Optimization of Test and Design-for-Testability Solutions for Many-Core System-on-Chip Designs

by

Mukesh Agrawal

Department of Electrical and Computer Engineering
Duke University

Date: ________________________
Approved:

Krishnendu Chakrabarty, Supervisor

John A. Board Jr.

Benjamin C. Lee

Montek Singh

Kishor A. Trivedi

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering in the Graduate School of Duke University
2014

Abstract

Optimization of Test and Design-for-Testability Solutions for Many-Core System-on-Chip Designs

by

Mukesh Agrawal

Department of Electrical and Computer Engineering
Duke University

Date: ____________________________
Approved:

____
Krishnendu Chakrabarty, Supervisor

____
John A. Board Jr.

____
Benjamin C. Lee

____
Montek Singh

____
Kishor A. Trivedi

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering
in the Graduate School of Duke University
2014

Abstract

With the continuous scaling of transistors to smaller dimensions, it has now become feasible to pack billions of transistors in a single chip. However, interconnect does not scale as well as transistors; hence a significant amount of research is focused today on finding viable alternatives to bus-based interconnects. Coupled with the problem of increasing interconnect delay is the challenge to contain the power footprint. A network-on-chip (NOC) interconnect fabric can alleviate many of these problems, and is therefore viewed as a promising interconnect paradigm of the future. In addition, NOC building blocks consist of reusable components and require minimal design effort in building a large and complex system. In addition to NOCs, 3D integration technology using through-silicon-vias (TSVs) can also alleviate the problems caused by long global interconnects and it offers more innovative design choices compared to traditional 2D integrated circuits (ICs).

Regardless of the choice of interconnect fabric, testing continues to pose a significant challenge. New optimization methods for designing the test-access mechanism (TAM) and minimizing test time are needed. On-chip routing protocols guided by network topology and traffic congestion constraints open new avenues for research in optimizing test-data delivery in NOCs. The thesis describes algorithms for test-data delivery optimization in NOC designs with hundreds of cores, where the NOC is used as a TAM. First, an algorithm based on subset-sum formulation to solve the test-delivery problem in NOCs with arbitrary topology is presented. For the important

class of NOCs with a grid topology, the optimization problem is modeled as an NOC partitioning problem and solved using dynamic programming in pseudo-polynomial time. Both the proposed methods yield high-quality results and are scalable to large SOCs with many cores.

Since the proposed methods involve concurrent testing of multiple cores, high power consumption may limit its adoptability in practice; therefore, the thesis also discusses a scheduling algorithm under power constraints using the same dynamic-programming framework. By leveraging the capability of modern multicast routers, the homogeneity of cores is exploited to further minimize test time in such large SOCs.

Three-dimensional (3D) stacking of ICs using through-silicon-vias (TSVs) is a promising integration platform for next-generation ICs. Several complex manufacturing steps involved in the fabrication of 3D-stacked ICs make these chips susceptible to defects. In the scenario where die yields are low, stacking of untested dies further reduces overall yield, and thereby reduces profitability. Moreover, testing at every stage of 3D integration may result in prohibitive test cost. Despite the benefits offered by 3D integration, test cost remains a major concern. The tests have to be inserted at appropriate places in the 3D integration flow. Automated tools are needed to analyze and understand test flows and achieve the desired goal. The thesis describes a generic cost model to account for various test costs involved in 3D integration and formalize the study of test flows as a search problem to minimize the overall cost. An algorithm based on A* to obtain an optimal test flow is presented. Adopting a formal approach to solving the cost-minimization problem provides useful insights that cannot be derived via selective enumeration of a smaller number of candidate test flows.

In 3D-stacked ICs, inaccessibility of TSVs prior to bonding makes it difficult to test the combinational logic between scan flip-flops and TSVs at a pre-bond stage.

Addition of wrapper cells on both ends of a TSV solves the testability problem, but at the cost of area overhead, increased latency and performance degradation. The thesis builds on prior work to reuse existing scan flip flops and shows that the general problem of minimizing the wrapper cells is equivalent to the graph-theoretic minimum clique-partitioning problem, and is therefore NP-hard. Efficient heuristic methods are adopted to solve the problem. The proposed methods incorporate a timing-guided and layout-aware approach to address practical timing considerations that were overlooked in prior work. Extensions are also made to post-bond testing stages.

Finally, the thesis describes an end-to-end design of a built-in self-test (BIST) infrastructure for 3D-stacked ICs that facilitates the use of BIST at multiple stages of 3D integration. The proposed BIST design is distributed, reusable, and reconfigurable, hence it is attractive for both pre-bond and post-bond testing. Architectural support for incorporating a static BIST schedule is also provided. Furthermore, two algorithms based on 2D bin packing problem to minimize test time under BIST-resource and power constraints are presented.

In summary, the thesis targets important optimization problems related to test-delivery in manycore SOCs that are assembled using a scalable and flexible integration platform. The proposed research has led to theoretical insights, simulation results, and a set of test and design-for-test methods to make testing effective and feasible from a cost perspective.

To my mother and father, Meera and Rajkishore, for always believing in me. And my wife, Supriya, for her endless love and patience.

Contents

Abstract **iv**

List of Tables **xiii**

List of Figures **xvi**

Acknowledgements **xxi**

1 Introduction **1**

1.1 Basics of Circuit Testing 2

1.1.1 Categories of Testing 2

1.2 Design for Testability (DfT) 3

1.2.1 Scan Design 3

1.2.2 Modular Testing Using Test Wrappers and Test-Access Mechanism (TAM) 6

1.2.3 Built-In Self-Test (BIST) 8

1.3 Network-on-chip 9

1.3.1 Basics of an NOC 10

1.3.2 NOC Testing 12

1.4 3D Integration Technology 13

1.5 Outline of Dissertation 18

2 Test Scheduling Algorithms for Network-on-Chip **21**

2.1 Generic Problem Statement and Prior Work 22

2.1.1 Formal problem statement 22

2.1.2 Related prior work 23
2.1.3 Chapter contributions 25
2.2 Test Scheduling for NOCs with Dedicated Routing 26
2.2.1 Problem description 26
2.2.2 Outline of the algorithm 27
2.2.3 Subset-sum problem (SSP) 28
2.2.4 Optimal pin distribution for a partition 29
2.2.5 Enumeration of candidate partitions 31
2.2.6 Application to NOC 33
2.3 Dynamic Programming Solution for a Grid Topology 34
2.3.1 Preliminaries 34
2.3.2 Problem formulation 35
2.3.3 Characterization of the optimal solution structure 39
2.3.4 Algorithm 42
2.3.5 Constructing the solution 44
2.3.6 Computational complexity analysis 44
2.4 Additional Constraints and Enhancements 45
2.4.1 Boundary-region constraint 45
2.4.2 Pin-assignment problem 46
2.4.3 Power constraints 47
2.4.4 Reducing the run-time complexity by a factor of P 52
2.5 Test Scheduling using Multicast Routing 53
2.5.1 Routing in NOCs using multicast routers 54
2.5.2 Problem formulation 55
2.5.3 ILP Formulation 57

2.5.4 Dynamic programming approach for rectangular partitioning . 61
2.6 Experimental Results . . . 61
2.6.1 Test scheduling without power constraints . . . 62
2.6.2 Power-constrained test scheduling . . . 74
2.6.3 Speedup technique . . . 75
2.7 Conclusion . . . 76

3 Test-Cost Modeling and Optimal Test-Flow Selection of 3D-Stacked ICs 77

3.1 Motivation and Chapter Contributions . . . 78
3.2 Cost Model . . . 81
3.3 Approximation to the Cost Model . . . 89
3.3.1 Model I . . . 89
3.3.2 Model II . . . 90
3.4 Problem Formulation . . . 91
3.5 Solution Based on A* Search Algorithm . . . 93
3.5.1 Computation of $h(n)$ . . . 94
3.5.2 Adaptation of A* to the cost-minimization problem . . . 95
3.5.3 $(1-\Delta)^{-1}$-approximation step . . . 97
3.6 Results . . . 98
3.7 Conclusion . . . 109
3.8 Appendix . . . 110
3.8.1 Proof for the complexity of the selection of test flow . . . 110
3.8.2 An Example for a two-die stack . . . 111
3.8.3 User interface for the test-flow-selection tool . . . 113

4 Reuse-Based Optimization for Pre-Bond and Post-Bond Testing of 3D-stacked ICs 122

4.1 Motivation and Prior Work 124
4.1.1 Fault coverage without wrapper cells 126
4.1.2 Related Prior Work 128
4.2 Problem Description 131
4.2.1 Selection criteria for flops 132
4.2.2 Wrapper-cell-count minimization (WCM) problem 133
4.2.3 Clique-partitioning problem 134
4.2.4 Polynomial-time reduction from CPP 134
4.3 Design Enhancement to Support Die-level Modular Testing 138
4.4 Heuristic Methods and ILP Model 142
4.4.1 OCAT Algorithm 142
4.4.2 MCAT Algorithm 144
4.4.3 Incorporating layout information 144
4.4.4 Incorporating timing constraints 145
4.4.5 Limiting the reuse of flops 146
4.4.6 ILP formulation for CPP 147
4.5 Results 150
4.5.1 Results on wrapper insertion and scan flop reuse 150
4.5.2 ATPG results 153
4.5.3 Comparison with ILP model 155
4.5.4 Layout and timing-aware assignment of flops to TSVs 156
4.6 Conclusion 160

5 A Distributed, Reconfigurable, and Reusable BIST Infrastructure for 3D-Stacked ICs 161

5.1 Motivation and Chapter Contributions 162
5.2 Related Prior work 164

5.3 BIST Framework 166
5.3.1 BIST instruction format 167
5.3.2 Instruction management 169
5.3.3 Relationship to test scheduling 171
5.3.4 Die-level controller 173
5.3.5 Resource-level controller and core-level controller 178
5.4 Test Scheduling under Power Constraints 179
5.4.1 Problem formulation 180
5.4.2 Skyline algorithm 182
5.4.3 Guillotine-split method 183
5.5 Results 185
5.6 Conclusion 191
6 Dissertation Summary and Future Directions **192**
6.1 Dissertation Summary 192
6.2 Future Directions 194
Bibliography **197**
Biography **206**

List of Tables

2.1 Comparison of the test times (clock cycles) obtained using the proposed subset-sum based method with [1] for NOCs having any arbitrary topology. 64

2.2 Results for t512505 (31 cores) for various values of K and P. 65

2.3 Results for p93791 (32 cores) for various values of K and P. 66

2.4 Comparison of test times (in cycles) for p93791 obtained from different methods with the tam-independent lower bound reported in [2]. . . . 67

2.5 Test times for 14x14 NOC for various values of K and P. 68

2.6 Test times for 20x20 NOC for various values of K and P. 69

2.7 Test time of dynamic programming vs. randomized baseline approach for 32x31 NOC, $P = 150$ 70

2.8 Table showing diminishing improvement in the reduction in test time as K increases for a 14x14 NOC. 70

2.9 Test times (in cycles) of a 10x10 NOC for multiple access points for a range of pin counts. 72

2.10 Test times (in cycles) of a 35x35 NOC for multiple access points for a range of pin counts. 73

2.11 Test times of p93791 for four access points under power constraints. . 75

2.12 Test times of a 10x10 NOC for multiple access points for 120 pins under power constraint. 76

2.13 CPU times for the speedup technique discussed in Section 2.4. 76

3.1 Table describing the given parameters used in the cost model. 82

3.2 Table describing the unknown variables used in the cost model. . . . 83

3.3 Input parameters for an example of 2-die stack. 99

3.4 Input parameters for an example of 4-die stack. 100

3.5 Input parameters for an example of four-die stack. 102

3.6 Selection of test insertion and tests for $\mathcal{D}_2$ on varying die yield. 104

3.7 Selection of test insertion and tests for $\mathcal{D}_1$ on varying test cost. 105

3.8 Number of nodes explored using various methods. 107

3.9 CPU times for various methods for the example of a four-die stack. . 107

3.10 Illustration of the cost model through an example of a two-die stack. 112

3.10 (continued) 113

4.1 Percentage of combinational logic present before the first level and after the last level of scan cells for the OpenCore [3] benchmark circuits. 126

4.2 Stuck-at and transition fault coverage for designs from OpenCore [3] with and without wrapper cells for TSVs. 126

4.3 Stuck-at and transition fault coverage for a 4-die implementation of *cf_rca_16* from OpenCore with and without wrapper cells for TSVs. . 127

4.4 Comparison of IEEE Std. 1500 wrapper cell with the proposed DfT architecture 140

4.5 Design details of two-die implementation of the OpenCore benchmarks used in experiments. 149

4.6 Design details of four-die implementation of the OpenCore benchmarks used in experiments. 149

4.7 Design details for the ITC'99 benchmarks used in experiments. 149

4.8 The pair (N_F, N_{WC}) for different designs and for different values of (p, q). 152

4.9 Mean and standard deviation of (N_F, N_{WC}) with randomly selected reuse candidates for various designs. 153

4.10 Stuck-At Fault Coverage (FC) and test pattern count (PC) for various methods for the benchmark circuits. 154

4.11 Transition Fault Coverage (FC) and test pattern count (PC) for various methods for the ITC'99 benchmark circuits. 155

4.12 Table comparing N_{WC} from different methods. 155

4.13 Table showing the effect of varying the layout threshold on N_F, N_{WC}, stuck-at fault coverage (FC), pattern count (PC) of individual dies for various designs. The number of edges in the corresponding CPP instance is also shown. 157

4.14 Table highlighting the effect of adding modular testing support to various designs. 158

4.15 Table comparing the area (in square microns) overhead on various designs with a dedicated wrapper for each TSV and with reuse-based method 159

5.1 Pattern count for achieving 98% fault coverage for various designs . . 186

5.2 Area of the synthesized netlists (in square microns) with and without BIST 186

5.3 A running example of tests with test length (TL) and power consumption (PW) values from [4] 189

5.4 Test length (in s) and power consumption (in mW) for Design 3 at 500 MHz clock frequency 189

5.5 Test times for pre-bond and post-bond tests using the two scheduling algorithms (see Table 5.3 and table 5.4 for units.) 189

5.6 Test times (A.U.) for the example from [4] under resource constraints. 190

5.7 Test times (in s) for testing the stack of Design 3 under resource constraints. 190

List of Figures

1.1 (a) Using a multiplexer to convert a flip-flop to a scan cell. (b) Multiple scan cells are stitched to form a scan chain. 4

1.2 Illustrating two different modes of a scan design. The active signals in each mode are highlighted. 5

1.3 A test wrapper based on IEEE 1500 standard [5]. 6

1.4 Components of a typical BIST system. 9

1.5 Illustration of a grid topology in an NOC-based SOC. 11

1.6 Two different stacking orientation of dies during 3D ICs. 15

2.1 Pseudocode for the procedure `SolveSubsetSum`. 28

2.2 Method for distributing P pins given a partition of set of cores. . . . 29

2.3 Pseudo code for the procedure `optimizeTAM`. 32

2.4 A figure showing the partitioning of a 4x4 NOC for parallel testing. . 35

2.5 An illustration of the impact of optimal partitioning on testing time. 36

2.6 Two examples of inadmissible partitioning. 36

2.7 Mapping of a partition to a binary tree (l_1, l_2, l_3, and l_4 are separators). 37

2.8 (a) Figure showing a position of a separator l_1 on the grid. (b) & (c) Figure showing two configurations of a rectangle $\mathbf{R}$ after a separator is placed. 39

2.9 The top-level dynamic programming procedure for partitioning. . . . 42

2.10 `ComputeAndStore` Procedure 43

2.11 `ConstructSolution` Procedure 43

2.12 `createPowerProfile` procedure. 48

2.13 Merging of power profiles. 49

2.14 Transformation of a four-peak profile to a profile having two peaks. . 49

2.15 Another example of profile transformation. 50

2.16 Modified `computeAndStore` for reducing the run-time complexity. . . 53

2.17 An illustration of (a) tree-based multicast, and (b) path-based multicast. 55

2.18 A possible partition on the set of links (a) using CFA; (b) using rectangular partitioning. 57

2.19 Results showing percentage reduction in test time by the subset-sum-based method over DP-based method with varying values of Δ for 32x31 NOC, $K = 4$. CPU time for each value of Δ is also shown. . . 71

3.1 3D integration of a stack and the associated test insertions. 80

3.2 Variation of fraction of components passing a test on fault coverage according to the William and Brown model. 89

3.3 A partial state space for an example of two-die stack. 92

3.4 A basic version of the A* algorithm. 95

3.5 A $(1-\Delta)^{-1}$-approximation algorithm. 97

3.6 Comparing our objective function with the objection function in [6] for a two-die stack. 99

3.7 Comparing our objective function with the objection function in [6] for a four-die stack. 101

3.8 Zooming into the highlighted region in Figure 3.7. The test flows in this region have contrasting values of the objective functions. The test flow that has a near-optimal value for the objective function (1.12 times the optimum value) in this work has the worst value for the objective function in [6]. 103

3.9 Comparing test selection at two insertions of $\mathcal{D}_2$. 106

3.10 Main window of the UI. 114

3.11 State of the UI after two dies have been added. 115

3.12 State of the UI after selecting a die on the left tree control. 116

3.13 State of the UI after selecting a test. 117

3.14 State of the UI after selecting a node corresponding to a stack. 118

3.15 Using the custom method to set bond-yield factors. 119

3.16 Editing bond-yield factor of dies individually. 120

3.17 Solution window of the UI. 121

4.1 Different approaches of partitioning [7]. 125

4.2 IEEE Standard wrapper cells [8]. 127

4.3 Illustration of how an existing flop can be reused for an inbound TSV: (a) addition of wrapper cell; (b) flop reuse. 129

4.4 Example showing an untestable s-a-1 fault at line A due to overlapping fan-out. 129

4.5 Using a wrapper cell in (a) makes the logic connected to outbound TSV observable. Using a 2-input XOR gate in (b) makes all single faults on its input observable by an existing scan flop. 131

4.6 Illustration for Lemma 4.2.1: reusing an additional wrapper cell for multiple TSVs. 133

4.7 Pseudocode for a deterministic labeling procedure. 135

4.8 Illustrating a one-to-one mapping from a general instance of CPP to an instance of WCM. 137

4.9 An example of a clique partition on a labeled graph. 137

4.10 A typical IEEE Std. 1500 wrapper cell. 138

4.11 Reusing an existing flop for both controlling the fan-out cone of an inbound TSV and observing the value on the TSV. FI and FO are fan-in and fan-out cone of the scan flop, respectively. 139

4.12 Reusing an existing flop for multiple inbound TSVs. FI and FO are fan-in and fan-out cone of the scan flop, respectively. 140

4.13 Reusing an existing flop for multiple outbound TSVs. 141

4.14 DfT enhancement for a bidirectional TSV. 142

4.15 Procedure *contract*. 142

4.16 Pseuocode description of the OCAT algorithm. 143

4.17 An example of edge contraction on edge (p,q). 143

4.18 Pseudocode description of MCAT. 144

4.19 Illustrating increased capacitive load on a scan flip-flop and delay on functional paths due to reuse. 146

4.20 A zoomed-in region of Die 0 of rca_cfa_16. TSVs are highlighted with small white squares. 156

5.1 (a) BIST framework from [9]; (b) Examples of correct and incorrect program order for the given test schedule using BIST instructions from [9]. 166

5.2 Overview of the BIST architecture. 167

5.3 An example of a 24-bit BIST control instruction format for a four-die stack with 16 BIST resources on each die. 168

5.4 (a) The instruction buffer management module; (b) instruction buffer with multiple slots for storing instructions. 171

5.5 Broadcast mechanism for the completed instructions. 172

5.6 Arbiters in adjacent dies and their interconnections. 173

5.7 An example of broadcasting `IID` from *Die n* to the `CTBS` of *Die n+1*. 'Z' refers to high-impedance state. 174

5.8 Figures showing (a) cascaded error registers, and (b) interconnections between die-level controllers in adjacent dies. 175

5.9 State machine for a die-level controller. 176

5.10 Timing diagram for shifting out the cascaded error registers. Before the shift sequence begins, all die-level controllers are in `TestFinishedAbove` state, which indicates that all tests have finished execution. 177

5.11 State machine for a resource-level controller. 178

5.12 (a) A hole and a skyline. There are 6 tests in this example—t_1 through t_6. (b) Free rectangles after the placement of a rectangle in the initial bin using the guillotine-split method. Two configurations are possible based on orientation of the split. 182

5.13 Illustration of the skyline and the guillotine-split methods under resource constraints. 183

5.14 Figures showing (a) layout for our experimental 3D-stacked IC designs, and (b) cores used in each design. 185

5.15 Timing simulation for a resource-level controller. 187

5.16 Timing simulation for the buffer management module. 188

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Professor Krishnendu Chakrabarty for his constant support and tremendous patience through out my Ph.D study. Not only did I find him extremely approachable whenever I was unable to make progress in my research, I also found him very friendly and open to discuss any problem that concerned me, despite his tight schedule. This dissertation would not have been possible without his excellent mentorship.

I would also like to thank the rest of my thesis committee: Prof. John A. Board Jr., Professor Benjamin C. Lee, Professor Kishor S. Trivedi, and Professor Montek Singh for their encouragement, insightful comments and questions.

I also wish to thank my friend Michael Z. Richter, who is also a co-author of our paper on minimzing test time of NOC-Based SOCs, for our stimulating discussions on various topics during his stay at Duke University and for our research collaboration after he left Duke.

I thank my colleagues and officemates from Duke University: Dr. Yan Luo, Dr. Brandon Noia, Dr. Qing Duan, Dr. Fangming Ye, Ran Wang, Kai Hu, and Zipeng Li, for all the fun we had together as graduate students. My other friends from Duke: Simone Wies, Sergej Deutsch (also my long-time roommate and office mate), Sarada, Prateek Jaipuria, Phanindhar Shivapooja, Akshay Raut, and a whole group of my friends from India, made my stay in Durham a very memorable one. I further wish to express my appreciation to my friends from my undergraduate days: Abhishek, Rakesh, Ashish, Deepak, Sankalp, Tathagata, Amar, Shaunak, and many others who gave me useful advices on a range of issues during the course of my doctoral study.

Last but not the least, I would like to thank my family: my parents, my parents-in-law, my siblings—Anand and Rupam, and my wife and my soul mate, Supriya; without their limitless love and support, I would never have reached this important milestone in my career.

1

Introduction

With the semiconductor industry continuously pursuing smaller transistor dimensions, device scaling is expected to soon reach fundamental limits. Global interconnects are a major barrier towards greater integration in the design of manycore chips. Interconnects do not scale at the same rate as transistors for smaller technology nodes, and long global interconnects lead to performance degradation because of increased delay and power consumption.

To overcome this bottleneck, the semiconductor industry is looking for alternatives to bus-based interconnects. A promising solution is a network-on-chip (NOC). Since NOC offers smaller interconnects, higher bandwidth, and better scalability, it is viewed as a technology that has the potential to replace traditional interconnect paradigms. The IC industry has also started exploring three-dimensional (3D) integration of wafers and dies using through-silicon-vias as interconnects. Compared to 2D integration, multiple device layers are manufactured and stacked vertically to utilize the third dimension. This solution leads to very short interconnect lengths, with a significant reduction in circuit delay and power consumption.

1.1 Basics of Circuit Testing

Before manufactured chips are shipped to customers, they must be thoroughly tested to eliminate defective parts. Testing is carried out by applying test stimuli, and checking circuit responses against good signatures. With increasing circuit complexity, testing is now a major component of chip manufacturing cost—as high as 50% [10]. Test specifications are derived from the device specification document that contains the design-related information, such as functional behavior, device characteristics, technology node, and operational requirements. A test plan is generated from test specifications. The type of test equipments to be used and the type of tests to be applied is specified in the test document. We discuss four major categories of testing in this section, namely, verification testing, production testing, burn-in, and incoming inspection [11].

1.1.1 Categories of Testing

In order to check whether the design meets its specification, verification testing is performed. This step verifies that the design is correct before it is sent for production. In this phase, functional tests are run, and a combination of formal techniques and simulation is used to detect design bugs. Any deviation from the specification is followed by the correction of design errors or changes in the design specification. A roadmap is then laid out for production testing.

Production testing targets every manufactured chip, and its goal is to screen defects by achieving a high coverage of modeled faults [11]. Since every device is tested for defects, cost-effective test application is mandatory. Test time must be minimized, since it is a major contributor to test cost.

Production tests alone do not guarantee high quality of every integrated circuit device. When used in real operating conditions, the time after which a device fails (time-to-failure) varies for each device. Reliability screening is customarily performed before the chips are shipped to customers [12]. Since the time-to-failure can be arbitrarily large for a device, accelerated test techniques, such as burn-in, are used for expediting the failure rate of the device. During burn-in, chips are subjected to a combination of production tests, high temperature, and over-voltage power supply [11].

During incoming inspection, a device is put under test by the customers and the system integrator. The test applied in this phase may be similar to production testing or a more comprehensive test is applied depending on the requirement. The goal is to ensure that a faulty device is not assembled into a system.

The collection of design techniques that have been developed to facilitate the testing of ICs is referred to as design-for-testability (DfT). These techniques are discussed in the following section.

1.2 Design for Testability (DfT)

Due to the complexity of today's IC designs, and the large number of manufacturing steps, comprehensive testing is a necessity to screen defects and ensure satisfactory yield. Testing cannot be carried out efficiently without dedicated hardware support. Major targets of DfT techniques include high fault coverage, minimization of test execution time, low performance impact and hardware overhead, etc. In this section, we discuss some commonly used DfT for digital logic circuits. DfT techniques for memory circuits, analog or mixed-signal blocks are not discussed here.

1.2.1 Scan Design

Today's microprocessors and ASICs include large sequential circuits consisting of hundreds of thousands of clock-controlled flip-flops. It is difficult to test sequential circuits because flip-flops must be initializable and the states of these flops must be

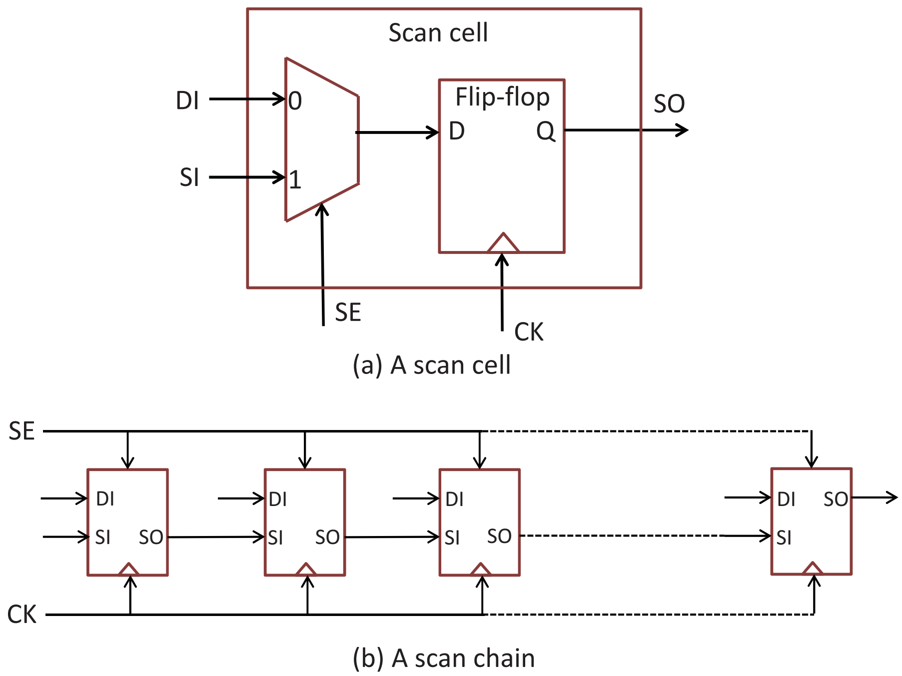
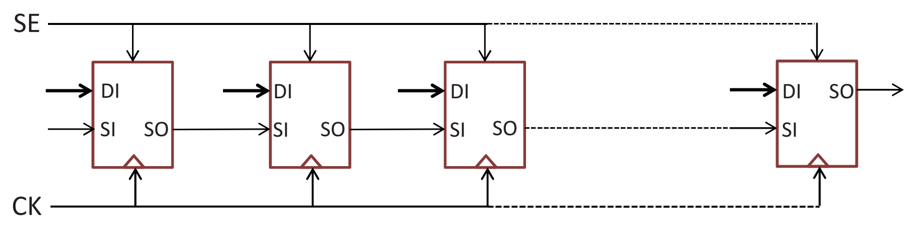


FIGURE 1.1: (a) Using a multiplexer to convert a flip-flop to a scan cell. (b) Multiple scan cells are stitched to form a scan chain.

observed. This requirement is quantified using two metrics: (i) *controllability*, or the difficulty of setting the system to a known state; (ii) *observability*, or the difficulty of propagating internal states of the system to its primary outputs.

In order to maximize testability, design techniques are used to modify flip-flops to scan cells and these scan cells are interconnected to form one or multiple shift registers, called *scan chains*. As shown in Figure 1.1(a), a scan cell has two different inputs that can be selected and the selection is controlled by the scan-enable (SE) signal. The first input, data input (DI), is driven by the combinational logic of a circuit, while the second input, scan input (SI), is driven by another scan cell in the scan chain. The scan-out (SO) pin of the first scan cell in a scan chain is connected to the SI pin of the second scan cell in the chain, and similarly, the other cells are interconnected to constitute the entire chain; see Figure 1.1(b). A scan chain



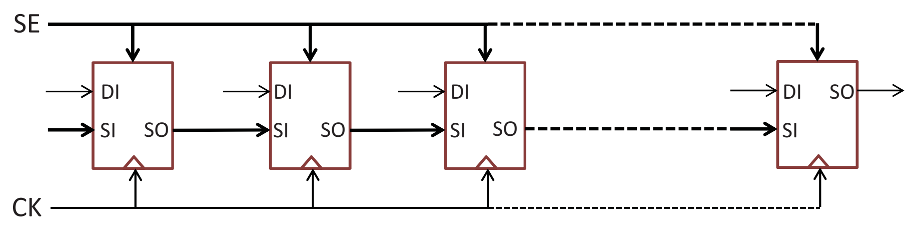


FIGURE 1.2: Illustrating two different modes of a scan design. The active signals in each mode are highlighted.

is externally controllable through the SI pin of the first scan cell and observable through the SO pin of the last scan cell in the chain. Since a design can have millions of flip-flops, scan cells are partitioned into multiple groups to create a scan chain per group.

The scan chains of a circuit can be configured in two modes: (i) in the normal/functional mode, data input is selected to update the output; (ii) in the test/shift mode, scan input is selected to update the output. The SE signal is held low in the normal mode and high in the test mode. In the test mode, a test vector is scanned into the scan chain, one bit at a time, to set the scan cells to desired states. The number of clock cycles needed for this purpose is equal to the number of scan cells in the longest chain. The test response to the scanned-in vector is captured by applying the functional clock during the capture cycle(s), and subsequently the states of flip-flops are observed by shifting out the contents of the scan register. The two

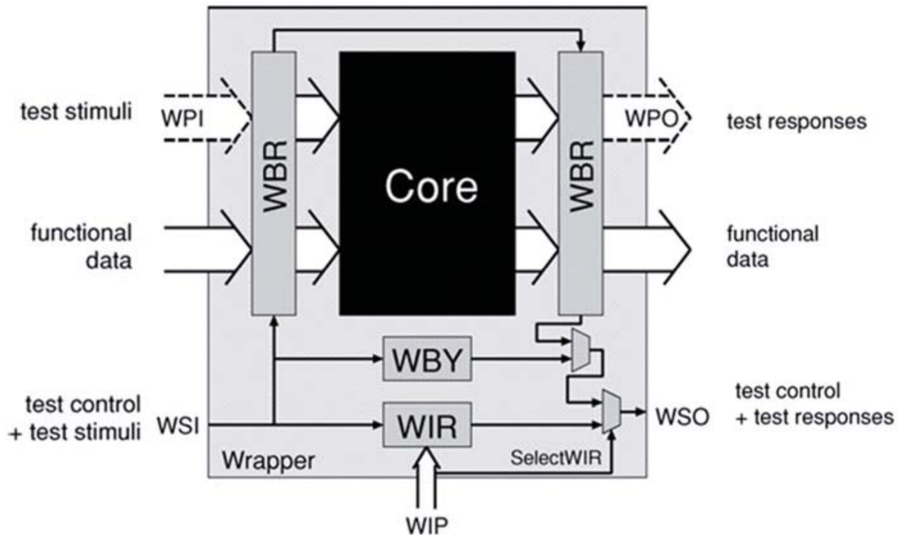


FIGURE 1.3: A test wrapper based on IEEE 1500 standard [5].

modes are shown in Figure 1.2. This shift mechanism makes it possible to set the scan chains to an arbitrary state by just controlling one or more primary inputs and observing the contents of the scan chains through one or more primary outputs. Although there is an associated cost of implementing scan design in terms of gate count, area overhead, and performance degradation, the benefits offered by scan design counterbalance these penalties.

1.2.2 Modular Testing Using Test Wrappers and Test-Access Mechanism (TAM)

A SOC today is composed of over a billion transistors. A modular approach towards the testing of such a large system involves a divide-and-conquer approach where modules (or cores) are tested separately [13, 14]. Test application and test access are facilitated through a modular test approach [2, 13, 15–19]. Testing poses difficult challenges when a hierarchical SOC is integrated using older-generation SOCs that are present as cores in the new hierarchical design [20].

In modular testing, a test wrapper is used to isolate an embedded core from the surrounding logic. A test access mechanism (TAM) is designed to deliver test data from the input/output (I/O) pins of the SOC to the embedded core. This strategy significantly reduces the effort required for test generation, since a module is considered independently of the other modules in the SOC. A pre-computed test set for each core can be reused if the core is instantiated multiple times in the SOC or if the design the core is reused in other SOC designs.

By using a standardized modular test approach, an SOC integrator can include a core provided by a core vendor without knowing the design details. Each core is equipped with a standardized test interface, such as that mandated by the IEEE 1500 standard [21]. We give a brief description of the 1500 standard through the example of wrapper design; see Figure 1.3. The configurable design of an IEEE 1500 wrapper consists of a wrapper instruction register (WIR) that is loaded with predefined instructions corresponding to functional mode or test mode. In addition, a wrapper serial interface port (WSP)—consisting of one-bit wrapper serial input (WSI), wrapper serial output (WSO), wrapper clock (WRCK), and other signals—is included to configure the WIR. A wrapper boundary register (WBR) is used to scan in (scan out) test patterns (test responses) that can either be applied to test the core or the interconnects between cores. When a core is tested, the WBR also feed test vectors to the internal scan chains of the core. To reduce the overall test time, a wrapper bypass register (WBY) is provided with every core to allow test patterns or responses to bypass the core.

When the wrapper is configured in *serial test mode*, the WBR and all the scan chains of the core are serially connected to form a single large chain. The test vectors are scanned in and out of this chain using the single-bit ports WSI and WSO, thereby leading to excessively long test application time. Therefore, the wrapper can also provide additional multi-bit lines of wrapper parallel input (WPI) and wrapper parallel output (WPO) for simultaneous loading and unloading of multiple scan chains. Several design and optimization problems have been addressed in the context of test access using the IEEE 1500 wrapper [2, 18, 22–24].

1.2.3 Built-In Self-Test (BIST)

Test of a digital system must be effective, quick, and applicable throughout the system lifetime. When the test component of a system is specified as one of the key system functionalities, the test circuit is implemented on the chip itself. The DfT technique, where a part of a circuit on a chip, board or system is used to test the logic circuit itself, is referred to as built-in self-test (BIST). Some of the benefits provided by BIST are listed below:

- BIST obviates the necessity for long test-pattern generation and application times.
- Test data is not required to be stored in automatic test equipment (ATE) for testing an SOC; test patterns are generated by the BIST hardware on the fly.
- Scan-based test is limited by the frequency at which test patterns can be applied. BIST eliminates this limitation by facilitating at-speed test; therefore, it allows the detection of delay faults.
- BIST reduces test cost because of the reduced dependence on test equipment.

BIST solutions can belong to either of two categories:

- **Online BIST** is used for testing the circuit in functional mode. It can be done either concurrently or nonconcurrently. When tests are applied simultaneously with the functional operations, it is referred to as concurrent online BIST. When a transient error is detected, the system is rolled back to a previously stored good state, and the tests are reapplied. An interrupt signal is generated

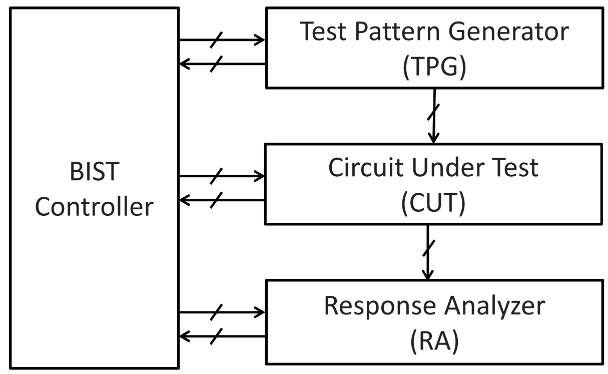


Figure 1.4: Components of a typical BIST system.

on detecting repeated failures. Nonconcurrent online BIST sessions are invoked when the circuit is idle.

- **Offline BIST** is applied under a special test mode and does not detect any errors in real time. It is either performed to test the circuit functionally, based on functional specification of the logic block to be tested, or structurally, based on the structure of the circuit and a given fault model.

Figure 1.4 shows a typical BIST system consisting of a test pattern generator (TPG) that generates test patterns automatically and a response analyzer (RA) that compacts the output responses of the CUT into a signature. The BIST controller is responsible for triggering BIST sessions and reporting pass or fail after matching the stored signature (called golden signature) with the signature produced by the CUT.

1.3 Network-on-chip

To support high data bandwidth in manycore SoCs, a suitable on-chip communication infrastructure is needed. Continued increase in wire lengths relative to feature size, and the associated problems of interconnect delay and power consumption, have motivated research on scalable communication infrastructures. Furthermore, the continuous reduction in the design cycle time requires not only the reuse of logic cores, but the reuse also of the on-chip interconnect fabric. Studies have shown that compared to buses, a packet-switched network-on-chip (NoC) is better suited for communication-intensive applications when the number of cores is as low as 8 [25]. For systems with up to 16 cores, the bus-based interconnection fabric was found to be better only in the cases of lighter workloads. For SoCs with a larger number of cores, an NoC offers many benefits, including reduced wire lengths, less power consumption, and better scalability. An NoC is therefore viewed as a promising alternative to today's bus-based communication fabric.

In this section, we give a brief overview of an NOC and describe recent advances in the testing methodology of an SOC crafted using a NOC.

1.3.1 Basics of an NOC

A typical network-on-chip consists of three basic components: routers, links and network interfaces [26]. Routers are responsible for routing communication data according to a routing protocol. Routers are interconnected using links that are composed of multiple channels. A network interface translates between a router's and a core's communication protocols. The Open Core Protocol (OCP) is an example of a core-centric interface protocol that facilitates rapid, plug-and-play SoC integration [27].

The topology of the network describes the logical layout of the NoC, i.e, the arrangement of NoC components [28]. In a grid-based layout, a router has multiple input and output ports, e.g., one port each for five different directions (north, south, east, west and core). A router receives data on one of its input ports and forwards it to one of the output ports. The path a packet takes from source to destination is decided by a routing algorithm. To keep the hardware overhead low, most routing

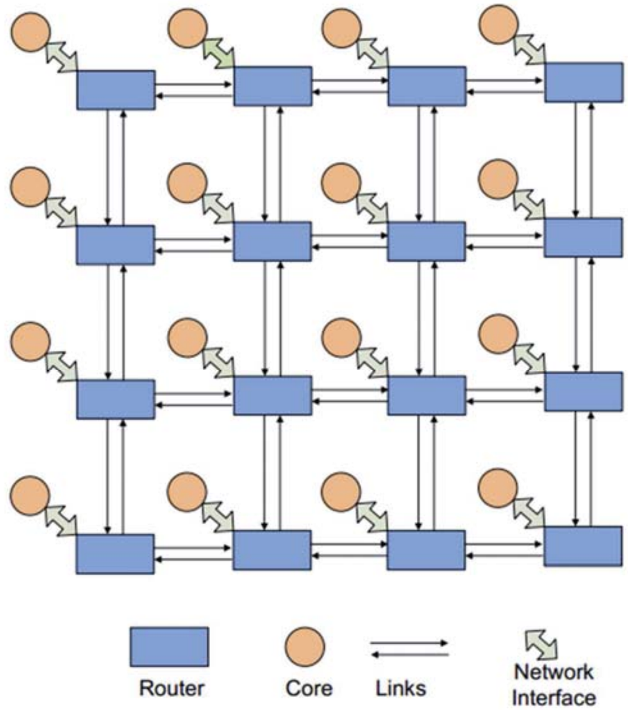


FIGURE 1.5: Illustration of a grid topology in an NOC-based SOC.

algorithms are designed to be deterministic, and are either based on the current router and the packet's destination (dimension-order-routing) or encoded into the packet by the sender (source routing). Implementations of some dimension-order-routing algorithms are presented in [29]. While the implementation of a source routing scheme depends on expensive look-up tables, dimension-order routing can be implemented with lightweight combinational logic. The XY-routing method is an example dimension-order routing, in which data is first routed along the X axis and then along the Y axis. Since this scheme does not take into consideration the on-chip network traffic, routing decisions are deterministic.

Cores communicate each other using packets. Packets are broken down into small pieces called *flits* (flow-control digits), which are the atomic data transported between routers. The most popular forwarding strategy is wormhole switching; paths are setup up by the *header flit*, which contains the destination address that is used for setting up a route for all the subsequent flits. The payload data is contained in the *body flits* that follow the header flit. The established route is then torn down by the *tail flit*, i.e., the last flit of the packet.

Figure 1.5 shows a grid (mesh) topology for an NOC-based SOC. Intel's 80-Core Teraflops research chip was designed using an 8x10 grid-based NOC [30]. Tilera, a chip vendor, announced a range of SOCs with up to 100 cores, all integrated using a grid topology for an NOC [31]. A scalable, cycle-accurate simulation of a 1000-core chip based on a mesh topology is presented in [32].

1.3.2 NOC Testing

The testing of NoC-based multi-core SoC by reusing the NoC infrastructure has been studied in the literature [33–41]. Instead of requiring a dedicated test-access mechanism (TAM) for testing the cores, the NoC can be used as a TAM for the SoC. This approach enables the use of existing functional interconnects for testing and it reduces test cost.

An automatic test equipment (ATE) is used to deliver test patterns to the logic cores in an SoC and to analyze test responses from the chip. Since the ATE represents a significant test investment, it is necessary to develop a systematic approach for utilizing the ATE resources optimally. Several studies reported reduction of test cost through effective utilization of test resources, such as tester memory and tester channels, to test an SoC using dedicated TAM [18, 22, 42]. Similar innovations are now required for multicore SoCs that use an NoC interconnection fabric.

NOC-based modular testing of cores also require additional DfT logic. A core is encapsulated by a test wrapper that translates data between the core's internal scan architecture to the core's external test pin width. The wrapper allows the core to receive and send test data over the NOC. Modular test wrapper designs that reuse functional interconnects has been studied in [1, 38]. An ATE accesses the NOC through the use of one or more *access points*. The access points translate between the test stream from the ATE and the packet-switched on-chip communication network. The width conversion between flit width and the number of test pins is also assisted by access points, as detailed in [35].

1.4 3D Integration Technology

Three-dimensional (3D) stacking involves the integration of multiple silicon dies in a vertical stack using short through-silicon vias (TSVs) [43]. For next-generation IC designs, 3D integration technologies offer multiple benefits, which include [44]:

- Decrease in latency and power consumption due to the reduction in interconnect length;
- Increase in memory bandwidth because memory layer(s) can be stacked over logic layer(s) using a large number of through-silicon-vias (TSVs).
- Since 3D integration exploits the availability of the third dimension, which is not exploited in a conventional 2D layout, transistors can be more densely packed, thereby reducing the form factor.
- Heterogeneous fabrication processes are possible for the same chip.

3D-stacked memory chips are already in production [45, 46] and the semiconductor industry is headed towards further exploitation of the benefits provided by 3D integration in a variety of product lines, such as 3D NOC [47], 3D memory-on-processor [48], and 3D FPGA [49]. The emergence of 3D logic-logic stacks has also been predicted for the near future [50]. Motivated by advances in design and technology, researchers have started investigating test and design-for-testability techniques for 3D ICs [8, 51–53].

Out of multiple 3D integration technologies, two techniques have emerged as being especially promising [44]: (i) monolithic integration; (ii) stacking integration. In the former approach, multiple device layers are grown on the same wafer sequentially. Once a layer of devices is manufactured, another layer of devices is grown on it after the deposition of an inter-level dielectric layer isolating the device layers. In contrast to the monolithic approach, the stacking integration approach processes each layer separately using conventional 2D fabrication methods, and subsequently stacks these layers vertically through bonding techniques. Since existing equipment and technology are least impacted in the stacking methodology, it is preferred over the monolithic manufacturing process [44, 53].

Multiple 3D interconnect technologies have been reported in the literature, including wire bonded, microbump, contactless, and TSV-based vertical interconnects [54]. The wire bonds create external connections between the board and the stack, or between dies themselves. These external wirings are located on the periphery of the stack and and probe pads are required across all metal layers to absorb mechanical stress due to the wired connections. This results in low density of interconnects. Microbumps are small metal balls on the die surface that allows communication with other dies. They are better alternatives to wire bonds because the interconnect density is increased and mechanical stress is reduced. Contactless approaches, including capacitive and inductive coupling methods, consist of less manufacturing steps, but because of manufacturing bottlenecks, cannot provide high interconnect density [54]. Because TSVs offer greater interconnect density than other approaches, it has emerged as the most promising technology, and an overwhelming majority of the reported research work on 3D stacking focus on TSV interconnects. Our research also addresses 3D stacking using TSV interconnects.

Based on the method of 3D stacking, the stacking process is further divided into three categories: (a) Wafer-to-wafer, where two or more wafers, each with multiple

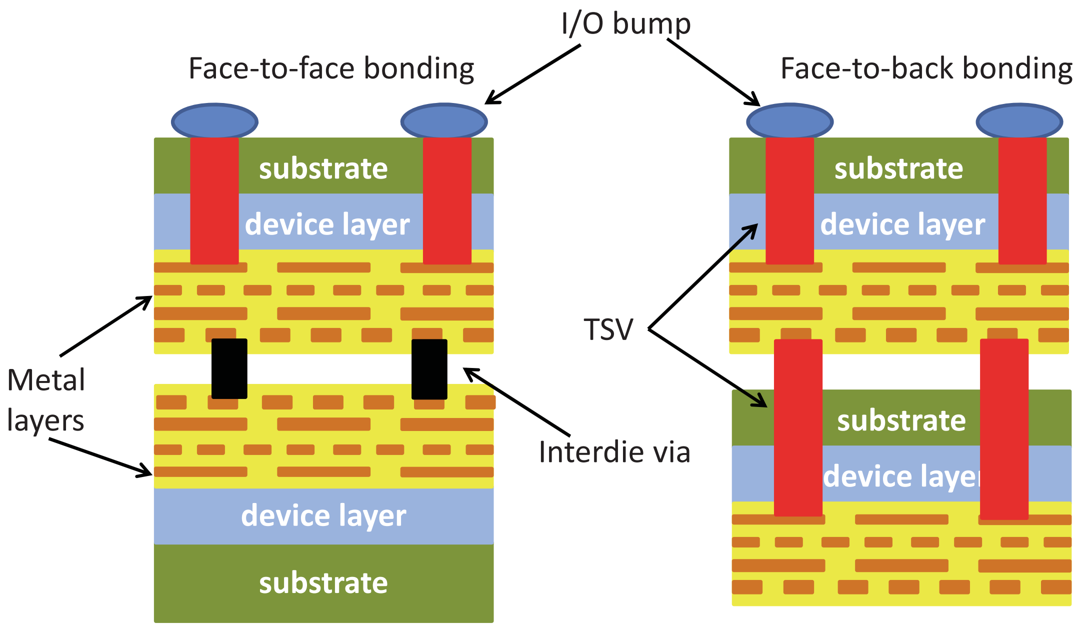


FIGURE 1.6: Two different stacking orientation of dies during 3D ICs.

instances of a circuit, are aligned on top of one another and bonded together before the resulting stack of wafers is diced to create 3D-stacked ICs; (b) Die-to-wafer, where a wafer is diced into individual dies, which are then stacked on another wafer; (c) Die-to-die stacking involves the dicing of every wafer into individual dies, and stacking dies from different wafers to create 3D chips. These methods lead to different level of the overall yield for the stacking process.

Depending on the moment at which TSVs are implanted in the substrate, the TSV processing step can be classified into two types: (i) Front-end-of-the-line (FEOL) approach, in which TSVs are added to a substrate before device and metal layers are created; (ii)Back-end-of-the-line approach, in which TSVs are added after the processing of the the active device and metal layers.

Since a TSV is embedded in the substrate, it must be exposed to build electrical connections across different layers of dies. This is carried out using "thinning", in which the substrate is ground away until the embedded TSVs are exposed. The thinning process makes the substrate brittle, and therefore dedicated carrier wafers are used for handling the delicate wafers for further processing. In order to create vertical stacks, a die or a wafer needs to be "aligned" and "bonded" to other die or wafer. During alignment, dies are carefully placed to ensure direct electrical connections between TSV. Misalignment induces faulty behavior in the circuit and could result in a huge yield loss. Bonding is achieved either through direct metal-to-metal bonding, direct bonding of silicon dioxide, or using adhesives [55].

Based on the stacking orientation, the stacking method can either be (i) face-to-face, in which the faces (metal side) of two dies are connected to one another, or (ii) face-to-back, in which the metal layer of one die is connected to the TSVs on the back (substrate side) of other dies. While pure face-to-face bonding method allows integration of only two dies vertically, a mix of the two methods can result in creation of stacks consisting of multiple dies. Other considerations such as thermal dissipation, power supply and cost limits the number of dies in a stack. Figure 1.6 depicts a stack consisting of two dies using the two approaches.

3D-stacked ICs are commercially available, but these are limited to memory-on-memory devices because the yield can be made high (due to homogeneity of dies and built-in self-repair) [56], and minimal testing requirements. Advances in design automation tools, test techniques and fabrication methods are needed to tap the enormous potential of 3D integration to build memory-on-logic and logic-on-logic devices consisting of heterogeneous cores.

Compared to the testing of conventional 2D ICs, there are several new challenges in the testing of 3D ICs that arise from the unique processing steps of wafer thinning, alignment, and bonding. These challenges are related to the development of wafer probing techniques, test access to modules in wafers and dies, design testability, test economics (cost-benefit analysis), and new defects because of the new processing steps [53].

During stacking, yield loss for each die is compounded; stacking of untested dies results in a significantly low yield. This necessitates pre-bond testing,or testing dies prior to bonding to obtain known-good-dies (KGDs). There are several problems associated with pre-bond testing. First, the thinned wafers are fragile, so testing should be done with very few probe touchdowns and using contacts with low contact-force probes. Second, the partitioning of the circuit at the design level or at the circuit level limits the tests that can be applied to partial logic. Additional test constraints due to a limit on the number of test TSVs and the presence of a limited number of I/O pins necessitate the need for design and optimization tools for better utilization of test resources. These problems are also present during mid-bond testing for partial stacks and post-bond testing for full stacks.

The chip vendors must take into account the cost of manufacturing and testing 3D ICs. As explained above, various integration approaches are possible in realistic scenarios. Each of them should be carefully evaluated to select a cost-effective approach. Furthermore, 3D ICs can be subjected to multiple tests at various stages of integration. Therefore, a comprehensive cost model is also required to determine "what to test" and "when to test" to minimize test cost. Tools are needed to analyze trade-offs associated with test strategies for 3D chips.

A die-level wrapper design was presented in [57] to modularize the testing of 3D ICs. The wrapper leverages existing test standards, namely, IEEE 1149.1 and IEEE 1500, to allow pre-bond tests, mid-bond tets and post-bond tests. Besides supporting test mode and functional mode, the wrapper is configurable and allows concurrent testing of multiple dies at a time. Test data can bypass dies leading to a possibility of creating an arbitrary test schedule of cores in the stack. This architecture uses wrapper cells on both ends of a TSV to provide testability to the die logic during pre-bond testing, when one end of the TSV is not connected to any die logic. The number of wrapper cells becomes prohibitively large if the number of TSVs is in the order of tens of thousands. The number of wrapper cells must be reduced.

The above discussions on NOC and 3D IC test have highlighted the difficulties being faced today in testing devices assembled using the above integration technologies. Optimization techniques are needed to make optimal use of limited resources, both in terms of test access and test scheduling. In this thesis, test scheduling algorithms for testing logic cores in NOC is presented that can pave the way for future work on more complex NOC devices. A cost model for minimizing the manufacturing test cost of 3D IC chips is also presented. A method based on A* search algorithm is presented to find an optimal test flow for minimizing overall cost. A graph-theoretic approach for minimizing the number of wrapper cells without compromising the fault coverage during pre-bond and post-bond tests is discussed. An integrated framework for enabling BIST in 3D stacked ICs is also presented.

1.5 Outline of Dissertation

The remainder of this dissertation is organized as follows.

Chapter 2 presents test-scheduling algorithms for minimizing SOC test time for a variety of NOC topologies and routing mechanisms. For NOCs with an arbitrary topology and dedicated routing paths, a formulation based on subset-sum problem is proposed. The test-time minimization problem is also formulated as a grid partitioning problem, and an algorithm based on dynamic programming is proposed. Making use of multicast routers, two methods—one based on integer linear programming (ILP), and the other on dynamic programming—are discussed to exploit the homogeneity of cores in large SOCs to further reduce the SOC test time. A method to account for power constraints is also discussed in the chapter. Through simulation results, we demonstrate the scalability of our approach for SOCs consisting of more than 1,000 cores.

Chapter 3 describes a generic cost model to account for various test costs involved in 3D integration and presents a formal representation of the solution space to minimize the overall cost. An algorithm based on A*—a best-first search technique—to obtain an optimal solution is presented. An approximation algorithm with provable bounds on optimality is proposed to further reduce the search space. In contrast to prior work, which is based on explicit enumeration of handful of a test flows, a formal optimization approach is adopted that allow us to select an effective test flow by systematically exploring an exponentially large number of candidate test flows. Experimental results highlight the effectiveness of the proposed method.

Chapter 4 proposes two heuristic algorithms for minimizing the number of wrapper cells needed to provide testability to logic between TSVs and scan cells. We prove that the general problem of minimizing the number of wrapper cells is equivalent to the graph-theoretic minimum clique-partitioning problem, and is therefore NP-hard. Reusing existing scan cells may lead to performance degradation and routing congestion; therefore, the proposed methods incorporate timing-guided and layout-aware approach to address practical concerns. DfT optimization techniques leveraging the reuse-based method during post-bond testing are also presented.

Chapter 5 describes an end-to-end design of a built-in self-test (BIST) infrastructure for 3D-stacked ICs that facilitates the use of BIST at multiple stages of 3D integration. The proposed BIST design is distributed, reusable, and reconfigurable. The distributed infrastructure enables application of BIST at any test stage—prebond or post-bond. Reusability facilitates the use of the same infrastructure for every die, and for every test stage. Reconfigurability provides the ability to selectively apply right test at the right time, i.e., it supports the execution of static test schedules that are created using a test-scheduling algorithm. The chapter also formulates a test-scheduling problem that aims at minimizing test time under BIST-resource and power constraints, and use two algorithms based on bin packing for solving the problem.

Finally, Chapter 6 summarizes the contributions of the dissertation and identifies directions for future work.

2

Test Scheduling Algorithms for Network-on-Chip

A dedicated TAM for a many-core SOC with an NOC interconnection fabric is not necessary because the NOC infrastructure itself can be used for test-data delivery to the cores. To ensure congestion-free test-data delivery and minimize test time, a number of scheduling techniques have been presented in the literature (e.g., [1, 33, 36, 37, 39, 40]).

In this chapter, we present mehods for test-delivery optimization for a variety of NOC topologies and routing mechanisms. The minimization of the overall test time using the NOC is a complex problem that requires the co-optimization of the number and placement of access points, the distribution of ATE channels to these access points, and the assignment of cores to access points for test-data delivery. Homogeneity of cores in SOC can also be exploited using multicast routing[1] to minimize test time. Previously proposed techniques neglect one or more of these important aspects.

[1] In multicast routing, data is sent from one source node to multiple destination nodes simultaneously.

The remainder of the chapter is organized as follows. Section 2.1 provides a generic problem statement applicable to all flavors of NOCs discussed in this chapter, reviews prior work, and lists the contributions. Section 2.2 introduces the subset-sum problem, and shows how it can be used for test scheduling in NOCs with arbitrary topologies and dedicated routing paths. Section 2.3 provides details about the proposed dynamic-programming algorithm for solving the test-scheduling problem in NOCs with a grid topology and XY routing. Test scheduling under power constraints is discussed in Section 2.4. Section 2.5 discusses how we can leverage the capability of multicast routers to fork flits to further reduce test time. Results are presented in Section 2.6, and Section 2.7 concludes the chapter.

2.1 Generic Problem Statement and Prior Work

When an access point delivers test data to a core, the path consisting of routers and links between the access point and the core is used for the entire duration while the core is being tested. Suppose L_1 is the set of links used for communication between an access point a_1 and any of the cores assigned to it. If we test a core (not assigned to a_1) through another access point a_2 such that the links that are used do not belong to L_1, then link contention is avoided. Thus, no link contention can occur as long as the above sets of links are pairwise disjoint. The SOC test time is the maximum time taken to test cores using any of the access points. The time to test cores through an access point a_1 is defined as the sum of the test times of the cores that are assigned to a_1. Our goal is to minimize the SOC test time.

2.1.1 Formal problem statement

Given an NoC-based SOC having $|\mathbf{C}|$ cores, $|\mathbf{A}|$ access points and P available test pins, our goal is to determine a distribution of the P pins to the $|\mathbf{A}|$ access points and an assignment of the cores to the access points such that the sets of links used for communication between the access points and their assigned cores are pairwise disjoint or non-overlapping, and that the overall test time (under this restriction) is minimized. We refer to this problem as the *contention-free assignment problem* or CFA.

We have adopted two approaches to solve the above problem.

- The first approach is to explicitly find a partition of the set of available links into pairwise disjoint subsets under the constraints imposed by location of access points.

- The second approach is to create a partition of the given NOC into $|\mathbf{A}|$ regions; cores with in the same region are tested sequentially and multiple regions are tested in parallel. On creating a partition of the NOC, this method implicitly finds a partition on the set of available links.

The location of access points have to be specified as an input for applying the first method, whereas in the second method, one access point is assigned to each region. Depending on the NOC layout and the routing capability of network routers, the most suitable of these methods can be selected.

2.1.2 Related prior work

The testing of the NOC infrastructure, fault detection and its reconfiguration in the presence of faults has been studied in [58, 59]. Our focus in this chapter is on test scheduling for the logic cores using NOC infrastructure under constraints imposed by the NOC and the ATE.

A scheduling algorithm that assigns a higher priority to cores requiring longer test time, and finds shorter test-delivery paths to these cores, was proposed in [36]. This approach was extended to take into account power constraints in [34, 37]. An optimization method to identify dedicated routing paths and incorporate precedence constraints and shared BIST resources was provided in [60].

A differential clocking scheme to reduce power consumption in the cores, and utilize the capability of an NOC to run at a frequency higher than scan clock, was presented in [39]. Multiple clock speeds can also be leveraged to reuse common links connected to cores on a time-sharing basis [40]. Simultaneous optimization of NOC testing and core testing was presented in [41].

A test wrapper design for the reuse of functional interconnects was introduced in [1, 38]. TAM width constraints imposed by the NOC flit width, and the associated problem of flit-bit under-utilization were highlighted in [61]. Other wrapper design approaches are presented in [62, 63], but these methods do not show how test time reduction can be achieved.

The compression of test input data for reducing the number of test input pins was studied in [64]. Test-output compression to reduce output pin-count was discussed in [33]. Our approach can be extended to incorporate input and output compression as proposed in [33, 64].

In [33], a general test-scheduling problem was formulated and solved using ILP. Furthermore, a heuristic approach was also presented, where contention was avoided for routers and links between cores that were assigned to different access points. However, the heuristic approach is based on an ILP model that does not scale well with the number of cores and the number of access points. No methodology was discussed on an appropriate selection of the number of pins or the number of access points. In case of a large number of cores, it is practically impossible to run the heuristic multiple times to decide how many access points or pins should be used. Moreover, no strategy for locating optimal positions of access points was discussed in [33]. Scalability is a common drawback of all the above methods. No results have been presented thus far an SOC with hundreds of cores.

The need for partitioning the NOC was introduced in [35] to avoid jitter-less transport of test and response data. A systematic method for creating partitions was however not considered; the work was more focussed on the design of DfT hardware for efficient test data delivery. Even though partitioning of the NOC has been recognized as being an important component of a test solution, none of the prior methods have studied the optimal partitioning problem.

2.1.3 Chapter contributions

The major contributions of the chapter are as follows:

- We present a novel algorithm for minimizing test time for NOCs with arbitrary topology and dedicated routing that consistently yields shorter test times than previous approaches. This algorithm can also be used to minimize test times for SOCs using dedicated TAMs.

- We model test-delivery optimization problem for an NOC-based many-core SOC with grid topology as a grid partitioning problem and develop a dynamic programming solution.

- We show that optimal grid partitioning leads to significant reductions in test time. The partitioning solution is optimal in that minimum test time is obtained for rectangular partitions. The test times obtained using DP for the rectangular NOC topology are close to (no more than 10% in most cases) TAM-independent provable lower bounds.

- We show how test scheduling can be carried out under power constraints.

- For NOCs with grid topology, we add an extra degree of test parallelism by leveraging the capability of multicast routers to transmit a flit[2] to multiple

[2] A network packet is divided into multiple flits.

destination ports simultaneously.

- We demonstrate the scalability of the proposed methods by deriving optimization results for realistic SOCs of today and for the future. We present results for a 400-core SOC (20x20 NOC) of today, and even larger SOCs of the future—992 cores (32x31 NOC) and 1,600 cores (40x40 NOC). The results were obtained in reasonable time using modest computing resources.

- The proposed dynamic-programming algorithm computes the Pareto frontier for a range of access-point counts and total test-pin counts in a single run. Therefore, a wealth of data can be obtained to effectively choose the number of access points and test channels for the SOC.

2.2 Test Scheduling for NOCs with Dedicated Routing

2.2.1 Problem description

A graph-theoretic formulation for test-time minimization for NOCs with dedicated routing is presented in [1]. A graph is first constructed using the topology of NOC with network tiles representing nodes and edges between nodes replacing links. The goal is to partition the graph into disjoint connected components, and distribute pins to each component such that the maximum test time of each component is minimized. The test time of a component is equal to the sum of the test times of the cores in the component. The test time of a core depends on the number of pins allotted to the component to which the core belongs. We use the same formulation for devising a test-scheduling algorithm for an arbitrary topology. We further assume a dedicated routing path for the testing of each core in line with previous work on test scheduling [1, 60, 65].

We build on the system model presented in [35] where the ATE interface (access points) and the core wrappers implement protocol and width conversion such that both the ATE and the circuit under test (CUT) can remain unaware of the on-chip routing protocol and NOC design. We assume a width conversion ratio of one in this work; any other ratio can easily be accounted for.

2.2.2 Outline of the algorithm

If we exclude the initial delay introduced during setting up of dedicated routes in NOC for test-data delivery and response collection, the test time minimization problem for NOCs with dedicated routing can be considered as a special case of the bus-based TAM optimization problem [1]. This is because any partition of the set of cores is a valid solution candidate to the latter, while only those partitions that represent connected components in the graphical representation of the given NOC topology are solution candidates to the former. Hence, we first revisit the TAM optimization problem described in [2, 18] for a bus-based TAM architecture. By limiting the partitions considered by our algorithm to those that form valid connected components, the algorithm is also directly applicable to the NOC test scheduling problem (Section 2.2.F).

The TAM optimization problem, discussed in [2, 18], can be stated as follows. Given a set of cores with respective test time information for a range of pin widths, total TAM width, and the maximum number of TAM partitions, we have to find an assignment of cores to the TAM partitions and the TAM width of each partition such that the overall test time is minimized. We view this problem from a different angle; we partition the set of cores into disjoint subsets, and find an optimal distribution of pins or TAM wires to these subsets to minimize test time. A systematic method of enumerating candidate partitions is discussed.

Our algorithm consists of three parts: a subset-sum formulation to effectively enumerate candidate partitions in a reasonable time, a greedy method to optimally distribute pins to a given partition, and a brief introduction to the subset-sum prob-

```
SolveSubsetSum (X)
SUM(1, x_1) ← true
for i ← 2 to n do
    for s ← L_SSP to U_SSP do
        if s = x_i or SUM(i − 1, s) = true or SUM(i − 1, s − x_i) = true then
            SUM(i, s) ← true
        end if
    end for
end for
return SUM
```

FIGURE 2.1: Pseudocode for the procedure `SolveSubsetSum`.

lem and a method to solve it. We first describe the subset-sum problem (SSP) before elaborating on how we utilize SSP to tackle the TAM-optimization problem.

2.2.3 Subset-sum problem (SSP)

The subset sum problem (SSP) is posed in the form of a question. Given a set of n integers, $\mathbf{X} = \{x_1, x_2, \cdots, x_n\}$, does there exist a non-empty subset, whose elements sum to zero? This problem is known to be NP-complete [66]. An equivalent version of this problem asks the question if there is a subset of the given set with the sum of its elements equal to a given integer s. There exists no algorithm that can solve a given instance of SSP in polynomial time. However, a method based on dynamic programming solves the problem optimally in pseudo-polynomial time [66]. Figure 2.1 outlines this method—we call it `SolveSubsetSum`—that has run-time complexity of $O(n(L_{SSP} - U_{SSP}))$, where L_{SSP} is the least element of the given set, and U_{SSP} is the sum of all positive integers of the set, and $L_{SSP} \leq s \leq U_{SSP}$.

The binary element $SUM(i, s)$ of the 2-D array SUM stores a *true* if it is possible to create a subset from the set $\{x_1, x_2, \cdots, x_i\}$ with sum s, otherwise it stores a *false*. From the procedure `SolveSubsetSum`, it is evident that $SUM(i, s)$ is true only under three conditions:

- The subset consists only of x_i, i.e., $s = x_i$.
- There exists a subset with sum s without using the element x_i, i.e., $SUM(i -$

```
distributePins (P)
for i ← 1 to P do
    max_time_subset ← subset with maximum test time
    Increment the pins assigned to max_time_subset by 1
    Update the test time of max_time_subset
end for
```

FIGURE 2.2: Method for distributing P pins given a partition of set of cores.

$1, s)$ is true.

- The subset contains x_i in addition to elements from the set $\{x_1, x_2, \cdots, x_{i-1}\}$, i.e., $SUM(i-1, s-x_i)$ is true.

This recursive formulation is the foundation for solving SSP in pseudo-polynomial time and we leverage this idea for TAM optimization. We use repeated invocations of the above method on multiple instances of SSP to enumerate several candidate partitions of the given set of cores in K subsets, and use a greedy approach to distribute pins to these subsets.

2.2.4 Optimal pin distribution for a partition

Suppose the given set of cores is partitioned into K disjoint subsets, and the cores in a subset are assigned to the same TAM. Moreover, no two cores from different subsets use the same TAM. The number of pins available for distribution is P, where P is the total TAM width.

We present a greedy algorithm for distributing P pins and prove that the greedy approach is optimal for a partition of a given set of cores. Figure 2.2 outlines the algorithm for pin distribution. In each iteration, the subset having the maximum test time is assigned an additional pin. The algorithm terminates after every pin is assigned.

Let $G_P = \{g_1, g_2, \cdots, g_P\}$ be the solution generated by the greedy approach, where g_i denotes the index of the subset to which the i^{th} pin is assigned. Clearly, $1 \leq g_i \leq K$, for all i. Let $O_P = \{o_1, o_2, \cdots, o_P\}$ be an arbitrary sequence of pin assignments that leads to an optimal solution.

Let $T_g(i)$ denote the SOC test time after the i^{th} pin is assigned using the greedy approach. Similarly, let $T_o(i)$ denote the SOC test time in the i^{th} step for the given arbitrary sequence. Furthermore assume that $\tau_g(i,b)$ is the test time for the b^{th} subset after the i^{th} iteration using the greedy approach. Let $\tau_o(i,b)$ denotes the same quantity for the arbitrary sequence.

Clearly, $T_g(i) = \max_b\{\tau_g(i,b)\} \Rightarrow \tau_g(i,b) \le T_g(i) \ \ \forall b$, and $T_o(i) = \max_b\{\tau_o(i,b)\} \Rightarrow \tau_o(i,b) \le T_o(i) \ \ \forall b$.

Lemma 2.2.1. *The test time does not increase in any iteration of the greedy method, i.e., $T_g(i+1) \le T_g(i)$.*

Proof. The test time of a core does not increase with the addition of a pin. Since a pin is added at each step or iteration to exactly one subset, the test time of that subset can only decrease or remain the same. The test times of other subsets remain unaffected. This is also true for the arbitrary sequence, i.e., $T_o(i+1) \le T_o(i)$. □

Lemma 2.2.2. *In the partial sequence $G_k = \{g_1, g_2, \cdots, g_k\}$, suppose that the pin k was added to the subset b_m during the k^{th} step, i.e., $g_k = b_m$ and $T_g(k) = \tau_g(k, b_m)$. If a pin is removed from a subset, say b', such that $b_m \neq b'$, then the updated test time of the SOC is strictly greater than $T_g(k)$.*

Proof. According to the greedy procedure, a pin is added to a subset only when it has the maximum test time during an iteration. Since a pin is removed from b', the pin must have been added to b' during an iteration, say $i < k$, such that $\tau_g(i, b') = T_g(i)$. From Lemma 2.2.1, $T_g(k) \le T_g(i) = \tau_g(i, b')$. Therefore, the updated test time of b' after removing a pin is $\tau_g(i, b')$. Note that the greedy procedure keeps on selecting the same subset until its test time is strictly lower than another subset. Since $b_m \neq b'$, $\tau_g(i, b') > T_g(k)$. Since the test time of b' dominates $T_g(k)$, the SOC test time is

$\tau_g(i, b') > T_g(k)$. Note that the addition of the removed pin to any other subset does not reduce the SOC test time.

If $b_m = b'$, then $\tau_g(i, b') \geq T_g(k)$ holds. □

Theorem 2.2.1. *The greedy algorithm of Figure 2.2 finds an optimal distribution of pins for a given partition on the set of cores.*

Proof. We prove the theorem by contradiction. At step i, if the distribution of pins is same for both the methods, $T_g(i) = T_o(i)$ holds. If the distribution of pins is not the same, then there exists at least one subset that is assigned at least one more pin in the greedy approach than what is assigned to it in the other approach. If we try to rearrange the pins in the greedy sequence to match the pin distribution obtained from the arbitrary approach, then by Lemma 2.2.2, the test time can only increase; hence $T_g(i) \leq T_o(i)\ \forall i$. □

2.2.5 Enumeration of candidate partitions

The TAM-optimization problem has been shown to be NP-complete using transformation from the bin-packing problem in [67]; therefore, a polynomial-time algorithm cannot be designed for solving the problem optimally. The test time of cores is mapped to integer elements of the given set in SSP. Assuming that only one pin available to each TAM, a set consisting of the test times of all cores using just a single pin is created. The test times vary from small to large values depending on the scan chain length and the number of test patterns of individual cores. If the range is very large, it may be computationally prohibitive to run dynamic programming (Figure 2.1), both in terms of time and space. Therefore, the test times are scaled down by a factor of the lowest test time before approximating them to nearest integer values. The procedure `optimizeTAM` of Figure 2.3 is called on this set to create K subsets. The greedy method outlined in Figure 2.2 is executed on the resulting partition and the test time is noted.

```
optimizeTAM (X, K, depth, candidate_partition)
if X = ∅ or depth = K − 1 then
    Call distributePins on candidate_partition
    update min_test_time
    return
end if
Compute L_TAM
SUM = SolveSubsetSum(X) // if depth = 0, need not be done repeatedly.
for s ← L_TAM − Δ to L_TAM + Δ do
    X_s = findSubset(SUM, s)
    Add X_s to candidate_partition, if X_s exists, otherwise continue
    X_filter ← X\X_s
    optimizeTAM (X_filter, K, depth + 1, candidate_partition)
    Remove X_s from candidate_partition
end for
```

FIGURE 2.3: Pseudo code for the procedure `optimizeTAM`.

The element $SUM(i, s)$, for every s, holds an answer to the question whether the set $\{x_1, x_2, \cdots, x_i\}$ has a subset, the elements of which sum to s. For every value of s, we get a unique subset, if $SUM(i, s)$ holds a *true*. The procedure `findSubset` finds such a subset; it searches through the entries of the 2-D array SUM and constructs a solution. This is the backbone for enumerating candidate partitions. Note that the procedure `optimizeTAM` is a recursive procedure and the maximum recursive depth is equal to the total number of subsets that are needed to be created. In each recursive step, a subset $\mathbf{X}_s$ is created with sum less than equal to s. Then a new set $\mathbf{X}_{filter}$ is obtained by removing the elements in $\mathbf{X}_s$ from X. While $\mathbf{X}_s$ is appended to the list *candidate_partition*, which stores the subsets in a candidate partition, the set $\mathbf{X}_{filter}$ is fed in the next level of recursion. After returning from the recursive call, $\mathbf{X}_s$ is removed from *candidate_partition*.

This methodology can potentially enumerate all partitions, but for keeping the computation time low, we do not search through all the entries of the array SUM. First a lower bound L_{TAM} is computed as $\frac{\sum x}{K - depth}$, where $x \in \mathbf{X}$. The variable s is varied from $L_{TAM} - \Delta$ to $L_{TAM} + \Delta$, where Δ is a parameter that controls the size of solution space that is to be searched. Since the size of $\mathbf{X}$ decreases in every recursive step, we reduce the value of Δ by half in our experiments in successive recursive steps (not shown in Figure 2.3). Either a timeout can be set on the overall procedure, or a limit be placed on the maximum number of candidate partitions that are to be enumerated.

2.2.6 Application to NOC

Similar to extension made for NOCs in [1], the method described in the previous subsection is modified for solving the test-delivery problem in an NOC. Listed below are the required modifications:

- The `findSubset` procedure only returns a subset, whose elements form a connected graph.
- In the `distributePins`, the number of pins assigned to each subset is no more than the channel width.

The method described in [1] creates several connected components in its initial step and subsequently attempts to find a partition that minimizes test time; there is no control on how many connected components are created in the final solution. In contrast, our approach takes the number of access points (or connected components) as an input parameter. In Section 3.6, we show how restricting the number of connected components in [1] adversely impacts test time. Even if the number of connected components is kept unrestricted, for larger NOCs with a grid topology, we demonstrate that the method in [1] produces a significantly large number of access points with test times that are worse than the test times produced by our method with much fewer access points. More access points leads to additional DFT cost and higher power consumption.

2.3 Dynamic Programming Solution for a Grid Topology

The above approach and the approach from [1] are not applicable to an NOC with mesh topology that only supports XY-routing protocol for on-chip communication, as they create connected components of arbitrary shapes and do not ensure that packets are delivered without any conflict when the routing mechanism is XY. Moreover, XY routing can be viewed as a special case of dedicating routing, if the paths taken by packets in the former case are encoded in packet headers in the latter; therefore, the method that we propose in this section can still be used for NOCs with grid topology that use dedicated routing. We will demonstrate that by using XY-routing with grid topologies, we obtain better solutions than reported in [1] for larger NOCs.

2.3.1 Preliminaries

The basic idea of our approach is to partition a two-dimensional grid-structured NOC into rectangular regions, and interface each region with the ATE such that the testing in each region can be carried out in parallel without any contention for routers and links. The cores in a region are tested sequentially, and the regions are tested in parallel. This concept is illustrated in Figure 2.4. The figure shows a partition of a 4x4 NOC into three rectangular regions. These regions are interfaced with an ATE through ATE interfaces. The ATE drives test in all three regions simultaneously.

The goal of partitioning is to minimize test time for the region with the maximum test time. This method is suitable for NOCs that implement the XY-routing protocol, which is a popular routing algorithm because of its easy hardware implementation and guaranteed deadlock-free routing.

The advantage of optimal partitioning is illustrated in Figure 2.5. Each sub-figure shows a different partition of the same 4x4 2D grid network, where each cell represents a network tile. The number in each cell is the number of clock cycles

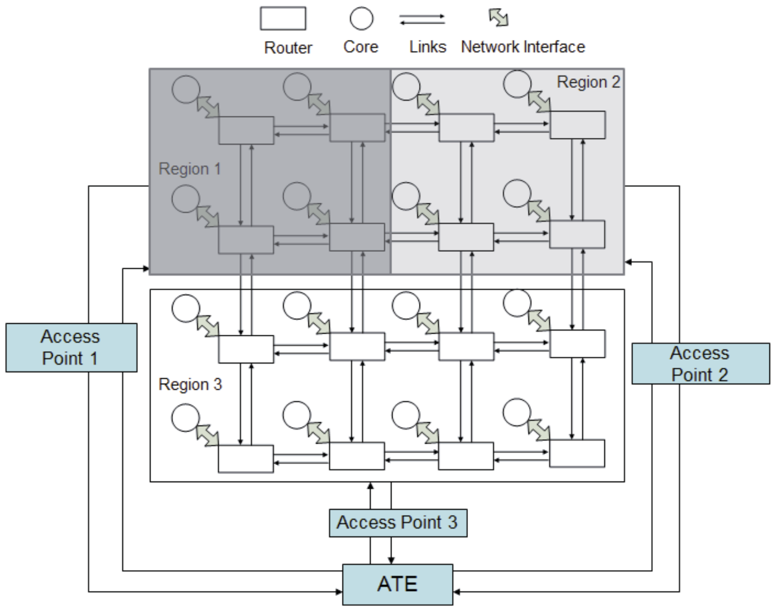


FIGURE 2.4: A figure showing the partitioning of a 4x4 NOC for parallel testing.

needed to test the core in that network tile. Note that the additional time needed to establish a connection to a core from an access point is neglected here because it is very small compared to the test times of the cores. For the purpose of illustration only, we assume that the test pins are evenly distributed to the access points. The quantity T refers to the total number of cycles that is required to test the chip. Since each region is tested in parallel, the maximum number of clock cycles needed to test any region is T. Figure 2.5(b) shows an optimal partitioning that reduces test time by 40% compared to the partitions shown in Figure 2.5(c). The number of regions is three in this figure, but in general, it is an input parameter to the proposed framework.

2.3.2 Problem formulation

An NOC $\mathbf{N}$ with a grid topology can be viewed as a rectangle in a two-dimensional Cartesian plane with the bottom-left corner of $\mathbf{N}$ coinciding with the origin $(0,0)$ of this plane. Any rectangle $\mathbf{R}$ can be uniquely identified with a four-tuple rep-

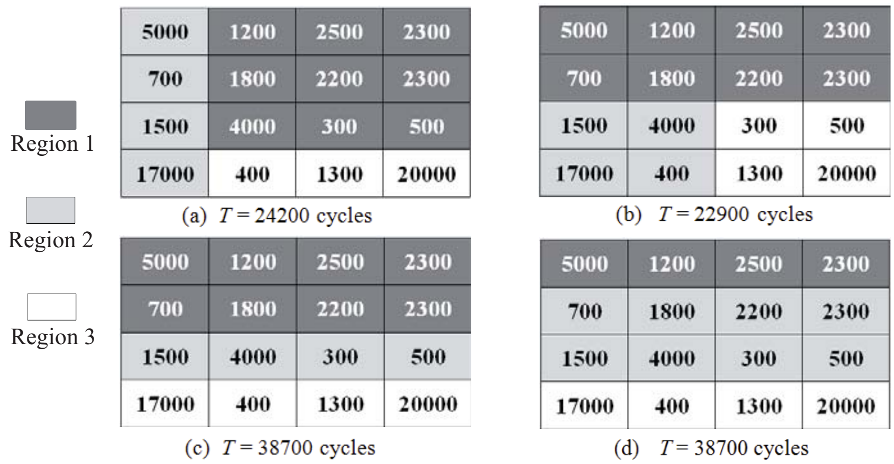


FIGURE 2.5: An illustration of the impact of optimal partitioning on testing time.

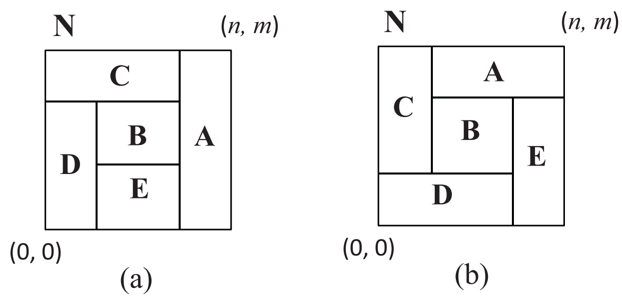


FIGURE 2.6: Two examples of inadmissible partitioning.

resentation $[i, j, l, w]$ where (i, j) is the bottom-left corner of $\mathbf{R}$, and l and w are the length and width of $\mathbf{R}$, respectively. This representation can be captured by a concise expression, $\mathbf{R} \equiv [i, j, l, w]$. If the dimension of $\mathbf{N}$ is mxn, we can write: $\mathbf{N} \equiv [0, 0, m, n]$.

Our goal is to optimally partition $\mathbf{N}$ into rectangular regions such that testing can be carried out independently in each region, thereby minimizing overall test time. This particular choice of regions being rectangular is motivated by two reasons: first, the popular XY-routing algorithm [68, 69] ensures that there is no conflict in test

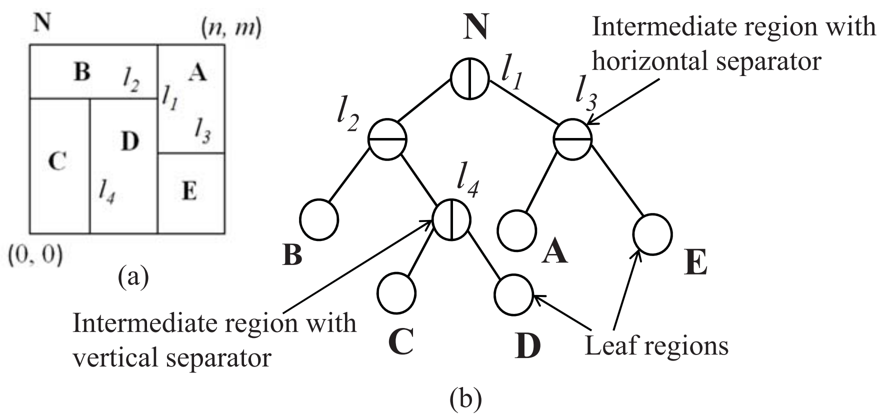


FIGURE 2.7: Mapping of a partition to a binary tree (l_1, l_2, l_3, and l_4 are separators).

data transportation because data will not cross the boundary of regions; second, it is more tractable to store and use results for subproblems in our dynamic programming-based approach.

We have to ensure that each region can be accessed by the ATE without requiring a dedicated TAM, hence, not all possible partitions of **N** are admissible. The two partitions shown in Figure 2.6, for example, are inadmissible; region **B** is enclosed from all sides and it cannot be tested independently. We say a region to be located at a boundary (or a *boundary region*) if it is not enclosed, i.e., if it has at least one *boundary edge*. An edge of such a region **R** is said to be a boundary edge if it is incident on one of the edges of the rectangle $[0, 0, m, n]$. In Figure 2.6(a), region **A** has three boundary edges, regions **C** and **D** have two each, **E** has one, and **B** has no boundary edges.

We create a partition with the help of a sequence of *separators*. A separator is any line segment parallel to either of the coordinate axes, and divides a region completely into two sub-regions. We impose the constraint that a separator has to be parallel to one of the axes. This constraint ensures that only rectangular regions are formed. We do not consider a sequence of line segments as shown in Figure 2.6(b) to create a partition; it can be easily shown that such a sequence will always create an enclosed region. A separator is horizontal (vertical) if it is parallel to the axis $y = 0$ ($x = 0$).

The partition shown in Figure 2.7(a) can be easily mapped to a binary tree with the region represented by the NOC as the root of the tree and each intermediate node representing the sub-region that is created by placing a separator in its parent region; see Figure 2.7(b). The orientations of separators are also captured in the tree in terms of how a non-leaf node is intersected. We call a region corresponding to a leaf of this tree as a $leaf\ region$. Alternatively, a leaf region can be defined as a region with no subdivisions.

We define the cost of a leaf region as the sum of the test times of the cores in that leaf region. The test time of a core is a function of the number of pins assigned to test the core. The number of pins used to test a core equals the number of pins assigned to the leaf region in which the core is present. The cost of an intermediate region **R** is the maximum cost over all costs of the leaf regions contained in **R**. The cost of **R** also depends on the corresponding partition size, where the size of a partition is the number of leaf regions created by that partition. Let us use $\pi_N(K, P)$ to denote the cost of **N**, where K is the given partition size (number of regions) and P is the number of available test pins. We can now formulate our problem as follows. Our goal is to determine an optimal sequence of separators such that:

- Each core is present in exactly one leaf region;
- Each leaf region is located at a boundary (boundary-region constraint);
- $\pi_N(K, P)$ is minimized.

For a given K and P, we use π_R to refer to the cost of any region **R**. Our goal is also to compute the pin assignment to each leaf region (pin-assignment problem), i.e., which of the K pins are used to access different leaf regions.

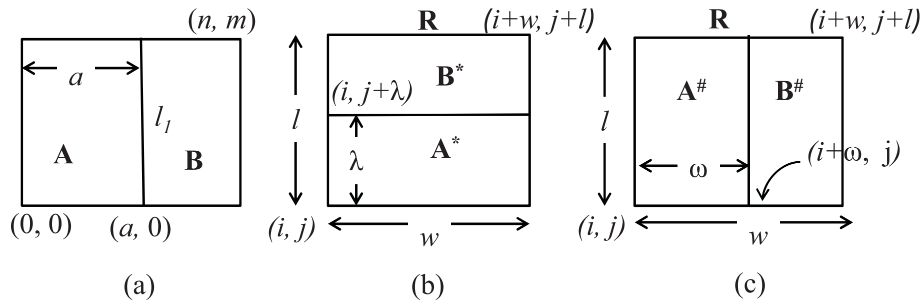


FIGURE 2.8: (a) Figure showing a position of a separator l_1 on the grid. (b) & (c) Figure showing two configurations of a rectangle $\mathbf{R}$ after a separator is placed.

2.3.3 Characterization of the optimal solution structure

Dynamic programming is an effective technique to attack optimization problems that possesses an optimal substructure, i.e. for which an optimal solution can be computed from optimal solution to its smaller subproblems. Dynamic programming is typically applied when the same subproblem appears multiple times when a problem is decomposed; storing a solution to each subproblem avoids re-computation when it arises again. In this section, we show why we use dynamic programming, and how we find an optimal partition and construct the solution to the problem of test delivery.

In order to obtain an optimal solution, we have to select a sequence of separators that create a partition of size K. To simplify the following discussion, we are not considering the effect of pin width on test time, so we assign a fixed test time to each core. The extension of the algorithm to cover pin distribution is subsequently presented in section 2.4.

Consider a mxn grid. Suppose that this grid has to be partitioned into two regions. In this case, only one separator is needed. If placed vertically, the separator can be placed at $n-1$ different positions. Similarly, $m-1$ possible positions for placing a horizontal separator accounts for a total of $m+n-2$ different positions. In order to minimize the test time, we compute the overall test time for each position of the separator and report the position resulting in the minimum test time. Increasing the partition size K to three requires an additional separator to be placed. If the first separator l_1 is placed as shown in Figure 2.8(a), the second separator l_2 can either be placed in Region **A** or Region **B**. There are $m + a - 2$ ways to place l_2 in **A**, and $m + n - a - 2$ ways to place in **B**, resulting in a total of $2m + n - 4$ ways. Thus for each choice of a position for l_1, there are several choices for placing l_2. The analysis can be continued in this way for larger values of K.

Suppose we are given an optimal partition for the grid and the position of l_1 is as shown in Fig. 2.8. The cost of **N** (π_N) is the maximum of the costs of the two regions, **A** and **B**. Let us consider three cases: a) the cost of region **A**, π_A, is greater than that of region **B** (π_B), i.e., $\pi_A > \pi_B$; b) $\pi_B > \pi_A$; c) $\pi_A = \pi_B$. In the first case, π_A should be the optimal cost for **A**. If not, optimal cost for **A** can replace π_A to give a better value for π_N, which is a contradiction because we assumed π_N to be optimal. A similar argument holds for the symmetrical second case: if π_B is not optimal, i.e., if there were a better value for π_B, we can always substitute this value to yield a better value of π_N, which would be a contradiction. The third case can be analyzed in the same way to conclude that an optimal solution for this partitioning problem encapsulates optimal solutions for its subproblems. This property indicates optimal substructure, which is a basic requirement for applying dynamic programming. Let us recursively express this property for the above example:

$$\pi_N = \max\{\pi_A, \pi_B\} \tag{2.1}$$

An optimal value of π_N can be obtained by sweeping l_1 across its all possible positions. For each different position ρ of l_1, the values for π_A and π_B changes. We can rewrite Equation (1) as follows:

$$\pi^\star = \min_{\rho}\{\max\{\pi_A, \pi_B\}\} \tag{2.2}$$

where $\pi^\star$ is an optimal solution to the actual problem and ρ varies over all the possible locations for placing l_1. Note that if the grid is to be partitioned into K regions, the sum of the partition sizes of the regions **A** and **B** will be K. The partition sizes of **A** and **B** are not accounted for in the above equation. The partition cost is a function of partition size. Since changing the partition size of a region has an impact on overall partition cost, the value of $\pi^\star$ depends on how many separators we assign to regions **A** and **B**. For a given P, let us use $\tilde{\pi}_R(k)$ to denote the cost of **R** when it is partitioned in k regions. Factoring in the partition sizes for **A** and **B**, Equation (2) becomes:

$$\pi^\star = \tilde{\pi}_N(K) = \min_{\rho}\{\min_{x}\{\max\{\tilde{\pi}_A(K-x), \tilde{\pi}_B(x)\}\}\} \tag{2.3}$$

where $0 < x < K$.

Let us formulate a general equation for all possible subproblems. For any region $\mathbf{R} \equiv [i, j, l, w]$, we rewrite Equation (3) by splitting it into two cases: the case when a separator is swept from top to down, and the case when it is swept from left to right. These two cases and the regions so formed are marked in Fig. 2.8(b) and Fig. 2.8(c). The final cost is the minimum cost obtained in these two sweeps.

$$\begin{aligned}\tilde{\pi}_R(k) \quad = \quad & \min\{\min_{\omega}\{\min_{x}\{\max\{\tilde{\pi}_{A^\#}(k-x), \tilde{\pi}_{B^\#}(x)\}\}\}, \\ & \min_{\lambda}\{\min_{x}\{\max\{\tilde{\pi}_{A^*}(k-x), \tilde{\pi}_{B^*}(x)\}\}\}\}\end{aligned} \tag{2.4}$$

where $0 < \omega < w$, $0 < \lambda < l$, $0 < x < k$, $\mathbf{R} \equiv [i, j, l, w]$, $\mathbf{A}^\# \equiv [i, j, l, \omega]$, $\mathbf{B}^\# \equiv [i+\omega, j, l, w-\omega]$, $\mathbf{A}^* \equiv [i, j, \lambda, w]$, and $\mathbf{B}^* \equiv [i, j+\lambda, l-\lambda, w]$.

We use Equation (4) in our algorithm. Since a subproblem is required to be solved in more than one larger problem, we use dynamic programming to solve this problem. The key idea is to store results for all subproblems and find the optimal cost $\pi^\star$, which is same as $\tilde{\pi}_N(K)$. Next we discuss the algorithm and the data structures used.

```
/*Enumerating all rectangles*/
for w in 1···n, l in 1···m, i in 0···n − w, j in 0···m − l do
  /*computing results for all partition sizes*/ STATE for k ← 2 to K
  computeAndStore(M, i, j, l, w, k)
end for
```

FIGURE 2.9: The top-level dynamic programming procedure for partitioning.

2.3.4 Algorithm

The algorithm begins by enumerating all possible rectangles in **N**, and for each such rectangle, computing and storing the optimal cost for all partition sizes k, $0 < k \leq K$. This is shown in Figure 2.9. The variables l and w are used to vary the length and width of the current rectangle, respectively. The pair (i, j) denotes the bottom-left coordinate of a rectangle. A five-dimensional integer array M is used for storing the optimal cost of all rectangles and for all partition sizes. The first four dimensions are used for identifying a rectangle, and the last dimension denotes the partition size. The variable k iterates over all partition sizes less than equal to K. The trivial case of $k = 1$ is not shown in the figure and $M[i, j, l, w, 1]$ is initialized to the sum of the test times of all cores present in the rectangle $[i, j, l, w]$. The procedure `computeAndStore`, shown in Figure 2.10, describes how the optimal solution is reached.

The element $M[i, j, l, w, k]$ is initialized to a large integer value and it is updated with a smaller value found on sweeping the separator, first from left to right, and then from top to bottom. The variable v stores the current position of the vertical separator. The optimal cost of the current rectangle depends on the partition cost of the two new subregions formed by the separator. The partition cost of subregions, in turn, depends on their respective partition sizes. We use a variable x for varying the partition size in one of the subregions. If x is the partition size of one of the subregions, then $k - x$ is the partition size for the other subregion.

```
computeAndStore(M, i, j, l, w, k)
M[i, j, l, w, k] ← ∞
/*placing a vertical separator at index v*/
for v ← i + 1 to i + w − 1  do
    /*Varying partition sizes in the two subregions created by the current separator.
    The partition size for one subregion is set to x, and for the other it is set to
    k − x*/
    for x ← 1 to k − 1  do
        t ← max(M[i, j, l, v − i, k − x], M[v, j, l, i + w − v, x])
        if M[i, j, l, w, k] > t then
            M[i, j, l, w, k] ← t
            B[i, j, l, w, k] ← [v, "v", k − x, x]
        end if
    end for
end for
/*Similar logic for placing a horizontal separator at index h*/
for h ← j + 1 to j + l − 1  do
    for x ← 1 to k − 1  do
        t ← max(M[i, j, h − j, w, k − x], M[i, h, j + l − h, w, x])
        if M[i, j, l, w, k] > t then
            M[i, j, l, w, k] ← t
            B[i, j, l, w, k] ← [h, "h", k − x, x]
        end if
    end for
end for
```

Figure 2.10: `ComputeAndStore` Procedure

```
ConstructSolution(B, i, j, l, w, k)
if k = 1 then
    print [i, j, l, w]
end if
position ← B[i, j, l, w, k].position
sizeA ← B[i, j, l, w, k].sizeA
sizeB ← B[i, j, l, w, k].sizeB
if B[i, j, l, w, k].orientation = "h" then
    /*Horizontal separator*/
    ConstructSolution(B, i, j, position − j, w, sizeA)
    ConstructSolution(B, i, position, j + l − position, w, sizeB)
else
    /*Vertical Separator*/
    ConstructSolution(B, i, j, l, position − i, sizeA)
    ConstructSolution(B, position, j, l, i + w − position, sizeB)
end if
```

Figure 2.11: `ConstructSolution` Procedure

2.3.5 Constructing the solution

In addition to producing the cost of an optimal partition, we also have to keep track of the sequence of separators that was chosen to construct this optimal solution. For that we store the required information in an array B. Each element of the five-dimensional array B is a four-tuple structure:

`<position, orientation, sizeA, sizeB>`

The position where the separator is placed is stored against the tuple element *position*, and the orientation of the separator, whether horizontal ("h") or vertical ("v"), is stored against *orientation*. A separator divides a rectangle into two subregions. The tuple elements *sizeA* and *sizeB* are used for storing the partition sizes for these two subregions. We can determine in advance the correspondence between elements and regions; *sizeA* can be associated with the left subregion, in case of a vertical separator, or with the bottom subregion, in case of a horizontal separator, while *sizeB* can be associated with the other remaining region. The procedure for constructing the solution is shown in Figure 2.11. This is a recursive procedure and the first call to this procedure is invoked with the input parameter set: $(0, 0, m, n, K)$.

2.3.6 Computational complexity analysis

For each possible value-pair of length l and width w, there are $(m-l+1)\cdot(n-w+1)$ rectangles. Therefore, the total number of enumerated rectangles is

$$\sum_{l=1}^{m}\sum_{w=1}^{n}(m-l+1)\cdot(n-w+1)$$

For each rectangle $\mathbf{R}$, $\tilde{\pi}_R(k)$ is computed for each partition size k, where $1 < k \leq K$. For each value of k, separators are placed at $l+w-2$ different positions to compute the optimal position. For each position of the separator, partition sizes in the resulting subregions are varied from 1 to $k-1$. Therefore, the time complexity can be obtained by computing the following expression:

$$\sum_{l=1}^{m}\sum_{w=1}^{n}\sum_{k=2}^{K}(m-l+1)\cdot(n-w+1)\cdot(l+w-2)\cdot(k-1)$$

$$= \begin{cases} \mathrm{O}(m^3n^2K^2), & \text{if } m \geq n \\ \mathrm{O}(m^2n^3K^2), & \text{otherwise} \end{cases}$$

which are simple polynomials of the input parameters, namely the NoC size and the size of the partition, thereby highlighting scalability of the proposed solution. Note that for the case $k = 1$, no further subdivision of the current rectangle is required and $M[i, j, l, w, 1]$ can be computed in $\mathrm{O}(m^2n^2)$ with an efficient implementation. The space-complexity equals the storage needed for the arrays M and B, which is $\mathrm{O}(m^2n^2K)$.

2.4 Additional Constraints and Enhancements

In this section, we incorporate the boundary-region constraint, the pin-assignment strategy, and power constraints into our solution approach.

2.4.1 Boundary-region constraint

The boundary-region constraint mandates the inclusion of only boundary regions in the final solution. Note that for any such region having more than one boundary edge, placing a separator does not create a blocked region. For the boundary regions having only one boundary edge, further subdivision may lead to a blocked region, depending on the orientation of the boundary edge and on the separator orientation. We must avoid such a division. The region **D** in Figure 2.7(a) has only one boundary edge, and it is coincident on the axis $y = 0$. While any horizontal separator creates a blocked subregion, a vertical separator will create subregions having one boundary edge each. A similar conclusion can be drawn for the symmetric case where a region has only one boundary edge that is parallel to the axis x = 0. In a symmetric manner, for a region with its only boundary edge parallel to the axis x = 0, a horizontal separator will create subregions with a boundary edge in both subregions. The vertical separator fails to obey the boundary-region constraint in this case. The above observations can be easily added to our proposed dynamic programming approach. An *if* condition is added between the Line 2 and Line 3 of the procedure `computeAndStore` in figure 2.10 to prevent the placement of any vertical separators if the current rectangle has only one boundary edge that is parallel to the y-axis of the coordinate plane. Similarly, an *if* condition is added between the Line 13 and Line 14 of the same procedure to prevent the selection of any horizontal separator that does not obey the boundary-region constraint.

2.4.2 Pin-assignment problem

The test time of a core varies with the TAM width assigned to it. In this work, we do not consider the case when the TAM width of a core exceeds the channel width. To incorporate the effect of pin count on the partition cost, we need to add one more dimension to the search space. The overall partition cost can now be expressed as $\pi_N(K, P)$ where P has to be distributed among all the K regions. The optimal substructure equation can be rewritten as:

$$\pi^\star = \pi_N(K,P) = \min_{\rho}\{\min_{x,p}\{\max\{\pi_A(K-x,p), \pi_B(x,P-p)\}\}\},$$

where $0 < x \leq K$, $0 < p < P$ and ρ varies over all possible positions of the separator being placed. This equation can be further extended to consider the cases of horizontal and vertical separator, as in Equation (4), but is being omitted here for the purpose of brevity. For solving this problem, we evaluate the optimal solution using the same approach as in Section 2.3. The dimension of array M is increased to six such that $M[i,j,l,w,k,p]$ stores the optimal partition cost of region $[i,j,l,w]$, if the partition size is k and the number of pins assigned to this region is p. For each value of k, we add one more loop that iterates over all the pin-counts possible. Moreover, when a separator is placed, in addition to varying the size of partitions of the subregions **A** and **B**, all possible distributions of the p pins to the two subregions are evaluated. For any region $\mathbf{R}$, $\pi_R(k,p)$ is not evaluated for $p < k$, because each region should be allotted at least one pin. We use the equality, $\pi_R(k,p) = \pi_R(k,p{-}1)$, if p takes values that are greater than k times the flit width. The array B is also a six-dimensional structure now with each element being a six-tuple structure. Two more tuples are added to specify the pins allotted to each subregion.

Adding an extra dimension of pin assignment to our problem increases the run-time complexity by a factor of P^2. The space-complexity also increases by a factor of P.

2.4.3 Power constraints

The reduction of test application time by manipulating power profiles of test sets has been studied in [70]. Given a power constraint, the problem of minimizing the test time is presented in [70]. A similar problem in the context of NOC has been formulated and solved using ILP techniques in [71]. The scheduling problem under power constraints was shown to be computationally harder than the general test scheduling problem in [71], and the proposed method does not scale well with the number of cores. This section elaborates on ideas presented in [70] to initially treat power as a design objective to minimize power consumption, and then subsequently as a design constraint to achieve minimization of the test application time.

Profiling of power consumption: Accounting for power during test scheduling entails the modeling of power consumed by the individual cores. A simplistic ap-

`createPowerProfile` ($\mathbf{R}$) //$\mathbf{R}$ is a leaf region

$List \leftarrow$ A list of cores present in the region $\mathbf{R}$ sorted in decreasing order of power consumed.
// time($List(i)$) : test time for the i^{th} core in $List$
// power($List(i)$) : power consumption of i^{th} core in $List$
// $ncores$: total number cores in $\mathbf{R}$
P_{hi}^{R} = power($List(1)$)
$min\_power \leftarrow \infty$; $test\_time \leftarrow \sum_{i=1}^{ncores}$ time($List(i)$); $l_{hi} \leftarrow 0$
for $i \leftarrow 1$ **to** $ncores$ **do**
  $pwr \leftarrow P_{hi}^{R} \times l_{hi}+$ power($List(i)$) $\times (test\_time - l_{hi})$
  if $min\_power < pwr$ **then**
    $min\_power \leftarrow pwr$; $P_{lo}^{R} \leftarrow$power($List(i)$)
    $L_{hi}^{R} \leftarrow l_{hi}$; $L_{lo}^{R} \leftarrow test\_time - l_{hi}$
  end if
  $l_{hi} = l_{hi}+$ time($List(i)$)
end for

FIGURE 2.12: `createPowerProfile` procedure.

proach is to flatten the power profile of a core to the worst-case instantaneous power consumption value, i.e., its peak value [70]. The simplicity and reliability of this model, called the *global peak power approximation model* (GP-PAM), is achieved at the cost of including significant *false power* in the model; false power is the power that is not consumed, but still being considered. This limits the degree of test concurrency that can be ideally achieved. The GP-PAM model is represented by a pair $[P_{global}, L_{global}]$, where P_{global} is the global peak power consumed over a test length of L_{global} cycles. Another model, proposed in [70], significantly reduces false power that enables greater test concurrency. This model, known as 2LP-PAM, creates a profile having two peaks, as opposed to a single global peak in GP-PAM. 2LP-PAM is represented by a vector pair $\{[P_{hi}, L_{hi}], [P_{lo}, L_{lo}]\}$. The entire test length of a core is divided into two segments; P_{hi} is the peak power for a test length of L_{hi}, peak power P_{lo} for a test length of L_{lo}, and $P_{hi} \geq P_{lo}$. The splitting of the power profile provides more flexibility in scheduling cores under a power constraint because it approximates false power more accurately than the GP-PAM model.

We adapt the benefits of 2LP-PAM in our DP approach, and discuss how to

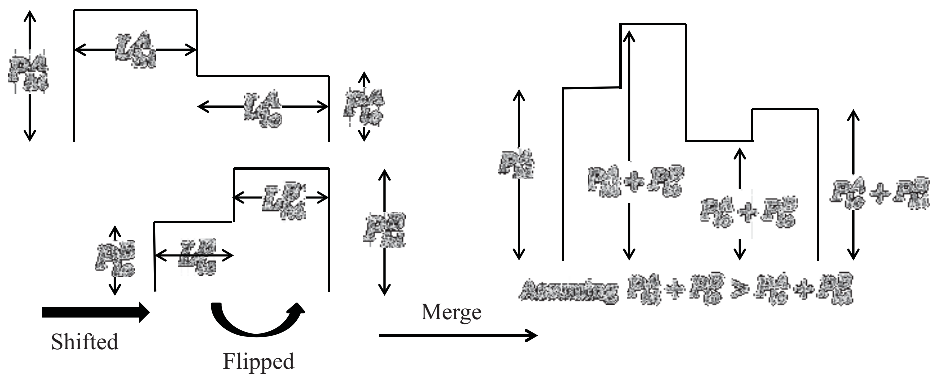


FIGURE 2.13: Merging of power profiles.

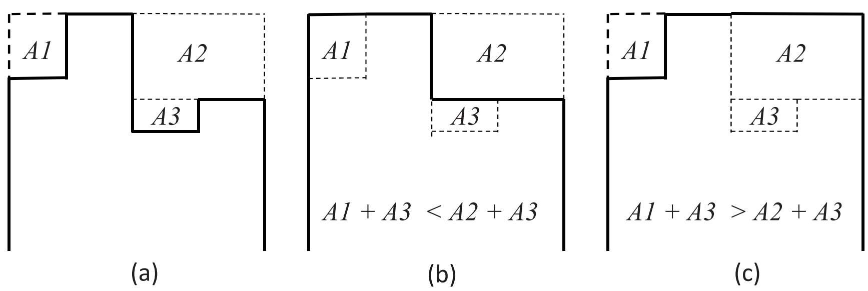


FIGURE 2.14: Transformation of a four-peak profile to a profile having two peaks.

construct power profile of a region — a collective profile of all the cores present in the region. We show how this technique helps in approximating power consumption and creating room for test concurrency between cores from different regions under a power constraint. Note that all cores in a given region are still scheduled sequentially, but the order in which they are scheduled affects the ability to simultaneously schedule cores in other regions. The power profile for a leaf region **R** is created by the procedure `createPowerProfile`, as outlined in Figure 2.12. The procedure creates the two peaks P_{hi}^R and P_{lo}^R, such that the area under the profile, $P_{hi}^R \times L_{hi}^R + P_{lo}^R \times L_{lo}^R$, is minimized. All the cores are sorted in decreasing order of the power that they consume during testing. Note that we assume uniform power consumption within

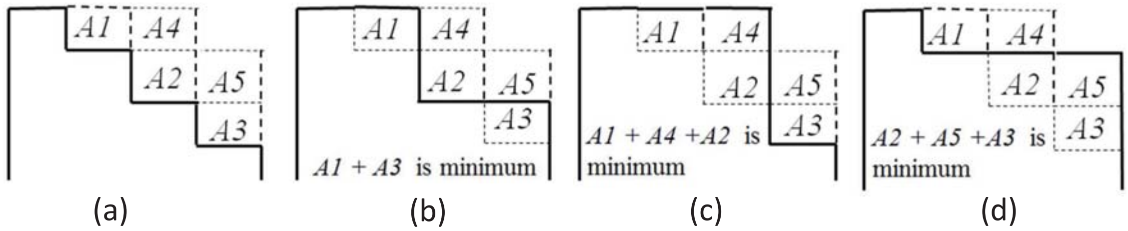


FIGURE 2.15: Another example of profile transformation.

a core during the time it is tested. The `createPowerProfile` procedure takes the sorted list of the cores present in a leaf region **R**, and finds an index i into the list that minimizes the area under the resulting profile. P_{hi}^{R} is the power consumption of first core in the list. The list is sequentially scanned to find the index i; L_{hi}^{R} is set to the sum of the test times of the cores indexed one to $i-1$ in the list, and L_{lo}^{R} takes the value of sum of the test times of the remaining cores. P_{lo}^{R} is the power consumption of the i^{th} core in the list.

Merging power profiles: We next explain how we merge power profiles of two regions to obtain power profile of the bigger rectangle under a given power constraint to minimize test lengths under different scenarios. Given are two power profiles $\{[P_{hi}^{A}, L_{hi}^{A}], [P_{lo}^{A}, L_{lo}^{A}]\}$ and $\{[P_{hi}^{B}, L_{hi}^{B}], [P_{lo}^{B}, L_{lo}^{B}]\}$ for regions **A** and **B**, respectively. For merging these profiles, such that the resultant profile does not exceed the power limit PL, we first select the region with lower test time (**B** in this case), and then shift and flip (mirror image) its power profile; refer to Figure 2.13. Shifting the profile ensures that cores of **B** are scheduled as late as possible, and flipping it ensures that the addition of P_{hi}^{A} and P_{hi}^{B} is avoided as much as possible, thereby leading to minimization of the power consumption of the two regions taken together. The creation of schedules for cores is abstracted out with shift and flip operations on profiles, and this helps in minimizing power consumption.

The shape of the resulting profile depends on the values of the peaks of the profiles that are merged, and the power constraint. For our running example, the merged profile can look like the profile with four peaks (Fig. 2.13). To be consistent with the two-peak power model, the number of peaks has to be reduced to two. In Fig. 2.14(a), different areas are marked that have to be collapsed with the original profile to obtain two peaks. Depending on the magnitude of the areas marked, the final profile can look like Fig. 2.14(b) or Fig. 2.14(c). The decision on which areas to collapse depends on the area under the profile, and the profile with the least area is chosen.

We next discuss the impact of power constraints on the merging of profiles. In the case when a power violation occurs, the profile for **B** is shifted until no violation is caused. If the two lower peaks from the two regions exceed the power limit PL, i.e., $P_{lo}^{A} + P_{lo}^{B} > PL$, then the two power profiles cannot overlap. In such a case, we sort the four peaks in descending order and place them side-by-side to create a profile shown in Figure 2.15(a). Different combinations of the marked areas are grouped together to reduce the number of peaks to two. The final profile can be any one of the profiles as shown in Figure 2.15(b)-(d).

We have implemented a dedicated transformation procedure for all kinds of profiles that can result from the merge operation. The enumeration of these cases and the corresponding transformations are straightforward and therefore omitted.

Integration with DP: The proposed method for power profile manipulation has been integrated with our DP approach. When a separator is placed in a region, rather than taking the maximum of the testing times of the two subregions as the test time for that region, we compute the new test time as the test length of the power profile obtained after merging the power profiles of the two subregions. Line

6 in Figure 2.10 is modified accordingly. Since any two power profiles are merged without violating power constraint, power profile of the NOC **N** does not violate the constraint.

The run-time complexity of the approach under power constraints remains unaffected. The procedure to merge two profiles takes a constant time for computing a merged profile. When creating a power profile for a leaf region using `createPowerProfile` procedure, the complexity increases by a factor of mn (an efficient implementation using DP increases the complexity by a factor of $\min(m, n) \cdot \log(mn)$), but this part only constitutes the initialization step and does not dominate the runtime of the main algorithm.

2.4.4 Reducing the run-time complexity by a factor of P

The run-time complexity of the proposed approach was found to be a function of P^2. The number of pins P is much larger in magnitude when compared to other parameters such as m, n, or k. We next show how the factor of P^2 can be reduced to P. The approach presented in the previous section distributes pins on both sides of the separator such that the sum of the pins on both sides is equal to a given pin count p, and selects the distribution that minimizes test time. All possible combinations are tried making the search exhaustive. This is repeated for all possible values of p; therefore, a factor of P^2 is seen in the computational complexity. Exhaustive search can be reduced to selective search by observing that given a position of separator, and an optimal distribution of p pins in the two subregions created by the separator, the optimal distribution for a count of $p+1$ pins is obtained by assigning the newly added pin to the subregion having greater test time.

The modified `computeAndStore` procedure that implements selective search is shown in Figure 2.16. While computing the minimum test time for a non-leaf region, we maintain an optimal distribution of pins in the two subregions for each position `computeAndStore`(M, i, j, l, w, k)

for v in $i+1 \cdots i+w-1$, x in $1 \cdots k-1$

*/*This is the optimal distribution for the base case (p = k). p1 pins to one subregion and p2 to the other.*/*

$p1 \leftarrow x$; $p2 \leftarrow k-x$;

for $p \leftarrow k$ **to** P

$t1 \leftarrow M[i, j, l, v-i, x, p1]$

$t2 \leftarrow M[v, j, l, i+w-v, k-x, p2]$

$t \leftarrow \max(t1, t2)$

if $M[i, j, l, w, k, p] > t$

$M[i, j, l, w, k, p] \leftarrow t$

$B[i, j, l, w, k, p] \leftarrow [v, \text{"}v\text{"}, x, k-x, p1, p2]$

//Assigning the next pin to the subregion having greater test time.

if $t1 \geq t2$ **then** $p1$++ **else** $p2$++

FIGURE 2.16: Modified `computeAndStore` for reducing the run-time complexity.

of separator v, for each pin-count p, and partition size x of one of the subregions (the partition size of the other sub-region is computed as $k - x$, where k is the partition size of the parent region for which we are computing the minimum test time). For the base case when the pin-count p equals k, the number of pins assigned to the subregions equals their respective partition sizes, i.e., x and $k - x$. This is the only possible (and hence optimal) distribution because each leaf region has to be driven by at least one pin. The arrays M and B are updated whenever a better result is found. A similar procedure is used for sweeping horizontal separators.

2.5 Test Scheduling using Multicast Routing

A common theme pervading prior work on testing NOC-based SOC is the transmission of test data through *unicast routing*—a packet is sent from one source node to only one destination node. In the presence of identical cores, or cores of the same type, test patterns can be routed to identical cores simultaneously using *multicast routing*, where test data is sent from one source node to multiple destination nodes. While collecting test responses, we can use MISR-based output compaction (similar to [33]) to send signatures of individual cores to the ATE. In this way, the same tester channels can be used for both streaming of test input data and receiving MISR signatures. In the absence of response compaction, the output response for every pattern has to be shifted out serially, and for each core individually, thereby negating the benefits of multicast routing. Therefore, we advocate the use of MISR-based response compaction.

2.5.1 Routing in NOCs using multicast routers

- In a *tree-based multicast*, a flit travels along a common path and forks (replicates) when necessary to achieve the shortest routing path to each destination in the destination set. Before sending a multicast message, a multicast tree is first constructed in the network by sending unicast packets to every destination. These unicast packets update the routing table at each router along their paths with information related to the identifier of the impending multicast message and the path that should be taken by the multicast message. This strategy leads to a creation of a multicast tree as shown in Figure 2.17(a). Suppose a multicast has to be initiated from node A to nodes B, C and D. Assuming that the underlying routing protocol is XY[3], unicast packets to nodes B, C, and D mark the routing tables at nodes E, F, and G, respectively, as the nodes where forking or replication of the impending multicast message has to happen.Note that the time taken to deliver a multicast message to all destinations is equal to the time taken by the farthest destination to receive the message.

- In a *path-based multicast*, a multicast flit is first routed to the nearest destination in the destination set. It is then delivered to the destination core and a copy is forwarded to the next nearest destination in the destination set. Figure 2.17(b) illustrates an example of this type of multicast routing.While this approach replicates flits only at the destination nodes, in the tree-based multicast

[3] In the XY-routing protocol, data is first routed along the X and then along the Y axis.

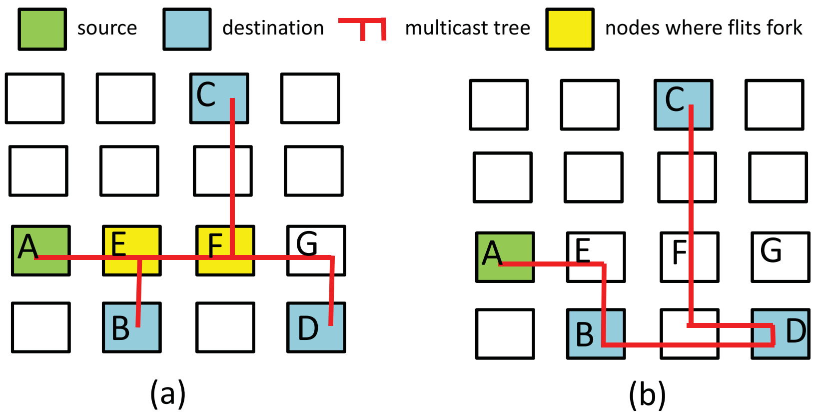


FIGURE 2.17: An illustration of (a) tree-based multicast, and (b) path-based multicast.

routing, the replication may occur at intermediate nodes that incur minimal routing overhead.

Both the above methods can facilitate delivery of test patterns as streaming data, but the latter takes more time than the former approach; therefore, we assume tree-based multicast for our discussion. A discussion on the design of routers that support multicasting is beyond the scope of this work. Recent advances on the design of multicast routers can be found in [72–74].

2.5.2 Problem formulation

The cumulative test time of cores assigned to an access point is redefined in the context of multicast routing. The time to test cores through an access point a_1 is defined as the sum of the test times of the cores, taken exactly one from each type, that are assigned to a_1 and are farthest from a_1, in addition to the time required to send signatures of every core assigned to a_1 back to a_1. The SOC test time is the maximum time taken to test cores using any of the access points.

We formulate the problem in two ways.

Contention-free assignment (CFA) problem: The CFA problem discussed in

Section 2.1 is adapted from the CFA problem presented in [33], which we refer to as CFA* henceforth. The CFA* problem, which was shown to be NP-complete in [33], is identical to the CFA problem except that the former does not use multicast routing. A given instance of CFA* can be reduced to an instance of CFA using the following transformation. Take an instance of CFA* and mark every core in it to be different from every other core. Next, create one copy each for every core in the instance of CFA* and place it with the original core side-by-side to create a bigger grid. This step doubles the number of cores, and hence the dimension of the grid along one axis. Assuming that we transport test data and response to the newly constructed NOC using multicast routing, the test scheduling problem is still NP-complete because finding an optimal test schedule for the original cores in the new grid is still computationally as hard as it was for the original grid.

Rectangular partitioning: Since CFA is NP-complete, it cannot be optimally solved in polynomial time. We use the rectangular partitioning formulation for grid topology presented in Section 2.3 for solving the problem optimally under the constraint of creating rectangular regions. The test time of a leaf region is redefined as the sum of the cycles needed to shift test patterns in and out of the test wrappers of cores, one from each type, that are present in the leaf region. Since we use MISR-based output compaction, the test time of a leaf region also includes the cycles needed to collect the test response from every core from the leaf region.

In contrast to optimally partitioning the set of available links into pairwise-disjoint subsets in the former problem (CFA), the second problem restricts the search only to rectangular partitions—those partitions that create rectangular regions. The latter formulation is a special case of the former. In the first problem, the positions of the access points are provided as an input to the problem, but in the second, those positions are to be determined. To avoid any inconsistency, we first obtain access point locations using a solution to the second problem, and use the same locations for

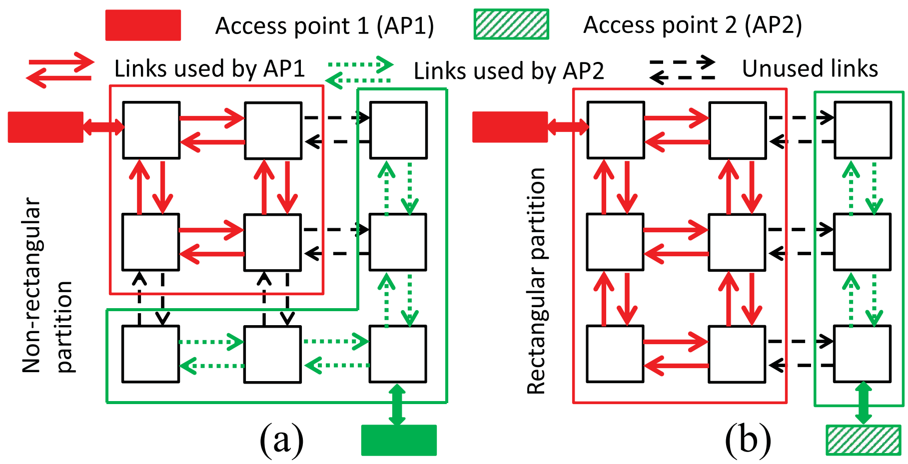


FIGURE 2.18: A possible partition on the set of links (a) using CFA; (b) using rectangular partitioning.

the first problem. Figure 2.18 contrasts the two formulations by depicting partitions that are possible with them. Note that we can get the partition shown in 2.18(b) using CFA as well.

2.5.3 ILP Formulation

In this section, we model the CFA problem using integer linear programming (ILP) to leverage commercially available ILP solvers for obtaining an optimal solution.

Data structures: The proposed model expands the ILP models discussed in [33]. Let $\mathbf{A}$ denote the set of access points, $\mathbf{C}$ denote the set of cores, and $\mathbf{W}$ and P represent the set of pin widths available for access points and the total number of available test pins, respectively. For each core $c \in \mathbf{C}$ and each pin width $w \in \mathbf{W}$, $TL_{c,w}$ encodes the test length, or the number of cycles needed to shift test data in and out of the test wrapper for core c using w pins. Similarly, $SL_{c,w}$ is the number of cycles needed to shift out the signature for core c using w pins. Two additional cycles are added in both cases to model header and tail flits. The symbol $PD_{c,a}$ stands for the path delay, or the number of cycles required for establishing a path from core c to access point a (or vice versa). In line with previous work on using the NOC as TAM, we assume a delay of three cycles per router, which also includes the source and destination routers of the path.

To model the different types of cores available, we have a set of core types $\mathbf{T}$, and a symbol $type_c$ to indicate the type of core c. Note that $type_c$ is an element of the set $\mathbf{T}$, and we use the expression $type_c = type_d$ to state that the cores c and d are of the same type. For cores $c, d \in \mathbf{C}$ and access point $a \in \mathbf{A}$, the entry $CCA_{c,d,a}$ in a pre-computed table $\mathbf{CCA}$ is 1 only if $type_c = type_d$ and the Manhattan distance from c to a is greater than that between d and a. If both the cores are located at the same distance from a, an arbitrary choice between the two cores is made to resolve the resulting conflict.

Descision variables: A matrix x of binary variables (indicator variables) tracks the assignments of cores to access points. The variable $x_{c,a}$ is 1 only if core c is assigned to access point a. Since each core is assigned to only one access point,

$$\forall c \in \mathbf{C} : \sum_{a \in \mathbf{A}} x_{c,a} = 1.$$

Another two-dimensional table p stores the binary variables $p_{a,w}$ to represent access point a is assigned w test pins. Because the total pins assigned to all access points cannot exceed P,

$$\sum_{a \in \mathbf{A}, w \in \mathbf{W}} w \cdot p_{a,w} \leq P.$$

Each access point is assigned a fixed number of pins; therefore,

$$\forall a \in \mathbf{A} : \sum_{w \in \mathbf{W}} p_{a,w} = 1.$$

A set of binary variables (tp_assgn) indicates assignment of core types to access points. The binary variable $tp\_assgn_{t,a}$ is set to 1 only if at least one core of type

$t \in \mathbf{T}$ is assigned to access point a. This constraint can be expressed as

$$tp\_assgn_{t,a} \geq x_{c,a}, \forall c \in \mathbf{C}, \forall a \in \mathbf{A},$$

where $t = type_c \in \mathbf{T}$.

If multiple cores of the same type is assigned to an access point a, then only the farthest of these cores from a is used to compute the test time for shifting in the test data to the test wrappers of these cores using a. The binary variable $z_{c,a}$ takes a value of 1 only if c is the farthest of all cores of the same type that are assigned to a. The inequality,

$$\forall c \in \mathbf{C}, \forall a \in \mathbf{A} : z_{c,a} <= x_{c,a}$$

, expresses the condition that $z_{c,a}$ cannot be 1 if core c is not assigned to access point a. Also, the inequalities

$$\forall t \in \mathbf{T}, \forall a \in \mathbf{A} : 1 \geq \sum_{c \in \mathbf{C}, type_c = t} z_{c,a} \geq tp\_assgn_{t,a}$$

state that only one core of type t is used to calculate the test time through access point a if there exists a core of type t that is assigned to a.

Out of two cores c and d of the same type and if c is farther than d from access point a, i.e., if the table entry $CCA_{c,d,a}$ is 1, the conditions (i) if $x_{c,a} = 1$, $z_{d,a} \leq 0$ and (ii) if $x_{c,a} = 0$, $z_{d,a} \geq 0$ are imposed. These if-conditions can be easily linearized using standard techniques.

Our problem formulation is restricted to finding only those schedules that do not have any link overlapping between any pair of the sets of links used to communicate between access points and the cores assigned to them. Therefore, similar to [33], we also have a contention table **CT** that stores a 1 against the position c, a, d, b if any packet travelling between a and c shares links with any packet travelling between b and d, where $a, b \in \mathbf{A}$ and $c, d \in \mathbf{C}$. In order to avoid link contention, the following inequality is added to the model:

$$\forall c, d \in \mathbf{C}, c \neq d; \forall a, b \in \mathbf{A}, a \neq b; CT_{c,a,d,b} = 1 : x_{c,a} + x_{d,b} \leq 1.$$

This constraint ensures that in a feasible solution, at most one assignment is chosen out of assignments that can create link contention.

The variable $ctap_{c,a}$ (core time on access point) stores the time required to test core c using access point a. This is expressed using the equation

$$ctap_{c,a} = \sum_{w \in \mathbf{W}} p_{a,w} \cdot TL_{c,a} + PD_{c,a}.$$

Similarly, for core c and access point a, the variable $stap$ stores the time to collect the signature that is generated after test patterns have been applied, which is given by the equation:

$$stap_{c,a} = \sum_{w \in \mathbf{W}} p_{a,w} \cdot SL_{c,a} + PD_{c,a}.$$

The total test time $TEST\_TIME$ is the maximum time required by any access point to test all the cores assigned to it using the number of pins allotted to it. This can be expressed with the help of the inequalities:

$$\forall a \in \mathbf{A}, \forall c \in \mathbf{C} : ta_{c,a} \geq ctap_{c,a} - (1 - z_{c,a}) \cdot M, \; sa_{c,a} \geq stap_{c,a} - (1 - x_{c,a}) \cdot M,$$

and

$$\forall a \in \mathbf{A}, TEST\_TIME \geq \sum_{c \in \mathbf{C}} (ta_{c,a} + sa_{c,a}).$$

The parameter M in the above inequalities is pre-computed to a value much larger than the expected maximum test time. As a consequence, the term $(1 - z_{c,a}) \cdot M$ and $(1 - x_{c,a}) \cdot M$ forces the intermediate variables $ta_{c,a}$ and $sa_{c,a}$, respectively, to zero if core c is not assigned to access point a.

Since this is a minimization problem, the objective is set to minimize $TEST\_TIME$.

2.5.4 Dynamic programming approach for rectangular partitioning

We use the dynamic programming approach discussed in Section 2.3. The only difference between the two problems (with and without multicasting) is the way in which the test time of a leaf region is computed. For the problem discussed in 2.3, the test time for a leaf region is derived by adding the test time of every core belonging to the leaf region. In the current problem, first we identify the cores, one of each type in the leaf region, that are farthest from the access point assigned to the region. Then we add test cycles that will be needed to shift test data in and out of the core wrappers of the identified cores. The cycles needed to test all other cores in the leaf region is ignored because all cores of the same type are tested in parallel. In addition, the cycles needed to send signatures of every core back to the access point is added to the test time of the region.

2.6 Experimental Results

In this section, we first describe how we created our test cases. The rest of the section includes the following:

- For NOCs with arbitrary (irregular) topologies, we compare the results obtained from our approach based on subset-sum with an implementation of [1].

- For a mesh-based NOC, we compare the results obtained using DP to that obtained with ILP [33]. Since the comparison is based on the problem instances that can be solved with ILP in a reasonable time, we take small problem instances first. These test cases provide a good baseline with which we compare the quality of solutions produced by our approach.

- Results showing the effect of power constraints on the total test time is reported and compared with [71].

2.6.1 *Test scheduling without power constraints*

We used six SOCs from the ITC'02 SOC Test Benchmarks [75], namely d695, g1023, p34392, p22810, t512505 and p93791, with 10, 14, 19, 28, 31 and 32 cores, respectively. Since we are interested in evaluating performance for a large number of cores, we created additional SOCs by taking cores from the last two SOCs and replicating them. The test times for all TAM widths (constrained by flit width) for each core were obtained using the *design_wrapper* algorithm [18].

For mesh-based NOCs, we adopted the same approach as outlined in [33, 36, 40] for generating the topology of the NOC. We assumed XY-routing, a switching delay of three clock cycles per router for access point-to-core or core-to-access point path establishment, and one cycle each for transmitting header and tail flits. A flit width of 32 bits was assumed. For each leaf region, we placed an access point at the middle of its longest boundary edge, and computed the cumulative routing delay of the region as the sum of routing delays for all the cores located in that region. It can be easily shown that, by taking the derivative of the routing delay and equating it to zero, the routing delay is minimized when the access point is placed at this specific location of the region. The routing delay for a core during path establishment is three times its Manhattan distance from the access point assigned to that core. We executed the ILP model using the FICO XPress-MP Solver [76] that was also used in [33].

Each region contains exactly one access point, hence the number of regions (partition size) in our work and the number of access points, as reported in [33], are synonymous. All results were obtained on a 16-core Intel(R) Xeon(R) machine with a processor speed of 2.53 GHz, 64 GB memory and a cache size of 12288 KB.

In the first experiment, we compare our method for an NOC with an arbitrary topology with the method presented in [1]. Since the latter is based on TR-Architect

[2], it does not restrict the number of access points (ATE interfaces) that are used; it reports the number of access points that are required to minimize the test time. We modify $ModifiedCreateStartSolution$ procedure to initialize a solution with a fixed number of access points (K). Table 2.1 shows the results for the two methods for two NOC test cases. The column K^* does not restrict the number of access points for the method of [1]. The column '[1], K^*' reports the number of access points required (K^*) along with the minimized test time. We use K^* to derive the test time for the proposed method, which is shown in the last column. The NOC examples (*irreg1* and *irreg2*) are taken from [1] and they both consist of nine routers. Each router is assigned a core taken from the SOC d695. Since there are nine routers and d695 consists of 10 cores, we ignore one core, namely $Module5$, for this experiment. Note that the greedy procedure of pin distribution is optimal at every iteration; hence, solutions are obtained for a range of pin counts in a single run. The CPU time was found to be negligible in all cases. Our method consistently reports better solutions than [1].

Next we present results for a mesh-based NOC. We took a 6x6 NOC obtained from the SOC t512505 (31 cores) and compared the results obtained using the two approaches. The ILP proposed in [33] does not address the problem of optimal access point placement. Therefore, we show two different sets of results for the ILP model; one obtained by a random placement of access points (ILP*) and the other obtained by placing the access point at the locations determined by DP (ILP**). The results for different values of K and P are shown in Table 2.2. The column Δ^* (Δ^{**}) records the relative difference in the test time reported by dynamic programming (DP) over that obtained from ILP^* (ILP^{**}). If the test time obtained by ILP is T_{ILP^*} ($T_{ILP^{**}}$) and that obtained from DP is T_{DP}, then $\Delta^* = \frac{T_{DP}-T_{ILP^*}}{T_{ILP^*}} \times 100\%$, and $\Delta^{**} = \frac{T_{DP}-T_{ILP^{**}}}{T_{ILP^{**}}} \times 100\%$.

Table 2.1: Comparison of the test times (clock cycles) obtained using the proposed subset-sum based method with [1] for NOCs having any arbitrary topology.

Design	#pins	$K=2$ [1]	$K=2$ †	$K=3$ [1]	$K=3$ †	K^* [1], K^*	K^* †, K^*
irreg1	16	33859	31588	33242	30530	30680, 4	30312, 4
	24	28226	21412	22290	20631	21073, 4	20631, 4
	32	20516	19372	19848	16739	17613, 5	15303, 5
	40	18940	17704	18486	13690	13007, 5	12810, 5
	48	18793	17116	17716	13266	10751, 5	10678, 5
	56	18793	15773	17716	12303	9869, 6	9869, 6
	64	18793	15760	17107	11852	9869, 6	9869, 6
irreg2	16	39450	32485	31241	30117	31241, 3	30117, 3
	24	28729	21412	22449	20796	21579, 3	20796, 3
	32	19771	19372	19034	15338	15621, 3	15338, 3
	40	19679	17704	18564	12953	12846, 5	12846, 5
	48	19679	16827	14602	12941	10625, 5	10537, 5
	56	19679	15758	14454	11519	9869, 5	9869, 5
	64	19679	15758	14381	11153	9869, 5	9869, 5

†Proposed method

Negative values of "Δ" indicate that the solutions obtained from ILP are worse than that from DP. The same experiment is repeated for another 6x6 NOC obtained from the SOC p93791 (32 cores); the corresponding results are tabulated in Table 2.3. The CPU time for DP is less than 1 s.

The ILP solver was allowed to execute until it found an optimal solution. Note that an "optimal solution" for ILP* may be worse than a solution obtained using DP due to the non-optimal placement of access points. Placing the access points at the locations suggested by the proposed approach almost always resulted in reduced test time. These results demonstrate that the proposed DP approach can produce results that are very close to that obtained by ILP. The DP approach leads to optimal solutions under the constraint of rectangular partitions. The ILP approach can lead to better solutions than DP by allowing non-rectangular partitions.

Table 2.2: Results for t512505 (31 cores) for various values of K and P.

K	P	Test time (clock cycles), T				
		ILP*	ILP**	DP	Δ^*	Δ^{**}
2	48	5675985	5637744	5644306	−0.55%	0.11%
3	48	5275163	5228434	5256900	−0.34%	0.54%
	64	5275163	5228434	5256900		
4	48	5228434	5228428	5228428	0%	0%
	64	5228434	5228428	5228428		
	72	5228434	5228428	5228428		

ILP*: ILP with random placement of access points.
ILP**: ILP with placement of access points guided by the proposed technique.

Table 2.4 compares the test times obtained using various methods with TAM-independent lower bounds on test time that are derived using [2]. (Note that these lower bounds cannot always be achieved due to bottleneck cores [18].) The third column reports the lower of the test times obtained using ILP* and ILP**. Since the lower-bound expression in [2] is not tied to any particular TAM design and utilizes only the volume of test data that must be transported, these bounds are also applicable to the test-time minimization problem in NOCs. Moreover, an NOC imposes additional constraints on the classical TAM optimization problem; therefore, the lower bounds of [2] hold in our NOC-based TAM scenario as well, and we expect actual test times to be larger than the lower bounds. Nevertheless, closeness to lower bounds is a measure of the effectiveness of an optimization method. For the problem instances listed in Table 2.4, the test-time results are only 5%–13% higher than provable lower bounds.

We show similar results for a larger 14x14 NOC obtained by replicating cores from the SOC benchmark p93791; see Table 2.5. We set a time limit of three hours for ILP for each value of K and P, and report the best intermediate results obtained within that time limit. It was observed that with an increase in the number of regions

Table 2.3: Results for p93791 (32 cores) for various values of K and P.

K	P	Test time (clock cycles), T					CPU Time	
		ILP*	ILP**	DP	Δ^*	Δ^{**}	ILP*	ILP**
2	48	663388	640033	681228	2.68%	6.4%	1.24 s	0.71 s
3	48	615042	606472	624062	1.49%	2.9%	11.29 s	12 s
	64	473990	474801	486605	2.6%	2.4%	4.6 s	20 s
4	48	605224	604344	610357	0.8%	1.0%	2 m	2 m 3 s
	64	461053	454080	484201	5%	6.6%	12.22 s	20 s
	72	412989	407574	417876	1.1%	2.52%	1m 3 s	1 h 47 m
5	64	458132	457362#	479515	4.67%	4.84%	7 m 27 s	3 h
	72	410475	400795	415483	1.22%	4.84%	2 m 57 s	6 m 40 s
	96	312328	310467#	329886	5.62%	6.25%	1 m 53 s	3 h

ILP*: ILP with random placement of access points.
ILP**: ILP with a placement of access points guided by the proposed technique.
#not an optimal solution—ILP timed-out after 3 hours.

K, ILP took longer time to report the first intermediate solution. We set three hours as the limit because no noticeable improvement was seen in the ILP intermediate solutions after this duration. The CPU time taken by DP is only 4 minutes and 8 seconds, which is negligible in comparison to the cumulative execution time of 1 day and 12 hours taken by ILP for all the 12 cases shown in Table 2.5. Our approach also reported better results for some larger values of K. Note that for instances for which ILP** yields lower test times than DP, a combination of the two methods can be used. An effective partition can be first identified using DP, and then the test-pin assignment problem can be solved using ILP, as in [33]. However, for larger problem instances, ILP is not feasible due to high computation requirements.

We next show results for a 20x20 NOC obtained by replicating cores from the two SOC benchmarks; see Table 2.6. Considering the size of NOC, a CPU time limit of 6 hours was set for ILP for each of the 12 cases shown in Table 2.6. No intermediate solutions were reported for larger values of K. While the cumulative CPU time taken by ILP* for all the cases was found to be 2 days and 7 hours, ILP** took 3 days. The

Table 2.4: Comparison of test times (in cycles) for p93791 obtained from different methods with the tam-independent lower bound reported in [2].

K	P	DP	ILP*/ILP**	Lower bound [2]
4	48	610357	604344	580744
5	64	479515	457362	435561
5	72	415483	400795	387167
5	96	329886	310467	290378

CPU time taken by ILP is clearly impractical, and the ILP approach does not scale with the number of cores and the size of the partition. In contrast, the CPU time for DP is only 25 minutes and 10 seconds.

To further demonstrate the scalability and benefits of the proposed approach, we evaluated the DP method for an SOC of the future that has nearly 1,000 cores (a 32x31 NOC). The DP procedure completes in 4 hours of CPU time. In order to evaluate the quality of the solution (test time obtained), we developed a simple baseline heuristic of generating a partition. Among all intermediate regions available for partitioning, the region having the largest number of network tiles was selected and a separator was placed randomly. Pins were distributed in the ratio of the dimension of the leaf regions. We generated 100 such partitions for each different values of K and a value of $P = 150$, and report the mean, minimum, and maximum test time for each case, as shown in Table 2.7. It can be seen from the table that DP provides consistently superior results—two orders of magnitude reduction in test time compared to the mean test time for the baseline case. Compared to the minimum test time for the baseline case, the test time reduction is in the range of 13% to 48%.

We also considered an SOC with $1,600$ cores (40x40 NOC). The DP solution (for $2 \leq K \leq 5$) was obtained in 3 hours of CPU time for $P = 150$. The test times obtained from DP was consistently lower than that for the randomized baseline method for all values of K. For example, for $K = 5$, we obtained 19.80%, 44%, and

Table 2.5: Test times for 14x14 NOC for various values of K and P.

K	P	Test time (clock cycles), T				
		ILP*	ILP**	DP	Δ^*	Δ^{**}
3	48	3715943	3664878	3764088	1.29%	2.70%
	64	2857587	2806197	2868364	0.37%	2.21%
4	72	2491517	2464016	2486965	−0.18%	0.93%
	96	1917893†	1906318	1989787	3.74%	4.38%
5	108	1662490	1645632	1682522	1.2%	2.24%
	120	1541735	1528065	1543730	0.13%	1.02%
6	108	1766538	1716503	1667207	−5.62%	−2.87%
	128	1679271	1413330	1432921	−14.67%	1.38%
7	128	1672190	1492955	1411292	−15.60%	−5.47%
	140	1371750	1320933	1298180	−5.36%	−1.7%
8	140	1768660	1543097	1288646	−27.14%	−16.48%
	152	1391710	1195750	1197054	−13.98%	0.1%

ILP*: ILP with random placement of access points.
ILP**: ILP with placement of access points guided by the proposed technique.
†Optimal solution obtained from ILP

69.33% reduction in test time compared to the minimum, average and maximum test times, respectively, obtained from the baseline method.

We also examined the scalability of the subset-sum-based method for large SOCs. We ran the procedure `optimize_TAM` on the 32x31 NOC for $K = 4$. Figure 2.19 shows the percentage reduction in test time reported by the procedure over the DP-based method for varying values of Δ and P. The figure also shows the CPU time needed by the procedure for each value of Δ. It can be seen that when Δ is high, the subset-sum-based method is capable of producing better results than the DP, but takes more CPU time. The reduction in test time was found to be as high as 3.3% when Δ was set to 20000. The test time reported by the procedure `optimize_TAM` was 1.4% more than that for DP for $\Delta = 5000$. The CPU time varied from 7.2 hours to 1.5 hours as Δ was swept from high to low values, whereas DP took 4 hours to

Table 2.6: Test times for 20x20 NOC for various values of K and P.

K	P	Test time (clock cycles), T				
		ILP*	ILP**	DP	Δ^*	Δ^{**}
3	48	$39120112^\dagger$	31977790	39710744	1.51%	24.18%
	56	$34327589^\dagger$	29937056	36128201	5.2%	20.68%
	64	$32993009^\dagger$	29252001	34967720	5.98%	19.53%
4	72	$28623706^\dagger$	22095610	27157342	−5.12%	22.9%
	84	$25376412^\dagger$	21334784	24143276	−4.85%	13.1%
	96	$23671502^\dagger$	20848957	23933118	1.1%	14.79%
5	96	22563287	19428477	20885613	−7.4%	7.5%
	108	21947600	17919339	19527804	−18.35%	8.97%
	120	23910854	16851542	18805376	−21.35%	11.59%
6	120	$NR^\ddagger$	19728600	16772337	—	−15%
	128	$NR^\ddagger$	22219900	16231904	—	−26.94%
	140	$NR^\ddagger$	20553216	15957241	—	−22.4%
7	128	$NR^\ddagger$	$NR^\ddagger$	15424776	—	—
	140	$NR^\ddagger$	$NR^\ddagger$	14892919	—	—
	152	$NR^\ddagger$	$NR^\ddagger$	13988145	—	—
8	140	$NR^\ddagger$	$NR^\ddagger$	12988319	—	—
	152	$NR^\ddagger$	$NR^\ddagger$	12768290	—	—
	164	$NR^\ddagger$	$NR^\ddagger$	12204338	—	—

ILP*: ILP with random placement of access points.
ILP**: ILP with placement of access points guided by the proposed technique.
†Optimal solution obtained from ILP
$NR^\ddagger$: ILP did not find any solution within the time limit (6 h).

produce the results. It will be seen later that for the 32x31 NOC, the CPU time for DP can further be lowered to 11 m 50 s using the speed-up technique discussed in Section 2.4.D. As mentioned in Section 2.2.E, the `optimize_TAM` procedure scales down the test times of individual cores for avoiding computational bottlenecks in solving subset-sum problem instances. This approximation step may lead to the elimination of some valid partitions from the solution space. Therefore, optimality is not guaranteed with the subset-sum-based approach.

Table 2.7: Test time of dynamic programming vs. randomized baseline approach for 32x31 NOC, $P = 150$

K	Test time obtained by DP, T_1	Test Time from Baseline Approach			$(\frac{T_2-T_1}{T_2})$ $\times 100\%$
		Mean, T_2	Minimum	Maximum	
4	48696503	95358264	56438372	166259458	95.82%
5	39058309	78704175	53480028	135262224	101.5%
6	32955187	73151835	47784917	117510883	121.97%
7	29912627	78841592	48190525	127542914	162.57%
8	27623723	82552584	52717461	128748359	198.84%

Table 2.8: Table showing diminishing improvement in the reduction in test time as K increases for a 14x14 NOC.

K	$P = 80$			$P = 120$		
	Test cycles ($T[1]$)	Test cycles (T (DP))	TTR†	$T[1]$	T(DP)	TTR
3	2576720	2604656	—	2517557	2570399	—
4	2783984	2273510	12.7%	1896896	1935007	24.7%
5	2746663	2248619	1.0%	2824656	1543730	20.2%
6	2432954	2229751	0.7%	2624430	1503099	2.6%
7	2523712	2222274	0.3%	1828087	1493571	0.6%
8	2265840	2214001	0.3%	1739505	1493571	0%
K^*	2276611, 12			1559961, 19		
Lower bound [2]	2080309			1386877		

†Test-time reduction (TTR): relative reduction in test time obtained by adding an access point.

We next examine our experimental results for further analysis. Table 2.8 reports the test times for a 14x14 SOC (196 cores) for partition sizes K varying from 3 to 8, and for three different values of P. The table also shows the results produced by [1] for all these cases. The TTR (test-time reduction) column shows the relative reduction of test time obtained by adding an additional access point (using the DP approach). It can be seen that the magnitude of reduction in test time gradually

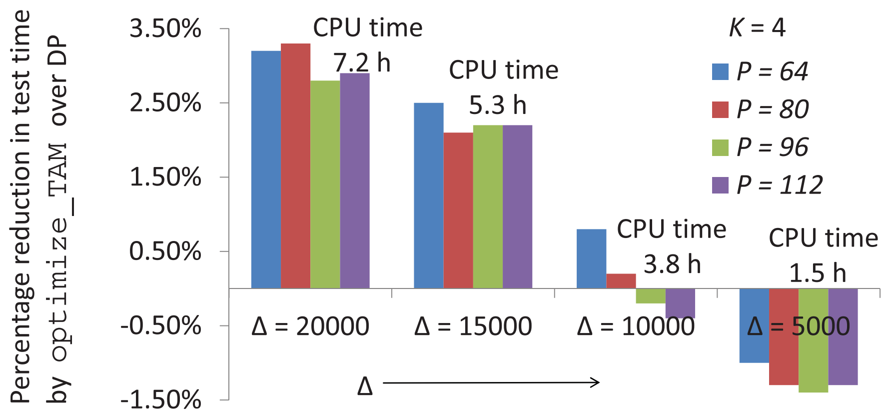


FIGURE 2.19: Results showing percentage reduction in test time by the subset-sum-based method over DP-based method with varying values of Δ for 32x31 NOC, $K = 4$. CPU time for each value of Δ is also shown.

decreases. This is because the test time of a core depends on the number of pins assigned to it and as the partition size increases, the number of pins available per region decreases. By increasing the pin count, we observe that the effect of sudden decrease in TTR can be moderated. For example, for $P = 80$, the TTR rapidly dipped to 0%, but we were able to moderate the sudden decline by allotting 120 pins, and get further benefits by increasing the pin count to 160. However, the number of available pins on the ATE is limited, hence it is natural to ask what is a suitable choice for the partition size and the pin-count that should be used, and how can we calculate these values. These questions will be addressed in future work.

The row K^* in Table 2.8 shows the result produced by [1] when no restriction is placed on the number of access points to be used. It can be seen that [1] reports an extremely large number of access points, which can be harder to implement in practice. Moreover, a large number of access points can lead to the associated problem of power consumption because of test parallelism. We report lower test times than [1] using fewer access points. For $P = 160$ (not shown in the table), the improvement

Table 2.9: Test times (in cycles) of a 10x10 NOC for multiple access points for a range of pin counts.

K	P	DP*	DP†	ILP†		Δ_1	Δ_2
		T_1	T_2	T_3	CT‡		
3	64	154513	40313	27366	43 m	73.91%	82.29%
	80	138740	38752	27318	49 m	72.07%	80.31%
	96	130574	38280	27318	49 m	70.68%	79.08%
4	64	131335	40313	27365	1 h 20 s	69.31%	79.16%
	80	118248	37219	26882	1 h 32 m	68.52%	77.27%
	96	110060	36244	26644	2 h 32 m	67.07%	75.79%
5	64	126393	40313	32822	7 h 46 m	68.11%	74.03%
	80	106274	37219	26377	8 h 13 m	64.98%	75.18%
	96	97390	35851	26164	8 h 25 m	63.19%	73.13%
6	64	120375	40313	32654	12 h 40 m	66.51%	72.87%
	80	100400	37219	26224	12 h 6 m	62.93%	73.88%
	96	87759	35851	25922	14 h 22 m	59.15%	70.46%
7	64	116587	40313	32642	22 h 42 m	65.42%	72.00%
	80	95336	37219	26218	19 h 23 m	60.96%	72.50%
	96	84004	35851	26528	24	57.32%	68.42%
8	64	115152	40313	33562	24 h	64.99%	70.85%
	80	94429	37219	29543	24 h	60.59%	68.71%
	96	81610	35642	28342	24 h	56.33%	65.27%

*unicast (Section 2.3), †multicast, ‡CPU Time; T_1, T_2, T_3 are corresponding test times

$\Delta_1 = \frac{T_1 - T_2}{T_1} \times 100 \quad \Delta_2 = \frac{T_1 - T_3}{T_1} \times 100$

achieved by our method over [1] is as high as 37.6% when K is restricted to 8. When K is not restricted, [1] resulted in a test time (using 23 access points) that is worse than the test time reported by DP with only 7 access points. Since our approach only creates rectangular partitions, a simple post-processing step, such as that implemented in the procedure $ModifiedReshuffle$ of [1], can further reduce test time by moving cores from one region to another. We also report lower bound values for the two values of P in the last row of Table 2.8. The test times that we obtained are only 9.4% (12.5%) larger than the provable lower bounds for $P = 80$ ($P = 120$).

We also present results for multicast routing. In our first experiment, we construct

Table 2.10: Test times (in cycles) of a 35x35 NOC for multiple access points for a range of pin counts.

K	P	DP (multicast)	ILP (multicast)		DP (unicast)
		Test Time	Test Time	CPU Time	Test Time
3	64	238432	332413	24 h	1325574
	80	196245	374421	24 h	1223658
	96	182487	391824	24 h	1194623
4	64	228682	462251	24 h	1216234
	80	190052	301632	24 h	1185769
	96	180558	408543	24 h	1135642
5	64	221058	-	24 h	1172562
	80	188522	-	24 h	1115248
	96	179554	-	24 h	1032542
6	64	214282	-	24 h	1098572
	80	182800	-	24 h	1045532
	96	178982	-	24 h	972354
7	64	214168	-	24 h	1012363
	80	182678	-	24 h	942452
	96	178543	-	24 h	896328
8	64	213272	-	24 h	992456
	80	182624	-	24 h	914423
	96	176442	-	24 h	842391

a 10-by-10 NOC-based SOC out of ten cores from $d695$ by replicating them 10 times and placing them randomly in the 10-by-10 grid. The results are tabulated in Table 2.9. The first and second column of the table show the number of regions K (or the number of access points) used for obtaining the solution, and the total number of pins P available, respectively. The CPU time for DP is only 5 s for multicast and 6 s for unicast. The method based on ILP was run for a maximum of one day in each case. The DP method, based on unicast, described in Section 2.3, is considered as the baseline method. As shown in the table, significant reduction in test time was obtained compared to the baseline method (82.29% to 56.33%), and ILP reported better test times than the DP-based method in all cases.

To examine scalability of the proposed methods and the benefits of multicast for testing manycore SOCs of the future, the experiments are repeated for a 35-by-35 NOC that consists of 1225 cores. The NOC is created using the same method as for the 10-by-10 NOC. Table 2.10 shows that for a larger number of access points, ILP did not produce results even after 24 hours of CPU time. Even for fewer access points, ILP gave inferior results (the best results obtained in 24 h) compared to DP (multicast). Both versions of the DP approach completed in less than 15 minutes. Our results therefore show that while ILP leads to better-quality results for smaller designs, DP is more scalable and therefore the preferred optimization method for larger designs.

2.6.2 Power-constrained test scheduling

To assess the impact of power constraint on test scheduling, we ran our approach on two NOCs: a 6x6 NOC and an NOC with 100 cores (10x10), both constructed out of cores from the benchmark circuit p93971. Due to the lack of information on power consumption of these cores, we assumed that the power consumption in a core is directly proportional to the sum of the number of core's inputs, outputs, bidirectional pins and memory elements — the same approach as adopted in [71]. All values for power consumption used were relative with respect to the total sum of the power consumption of all cores, which is referred to as "system" power consumption in [71]. We therefore refer to power in terms of a normalized value relative to the total system power.

Table 2.11 compares the test length obtained by our approach with that obtained using the ILP model from [71]. All power constraints are defined as a fraction of the system power. Scheduling with the 1.0 power constraint is equivalent to scheduling without power constraints, as no schedule can exceed the total system power. Since our approach approximates the power consumption for a set of cores using manip-

Table 2.11: Test times of p93791 for four access points under power constraints.

Power Constraint[1]	Test Length		Test Length from [71]	
	64 pins	32 pins	64 pins	32 pins
1.00	484201	936737	467441	912781
0.45	485869	936767	470926	916390
0.40	487541	936737	480361	918546
0.35	487541	936767	501982	968905
0.30	568107	936737	543706	981134

[1] Power constraint relative to total power consumption in SOC.

ulation on power profiles to create an approximate profile, the performance of the approach depends on how tightly the approximation scheme bounds the actual power profile from above. The test lengths were found to match closely with the results obtained using [71] for all values of the power limit.

Because our approach is necessitated by the intractability of problem instances involving large NOCs, we present the results for a 10x10 NOC in Table 2.12 for different power constraints and partition sizes. Since, as in this case, each core contributes very little to the system power consumption, the power constraint was set to 25% of the system power consumption at first, and then subsequently the power budget was reduced by 5% at each step. Increasing the power constraint always increases the test length. Under strict power constraints, additional access points will only increase test time compared to relaxed power constraints. The run-time complexity remains the same as before, and no appreciable difference in runtime was found for the reported cases.

2.6.3 Speedup technique

We next show the effect of the speedup technique, discussed in Section 2.4, on the computation time for DP. In Table 2.13, the third column corresponds to the approach taken for reducing the run-time complexity by a factor of P. The speedup is

Table 2.12: Test times of a 10x10 NOC for multiple access points for 120 pins under power constraint.

PC[1]	Number of access points (K)				
	4	5	6	7	8
1.00	993225	772398	750978	748576	746915
0.25	993225	772398	750978	748576	752042
0.20	1005859	869850	799056	795940	848257
0.15	1223566	961602	980472	1036158	1036158

[1] Power constraint relative to total power consumption in SOC.

Table 2.13: CPU times for the speedup technique discussed in Section 2.4.

Dimension	Original DP	Improved DP
10 x 10 ($K = 8$, $P = 152$)	2 min 5 s	9 s
14 x 14 ($K = 8$, $P = 152$)	4 m 8 s	15 s
20 x 20 ($K = 8$, $P = 152$)	25 m 10 s	1 m 16 s
32 x 31 ($K = 8$, $P = 150$)	4 h	11 m 50 s

clearly evident for larger NOCs.

2.7 Conclusion

We have developed a scalable solution to the problem of optimizing test-data delivery in an NOC-based manycore SOC. A formulation based on the subset-sum problem has been proposed for NOCs with dedicated routing and arbitrary topologies. For grid topologies supporting XY routing, test-time minimization has been solved using DP, which computes optimal solutions for rectangular partitions. Results for NOC-based manycore SOCs constructed from ITC 2002 benchmarks have shown that the proposed method yields high-quality results, and scale to large SOCs with many cores. Test scheduling under power constraints and a speedup technique have been incorporated. Since dynamic programming solutions are recursively constructed from solutions to underlying subproblems, the proposed method can inherently facilitate design-space exploration for effective test planning.

3

Test-Cost Modeling and Optimal Test-Flow Selection of 3D-Stacked ICs

Test cost has emerged as a potential showstopper in the adoption of 3D integration. The choice of test flow, i.e., what tests are used and when they are applied during 3D integration ("what to test", "when to test") affects test cost. 3D stacking involves many possible test insertions. Due to multiple yield and test cost parameters corresponding to different dies and tests, such as for pre-bond, post-bond, and partial stack, an exponentially large number of test flows must be evaluated. Therefore, analysis methods and tools are needed for test-cost optimization and automated test-flow selection.

In this chapter, we address test-cost optimization for 3D ICs by developing a cost model that takes into account various test costs at each step of the stacking process. The model is generic and flexible in that it provides placeholders for different test costs that are typically incurred during 3D integration. The proposed model can be adapted for wafer-to-wafer (W2W), die-to-wafer (D2W), and die-to-die (D2D) stacking. We formulate the optimization problem as a search problem and describe a method based on the A* search algorithm [77] to provide an optimal solution.

Although there is no straightforward theoretical bound on the runtime of this search algorithm, A* has been shown in practice to be efficient for various search problems [78–80]. We further propose an approximation step that effectively reduces the state space of the search problem and provides a solution within provable bounds to an optimal solution. Results are presented to highlight the impact of various parameters on test cost and test-flow selection.

The rest of the chapter is organized as follows. Section 3.1 details the motivation for this work and lists the contributions. The notation used in the chapter and the cost model are presented in Section 3.2. Section 3.3 presents two models to further simplify the cost model. Section 3.4 formulates test-cost optimization and test-flow selection as a search problem, and a solution based on A*-search algorithm is presented in Section 3.5. Results are shown in Section 3.6 and conclusions are drawn in Section 3.7.

3.1 Motivation and Chapter Contributions

For today's 2D ICs, tests can be applied at two stages: (i) at the wafer level (wafer sort); (ii) after the chip is packaged. Depending on the yield, wafer sort can be a significant cost saver by alleviating the need for packaging defective dies.

In the manufacturing of a 3D-stacked, tests can be applied at multiple stages—individual wafers or dies can be tested prior to bonding (pre-bond test) and testing can be carried out again after partial stacks are created (mid-bond test). A post-bond stack test can be applied to the complete stack with all the dies. During testing of a partial stack, dies at different layers in the stack can be tested (or re-tested), or their testing can be omitted.

Fig. 3.1 sketches the typical integration of three dies to form a stack. This process involves incremental stacking of one die at a time. During testing of a stack, tests targeting faults in either of the dies present in the stack can be applied. Hence, in addition to pre-bond testing, a die can be tested at multiple stages of stacking as well. Moreover, there can be multiple types of tests, having different fault coverage and test application costs. If a test with lower cost is selected, that can reduce test cost upfront, but the lower fault coverage can result in the stacking of defective dies, thereby lowering the overall stack yield, and consequently increasing the overall cost of stack creation. Therefore, test-flow selection involves selection of the test insertions (also referred to as test moments in the literature [81]), and tests that provide the best trade-off between cost and quality. For this simple example, there exist $2^{3+3+2} = 256$ different combination of test insertions: tests for Die 1 and Die 2 can be applied at all the three stages and there are two stages in which tests for Die 3 can be applied. Moreover, for each selection of insertions, there are different ways in which tests can be selected out of a given test set. It can be easily shown (as discussed in the appendix) that the total number of possible test flows is $\mathrm{O}(N^{l^2})$, where l is the total number of dies in a stack and N is the number of tests available per test insertion.

The optimization problem rapidly becomes intractable with the addition of dies, and because of the added complexity associated with selection of tests. If we consider inter-die interconnect tests, even more test flows are possible. Our goal is to minimize the total cost per "good" package that includes test cost and the part of manufacturing cost that is affected by yield and test escapes. A package is called good if it passes the package test. Since the application of the package test is the last step of the integration flow that we consider, we assume that the package is always tested with highest-quality of tests available to test the package.

Several papers have been published recently on various aspects of test-cost modeling and optimization for 3D ICs [81–87]. These papers include attempts to hand-pick a test flow or select a test flow based on explicit enumeration of a few candidate test flows. A tool was developed to estimate overall cost of a 3D test flow in [84]. A

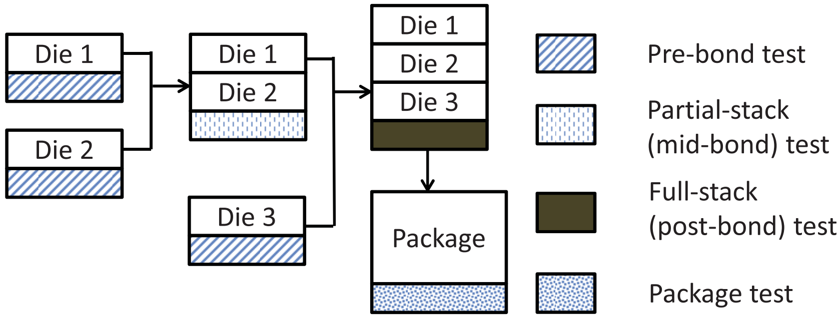


FIGURE 3.1: 3D integration of a stack and the associated test insertions.

limited number of 3D and 2.5D test flows were studied in [81] and [85], respectively. The impact of mid-bond testing (testing at intermediate stages of integration) and logistics on the cost of test flows was reported in [86]. A model to predict the impact of test-flows on product quality in terms of defective parts per million (DPPM) was examined in [87]. However, prior work does not provide any means for systematically exploring (e.g., through implicit enumeration) the solution space of all possible test flows and reporting the test flow that minimizes test cost.

The major contributions of this chapter include:

- A generic cost model to incorporate different kinds of test costs involved in 3D integration.
- A formulation of cost-optimization as a search problem.
- A method based on the A*-search algorithm to find an optimal solution.
- An approximation strategy in addition to the A*-based method to reduce the state space.
- Rational insights into the relationship between yield, test cost, and the selection

of various pre-bond and stack tests on the basis of experimental results.

3.2 Cost Model

In this section, we formally define the cost model that we use for the test-flow selection problem. We highlight the notation for various parameters and decision variables that constitute the optimization problem in Table 3.1 and Table 3.2, respectively. The table entries include key parameters such as the pre-bond and stack tests that are available, test costs, yield, packaging cost, decision variables, etc. The definitions of these parameters are not repeated here for the sake of brevity. The reader is referred to the notation tables for details.

For our discussion, we refer to a "test set", i.e., a set of test patterns, by "test". A collection of tests is referred to as a "set of tests".

Without loss of generality, we use the Williams and Brown model [88] for calculating the number of instances of a die that are detected to be fault free after applying a test of a certain fault coverage. The model postulates the following relationship between defect level DL, fault coverage fc, and yield λ: $DL = 1 - (\lambda)^{1-fc}$. If n is the total number of chips manufactured using a process having production yield λ, the number of non-faulty chips is $n \cdot \lambda$. If a test having fault coverage fc is applied, the number of chips that are detected to be non-faulty is given by $\frac{n \cdot \lambda}{1-(1-\lambda^{1-fc})} = n \cdot \lambda^{fc}$. Note that if a perfect test was applied, the count of non-faulty chips would have been $n\lambda$. This means that $n\lambda^{1-fc}$ instances have escaped the test, and can potentially be detected on application of a better test later during the 3D integration process. Other models can be easily used in place of the Williams and Brown model in the proposed optimization framework—the proposed framework is general and not specific to any particular yield model.

We next quantify the total cost of manufacturing and pre-bond testing of the dies.

Table 3.1: Table describing the given parameters used in the cost model.

Symbol	Meaning
	Parameters related to pre-bond testing
l	Total number of dies in the stack
$\mathcal{D}_i$	i^{th} die in the stack
DC_i	Manufacturing cost of each instance of $\mathcal{D}_i$
n	Number of instances of $\mathcal{D}_1$ manufactured.
λ_i	Yield for die $\mathcal{D}_i$.
B_i	Number of pre-bond tests of $\mathcal{D}_i$, $\forall i$.
PB_{ij}	j^{th} pre-bond test for $\mathcal{D}_i$, where $1 \leq j \leq B_i$.
b_{ij}	Cost associated with pre-bond test PB_{ij}.
$fc(\tau)$	A function that returns fault coverage of a selected test τ. If the cost-optimization tool does not select a test, the function returns zero.
	Parameters related to stack testing
$\mathcal{S}_k$	A stack of dies $\mathcal{D}_1$, $\mathcal{D}_2$, ... $\mathcal{D}_k$, $2 \leq k \leq l$. The stack $\mathcal{S}_k$ is partial for $k < l$.
SC_k	Manufacturing cost of each instance of $\mathcal{S}_k$
ω_{ik}	The component of bond yield of stack $\mathcal{S}_k$ related to defects induced in die $\mathcal{D}_i$ during the stacking of $\mathcal{S}_k$. The overall bond yield for stack $\mathcal{S}_k$ is given by $\prod_{i=1}^{k} \omega_{ik}$.
N_i	Number of different tests that can be applied to $\mathcal{D}_i$ during stack testing.
$\mathcal{T}_{ij}$	j^{th} stack test for $\mathcal{D}_i$, where $1 \leq j \leq N_i$.
t_{ij}	Cost of applying stack test $\mathcal{T}_{ij}$ (this remains the same regardless of the moment at which $\mathcal{T}_{ij}$ is applied).
a_{ijk}	A given binary 3-D matrix that indicates whether $\mathcal{T}_{ij}$ can be applied during stack testing of $\mathcal{S}_k$. Not all tests are applicable at every stage.
PC	Cost of packaging and testing a full stack
	Parameters related to interconnect testing
$\mathcal{I}_i$	Interconnects between dies $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$, $1 \leq i < l$
ic_i	Cost of testing interconnect $\mathcal{I}_i$.
ρ_i	Interconnect yield of $\mathcal{I}_i$.

The cost of manufacturing and testing n instances of $\mathcal{D}_1$ is $n \cdot (DC_1 + \sum_{j=1}^{B_1} x_{1j} \cdot b_{1j})$. The term $\sum_{j=1}^{B_1} x_{1j} \cdot b_{1j}$ refers to the cost of the pre-bond test that is applied. If for die $\mathcal{D}_i$, α_i is the pre-bond test selected from the set of B_i pre-bond tests, the number of instances of of $\mathcal{D}_1$ that are determined to be fault free is $n \cdot \lambda_1^{fc(\alpha_1)}$. Note that $fc(\alpha_i)$ is the fault coverage of the selected pre-bond test for $\mathcal{D}_i$. If none of the pre-bond tests for $\mathcal{D}_1$ is selected, then the function 'f' returns a zero (see Table 3.1) to give $n \cdot \lambda_1^{fc(\alpha_1)} = n$.

Table 3.2: Table describing the unknown variables used in the cost model.

Symbol	Meaning
m_k	Number of instances of $\mathcal{S}_k$ manufactured.
m'_k	Number of instances of $\mathcal{S}_k$ that have been detected to be fault-free after stack testing.
x_{ij}	A binary variable that is 1 if pre-bond test PB_{ij} of $\mathcal{D}_i$ is carried out, otherwise it is zero.
y_{ijk}	A binary variable that is 1 if $\mathcal{D}_i$ is tested by applying $\mathcal{T}_{ij}$ during stack testing of $\mathcal{S}_k$. It can be assigned 1 only if $a_{i,j,k} = 1$.
z_{ik}	A binary variable indicating whether $\mathcal{I}_i$ is tested during testing of $\mathcal{S}_k$. The test time for interconnects is small compared to test time for dies [8], so we merge the testing of $\mathcal{I}_i$ with the stack test that first uses $\mathcal{I}_i$ for transporting test data; hence z_{ik} depends on y_{ijk}.

Note that we need to manufacture that many instances of $\mathcal{D}_2$ as there are instances of $\mathcal{D}_1$ available for stacking. If a pre-bond test is applied on $\mathcal{D}_2$, we must manufacture $\frac{n \cdot \lambda_1^{fc(\alpha_1)}}{\lambda_2^{fc(\alpha_2)}}$ instances of $\mathcal{D}_2$. Therefore, the total cost of manufacturing and testing $\mathcal{D}_2$ is

$$\frac{n \cdot \lambda_1^{fc(\alpha_1)}}{\lambda_2^{fc(\alpha_2)}} \cdot (DC_2 + \sum_{j=1}^{B_2} x_{2j} \cdot b_{2j}).$$

For creating m_k instances of $\mathcal{S}_k$, m'_{k-1} instances of $\mathcal{S}_{k-1}$ are available for stacking with $\mathcal{D}_k$. The total cost of manufacturing $\mathcal{D}_k$ is then given by

$$\frac{m_k}{\lambda_k^{fc(\alpha_k)}} \cdot (DC_k + \sum_{j=1}^{B_k} x_{kj} \cdot b_{kj}).$$

Note that $m_k = m'_{k-1}$. The total cost of manufacturing and running pre-bond tests for all dies (C_1) is then given by the following equation.

$$C_1 = n \cdot (DC_1 + \sum_{j=1}^{B_1} x_{1j} \cdot b_{1j})$$
$$+ \frac{n \cdot \lambda_1^{fc(\alpha_1)}}{\lambda_2^{fc(\alpha_2)}} \cdot (DC_2 + \sum_{j=1}^{B_2} x_{2j} \cdot b_{2j}) \tag{3.1}$$
$$+ \sum_{i=3}^{l} \frac{m_i}{\lambda_i^{fc(\alpha_i)}} \cdot (DC_i + \sum_{j=1}^{B_i} x_{ij} \cdot b_{ij})$$

Clearly, each die needs to be tested by at most one pre-bond test. In other words,

$$\sum_{j=1}^{B_i} x_{ij} \leq 1, \forall i.$$

The total cost of stacking l dies ($C2$) is given by: $C_2 = \sum_{k=2}^{l} m_k \cdot SC_k$, where the number of stacks of k dies (m_k) is determined by the number of dies and partial stacks available after defect screening.

The cost of testing the interconnects between the dies $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ during the testing of S_k is $m_k \cdot z_{ik} \cdot ic_i$. The total cost of testing all the interconnect layers (C_3) is given by:

$$C_3 = \sum_{i=1}^{l-1} ic_i \cdot (\sum_{k=i+1}^{l} m_k \cdot z_{ik}) \tag{3.2}$$

The cost of testing interconnect is much less compared to the cost incurred on testing dies [8]; therefore, to avoid unnecessary complexity in the optimization problem, we restrict an interconnect layer be tested no more than once. Hence, the constraint $\sum_{k=i+1}^{l} z_{ik} = 1$ is imposed on the variables z_{ik} for $i < l$.

The total cost of stack testing (C_4) can now be stated as:

$$C_4 = \sum_{k=2}^{l} m_k (\sum_{i=1}^{k} \sum_{j=1}^{N_i} a_{ijk} y_{ijk} . t_{ij}). \tag{3.3}$$

Since we do not apply more than one stack test for a die at any given test stage, the following constraint on the variables y_{ijk} is applied: $\sum_{j=1}^{N_i} a_{ijk} y_{ijk} \leq 1$.

Finally, we account for the cost associated with packaging and final test, which is given by the equation $C_5 = m_l^{'} \cdot PC$.

The total cost for manufacturing and testing the 3D IC is $CT = C_1 + C_2 + C_3 + C_4 + C_5$. If the total number of packages that are deemed fault free after apply the package test is P, our objective is to minimize $\frac{CT}{P}$ by assigning appropriate values to x_{ij} and y_{ijk}. The variables z_{ik} depend on y_{ijk}, as explained in Table 3.2.

In addition, if we assume that defects are induced only on top two dies during the stacking process, and only these dies are considered for testing at this stage, as in [81], we use constraint $\omega_{ik} = 1$ for $i \leq k - 2$.

A die can be tested multiple times with different tests having different fault coverage. If two tests, say τ_1 and τ_2, are derived using the same fault model, i.e., target the same set of defects, then their effective fault coverage, $EFC(\tau_1, \tau_2)$, can be computed by merging the corresponding list of faults that they detect, and reporting the fault coverage obtained from the new list. Note that while merging the individual fault lists, duplicate entries are removed to avoid counting a detected fault multiple times. The effective fault coverage, or EFC, can be computed for more than two tests in a similar way. This method requires availability of fault lists for every available test. Approximate models that circumvent this requirement are presented in Section 3.3.

A single fault model is often not sufficient to cover all types of defects; therefore, we must account for multiple fault models in the proposed cost model. It is well known that the relationship between different fault models is difficult to quantify. Questions such as, does high coverage for one fault model ensures high coverage for another fault model, have yet to be satisfactorily answered. Recent work [89] argues that a near-100% single-stuck-at (SSA) fault coverage does not ensure high coverage of transition faults by constructing a circuit, which the author refers to as an "extreme" example, and demonstrating that 0% transition fault coverage is achievable with a close-to-100% SSA coverage for that circuit. He further proves, however, that 100% single stuck-at fault coverage leads to 100% transition fault coverage. Delving into these research topics is beyond the scope of this work, and we make simplifying assumptions for our calculation. Assuming that "extreme" circuits are not often found in real-life examples, we may write the aggregate fault coverage of two tests, say τ_1 and τ_2, that are for different fault models, as $AFC(\tau_1, \tau_2) = \max\{fc(\tau_1), fc(\tau_2)\}$. If multiple fault models were not present, we would have just used the fault coverage of the given test for the purpose of calculation; therefore, using a simple expression is justifiable. In Section 3.3, we do not specify fault lists of individual tests, and only numeric value of fault coverage of each test is provided as an input; therefore, a test engineer, instead of using the simplistic equation above, can provide a value that is deemed to be appropriate to the cost optimization tool. This is particularly useful in the following case. If it is required from the tool to select a combination of tests targeting multiple fault models, then the combination has to be provided as one of the inputs to the optimization tool. Note that the aggregated fault coverage, or AFC of two tests for the same fault model is same as their effective fault coverage, or their EFC.

For computing the aggregate fault coverage, or AFC, of more than two tests, the set of given tests is first partitioned on the basis of fault models that they target. Then EFC is computed for every partition, and the maximum EFC among them is reported as the AFC of the given set of tests. It can be seen that EFC or AFC for a given set of tests cannot exceed 100%.

We augment the notation of the aggregate fault coverage as $AFC_{i,j,k}$, where $i \leq j \leq k$, to denote the aggregate fault coverage of all tests that are applied on die $\mathcal{D}_i$ between mid-bond testing of stack $\mathcal{S}_i$ and stack $\mathcal{S}_k$, both inclusive. We also use the symbol $AFC_{i,1,k}$ to denote the aggregate fault coverage of all tests applied to $\mathcal{D}_i$ between test insertions of its pre-bond test and the corresponding mid-bond test of $\mathcal{S}_k$.

We next show how to compute the number of non-faulty stacks after applying mid-bond tests. The William and Brown model is repeatedly applied to estimate the number of instances after multiple tests are applied to a stack. The following assumptions are made for our subsequent discussion.

Assumption 1 The occurrence of defects due to different manufacturing steps are independent to each other.

Assumption 2 We assume that failure of dies in a stack during mid-bond testing and post-bond testing are uncorrelated. Since dies in a heterogeneous stack are likely to be dissimilar, the above independence assumption can be justified in practice.

If tests are only applied to $\mathcal{D}_1$, then the number of stacks remaining after mid-bond test of $\mathcal{S}_k$ is applied is given by

$$m'_k = n \cdot \lambda_1^{AFC_{1,1,k}} \cdot \prod_{j=2}^{k} \omega_{1,j}^{AFC_{1,j,k}}$$

Note that every yield parameter captures a different manufacturing step, the above expression is obtained by the assumption of independence of occurrence of defects to these manufacturing steps.

If tests are applied on all dies, then we can rewrite the above equation as

$$m'_k = n \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k}} \cdot \prod_{j=2}^{k} \omega_{i,j}^{AFC_{i,j,k}})$$

The yield components from all dies are multiplied together due to the assumption of the independence of die failures to tests.

Note that the instances of $\mathcal{D}_i$, for $i \geq 2$, are added to the stack after pre-bond tests are applied to them. If pre-bond tests are applied to dies before stacking, then the defects that have already been screened during pre-bond test will not cause further yield loss. After accounting the elimination of faulty dies during their respective pre-bond tests, the above equation can then be corrected to

$$m'_k = \frac{n \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k}} \cdot \prod_{j=2}^{k} \omega_{i,j}^{AFC_{i,j,k}})}{\prod_{i=2}^{k} \lambda_i^{fc(\alpha_i)}}$$

$$\implies m'_k = n\lambda_1^{fc(\alpha_1)} \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k} - fc(\alpha_i)} \cdot \prod_{j=2}^{k} \omega_{i,j}^{AFC_{i,j,k}}) \tag{3.4}$$

An example on the computation of m'_k is presented for a two-die stack in the appendix.

If we further assume that the fault coverage of each interconnect test is 100%, the expression for m'_k can be extended to

$$m'_k = n\lambda_1^{fc(\alpha_1)} \cdot \prod_{i=1}^{k} (\lambda_i^{AFC_{i,1,k} - fc(\alpha_i)} \cdot \prod_{j=2}^{k} \omega_{i,j}^{AFC_{i,j,k}}) \prod_{j=i+1}^{k} \rho_i^{z_{ij}}$$

The above equation accounts for the interconnect yield ρ_i of the interconnect layer $\mathcal{I}_i$.

As stated in Section I, the cost model described above can be adopted for W2W, D2D, and D2W integration. For W2W integration, if the pre-bond tests are skipped, the x_{ij} variables can be constrained to zero. Some mid-bond tests can be skipped (and the decision variables fixed appropriately) due to constraints on test access. For D2W stacking, the cost model can consider pre-bond tests as decision variables, but certain mid-bond tests may be omitted and corresponding decision variables fixed. Note that D2D stacking offers the most flexibility in terms of test flows, hence it leads to the largest number of decision variables.

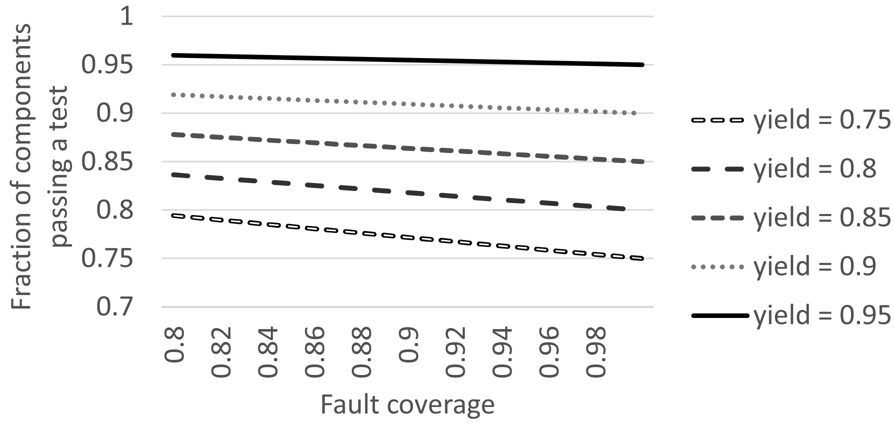


FIGURE 3.2: Variation of fraction of components passing a test on fault coverage according to the William and Brown model.

3.3 Approximation to the Cost Model

Test engineers may not have access to fault lists of tests that are available for testing dies, thus making it impossible to estimate the cost associated with a test flow, or to find an optimal test flow using the cost model presented in Section 3.2. In this section, we present two models that can be used for estimating cost incurred by a test flow without requiring the fault list.

3.3.1 Model I

We simplify the calculation of the effective fault coverage (EFC) of two or more tests that target faults from the same fault model. The EFC of a set of tests cannot be less than the maximum fault coverage of individual tests, and at best, it can be 100%. Therefore, we make a pessimistic assumption that the EFC of a set of tests is equal to the maximum fault coverage of all tests in the test set. Note that the assumption is true only in the case when the test having the maximum fault coverage is the superset of every other element tests.

Figure 3.2 shows the fraction of components passing a test as the fault coverage of the test varies from 80% to 100% (according to the William and Brown model). This is shown for different yields values ranging from 0.75 to 0.9. Note that the fraction does not vary much as the value of fault coverage changes, especially for high yield values. If we report EFC of a set of tests as 80%, and the component being tested has a yield of 90%, we are inaccurate only by a margin of $0.9^{0.8} - 0.9^{1.0} = 1.9\%$ in the worst case, because EFC can be at best 100% when EFC is computed using fault lists. With better yields, the difference further reduces. Moreover, as the manufacturing processes matures, yield values will be on the higher side. Therefore, assumption in this model is practical for the purpose of this calculation.

With this model, $AFC_{i,j,k}$ is reduced to the fault coverage of the test with maximum fault coverage among the tests that are applied on $\mathcal{D}_i$ between the mid-bond test insertions of $\mathcal{S}_j$ and $\mathcal{S}_k$, and $AFC_{i,1,k}$ is reduced to the maximum fault coverage of any test that is applied to $\mathcal{D}_i$ during or before the mid-bond testing of $\mathcal{S}_k$.

3.3.2 Model II

This model is same as the model presented in [6]. The defects introduced by a manufacturing step is "forgotten" once a test is applied to screen those defects. In other words, the test escapes after applying a test are not considered faulty subsequently. In effect, $AFC_{i,j,k}$ is reduced to the fault coverage of the first test that is applied on $\mathcal{D}_i$ between the mid-bond test insertions of $\mathcal{S}_j$ and $\mathcal{S}_k$, both inclusive, and $AFC_{i,1,k}$ is reduced to the fault coverage of the first test that is applied to $\mathcal{D}_i$ during or before the mid-bond testing of $\mathcal{S}_k$.

The proposed approximations to the cost model is illustrated through an example of two-die stack in the appendix.

3.4 Problem Formulation

The optimization problem tackled in this chapter is complex, and it involves many trade-offs. We formulate the problem as a typical search problem that is defined by states, actions, a goal, and a performance measure or a cost function. For finding a solution, a problem solver starts in the initial state, jumps to successive states based on actions taken while in previous states, and terminates when it achieves the goal with the optimum value of the cost function. The state space is the set of all states reachable from the initial state.

Let us define every component of the corresponding search problem using a simple example. In the initial state, no dies have been manufactured, and the actions to be considered in this state are selection of a pre-bond test for $\mathcal{D}_1$ or apply no test at all. Depending on which action, or test, is chosen, the problem solver reaches another state, where n instances of $\mathcal{D}_1$ are manufactured and tested using the pre-bond test selected in the parent state. Every child state of the initial state differs in the pre-bond test that is applied to it. In a newly added state, the problem solver has to chose from the available set of pre-bond tests for $\mathcal{D}_2$, following which new states corresponding to mid-bond test insertions of $\mathcal{S}_2$ are added to the state space explored so far. After test insertions of $\mathcal{S}_2$ are explored, actions for selecting a pre-bond test of $\mathcal{D}_3$ becomes available. New states corresponding to the pre-bond test of $\mathcal{D}_3$ are added, after which actions related to post-bond test insertions of $\mathcal{D}_3$ are considered. This process of choosing actions and addition of new states are carried out until we explore the goal states corresponding to packaging and applying the package test. Every set of new states that we add corresponds to a test insertion. The state space can be represented by a graph in which the nodes are states and the arcs are actions, or choices in our problem.

Figure 3.3 shows the state space for an example of a two-die stack. We have

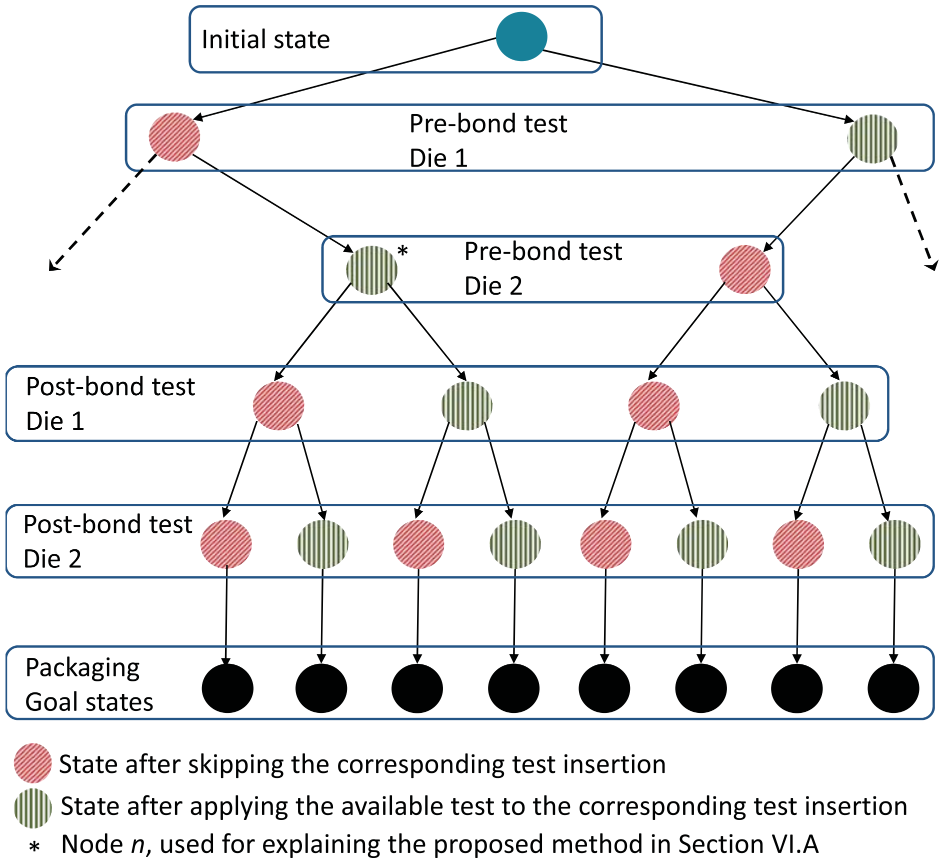


FIGURE 3.3: A partial state space for an example of two-die stack.

four levels of states after the initial state, each corresponding to a test insertion. In this example, we assume that only one test is available for each die at each test insertion; therefore, a branching factor of two is seen at every level of states—one branch for discarding the corresponding test insertion, and the other for selecting the only available test for the test insertion. During a mid (post)-bond test stage after stacking, a decision regarding testing the constituent dies of the partial (full) stack is made sequentially starting from the bottom-most die. For example, in Figure 3.3, choices for *Die* 1 are considered before choices for *Die* 2 in the path to a goal node during the post-bond test stage. This provides a systematic method of exploring the state space for reaching the goal. The only choice possible in the states corresponding to the last test insertion is to package the full stack and apply the package test; therefore, there is no branching at that test insertion to reach the last level of states that are goal states. Due to lack of branching in the penultimate level, we can readily merge the last two levels, but we show them separately here for the purpose of clarity.

It can be seen that every state in the last level is a goal state, or a goal node; therefore, a path from the initial state to any of these states is a valid solution candidate, i.e., a valid test flow. Moreover, there exists a unique path from the initial state to a goal state because the state-space graph is a tree. We require from the problem solver to return a path that has the minimum *path cost*, or the minimum cost per good package. In a traditional search problem, path cost of a path is the sum of the individual cost of each arc constituting the path; however, our cost function also accounts for the total good packages, which may be different for two goal nodes, thereby making costs of internal arcs undefined until the full path is made known. The dependence of arc costs on full paths makes our search problem different, and presumably more difficult.

3.5 Solution Based on A* Search Algorithm

A* search is a popular form of best-first search [90]. It evaluates a node n using a function $f(n)$ that is defined as the sum of $g(n)$, the actual cost of reaching node n from the start node, and a heuristic function $h(n)$, an estimate of the cost to reach a goal node from n. Since $h(n)$ is an estimated cost of the cheapest path from n to the goal, the function $f(n)$ can be interpreted as an estimate of the cheapest solution through n. For a goal node, $f(n)$ is the actual value of the cost function.

The function $h(n)$ is usually computed through a computationally inexpensive heuristic. For a minimization problem, $h(n)$ is an underestimate of the actual cost of reaching the goal using any path starting from n, i.e., $h(n)$ *never overestimates* the cost to reach the goal. If nodes do not repeat during traversal of the search graph, it can be shown that this property of $h(n)$ makes A* optimal.

A basic version of A* is outlined in Figure 3.4. The algorithm maintains a priority queue of active nodes or the states seen so far. The node with the minimum value of $f(n)$ is at the top of the queue. It also maintains a map, which we call *parent*, to reconstruct the solution once a goal node is found. The node that is dequeued from the priority queue is explored and all its neighbors are placed in the priority queue. Since our search graph is a tree, a node once explored (or dequeued) cannot repeat, and hence cannot be queued back to the priority queue. Therefore, $g(n)$ of an explored node is never updated, as is the case with running A* on graphs that are not trees. Note that inside the for loop, a goal node may be enqueued to the priority queue. We may eventually have multiple goal nodes present in the priority queue simultaneously, but the algorithm terminates as soon as the first goal node is dequeued.

It remains to be shown that a suboptimal goal node (say G) is never dequeued. Let the cost of an optimal solution be C^*. Since $h(n)$ of every goal node is equal to zero by definition, $f(G) = g(G) + h(G) = g(G) > C^*$. Suppose n is a node on the path to an optimal goal node and it is present in the priority queue (such a node always exists if a solution exists). Because $h(n)$ never overestimates the true cost a path from n, $f(n) = g(n) + h(n) \leq C^*$. Therefore, $f(n) \leq C^* < f(G)$. It implies that the suboptimal node G cannot be explored, and A* must return an optimal solution.

3.5.1 Computation of $h(n)$

The heuristic function $h(n)$ must never overestimate the cost of any path from n to a goal node; therefore, $h(n)$ in this work is computed by minimizing the cost at each step of the stacking process. We do the following for computing $h(n)$:

- We set the test cost at those test insertions to zero for which no decision has `basic_a_star`(*start_node*)

```
priority_q ← a priority queue with f(n) as key
Enqueue start_node to the priority_queue
parent(start_node) ← null
while priority_q is not empty do
    n ← peek priority_q
    if n is a goal node then
        Report the path from start_node to n using parent
        break
    end if
    for each neighbor n′ of n do
        parent(n′) ← n
        compute f(n′)
        enqueue n′ to priority_q
    end for
end while
```

FIGURE 3.4: A basic version of the A* algorithm.

been made so far for reaching n. For example, for node n in the state space in Figure 3.3, no decision has been taken for post-bond test of *Die* 1 and *Die* 2; therefore, it is assumed that no money is spent on those test insertions.

- Since the total cost incurred on manufacturing die $\mathcal{D}_i$ $(i > 0)$ is dependent on the number of instances of partial stack $\mathcal{S}_{i-1}$ available after testing (m'_{i-1}), m'_{i-1} is underestimated by assuming that highest-quality tests are applied at those test insertions for which no decision has been made so far for reaching n. For example, for node n in Figure 3.3, m'_2 is underestimated by assuming that all defective components are removed by applying appropriate tests for the unexplored states below n.

- The above assumption is also made for computing the amount spent on stack creation.

3.5.2 Adaptation of A to the cost-minimization problem*

If $g(n)$ is the actual cost incurred in reaching node n, and $h(n)$ is computed as outlined in the previous subsection, the function $f(n) = g(n) + h(n)$ can be used for finding a test flow that minimizes the total cost incurred during the stacking process.

We are interested in minimizing the cost per good package; therefore, we redefine $f(n)$ as

$$f(n) = \frac{g(n) + h(n)}{q(n)},$$

where $q(n)$ refers to an estimate of the quantity, or the number of good packages finally created after the package testing is applied. Since $q(n)$ is the denominator, we must overestimate $q(n)$. The basis for the overestimation is given by the following lemma.

Lemma 3.5.1. *The number of instances of good packages obtained after applying the package test is* $n\lambda_1 \prod_{i=2}^{l} \lambda_i^{1-fc(i)} \prod_{j=2}^{l} \prod_{i=1}^{i<=j} \omega_{ij}$*, where* $fc(i)$ *is the fault coverage of the pre-bond test applied on* $\mathcal{D}_i$*.*

Proof. As assumed earlier, highest-quality tests are applied during the package test that eliminates every defective package from the lot of packaged ICs. Therefore, we can set $AFC_{i,j,k} = 1$ and $AFC_{i,1,k} = 1$ in Equation 3.4 to obtain the number of packages as $n\lambda^{fc(\alpha_1)} \cdot \prod_{i=1}^{k}(\lambda^{1-fc(\alpha_i)} \cdot \prod_{j=2}^{k} \omega_{i,j})$. After rearranging factors, we will obtain the expression for the total number of good packages as given in the lemma. $\square$

We overestimate $q(n)$ on the basis of the above lemma. As can be seen from the above expression, the total number of good packages differ for different flows only based on whether pre-bond tests of dies have been applied. Since $\lambda_i^{1-fc(\alpha_i)} < 1$ and $0 \leq fc(\alpha_i) < 1$, $fc(\alpha_i)$ can be set to 1, to overestimate the total number of good packages, or $q(n)$. We just assume that a pre-bond test (with 100% fault coverage) is applied to those dies for which no decision has yet been made for reaching node n in the state space.

```
modified_a_star(start_node, Δ)
priority_q ← a priority queue with f(n) as key
Enqueue start_node to the priority_queue
parent(start_node) ← null
current_solution ← ∞
while priority_q is not empty do
   n ← peek priority_q
   if n is a goal node then
      Report the path from start_node to n using parent
      break
   end if
   for each neighbor n′ of n do
      parent(n′) ← n
      compute f(n′)
      if n′ is goal node then
         if f(n′) < current_solution then
            current_solution ← f(n′)
         end if
      else if f(n′) ≥ (1 − Δ) · current_solution then
         continue
      end if
      enqueue n′ to priority_q
   end for
end while
```

FIGURE 3.5: A $(1-\Delta)^{-1}$-approximation algorithm.

3.5.3 $(1-\Delta)^{-1}$*-approximation step*

On running the basic A* algorithm on several instances of our problem, we found that we were unnecessarily exploring nodes that are not goal nodes, and have values of $f(n)$ that are reasonably close to the optimum cost C^*. To better control exploration of the search space, and thereby reduce the runtime of the algorithm, we add a parameter Δ, where $0 \leq \Delta < 1$, and propose a modification to the basic A* algorithm. Figure 3.5 shows the modified version of the A* algorithm.

The variable *current_solution* is the minimum of all costs of goal nodes seen so far during the search process. It is updated with $f(n')$ on finding a goal node n' if $f(n')$ is lower than the current value of *current_solution*. A node n' is not added to the queue if n' is not a goal node and $f(n') \geq (1-\Delta) \cdot current\_solution$. This prevents exploration of n' in future. The following theorem establishes an approximation bound on the result derived from the approximation step.

Theorem 3.5.1. *The algorithm shown in Figure 3.5 guarantees a solution that is provably* $(1 - \Delta)^{-1}$ *times* C^* *(optimum cost) in the worst case, where* $0 \leq \Delta < 1$.

Proof. If an optimal goal node G^* ($f(G^*) = C^*$) is present in the priority queue, then from the proof of the optimality of A*, it can be readily concluded that G^* is always explored, and an optimal solution is returned. A sub-optimal solution is only returned in the case when a non-goal node n' lies in the path to reach G^* and n' is not added to the priority queue because $f(n') \geq (1-\Delta) \cdot f(G)$ for a sub-optimal goal node G present in the priority queue with the least f-value among all goal nodes present in the priority queue. If the cost returned after termination of the algorithm is C, then the following relationship holds.

$$f(n') \leq C^* < C \leq f(G) \leq \frac{f(n')}{1-\Delta} \leq \frac{C^*}{1-\Delta}$$

$$\implies C^* < C \leq (1-\Delta)^{-1} C^* \qquad \square$$

With Δ set to zero, optimality of the solution is not affected. On setting $\Delta = 0.5$, the approximation factor becomes two, and the solver provides a solution with the value of the cost function no worse than twice its optimum value. As Δ increases, we can also see a significant reduction in the number of states explored by the problem solver, thereby reducing the run time.

3.6 Results

First, for a small example of two dies, we compare the two objective functions—total cost per good package and total cost [6]. Next for a bigger example of four dies, we make important observations about selection of tests and test insertions. We compare the methods based on A* and the approximation algorithm with another exact method based on trivial exhaustive enumeration.

Table 3.3: Input parameters for an example of 2-die stack.

Die	Manufacturing cost	Test cost (100% coverage)	Stacking cost	Packaging cost
$\mathcal{D}_1$	\$1.80	\$0.35	\$0.40	\$3.50
$\mathcal{D}_2$	\$2.20	\$0.20		

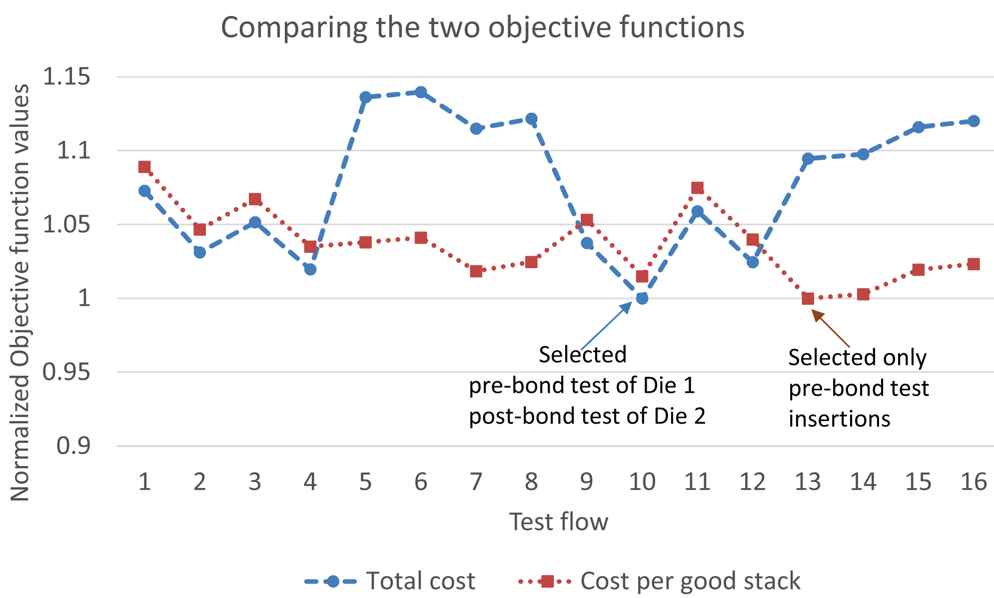


FIGURE 3.6: Comparing our objective function with the objection function in [6] for a two-die stack.

Using the wafer-cost estimator from [91] for a standard wafer with 300mm diameter and an edge clearance of 3mm, and applying the "Gross Die per Wafer formulas" from [92] for such wafers, the cost of manufacturing per die was estimated to be \$1.77 in [87]. On adding the cost of manufacturing TSVs, the cost of manufacturing per die increases to \$1.92 [87]. The cost of manufacturing TSV is 60% of the cost of stacking operation [93], thus making the stacking cost \$0.25 per stack. The cost of pre-bond tests were assumed to be \$0.50 for 99.6% fault coverage in [87]. We have assumed similar values for the various parameters in our test cases.

For the first experiment, a two-die stack is considered. Table 3.3 provides details

Table 3.4: Input parameters for an example of 4-die stack.

Die	Manufacturing cost	Test cost		
		100% coverage	95% coverage	90% coverage
$\mathcal{D}_1$	\$1.80	\$0.35	\$0.18	\$0.09
$\mathcal{D}_2$	\$2.20	\$0.20	\$0.10	\$0.05
$\mathcal{D}_3$	\$2.30	\$0.30	\$0.15	\$0.08
$\mathcal{D}_4$	\$1.90	\$0.25	\$0.13	\$0.07

of the input parameters used in the example. Die yields of 0.9 and stack-yield components for both the dies are assumed to be 0.95. Figure 3.6 compares the objective functions of "cost per good package" with "total cost". There are $2^4 = 16$ test flows possible in this simple example. For each test flow, data points are normalized with respect to the optimum value obtained from the corresponding objective functions. While the former metric selected only the pre-bond test insertions as the optimum flow, the latter metric selected the pre-bond test for $\mathcal{D}_1$ and the post-bond test of $\mathcal{D}_2$. It can also be seen that while total cost declines sharply from eighth test flow to ninth test flow, profit margin is negatively affected per good stack. Using total cost as the objective function offsets the profit margin per good stack by 2% for this example, but for large examples, the profit margin may be significantly reduced with the objective function used in [6].

Next we consider a larger example consisting of four dies. Table 3.4 lists the input parameters used in the example. For each test insertion, we have three tests in this example. The cost and the associated fault coverage are listed in the table. The cost of creating stack is assumed to be 40 cents per stack and the cost of applying package test is set at 3.50 per package. The stack yield component for the top two dies for a stacking operation is assumed to be 0.95, and for the remaining underlying dies, the stack-yield component is set at 0.99.

If only one test with 100% coverage is assumed to be available, the total number of test flows that are possible for this example is 8192. Figure 3.7 shows the normalized

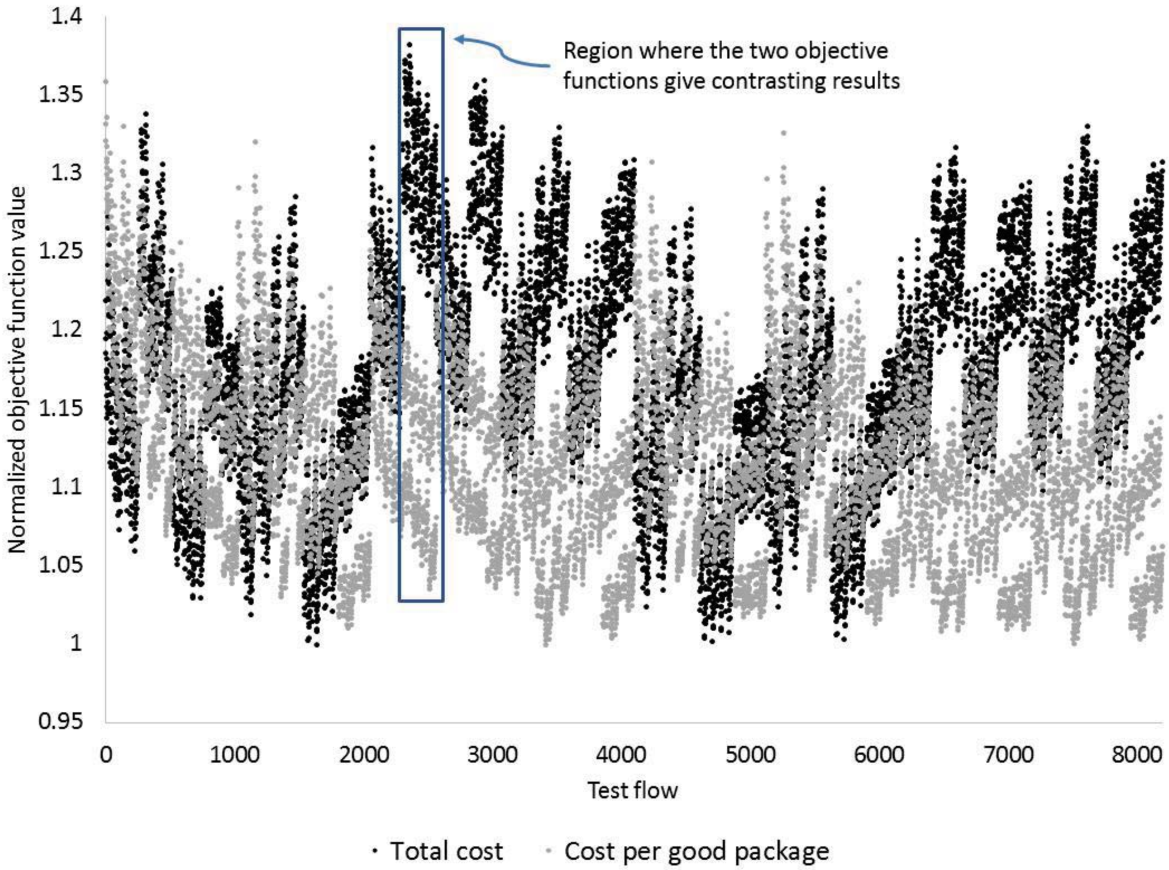


FIGURE 3.7: Comparing our objective function with the objection function in [6] for a four-die stack.

values of the two objective functions for every test flow. Die yield of 0.9 is assumed for this experiment. We see several regions in the figure where test flows lead to contrasting values of the objective functions. Figure 3.8 zooms into one such region. Note that the objective function of "total cost" is not a "bad" choice for several test flows, and even those test flows that are good with respect to the objective function of "cost per good package" require comparatively more money to be spent. When we have a limited supply of resources for manufacturing and testing of 3D-Stacked ICs, our approach can be adapted to report an optimal test flow under an additional constraint on the total amount that can spent. The A*-based method can discard

Table 3.5: Input parameters for an example of four-die stack.

Die	Die-yield threshold to select a pre-bond test		
	1 test	2 tests	3 tests
$\mathcal{D}_1$	0.88	0.93	0.96
$\mathcal{D}_2$	0.94	0.96	0.98
$\mathcal{D}_3$	0.96	0.97	0.98
$\mathcal{D}_4$	0.98	0.98	1.0

nodes with the value of $g(n) + h(n)$ exceeding the available resources.

A defective die, if it remains untested before stacking, can make the entire stack defective. Moreover, if the die yield is low, it becomes increasingly important to screen the die for defects prior to bonding. Therefore, it is only natural to think that if we vary the die yield of a single die from low to high values after fixing every other yield parameters to known values, we will obtain a threshold value of the die yield, below which a pre-bond test is always selected. For the example of four dies, first we assume that only one test of 100% fault coverage is available for each die at the corresponding test insertions. For finding the threshold die-yield value for a die, we sweep the yield for that die from small to large values, and select that value after which we see a change in the pre-bond test selection. In this experiment, we set yield of all other dies at 0.99 (a very high value). Furthermore, we also varied the number of available tests, from one to three, for each die and measured the threshold value of die yield for pre-bond test selection. First we add a test with 95% fault coverage and then the test with 90% fault coverage is added to the pool of available tests.

Table 3.6 shows the threshold values of die-yield for different dies. We can draw the following inferences from this experiment.

- Pre-bond test selection is dependent on the value of die yield. Based on the interplay of given parameter values, each die has a threshold value of die yield below which a pre-bond test is always selected.

- From Table 3.4, it is clear that as more pre-bond tests are available, the benefit

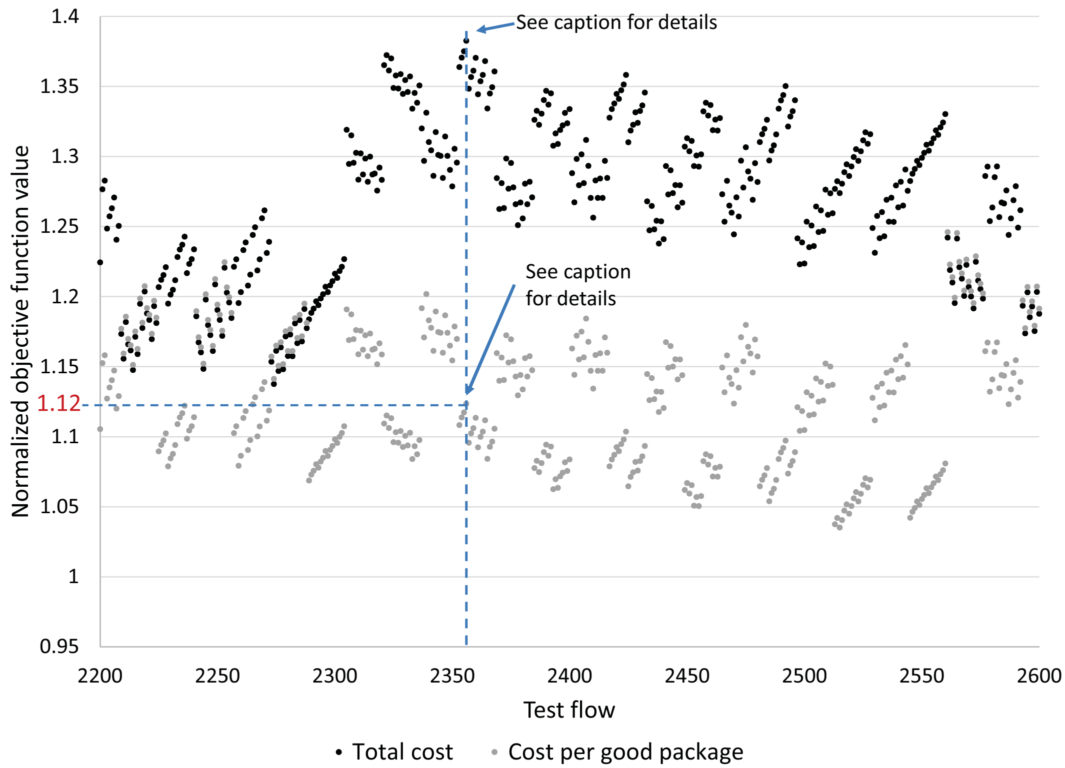


FIGURE 3.8: Zooming into the highlighted region in Figure 3.7. The test flows in this region have contrasting values of the objective functions. The test flow that has a near-optimal value for the objective function (1.12 times the optimum value) in this work has the worst value for the objective function in [6].

of selecting a pre-bond test increases. Cheaper pre-bond tests are now selected even for those values of die yield for which pre-bond tests were skipped earlier.

- The tendency to select pre-bond test increases consistently with the stack size. For example, we see higher threshold die-yield value for dies that are higher up the stack. This can be explained by observing that a defective die, if bonded to a stack that has already been tested to be non-faulty, will make the entire stack faulty; hence it is better to discard the defective die before it enters the stack.

If a cheaper test is available, it is chosen to reduce the test cost (depending on the

Table 3.6: Selection of test insertion and tests for $\mathcal{D}_2$ on varying die yield.

<table>
<tr><th rowspan="2">#Tests</th><th rowspan="2">Test insertion</th><th colspan="3">Range of die-yield values</th></tr>
<tr><th>100% coverage</th><th>95% coverage</th><th>90% coverage</th></tr>
<tr><td rowspan="4">1</td><td>pre-bond</td><td>0.6 – 0.94</td><td rowspan="4">Not available</td><td rowspan="8">Not available</td></tr>
<tr><td>mid-bond of $\mathcal{S}_2$</td><td>0.95 – 0.97</td></tr>
<tr><td>mid-bond of $\mathcal{S}_3$</td><td>0.6 – 0.99</td></tr>
<tr><td>mid-bond of $\mathcal{S}_4$</td><td>—</td></tr>
<tr><td rowspan="4">2</td><td>pre-bond</td><td>—</td><td>0.6 – 0.96</td></tr>
<tr><td>mid-bond of $\mathcal{S}_2$</td><td>0.6 – 0.79</td><td>0.80 – 0.99</td></tr>
<tr><td>mid-bond of $\mathcal{S}_3$</td><td>—</td><td>0.6 – 0.99</td></tr>
<tr><td>mid-bond of $\mathcal{S}_4$</td><td>—</td><td>—</td></tr>
<tr><td rowspan="4">3</td><td>pre-bond</td><td>0.6 – 0.67</td><td>—</td><td>0.68 – 0.97</td></tr>
<tr><td>mid-bond of $\mathcal{S}_2$</td><td>0.68 – 0.80</td><td>0.81 – 0.92</td><td>0.93 – 0.99</td></tr>
<tr><td>mid-bond of $\mathcal{S}_3$</td><td>—</td><td>—</td><td>0.6 – 0.99</td></tr>
<tr><td>mid-bond of $\mathcal{S}_4$</td><td>—</td><td>—</td><td>—</td></tr>
</table>

fault coverage of the test). But the decrease in test quality has to be compensated. For the running example, we observed that if a pre-bond test of lower fault coverage is chosen, a post-bond test of higher fault coverage is almost always added in the selected test flow. For example, when just one test of 100% fault coverage is available to $\mathcal{D}_2$, the pre-bond test is selected until the yield reaches its threshold value of 94% (see Table 3.6). Starting from 95% onwards, we observed that the pre-bond test is skipped and a mid-bond test for $\mathcal{D}_2$ during stack testing of $\mathcal{S}_2$ is applied later, which is also skipped after die yield exceeds 97%. Another mid-bond test for $\mathcal{D}_2$ during stack testing of $\mathcal{S}_3$ is always selected for all values of the die yield.

Next an extra test for $\mathcal{D}_2$ is added. The fault coverage of the added test is 95%, and it is half the cost of the test with 100% fault coverage. We see that the added test is now selected as pre-bond test. On adding a test, the threshold value of die-yield to select a pre-bond test increases to 96%. On top of that, an additional mid-bond test of 100% fault coverage is also selected for $\mathcal{D}_2$ during testing of stack $\mathcal{S}_2$. For yield values greater than 80%, the test selection changes to the test with 95% coverage for this test insertion. The test with lower fault coverage is also chosen for the test

Table 3.7: Selection of test insertion and tests for $\mathcal{D}_1$ on varying test cost.

<table>
<tr><th rowspan="2">#Tests</th><th rowspan="2">Test insertion</th><th colspan="3">Range of test cost</th></tr>
<tr><th>100% coverage</th><th>95% coverage</th><th>90% coverage</th></tr>
<tr><td rowspan="4">1</td><td>pre-bond</td><td>$0.10–$0.34</td><td rowspan="4">Not available</td><td rowspan="8">Not available</td></tr>
<tr><td>mid-bond of $\mathcal{S}_2$</td><td>$0.10–$1.32</td></tr>
<tr><td>mid-bond of $\mathcal{S}_3$</td><td>—</td></tr>
<tr><td>mid-bond of $\mathcal{S}_4$</td><td>—</td></tr>
<tr><td rowspan="4">2</td><td>pre-bond</td><td>—</td><td>$0.05–$0.32</td></tr>
<tr><td>mid-bond of $\mathcal{S}_2$</td><td>$0.10-$0.13</td><td>$0.07-$0.75</td></tr>
<tr><td>mid-bond of $\mathcal{S}_3$</td><td>—</td><td>—</td></tr>
<tr><td>mid-bond of $\mathcal{S}_4$</td><td>$0.10-$0.11</td><td>—</td></tr>
<tr><td rowspan="4">3</td><td>pre-bond</td><td>—</td><td>—</td><td>$0.025–$0.305</td></tr>
<tr><td>mid-bond of $\mathcal{S}_2$</td><td>$0.10–$0.13</td><td>$0.07–$0.135</td><td>$0.07-$0.375</td></tr>
<tr><td>mid-bond of $\mathcal{S}_3$</td><td>—</td><td>—</td><td>$0.025</td></tr>
<tr><td>mid-bond of $\mathcal{S}_4$</td><td>—</td><td>—</td><td>$0.025-$0.055</td></tr>
</table>

insertion of $\mathcal{D}_3$ during mid-bond test of $\mathcal{S}_3$.

We draw two conclusions here.

- As more tests are available, more test insertions are chosen—typically tests with different coverage values are chosen to compensate for the decrease in quality and to minimize cost at the same time.
- As die yield increases, we see benefit in selecting, at earlier test insertions, tests of relatively lower quality.

Next we add one more test of 90% coverage with its test cost being half that of the test with 95% coverage. In addition to Table 3.6, we compare test selection at the two test insertions of $\mathcal{D}_2$, namely, pre-bond test and mid-bond test of $\mathcal{S}_2$ in Figure 3.9. As can be seen from the figure, the tests selected for the two test insertions are always complimentary to each other. If an expensive test is chosen for the pre-bond test insertion, a cheaper test is chosen as a mid-bond test, and vice versa. After 67% of die yield value, the pre-bond test of 90% coverage (cheapest test) is chosen, and at the mid-bond test insertion, as the die yield increases, the quality of the chosen

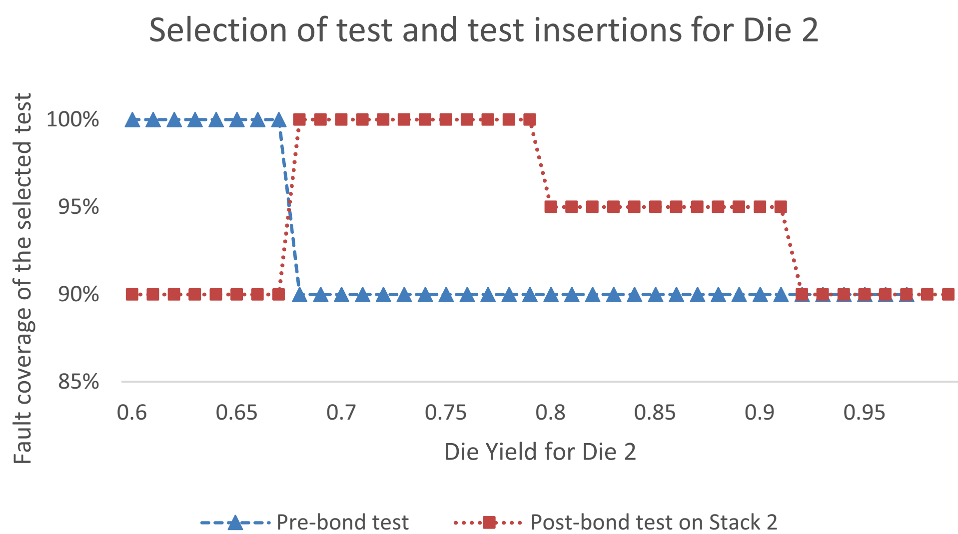


FIGURE 3.9: Comparing test selection at two insertions of $\mathcal{D}_2$.

test decreases. In this example with three tests, $\mathcal{D}_2$ is always tested during mid-bond test of $\mathcal{S}_3$, and the fault coverage of the selected test is 90%.

Next we examine the effect of varying the test cost on selection of tests and test insertions for $\mathcal{D}_1$. We varied the test cost of the die—from a small fraction to a large fraction of its manufacturing cost ($1.80). The test cost of the test with 100% coverage is varied from $0.10 to $1.50 for $\mathcal{D}_1$. The ratio between the test cost of different test is kept the same as before for each case, i.e., the cost of the test with fault coverage of 95% is varied between $0.05 to $0.75 and that of the test with 90% fault coverage is varied between $0.025 to $0.375. The die yield of every die is set to 0.9. We see that the cheapest pre-bond test is always selected except when the test cost becomes expensive and the cost incurred on applying a pre-bond test is no more commensurate with the benefit in applying the test(see Table 3.7). Increasing the number of available test increases the number of selected test insertion, but with an increase in the test cost, even those test insertions are skipped that were selected

Table 3.8: Number of nodes explored using various methods.

N^1	two-die stack			three-die stack			four-die stack		
	EN[2]	A*	$(1-\Delta)^{-1}$	EN[2]	A*	$(1-\Delta)^{-1}$	EN[2]	A*	$(1-\Delta)^{-1}$
1	21	12	6	511	75	51	16383	710	585
2	121	11	9	9841	157	120	2391484	7020	5673
3	341	28	15	87381	915	386	8.95×10^7	73063	50452
4	781	52	17	488281	1510	936	1.53×10^9	268029	191848
5	1556	61	25	2015539	3944	2195	1.57×10^{10}	1358892	941427
6	2801	110	36	6725601	11467	5374	1.13×10^{11}	3543762	2960101

[1] #Tests per test insertion
[2] Exhaustive enumeration

Table 3.9: CPU times for various methods for the example of a four-die stack.

#Tests per test insertion	CPU times		
	Exhaustive enumeration	A*-based method	$(1-\Delta)^{-1}$ approx. strategy
1	< 1 s	< 1 s	< 1 s
2	2 s	< 1 s	< 1 s
3	73 s	< 2	1 s
4	20 m 27 s	6 s	5 s
5	3 h 50 m	41 s	27 s
6	28 h 9 m	7 m 10 s	3 m 2 s

earlier for tests with lower costs.

We next compare the proposed methods with a method that exhaustively enumerates all goal nodes (solution candidates). The test cost for a die varies with the fault coverage of the tests being applied to it. We use the number of test patterns as a surrogate measure of test cost [94]. The fault coverage of the available tests are set between 75% to 100%, and the ratio between test costs of multiple tests is set to match the ratio between the test patterns needed for achieving the fault coverage of the corresponding tests. In addition, we use cost versus fault coverage data provided in [87] as an input parameter for one of the dies in our experiment.

Table 3.8 compares the number of nodes explored by different methods. The total number of nodes in the search tree is given by the expression $\sum_{i=0}^{Z}(N+1)^i$, where N is the number of tests available at each test insertion and Z is the number of total test insertions. It can be easily shown that $Z = \frac{l^2+3\times l-2}{2}$, where l is the number of dies in the 3D stack (refer to the appendix). The table also reports the number of nodes explored by the $(1-\Delta)^{-1}$ approximation strategy with the value of Δ set to 0.05. We see that the proposed method explores significantly fewer nodes than the total nodes in the search tree and the approximation scheme further reduces this number. While the result obtained by A*-based method is provably optimal by construction, we always obtained either optimal or near-optimal solutions using the approximation strategy. Even when Δ is 0.05, which can report a solution with a value of the objective function that is $\frac{1}{1-0.05} \approx 1.053$ times its optimum value in the worst case, we found the approximation ratio to be less than 1.001 in every reported case. Note that the approximation strategy starts discarding a "seen" node from subsequent exploration only after a goal node is found. Until then, the A* method is executed. Since the A* method is already guiding the search towards a "good" solution, the first solution that appears on the frontier of "seen" nodes is close to optimal, and a near-optimal solution is subsequently reported.

All results were obtained on a 32-core Intel(R) Xeon(R) machine with a processor speed of 2.60 GHz, 64 GB memory and a cache size of 20480 KB. Implementation was done using the Java programming language and 8 GB of memory was allocated to the Java virtual machine for all runs. CPU times for the example of the four-die stack is shown in Table 3.9. The run time increases exponentially with increase in the number of tests per test insertion. While the baseline method requires negligible space to the number of test insertions, the memory requirement of the proposed methods increases linearly with the number of states explored, which is true for every A*-based approach. We also ran an experiment on five dies with three tests per test insertion. While the method based on exhaustive enumeration took 72 hours (3 days), the A* based method and the proposed approximation method took 40 minutes and 28 minutes, respectively. For a realistic number of dies per stack and a realistic number of tests per test insertion, we significantly outperform the baseline method with respect to the CPU time.

Thus we note that exhaustive enumeration is impractical for realistic scenarios. Note also that different manufacturing flows lead to different manufacturing cost and die yield, which affect the optimal test flow. In order to evaluate manufacturing flows along with the associated test flows, the test-flow selection tool must be to be invoked repeatedly. For example, depending on the type of vias used, e.g., via-first or via-last, the silicon footprint of a die changes [95], thereby resulting in different manufacturing cost and die yield values. As a result, multiple invocation of the test-flow selection tool may be required to assess the economic viability of these manufacturing flows. The enormous CPU time taken by trivial exhaustive enumeration-based search makes its usage impractical for multiple runs. Since exhaustive enumeration is computationally so expensive, it is likely to require repeated invocation of the test-flow selection even for a fixed manufacturing flow. This can happen because not all options can be considered in a single run of test-flow selection due to the exponential rate at which complexity grows with the number of options. With the proposed intelligent search technique, run time complexity is limited to less than an hour and all options can be considered in one run.

3.7 Conclusion

We have studied the test-flow selection problem for achieving cost minimization, and proposed a generic and flexible cost model to account for various test costs incurred during 3D integration. We have formulated the optimization problem as a search problem and proposed a method based on the A*-search algorithm to find an optimal solution. A $(1 - \Delta)^{-1}$ approximation step has also been presented that further reduces the run time of the algorithm. Solutions to the test-flow selection problem depends on the problem instance. Because of the interplay of given parameter values, optimal choices of tests and test insertions are dependent on problem instances;

therefore, a generic set of rules cannot be specified for minimizing cost. Nevertheless, experimental results have helped us to rationally explain the choices made by the problem solver for obtaining an optimal solution. The cost-optimization method has provided us interesting insights into relationships between yield, test cost, and the selection of various pre-bond and stack tests.

3.8 Appendix

3.8.1 Proof for the complexity of the selection of test flow

Given a stack of l dies, we show that the total number of ways in which test insertions can be selected is $\mathrm{O}(2^{l^2})$. Corresponding to each pre-bond stage, there is one test insertion each. Since k^{th} die can potentially be tested at each subsequent stacking stages, this accounts for an additional $l-k+1$ test insertions for the k^{th} die ($l-1$ insertions for $k=1$). Therefore, the total number of possible test insertions is given by the expression:

$$l + \sum_{k=2}^{l} (l-k+1) + l - 1$$

$$= 2l - 1 + \frac{l(l-1)}{2} = \frac{l^2 + 3l - 2}{2}.$$

A test insertion can be selected or discarded independently of the selection (or omission) of other insertions, hence the number of ways in which test insertions can be selected is $2^{\frac{l^2+3l-2}{2}} = \mathrm{O}(2^{l^2})$.

If the number of available tests per test insertion is N, then there are $N+1$ choices at each test insertion. Therefore, the number of possible test flows is $\mathrm{O}((N+1)^{l^2}) = \mathrm{O}(N^{l^2})$. In general, the number of possible test flows is the product of the available number of choices at each test insertion.

3.8.2 An Example for a two-die stack

The cost model is illustrated through an example of a two-die stack in Table 3.10. Three tests with fault coverage of 100%, 95% and 90%, respectively, are assumed to be available for testing die $\mathcal{D}_1$ at both of its test insertions. The aggregate fault coverage (AFC) of tests with 90% and 95% of fault coverage is assumed to be 97%. For testing $\mathcal{D}_2$, only one test with fault coverage of 100% is available. Every possible test flow is enumerated, and an expression for the number of stacks available after applying post-bond test of stack $\mathcal{S}_2$ (m'_2) for each test flow is shown in Table 3.10. This is the number of stacks that is packaged and further tested using the package test. The number of packages available after applying the package test is also provided against each test flow.

If $\mathcal{D}_1$ is tested using tests with fault coverage of 90% and 95% at different test insertions (see test flows with indices 39, 40, 47, 48, 53, 54, 61 and 62 in Table 3.10), in the column corresponding to the cost model presented in Section 3.2, we see that that λ_1 is raised to the power of the aggregate fault coverage of the two tests because the defect introduced during the manufacturing of die is tested twice. The power of $\omega_{1,2}$, on the other hand, is always the fault coverage of the test applied during the post-bond test because the defects introduced during stacking is tested only once.

For Model I discussed in Section 3.3.A, the aggregate fault coverage is rounded to the maximum fault coverage of the applied tests, and for Model II, test escapes from earlier test insertions were ignored in calculation of m'_2. The difference between these models can be clearly seen for the test flows with indices from 51 through 64.

If $\mathcal{D}_2$ is tested using its only available pre-bond test, then λ_2 does not appear in the expression for m'_2 because manufacturing defects for the die are already screened out by applying the pre-bond test. If another pre-bond test with fault coverage fc was applied to $\mathcal{D}_2$, we would have seen a factor of λ_2^{1-fc} in the expression.

Table 3.10: Illustration of the cost model through an example of a two-die stack.

Test-flow index	Fault coverage of the selected test (%)				Expression for m'_2			Total package count
	Pre-bond		Post-bond		Cost model from Section 3.2	Models from Section 3.3		
	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_1$	$\mathcal{D}_2$		Model I	Model II	
1	—	—	—	—	n	n	n	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
2	—	—	—	100	$n\lambda_2\omega_{2,2}$	$n\lambda_2\omega_{2,2}$	$n\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
3	—	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
4	—	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
5	—	—	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
6	—	—	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
7	—	—	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
8	—	—	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
9	—	100	—	—	n	n	n	$n\lambda_1\omega_{1,2}\omega_{2,2}$
10	—	100	—	100	$n\omega_{2,2}$	$n\omega_{2,2}$	$n\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
11	—	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
12	—	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
13	—	100	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
14	—	100	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
15	—	100	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
16	—	100	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
17	100	—	—	—	$n\lambda_1$	$n\lambda_1$	$n\lambda_1$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
18	100	—	—	100	$n\lambda_1\lambda_2\omega_{2,2}$	$n\lambda_1\lambda_2\omega_{2,2}$	$n\lambda_1\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
19	100	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
20	100	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
21	100	—	95	—	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
22	100	—	95	100	$n\lambda_1\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
23	100	—	90	—	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
24	100	—	90	100	$n\lambda_1\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
25	100	100	—	—	$n\lambda_1$	$n\lambda_1$	$n\lambda_1$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
26	100	100	—	100	$n\lambda_1\omega_{2,2}$	$n\lambda_1\omega_{2,2}$	$n\lambda_1\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
27	100	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
28	100	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
29	100	100	95	—	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
30	100	100	95	100	$n\lambda_1\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
31	100	100	90	—	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
32	100	100	90	100	$n\lambda_1\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
33	95	—	—	—	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
34	95	—	—	100	$n\lambda_1^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
35	95	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.95}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
36	95	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
37	95	—	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
38	95	—	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
39	95	—	90	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$

Continued on next page

The number of packages remaining after applying package test is $n\lambda_1\omega_{1,2}\omega_{2,2}$ for the flows when a pre-bond test is applied to $\mathcal{D}_2$, otherwise the package count reduces to $n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$. If a pre-bond test with fault coverage fc was applied to $\mathcal{D}_2$, then the number of packages available would have been $n\lambda_1\omega_{1,2}\lambda_2^{1-fc}\omega_{2,2}$.

Table 3.10: (continued)

Test-flow index	Fault coverage of the selected test (%)				Expression for m_2'			Total package count
	Pre-bond		Post-bond		Cost model from Section 3.2	Models from Section 3.3		
	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_1$	$\mathcal{D}_2$		Model I	Model II	
40	95	—	90	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
41	95	100	—	—	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
42	95	100	—	100	$n\lambda_1^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
43	95	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.95}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
44	95	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
45	95	100	95	—	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
46	95	100	95	100	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
47	95	100	90	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
48	95	100	90	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
49	90	—	—	—	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
50	90	—	—	100	$n\lambda_1^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
51	90	—	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.9}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
52	90	—	100	100	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
53	90	—	95	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
54	90	—	95	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
55	90	—	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
56	90	—	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\lambda_2\omega_{2,2}$	$n\lambda_1\omega_{1,2}\lambda_2\omega_{2,2}$
57	90	100	—	—	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
58	90	100	—	100	$n\lambda_1^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
59	90	100	100	—	$n\lambda_1\omega_{1,2}$	$n\lambda_1\omega_{1,2}$	$n\lambda_1^{0.9}\omega_{1,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
60	90	100	100	100	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
61	90	100	95	—	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
62	90	100	95	100	$n\lambda_1^{0.97}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.95}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.95}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
63	90	100	90	—	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$
64	90	100	90	100	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1^{0.9}\omega_{1,2}^{0.9}\omega_{2,2}$	$n\lambda_1\omega_{1,2}\omega_{2,2}$

3.8.3 User interface for the test-flow-selection tool

We have developed a graphical user interface (UI), written in the Java programming language, for the cost modeling and test-flow selection tool. The UI allows specifying values to the input parameters of the cost model, and launch the optimization method to produce an optimal test flow. The procedure to launch the tool and method to interact with different controls of the first version of the tool are documented below.

Installing Java runtime environment: The tool is built using the JDK 8; therefore, client systems must have JRE 8 or above to run this tool. Follow the link `http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.`

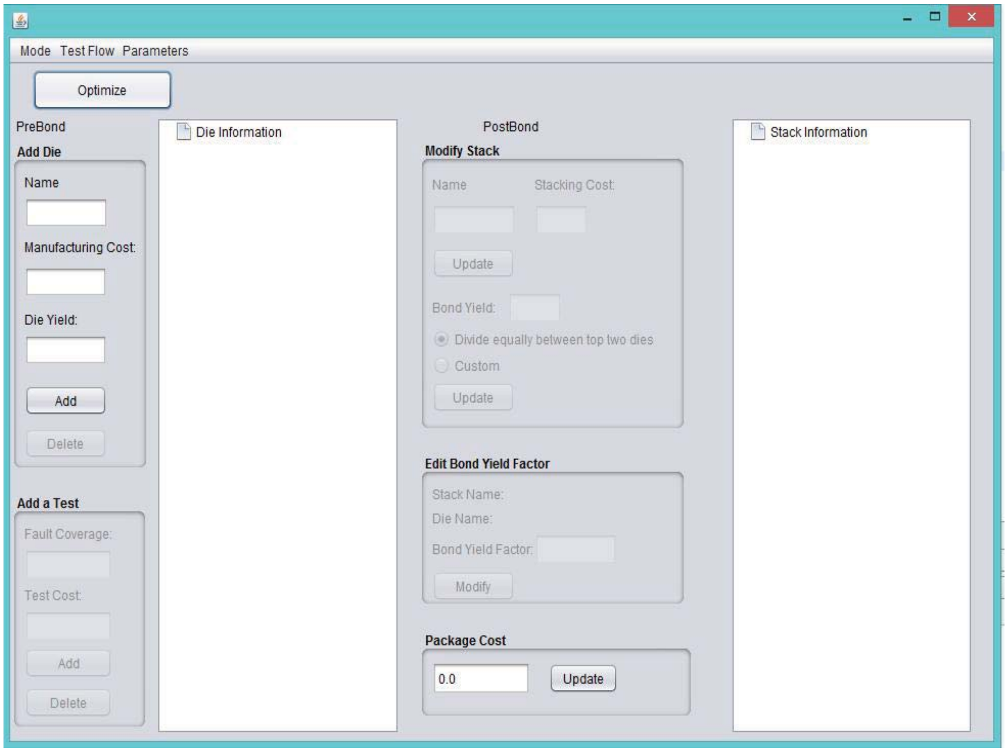


FIGURE 3.10: Main window of the UI.

`html`, accept the license agreement, and download and install the desired installation package.

Launching the tool: The application can be launched using two methods:

- By double clicking the jar file.
- From the command line, run "java -jar CostModelingUI.jar". This is the preferred way because errors or exceptions, if any, are dumped on the command line interface, which is helpful in debugging issues with the tool.

Main window: Figure 3.10 shows the main window of the application. It can be seen that there are multiple panels on the window. Each panel is activated or deactivated based on the context of the state of the tool. The state of the tool is

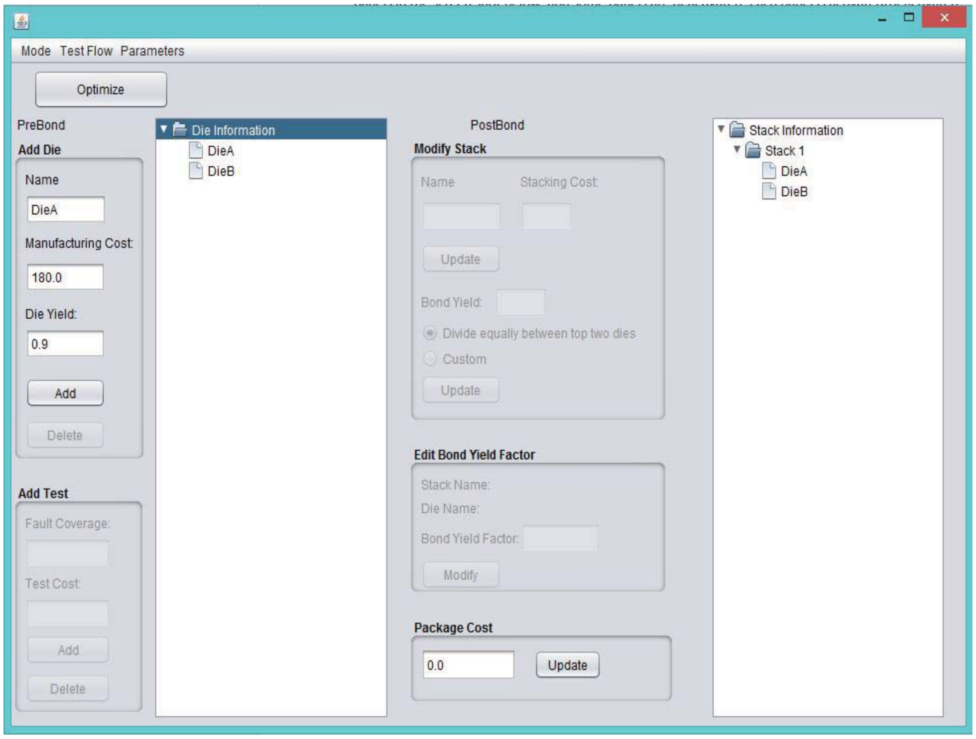


FIGURE 3.11: State of the UI after two dies have been added.

decided by the user activity. In Figure 3.10, the UI is ready to add a die using the "Add Die" panel.

In Figure 3.11, two dies have already been added. The tree control on the left, with "Die information" as the root of the tree, shows information related to dies. The tree control on the right, rooted at "Stack Information" node, displays information related to partial and full stacks. A partial stack can be created using two dies; therefore, we see a node corresponding to "Stack 1" on the right tree control, which further shows the dies that it consists of.

Note that in order to add more dies, the "Die Information" node on the tree panel on the left has to be selected. Clicking on a die in the left tree activates the "Add Test" panel and "Modify Die" panel (see Figure 3.12). The "Modify Die" panel replaces the "Add Die" panel. A test for a die can be added using the "Add

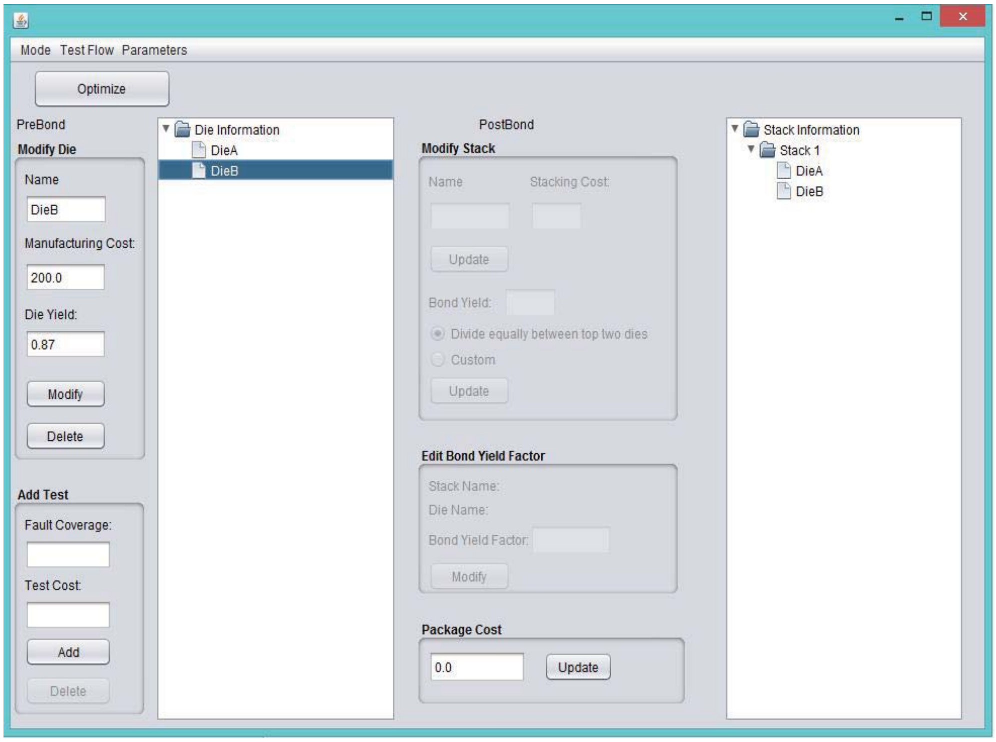


FIGURE 3.12: State of the UI after selecting a die on the left tree control.

Test" panel, and information, such as name of the die, manufacturing cost of the die, and die yield can be updated using the "Modify Die" panel.

Clicking a node on any tree activates relevant panels to take user input for that node, and fields in the activated panels are automatically updated with the values using which that node was created. For example, when Die B was clicked, the "Name", "Manufacturing Cost", and "Die Yield" field on the "Modify Die" panel is updated with the values with which DieB was created. Existing values for parameters for a die can be updated or the die can be completely removed from the model using the 'Modify Die" panel. In Figure 3.12, the manufacturing cost is shown in cents. If a user wants to work in a different unit, he should make sure that he provides all other cost values in the same unit. The UI tool does not help in unit conversion.

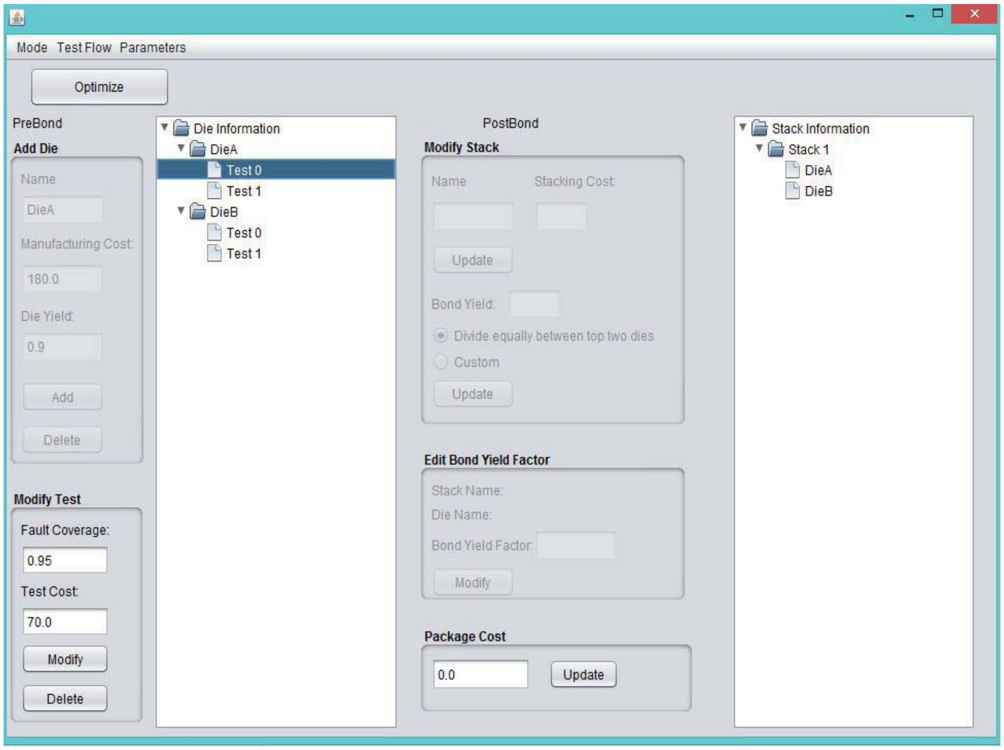


FIGURE 3.13: State of the UI after selecting a test.

Using the "Add test" panel, we add two tests for each of the dies. Figure 3.13 shows the state of the UI after two tests for each die have been added. Since "Test 0" is selected, the "Add Test" panel is replaced by "Modify Test" panel, and "Modify Die" panel is deactivated, which can be reactivated by clicking on a die again.

Next we show how parameters are specified using the tree control on the right. On clicking a stack node, for example, "Stack 1" in Figure 3.14, the "Modify Stack" panel gets activated. There is no "Add Stack" panel because partial stacks are created automatically on addition of dies using the "Add Die" panel.

It can be seen that the user has to specify the parameter values, such as the stacking cost, the bond yield, etc. The bond yield of a partial stack is the product of bond yield factors of the underlying dies. The bond yield factors capture the defects introduced in the constituent dies during creation of the partial stack. For

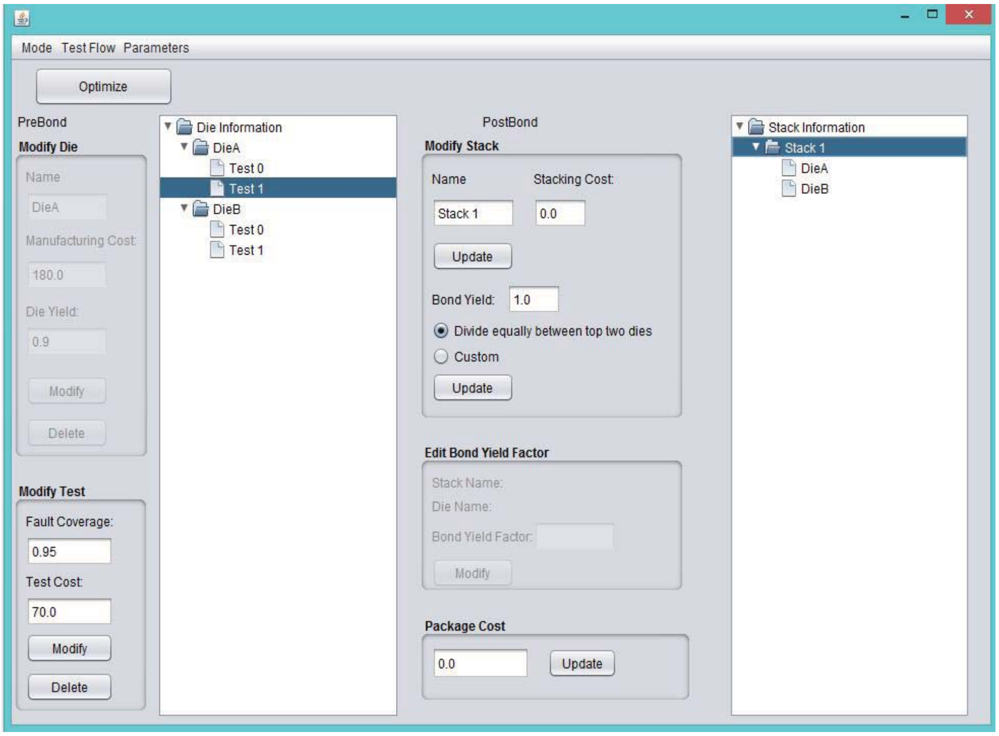


FIGURE 3.14: State of the UI after selecting a node corresponding to a stack.

this purpose, a group of radio buttons can be seen just below the "Bond Yield" field that helps in specifying the bond-yield factors for individual dies. Some researchers assume that only top two dies expect to see additional failures during this stack creation process. Therefore, we have an option of dividing the bond yield equally among the top two dies. For example, on selecting this option, a bond yield of 0.95 will be divided equally as square root of 0.95 among the top two dies. The other option is to provide custom values to the bond yield factors of the underlying dies. This can be achieved using the following two methods:

- Select the "Custom" radio button, and click on update. A new window pops up through which bond yield factors for every constituent die can be specified. See Figure 3.15 for more details. More dies have been added for a better understanding of the UI. The "Stack 3" consists of four dies; therefore, the

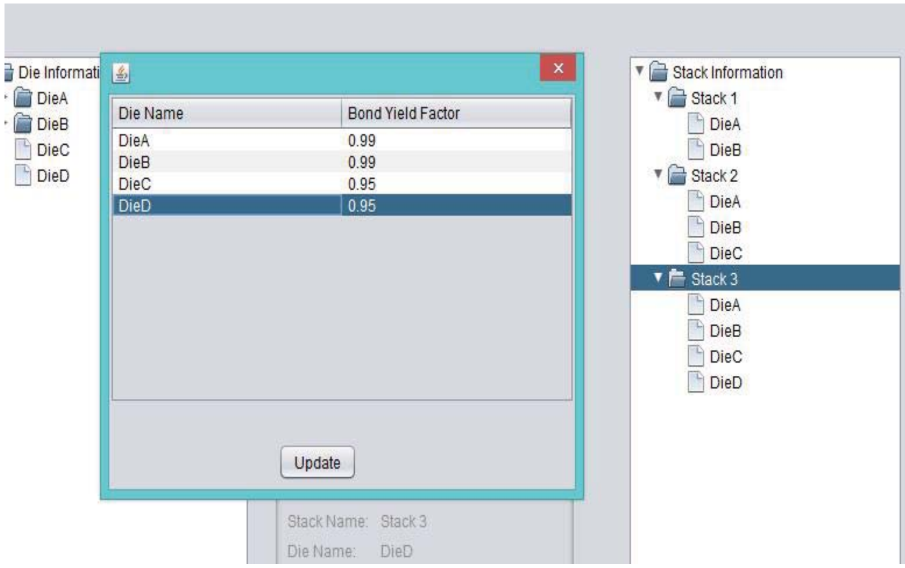


FIGURE 3.15: Using the custom method to set bond-yield factors.

pop-up window allows setting the bond-yield factors of these four dies. In this example, we set these factors as 0.99, 0.99, 0.95, 0.95 for Die A, Die B, Die C, and Die D, respectively. The resulting bond-yield will be the product of these values.

- The second method is to set these factors for the underlying dies individually using the "Edit Bond Yield Factor" for each die individually, one at a time. This panel gets activated by clicking on the desired die node under the desired stack node. For example, the bond-yield factor for "Die D" of "Stack 3" can be changed as shown in Figure 3.16.

Package cost, including the cost of creating the package and thoroughly testing the entire package, can be updated using the "Package Cost" panel. This remains active all the time.

Saving and loading parameters: Through the UI, it is possible to save and load

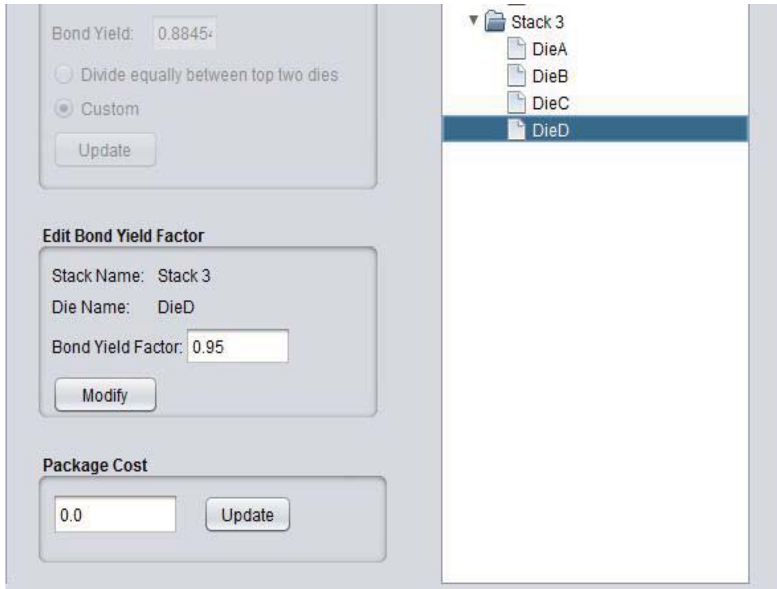


FIGURE 3.16: Editing bond-yield factor of dies individually.

parameters using the "Parameters" menu on top left of the UI. After the parameters have been saved in a file, it is advisable to not tamper with the file manually because the tool may fail to load the file later. The "New" menu item deletes all parameters and reverts the UI back to its initial state.

Solution window The solution is presented in a pop-up window as shown in Figure 3.17. It tells what tests should be applied to which die, and at what time.

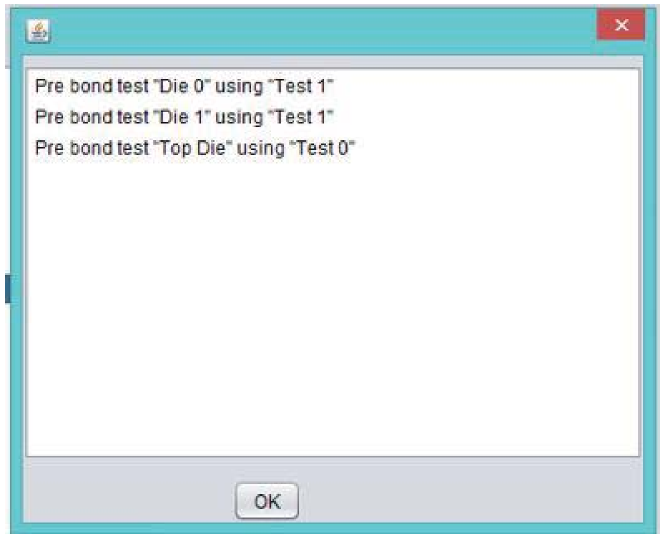


FIGURE 3.17: Solution window of the UI.

4

Reuse-Based Optimization for Pre-Bond and Post-Bond Testing of 3D-stacked ICs

Prior to bonding a die in a 3D stack, TSVs are not fully accessible because one of their ends is not connected to logic on other dies. Such one-ended TSVs lead to pre-bond testability problems of reduced controllability and observability. The combinational part of die logic between the last level of scan cells and outbound TSVs cannot be observed, and that between inbound TSVs and first level of scan cells cannot be controlled. While this problem is more serious for logic-on-logic stacks, it is also a concern for the logic die in memory-on-logic designs. Pre-bond testing is however necessary for achieving high stack yield; it is desirable to have known-good-dies (KGDs) before stacking. Without design-for-test (DfT) innovations, the dies cannot be adequately tested before bonding.

To overcome testability bottleneck, wrapper cells are needed at the two ends of a TSV [8]. Such wrapper cells also facilitate post-bond testing of dies and the interconnects between dies. Since the number of TSVs on a die can be of the order of tens of thousands, the use of wrapper cells for each TSV can lead to significant area overhead. Moreover, wrapper cells on functional paths can lead to higher latency and performance degradation.

To reduce the overhead of wrapper cells, the reuse of existing primary inputs, primary outputs and scan cells for increasing the controllability and observability was proposed in [96, 97]. Wrapper cells are required only for TSVs that cannot be controlled or observed using existing scan cells in the design. Heuristic methods were developed to reduce the number of wrapper cells without affecting fault coverage. However, prior work provides no insights on the minimum number of wrapper cells that are required for full testability, i.e., comparable to the case when the die is a stand-alone chip with I/O pads. These heuristics are therefore ad hoc and the effectiveness of the solutions provided by them needs to be carefully evaluated.

In this chapter, we show that the general problem of minimizing the wrapper-cell count is NP-hard. The proof of complexity is based on polynomial-time reduction from the well-known NP-complete problem of finding a minimal clique partition in a graph [98]. We adopt efficient heuristic algorithms to solve this problem, and compare our results with the heuristic method proposed in [96]. We also compare our results with exact results obtained using integer linear programming (ILP) on smaller benchmark circuits. Unlike prior work, the proposed methods also incorporate a timing-guided and layout-aware approach to address practical timing consideration arising due to scan cells reuse.

The remainder of this chapter is organized as follows. Section 4.1 provides details about the motivation for this work and discusses related prior work. The problem statement is formulated in Section 4.2, and we show that the optimization problem is NP-hard. Section 4.3 extends reuse-based optimization approach to post-bond testing. In Section 4.4, we adopt heuristic graph-theoretic algorithms and propose a generic approach that can be used to incorporate timing and layout information. Experimental results are discussed in Section 4.5. Finally, Section 4.6 concludes the chapter.

4.1 Motivation and Prior Work

The TSVs in a 3D-stacked IC are used to connect metal layers using landing pads across multiple dies. In accordance with terminology used in literature, we refer to TSVs that drive gates in a die as *inbound* TSVs for that die, and *outbound* TSVs for a die are TSVs driven by that die. A TSV that is both inbound and outbound TSV for a die is called a *bidirectional* TSV. We only consider inbound and outbound TSVs in this work, but the solutions for them can be extended for bidirectional TSVs.

Various options for partitioning a design into multiple active layers are explored in [48]. Trade-offs between design effort and latency-power-area benefit are analyzed for different granularity of partitioning, i.e., partitioning at the core level, functional-block level, gate level, or transistor level. At the granularity of cores, a multicore design can be built by stacking different cores on top of each other, or stacking a layer of in-package L2/L3 cache over a layer of cores. At the next level of granularity, stages of processor pipeline can be split across multiple dies, as also proposed in [50]. A functional block can be split across multiple dies at the next level of granularity, e.g., splitting of cache lines, or bit-slicing of functional units. At the finest level is the transistor level, e.g, all NMOS transistors can be placed on one die and all PMOS devices on another die, but the state-of-the-art via technology is not ready to support this scheme of partitioning in the near future [48]. Moreover, it requires a tremendous amount of redesign effort. Applying pre-bond tests will also require fault modeling at the transistor level. For all other granularity of partitioning, redesign effort is less and simpler gate-level fault models can be used for pre-bond testing. Therefore, we do not consider transistor-level granularity of partitioning in this work.

For the first three levels, partitioning can be done in three ways [7]: (i) combinational to sequential, where sequential elements drive one end of TSVs, and the

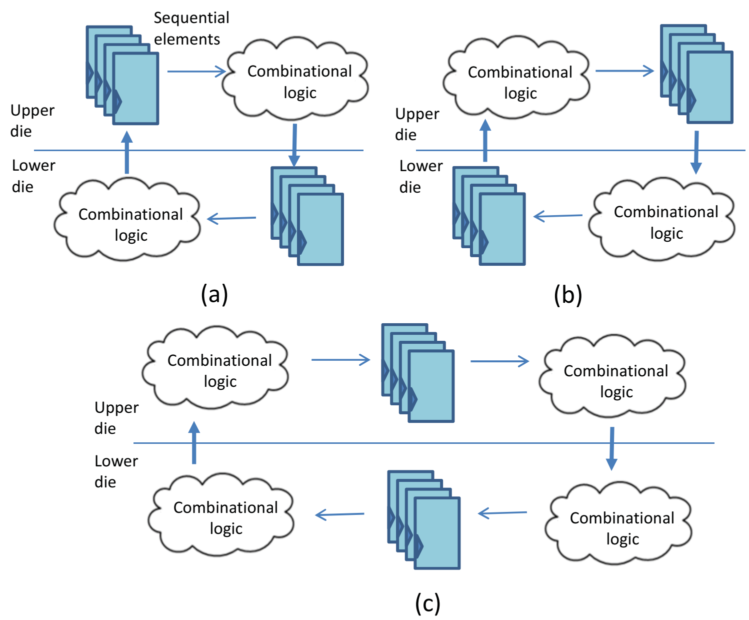


FIGURE 4.1: Different approaches of partitioning [7].

TSVs drive combinational logic on the other side (see Figure 4.1(a)); (ii) sequential to combinational, where combinational logic drive the outbound TSVs on one die and the corresponding inbound TSVs on another die drive sequential elements (see Figure 4.1(b)); and (iii) combinational to combinational, where combinational logic is present on both ends of TSVs (refer to Figure 4.1(c)). It is argued in [7] that a fourth approach of partitioning, namely, "sequential to sequential" is unlikely for most designs, as buffers and delay elements are always present in such paths to avoid potential hold violations due to clock skew. Our experimental circuits are a hybrid of the above partition approaches. Note that in the first approach, it is not possible to observe the faults activated in the combinational blocks during pre-bond testing, whereas controllability is affected during pre-bond testing of the combinational blocks in the second approach. In the third approach, testability is compromised during pre-bond as well as post-bond testing phases.

Appropriate DfT solutions are required to provide controllability and observabil-

Table 4.1: Percentage of combinational logic present before the first level and after the last level of scan cells for the OpenCore [3] benchmark circuits.

Design	Die 0		Die 1	
	First level	Last level	First level	Last level
des_perf	1.29%	4.83%	2.87%	0.82
cf_rca_16	28.7%	1.08%	24.64%	0.93%
cf_fft_256_8	1.6%	12.44%	5.24%	25.47%

Table 4.2: Stuck-at and transition fault coverage for designs from OpenCore [3] with and without wrapper cells for TSVs.

Design	Fault model	Die 0		Die 1	
		With wrapper	Without wrapper	With wrapper	Without wrapper
des_perf	Stuck-at faults	100%	86.74%	100%	80.21%
	Transition faults	99.99%	80.69 %	99.67 %	61.08%
cf_rca_16	Stuck-at faults	99.94%	26.87%	99.9%	75.18%
	Transition faults	94.88%	11.73%	97.72%	65.94%
cf_fft_256_8	Stuck-at faults	99.96%	53.25%	99.89%	51.14%
	Transition faults	94.42%	34.37 %	95.89 %	61.08%

ity to TSVs and the logic connected to them. In this section, we first assess the impact of open-ended TSVs on stuck-at and transition fault coverage during pre-bond testing. We show that the fault coverage is considerably reduced if wrapper cells are not used. Next, we overview prior work [96, 97] that addresses this problem by reusing existing scan cells. Testability issues related to post-bond testing are addressed in Section 4.3.

4.1.1 Fault coverage without wrapper cells

Table 4.1 shows the number of combinational logic gates before the first level and after the last level of scan cells (as a percentage of total gate count) in the netlist for two-die stack implementation of OpenCore benchmarks [3], namely *cf_fft_256_8*, *cf_rca_16*, and *des_perf*. Note that we are reporting the count of those gates that only have inbound TSVs in their fan-in cone for the first-level of combinational logic, and those gates that only have outbound TSVs in their fan-out cone for the last level of

Table 4.3: Stuck-at and transition fault coverage for a 4-die implementation of *cf_rca_16* from OpenCore with and without wrapper cells for TSVs.

Die Number	Stuck-at faults		Transition faults	
	With wrapper	Without wrapper	With wrapper	Without wrapper
Die 0	99.93%	21.32%	97.69%	9.69%
Die 1	99.92%	20.08%	97.30%	8.42%
Die 2	99.93%	20.14%	97.78%	8.45%
Die 3	99.92%	75.22%	94.42%	65.82%

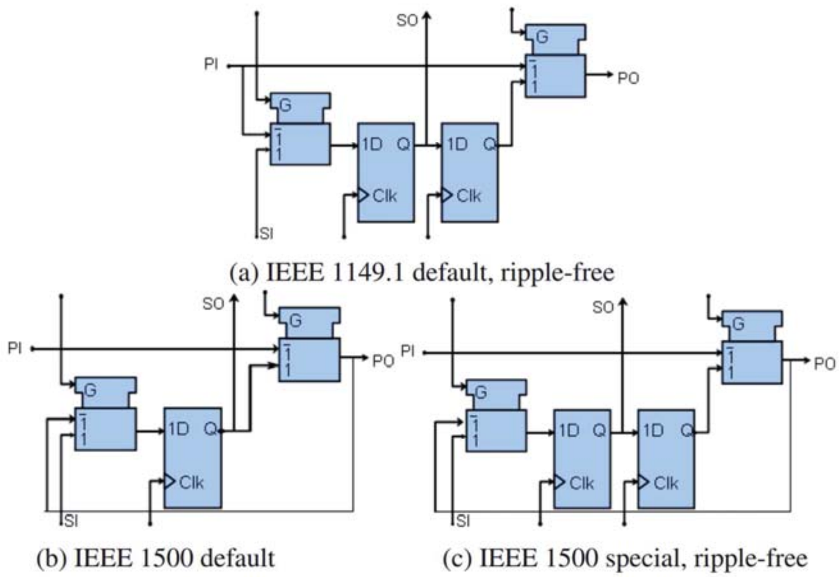


Figure 4.2: IEEE Standard wrapper cells [8].

combinational logic. In addition to these gates, the faults on gates that have both inbound (outbound) TSVs and flops, or PI (PO) in their fan-in (fan-out) cone may not be activated (observed); therefore, the numbers in Table 4.1 are conservative with respect to the decrease in fault-coverage that will be observed in the absence of wrapper cells.

Table 4.2 shows how the stuck-at and transition fault coverage is affected when no wrapper cells are added to make the TSVs controllable and observable at the pre-bond stage. The circuits from the OpenCore benchmarks are each partitioned into two dies for performance optimization, and the fault coverage for the two dies in each benchmark are listed for two cases: (i) wrapper cells are added for increasing testability; (ii) wrapper cells are not inserted. For the same two cases, Table 4.3 shows stuck-at and transition fault coverage for a 4-die implementation of *cf_rca_16*. It further highlights the importance of equipping die logic with sufficient testability for pre-bond testing. A commercial ATPG tool was used to calculate the fault coverage in each case. Based on these results, we conclude that it is necessary to incorporate a DfT solution for pre-bond testing.

However, the addition of wrapper cells leads to die-area overhead. As shown in Figure 4.2, the wrapper cells are typically based on either IEEE Std. 1149.1 or IEEE Std. 1500 for supporting various test modes. Using these cells require an additional area of one or two flip-flops per end for each TSV. As discussed in [8], when dies are designed by different teams or companies, it is recommended that ripple-protected wrapper cells be used; therefore, even higher overhead (four flip-flops) is required at each TSV to achieve testability. Therefore, the insertion of wrapper cells at every TSV imposes significant overhead.

4.1.2 Related Prior Work

It is shown in [96, 97] how existing scan cells can be reused to provide testability to dies prior to bonding. For an inbound TSV, a scan cell is selected such that it does not have an overlapping fan-out cone with the TSV, as shown in Figure 4.3(a). The output of the selected scan cell is multiplexed with the TSV; see Figure 4.3(b). Figure 4.4 illustrates how fault coverage is affected if the fan-out cones of a TSV and the selected scan cell overlap. The error due to the s-a-1 fault at line A cannot be propagated to the output. Although the fault is activated, it is not possible to create

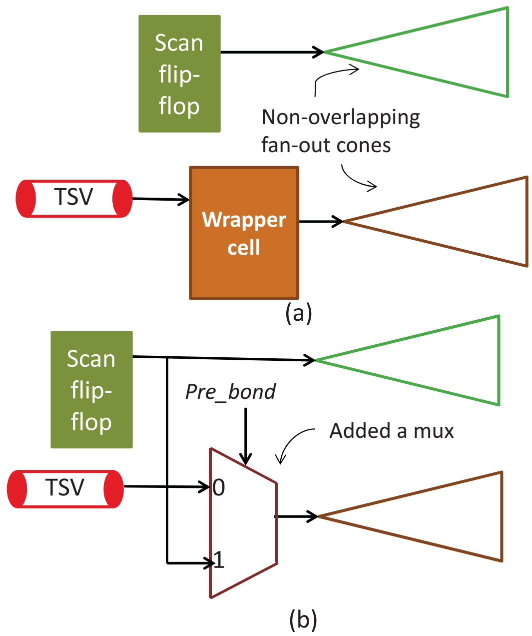


FIGURE 4.3: Illustration of how an existing flop can be reused for an inbound TSV: (a) addition of wrapper cell; (b) flop reuse.

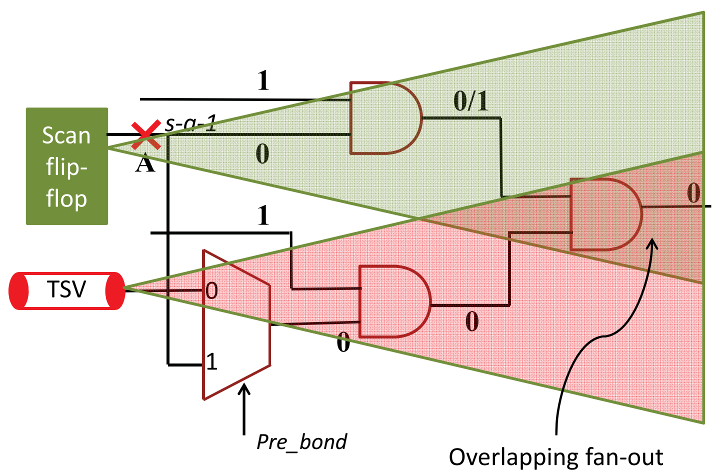


FIGURE 4.4: Example showing an untestable s-a-1 fault at line A due to overlapping fan-out.

a sensitized path to an observable output. If the fan-out cones are non-overlapping, then there is no loss in testability if the corresponding scan cell is selected. Similarly, for outbound TSVs, the criterion for choosing a scan cell is that the fan-in cones of an outbound TSV and the selected scan cell do not overlap. In order to use a scan flop to detect faults in the fan-in cone of an outbound TSV, a multiplexer and a XOR gate are required, as shown in Figure 4.5. A heuristic was proposed in [96] for selecting a scan cell based on the above requirements, and the number of reused scan cells was maximized. It was shown how the reuse of scan cells results in a significant reduction in the number of wrapper cells required. This DfT solution leads to an increase in the pattern count, but it was reported in [96] that the test-data volume reduced due to a reduction in the number of bits per pattern. A drawback of this solution is that it is targeted only at pre-bond testing; post-bond interconnect testing becomes more complex and it has to rely on ATPG for the In-Test mode.

A limitation of [96] is that after reusing a scan flop for a TSV, the scan flop is not considered for further reuse by other TSVs. The heuristic proposed in [96] does not update the fan-out (fan-in) of scan cells that are reused. Moreover, the ad hoc approach in [96] does not provide any insights on the minimum number of wrapper cells required for full testability. Another limitation of [96] is that it does not consider timing constraints that impose restrictions on the use of scan flops that are at the ends of critical paths.

In [97], a hypergraph-based partitioning algorithm was presented to partition a netlist into two dies such that the number of scan flops connected to TSV ends is maximized. However, such a partitioning criterion is impractical, since partitioning a circuit needs to exploit the benefits offered by TSVs, such as higher performance and reduced power consumption. The authors of [97] proposed a heuristic method for assigning flops to TSVs using concepts from graph theory. However, their approach searches for a solution locally; only scan flops that are connected to TSV ends are reused in a greedy manner. Our solution satisfies the conditions required for testability, but instead of being an ad hoc approach as in [96] [97], it is based on a formal method that provides globally optimal results. We first show that the general problem of reducing the number of additional wrapper cells is NP-hard. We construct

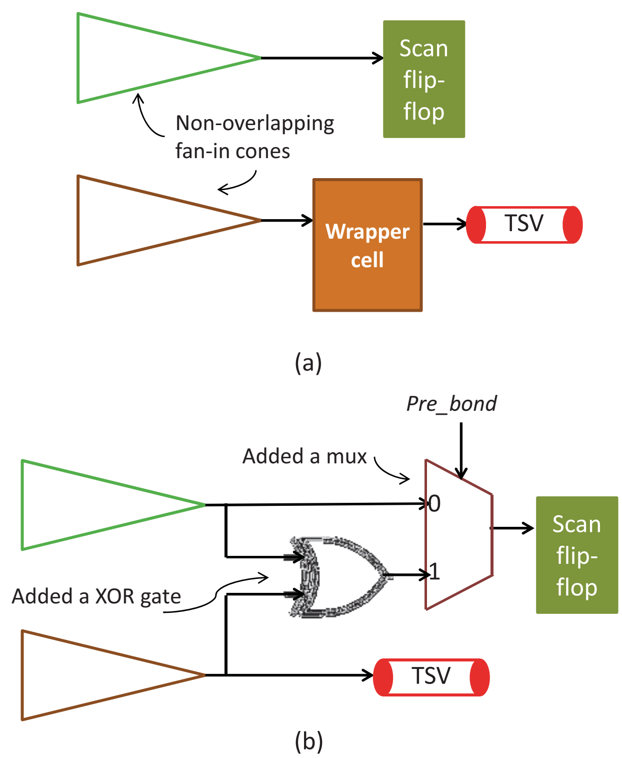


FIGURE 4.5: Using a wrapper cell in (a) makes the logic connected to outbound TSV observable. Using a 2-input XOR gate in (b) makes all single faults on its input observable by an existing scan flop.

a polynomial-time reduction from the minimal clique-partitioning problem, which is known to be NP-complete [98], and subsequently adopt heuristic algorithms to solve this problem. We also implement an exact approach for solving this problem using ILP, and compare between solutions obtained using various algorithms.

4.2 Problem Description

In this section, we present more details about the optimization problem. We assume a full-scan design, and refer to a scan flip-flop simply as a flop in the subsequent discussion.

4.2.1 Selection criteria for flops

Definition 4.2.1. *The fan-out cone of a gate is obtained by tracing downstream from every output of that gate, and the tracing is bounded by flops and primary outputs (POs). Similarly, the fan-in cone of a gate is obtained by tracing upstream from every input signal of that gate, and the tracing is bounded by flops and primary inputs (PIs). The fan-out (fan-in) cone of an inbound (outbound) TSV is defined in a similar manner.*

We first consider the following problem. Given a circuit consisting of TSVs and flops, assign a flop to each TSV, such that:

- the fan-out cones of an inbound TSV and the assigned flop do not overlap;
- the fan-in cones of an outbound TSV and the assigned flop do not overlap.

If no assignment is possible for a TSV, a wrapper cell is added. Figure 4.6 shows how an added wrapper cell can be reused for multiple TSVs. A wrapper cell can be assigned to multiple TSVs; but note that once an assignment is made, the fan-out/fan-in cone of the assigned flop (or wrapper cell) is changed. If a flop is assigned to an inbound TSV, then the new fan-out cone of the flop will also include the fan-out cone of the inbound TSV. Similarly, the fan-in cone of a flop includes the fan-in cones of the outbound TSVs assigned to it. Based on this information, the following lemma can be stated.

Lemma 4.2.1. *Any two inbound TSVs that are assigned to a common flop do not have overlapping fan-out cones. Similarly, any two outbound TSVs that are assigned to a common flop do not have overlapping fan-in cones.*

Proof. The lemma can be proved using contradiction. Suppose that two inbound TSVs have overlapping fan-out cones. Then any flop that is assigned to one TSV

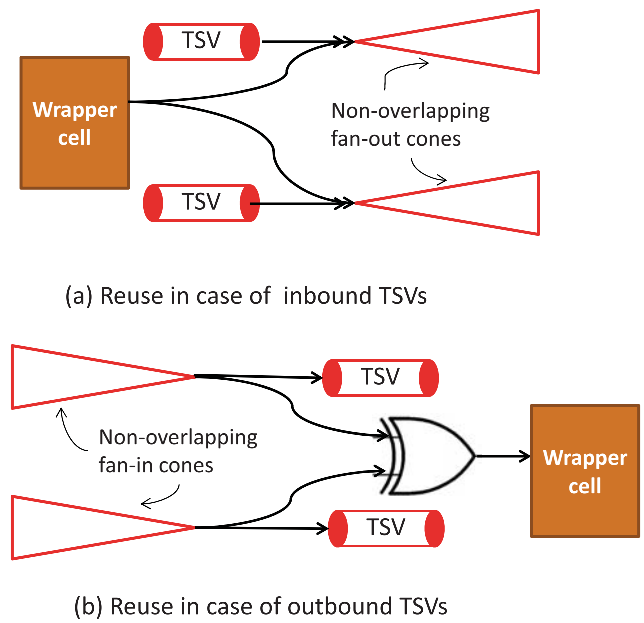


FIGURE 4.6: Illustration for Lemma 4.2.1: reusing an additional wrapper cell for multiple TSVs.

cannot be assigned to the other; otherwise the selection criteria is violated. This is because the fan-out cone of the assigned flop is now a superset of the fan-out cone of the first TSV after the assignment. However, we assumed that the two TSVs were assigned the same flop, which is a contradiction. The proof for the case of outbound TSVs is similar. □

Next, we discuss our optimization problem and the clique-partitioning problem from graph theory, and establish the relationship between these two problems.

4.2.2 Wrapper-cell-count minimization (WCM) problem

Given a design with flops and TSVs, minimize the number of additional wrapper cells that are required to provide complete testability to all TSVs (and connected logic), such that the flop selection criteria are not violated. We refer to this problem as wrapper-cell-count minimization (WCM). Note that if no additional wrapper cells are required, then all the TSVs are assigned the existing flops in the design.

4.2.3 Clique-partitioning problem

Definition 4.2.2. *A clique is a complete graph, i.e., a graph in which each node is connected to every other node. It can be easily seen that the number of edges in a clique with n nodes is $\binom{n}{2}$, i.e., $\frac{n(n-1)}{2}$.*

Given a graph, the problem of partitioning the vertex set of the graph into disjoint sets, such that each set forms a clique and the number of sets is minimized is known as minimal clique-partitioning problem. The above problem of finding is known to be NP-hard [98]. The decision version of this problem, whether a given graph has k cliques, is known to be NP-complete. We use the clique-partitioning problem, or CPP, to prove that WCM problem is NP-hard.

Note that solving CPP is equivalent to solving a graph coloring problem [98], and any method for solving one problem can be used to solve the other problem. We use this relationship to utilize known integer linear programming (ILP) models for the graph coloring problems to obtain an exact solution for WCM, which can be applied to solve small problem instances.

4.2.4 Polynomial-time reduction from CPP

Given a general instance of CPP, we describe a construction that can be used to prove that WCM is NP-hard. Given a graph $\mathbf{G}$, and an integer k, we construct an instance of WCM, and show that $\mathbf{G}$ can be partitioned into k cliques if and only if the instance of WCM can be solved using k cells (includes both existing flops and additional wrapper cells). We use the following definition in our construction.

Definition 4.2.3. *A set of nodes in a graph is called an* independent set *if no two nodes of the set are connected.*

label($\mathbf{G}$) //Vertex set, $V = \{v_1, v_2, ..., v_{|V|}\}$.

$M \leftarrow \{v_1\}$
for $v \leftarrow v_2$ **to** $v_{|V|}$ **do**
  if v is not connected to any vertex in M **then**
    $M \leftarrow M \cup \{v\}$
  end if
end for
Label all nodes in M as "flop"
Label all nodes not in M as "TSV"
return independent set M

FIGURE 4.7: Pseudocode for a deterministic labeling procedure.

Without loss of generality, let us assume that the vertex set V of graph $\mathbf{G}$ is ordered in any arbitrary way, i.e., $V = \{v_1, v_2, ..., v_{|V|}\}$. Then using vertex v_1, a polynomial-time and deterministic labeling procedure, such as in Figure 4.7, can be developed that returns a maximal independent set containing vertex v_1. Let the size of the returned set be m. Then $\mathbf{G}$ consists of $m + n$ nodes, where $n = |V| - m$. The labeling procedure also labels these m nodes as flops, hereafter referred to as 'flop nodes', and labels the remaining n nodes as TSV ('TSV nodes'). The labeling corresponds to an instance of WCM; the m nodes correspond to m existing flops in the design that can be assigned to TSVs, and n TSV nodes correspond to n TSVs. In this graph model, an inbound TSV is not differentiated from an outbound TSV because this does not affect the proof. Because of this labeling scheme, there are no edges between flop nodes. An edge between two TSV nodes exists only if the corresponding TSVs do not have overlapping fan-outs/fan-ins. Similarly, an edge between a TSV node and a flop node exists only if fan-outs/fan-ins of the corresponding flop and TSV do not overlap. An edge between a flop node and a TSV node in the graph means that the corresponding flop and the inbound (outbound) TSV do not have overlapping fan-out (fan-in) cone in the circuit. Similarly, if two TSV nodes in the graph have an edge between them, then the fan-out (fan-in) cones of the corresponding inbound (outbound) TSVs do not overlap in the circuit. We prove below that partitioning $\mathbf{G}$ into k cliques is equivalent to using k cells (both reused flops and wrapper cells) for all the TSVs.

Lemma 4.2.2. *If it is possible to partition* **G** *into k cliques, then $k \geq m$.*

Proof. Since **G** has an independent set of size m, these m nodes will belong to different cliques, i.e., a clique will have at most one of these m nodes. Therefore, the proof follows from the definition of a clique. □

Let us assume that the labeled graph **G** can be partitioned into k cliques. Then it immediately follows from the labeling scheme that a flop can be assigned to a set of TSVs, only if the corresponding nodes in **G** form a clique. Since the edges represent the "non-overlapping condition" between nodes, a clique in **G** corresponds to a feasible assignment.

From the above lemma, a clique cannot have more than one flop node. If a clique consists only of one flop node, then the node is not assigned to any TSV. If a clique consists only of TSV nodes (there are $k-m$ such cliques), then an additional wrapper cell is required. If **G** can be divided into k cliques, then the number of additional wrapper cells needed is equal to $k - m$.

Conversely, it can be seen that if k cells are needed for providing testability to all the TSVs (and connected logic), then **G** can be partitioned into k cliques. This follows from the fact that a flop node and a set of TSV nodes form a clique in **G** only if the corresponding flop node can be assigned to the corresponding TSVs. Also, if a group of TSVs is assigned a new wrapper cell, then the corresponding TSV nodes form a clique in **G**.

Refer to Figure 4.8 as an example. The figure shows a one-to-one correspondence between a general instance of CPP and the transformed instance of WCM. In this example, $m = 2, n = 3$ and the number of cliques (k) formed is 2. Therefore, the number of additional wrapper cells needed is $k - m = 0$. The graph is labeled after running the labeling procedure in Figure 4.7. Since the labeling procedure is

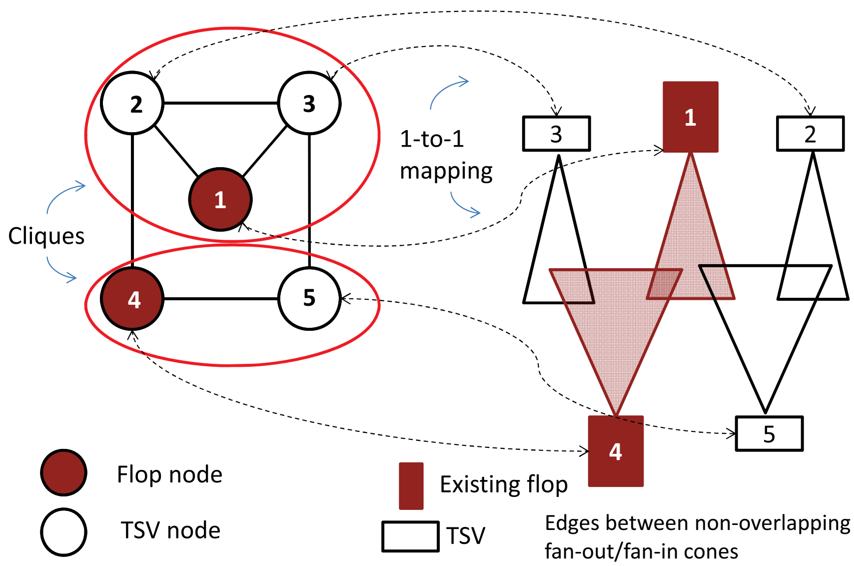


FIGURE 4.8: Illustrating a one-to-one mapping from a general instance of CPP to an instance of WCM.

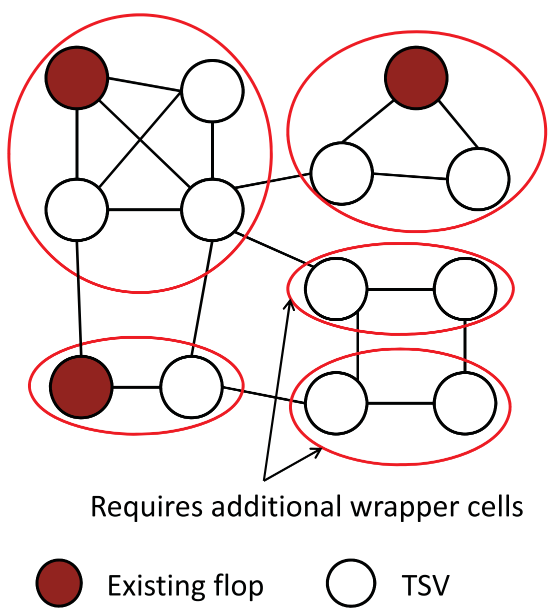


FIGURE 4.9: An example of a clique partition on a labeled graph.

deterministic, there is a one-to-one mapping between the two instances. Moreover, the run-time complexity of the labeling procedure is $O(|V|^2)$; therefore, the instance

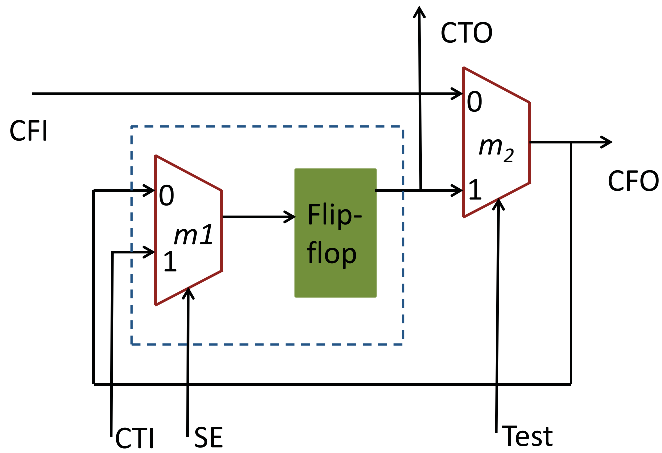


FIGURE 4.10: A typical IEEE Std. 1500 wrapper cell.

transformation step runs in polynomial time.

Figure 4.9 shows another example with $m = 3, n = 10$, and $k = 5$, and the number of additional wrappers inserted is $k - m = 2$.

Based on the above discussion, we can conclude that WCM is NP-hard, since we have carried out a polynomial transformation of a given instance of CPP to an instance of WCM.

4.3 Design Enhancement to Support Die-level Modular Testing

A DfT architecture based on IEEE Std. 1500 for supporting modular testing of dies and embedded cores is proposed in [99]. This wrapper design is also the basis for ongoing IEEE P1838 standardization efforts for 3D-stacked ICs [100]. Figure 4.10 shows an IEEE Std. 1500 wrapper cell. The combination of the multiplexer m_1 and the flip-flop shown in the figure can be viewed as a scan flip-flop; therefore, the select line of m_1 is referred to as *scan enable* or SE. Note that the wrapper cell enables data capture on the line CFI, which is not possible through the reuse methodology

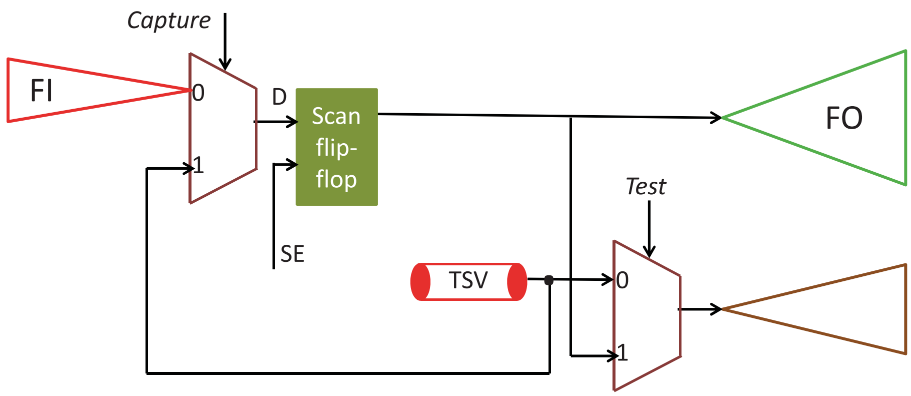


FIGURE 4.11: Reusing an existing flop for both controlling the fan-out cone of an inbound TSV and observing the value on the TSV. FI and FO are fan-in and fan-out cone of the scan flop, respectively.

described in Figure 4.3(b) for inbound TSVs. Similarly, using the solution proposed in Figure 4.5(b), the flop that is used for observing faults present in the fan-in cone of a outbound TSV cannot be used for controlling the TSV. In contrast, the 1500 output wrapper cell can control the outbound TSV that is wrapped. These shortcomings make the reuse scheme incompatible to modular testing of dies during post-bond test, as envisioned in [99]. Therefore, we propose DfT enhancements that support the various operating modes of IEEE Std. 1500.

Figure 4.11 shows how a multiplexer is added for capturing the values on an inbound TSV. When the *Capture* line is asserted, the scan flop captures the value on the TSV line, otherwise the flop captures the output on its fan-in cone; therefore, this design does not interfere with the normal mode. Since we want a flop to be reused multiple times, XOR gates are used for compacting the values on multiple TSV lines (see Figure 4.12).

For solving the controllability problem of the outbound TSVs, we require an extra multiplexer per TSV. Figure 4.13 describes the proposed scheme. A high value on the *Transfer* line pushes the content of the scan flip flop to the outbound TSVs. The

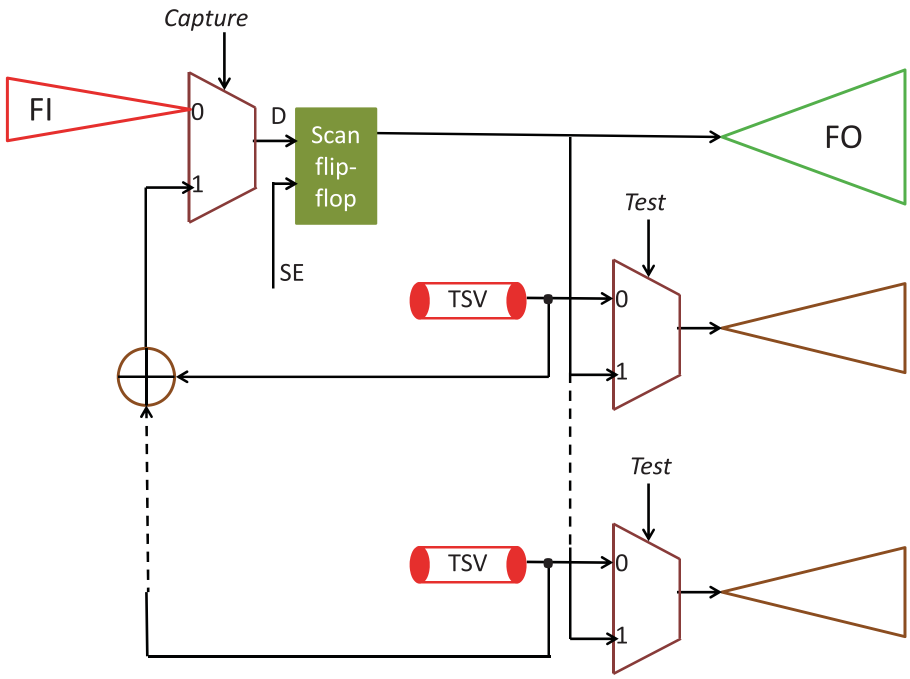


FIGURE 4.12: Reusing an existing flop for multiple inbound TSVs. FI and FO are fan-in and fan-out cone of the scan flop, respectively.

design enhancements made for both inbound and outbound TSVs together mimic the behavior of 1500 Std wrapper cells under various operating modes. Just as the wrapper cells are stitched together to form a ring around a die in the die-wrapper architecture [99], the scan flops that are being reused are stitched together to form a chain.

Table 4.4: Comparison of IEEE Std. 1500 wrapper cell with the proposed DfT architecture

Mode	Inbound					Outbound				
	IEEE Std. 1500		Reuse-based			IEEE Std. 1500		Reuse-based		
	Test	SE	Capture	Test	SE	Test	SE	Transfer	Test	SE
Normal	0	X	0	0	0	0	X	0	0	0
Inward-facing	1	1	X	1	1	0	0	X	1	0
Outward-facing	0	0	1	X	0	1	1	1	X	1
Safe	1	1	0	1	1	1	1	1	0	1

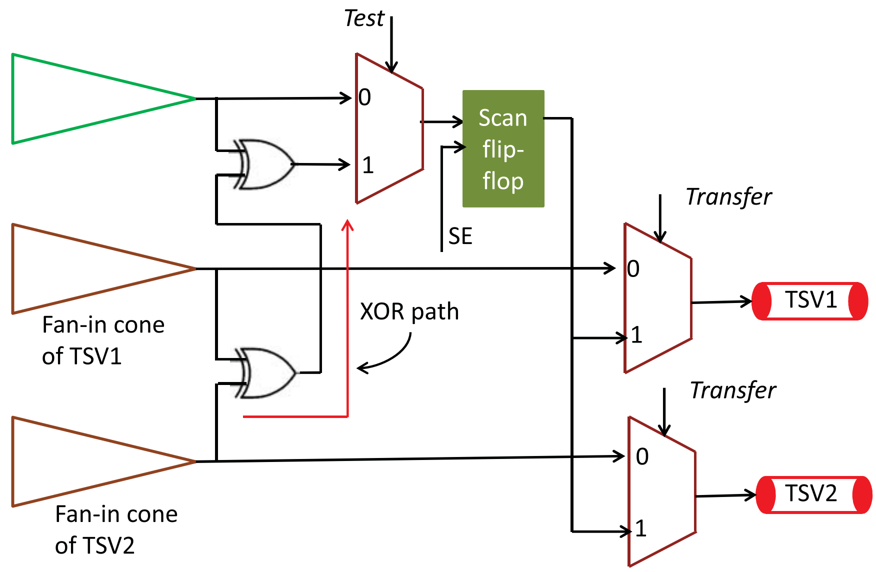


FIGURE 4.13: Reusing an existing flop for multiple outbound TSVs.

Table 4.4 lists various modes that are supported by a wrapper boundary register (WBR) in IEEE Std. 1500 [101]. The table compares signal values of IEEE Std. 1500 wrapper cells with that of the proposed DfT to support those modes, for both inbound and outbound TSVs. These modes collectively form the basis of the rich instruction set of the IEEE 1500 standard. In the inward-facing mode, the inputs are controlled in the IEEE 1500 standard. In the proposed scheme, the inbound TSVs are controlled by setting the appropriate signals, as shown in the table. The outputs or the outbound TSVs are controlled in the outward-facing mode. The inputs (outputs) are observed during the outward-facing (inward-facing) mode. The safe state allows the setting of inputs and outputs to a known state. The symbol 'X' indicates a don't-care. While shifting test patterns during INTEST, the *Capture* signal can be a don't-care (because the scan flops will be in the shift mode), but it should be set to 0 during the capture cycle to capture the values on the fan-in cones of the reused scan flops.

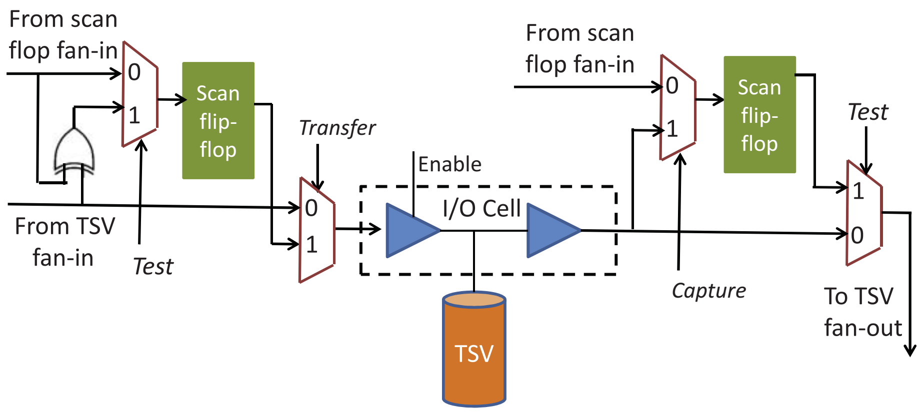


FIGURE 4.14: DfT enhancement for a bidirectional TSV.

---

contract($\mathbf{G}, e$) //e is an edge of $\mathbf{G}$

1: Put the nodes on the two ends of e, say p and q, in the same clique.
2: Replace the nodes p and q with a single node r. A vertex v in $\mathbf{G}$ is connected to r only if v is common neighbor to p and q. This implies that all edges which are incident on p and not on q, and vice versa, are deleted.
3: **return** modified graph $\mathbf{G}, r$.

FIGURE 4.15: Procedure *contract*.

For a bidirectional TSV, DfT structures must be added for both fan-in and fan-out cones of the TSV. Figure 4.14 illustrates the proposed design for a bidirectional TSV.

4.4 Heuristic Methods and ILP Model

As discussed in Section III, CPP and graph coloring are equivalent problems, and since we can solve WCM by targeting CPP, we can use graph coloring in our work. We adopt Tseng's algorithm from [102] and Method-1 from [103]. We refer to these adaptations as "One-Clique-at-a-Time" (OCAT) and "Multiple-Cliques-at-a-Time" (MCAT), respectively. We next describe below these algorithms, and following that we discuss an ILP model for generating optimal (exact) solutions.

4.4.1 OCAT Algorithm

OCAT Algorithm

Pick the edge (p, q) that has the maximum number of common neighbors (a vertex is a common neighbor of an edge if it is connected to both vertices of the edge).
If the graph has no edges, then *Terminate*.
In case of multiple choices, choose the edge (p, q) that results in the deletion of the fewest number of edges due to $contract(\mathbf{G}, (p, q))$.
$\mathbf{G}, r \leftarrow contract(\mathbf{G}, (p, q))$.
if the vertex r has zero degree **then**
    Go to 1
else
    pick an edge e, such that e is incident on vertex r and the vertex on the other end of e has the maximum number of common neighbors with r.
    In case of multiple choices, choose edge e that results in the deletion of the fewest number of edges due to $contract(\mathbf{G}, e)$.
end if
$(p, q) \leftarrow e$
Go to 2

FIGURE 4.16: Pseuocode description of the OCAT algorithm.

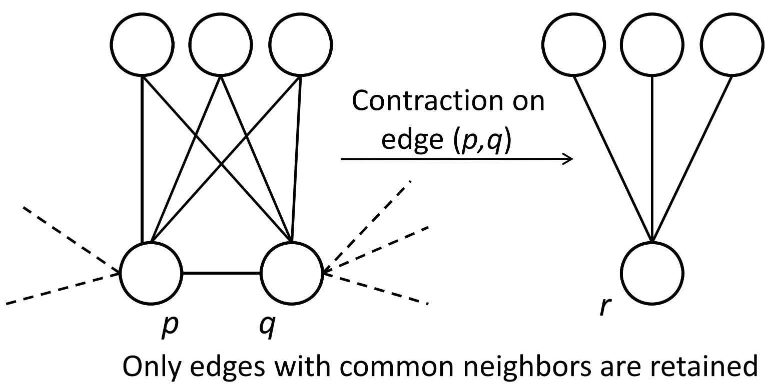


FIGURE 4.17: An example of edge contraction on edge (p,q).

Tseng [102] used clique partitioning for solving optimization problems in the design of processors at the register-transfer level. The problems of allocation of registers, data operators and interconnect units were modeled as CPP, and a heuristic algorithm was developed. This algorithm constructs maximal cliques one at a time, by selecting an edge that is "best" connected, and instantiating a clique with the selected edge. Edge contraction, as explained in Figure 4.15, is applied repeatedly

```
MCAT Algorithm
if all vertices have zero degree then
    Go to 8
end if
Choose a non-zero degree vertex p with least degree. Conflicts are resolved arbitrarily
Choose a vertex q, such that (p, q) ∈ E, and q has smallest degree among all neighbors of p
G, r ← contract(G, (p, q))
Go to
Any vertex not already clustered is mapped to singular cliques (cliques of size 1).
```

Figure 4.18: Pseudocode description of MCAT.

with the selected edge, until no edges are left in the graph. The pseudocode for the adaptation of this algorithm (OCAT) is presented in Figure 4.16.

4.4.2 MCAT Algorithm

This approach also selects edges one by one for edge contraction. However, the order in which edges are selected is different from [102]. Moreover, it does not find maximal cliques in a greedy manner. In [102], maximal cliques are formed one at a time, but in this approach, multiple cliques can be formed simultaneously. We refer to this approach as "multiple-cliques-at-a-time" (MCAT). The algorithm is detailed in Figure 4.18.

4.4.3 Incorporating layout information

Until now, we considered all flops in the die for assignment to a TSV. However, flop assignments should be made under wire-length constraints. The assignment of a flop to a TSV such that the flop is far from the TSV in the circuit layout is not desirable because it can lead to routing congestion, and subsequently to performance degradation. Therefore, we have developed a layout-aware approach that accounts for wire lengths in the optimization method. First, a layout is generated for the original netlist optimization for pre-bond testability. After a graph is constructed corresponding to a WCM instance, an edge between a flop and a TSV is dropped if the distance (wire-length) between the corresponding flop and the TSV is above a preset threshold (l_th). Dropping this edge ensures that the corresponding nodes are not grouped in the same clique. Another practical constraint arises from the placement of TSVs in clusters. Therefore, reusing a scan flop for TSVs belonging to different clusters should be avoided. If TSVs are placed in clusters, edges between TSVs lying in different clusters are dropped in the graph model for MCW. The proposed optimization methods can then be applied to the modified graph.

4.4.4 Incorporating timing constraints

The size of a clique is equal to the number of gates the corresponding flop from the clique has to drive, in addition to the existing gates that the flop was driving. Therefore flop reuse leads to an increase in the capacitive load for that flop, which increases the delay of paths that originate from that flop; see Figure 4.19. This can lead to performance degradation. The size of a clique also determines the number of XOR gates used for compacting the output response on the corresponding TSVs. Although the functional path faces additional delay of only one multiplexer; however, during test mode, the delay in the test path is exacerbated by the presence of multiple XOR gates. Because of these two reasons—increased capacitive load on the driving flop, and increased delay in test paths—we limit the size of the clique that can be formed. The parameter q is used for controlling the size of clique—q is the number of times a flop can be reused.

We must avoid adding multiplexers and XOR gates on critical paths. We must also find all such paths that have the potential to suffer performance degradation due to proposed DfT methodology. We remove those flops and TSVs from reuse consideration that are either sources or sinks to paths that are critical or have the potential to become critical after DfT insertion. We find path delays of the worst path from every scan flop and TSV, and sort them in order of path delays (highest

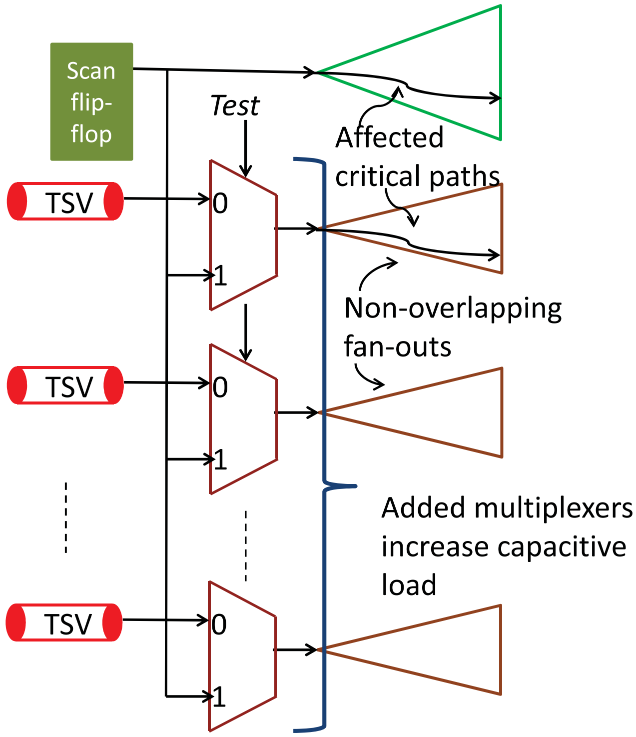


FIGURE 4.19: Illustrating increased capacitive load on a scan flip-flop and delay on functional paths due to reuse.

to lowest). Next we discard a fixed fraction (say 5%) of the flops and TSVs from the top of the list, and thus eliminate potential timing violations. We repeat the same method for flops that are sinks for longest paths in the circuit.

4.4.5 Limiting the reuse of flops

In the above heuristic methods (and in prior work [96, 97]), no limits are placed on the size of cliques that are formed during optimization. In case of a clique corresponding to inbound TSVs, the size of the clique is equal to the number of gates the corresponding flop from the clique has to drive. Hence flop reuse leads to an increase in the capacitive load for that flop, which increases the delay of paths that originate from that flop; see Figure 4.19. If layout data is available, parasitic and timing analysis can be used to rule out some flops from being considered as reuse candidates. A limit must also be placed on the number of times a flop can be reused, say q. Moreover, we can pre-select a fixed number of flops, say p, that can be reused. The search process is guided by the values of p and q. During contraction of edges in the proposed method, whenever a contracted edge creates a clique of size q, one of the following actions is taken:

- If the new clique formed in this way already contains a flop node (this new clique does not have edges to flop nodes), all edges going to intermediate cliques are deleted so that the new clique can get only a TSV node in further iterations.

- If the new clique so formed does not contain a flop, all edges from this clique to other TSVs or intermediate cliques are deleted, so that only a flop node can be selected in subsequent iterations.

If the contraction results in a clique of size $q + 1$, all edges are dropped from the clique. It can be easily seen that using the above edge-deletion rules, any flop or a newly added wrapper cell is not reused more than q times.

Similarly, a limit should be placed on the size of cliques corresponding to outbound TSVs. The XOR gates are placed on fan-in path of the scan flop. Reusing a flop multiple times for providing observability to the logic connected to outbound TSVs leads to performance degradation. Therefore, a limit is also placed on the maximum number of outbound TSVs that can be assigned to the flop.

4.4.6 ILP formulation for CPP

An ILP model allows us to leverage existing solvers to obtain optimal solutions. Even though ILP is not always scalable for large problem sizes, it allows us to evaluate the quality of heuristics by comparing heuristic solutions with exact solutions for smaller problem sizes. In order to develop an ILP model, we consider the following definition.

Definition 4.4.1. *The complement of a graph* **G** *is another graph* **H** *having the same set of vertices such that any two vertices are adjacent in* **G** *if and only if they they are not adjacent in* **H**.

The graph-coloring problem can be formulated as an ILP model. The goal of this problem is to minimize the number of colors required to color all the nodes of a given graph, such that no adjacent nodes are assigned the same color. It can be seen easily that the nodes that are colored using the same color in graph **G** forms a clique in the complement graph **H**, and minimizing the number of colors used in **G** is equivalent to minimizing the number of clique partitions in **H**. The number of vertices (N) in the graph can be used as a bound on the number of colors used. The binary variables y_k and x_{ik} are defined in the following way. If color k ($k = 1, 2, ..., N$) is used, then $y_k = 1$, otherwise $y_k = 0$. The binary variable x_{ik} ($i = 1, 2, ..., N$) indicates whether the node i is assigned the color k. The ILP model is given below.

$$\text{Minimize } \sum_{k=1}^{N} y_k$$

Subject to

$$\sum_{k=1}^{N} x_{ik} = 1 \quad \forall i$$

$$x_{ik} - y_k \leq 0 \ , \ \ x_{ik} + x_{jk} \leq 1 \quad (i, j) \in E, \quad \forall k$$

$$x_{ik}, y_k \quad \text{are binary}$$

The first constraint ensures that each node is colored. The second constraint makes sure that the color k is counted only if color k is used by a node, and the last constraint guarantees that the connected nodes do not get the same color.

Table 4.5: Design details of two-die implementation of the OpenCore benchmarks used in experiments.

Design	Two-die implementation				
	#TSVs	Die 0		Die 1	
		#FFs	#Gates	#FFs	#Gates
des_perf	369	975	11946	1009	12321
cf_rca_16	582	10403	69686	10077	66458
cf_fft_256_8	1569	33918	111320	41805	112230

Table 4.6: Design details of four-die implementation of the OpenCore benchmarks used in experiments.

Design	Four-die implementation								
	#TSVs	Die 0		Die 1		Die 2		Die 3	
		#FFs	#Gates	#FFs	#Gates	#FFs	#Gates	#FFs	#Gates
des_perf	1220	558	6988	517	6657	449	6619	460	5987
cf_rca_16	1451	5363	40863	5357	40339	5040	37862	4720	37560
cf_fft_256_8	2100	19962	78752	17938	71250	20663	78376	70904	17160

Table 4.7: Design details for the ITC'99 benchmarks used in experiments.

Design	#TSVs	Die 0		Die 1	
		#FFs	#Gates	#FFs	#Gates
b01	7	3	23	2	26
b02	6	3	13	1	15
b03	28	13	75	17	85
b04	64	15	370	15	367
b06	16	4	20	5	19
b15	480	195	3858	254	4509
b20	508	247	9736	243	9946
b21	514	248	10011	242	10016
b22	794	363	15881	372	13281

We extend the above basic formulation to put a constraint on the number of times a flop or a wrapper cell can be reused. The constraint can be stated as $\sum_t x_{tk} \leq q$, where t varies over indexes of all TSV nodes.

We use the results obtained by the above ILP model to evaluate the results obtained using the proposed heuristics for smaller benchmarks.

4.5 Results

In this section, we first discuss the benchmark circuits that we use for our experiments. Next we compare the results obtained using the heuristic algorithms discussed in Section 4.4 with that obtained using the method described in [96]. The results are compared to assess the following:

- the number of existing scan flops that are reused, and the number of (additional) wrapper cells that are needed;
- the impact of the DfT solution on stuck-at and transition fault coverage, and the number of test patterns;
- the area overhead.

For our experiments, we use three benchmark circuits, namely *des_perf, cf_rca_16*, and *cf_fft_256_8*, from OpenCore benchmark suite [3], and nine benchmark circuits (*b01, b02, b03, b04, b06, b15, b20, b21,* and *b22*) from ITC'99 benchmarks [104]. For the OpenCore designs, the partitioning was done for performance optimization. For the ITC'99 benchmark circuits, hMetis [105]—a hypergraph-patitioning tool—was used. The designs from ITC'99 are partitioned into two dies, and for OpenCore benchmark circuits we used both two-die and four-die implementation of 3D stack. Table 4.5, 4.6,and Table 4.7 shows design-related details about the two sets of benchmark circuits, respectively.

4.5.1 Results on wrapper insertion and scan flop reuse

Table 4.8 shows the number of wrapper cells needed (N_{WC}) and the number of flops reused (N_F) for each benchmark circuit. Results are provided for prior work [96], OCAT, and MCAT. The table shows aggregated results for all dies for each design for multiple values of p and q. As expected, as the number of available reusable candidates p increases, the number of wrapper cells needed (N_{WC}) decreases when q (the maximum number of times a flop or a cell can be reused) is fixed. Increasing q after fixing p further decreases N_{WC}. The ad hoc heuristic method from [96] is oblivious to the parameter q, hence the results do not change as q varies. The results produced by OCAT, and MCAT are superior in terms of the number of wrapper cells needed. For the designs from OpenCore, we found that very few wrapper cells are needed. This suggests that very few TSVs and flops have overlapping fan-in (fan-out) cones for these set of designs.

We further find that compared to MCAT, the OCAT algorithm is more effective in reducing the number of reused flops (N_F). This can be explained by observing that OCAT creates one clique at a time, and it creates a clique in a greedy manner. However, MCAT performs better than OCAT in minimizing the objective function, i.e., N_{WC}, especially when the number of scan flops that are available for reuse is limited. While MCAT leads to fewer wrapper cells, it tends to require more flop reuse. Hence the choice of the heuristic can be based on the relative priorities of these two measures. Moreover, the computational complexity of MCAT is less than OCAT by a linear factor of the number of vertices in a given graph for solving the clique-partitioning problem [103].

The CPU time for each experiment (each row of Table 4.8) on a 16-core Intel(R) Xeon(R) CPU, running at 2.5 GHz with 16 GB RAM, was in the range of minutes for smaller benchmarks for each method. As the size of circuit increases, MCAT starts performing better than OCAT in terms of the execution time. This is because OCAT also stores the information about common neighbors between two nodes, and has to update the same information after an edge contraction. With the largest benchmark $cf\_fft\_256\_8$, while MCAT, and the approach discussed in [96] ran within a few minutes, OCAT ran for 30 minutes.

In order to study the impact of timing constraints on flop selection, we carried

Table 4.8: The pair (N_F, N_{WC}) for different designs and for different values of (p, q).

Design	(p, q)	Proposed method		Prior work [96]
		OCAT	MCAT	
b20	$(0.5N_{TSV}, 8)$	(151, 120)	(181, 120)	(253, 255)
	$(0.5N_{TSV}, 4)$	(159, 200)	(189, 200)	(253, 255)
	$(0.7N_{TSV}, 8)$	(201, 70)	(233, 70)	(254, 154)
	$(0.7N_{TSV}, 4)$	(209, 70)	(250, 70)	(254, 154)
	$(0.2N_{TSV}, 2)$	(101, 211)	(101, 198)	(101, 407)
b21	$(0.5N_{TSV}, 8)$	(158, 105)	(186, 105)	(256, 255)
	$(0.5N_{TSV}, 4)$	(169, 105)	(201, 105)	(256, 255)
	$(0.7N_{TSV}, 8)$	(210, 53)	(242, 53)	(359, 155)
	$(0.7N_{TSV}, 4)$	(219, 53)	(296, 53)	(359, 155)
	$(0.2N_{TSV}, 2)$	(102, 201)	(102, 194)	(102, 412)
b22	$(0.5N_{TSV}, 8)$	(262, 35)	(336, 19)	(397, 397)
	$(0.5N_{TSV}, 4)$	(262, 35)	(336, 19)	(397, 397)
	$(0.7N_{TSV}, 8)$	(282, 0)	(419, 0)	(555, 139)
	$(0.7N_{TSV}, 4)$	(283, 16)	(434, 0)	(555, 139)
	$(0.2N_{TSV}, 2)$	(158, 242)	(158, 239)	(101, 407)
des_perf	$(0.5N_{TSV}, 8)$	(176, 0)	(585, 0)	(607, 613)
	$(0.5N_{TSV}, 4)$	(408, 0)	(585, 0)	(607, 613)
	$(0.7N_{TSV}, 8)$	(261, 0)	(612, 0)	(850, 370)
	$(0.7N_{TSV}, 4)$	(440, 0)	(612, 0)	(850, 370)
	$(0.2N_{TSV}, 2)$	(242, 494)	(242, 494)	(242, 978)
cf_rca_16	$(0.5N_{TSV}, 8)$	(191, 1)	(368, 1)	(723, 728)
	$(0.5N_{TSV}, 4)$	(398, 1)	(415, 1)	(723, 728)
	$(0.7N_{TSV}, 8)$	(313, 1)	(423, 1)	(1007, 444)
	$(0.7N_{TSV}, 4)$	(483, 1)	(533, 1)	(1007, 444)
	$(0.2N_{TSV}, 2)$	(289, 716)	(289, 716)	(289, 1262)
cf_fft_256_8	$(0.5N_{TSV}, 8)$	(339, 0)	(429, 0)	(1046, 1054)
	$(0.5N_{TSV}, 4)$	(680, 0)	(743, 0)	(1046, 1054)
	$(0.7N_{TSV}, 8)$	(810, 0)	(846, 0)	(1463, 637)
	$(0.7N_{TSV}, 4)$	(835, 0)	(932, 0)	(1463, 637)
	$(0.2N_{TSV}, 2)$	(251, 972)	(251, 972)	(418, 1682)

out a series of experiments as described below. Since we do not have layouts for the netlists for the 3D stack designs, we were not able to extract accurate timing information. Instead, we set the fraction of flops, referred to as α, that cannot be reused (due to timing constraints) to a range of values, e.g., $\alpha = 0.1, 0.2$, and 0.3. For each choice of α, we randomly select αN flops, where N is the total number

Table 4.9: Mean and standard deviation of (N_F, N_{WC}) with randomly selected reuse candidates for various designs.

Design	α	Random selection		Non-random selection	
		Mean	Std. Dev.	MCAT	Ad hoc
b15	0.1	(336, 8.4)	(3.6, 2.8)	(339, 5)	(404, 76)
	0.2	(306.4, 19.8)	(6, 4.6)	(307, 11)	(359, 121)
	0.3	(280.65, 33.25)	(5.9, 4.4)	(280, 24)	(314, 166)
b20	0.1	(305.6, 23)	(7.3, 4.2)	(310, 18)	(441, 67)
	0.2	(294.8, 50)	(15.4, 6.7)	(324, 37)	(392, 116)
	0.3	(284, 76.2)	(12.8, 5.9)	(280, 70)	(354, 154)
b21	0.1	(324, 14.1)	(5.1, 3.1)	(331, 9)	(414, 100)
	0.2	(320.2, 38.3)	(8.2, 6)	(328, 29)	(388, 126)
	0.3	(314.6, 66.7)	(11.8, 5.3)	(338, 53)	(359, 155)

of flops in the die, and mark these flops as not reusable. The random selection is repeated 20 times to obtain some level of statistical significance. Table 4.9 shows the results (mean and standard deviation) on the pair of flops that are reused and the number of wrapper cells required for testability in these cases. These results are compared to that obtained with no constraints on specific reuse, but an upper limit of $1-\alpha$ is set on the fraction of flops that can be reused (under MCAT column). We also report comparative data obtained using the ad hoc heuristic from [96]. We note that more wrapper cells are required when specific flops are deemed non-reusable, since flops can no longer be freely reused. In practice, when layout data and accurate timing information is available, our method can be used by simply deleting nodes in the graph that correspond to flops that cannot be reused.

4.5.2 ATPG results

To demonstrate that the overall fault coverage is not adversely affected by the proposed DfT method, we compare fault coverage for different methods. After assignment of flops were made to the TSVs, the netlist was edited to insert multiplexers and XOR gates at locations specified by results from all the methods. A commercial ATPG tool was run to measure the fault coverage and number of patterns needed

Table 4.10: Stuck-At Fault Coverage (FC) and test pattern count (PC) for various methods for the benchmark circuits.

Design	(p,q)	OCAT		MCAT		Ad hoc [96]		With wrapper cells	
		FC	PC	FC	PC	FC	PC	FC	PC
b20	$(0.5N_{TSV}, 8)$	99.98%	2683	99.98%	2655	99.98%	2719	99.98%	2683
	$(0.5N_{TSV}, 4)$	99.98%	2642	99.98%	2688	99.98%	2719		
	$(0.7N_{TSV}, 8)$	99.98%	2644	99.98%	2652	99.98%	2643		
	$(0.7N_{TSV}, 4)$	99.98%	2678	99.98%	2657	99.98%	2643		
	$(0.2N_{TSV}, 2)$	99.98%	2699	99.98%	2590	99.98%	2638		
b21	$(0.5N_{TSV}, 8)$	99.99%	2878	99.99%	2936	99.99%	2863	99.99%	2893
	$(0.5N_{TSV}, 4)$	99.99%	2893	99.99%	2908	99.99%	2863		
	$(0.7N_{TSV}, 8)$	99.99%	2883	99.99%	2929	99.99%	2906		
	$(0.7N_{TSV}, 4)$	99.99%	2926	99.99%	2920	99.99%	2906		
	$(0.2N_{TSV}, 2)$	99.99%	2927	99.99%	2856	99.99%	2912		
b22	$(0.5N_{TSV}, 8)$	99.97%	3254	99.97%	3236	99.97%	3295	99.97%	3170
	$(0.5N_{TSV}, 4)$	99.97%	3265	99.97%	3323	99.97%	3295		
	$(0.7N_{TSV}, 8)$	99.97%	3281	99.97%	3245	99.97%	3230		
	$(0.7N_{TSV}, 4)$	99.97%	3257	99.97%	3271	99.97%	3230		
	$(0.2N_{TSV}, 2)$	99.97%	3215	99.97%	3298	99.97%	3264		
des_perf	$(0.5N_{TSV}, 8)$	100 %	832	100%	839	100%	867	100%	845
	$(0.5N_{TSV}, 4)$	100%	842	100%	873	100%	883		
	$(0.7N_{TSV}, 8)$	100%	857	100%	851	100%	864		
	$(0.7N_{TSV}, 4)$	100%	862	100%	855	100%	891		
	$(0.2N_{TSV}, 2)$	100%	830	100%	863	100%	886		
cf_rca_16	$(0.5N_{TSV}, 8)$	99.92 %	6754	99.92%	6798	99.92%	6865	99.92%	6688
	$(0.5N_{TSV}, 4)$	99.92%	6805	99.92%	6821	99.92%	6895		
	$(0.7N_{TSV}, 8)$	99.92%	6831	99.92%	6820	99.92%	6923		
	$(0.7N_{TSV}, 4)$	99.92%	6822	99.92%	6836	99.92%	6902		
	$(0.2N_{TSV}, 2)$	99.92%	6742	99.92%	6778	99.92%	6856		
cf_fft_256_8	$(0.5N_{TSV}, 8)$	99.92 %	17368	99.92%	17354	99.92%	17532	99.92%	17283
	$(0.5N_{TSV}, 4)$	99.92%	17393	99.92%	17411	99.92%	17545		
	$(0.7N_{TSV}, 8)$	99.92%	17418	99.92%	17424	99.92%	17534		
	$(0.7N_{TSV}, 4)$	99.92%	17354	99.92%	17365	99.92%	17531		
	$(0.2N_{TSV}, 2)$	99.92%	17372	99.92%	17355	99.92%	17504		

to achieve that coverage. The results obtained from each method are compared with the case when wrapper cells are used for each TSV; see Table 4.10. As expected, irrespective of the approach used to optimize the wrapper-cell count, the fault coverage does not deviate from its value when wrapper cells were inserted at each TSV. The number of test patterns is only moderately higher in most of the cases, and even lower in some cases. While it is not clear which method is more effective in minimizing the pattern count, the number of bits per pattern for MCAT and OCAT

Table 4.11: Transition Fault Coverage (FC) and test pattern count (PC) for various methods for the ITC'99 benchmark circuits.

Design	(p, q)	OCAT		MCAT		Ad hoc [96]		With wrapper cells	
		FC	PC	FC	PC	FC	PC	FC	PC
b20	$(0.5N_{TSV}, 8)$	99.23%	4408	99.24%	4438	99.27%	4385	99.28%	4508
	$(0.5N_{TSV}, 4)$	99.24%	4383	99.24%	4454	99.27%	4385		
	$(0.7N_{TSV}, 8)$	99.18%	4420	99.22%	4395	99.22%	4396		
	$(0.7N_{TSV}, 4)$	99.21%	4340	99.22%	4414	99.22%	4396		
	$(0.2N_{TSV}, 2)$	99.21%	4401	99.25%	4405	99.22%	4418		
b21	$(0.5N_{TSV}, 8)$	99.67%	4868	99.67%	4777	99.69%	4785	99.42%	4826
	$(0.5N_{TSV}, 4)$	99.66%	5077	99.67%	4860	99.69%	4785		
	$(0.7N_{TSV}, 8)$	99.68%	4850	99.68%	4847	99.67%	4812		
	$(0.7N_{TSV}, 4)$	99.67%	4883	99.67%	4829	99.67%	4812		
	$(0.2N_{TSV}, 2)$	99.68%	4857	99.68%	4856	99.7%	4824		
b22	$(0.5N_{TSV}, 8)$	99.27%	5461	99.28%	5409	99.31%	5451	99.27%	5397
	$(0.5N_{TSV}, 4)$	99.30%	5436	99.29%	5489	99.31%	5451		
	$(0.7N_{TSV}, 8)$	99.25%	5399	99.26%	5415	99.26%	5508		
	$(0.7N_{TSV}, 4)$	99.27%	5396	99.26%	5452	99.26%	5508		
	$(0.2N_{TSV}, 2)$	99.26%	5378	99.29%	5430	99.30%	5414		

Table 4.12: Table comparing N_{WC} from different methods.

Design	N_F Obtained			N_{WC} Obtained		
	OCAT	MCAT	ILP	OCAT	MCAT	ILP
b01	3	4	4	7	7	7
b02	1	1	1	5	5	5
b03	11	15	10	4	3	3
b04	34	38	35	0	0	0
b06	7	8	9	3	3	3

is much lower than for [96] because the number of wrapper cells inserted using these methods is significantly lower. As shown in Table 4.11, similar results were observed for transition fault coverage.

4.5.3 Comparison with ILP model

Table 4.12 shows the optimum number of additional wrapper cells (N_{WC}) needed for each small circuits taken from the set of ITC99 benchmarks. The table reports the cumulative number of additional wrapper cells needed for both the dies. The variable q was fixed to a value of 4 and p was fixed at $0.7 \cdot N_{TSV}$ for all the experiments. The table also reports the number of flops that need to be reused. While the MCAT

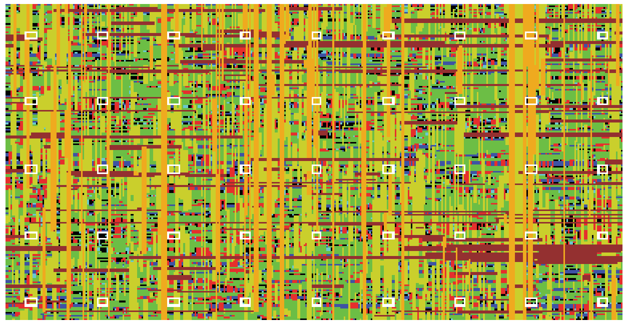


FIGURE 4.20: A zoomed-in region of Die 0 of rca_cfa_16. TSVs are highlighted with small white squares.

procedure yielded an optimal solution (in terms of N_{WC}) in all the cases, the OCAT procedure deviated from the optimum value for the *b03* benchmark.

These results indicate that the heuristic methods are very effective and they tend to provide optimal or near-optimal results.

4.5.4 Layout and timing-aware assignment of flops to TSVs

We used Cadence Encounter to perform place-and-route and clock-tree synthesis for all dies separately using the 45nm Nangate standard cell library. In the floor plan, we distributed the TSVs evenly across a die forming a grid in order to keep the TSV density (number of TSVs per unit area) uniform across the die area. The TSVs have a keep-out zone with a width of four times the minimum-sized inverter, and the height of a standard cell. A zoomed-in region of Die 0 of cf_rca_16 along its edge is shown in Figure 4.20. The figure highlights the TSVs with small white squares; note that the TSVs are placed at regular intervals in a grid.

The distance between a TSV and a flop is estimated by the Euclidean distance between the coordinates of the TSV and the D-pin of the flop. Similarly, the distance between two TSVs is calculated using the Euclidean metric. If this distance is below a threshold value (l_{th}), then the corresponding edge, if it exists, is dropped from the graph. We also filtered out TSVs and flops from the graph model based on the

Table 4.13: Table showing the effect of varying the layout threshold on N_F, N_{WC}, stuck-at fault coverage (FC), pattern count (PC) of individual dies for various designs. The number of edges in the corresponding CPP instance is also shown.

Design	l_{th}	Die 0					Die 1				
		N_F	N_{WC}	FC	PC	#edges	N_F	N_{WC}	FC	PC	#edges
$b20$	20	91	98	98.93%	1245	836	111	80	99.14%	1431	882
	40	111	70	99.22%	1264	3032	125	34	99.63%	1480	2923
	60	120	37	99.56%	1232	6746	143	4	99.95%	1461	6405
	80	129	21	99.76%	1234	11634	149	0	100%	1413	10749
	100	159	2	99.94%	1216	16549	149	0	100%	1453	14561
$b21$	20	109	84	99.09%	1484	847	115	74	99.21%	1405	952
	40	127	39	99.56%	1508	2653	123	40	99.57%	1449	3209
	60	136	18	99.79%	1519	5705	137	5	99.94%	1439	7070
	80	154	0	99.97%	1522	9531	138	0	100%	1457	11560
	100	158	0	99.98%	1502	13264	140	0	100%	1473	15520
$b22$	20	133	132	99.09%	1763	939	190	90	99.28%	1523	1446
	40	171	75	99.47%	1794	3292	207	27	99.76%	1548	4999
	60	184	24	99.82%	1768	7345	226	9	99.90%	1529	11089
	80	187	10	99.90%	1786	13156	213	0	99.98%	1522	19017
	100	185	1	99.97%	1770	18836	210	0	99.98%	1536	26467
des_perf	20	81	48	99.67%	275	2878	159	68	99.56%	265	3713
	40	89	39	99.73%	262	7740	191	33	99.76%	263	8224
	60	95	33	99.82%	267	13068	214	14	99.93%	258	17263
	80	102	15	99.88%	258	18827	228	1	99.95%	262	24327
	100	110	2	99.93%	260	25264	229	0	99.96%	257	28324
cf_rca_16	20	204	15	99.88%	3871	12220	250	20	99.79%	3670	13812
	40	212	11	99.90%	3865	20082	265	8	99.78%	3658	28225
	60	220	1	99.92%	3856	28282	262	0	99.84%	3654	37733
	80	221	0	99.94%	3849	38873	260	0	99.90%	3662	46525
	100	216	0	99.94%	3852	49134	257	0	99.90%	3648	51354
$cf\_fft\_256\_8$	20	509	75	99.32%	4345	30454	519	68	99.62%	4588	31262
	40	517	53	99.56%	4322	47317	534	32	99.76%	4581	49158
	60	525	29	99.82%	4328	64192	545	14	99.73%	4572	68253
	80	534	15	99.95%	4337	79854	547	6	99.86%	4552	85673
	100	539	1	99.96%	4327	93115	551	0	99.89%	4566	96849

discussion in Section 4.4.D. Synopsys PrimeTime was used for creating a sorted array of flops based on timing delays of paths to and from the flops. We ran experiments on the benchmark circuits by varying l_th, and measured its effect on the values of N_F, N_{WC}, stuck-at fault coverage, and pattern count. The results are shown in Table 4.13. Since MCAT is computationally less expensive and provides results as good

Table 4.14: Table highlighting the effect of adding modular testing support to various designs.

Design	Without support for modular testing			With support for modular testing						
				Die 0			Die 1			Overall test time (in cycles)
	PC	SC	Test time (in cycles)	PC	SC	Test time (in cycles)	PC	SC	Test time (in cycles)	
*b*20	2201	492	1085585	1216	249	304249	1453	243	354775	659024
*b*21	2389	490	1173489	1502	248	374246	1473	242	358181	732427
*b*22	2708	736	1996532	1763	364	643859	1523	372	568451	1212310
A	272	1984	541904	260	975	254735	257	1009	260579	515314
B	7598	20480	1.56×10^8	3852	10403	4.01×10^7	3648	10077	3.68×10^7	7.69×10^7
C	5282	75723	4.00×10^8	4327	33918	1.47×10^8	4566	41805	1.91×10^8	3.38×10^8

A: des_perf B: cf_rca_16 C: $cf\_fft\_256\_8$ PC: #patterns SC: Scan chain length

as the OCAT approach, we used the former for obtaining our results. As expected, on increasing l_{th}, the number of existing flops that were reused increased and the number of additional wrapper cells required reduced. There is no significant change in the pattern count or the fault coverage. For each case, the table also shows a column for the number of edges in the corresponding CPP instance. As we increase l_{th}, the number of edges increases considerably. The increase in the number of edges indicate that for every TSV, more flops are available for reuse. The runtime of the optimization procedure is also affected because of this increase.

In order to verify that the modified netlists do not suffer from performance degradation, we compared slack time of the critical paths of the modified netlists with that of the corresponding netlists having one wrapper cell dedicated to each TSV. For filtering flops and TSVs that are either sources or sinks to the critical paths, we used the technique described in Section 4.4.D. When zero flops or TSVs were filtered, we saw performance degradation along critical paths because of the addition of multiplexers on those paths. Moreover, we saw that the worst path in our approach is always an 'XOR path' added in the designs to observe the faults on fan-in cones of multiple outbound TSVs (see Figure 4.13). However, these cannot be sensitized in functional mode, and they are activated only during test mode. Moreover, only the single stuck-at faults are detected along such XOR paths, and the clock frequency needed for test is much smaller than the functional frequency. Therefore, transition

Table 4.15: Table comparing the area (in square microns) overhead on various designs with a dedicated wrapper for each TSV and with reuse-based method

Design	Die 0					Die 1				
	Original die area	With wrapper		Reuse-based		Original die area	With wrapper		Reuse-based	
		Area	OH	Area	OH		Area	OH	Area	OH
$b20$	12946	16636	28.5%	15742	21.6%	13333	17023	27.67%	14570	9.27%
$b21$	13453	17185	27.74%	14759	9.70%	13428	17163	27.81%	14651	9.1%
$b22$	21196	26967	27.21%	22992	8.47%	17748	23526	32.54%	19817	11.65%
A	20533	24609	19.85%	22738	10.73%	21118	25247	19.55%	23252	10.10%
B	144060	166464	15.55%	147701	2.5%	138611	160416	15.73%	142367	2.71%
C	361297	431540	19.44%	367872	1.82%	409628	493859	20.56%	416837	1.76%

A: des_perf B: cf_rca_16 C: $cf\_fft\_256\_8$ OH: Area overhead

faults will not be targeted along these paths, and they can be discarded from consideration as the "worst-case" longest paths. After removing 'XOR paths' and further removing the top 5% of flops and TSVs that are potential sources or sinks to critical paths (as discussed in Section 4.4.D.), we did not find performance degradation in the designs.

To evaluate the benefits provided by our DfT technique to support die-level modular testing, we compared test time obtained with the modular approach to that obtained using a non-modular approach, i.e., ATPG on the flattened netlist. We generated a netlist for each die of a design without adding the DfT structures needed for post-bond testing; however, the optimization necessary for providing pre-bond testability was carried out. A top-level netlist was generated for each design that instantiates the individual dies, and ATPG results were obtained on the new netlists. The results are compared to the ATPG results obtained from the case when we supported modular testing in Table 4.14. In the latter case, we can retarget the patterns generated during pre-bond testing; therefore, the test time for post-bond testing is obtained by adding the test time of individual dies. Since interconnect testing only marginally affects the overall test time, it is ignored in this example. The value of l_{th} was set to 100 microns and $q = 4$ was used in each case. We observed a significant savings in test time with the modular approach. Note that in practice, some amount of ATPG on the flattened design might be desirable to test the interconnects between dies by sending signals across die boundaries.

Table 4.15 compares the area overhead due to the reuse-based method with that from the case when one wrapper cell is dedicated to each TSV. We see considerable savings in area overhead because of the reuse-based method. The gain is expected to be even more significant for larger circuits with tens of thousands of TSVs and hundreds of thousands of scan flops.

4.6 Conclusion

We have studied the problem of minimizing the wrapper-cell count in 3D-stacked ICs using a formal approach based on graph theory. We have proven that this problem is NP-hard by constructing a polynomial-time reduction from the clique-partitioning problem, which is known to be NP-complete. Since the clique-partitioning problem is also equivalent to the graph-coloring problem, this work paves the way for the adoption of heuristic algorithms for graph coloring, instead of less effective ad hoc techniques. We have implemented two such heuristic algorithms for the wrapper-cell minimization problem. We have further shown how timing and layout information can be incorporated in the graph model to address the problem of increased capacitive load and delay on critical paths due to multiple flop reuse. An ILP model has also been presented to evaluate the quality of results obtained from the heuristic methods. Experimental results for benchmark circuits show that the proposed methods can reduce the number of wrapper cells significantly without affecting fault coverage or pattern count.

5

A Distributed, Reconfigurable, and Reusable BIST Infrastructure for 3D-Stacked ICs

Test cost has emerged as a potential bottleneck in the adoption of 3D ICs. In order to contain the rapidly escalating test cost, reusability of test structures and strategies has been advocated in the literature [17, 57, 99]. Built-in self-test (BIST) is known to be an effective reuse-based test technique, which simplifies test application and requires minimal external intervention. All prior work on designing an end-to-end BIST methodology for launching BIST runs are meant for traditional two-dimensional SOCs and extending these methods to the third dimension is not trivial.

As discussed in Chapter 3, tests can be applied at many possible test stages or *test insertions* during the manufacturing of 3D-stacked ICs. Since every die should be tested in isolation (prior to bonding), and also after stacking, a centralized test controller is not useful for driving BIST during every testing stage. Therefore, we must have a distributed BIST infrastructure support. To avoid redesign effort and die area, the same BIST infrastructure should be reusable at every testing stage. In addition, test schedules are not expected to be the same at every test insertion due to power and resource constraints. Therefore, flexibility (or reconfigurability) of the BIST framework is desired to launch BIST according to a given static test schedule. In this chapter, we propose a BIST framework that addresses the above challenges and also propose two test-scheduling algorithms—adapted from well-known algorithms for for solving 2D bin-packing problem—for minimizing test time under power and resource constraints.

The remainder of the chapter is organized as follows. Section 5.1 provides motivation of the work and list chapter contributions. Section 5.2 discusses related prior work. Section 5.3 provides details of the proposed BIST framework and how this framework is related to the test-scheduling problem. The test-scheduling problem for BIST-ready cores under resource and power constraints is described in Section 5.4. Results are presented in Section 5.5 and conclusions are drawn in Section 5.6.

5.1 Motivation and Chapter Contributions

A large body of BIST work is based on the development of a single centralized test controller ([106], [107]) that performs device-level BIST scheduling and supports only session-based scheduling (one session at a time). In 3D integration of ICs, multiple dies are integrated to form a stack and tests can be applied at many possible test insertions. In order to enable BIST-based testing at multiple test insertions, there is a need for a distributed BIST framework. The building blocks of the framework should work in isolation during pre-bond (i.e., before stacking) testing, as well as interact with each other at post-bond (i.e., after stacking) test stages. Reusability of test structures is a key principle that guides our design of a distributed BIST infrastructure. The BIST framework should be reusable across dies and across test insertions to minimize redesign effort and overall silicon footprint.

The impact of the choice of test flow on the overall cost of manufacturing and testing of 3D ICs is analyzed in [6, 81, 82]. The choice of test flow, i.e., what tests are used and when they are applied during 3D integration ("what to test", "when to test") affects the cost. To improve yield of the 3D integration process and to manage escalating test cost, the BIST framework should provide reconfigurability to enable the simultaneous testing of multiples cores and the selective application of tests at various test insertions.

Test schedules created for pre-bond testing of individual dies may not be applicable during post-bond testing because multiple dies can be tested concurrently and the use of the same schedule can lead to the violation of power constraints. A sequential approach to the testing of dies is likely to lead to sub-optimum test times. Therefore, it is desirable to generate test schedules for every test insertion separately. Resource sharing is another reason for creating test schedules multiple times. Since a single BIST resource can be shared by multiple cores, resource contention should also be considered during the creation of test schedules. For example, a memory BIST (MBIST) or a built-in self-repair (BISR) block can be used for testing or self-repairing multiple memory cores [108, 109]. Moreover, a BIST resource can be used for testing a core located on a different die, e.g., memory vendors may not choose to add MBIST logic on memory dies with regular structures. Although there is published work ([107], [9]) on BIST methods that support scheduling, there is no work known to us that integrates scheduling with a BIST framework in the context of 3D ICs.

We address the above test challenges in this work. The key contributions of this chapter are as follows.

- A unified framework for end-to-end BIST-based testing of embedded cores, dies, and interconnects is proposed.
- Our BIST framework is distributed, reusable, and reconfigurable. The BIST controllers used in the pre-bond testing stage can also be used during later

stages of testing. The controllers can be easily programmed to selectively apply tests when required.

- A method to translate a test schedule into a set of BIST control instructions.
- A test-scheduling problem under power and resource constraints is formulated, and two solutions based on rectangle-packing algorithms are described.
- An architecture that is compatible with emerging 3D test standards, such as IEEE P1838 test architecture for 3D-stacked ICs.

5.2 Related Prior work

A framework for interconnect testing of 3D stacks consisting of memory-on-logic dies was proposed in [108]. A cost-benefit analysis was carried out for sharing MBIST for multiple memory dies and integrating MBIST controllers on logic dies. BIST solutions based on ring oscillators for pre-bond testing of TSVs were proposed in [110, 111]. For post-bond testing of TSVs, a March test was developed for detecting crosstalk faults in neighboring TSVs [112]. These methods address the problem of testing specific faults arising at different stages of testing, but the general problem of designing a unified BIST infrastructure was not considered.

When multiple cores, which may be located on different dies in a stack, are tested concurrently, test schedules must be created to avoid violation of power constraints. Scheduling imposes constraints on when a particular test can be carried out. Prior work [113] on creating test schedules for BIST-ready cores in 3D-stacked ICs considered the scheduling problem without imposing resource constraints, and ignored how a schedule should be loaded in the stack such that once the schedule is loaded, all tests are executed without external intervention.

For two-dimensional SOCs, a hierarchical BIST infrastructure was proposed in

[9], including a method to capture the scheduling information through special BIST control instructions. A central test processor that interfaces with test-access port (TAP) was designed to communicate with other embedded test processors at the next level of hierarchy, i.e., at the module level. As shown in Fig. 5.1(a), a module can consist of several submodules. The embedded test processors are pre-loaded with test programs or instructions for testing the internal logic of the module. An embedded test processor can be programmed to execute BIST on any submodule. The central processor can control the order in which the embedded test processor can execute instructions to test the submodules; therefore, the submodules within a module can be tested in an arbitrary order. Minimizing test time under resource and power constraints entails concurrent testing of cores that are time-compatible [114], and a test has to wait for incompatible tests to be completed before it can be scheduled. Because the programs are statically stored in the test processors, it may not be possible to achieve a desired degree of test concurrency or to model scheduling dependency between all possible pairs of tests. This problem is exacerbated when we consider testing of stacks (partial or final) in a 3D context, where cores in multiple dies can be tested concurrently under power constraints.

Second, the order of instruction execution is sequential in nature in [9]; the processor halts on `WAIT` instructions introduced in [9] until the BIST on which the execution has halted is completed. Suppose the test processor is waiting for test A, a different BIST (say B) has reached completion, and another test C is waiting for B. Then C cannot start execution until the test processor resumes execution after A's completion. However, a test program can be reordered to overcome this drawback; see Figure 5.1(b). Although `WAIT` is a desirable instruction for meeting constraints imposed by resource and power, we lose flexibility in test programming. With an incorrectly written program, we run the risk of increasing test time and violating power constraints. Our infrastructure overcomes this problem by not halting on any

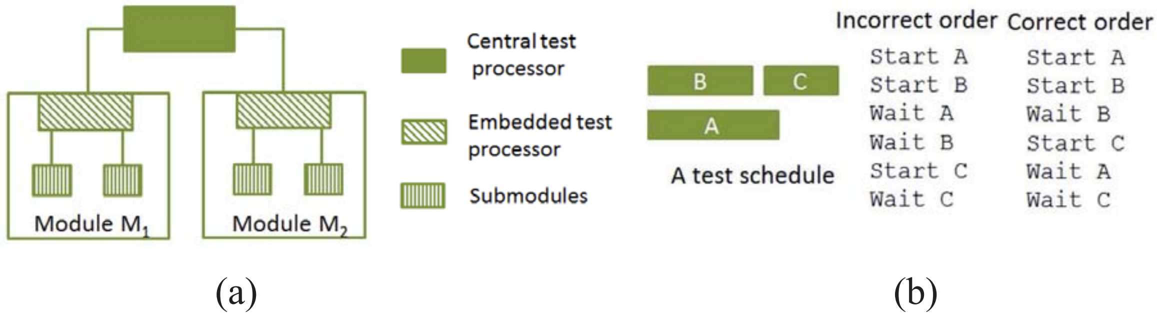


FIGURE 5.1: (a) BIST framework from [9]; (b) Examples of correct and incorrect program order for the given test schedule using BIST instructions from [9].

instruction, and by relying on a completed BIST instruction to launch the execution of other instructions that are waiting for it.

5.3 BIST Framework

The proposed BIST infrastructure is distributed, reusable, and reconfigurable, and it provides support for end-to-end testing of 3D-stacked ICs. We assume BIST-ready cores with core-specific BIST controllers connected to cores. We refer to core-specific controllers as a resource in addition to any other BIST infrastructure that the core-level controllers may use, e.g., LFSR/MISR pair. For instance, a BIST controller for pre-bond or post-bond testing of TSVs is also categorized as resource in this work. A single resource may be used for multiple tests, one at a time.

The building blocks of the proposed BIST framework consist of die-level controllers, resource-level controllers, buffer management modules, a completed-test broadcast bus (`CTBS`), and `CTBS` arbiters (see Figure 5.2). BIST control instructions are loaded through the bottom-most die. A die-level controller controls every instruction going through the corresponding die. It either dispatches an instruction to the appropriate resource-level controller or forwards it to the upper die if the instruction is to be executed on a different die. Instructions may not be executed as

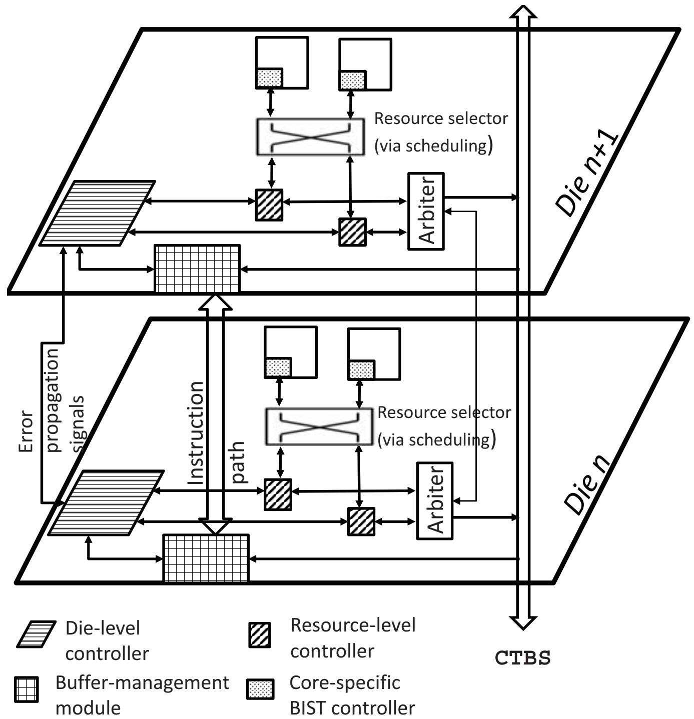


FIGURE 5.2: Overview of the BIST architecture.

soon as they reach their target dies due to resource or power constraints. A buffer management module is thus used at every die for storing BIST instructions until they become ready for execution. After a BIST instruction is completed, all instructions that are waiting for its completion are notified via the broadcast bus; access to this bus is controlled by the arbiter located on every die. The resource-level controller configures the resources and invokes each BIST run. The resource selector matches cores to resource-level controllers based on the test schedule. Next we describe each component in detail.

5.3.1 BIST instruction format

The dies communicate with each other using BIST instructions that are encoded using the format shown in Fig. 5.3. The most significant bit is the valid instruction bit, which if 1, indicates that the instruction is valid; otherwise it is invalid. An

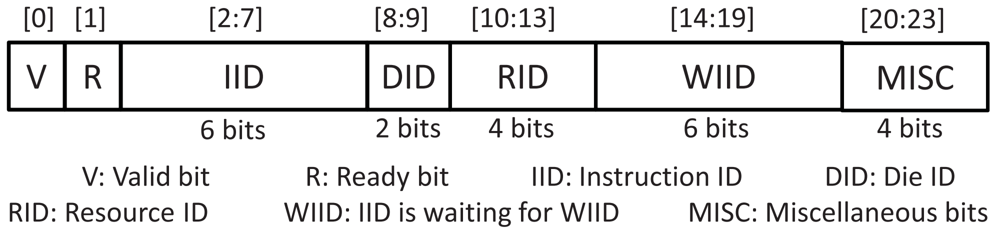


FIGURE 5.3: An example of a 24-bit BIST control instruction format for a four-die stack with 16 BIST resources on each die.

instruction is always valid when it is loaded, and it becomes invalid later during its lifetime, as described in Section 5.3.B. The next bit indicates whether the instruction is in `READY` or `WAIT` state. An instruction that is waiting for another instruction to finish is categorized as a `WAIT` instruction, otherwise it is a `READY` instruction—ready to be executed. An instruction can be issued in both `READY` and `WAIT` mode. A `WAIT` instruction is converted to a `READY` instruction through a broadcast message that we explain in Section 5.3.C.

The next set of bits constitutes the instruction identifier (`IID`) and the Die ID (`DID`), where the latter is an index to a die in the stack. A die-level controller determines whether it should consume the incoming instruction or forward it to the controller up the stack. When an instruction is forwarded, the `DID` field is decremented by one; therefore, the `DID` field of the instruction format is dynamic, and keeps decreasing for an instruction as it moves up the stack. The instruction is absorbed only if `DID` becomes zero. This scheme decouples the die-level controllers from the knowledge of the positions of their respective dies within the stack; it is not necessary for a die to know its position in the stack, and test circuitry for dies can be designed independently. The next set of bits identifies the resource that will be used for executing BIST, which we call resource identifier or `RID`. If an instruction is in `READY` state, the field `WIID` is used for identifying the instruction for which the current instruction is waiting.

The last set of bits refers to the miscellaneous bits (`MISC`), which can be used for test initialization needed for running BIST. Since we are interested in supporting a system, where a single resource can be used for testing multiple cores, the `MISC` bits can be used for configuring the BIST controller to access the targeted core. The width of the multi-bit fields can be determined by the number of dies in the full stack, the maximum number of resource-level controllers (or resources) on any die, and by the maximum number of tests to be launched in a single BIST session. For example, in Fig. 5.3, for a four-die stack with 16 BIST resources on each die, we need two bits for encoding `DID`, four bits for `RID`, and six bits each for `IID` and `WIID` for scheduling 64 BIST-based tests in a single session. In the next subsection, we discuss the life-cycle management of a BIST instruction loaded in the stack.

5.3.2 Instruction management

Every BIST control instruction is fed through the bottom-most die. During testing, multiple cores can be tested simultaneously, which implies that multiple instructions have to be placed successively. Some of these instructions may not be processed immediately because of temporary unavailability of a BIST resource or due to power constraints. Such instructions have to be stored temporarily in the stack. In order to alleviate this problem, we propose a buffer-management module per die that consists of buffers for storing the incoming and outgoing instructions, and the related buffer controller. This approach, in conjunction with other features of our framework (discussed in subsequent subsections), makes it possible to process instructions non-sequentially and opens potential new directions in dynamic BIST scheduling, which are however left for future work.

As shown in Fig. 5.4(a), the buffer management module of a die consists of an input buffer and an output buffer. All incoming BIST instructions are stored in the input buffer and those instructions that the die-level controller cannot absorb (because the instructions are meant for a different die) are migrated to the output buffer. An instruction in the output buffer of *Die n* is transferred to the input buffer of *Die n*+1, only if the buffer management module of *Die n*+1 is in a state to accept instructions, i.e., if the input buffer of *Die n*+1 has free slots; otherwise the instruction has to be temporarily stored in *Die n*. As shown in Fig. 5.4(b), there can be multiple slots in a buffer. Each slot is occupied by an instruction. Whether a slot is occupied by an instruction is determined by the valid bit of the instruction. Every buffer has a free-slot pointer that points to the slot where the next instruction has to be filled. Therefore, the free-slot pointer always points to a slot containing an invalid instruction.

After an instruction occupies a free slot, the free-slot pointer jumps to another free slot in the buffer. The buffer controller tracks the availability of free slots and instructions are accepted only if a free slot is available. A slot becomes free as soon as an instruction has either been consumed by the die-level controller or it has been moved up the stack. At every cycle, the die-level controller checks the instruction pointed to by the instruction pointer and determines whether the instruction can be processed, and the instruction pointer is then advanced to the next slot. After an instruction is accepted for processing, it is invalidated and the corresponding slot in the buffer is made free. Since the instruction pointer moves at every cycle, a `READY` instruction is not halted unnecessarily. Moreover, the instruction pointer can be updated using various methods, thereby providing many opportunities for dynamic BIST scheduling. Since dynamic BIST scheduling is beyond the scope of this work, we simply move the instruction pointer to the next slot in a round-robin fashion.

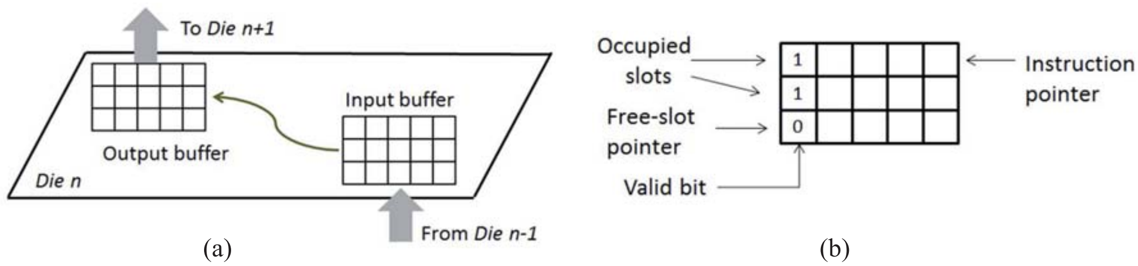


FIGURE 5.4: (a) The instruction buffer management module; (b) instruction buffer with multiple slots for storing instructions.

5.3.3 Relationship to test scheduling

We listed the limitations of the design proposed in [9] in Section 5.2, and highlighted how it does not meet the requirements for 3D-stacked ICs. First, it is not possible to translate scheduling-dependency relationship between any pair of cores into BIST control instructions. Second, the order of BIST instructions in the test program is crucial, thereby making it prone to errors and leading to potential increase in test time and power-constraint violation. In this work, we address these shortcomings by adding a *Completed Test Broadcast Bus* (`CTBS`) that runs across all dies with registers at the die boundaries. Once BIST for a core is completed, the corresponding resource-level controller sends a broadcast message across all the dies announcing the completion of the related instruction. The broadcast message consists of the instruction ID (`IID`) of the instruction that was completed and a valid bit to indicate the validity of the broadcast message.

Fig. 5.5 gives a logical view of this framework. The buffer management module of each die keeps polling the `CTBS` and changes the status of every instruction waiting for the completed instruction to the `READY` state. A `CTBS` arbiter grants permission to use the `CTBS` upon receiving a *CLAIM* request from a resource-level controller. The arbiter relinquishes the control of `CTBS` only when it is not being used by any

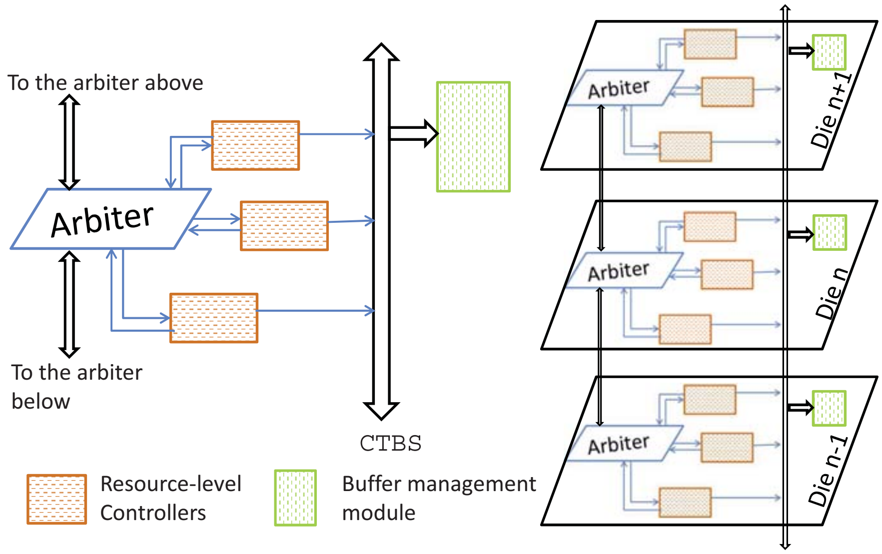


FIGURE 5.5: Broadcast mechanism for the completed instructions.

other controller. We have adopted a first-come first-serve approach for arbitrating among the claim requests from different controllers, and the arbiter may prioritize among multiple resource-level controllers that assert their respective *CLAIM* signals simultaneously. The arbiter relinquishes the control of the bus through *GRANT* lines, after which the `IID` of the completed instruction is put on the `CTBS` by the corresponding controller. An arbiter communicates with the arbiters present on adjacent dies for propagating the `IID` through the stack.

Fig. 5.6 shows the interconnections between arbiters in adjacent dies, highlighting the control signals that are used for communication. Since the `IID` has to be propagated across all dies, when an arbiter receives a *CLAIM* request from a resource-level controller, it sends claim requests to the arbiters of the adjacent dies through the *CLAIM_Aout* port to the die above it and through the *CLAIM_Bout* port to the die below it. Fig. 5.7 shows an example of how the `IID` of a completed instruction on *Die n* is latched to `CTBS` of the upper die (*Die n*+1). The arbiter on *Die n*+1 receives the claim request through its port *CLAIM_Bin*. If the arbiter is idle, which means that the corresponding `CTBS` is free to use, the claim request is granted and

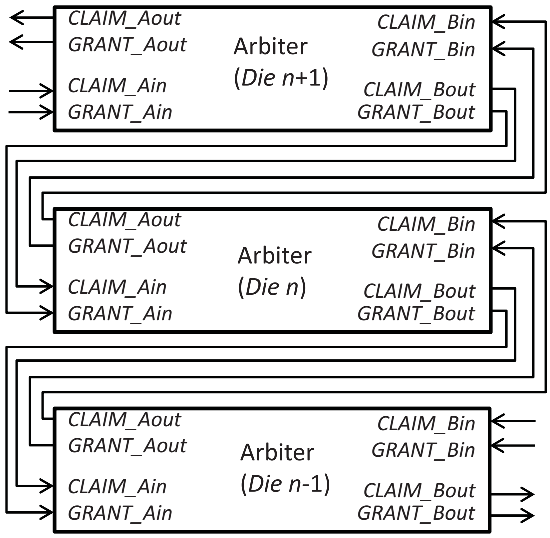


FIGURE 5.6: Arbiters in adjacent dies and their interconnections.

`IID` is latched to the `CTBS` in the next cycle. The arbiter on *Die n+1* also sends an acknowledgement through its *GRANT_Bin* port to *Die n*. On receiving an acknowledgement (through *GRANT_Ain* port) from the arbiter above, the arbiter on *Die n* removes its claim request. Note that when an arbiter receives a claim request from one direction, it sends a claim request in the other direction for propagating the `IID`. For the above example, the arbiter on *Die n+1*, sends a claim request to *Die n+2* for propagating the `IID`, and so on. During pre-bond testing, *CLAIM_Ain* and *CLAIM_Bin* are held low, and *GRANT_Ain* and *GRANT_Bin* are held high. These lines can be controlled through a single I/O pad during pre-bond test.

5.3.4 Die-level controller

Each die has a die-level controller that interacts with the buffer management module of the same die to retrieve those instructions that are in the `READY` state. If the `DID` field is zero, the controller dispatches it to the resource-level controller identified by the field `RID`. The controller also interacts with die-level controllers of adjacent dies

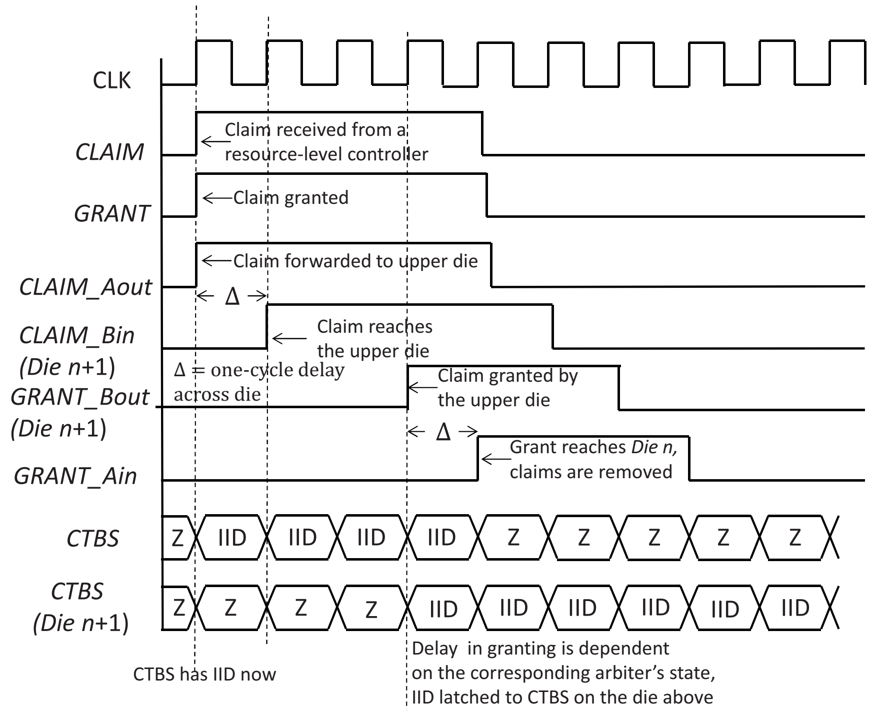
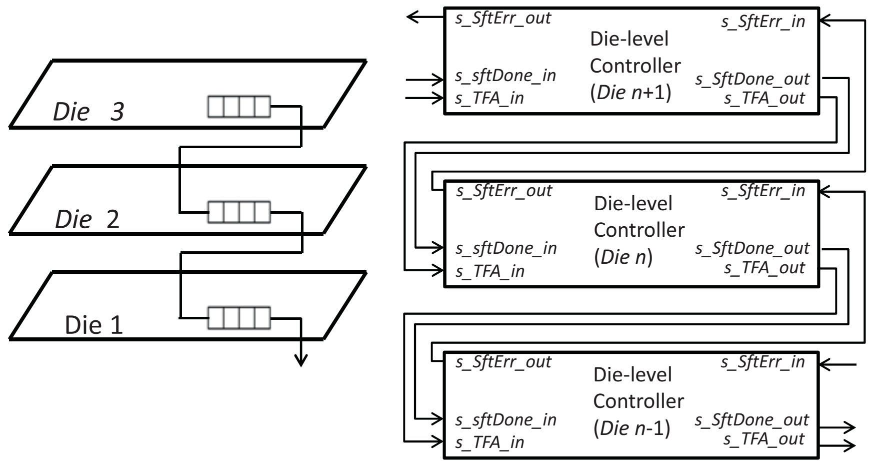


FIGURE 5.7: An example of broadcasting `IID` from *Die n* to the `CTBS` of *Die n+1*. 'Z' refers to high-impedance state.

for synchronized error propagation. The response collected after running BIST on every die is sent back to the ATE for debugging. In this work, we only use a pass/fail bit per core to indicate whether the core has passed the test. On every die, there is an error register that stores these pass/fail bits. As shown in Fig. 5.8(a), the error registers on multiple die are cascaded to form a single large shift register when the response has to be collected, and the error bits are shifted out in a serial fashion. The response is collected only when each die has finished executing all instructions assigned to it. The die-level controllers interact with each other for synchronizing the shifting of cascaded error registers.

Fig. 5.8(b) shows the three input ports and the three output ports of the controller that are used for communicating with die-level controllers in adjacent dies.



(a) (b)

FIGURE 5.8: Figures showing (a) cascaded error registers, and (b) interconnections between die-level controllers in adjacent dies.

The three input ports and their connectivity to the corresponding output ports are as follows:

- *s_TFA_in* indicates that all tests in dies above the current die are finished. It is connected to the output port *s_TFA_out* of the upper die.

- *s_SftErr_in* port receives a request sent from the die-level controller of the lower die, which enables the controller to start shifting the error register. This port is connected to the signal *s_SftErr_out* of the lower die.

- *s_SftDone_in* port receives a signal indicating that every bit of the error registers from all dies above the current die have been shifted out. This port is connected to the *s_SftDone_out* port of the upper die.

We next explain the finite-state machine (FSM), shown in Fig. 5.9, for the die-level controller. The controller starts in the `Idle` state and remains there until a valid BIST instruction is loaded. The signal *s_bist_init* is asserted by a valid and

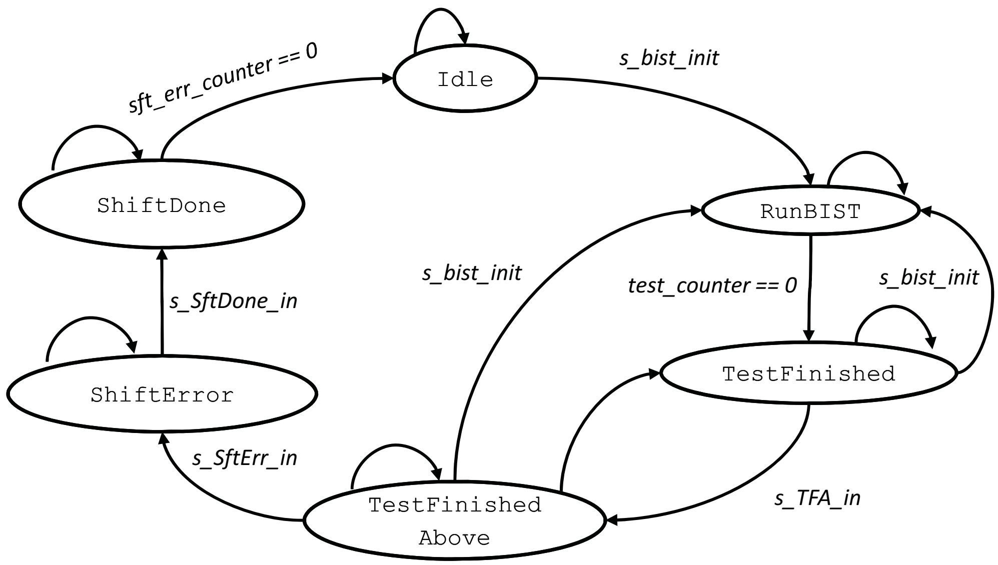


Figure 5.9: State machine for a die-level controller.

ready BIST instruction and the controller enters the `RunBIST` state. The controller dispatches the command to the corresponding resource-level controller with `RID` and `MISC` bits of the instruction. The valid bit of the instruction is invalidated in the instruction buffer, marking the corresponding slot free. A counter is incremented whenever BIST for a core is initiated, and it is decremented when the controller receives the *bist_cmpl* signal from a resource-level controller indicating the release of the corresponding resource after completion of BIST. When the counter value becomes zero, the controller enters the state `TestFinished`. From this state, the controller can either enter `RunBIST` again on seeing a valid instruction or it can enter the state `TestFinishedAbove`, which indicates that all tests in dies above the present die are finished, and that the controller is ready to shift its error register. The transition from `TestFinished` to `TestFinishedAbove` is triggered by the signal *s_TFA_in* from the die above. From this state, the state machine can transition to

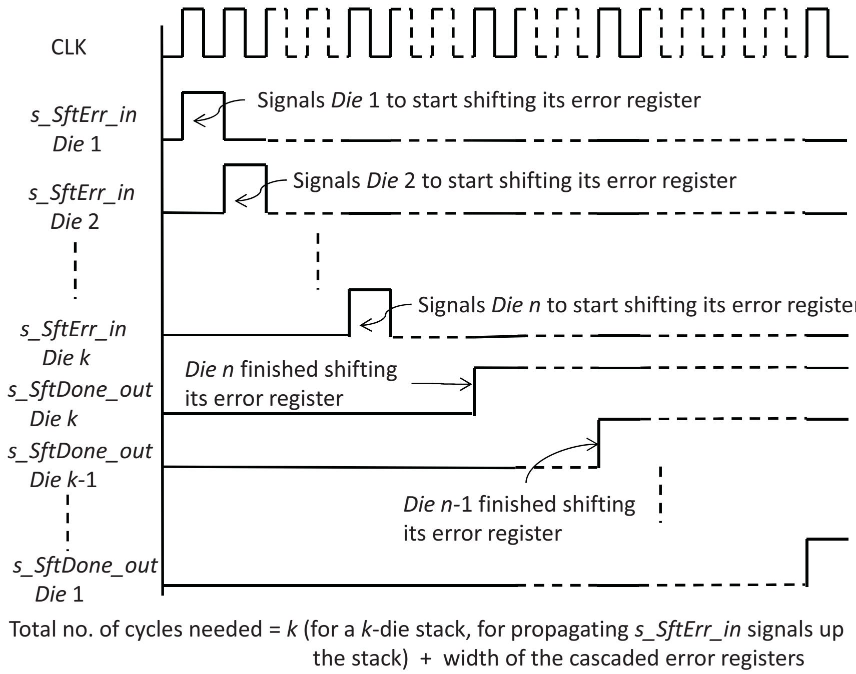


FIGURE 5.10: Timing diagram for shifting out the cascaded error registers. Before the shift sequence begins, all die-level controllers are in `TestFinishedAbove` state, which indicates that all tests have finished execution.

`RunBIST` and `TestFinished` on seeing a valid instruction to the current die and the die above, respectively. Furthermore, in `TestFinishedAbove`, the die-level controller waits for the signal *s_SftErr_in* from the die below to start shifting the error register.

Note that the `TestFinishedAbove` state is defined recursively, and it encapsulates the information that all tests above a given die have finished execution. Before beginning to shift the error register, the controller enters the `ShiftError` state on receiving a signal through its *s_SftErr_in* port. In the next cycle, the controller on the upper die is signalled to shift its error register. Fig. 5.10 shows a timing diagram for a sequence of operations leading to shifting out of the cascaded error registers. The bottommost die (*Die* 1) is triggered to start shifting its error register, after which *Die* 2 is triggered. This sequence is carried out iteratively until the request

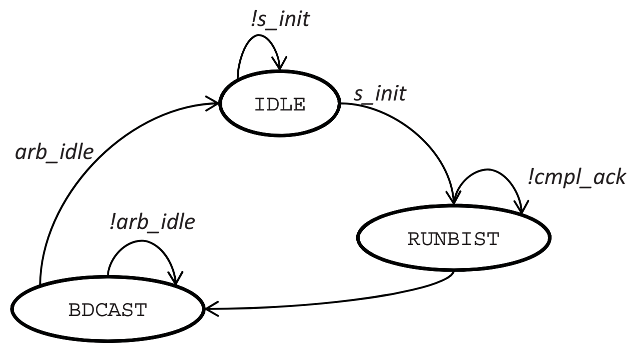


FIGURE 5.11: State machine for a resource-level controller.

reaches the topmost die (*Die k*), and the error register on all dies are cascaded into one large shift register. When *Die k* is done with shifting its shift register, it sets the signal *s_SftDone_out* to 1, and the controller in the lower die enters the `ShiftDone` state. This state initializes a counter with the width of the error register on the corresponding die and tracks the number of bits that remain to be shifted. After all bits are shifted, the controller transitions to the `Idle` state, and asserts *s_SftDone_out*, which in turn, triggers a similar sequence of events iteratively until the controller in the bottommost die has shifted out the contents of its error register. The controller of all dies enter their `Idle` state in this fashion.

During pre-bond testing, the input control signals to the controller at the die boundary are set to high, and a die-level controller can work in isolation to other controllers as well. During partial-stack (mid-bond) testing, the unconnected input signals to the controllers of the topmost and the bottommost dies can be set to 1. Note that with this design and the proposed scheme for communication, the controllers are reusable at multiple test insertions.

5.3.5 Resource-level controller and core-level controller

At the next level of hierarchy are the resource-level controllers. A resource-level controller is dedicated to a BIST resource and is responsible for initializing and managing the resource. It is present on the same die as the corresponding resource, but the resource can be used for testing cores located in a different die. Fig. 5.11 describes a typical resource-level controller. Initially, the controller is in `IDLE` state, and moves to the `RUNBIST` state after receiving the signal *s_init*. In this state, the resource takes control, and the resource-level controller simply waits for the resource to complete the BIST. The resource (core-level controller if the BIST is testing a core) first sets up the path to the intended core, initializes the LFSR/MISR, and finally runs BIST. After BIST is completed, the resource-level controller gets a completion acknowledgement *cmpl_ack* from the resource, and it moves to the `BDCAST` state where the `IID` of the completed instruction is broadcast on the `CTBS`. The FSM makes a transition to the `IDLE` state if the arbiter on the same die is idle, identified by the signal *arb_idle*. The implementation of the core-level controller is specific to the core and the types of tests that are be applied to it.

The distributed BIST infrastructure supports testing of individual dies in isolation, and facilitates partial or full stack testing. The design of every component of the infrastructure is uniform across dies, thus making it reusable. The synergy between different units of the infrastructure enables concurrent testing of multiple cores on different dies and eases the selective application of tests at various test insertions.

5.4 Test Scheduling under Power Constraints

The power consumption of SOCs during test can be up to twice as much as in normal mode [4]. Power/thermal concerns are even more important for 3D-stacked ICs. Power-constrained test scheduling is therefore essential to prevent overheating and subsequent damage; a test scheduling algorithm must limit the degree of concurrency after taking into account the power budget. In the manufacturing of a 3D-stacked IC, tests can be applied at multiple stages—dies can be tested prior to bonding or after bonding, and tests can also be applied to partial stacks. Each stage may have different power constraints. Using pre-bond test schedules for individual dies in a post-bond test may violate power constraints; therefore, it is desirable to create different schedules at every test stage. Moreover, limited BIST resources can be shared across dies, which is another motivation for creating different schedules for multiple test insertions.

Several approaches have been proposed in the literature to address the test-time minimization problem under power constraints for two-dimensional SOCs [4, 114]. Test-scheduling for BIST-ready cores in 3D-stacked IC is presented in [113], but it ignores resource constraints. In this section, we formulate the problem of test-time minimization for 3D-stacked ICs under resource and power constraints, and provide two solutions based on rectangle packing.

5.4.1 Problem formulation

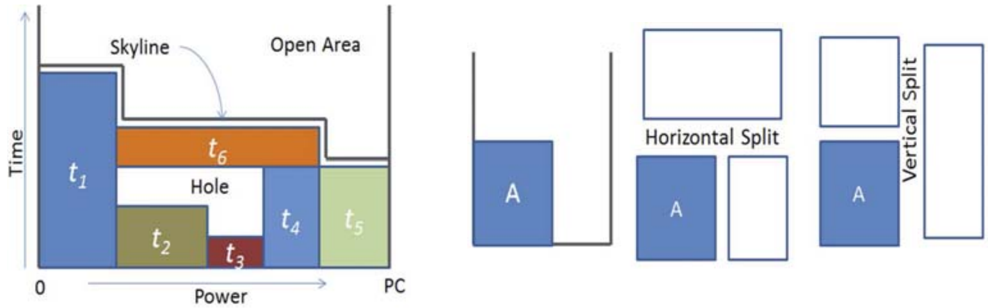
Resource constraints were considered in [114] using the notion of a test-compatibility graph (TCG), where each node represents a test and an arc between two nodes indicates that the corresponding tests are compatible. Compatible tests can be scheduled at the same time. Certain tests may not be structurally compatible, e.g., applying EXTEST and INTEST to a core concurrently is structurally not possible. Although TCG can capture structural compatibility between tests, but we find TCG to be an inflexible and insufficient structure to capture the dynamic nature of resource constraints for our problem for two reasons:

- First, if R BIST resources are shared between T tests, where $T > R$, only T out of R tests can be scheduled concurrently. The remaining tests can only be scheduled on availability of resources. This information cannot be captured using edges of a TCG.

- Second, if two tests are incompatible due to unavailability of resources during a testing stage in the 3D integration flow, they may become compatible after stacking of more dies containing additional resources.

Therefore, in addition to a structural compatibility graph, we use an RxT resource-compatibility matrix of binary values for R resources and T tests, with a 1 in the i^{th} row and j^{th} column in the matrix indicating that the i^{th} resource may be used by the j^{th} test. During scheduling, a test can only use a free resource; if no resource is free, the test is delayed until a resource is freed. Given a structural compatibility graph, a resource-compatibility matrix, test time and power consumption values of tests, the objective of scheduling is to minimize test time without exceeding the power limit PC. We use the same formulation for both pre-bond and post-bond test insertions—the only difference being that the dimensions of the compatibility matrix increases with the addition of dies in the stack (post-bond tests). A similar problem can be crafted for finding the minimum number of resources needed to keep test time below a maximum threshold. In Section 5.5, we show the effect of varying the number of resources on test time.

A known NP-hard problem—2D bin packing, where a given set of rectangles must be packed in a bin with fixed width to minimize the bin height—can be reduced to the test-scheduling problem in polynomial time. Each test can be represented by a rectangle of height and width equal to the length and power consumption of the test, respectively. We create one test each for a rectangle in the given instance of the bin packing and assign the power budget (PC) to be equal to the width of the given bin. In the newly constructed instance of the test-scheduling problem, all tests are compatible to each other. Note that the given set of rectangles can be packed to a bin of height M if and only if we can solve test-scheduling problem to achieve test time M. Therefore, the test-scheduling problem is NP-hard and we adapt two known heuristics to solve this problem.



(a) (b)

FIGURE 5.12: (a) A hole and a skyline. There are 6 tests in this example—t_1 through t_6. (b) Free rectangles after the placement of a rectangle in the initial bin using the guillotine-split method. Two configurations are possible based on orientation of the split.

5.4.2 Skyline algorithm

For an optimal packing of bins, a list of free areas or holes has to be maintained (see Figure 5.12(a)). The related problems involved in fitting a rectangle in holes are: (a) what data structure is used to represent holes; (b) how a new rectangle is accommodated inside a hole; and (c) how the data structure for a hole is updated once a rectangle is placed inside the hole [115]. These problems require a tedious manipulation of the available free space as the rectangles are packed. An alternative is to only consider the *skyline* created by the rectangles, which is defined as the outline of packed rectangles against the open area.

The data structure for storing a skyline can be a doubly linked-list and each element of the linked-list stores the x-coordinates of both end-points and y-coordinate of a *step* in the skyline; a skyline can be viewed as a series of steps going up and down the contour of packed rectangles. The procedure begins by sorting the list of rectangles in the order of their height (largest to smallest), i.e., based on test times of the corresponding cores. Other variants of the algorithm can be designed by sorting the rectangles based on their width, or by sum of width and height, or by area. The next rectangle to be packed is picked from the list of rectangles in the sorted order

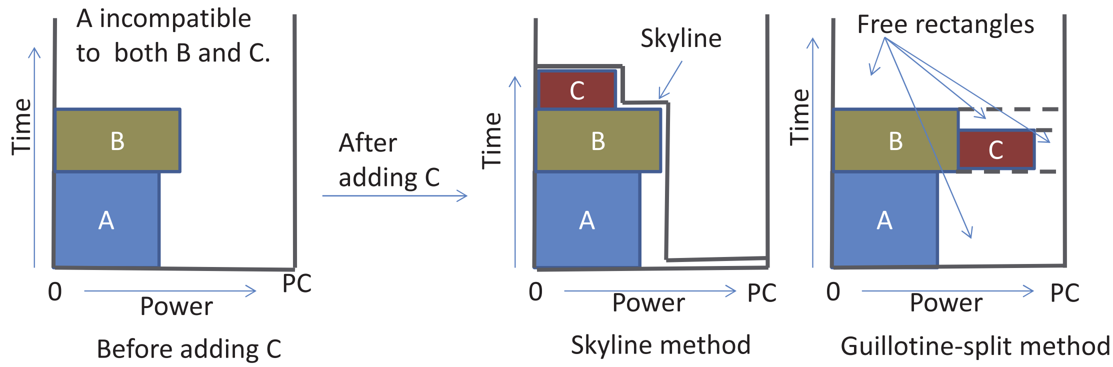


FIGURE 5.13: Illustration of the skyline and the guillotine-split methods under resource constraints.

and it is tried to fit on the skyline. The lowest step in the skyline is chosen first, and it is determined whether the rectangle can be packed on this step before considering higher steps. The lowest possible step that can fit the rectangle is selected because our objective is to minimize test time. We adopt the bottom-left (referred to as BL in the literature on bin packing) variant of the algorithm here where the bin is packed as low and as left as possible. Placing a rectangle updates the skyline data structure, but some free space is wasted during the update.

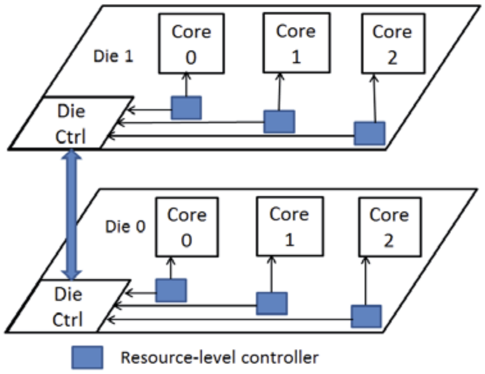
5.4.3 Guillotine-split method

We implement another approach for packing rectangles based on guillotine-split placement [116]. The algorithm starts with the sorted list of rectangles and the empty bin, and the first rectangle is placed in the bottom-left corner of the bin. The remaining L-shaped free space can be split into two disjoint free rectangles. There are two possible choices of carrying out the split, as shown in Figure 5.12(b). The rest of the algorithm is concerned with packing the remaining rectangles (tests) in the free rectangles (unfilled regions in the bin). Placing a rectangle in a free rectangle and splitting the used rectangle into two disjoint free rectangles is done iteratively until all rectangles are included. We choose horizontal cuts because the list of tests is sorted in descending order of their test times, and the next rectangle to be packed always has shorter height than the rectangles that are already packed. The lowest-lying free rectangle that can accommodate the next rectangle is selected for packing. While the skyline algorithm may lead to a session-less test schedule, the guillotine-split approach always results in session-based test schedules. Several techniques are known to manage free rectangles for effectively packing rectangles [116].

In the skyline-based approach, a rectangle is always placed against the skyline; it cannot be levitated to avoid resource contention. For example, in Figure 5.13, rectangle B is placed above rectangle A because the two rectangles are not test-compatible. Rectangle C, which is also not compatible to A, gets placed above B in the skyline-based approach, but C can be placed with B in tandem in the split-based approach.

As resource constraints are also considered, we analyze candidate positions for checking resource conflicts. As tests are scheduled, resources are updated with time slots during which they are active. This information is utilized to analyze a candidate step (free rectangle) for the skyline-based (guillotine-split) method for identifying resource conflicts. For every test, O(T) steps (or free rectangles) are analyzed. Querying a resource associated with the test for analyzing resource conflicts takes O(T) time. Since a test can be compatible with multiple resources, analyzing a candidate position for every compatible resource takes O(TR) time. Therefore, both the above algorithms with resource constraints can be implemented to run in O(T^3R) time. For most practical purposes, each test has a small number of compatible BIST resources, hence the run-time complexity of the algorithm is O(T^3).

In Section 5.5, we show that the guillotine-split method outperforms the skyline-based method under tighter resource constraints.



(a)

	Core 0	Core 1	Core 2
Design 1	*b*20	*b*21	*b*22
Design 2	*cf_rca_*16	*des_perf*	*b*20
Design 3	*des_perf*	*cf_rca_*16	*cf_fft_*256_8

(b)

FIGURE 5.14: Figures showing (a) layout for our experimental 3D-stacked IC designs, and (b) cores used in each design.

5.5 Results

We use three designs, namely *des_perf*, *cf_rca_*16, and *cf_fft_*256_8, from the Open-Core benchmark suite, and three circuits (*b*20, *b*21, and *b*22) from ITC'99 benchmarks. Structural models for these benchmarks are available in Verilog. Each design is partitioned into two dies. For the Open-Core designs, the partitioning was done for performance optimization. For the ITC'99 benchmark circuits, hMetis [23]—a hypergraph-patitioning tool—was used. We used Cadence RTL Compiler and the NanGate 45nm Open-Cell library for synthesizing gate-level netlists of the BIST controllers (from Verilog descriptions) and the above benchmark designs. As shown in Figure 5.14(a), we created synthetic stacked ICs using two dies with three cores per die. The details of the three designs that we used for our experiments are provided in Figure 5.14(b). Table 5.1 shows the number of patterns required for achieving 98% stuck-at fault coverage for each design.

We first compare the area of synthesized netlists, with and without BIST. Without loss of generality, the flip-flops in each design were stitched in a single scan chain,

Table 5.1: Pattern count for achieving 98% fault coverage for various designs

	$b20$	$b21$	$b22$	$des\_$ $perf$	$cf\_$ rca_16	cf_fft $\_256\_8$
Die 1	3968	12248	4544	1280	20424	8192
Die 2	14784	18336	15168	960	24368	6656

Table 5.2: Area of the synthesized netlists (in square microns) with and without BIST

Designs	Without BIST	With BIST	Area overhead (%)
Design 1	94176	98991	5.11
Design 2	413142	418155	1.21
Design 3	1324574	1329562	0.37

the scan chains were driven by a 50-bit LFSR, and responses were compacted using a 50-bit LFSR. Table 5.2 shows that the area overhead due to BIST is negligible. The percentage area overhead is least for Design 3 because it is the largest of all designs considered in our experiments; we expect the overhead to be even lower for larger and real designs. Since the design of controllers remain unchanged and the instruction width is logarithmic in the number of dies, resources and the number of tests, the proposed framework is scalable to larger designs.

Next we evaluate the impact on timing for the designs due to BIST insertion. Cadence RTL Compiler was used for finding the slack time of the most critical path. No impact on critical paths was observed. The slack times of ITC'99 benchmark circuits are negative for clock frequencies higher than 100 MHz, and for the Open-Core designs, the slack times are negative only for frequencies higher than 500 MHz. These clock frequencies are therefore upper limits on circuit performance. After incorporating BIST, the slack times were found to be equal to those from the original designs for the maximum allowable clock frequencies.

We also assess the impact of incorporating BIST on power consumption. Cadence RTL Compiler was used for measuring power consumption for the two cases—with

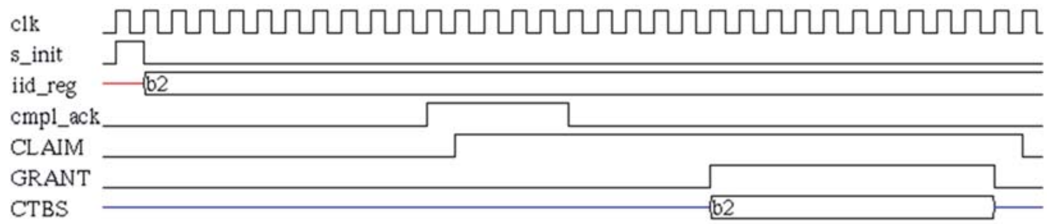


FIGURE 5.15: Timing simulation for a resource-level controller.

BIST and without BIST. Due to unavailability of functional test benches for the benchmark circuits, instead of using pattern-based simulation, we use two parameters, namely, `lp_default_toggle_rate` and `lp_default_probability`, used by Cadence compiler for estimating power consumption. These parameters indicate the toggle rate (per nanosecond) and signal probability of the input signals, respectively. The parameter `lp_default_toggle_rate` was varied in the range of [0.02,0.08] and the parameter `lp_default_probability` was varied in the range of [0.1,0.5]. After adding BIST to the designs, on average, the power consumption increased by only 1.33%, 0.18%, and 0.12% for Design 1, Design 2, and Design 3, respectively.

The BIST framework was described at a behavioral level in Verilog. It was then simulated using ModelSim. Figure 5.15 shows timing simulation results for a resource-level controller. The controller receives *s_init* to execute the BIST instruction, and the corresponding `IID` is latched to *iid_reg* in the next clock cycle. The BIST is completed when the controller receives the signal *cmpl_ack*, and *CLAIM* request is generated to broadcast the `IID` on the `CTBS`. After receiving a `GRANT`, the `IID` is routed on the `CTBS`.

Figure 5.16 shows the results obtained from simulating the instruction buffer management module. Instructions are fed to the bottom die, i.e., *Die* 1, through the *data_in* port every cycle. Instructions are shown in hexadecimal format. Only the fields `IID, DID`, and `RID` fields are shown per instruction for simplicity with four bits assigned to each field, e.g., the instruction 101 has `IID` as 1, `DID` as 0, and `RID`

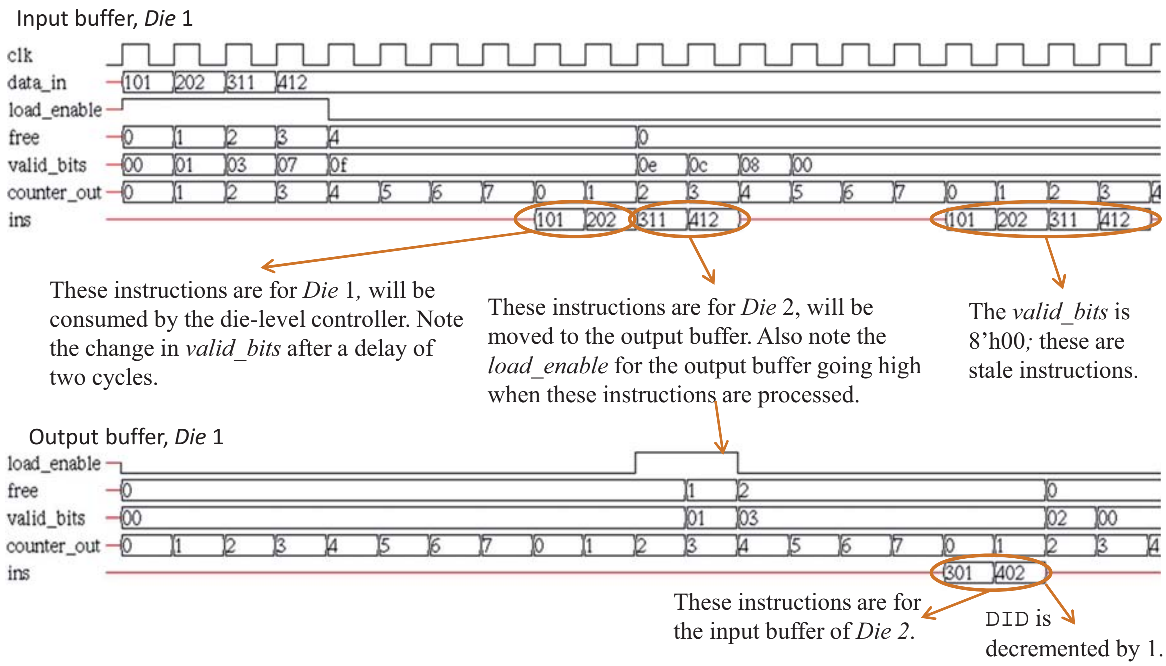


FIGURE 5.16: Timing simulation for the buffer management module.

as 1.. All instructions are assumed to be ready. These instruction are loaded in the input buffer of *Die* 1 when the signal *load_enable* is at a high value. An instruction is loaded in the slot referenced by the *free* pointer. Initially, all slots are free, i.e., no slot has a valid instruction; therefore, *valid_bits* is 8'h00 (this is a simulation with eight slots per buffer). As an instruction is loaded, the *free* pointer starts pointing to a next free slot, and *valid_bits* is updated. The port *ins* is the instruction pointed by the instruction pointer *counter_out* (a simple three-bit counter for referencing the eight slots in a round-robin fashion). Note that for the first eight cycles, no instruction is observed from the *ins* port because the *counter_out* is not pointing to a slot with valid instruction. After an instruction that is meant for forwarding up the stack is found, the *load_enable* port of the output buffer is set to high and the `IID` field is decremented by one, and the instruction is loaded in the output buffer, from where it is moved to the input buffer of *Die* 2. The forwarding is repeated until

Table 5.3: A running example of tests with test length (TL) and power consumption (PW) values from [4]

Die1 (A.U.)			Die2 (A.U.)		
Test	TL	PW	Test	TL	PW
t_1	12	3	t_2	11	5
t_3	9	9	t_4	8	12
t_5	8	4	t_6	8	2
t_7	8	1	t_8	6	7
t_9	6	6	t_{10}	5	7
t_{11}	5	5	t_{12}	4	11
t_{13}	4	2	t_{14}	3	3
t_{15}	3	1	t_{16}	2	5
t_{17}	2	4	t_{18}	1	12
t_{19}	1	8	t_{20}	1	7

Table 5.4: Test length (in s) and power consumption (in mW) for Design 3 at 500 MHz clock frequency

Test	TL	PW
Die 1, Core 0	0.55	59.4
Die 1, Core 1	0.45	549
Die 1, Core 2	0.002	8.7
Die 2, Core 0	0.57	69.5
Die 2, Core 1	0.51	25.1
Die 2, Core 2	0.003	9.9

Table 5.5: Test times for pre-bond and post-bond tests using the two scheduling algorithms (see Table 5.3 and table 5.4 for units.)

Example	PC	Skyline			Guillotine split		
		Die 1	*Die* 2	Stack	*Die* 1	*Die* 2	Stack
[4]	15	21	33	53	20	29	44
	25	12	18	27	17	17	28
	35	12	11	15	12	11	15
Design 3	560	1	0.57	1.02	1	0.57	1.02
	620	0.55	0.57	1	0.55	0.57	1.02
	680	0.55	0.57	0.97	0.55	0.57	1.02

the instruction reaches the desired die.

We next present test-scheduling results. We use the example from [4] and Design 3 to show how test time varies with the number of available resources and power constraints. Table 5.3 lists 20 tests with test length (TL) and power consumption (PW) values in arbitrary units (A.U.) from [4]. The odd-numbered tests are marked as tests that target *Die* 1 and the even-numbered tests are for *Die* 2. Table 5.4 shows the TL and PW values for Design 3. If every test has its own dedicated BIST

Table 5.6: Test times (A.U.) for the example from [4] under resource constraints.

PC	$R_G = 1$						$R_G = 2$						$R_G = 3$					
	Skyline			GS			Skyline			GS			Skyline			GS		
	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{S}$	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{S}$	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{S}$	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{S}$	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{S}$	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{S}$
15	36	33	62	26	31	52	31	33	54	21	29	44	31	33	54	21	29	44
25	29	21	57	25	21	41	21	18	36	21	17	32	21	18	34	21	17	32
35	29	21	57	25	21	41	21	12	29	21	15	25	21	11	29	21	15	25

R_G: #Resources per test group GS: Guillotine split
$\mathbb{D}_1$: *Die* 1 $\mathbb{D}_2$: *Die* 2 $\mathbb{S}$: Stack

Table 5.7: Test times (in s) for testing the stack of Design 3 under resource constraints.

PC	Skyline		Guillotine split	
	R_G=2	R_G=3	R_G=2	R_G=3
580	1.523	1.023	1.09	1.02
620	1.523	1	1.09	1.02
680	1.523	0.97	1.09	1.02

resource, i.e., tests are compatible with each other, minimum test times achieved using the two algorithms are shown in Table 5.5 with different values of the power budget (PC) in mW. The skyline-based approach gives better results with relaxed power constraints.

Next we add resource constraints to the example from [4]. We assume that tests t_i and t_{i+2} are structurally incompatible, where $1 \leq i \leq 18$. Furthermore, we categorize the test set into three test groups—t_{3i}, t_{3i+1}, and t_{3i+2}, where $1 \leq i \leq 6$—such that, tests belonging to the same test group can share BIST resources. We vary the number of resources available in each group (R_G) and report the test times in Table 5.6. With resource constraints, the guillotine-split algorithm consistently gives better results than the skyline-based approach. When $R_G = 1$, i.e., when there is a single copy of resource per test group, we see that relaxing the power constraint does not reduce test time, which indicates that resource unavailability is the bottleneck.

On adding more resources per test group, we see benefits of relaxing the power constraint. We repeated the experiment on Design 3 at the stack level with all cores kept in the same test group. With stricter constraints, the split-based method gave better results than the skyline approach.

5.6 Conclusion

We have presented an end-to-end framework for enabling selective BIST-based testing at multiple test insertions during the integration of 3D-stacked ICs. The proposed framework is distributed, reusable and reconfigurable. The components of this framework interact with each other through handshaking protocols. The same test structures are used for pre-bond and post-bond tests, and the controllers can be easily programmed to apply tests selectively at different test insertions. Static schedules created under resource and power constraints can be encoded using a set of BIST control instructions that are loaded in the stack for running BIST without external intervention. We have also proposed a BIST test-scheduling problem under power and resource constraints for 3D-stacked ICs and described two solutions based on rectangle bin-packing algorithms. The BIST design has been validated through detailed simulations, and results on benchmark circuits show that the proposed BIST framework has negligible impact on area, timing, and power consumption.

6

Dissertation Summary and Future Directions

Continual advances in semiconductor technology has driven the semiconductor industry to pursue higher levels of integration. Scalability of interconnects is a major barrier in such large-scale integration, especially for smaller technology nodes. Advanced integration platforms, such as NOC and TSV-based 3D are promising alternatives to traditional bus-based interconnects. Containing the escalating test cost remains a major concern in the era of manycore SOCs. In this dissertation, we have targeted some of the challenges related to SOC testing faced by industry in adopting new integration platforms and proposed solutions to them.

6.1 Dissertation Summary

This dissertation has significantly advanced the state-of-the-art on test scheduling algorithms for NOC-based SOCs by exploiting regularity in structure of NOCs. We have comprehensively answered the questions "what to test" and "when to test" using a cost-optimal test-flow selection algorithm. Novel DfT optimization techniques have been presented to reduce the number of wrapper cells that are needed to provide testability to the logic connected to TSVs. Advances have been made towards some of the key challenges in running BIST for 3D-stacked ICs.

Chapter 2 presented test-scheduling algorithms for minimizing SOC time for a variety of NOC topologies and routing mechanisms. Formulations based on subset-sum problem, grid partitioning, and integer linear programming were proposed. Methods based on dynamic programming have been developed for solving the test-minimization problems. Results indicate the scalability of our solutions for very large SOCs.

Chapter 3 described a generic cost model to account for the various test costs involved in 3D integration and presented a formal representation of the solution space to minimize the overall cost. An algorithm based on the A* search was presented to find an optimal flow. By systematically exploring the solution space, an exponentially large number of candidate test flows were implicitly examined to report the most cost-effective test flow.

Chapter 4 discussed the problem of minimizing the wrapper-cell count in 3D-stacked ICs using a formal approach based on graph theory. The "hardness" of the problem has been ascertained by establishing its relationship with well-known NP-hard problems, namely, clique partitioning and graph coloring. Heuristic methods from the literature on clique partitioning have been adopted and modified to incorporate timing and layout information to address the problems of increased capacitive load and delay on critical paths due to multiple flop reuse.

Chapter 5 presented an end-to-end framework for enabling selective BIST-based testing at multiple test insertions during the integration of 3D-stacked ICs. The proposed framework is distributed, reusable and reconfigurable. The chapter also formulated a test-scheduling problem that aims at minimizing test time under BIST-resource and power constraints, and used two algorithms based on bin packing for solving the problem. The BIST design has been validated through detailed simulations, and results on benchmark circuits show that the proposed BIST framework has negligible impact on area, timing, and power consumption.

6.2 Future Directions

Significant progress continues to be made towards the reduction of manufacturing test cost, but test cost remains a significant challenge in the semiconductor industry. This concern is highlighted in ITRS 2013 [117]. With the emergence of new integration technologies, new avenues for test-cost reduction are available. Since test time remains one of the top test-cost drivers [117], it is imperative to develop more effective solutions to reduce test time. For 3D-stacked devices, availability of new test insertions in the form of mid-bond tests to the partial stack and post-bond tests to the full stack adds to the complexity of test-cost analysis. This thesis research has opened up the following new directions.

- *Power-constrained test scheduling in NOC-based SOCs*

  We developed a heuristic based on power-profile manipulation for power-constrained test scheduling in NOC-based SOCs. Much work is needed on developing this further for a test methodology that uses multicast routing. Multicast routing has reported a significant reduction in test time; however, parallel testing of identical cores using the same access point exacerbates the power consumption problem. Therefore, methods are needed to minimize test time in NOCs using multicast routing under power constraints.

  Although power consumption is a real-number quantity, for the purpose of optimization, power consumption for a core can be discretized to take specific values. In the same way as we posed and tackled the pin-assignment problem within the dynamic-programming framework, it is expected that test-scheduling under power constraints can also be transformed into "power-assignment problem"—problem of optimally distributing discrete power values to each access point—such that overall test-time is minimized under power constraint. This approach may lead to better test times compared to what we obtain using the method based on power-profile manipulation.

- *Test-cost analysis for 3D-stacked ICs*

  In contrast to traditional test flows, mid-bond tests and post-bond tests are new test insertions that are needed to manufacture products at acceptable level of defective parts per million or DPPM. In our work on test-cost analysis, we have not explicitly included defect models for new manufacturing steps. New defect/fault models are needed for these new steps. Incorporating new defect models in the test-cost model is required to accurately evaluate trade-offs between different test choices.

  3D integration platform also offers a variety of design/manufacturing choices. These choices affect important parameters, such as die yield and die manufacturing cost, differently; hence, optimal test flows may differ depending on the choices made for manufacturing. An integrated cost model that incorporates design choices is therefore required. Advanced algorithms will be needed to report optimal design and test flows.

- *BIST for 3D-stacked ICs*

  The proposed method of distributed, reusable, method needs external intervention at the granularity of BIST control instructions. This was designed keeping in mind that BIST should also support static test schedules created using a test-scheduling algorithm. It may be desired that the BIST infrastructure drives itself adhering to the basic philosophy of BIST, and creates dynamic test schedule under power and resource constraints. In such a case, effort is needed to dynamically identify power-hungry dies/cores and moderate the test activity in a chip. This will further require ability to pre-empt an ongoing test, save its state, and resume its execution from the saved state. Identifying power consumption of a chip requires additional hardware overhead such as on-chip temperature sensors, but these sensors are likely to be present today for performance monitoring in manycore SOCs. Advanced methods for generating dynamic test schedules based on temperature sensor readings are needed for more effective BIST.

Bibliography

[1] A. M. Amory *et al.*, “A New Test Scheduling Algorithm Based on Networks-on-Chip as Test Access Mechanisms,” *J. Parallel Distrib. Comput.*, vol. 71, pp. 675–686, May 2011.

[2] S. Goel and E. Marinissen, “Effective and Efficient Test Architecture Design for SOCs,” in *Proc. International Test Conference*, pp. 529–538, 2002.

[3] “Open Cores Standard.” `http://opencores.org/`.

[4] V. Muresan *et al.*, “Power-Constrained Block-Test List Scheduling,” in *Proc. Intl. Workshop on Rapid System Prototyping*, pp. 182–187, 2000.

[5] E. J. Marinissen *et al.*, “On IEEE P1500’s Standard for Embedded Core Test,” *J. Electronic Testing*, vol. 18, no. 4-5, pp. 365–383, 2002.

[6] M. Agrawal and K. Chakrabarty, “Test-Cost Optimization and Test-Flow Selection for 3D-Stacked ICs,” in *Proc. VTS*, pp. 276–281, Apr. 2013.

[7] G. Neela and J. Draper, “Logic-on-Logic Partitioning Techniques for 3-Dimensional Integrated Circuits,” in *Intl. Symp. Circuits and Systems (ISCAS)*, pp. 789–792, May 2013.

[8] E. J. Marinissen and Y. Zorian, “Testing 3D Chips Containing Through-Silicon Vias,” in *Proc. ITC*, 2009.

[9] A. Benso *et al.*, “HD2BIST: a Hierarchical Framework for BIST Scheduling, Data Patterns Delivering and Diagnosis in SoCs,” in *Proc. ITC*, pp. 892–901, Oct. 2000.

[10] “International Technology Roadmap for Semiconductors (ITRS) 2011.” `http://www.itrs.net/Links/2011ITRS/Home2011.htm`.

[11] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits.* Boston: Kluwer academic publishers, 2000.

[12] L. Yan and J. English, “Economic Cost Modeling of Environmental-Stress-Screening and Burn-In,” *Trans. Reliability*, vol. 46, no. 2, pp. 275–282, 1997.

[13] Y. Zorian, E. Marinissen, and S. Dey, "Testing Embedded-Core-Based System Chips," *Computer*, vol. 32, pp. 52–60, June 1999.

[14] E. Marinissen *et al.*, "Towards a Standard for Embedded Core Test: An Example," in *Proc. International Test Conference (ITC)*, pp. 616–627, 1999.

[15] E. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," in *Proc. International Test Conference (ITC)*, pp. 284–293, 1998.

[16] P. Harrod, "Testing Reusable IP-A Case Study," in *Proc. International Test Conference (ITC)*, pp. 493–498, 1999.

[17] P. Varma and S. Bhatia, "A Structured Test Re-Use Methodology for Core-Based System Chips," in *Proc. International Test Conference (ITC)*, pp. 294–302, 1998.

[18] V. Iyengar *et al.*, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip," *JETTA*, vol. 18, pp. 213–230, 2002.

[19] Y. Huang *et al.*, "On Concurrent Test of Core-Based SOC Design," *J. Electronic Testing*, vol. 18, no. 4-5, pp. 401–414, 2002.

[20] K. Chakrabarty, V. Iyengar, and M. Krasniewski, "Test Planning for Modular Testing of Hierarchical SOCs," *Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 435–448, March 2005.

[21] F. da Silva, T. Mclaurin, and T. Waayers, "The Core Test Wrapper Handbook: Rationale and Application of IEEE Std. 1500," Springer Verlag, 2006.

[22] V. Iyengar *et al.*, "Efficient Test Access Mechanism Optimization for System-on-Chip," *IEEE Trans. CAD*, vol. 22, pp. 635–643, May 2003.

[23] E. Larsson and Z. Peng, "Test Scheduling and Scan-Chain Division under Power Constraint," in *Proc. Asian Test Symposium (ATS)*, pp. 259–264, Nov. 2001.

[24] E. Larsson and Z. Peng, "An Integrated Framework for the design and Optimization of SOC Test Solutions," *J. Electronic Testing*, vol. 18, pp. 385–400, 2002.

[25] C. Zeferino *et al.*, "A Study on Communication Issues for Systems-on-Chip," in *Symp. Integrated Cir. and Sys. Design*, pp. 121–126, Sep. 2002.

[26] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Comput. Surv.*, vol. 38, pp. 1–51, 2006.

[27] "OCP-IP." `http://www.ocpip.org/`.

[28] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, pp. 70–78, 2002.

[29] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient Unicast and Multicast Support for CMPs," in *Proc. MICRO-41 Microarchitecture 2008 41st IEEE/ACM Int. Symp*, pp. 364–375, 2008.

[30] `http://www.intel.com/pressroom/kits/teraflops/`.

[31] `http://www.tilera.com`.

[32] M. Lis, P. Ren, M. H. Cho, K. S. Shim, C. W. Fletcher, O. Khan, and S. Devadas, "Scalable, accurate Multicore Simulation in the 1000-Core Era," in *Intl. Symp. Performance Analysis of Systems and Software (ISPASS)*, pp. 175–185, 2011.

[33] M. Richter and K. Chakrabarty, "Test Pin Count Reduction for NOC-Based Test Delivery in Multicore SOCs," in *Proc. DATE*, pp. 787–792, Mar. 2012.

[34] E. Cota *et al.*, "Reusing an On-Chip Network for the Test of Core-Based Systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, pp. 471–499, Oct. 2004.

[35] A. M. Amory *et al.*, "DfT for the Reuse of Networks-on-Chip as Test Access Mechanism," in *Proc. IEEE VTS*, pp. 435–440, May 2007.

[36] E. Cota *et al.*, "The Impact of NoC Reuse on the Testing of Core-Based Systems," in *Proc. IEEE VTS*, pp. 128–133, May 2003.

[37] E. Cota *et al.*, "Power-Aware NoC Reuse on the Testing of Core-Based Systems," in *Proc. Int. Test Conf*, pp. 612–621, Sep. 2003.

[38] A. M. Amory *et al.*, "Wrapper Design for the Reuse of Networks-on-Chip as Test Access Mechanism," in *Proc. ETS*, pp. 213–218, May 2006.

[39] J. M. Nolen and R. Mahapatra, "A TDM Test Scheduling Method for Network-on-Chip Systems," in *Proc Int. Workshop on Microprocessor Test and Verificatiom and validation*, pp. 90–98, Nov. 2005.

[40] C. Liu *et al.*, "Power-Aware Test Scheduling in Network-on-Chip Using Variable-Rate On-Chip Clocking," in *Proc. VTS*, pp. 349–354, May 2005.

[41] C. Liu *et al.*, "Reuse-Based Test Access and Integrated Test Scheduling for Network-on-Chip," in *Proc. DATE*, pp. 303–308, Mar. 2006.

[42] A. Sehgal *et al.*, "SOC Test Planning Using Virtual Test Access Architectures," *IEEE Trans. VLSI Sys*, vol. 12, pp. 1263–1276, Dec. 2004.

[43] K. Banerjee *et al.*, "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration," *Proc. IEEE*, vol. 89, pp. 602–633, May 2001.

[44] Y. Xie, "Processor Architecture Design Using 3D Integration Technology," in *Proc. Intl. Conf. VLSI Design*, pp. 446–451, Jan. 2010.

[45] U. Kang *et al.*, "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology," *IEEE J. Solid-State Circuits*, vol. 45, pp. 111–119, Jan. 2010.

[46] M. Kawano *et al.*, "A 3D Packaging Technology for 4 Gbit Stacked DRAM with 3 Gbps Data Transfer," in *Proc. IEDM*, pp. 1–4, 2006.

[47] B. Feero and P. Pande, "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *IEEE Transactions on Computers*, vol. 58, pp. 32–45, Jan. 2009.

[48] G. H. Loh, Y. Xie, and B. Black, "Processor Design in 3D Die-Stacking Technologies," *IEEE Micro*, vol. 27, pp. 31–48, May-June 2007.

[49] C. Ababei, P. Maidee, and K. Bazargan, "Exploring Potential Benefits of 3D FPGA Integration.," in *Field Programmable Logic and Application*, vol. 3203, pp. 874–880, Springer, 2004.

[50] B. Black *et al.*, "3D Processing Technology and its Impact on iA32 Microprocessors," in *Proc. ICCD*, pp. 316–318, 2004.

[51] B. Noia and K. Chakrabarty, "Pre-Bond Probing of TSVs in 3D Stacked ICs," in *Proc. ITC*, 2011.

[52] B. Noia *et al.*, "Test-Architecture Optimization and Test Scheduling for TSV-Based 3-D Stacked ICs," *IEEE Trans. CAD*, vol. 30, Nov. 2011.

[53] H.-H. Lee and K. Chakrabarty, "Test Challenges for 3D Integrated Circuits," *IEEE Design & Test of Computers*, vol. 26, pp. 26–35, Sept.-Oct. 2009.

[54] W. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. Sule, M. Steer, and P. Franzon, "Demystifying 3D ICs: the Pros and Cons of Going Vertical," *Design & Test of Computers*, vol. 22, no. 6, pp. 498–510, 2005.

[55] P. Garrou, C. Bower, and P. Ramm, *Handbook of 3D Integration Technology and Applications of 3D Integrated Circuits*. WeinHeim, Germany: Wiley-VCH, 2008.

[56] L. Jiang, R. Ye, and Q. Xu, "Yield Enhancement for 3D-Stacked Memory by Redundancy Sharing Across Dies," in *Proc. ITC*, pp. 230–234, Nov. 2010.

[57] E. Marinissen, J. Verbree, and M. Konijnenburg, "A Structured and Scalable Test Access Architecture for TSV-based 3D Stacked ICs," in *Proc. VLSI Test Symposium (VTS)*, pp. 269–274, April 2010.

[58] P. P. Pande *et al.*, "Design, Synthesis, and Test of Networks on Chips," *IEEE Design & Test of Computers*, Sep. 2005.

[59] A. Dalirsani *et al.*, "Structural Test for Graceful Degradation of NoC Switches," in *Proc. European Test Symp*, pp. 183–188, May 2011.

[60] C. Liu *et al.*, "Test Scheduling for Network-on-Chip with BIST and Precedence Constraints," in *Proc. Int. Test Conf.*, pp. 1369–1378, Oct. 2004.

[61] A. van den Berg *et al.*, "Bandwidth Analysis for Reusing Functional Interconnect as Test Access Mechanism," in *Proc. ETS*, pp. 21–26, May 2008.

[62] M. Li *et al.*, "An Efficient Wrapper Scan Chain Configuration Method for Network-on-Chip Testing," in *Proc. Symp. Emerging VLSI Technologies and Architectures*, pp. 135–140, Mar. 2006.

[63] F. A. Hussin *et al.*, "Optimization of NoC Wrapper Design Under Bandwidth and Test Time Constraints," in *Proc. ETS*, pp. 35–42, May 2007.

[64] J. Dalmasso *et al.*, "Improving the Test of NoC-Based SoCs with Help of Compression Schemes," in *Proc. Annual Symp. VLSI*, pp. 139–144, Apr. 2008.

[65] E. Cota and C. Liu, "Constraint-Driven Test Scheduling for NoC-Based Systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 11, pp. 2465–2478, 2006.

[66] S. Martello and P. Toth, "Subset-sum problem," in *Knapsack Problems: Algorithms and Computer Interpretations*, Wiley-Interscience, 1990.

[67] Y. Huang *et al.*, "Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design," in *IEEE Asian Test Symp*, 2001.

[68] D. Wentzlaff *et al.*, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, pp. 15–31, September 2007.

[69] P. Gratz *et al.*, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro*, vol. 27, pp. 41–50, September 2007.

[70] N. Nicolici and B. M. Al-Hashimi, "Power profile manipulation," in *Power-Constrained Testing of VLSI Circuits*, Kluwer Academic Publishers, 2003.

[71] M. Richter and K. Chakrabarty, "Optimization of Test Pin-Count, Test Scheduling, and Test Access for NoC-Based Multicore SOCs," *IEEE Trans. Computers*, 2013. Accepted.

[72] F. Samman, T. Hollstein, and M. Glesner, "Multicast Parallel Pipeline Router Architecture for Network-on-Chip," in *Proc. Design, Automation and Test in Europe*, pp. 1396–1401, 2008.

[73] N. Jerger, L.-S. Peh, and M. Lipasti, "Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support," in *Intl. Symp. Computer Architecture*, pp. 229–240, 2008.

[74] J. Postman, T. Krishna, C. Edmonds, L.-S. Peh, and P. Chiang, "SWIFT: A Low-Power Network-On-Chip Implementing the Token Flow Control Router Architecture With Swing-Reduced Interconnects," *IEEE Tran. VLSI Sys*, vol. 21, no. 8, pp. 1432–1446, 2013.

[75] E. J. Marinissen *et al.*, "A Set of Benchmarks for Modular Testing of SOCs," in *Proc. Int. Test Conf.*, pp. 519–528, Oct. 2002.

[76] "FICO Xpress Optimization Suite."

[77] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[78] D. Klein and C. D. Manning, "A Parsing: Fast Exact Viterbi Parse Selection," in *Proc. Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pp. 40–47, 2003.

[79] J. Liu and Y. Liu, "Application of A* Algorithm in Traffic Navigational System," in *Intl. Symp. Information Engineering and Electronic Commerce (IEEC)*, July 2010.

[80] A. Autere, "A* as an Optimal Resource Allocation Policy in Path Finding Problems," in *Proc. Florida Artificial Intelligence Research Society Conference*, pp. 139–144, 2001.

[81] M. Taouil *et al.*, "Test Cost Analysis for 3D Die-to-Wafer Stacking," in *Proc. ATS*, pp. 435–441, 2010.

[82] Y. Chen *et al.*, "Cost-Effective Integration of Three-dimensional (3D) ICs Emphasizing Testing Cost Analysis," in *Proc. ICCAD*, pp. 471–476, 2010.

[83] Y.-W. Chou *et al.*, "Cost Modeling and Analysis for Interposer-Based Three-Dimensional IC," in *Proc. VTS*, pp. 108–113, 2012.

[84] M. Taouil, S. Hamdioui, E. Marinissen, and S. Bhawmik, "3D-COSTAR: A Cost Model For 3D Stacked ICs," in *International Workshop on Testing Three-Dimensional Stacked Integrated Circuits*, 2012.

[85] M. Taouil, S. Hamdioui, E. J. Marinissen, and S. Bhawmik, "Using 3D-COSTAR for 2.5 D Test Cost Optimization," in *3D Systems Integration Conference (3DIC)*, pp. 1–8, 2013.

[86] M. Taouil, S. Hamdioui, E. J. Marinissen, and S. Bhawmik, "Impact of Mid-Bond Testing in 3D Stacked ICs," in *Intl. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 178–183, 2013.

[87] M. Taouil, S. Hamdioui, and E. J. Marinissen, "Quality versus Cost Analysis for 3D Stacked ICs," in *Proc. VTS*, 2014.

[88] T. Williams and N. Brown, "Defect Level as a Function of Fault Coverage," *IEEE Trans. Computers*, vol. C-30, pp. 987–988, Dec. 1981.

[89] J. Schat, "On the Relationship between Stuck-at Fault Coverage and Transition Fault Coverage," in *Proc. DATE*, pp. 1218–1221, 2009.

[90] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall, 2009.

[91] "Sematech Wafer Cost Comparison Calculator." `http://ismi.sematech.org/modeling/agreements/wafercalc.htm`.

[92] D. De Vries, "Investigation of Gross Die per Wafer Formulas," *Tran. Semiconductor Manufacturing*, vol. 18, pp. 136–139, Feb 2005.

[93] D. Velenis *et al.*, "Impact of 3D Design Choices on Manufacturing Cost," in *Intl. Conf. 3D System Integration*, Sept. 2009.

[94] S. Edbom and E. Larsson, "An Integrated Technique for Test Vector Selection and Test Scheduling under Test Time Constraint," in *Asian Test Symposium (ATS)*, pp. 254–257, Nov. 2004.

[95] D. H. Kim, K. Athikulwongse, and S.-K. Lim, "A Study of Through-Silicon-Via Impact on the 3D-Stacked IC Layout," in *Intl. Conf. Computer-Aided Design*, pp. 674–680, Nov 2009.

[96] J. Li and D. Xiang, "DfT Optimization for Pre-Bond Testing of 3D-SICs containing TSVs," in *Proc. ICCD*, pp. 474–479, Oct 2010.

[97] A. Kumar, S. Reddy, I. Pomeranz, and B. Becker, "Hyper-Graph Based Partitioning to Reduce DFT Cost for Pre-Bond 3D-IC Testing," in *Proc. DATE*, pp. 1–6, March 2011.

[98] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, pp. 85–103, 1972.

[99] E. J. Marinissen, C.-C. Chi, M. H. Konijnenburg, and J. Verbree, "A DfT Architecture for 3D-SICs Based on a Standardizable Die Wrapper," *J. Electronic Testing*, vol. 28, pp. 73–92, Feb. 2012.

[100] "P1838 - Standard for Test Access Architecture for Three-Dimensional Stacked Integrated Circuits." `https://standards.ieee.org/develop/project/1838.html`.

[101] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability.* Academic Press, 2006.

[102] C.-J. Tseng and D. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. CAD*, vol. 5, pp. 379–395, July 1986.

[103] J. Bhasker and T. Samad, "The Clique-Partitioning Problem," *Computers & Mathematics with Applications*, vol. 22, no. 6, pp. 1–11, 1991.

[104] "ITC 1999 Benchmark." `http://www.cerc.utexas.edu/itc99-benchmarks/bench.html`.

[105] N. Selvakkumaran and G. Karypis, "Multiobjective Hypergraph-Partitioning Algorithms for Cut and Maximum Subdomain-Degree Minimization," *IEEE Trans. on CAD*, vol. 25, no. 3, pp. 504–517, 2006.

[106] F. Beenker *et al.*, "Implementing Macro Test in Silicon Compiler Design," *IEEE Design & Test of Computers*, vol. 7, Apr. 1990.

[107] O. F. Haberl and T. Kropf, "HIST: A Methodology for the Automatic Insertion of a Hierarchical Self Test," in *Proc. ITC*, Sep. 1992.

[108] M. Taouil *et al.*, "Exploring Test Opportunities for Memory and Interconnects in 3D ICs," in *Proceedings Design and Test Symposium (IDT)*, pp. 1–6, Dec. 2013.

[109] C.-S. Hou and J.-F. Li, "Allocation of RAM Built-In Self-Repair Circuits for SOC Dies of 3D ICs," in *Proc. VTS*, pp. 270–275, Apr. 2013.

[110] S. Deutsch and K. Chakrabarty, "Non-Invasive Pre-Bond TSV Test Using Ring Oscillators and Multiple Voltage Levels," in *Proc. DATE Conf.*, pp. 1065–1070, Mar. 2013.

[111] Y. Fkih *et al.*, "A 3D IC BIST for Pre-Bond Test of TSVs Using Ring Oscillators," in *Proc. NEWCAS*, pp. 1–4, Jun. 2013.

[112] Y.-J. Huang, J.-F. Li, and C.-W. Chou, "Post-Bond Test Techniques for TSVs with Crosstalk Faults in 3D ICs," in *Proc. VLSI-DAT*, Apr. 2012.

[113] B. SenGupta, U. Ingelsson, and E. Larsson, "Scheduling Tests for 3D Stacked Chips under Power Constraints," *JETTA*, vol. 28, Feb. 2012.

[114] R. Chou, K. Saluja, and V. Agrawal, "Scheduling Tests for VLSI Systems under Power Constraints," *IEEE Trans. VLSI Systems*, vol. 5, pp. 175–185, Jun. 1997.

[115] B. Chazelle, "The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation," *IEEE Computer*, vol. C-32, pp. 697–707, Aug. 1983.

[116] J. Jylänki, "A Thousand Ways to Pack the Bin-A Practical Approach to Two-Dimensional Rectangle Bin Packing," *retrieved from http://clb. demon. fi/files/RectangleBinPack. pdf*, 2010.

[117] "International Technology Roadmap for Semiconductors (ITRS) 2013." `http://www.itrs.net/Links/2013ITRS/Home2013.htm`.

Biography

PERSONAL DATA

- Mukesh Agrawal
- Place of birth: Patna, India
- Data of birth: August 12, 1985

EDUCATION

- Ph.D., Duke University, USA, 2014 (expected)
- M. Tech & B. Tech, Indian Institute of Technology Kharagpur, India, 2008

PUBLICATIONS

1. M. Agrawal and K. Chakrabarty, “Test-Cost Modeling and Optimal Test-Flow Selection of 3D-Stacked ICs”, submitted to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

2. M. Agrawal, K. Chakrabarty and R. Widialaksono, “Reuse-Based Optimization for Pre-Bond and Post-Bond Testing of 3D-Stacked ICs”, accepted for publication in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

3. K. Chakrabarty, M. Agrawal, S. Deutsch, B. Noia, R. Wang and F. Ye, “Test and Design-for-Testability Solutions for 3D Integrated Circuits” (Invited Paper), *Journal of Information Processing (Information Processing Society of Japan)*, vol. 7, pp. 56-73, August 2014.*

4. M. Agrawal, M. Richter and K. Chakrabarty, “Test-Delivery Optimization in Manycore SOCs”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, pp. 1067-1080, July 2014.

5. M. Agrawal and K. Chakrabarty, “A Distributed, Reconfigurable, and Reusable BIST Infrastructure for 3D-Stacked ICs”, in *Proc. IEEE International Test Conference*, 2014.

6. M. Agrawal and K. Chakrabarty, "Test-Time Optimization in NOC-Based Manycore SOCs using Multicast Routing", in *Proc. IEEE VLSI Test Symposium*, pp. 276-281, 2014.

7. M. Agrawal and K. Chakrabarty, "A Graph-Theoretic Approach for Minimizing the Number of Wrapper Cells for Pre-Bond Testing of 3D-Stacked ICs", in *Proc. IEEE International Test Conference*, 2013.

8. M. Agrawal and K. Chakrabarty, "Test-Cost Optimization and Test-Flow Selection for 3D-Stacked ICs", in *Proc. IEEE VLSI Test Symposium*, pp. 276-281, 2013.

9. M. Agrawal, M. Richter and K. Chakrabarty, "A Dynamic Programming Solution for Optimizing Test Delivery in Multicore SoCs", in *Proc. IEEE International Test Conference*, 2012.

10. M. Agrawal, Q. Duan, K. Chakrabarty, J. Zeng, I.-J. Lin, G. Dispoto and Y.-S. Lee, "Digital Print Workflow Optimization under Due-Dates, Opportunity Cost and Resource Constraints", in *Proc. IEEE International Conference on Industrial Informatics (INDIN)*, 2011.*

11. M. Agrawal, K. Chakrabarty, J. Zeng, I.-J. Lin, G. Dispoto, "Simultaneous Task Scheduling and Resource Binding for Digital Print Automation", *IIE Industrial Engineering Research Conference (IERC)*, 2011.*

12. M. Agrawal, S. Karmakar, D. Saha and D. Mukhopadhyay, "Scan-Based Side-Channel Attacks on Stream Ciphers and their Counter Measures", in *Proc. International Conference on Cryptology in India: Progress in Cryptology*, 226-238, 2008.*

*Not related to PhD thesis work.