONE
RAM2	0	20	t2b	9	120	30	TG2	SA2	NONE
SA1	0	30	NONE						
TG2	10	0	NONE						
ROM2	10	10	t3i	0	10	10	TC	TC	RAM4
ROM2	10	10	t3e	11	140	50	TC	TC	NONE
ROM2	10	10	t3b	11	140	50	TG1	SA1	NONE
RAM3	10	20	t4i	0	10	10	TC	TC	RAM2
RAM3	10	20	t4e	6	106	20	TC	TC	NONE
RAM3	10	20	t4b	6	106	20	TG2	SA2	NONE
SA2	10	30	NONE						
TG4	20	0	NONE						
RAM4	20	10	t5i	0	10	10	TC	TC	RAM1
RAM4	20	10	t5e	4	48	11	TC	TC	NONE
RAM4	20	10	t5b	4	48	11	TG2	SA2	NONE
RL2	20	20	t6i	0	10	10	TC	TC	RAM3
RL2	20	20	t6e	0	176	80	TC	TC	NONE
RL2	20	20	t6b	0	176	80	TG4	SA4	NONE
SA4	20	30	NONE						
TG5	30	0	NONE						
RAM1	30	10	t7i	0	10	10	TC	TC	RF
RAM1	30	10	t7e	10	141	35	TC	TC	NONE
RAM1	30	10	t7b	10	141	35	TG2	SA2	NONE
RL1	30	20	t8i	0	10	10	TC	TC	RL2
RL1	30	20	t8e	0	148	70	TC	TC	NONE
RL1	30	20	t8b	0	148	70	TG5	SA5	NONE
SA5	30	30	NONE						
TG3	40	0	NONE						
RF	40	10	t9i	0	10	10	TC	TC	RL1
RF	40	10	t9e	10	48	5	TC	TC	NONE
RF	40	10	t9b	10	48	5	TG3	SA3	NONE
TC	40	20	NONE						
SA3	40	30	NONE						

Translating it into the format used to run the experiments in this thesis, the result would be

```
//Modified ASIC testbench
//Number of block tests  27
//Block tests  names
t6e t6b t8e t8b t1e t1b t3e t3b t7e t7b t2e t2b t4e t4b t5e t5b t1i t2i t3i t4i t5i t6i t7i t8i t9i t9e t9b
```

BT Name	BT PD	BT TL	CBT Nb	BT Compatibility List
t6e	176	80	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t6b	176	80	26	t1i t1e t1b t2i t2e t2b t3i t3e t3b t4i t4e t4b t5i t5e t5b t6i t6e t7i t7e t7b t8i t8e t8b t9i t9e t9b
t8e	148	70	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t8b	148	70	26	t1i t1e t1b t2i t2e t2b t3i t3e t3b t4i t4e t4b t5i t5e t5b t6i t6e t6b t7i t7e t7b t8i t8e t9i t9e t9b
t1e	140	50	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t1b	140	50	25	t1i t1e t2i t2e t2b t3i t3e t4i t4e t4b t5i t5e t5b t6i t6e t6b t7i t7e t7b t8i t8e t8b t9i t9e t9b
t3e	140	50	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t3b	140	50	25	t1i t1e t2i t2e t2b t3i t3e t4i t4e t4b t5i t5e t5b t6i t6e t6b t7i t7e t7b t8i t8e t8b t9i t9e t9b
t7e	141	35	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t7b	141	35	23	t1i t1e t1b t2i t2e t3i t3e t3b t4i t4e t5i t5e t6i t6e t6b t7i t7e t8i t8e t8b t9i t9e t9b
t2e	120	30	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t2b	120	30	23	t1i t1e t1b t2i t2e t3i t3e t3b t4i t4e t5i t5e t6i t6e t6b t7i t7e t8i t8e t8b t9i t9e t9b
t4e	106	20	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t4b	106	20	23	t1i t1e t1b t2i t2e t3i t3e t3b t4i t4e t5i t5e t6i t6e t6b t7i t7e t8i t8e t8b t9i t9e t9b
t5e	48	11	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t5b	48	11	23	t1i t1e t1b t2i t2e t3i t3e t3b t4i t4e t5i t5e t6i t6e t6b t7i t7e t8i t8e t8b t9i t9e t9b
t1i	10	10	8	t1b t2b t4b t5b t6b t7b t8b t9b
t2i	10	10	8	t2b t3b t4b t5b t6b t7b t8b t9b
t3i	10	10	8	t1b t2b t3b t4b t6b t7b t8b t9b
t4i	10	10	8	t1b t3b t4b t5b t6b t7b t8b t9b
t5i	10	10	8	t1b t2b t3b t4b t5b t6b t8b t9b
t6i	10	10	8	t1b t2b t3b t5b t6b t7b t8b t9b
t7i	10	10	8	t1b t2b t3b t4b t5b t6b t7b t8b
t8i	10	10	8	t1b t2b t3b t4b t5b t7b t8b t9b
t9i	10	10	8	t1b t2b t3b t4b t5b t6b t7b t9b
t9e	48	5	9	t1b t2b t3b t4b t5b t6b t7b t8b t9b
t9b	48	5	26	t1i t1e t1b t2i t2e t2b t3i t3e t3b t4i t4e t4b t5i t5e t5b t6i t6e t6b t7i t7e t7b t8i t8e t8b t9i t9e

Appendix B· Publications

V Muresan, D Crisu and X Wang, From VHDL to FPGA - A Case Study of a Fuzzy Logic Controller, Proceedings of The International Conference of Young Lecturers and PhD Students, Miskolc, Hungary, August 11-17, 1997, Section Proceeding, Engineering Science II, pp 83-90

V Muresan, N Nicolici and X Wang, ASIC Design of an ATM Switch Featuring Multichannel Bandwidth Allocation, Proceedings of The Second International Conference and Exhibition on Information Infrastructure - Information Super Highway (ICEII'98), Beijing, China, April 26-29, 1998

V Muresan, X Wang and J J Yan, ASIC Design of a Fuzzy Logic Controller, Proceedings of The IASTED International Conference on Control and Applications (CA'98), August 12-14, 1998, Hawaii, USA, pp 93-97, ISBN 0-88986-272-9

V Muresan, V Muresan, X Wang and M Vladutiu, Design and Implementation of a Didactic Test System for FPGA Devices, Special issue of Transactions on Automatic Control and Computer Science, dedicated to the Third International Conference on Technical Informatics (CONTI'98), "Pohtchnica" University of Timisoara, Timisoara Romania, October, 1998, pp 206-215, ISBN 1224-600X

V Muresan, X Wang, V Muresan, M Vladutiu, Distribution-Graph Based Approach and Extended Tree Growing Technique in Power-Constrained Block-Test Scheduling, Proceedings IEEE Ninth Asian Test Symposium (ATS'00), Dec 4-6, 2000, Taipei, Taiwan, pp 465-470, ISBN 0-7695-0887-1, IEEE Computer Society Order Number PR00887

V Muresan, X Wang, V Muresan, M Vladutiu, A Comparison of Classical

Scheduling Approaches in Power-Constrained Block-Test Scheduling, Proceedings IEEE International Test Conference (ITC'00), Atlantic City, USA, October 3-5, 2000, pp 882-891, IEEE Cat No 00CH37159

V Muresan, X Wang, V Muresan, M Vladutiu, Power-Constrained Block-Test List Scheduling, Proceedings 11th International Workshop on Rapid System Prototyping (RSP'00) Shortening the Path from Specification to Prototype, Paris, France, June 21-23, 2000, pp 182-187, ISBN 0-7695-0668-2, Cat No PR00668

V Muresan, X Wang, V Muresan, M Vladutiu, The Left Edge Algorithm in Block Test Scheduling under Power Constraints, Proceedings 2000 IEEE International Symposium on Circuits and Systems (ISCAS'00), May 28-31, 2000, Geneva, Switzerland, pp 351-354, IEEE Cat No 00CH36353

V Muresan, X Wang, V Muresan, M Vladutiu, List Scheduling and Tree Growing Technique in Power-Constrained Block-Test Scheduling, Proceedings of the IEEE European Test Workshop (ETW'00), May 23-26, 2000, Cascais, Portugal, pp 27-32

V Muresan, X Wang, V Muresan, M Vladutiu, The Left Edge Algorithm and the Tree Growing Technique in Block Test Scheduling under Power Constraints, Proceedings 18th IEEE VLSI Test Symposium (VTS'00), April 30 - May 4, 2000, Montreal, Canada, pp 417-422, ISBN 0-7695-0613-5

T Khadir, V Muresan, M Collier, X Wang, The VHDL Model Of A Three Stage ATM Switch Featuring A Cell-Level Path Allocation Algorithm, Proceedings IASTED International Conference on Applied Informatics (AI'01), Innsbruck, Austria, February 19-22, 2001, pp 235-241

V Muresan, X Wang, V Muresan, M Vladutiu, A Combined Tree Growing Technique for Block-Test Scheduling under Power Constraints, Proceedings 2001 IEEE International Symposium on Circuits and Systems (ISCAS'01), May 6-9, 2001, Sydney, Australia, pp $V_2 55 - V_2 58$, IEEE Cat No 01CH37196C

V Muresan, X Wang, V Muresan, M Vladutiu, Mixed Classical Scheduling Algorithms and Tree Growing Technique in Block-Test Scheduling under Power Constraint, Proceedings 12th International Workshop on Rapid System Prototyping (RSP'01) - Shortening the Path from Specification to Prototype, Monterey, California, USA, June 25-27, 2001, pp 162-167, ISBN 0-7695-1206-2, Cat No PR00668

V Muresan, X Wang, V Muresan, M Vladutiu, Greedy Tree Growing Heuristics on Block-Test Scheduling under Power Constraints, invited paper, to be published in VLSI Design - International Journal of Custom-Chip Design, Simulation, and Testing