VLSI Testing of System on Chip : Incomplete Testing

Thesis submitted to the
Indian Institute of Technology Guwahati
for the award of the degree

of

Doctor of Philosophy
in
Computer Science and Engineering

Submitted by
Kunwer Mrityunjay Singh

Under the guidance of
Prof. Jatindra Kumar Deka and Prof. Santosh Biswas

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
October 23, 2024

Abstract

System on Chip (SoC) consolidates an entire computer or electronic system onto a single chip by employing reusable embedded cores. An SoC typically comprises diverse cores, encompassing elements like memory, which involves both *Random Access Memory* (RAM) and Flash Memory, in addition to a *Graphics Processing Units* (GPU). It also includes *Input/Output* (I/O) Interfaces, Peripherals, an *Audio Processing Unit* (APU), a *Image Signal Processor* (ISP), Wireless Interfaces, Sensor Interfaces, Security Modules, Bus Systems, Clock and Timing Control, Power Management capabilities, Analog Interfaces, a *Direct Memory Access* (DMA) Controller, and more. SoCs may exhibit various logical and manufacturing flaws, such as Stuck-at Faults, Transition Faults, Path Delay Faults, Bridge Faults, and so forth. These imperfections can impact the performance of devices employing SoCs, potentially leading to deviations from the intended output. Therefore, it is imperative to conduct comprehensive testing of these integrated chips before they are introduced to the market to ensure the optimal performance of devices incorporating SoCs. Testing SoCs presents an array of challenges, encompassing complexities in design, intricate interconnections, limited accessibility to internal components, temperature fluctuations during testing, the imperative need for hardware and software synchronization, factors such as *Test Power Consumption* (TP), *Test Access Time* (TAT), and generated *Test Data Volume* (TDV), as well as testing expenses. Some cores may be deeply embedded within the SoC, necessitating the utilization of a *Test Access Mechanism* (TAM) for their accessibility and evaluation. The *Joint Test Action Group* (JTAG) architecture offers a standardized approach for testing devices with boundary scan cells, enabling the examination of input and output pins, as well as their connecting pathways.

Numerous established and validated standardized methods are at the disposal of SoC testing for comprehensive assessment. Nevertheless, the challenge in SoC testing expands when dealing with the exhaustive testing of individual cores and their intricate interconnections.

As contemporary manufacturing technology advances, SoCs consist an escalating number of cores, leading to the production of larger and more complex SoCs. These standard testing methods, while rigorous, are often slow and may become unfeasible for testing larger SoCs. In this thesis, we introduce heuristic-based *Incomplete Testing* approaches that, while partial in their scope, offer viable alternatives for testing SoCs of substantial scale.

The entire thesis work comprises several contributions organized into eight phases. In the initial phase, an overview of the thesis is provided. Phase 2 entails an extensive literature survey focusing on comprehensive and incomplete testing of SoCs. In Phase 3, a heuristic-based incomplete testing method is introduced considering stuck-at faults. Its primary objective is to minimize the generated TDV. Phase 4 extends the incomplete testing approach to optimize TAT and perform an analysis of the impact of incomplete testing on TAT. Phase 5 extends the incomplete testing approach to optimize TP and perform an analysis of the impact of incomplete testing on TP. Additionally, a TAM architecture is proposed to support incomplete testing. Phase 6 explores an alternative incomplete testing method involving approximate computing and illustrates its impact on TAT and TP. In Phase 7, a boolean satisfiability based method is developed for incomplete SoC testing, with a particular consideration for bridging faults. In the final phase, the thesis is concluded, and potential future avenues and prospects for this body of work are delineated. Our theoretical techniques are substantiated by experimental results and our published works.

Declaration

I certify that:

a. The work contained in this thesis is original and has been done by me under the guidance of my supervisors.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in preparing the thesis.

d. I have conformed to the norms and guidelines given in the *Ethical Code of Conduct of the Institute.*

e. Whenever I have used materials (theoretical analysis, figures, data, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Kunwer Mrityunjay Singh

Copyright

Signature of Author...

Kunwer Mrityunjay Singh

Certificate

This is to certify that this thesis entitled, **"VLSI Testing of System on Chip : Incomplete Testing"**, being submitted by **Kunwer Mrityunjay Singh**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under our supervision and guidance. The thesis, in our opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulation of the institute. To the best of our knowledge, it has not been submitted elsewhere for the award of the degree.

....

Dr. Jatindra Kumar Deka
Professor
Department of Computer Science and Engineering
IIT Guwahati

....

Dr. Santosh Biswas
Professor
Department of Electrical Engineering and Computer Science
IIT Bhilai

Dedicated to

My Mother

who bestowed upon me a second chance at life did so by generously donating her kidney for my kidney transplant.

My Father

For everything I achieved till now

My Wife

For taking care of my second life after kidney transplant

My Sisters and Family

For being my strength

Acknowledgments

A lot of people have contributed to the production of this dissertation. I owe my gratitude to all those people who have made this possible.

I wish to express my deepest gratitude to my supervisors, Prof. Jatindra Kumar Deka and Prof. Santosh Biswas for their valuable guidance, inspiration, and advice. I feel very privileged to have had the opportunity to learn from, and work with them. Their constant guidance and support not only paved the way for my development as a research scientist but also changed my personality, ability, and nature in many ways. I have been fortunate to have such advisors who gave me the freedom to explore on my own and at the same time the guidance to recover when my steps faltered. Besides my advisors, I would like to thank the rest of my thesis committee members: Prof. Diganta Goswami, Dr. Aryabartta Sahu, and Dr. Partha Sarathi Mandal, for their insightful comments and encouragement. Their comments and suggestions helped me to widen my research from various perspectives.

I would also like to express my heartful gratitude to the director, the deans and other managements of IIT Guwahati whose collective efforts has made this institute a place for world-class studies and education. I am thankful to all faculty specially Dr. Arnab Sarkar and staff of Department of Computer Science and Engineering for extending their co-operation in terms of technical and official support for the successful completion of my research work.

Contents

1 Introduction **1**
- 1.1 Background: SoC and Test Architecture . . . 2
- 1.2 Challenges . . . 7
- 1.3 Motivation . . . 9
- 1.4 Objectives . . . 12
- 1.5 Contributions . . . 12
- 1.6 Organization of the Thesis . . . 14

2 Literature Survey **17**
- 2.1 Basics of Digital VLSI Testing . . . 18
  - 2.1.1 Fault Coverage . . . 18
  - 2.1.2 Test Coverage . . . 19
- 2.2 Fault Models . . . 19
  - 2.2.1 Stuck-at Fault Model . . . 19
  - 2.2.2 Bridging Fault Model . . . 20
  - 2.2.3 Delay Fault Model . . . 21
  - 2.2.4 Fault Collapse Model . . . 21
  - 2.2.5 Transition Fault Model . . . 21
  - 2.2.6 Open Fault Model . . . 21
- 2.3 Reviews of VLSI Test Technologies . . . 22
  - 2.3.1 ATE . . . 23
  - 2.3.2 ATPG . . . 24
  - 2.3.3 Fault Simulation . . . 24

2.3.4 DFT 24
2.3.5 BIST 25
2.3.6 Boundary Scan Testing 25
2.4 SoC 25
2.5 Core Based SoC Design 25
2.6 SoC Test Architecture 27
2.6.1 IEEE 1500 std. for SoC Testing 27
2.6.2 Core Wrapper 28
2.6.3 WIR 28
2.6.4 Test Access Mechanism (TAM) 29
2.7 Challenges in SoC Testing 32
2.8 Strategies to Reduce the TDV 35
2.8.1 Scan Chain Optimization 35
2.8.2 Test Compression Techniques 36
2.8.3 Test Data Partitioning 38
2.8.4 Compaction Techniques 39
2.8.5 Combination of Compression and Compaction Techniques 39
2.9 Strategies to Reduce the TAT 40
2.10 Strategies to Reduce TP 42
2.11 Confidence Aware Testing Methods 44
2.12 Incomplete Testing 45

3 Incomplete Testing of SoC : Heuristic Approach 49
3.1 Introduction 49
3.2 Problem Statement 50
3.3 Proposed Method 50
3.3.1 Algorithm 1: *Least Significant Test Bits* (LSTB) Detector 51
3.3.2 Complexity of Algorithm 1 53
3.4 Example 53
3.5 Experimental Results 58
3.5.1 SoC Architecture with Combinational Cores 58
3.5.2 Results for TDV and FC 59
3.5.3 SoC Architecture with Sequential Cores 60

3.5.4 Comparison of our Method with Others Methods 66
3.6 Conclusion and Future Work 67

4 TAT Aware Incomplete Testing **69**
4.1 Introduction 69
4.2 Incomplete Testing 70
4.3 Problem Statement 70
4.4 Proposed Method 72
4.4.1 Particle Structure 72
4.4.2 Calculation of TAT 73
4.4.3 Population Size 74
4.4.4 Fitness Function 75
4.5 Incomplete Testing and Reduction in TAT 75
4.5.1 Problem Statement 75
4.5.2 Proposed Modified TAT Equations for Incomplete Testing 75
4.6 Experimental Results 78
4.6.1 SoC with Combinational Cores 79
4.6.2 SoC with Sequential Cores and Combinational Cores . . . 81
4.6.3 Comparison with Other Methods 84
4.7 Conclusion and Future Work 85

5 TP Aware Incomplete Testing **87**
5.1 Introduction 87
5.2 Incomplete Testing 88
5.3 Calculation of TP 88
5.3.1 Standard Equations to Compute TP 90
5.4 Problem Statement 93
5.5 Proposed TAM Architecture 93
5.5.1 The IEEE 1500 Wrapper Architecture 93
5.5.2 WBY 95
5.5.3 WS_BYPASS Instruction 96
5.5.4 Proposed TAM for Incomplete Testing 97
5.5.5 Incomplete Testing and Reduction in TP 100

5.6 Experimental Results 101
5.6.1 Results for TP for SoC with Combinational Cores 102
5.6.2 Results for TP for SoC with Sequential and Combinational Cores 108
5.7 Conclusion and Future Work 112

6 Incomplete Testing Based on Approximation and its Impact on TP and TAT 115
6.1 Introduction 115
6.2 Problem Formulation 116
6.3 Proposed Method 116
6.3.1 Step 1: 120
6.3.2 Step 2: 120
6.3.3 Step 3: 120
6.4 Impact on Fault Coverage (FC) 123
6.5 Impact on TDV 126
6.6 Impact on TAT 128
6.6.1 Computation of TAT 128
6.6.2 Reduction in TAT 128
6.7 Impact on TP 130
6.7.1 Reduction In TP In Incomplete Testing 130
6.8 Experimental Results 131
6.8.1 Results for FC and TDV 132
6.8.2 Results for TAT 133
6.8.3 Results For TP 136
6.9 Conclusion and Future work 137

7 *Automatic Test Pattern Generation* (ATPG) for Incomplete Testing of SoC having Bridge Faults 139
7.1 Problem Formulation 140
7.2 Proposed Scheme : Boolean Satisfiability Method for Incomplete Testing of SoC 140
7.2.1 CNF 141

7.2.2 CNF Formulation for Original Circuit 142
7.2.3 Introduction of Bridging Faults in Original Circuit 143
7.2.4 Reduction of Boolean Satisfiability Equation for Incomplete Testing 145
7.2.5 Satisfying the Formula 147
7.2.6 Heuristic Approach using Particle Swarm Optimization . . 148
7.3 Experimental Results 150
7.4 Conclusion and Future Work 153

8 Conclusions and Future Perspectives 155
8.1 Summarization 155
8.2 Future Works 158

References 161

List of Figures

1.1 Core Based SoC Design 3
1.2 Conceptual Architecture of Core Based Testing 5

2.1 Basic Testing Outlook [44] 22
2.2 Core Based SoC Design [13] 26
2.3 IEEE 1500 Std. Test Architecture 28
2.4 Test Interface of Core Wrapper [59] 29
2.5 Test Data Compression [64] 36

3.1 Core with I/O 53
3.2 Test Vector Set (T_c) for a Core 54
3.3 Test Vector Set T_c for the Core in Example 55
3.4 Test Vector Modifications 57
3.5 Architecture of SoC S1. 59
3.6 Architecture of SoC S2. 61
3.7 TDV Characteristics 64
3.8 Characteristics in Trade-off between FC and TDV reduction . . . 65

4.1 Flow Diagram for TAT Aware Incomplete Testing of SoC 71
4.2 Sample SoC 73
4.3 Particle Structure 73
4.4 SoC S1 79
4.5 SoC S2 81
4.6 TAT Characteristics 83

4.7 SoC S3 84

5.1 Flow Diagram for Incomplete Testing of SoC 89
5.2 Computation of transitions in wrapper cells 90
5.3 Transitions while shifting test vectors 91
5.4 IEEE 1500 wrapper core design 94
5.5 WBY internal structure 95
5.6 WRCK waveform 96
5.7 IEEE 1500 Design for Incomplete Testing 98
5.8 TAM architecture with WBY Normal Mode 99
5.9 TAM architecture with WBY Bypass Mode 99
5.10 Proposed TAM architecture 99
5.11 Equations to calculate transitions in wrapper chain after insertion of two test vector sequentially 102
5.12 Bus Width and TP saving in % 109
5.13 SoC S2 110
5.14 SoC S2 111
5.15 TP Characteristics 112

6.1 Multiplication of two 3 bit numbers 116
6.2 3 bit multiplier 117
6.3 Example of multiplication of 3 bit numbers 118
6.4 Multiplier circuit 119
6.5 Benchmark circuit of c17 121
6.6 c17 circuit 121
6.7 Equivalent graph of c17 circuit 122
6.8 Backtrack path for c17 circuit 123
6.9 Equivalent graph of c17 circuit for testing 124
6.10 c17 benchmark circuit incomplete testing 124
6.11 c17 benchmark circuit incomplete testing 125
6.12 Test vector set for c17 benchmark circuit 126
6.13 Modified test vector set for c17 benchmark circuit 127
6.14 Test patterns for c17 benchmark circuit in complete testing 128
6.15 Test patterns for c17 benchmark circuit in incomplete testing . . . 128

6.16 Modified TAM architecture for incomplete testing 130
6.17 SOC used for experiments 132

7.1 141
7.2 Various Gates having CNF formula 142
7.3 C17 benchmark circuit 142
7.4 Bridging Fault between L6 and L7 144
7.5 Testing of Bridging Fault 144
7.6 Equivalent graph of C17 146
7.7 Path from output to input not to be tested 146
7.8 Miter Circuit 149
7.9 SoC for experiments 151

List of Algorithms

1 PSO Based LSTBs Detector 52

2 Algorithm for finding GATES not to be tested 121
3 Backtrack (i) 122

4 Algorithm for incomplete testing considering bridging faults 147
5 backtrack (m) 147

List of Tables

3.1 TDV and FC comparison for complete and incomplete testing . . 59
3.2 Comparison in Fault coverage with original and modified test vector set . . . 61
3.3 Reduction in TDV . . . 63
3.4 Trade-off between Fault Coverage and TDV . . . 65
3.5 Comparison for reduction in TDV with other compression techniques 66
3.6 Comparison for reduction in TDV with compression techniques . . 67

4.1 TAT for complete testing . . . 80
4.2 TAT for incomplete testing . . . 80
4.3 Comparison between complete testing and incomplete testing . . 81
4.4 Optimal Assignment and Reduction in TAT for 5% compromise in FC . . . 82
4.5 Comparison between proposed method and other method 85

5.1 TP for Bus Width Distribution of (1,47) . . . 103
5.2 TP for Bus Width Distribution of (4,44) . . . 104
5.3 TP for Bus Width Distribution of (40,8) . . . 104
5.4 TP for Bus Width Distribution of (13,35) . . . 105
5.5 TP for Bus Width Distribution of (16,32) . . . 106
5.6 TP for Bus Width Distribution (18,30) . . . 106
5.7 TP for Bus Width Distribution of (28,20) . . . 107
5.8 Analysis of whole SoC S1 for TP Saving . . . 108

5.9 TP Analysis for SoC with various Bus Width Distribution 112

6.1 Results for FC and TDV 133
6.2 Results for TAT 134
6.3 Results for TP for complete testing 134
6.4 Results for TP for incomplete testing 135
6.5 Results for comparison of TP 135

7.1 Comparison of our method with the existing methods for wired-AND bridging faults 152
7.2 Comparison of our method with the existing methods for wired-OR bridging faults 152

List of Acronyms

SoC	*System on Chip*	
FC	*Fault Coverage*	59
TAT	*Test Access Time*	
TP	*Test Power Consumption*	
TDV	*Test Data Volume*	
IC	*Integrated Circuit*	2
IP	*Intellectual Property*	3
ASIC	*Application-Specific Integrated Circuits*	3
I/O	*Input/Output*	
APU	*Audio Processing Unit*	
DMA	*Direct Memory Access*	
ISP	*Image Signal Processor*	
UDL	*User-Defined Logic Blocks*	4
DFT	*Design-for-Testability*	4
CUT	*Circuit Under Test*	4
ATE	*Automatic Test Equipment*	5
BIST	*Built-In Self Test*	5
TAM	*Test Access Mechanism*	

CPU	*Central Processing Unit*	9
ATPG	*Automatic Test Pattern Generation*	iv
PCB	*Printed Circuit Board*	23
ADC	*Analog-to-Digital Converters*	27
DAC	*Digital to Analog Converters*	27
PLL	*Phase Locked Loop*	27
PSO	*Particle Swarm Optimization*	12
LSTB	*Least Significant Test Bits*	ii
FDR	*Frequency Directed Run Length*	38
ILP	*Integer Linear Programming*	41
LFSR	*Linear Feedback Shift Register*	43
WBR	*Wrapper Boundary Registers*	94
CNF	*Conjunctive Normal Form*	15
VLSI	*Very Large Scale Integration*	4
LSSD	*Level-Sensitive Scan Design*	24
TPG	*Test-Pattern Generator*	25
ORA	*Output Response Analyzer*	25
JTAG	*Joint Test Action Group*	
GPU	*Graphics Processing Units*	
DSP	*Digital Signal Processors*	11
RAM	*Random Access Memory*	
IP	*Intellectual Property*	3
WBR	*Wrapper Boundary Register*	94
WDR	*Wrapper Data Register*	96
WIR	*Wrapper Instruction Register*	28
WBY	*Wrapper Bypass Register*	95

WRCK	*Wrapper Clock Terminal*	100
WSI	*Wrapper Serial Input*	30
WSO	*Wrapper Serial Output*	30
WSP	*Wrapper Serial Port*	30
CTL	*Core Test Language*	93
TDI	*Test Data Input*	28
TDO	*Test Data Output*	28
TMS	*Test Mode Select*	28
TCK	*Test Clock*	28
TCR	*Test Control Register*	28
TMC	*Test Mode Controller*	29
WSC	*Wrapper Serial Control*	31
WPP	*Wrapper Parallel Port*	31
WPI	*Wrapper Parallel Input*	31
WPO	*Wrapper Parallel Output*	31
WPC	*Wrapper Parallel Control*	32
TRC	*Test Response Compression*	36
TDC	*Test Data Compression*	36
ILP	*Integer Linear Programming*	41
MAC	*Multiply-Accumulate*	45
DPB	*Decoded Picture Buffer*	46

Chapter 1

Introduction

In today's modern age, we frequently utilize electronic gadgets, smartphones, and multimedia devices, all of which rely on SoC technology. Prior to launching any product into the market, there are two crucial phases it must undergo: (1) Manufacturing and (2) Testing. Both manufacturing and testing domains offer extensive opportunities for research. Testing plays a pivotal role in ensuring that a product is free of defects and ready for consumers. To guarantee the performance of a product, it's imperative to conduct SoC testing. These SoCs can encompass a wide range of deeply embedded cores, each responsible for executing various tasks within these devices. SoC testing essentially revolves around testing each individual core.

To carry out testing on SoCs, these cores must be accessible via the input and output pins of the SoC. Core testing is accomplished by applying test vectors to these cores through these input and output pins. There exist numerous challenges in the realm of SoC testing [1, 2], such as achieving comprehensive fault coverage, managing the storage of test vectors, ensuring the smooth transport of test vectors from testing equipment to the cores, and dealing with limitations in transportation channel bandwidth, among others.

Traditional testing methods are available to address these challenges. However, in today's consumer-driven landscape, customers demand more from their devices. They want to run a multitude of applications on their gadgets, which in turn necessitates the integration of additional cores into SoCs. The inclusion of more cores in SoCs has a profound impact on TDV. As the number of test vectors grows, the testing process becomes increasingly complex. Transporting these test vectors from testing equipment to the input pins of the cores becomes a formidable task, especially when contending with the limitations of transportation channel bandwidth.

Storing a substantial volume of test vectors comes with the drawback of increasing the required storage space. Expanding the size of test vectors leads to complications in transporting them, storing them, and dealing with the limitations of channel bandwidth. Furthermore, testing a greater number of cores amplifies both the TAT and TP. Assessing a higher quantity of cores demands more time for comprehensive SoC testing, and sometimes this time frame can become impractical. Consequently, the principal challenges in SoC testing encompass the escalation of TDV, TAT and TP among others.

Under these conditions, conventional testing methods tend to become slow, expensive, and on occasion, inviable. In today's price-competitive market, cost also becomes a decisive factor. Manufacturers face a challenging task in delivering a product that meets all customer requirements while maintaining a low price point. Achieving such a balance often requires compromising on quality. In this thesis, several methods are proposed that strike a balance between the quality of testing and the conservation of TDV, TAT, and TP.

1.1 Background: SoC and Test Architecture

SoC is an *Integrated Circuit* (IC) that consolidates all the elements of a computer or electronic system onto a single chip. Typically, a SoC encompasses various processing units such as microprocessors, microcontrollers, and digital signal processors. It also incorporates memory components like RAM, ROM, and flash memory, along with essential elements like timing sources such as oscilla-

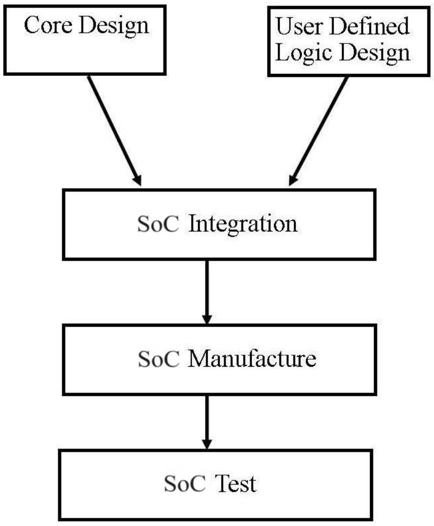


Figure 1.1: *Core Based SoC Design* [13]

tors and analog-to-digital converters [3, 4]. When employed in communication devices, SoCs include cellular and other radios to facilitate 4G, Bluetooth, or Wi-Fi connectivity. An illustrative example of a SoC is the AMD Geode. SoCs have gained significant popularity in today's market. They are continually advancing in terms of performance, with examples like the Apple A15 Bionic (utilized in the iPhone 13 series) [5], Qualcomm Snapdragon 888 (found in various high-end Android smartphones [6]), Samsung Exynos 2100 (used in select Samsung Galaxy devices [7]), AMD Ryzen 5000 Series (deployed in desktop computers [8]), Intel Core 11th Gen (employed in both laptops and desktops [9]), Nvidia Tegra X1 (utilized in the Nintendo Switch [10]), Tesla Full Self-Driving (FSD) Computer (employed in Tesla vehicles [11]), and Google Tensor Processing Units (TPUs, utilized in data centers [12]).

1. **Core Based SoC Design [13]**

   In SoC, *Intellectual Property* (IP) cores are the main building blocks. Even cores can be in deep. IP cores or *Application-Specific Integrated Circuits*

(ASIC) can also consist of processors and local memories. A general SoC design process is shown in Figure 1.1. In SoC, individual cores and their *User-Defined Logic Blocks* (UDL) are designed. Then composite manufacturing and testing are conducted for the whole SoC. The main objective of testing *Very Large Scale Integration* (VLSI) chips is to guarantee that the circuits are fault free and meet the desired specification. The fabrication process includes photo lithography, printing, etching, and doping. Due to the environment and various real-time parameters, the fabricating processes are not perfect and unwanted imperfections can abort the IC operation. The advances in VLSI technology lead to an increase in the circuit's complexity, making testing more difficult. Various challenges emerge in testing the SoC. Usually, a rigorous process is required to test the SoC thoroughly. However, various IP cores are embedded deep inside and not accessible due to limited test pins. *Design-for-Testability* (DFT) [14, 15] strategies recede the accessibility issues and enhance the efficiency of the testing process. We apply input test vectors to the *Circuit Under Test* (CUT) for detecting faults and compare the collected output responses with the expected output responses.

2. **Core Based SoC Testing**

Core-based testing emerge when IC design transitions to the SoC model, in which cores or IPs cores serve as the fundamental components of a design. The conceptual architecture of core-based testing can be better understood by Figure 1.2, where core-based test architecture has three main components [16].

(a) **Test Vectors Source and Sink**

A circuit is faulty when the wrong output signal is generated due to a defect in the circuit. A defect in a circuit can cause a fault in the core, leading to system failure. To test a core with n inputs and m outputs, a set of input test vectors that is nothing but a collection of

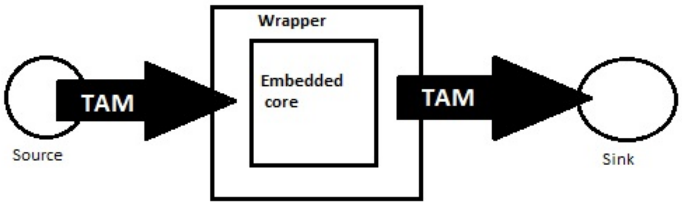


Figure 1.2: *Conceptual Architecture of Core Based Testing* [13]

bits, is applied to core inputs, and the output responses are collected. If the collected output responses differ from the expected output responses, then there is a fault in the core. In order to thoroughly test a circuit, many test vectors are required. The primary function of the source is to generate the test vectors for the core testing. Test vector generation depends on the circuitry inside the core. The sink's job is to collect the core's output responses while applying the test vectors to the input pins and compare the output responses with the expected output responses. Real-time test vector generation is performed in the test source, while the test sink performs real-time response evaluation. Source and sink can either be actualized off-chip *Automatic Test Equipment* (ATE) [17, 18], on-chip *Built-In Self Test* (BIST) [19, 20], On line testing [21–23] or a blend of all.

(b) Boundary Scan Test Standard

Boundary scan, which is also referred to as the IEEE 1149.1 or JTAG standard, is a versatile set of testing methods intended to address a broad spectrum of testing challenges. These challenges encompass everything from the chip level to the system level, encompassing logic cores and the connections between them, and encompassing both digital and analog or mixed-mode circuits. Boundary scan is not limited to ordinary digital designs; it is equally applicable to very high-speed designs. It's worth noting that among the IEEE-approved test standards, Boundary scan, or IEEE 1149.1/JTAG, stands out as one of the most successful, with another test standard, IEEE 1500 , being approved by the IEEE in 2005.

(c) **Core Test Wrapper**

It works as a bridge between embedded cores and the rest of the environment around the cores. Wrapper connects the core's terminal to the TAM and the rest of the IC. The core test wrapper prepares the interface between the core and the system's environment. It avails the facility to switch between two modes. One is normal operation mode, while another is core-internal test mode. In normal operation mode, the core is connected to the system's environment, and the wrapper is transparent. In core-internal test mode, the TAM is connected to the core so that test vectors can be applied to the core's input and analyze the responses at the core's output. Along with these two modes, the wrapper can have various other modes like detach mode to disjoin the core from the system's environment. Similarly, the wrapper has a bypass mode for TAM. The width of TAM does not necessarily match the number of core terminals. The number of core terminals depends on the internal function and core's applications. Hence the width of the TAM is typically decided by the bandwidth of source and sink. If TAM's width is smaller than the number of core terminals, then the wrapper adapts the width by converting serial-to-parallel at core's inputs and parallel-to-serial at the core's output.

(d) **TAM**

TAM transports the test vectors from the source to the embedded cores and output responses to the sink. TAM is utilized as a test data high-way. It spans the physical distance between source and core, as well as between core and sink. There may be several TAM architectures. Even for one SoC, there can be several TAM designs that coexist. The two essential parameters of any TAM are its width and its length. The TAM width alludes to its transport capacity. The TAM's bandwidth prerequisite is to match the data transfer rate needed by the test requirements. The maximum bandwidth is defined by the bandwidth of the source or the sink. TAM's length is defined as the actual physical distance it needs to cover between source and core or core and sink. On-chip sources and sinks may abbreviate the length of a TAM. Sharing a TAM among multiple cores may abbreviate the total TAM length of SoC. The TAM connects IC pins to the core terminals for the off-chip source and sink. TAM with flexible bandwidth is called the test bus. Test buses can be either unidirectional or bidirectional. Multiple cores can share a single test bus.

1.2 Challenges

Testing is an essential phase while manufacturing the SoCs before launching the product. Due to the larger size of SoC and an increase in the number of cores, the SoC testing is itself a vast research area [13], [24], [25]. Inefficient testing can increase the cost of testing and the consumption of resources. We require efficient test methods to make testing efficient, feasible, and cheap.

1. **Deeply Embedded Cores** [26] : TAM is required to efficiently access and test a deeply embedded core inside SoC. It is preferred that the cores to be integrated, have a plug-and-play feature under the TAM to make system integration manageable.

2. **Hierarchical Cores** [27]: There can be a core inside a core in a hierarchical manner. TAM, which is made for the core at the top level of the hierarchy,

will not test all the cores deep inside. Hence, a hierarchical test structure is required to test all the cores.

3. **Higher Performance Core I/Os than SoC Pins** [28] : Cores often have much higher clock rates than what SOC pins can offer. External testers can't typically handle these high speeds, and increasing the test clock rate would make the design overly complex and costly. Therefore, the best approach is to use regular functional units to create an efficient and cost-effective at-speed test environment.

4. **Costly and Ineffective External Automated Testing Equipment (ATE)** [29]: Specifications for ATE, which tests digital, analog, and memory devices, vary greatly. Depending solely on external ATE for SoC testing requires it to handle all test signals, making it very costly. Shifting some testing functions onto the chip can reduce the reliance on external ATE and lower testing expenses.

5. For extensive test data, it is tough to maintain efficient transportation between ATE and SoC for testing.

6. Few testing methods require software programs to perform testing. Storage of these software programs is also a challenging task.

7. **Large TDV** [30]: Nowadays, customers use more applications on their devices so SoC has a large number of cores inside it. Due to this large number of cores, the size of the test vector set is also increased. Due to large test data, it is challenging to store the test vectors. To reduce test data, compression techniques are used. The coder-decoder circuits are implemented for compression and decompression, introducing area overhead.

8. **Large TAT** [31]: Due to more number of cores and enormous test data, testing takes a long time. It is difficult to test the SoC thoroughly. Sometimes testing takes an infeasible amount of time. Efficient methods are

required to reduce TAT. To test the increased number of cores inside SoC, it is required to test all the cores thoroughly. We need to apply a large number of test vectors to all the core's input pins. The test vectors are transported to input pins of cores by utilizing test buses inside the SoC. Various cores share the test buses. Efficient scheduling for the assignment of cores to these buses is required.

9. **Large TP** [32]: All the test vectors are applied to the core's input pins through the scan chain. Test vectors are applied one after another, which introduces bit-flipping in the wrapper cells of scan chains. This bit-flipping is directly responsible for test power consumption. The bit-flipping in scan chains is enhanced due to increased test vectors, ultimately resulting in enormous test power consumption. Efficient TAM design and methods are required to manage this test power consumption.

10. Any product goes through two phases before launching in the market. One is manufacturing, and another is testing. The cost of the product is decided by manufacturing cost and testing cost. Due to a large number of cores and enormous test data, testing becomes costly. Sometimes the cost of testing exceeds the cost of manufacturing the product. Enhanced testing cost makes the product expensive.

1.3 Motivation

In the present scenario, SoC has become a vital part of our lives. SoC typically combines a central *Central Processing Unit* (CPU), I/O ports, memory, on a single chip. Depending on the applications it can have digital, analog, mixed-signal processing units. As different components are integrated on a single chip, SoCs takes less area. Hence, SoCs are very common in the mobile computing and edge computing markets. SoCs are extensively used in embedded systems and the Internet of Things. The SoC is being used in many products like smartphones, tablets, digital cameras, etc. Before launching a product in the market it goes through the testing phase. Customer's expectations from the product are more. They want to use a large number of applications but on the cheap price of the product. For example, people are using smartphones widely where SoC is the backbone. In the modern era every year new models are being launched and customers change the products frequently. Nowadays when everything like paying bills, booking tickets, taking photos, videos can be performed on digital equipment based on SoCs. Several customers are financially weak and can't afford costly equipment, but want to use various applications on the product. They are forced to compromise with the quality of the product. For example, while purchasing smartphones they can compromise with the resolution of the screen, few multimedia applications where quality can be a little bit compromised. Which allows manufacturers to avail product on cheap price in this price-competitive market. Testing plays an important role in the cost factor. An increase in the number of applications in the product leads to an increase in the number of cores in SoC which affect other factors like TDV, TAT and test power consumption. The augmentation leads to hike TAT as well. Testing more number of cores also consumes more testing power.

In this dissertation, a novel method for *Incomplete Testing* of SoCs is proposed, inspired by approximation computing approaches used in various digital circuits [154], [132]. An approximate circuit is a type of electronic circuit that is designed to operate with some tolerance or imprecision in its components, signals, or calculations. Unlike traditional circuits that aim for precise and accurate computations, approximate circuits intentionally allow for a certain level of error or inaccuracy in order to achieve benefits such as reduced power consumption, faster processing speed, or lower hardware complexity.

These circuits are particularly relevant in applications where absolute precision is not critical, and a certain degree of error can be tolerated without significant impact on overall performance. Approximate circuits are commonly explored in the field of low-power design, where energy efficiency is a primary concern, as well as in applications such as machine learning and signal processing, where some level of imprecision may be acceptable.

Approximate computing leverages the fact that in most *Digital Signal Processors* (DSP) applications like image, audio, and video processing, the final output is perceived by human senses, which are insensitive to small deviations [131]. Researchers have devised approximate arithmetic units, including approximate adders [132, 155–157], and multipliers [158–160]. For example, in an image processing application, an image represented by a two-dimensional array with various pixel values is affected by multiplier and adder circuits. Replacing these circuits with approximate versions introduces slight deviations in the array values, often imperceptible to human eyes.

Several techniques have been proposed to alleviate challenges related to TDV, TAT, and TP. Predominantly, compression and compaction-based methods are popular for reducing TDV, with the storage requirements of ATE contingent upon TDV. Both TAT and the transmission bandwidth for applying test data are influenced by TDV, consequently impacting the overall cost of testing SoCs and, in turn, the cost of the final product. While traditional methods are accurate, they often exhaust resources, consume excessive power, and demand considerable time, making them rigorous, sluggish, costly, and sometimes infeasible for larger SoCs.

This dissertation introduces various testing techniques that intentionally introduce a controlled level of inaccuracy, offering advantages in terms of power efficiency, speed, hardware simplicity, and reduced testing costs for SoCs, where absolute precision is not deemed essential. The proposed methodologies to test SoC involve a trade-off between testing quality and parameters such as TDV, TAT, and TP.

1.4 Objectives

This dissertation's primary concern is to develop efficient methods to test the SoC considering various constraints of fault coverage, TAT, and TP. In particular, the objectives of our work is summarized as follows:

1. Development of effective theoretical and practical techniques for incomplete testing of the SoC.
2. To introduce incomplete testing methods that are TAT and TP aware.
3. Development of an alternative efficient model and the implementation of an incomplete testing method based on approximate computing.
4. Extend approximate incomplete testing methods to enhance TAT and TP optimization.
5. To expand incomplete testing for SoC units with bridging faults while taking into account TAT constraints.

1.5 Contributions

As a part of this work, various methods for incomplete testing of SoC are proposed, and an analysis is performed to evaluate the impact of incomplete testing on numerous parameters like TDV, TAT, and TP.

1. **Incomplete Testing of SoC : Heuristic Approach :** Due to more cores and large TDV, it is not feasible to thoroughly test the SoC. Test vectors are applied to the cores by using scan chains having wrapper cells. We proposed a method that searches the least important bits among the test vectors. The least important bits are those bits that affect the fault coverage by the least margin. A *Particle Swarm Optimization* (PSO) based heuristic approach is introduced to make searching efficient and feasible in

a reasonable time. Our method shows a significant reduction in TDV but with a little compromise with the fault coverage.

2. **TAT Aware Incomplete Testing :** In order to minimize TAT, it is necessary to efficiently allocate cores to the test buses. This study introduces a heuristic method for arranging the assignment of cores to the test buses. Additionally, an assessment is conducted to gauge how incomplete testing affects TAT.

3. **TP Aware Incomplete Testing :** For testing purposes, it's essential to employ test vectors on the input pins of the core and observe the resulting output responses. These test vectors are introduced into the cores via various scan chains, which essentially comprise wrapper boundary cells. As the test vectors are successively applied to the core's inputs, each new vector replaces the previous one, causing a change in every bit within the wrapper boundary cell. This alteration, where bits shift from 1 to 0 or 0 to 1, is referred to as"bit flipping," and it has a direct relationship with the power consumption during testing. In the context of incomplete testing, an abbreviated set of test vectors is generated using ATPG. To apply this reduced set of test vectors to the input pins, a modified TAM architecture is proposed in this study, thereby influencing test power consumption.

4. **Incomplete Testing Based on Approximation and its Impact on TP and TAT :** In this work, approximate testing based on approximation-computing is proposed. An ATPG method is proposed for the approximate testing. Further, an analysis is shown to evaluate the impact of approximate testing on TDV, TAT, and TP.

5. **ATPG for Incomplete Testing of SoC having Bridge Faults :** Individual wires and pins are supposed to be stuck at logical '1', '0', and 'X' (don't care bit). These faults are called stuck-at 0 or stuck-at 1 faults. All the above methods were considering these stuck-at faults. The circuits

may consist of faults other than stuck-at faults. One such kind of fault is known as bridging fault. A bridging fault consists of two wires that are connected when they should not be. Depending on the logic circuitry employed, this may result in a wired-OR or wired-AND logic function. If two wires are shorted accidentally, then these bridging faults occur. If there are n number of wires, then n(n-1) bridging faults are possible. In this work, an incomplete testing method is proposed for detecting these bridging faults. Finally, the analysis alludes the reduction in TAT while testing the SoC having bridging faults.

1.6 Organization of the Thesis

The thesis is organized into eight chapters. Chapter 3 through Chapter 7 pertain to the five contributions mentioned above. The summary of the chapters are as following:

- **Chapter 1: Introduction**
  This chapter concludes with challenges of SoC testing and our contributions.

- **Chapter 2: Literature Survey**
  In this chapter a detailed discussion is carried out. Conventional methods for the optimization of TDV, TP and TAT are discussed.

- **Chapter 3: Incomplete Testing of SoC : Heuristic Approach**
  In this chapter, an ATPG method is proposed to perform incomplete testing. A "PSO based LSTB detector" is proposed to detect the least important bits among the test vectors and generate the test vectors for incomplete testing by removing these least important bits from the test vector set. Finally, the experimental results compare TDV for traditional complete testing with TDV for incomplete testing.

- **Chapter 4: TAT Aware Incomplete Testing**

  Moreover, revised mathematical formulas have been developed for TAT computation. In order to reduce TAT to a minimum, an ideal allocation of cores to test buses becomes imperative. An approach based on PSO is suggested for the assignment of cores to test buses. In conclusion, empirical findings indicate a substantial decrease in TAT during incomplete testing.

- **Chapter 5: TP Aware Incomplete Testing**

  In this chapter, we elaborate on a transition metric designed to assess the bit-flipping phenomenon within scan chains. By employing this transition metric, we compute the TP during incomplete testing. A modified TAM architecture is proposed for implementing test vectors on the core's input pins. To conclude, our experimental results indicate a substantial reduction in both TAT during the execution of incomplete testing.

- **Chapter 6: Incomplete Testing Based on Approximation and its Impact on TP and TAT**

  In this chapter, an incomplete testing method is proposed. This method is based on approximate-computing. An algorithm is proposed to generate the test vectors. Further, TAT and test power consumption is calculated and compared with the traditional complete testing approach. Finally, experimental results are shown for TDV, TAT, and TP.

- **Chapter 7: ATPG for Incomplete Testing of SoC having Bridge Faults**

  Each digital circuit's output can be formulated and translated into conjunctive normal form, or *Conjunctive Normal Form* (CNF) (also known as Product of Sums). This CNF formula evaluates to 1 if and only if the values of variables are consistent with the circuit's truth table. For a fault-free circuit, the CNF formula of the circuit needs to be satisfiable. In this chapter, the application of Boolean Satisfiability to circuit modeling is described.

A method is proposed to introduce a bridging fault in the circuit. After introducing a bridging fault in the circuit, a faulty circuit is obtained. A differential circuit is prepared by using the faulty circuit and fault-free circuit. The CNF formula of this differential circuit needs to be satisfiable. A Boolean Satisfiability based ATPG algorithm is proposed, and test vectors are generated for SoC having bridging faults.

- **Chapter 8: Conclusions and Future Perspectives**
  The thesis concludes with this chapter. This chapter deciphers the work in progress, possible extensions, and future work.

Chapter 2

Literature Survey

The evaluation of SoC designs holds a pivotal role in assuring the dependability and functionality of contemporary integrated circuits. With the constant evolution and growing complexity of SoC architectures, the demand for all-encompassing testing techniques becomes increasingly vital. In this section, we present an extensive examination of existing research, methodologies, and advancements in the realm of SoC testing. By scrutinizing the current state of SoC testing, we intend to offer a comprehensive view of the obstacles, strategies, and solutions that researchers have devised to meet the distinct testing demands of SoC designs. This section forms the basis for subsequent sections, allowing us to pinpoint gaps in the current body of knowledge and propose fresh approaches to enhance the efficiency and effectiveness of SoC testing.

The amalgamation of SoC, which encompasses the integration of various cores onto a single chip, has become commonplace in SoC. Recent progress in manufacturing technology has facilitated the creation of devices featuring hundreds of millions of transistors, posing a plethora of novel testing hurdles. Current electronic devices and VLSI gadgets now incorporate millions of transistors and gates, with contemporary processors boasting over a million transistors [13,33,34]. This feat has been attainable due to the continual reduction in feature size, denoting the dimensions of the components. In present deep sub-micron technologies, the feature size has diminished to less than several nanometers [35,36]. This downsizing of the feature size allows devices to operate at elevated frequencies and clock speeds, rendering them sleek and compatible. Nonetheless, it also heightens the likelihood of defects arising in integrated circuits, resulting in defective chips. Consequently, meticulous testing is a critical phase in guaranteeing the integrity and dependability of a system [13, 33, 34].

2.1 Basics of Digital VLSI Testing

Digital VLSI testing refers to the process of verifying and validating the functionality and reliability of digital integrated circuits. With the increasing complexity and miniaturization of VLSI chips, it has become essential to employ effective testing techniques to ensure their proper operation. The main objective of digital VLSI testing is to detect and diagnose manufacturing defects or faults that may be present in the integrated circuits. These defects can occur during the fabrication process and may result in malfunctions or failures of the chips. By identifying and isolating these faults, manufacturers can improve the yield and reliability of their VLSI chips.

VLSI testing involves several fundamental terms and quantitative measures for evaluating the effectiveness of the testing process. These basic terms include:

2.1.1 Fault Coverage

Fault coverage is a measure of the percentage of detected faults or defects in a circuit or system during testing. It indicates how well the testing process is able to identify and capture potential faults [37]:

$$Fault\ coverage = \frac{Number\ of\ detected\ faults}{Total\ number\ of\ faults} \tag{2.1}$$

whereas a *Defect* refers to a flaw or physical flaw that has the potential to result in a fault. A *Fault* represents a defect and mirrors a physical condition that leads to the failure of a circuit to operate as intended. A *Failure* denotes a deviation from the anticipated behavior and performance of a circuit or system, signifying an irreversible state of a component that requires repair to fulfill its intended design objective.

2.1.2 Test Coverage

Test coverage refers to the extent to which the circuit or system is exercised by the test cases. It measures the percentage of specific elements or features of the circuit that have been tested, such as statements, branches, or paths.

2.2 Fault Models

Due to the wide range of defects found in VLSI systems, creating effective tests for actual defects can be challenging. To address this issue, the use of fault models becomes essential for the creation and assessment of test vectors. A reliable fault model typically needs to meet two key criteria: (1) It must faithfully represent the behavior of defects, and (2) it should be computationally efficient when it comes to tasks like fault simulation and test pattern generation. Over the years, numerous fault models have been suggested [38].

2.2.1 Stuck-at Fault Model

The Stuck-at Fault Model is one of the most widely used fault models in digital circuit testing. In this model, it is assumed that a signal within the circuit can become stuck at a specific logic value (either 0 or 1) due to a fault. The fault is typically represented by specifying a specific net or wire within the circuit where the stuck-at fault occurs. The Stuck-at Fault Model assumes that a fault causes a permanent and consistent change in the signal value, regardless of the circuit's inputs or subsequent operations. It is a simple yet effective model for identifying and detecting faults that can result in permanent logic errors or failures in a circuit. To test for stuck-at faults, test patterns are generated that specifically target the detection of these faults. The patterns are designed to activate the nets or wires in the circuit and observe the output responses to identify any discrepancies or deviations from the expected behavior. By applying a comprehensive set of test patterns and observing the responses, the presence of stuck-at faults can be detected and isolated. The Stuck-at Fault Model provides a practical and systematic approach to identify and address potential faults within a digital circuit. It helps ensure the reliability and correctness of the circuit's logic and is widely utilized in the design and testing of digital integrated circuits.

2.2.2 Bridging Fault Model

A short between two elements is commonly termed a bridging fault. These elements can be transistor terminals or connections between transistors and gates. When an element is shorted to power (VDD) or ground (VSS), it is equivalent to the stuck-at fault model. However, in the case where two signal wires are inadvertently shorted, bridging fault models become necessary. In the initial bridging fault model proposal, the logic value of the shorted nets was represented as a logical AND or OR operation of the logic values on the shorted wires. This model is known as the wired-AND/wired-OR bridging fault model. The wired-AND bridging fault implies that the signal net formed by the two shorted lines will assume a logic 0 if either of the shorted lines is transmitting a logic 0, whereas the wired-OR bridging fault implies that the signal net will assume a logic 1 if either of the two lines is transmitting a logic 1. The wired-AND/wired-OR bridging fault model was initially created for bipolar VLSI and does not precisely depict the characteristics of bridging faults commonly encountered in CMOS devices. Consequently, a primary bridging fault model was introduced for CMOS VLSI, where it is assumed that one driver primarily influences the logic value on the two connected nets. A new bridging fault model has been suggested, drawing from the characteristics of resistive shorts observed in certain CMOS VLSI devices [39]. This fault model, known as the dominant-AND/dominant-OR bridging fault, involves a single driver predominantly influencing the logic state of the connected nets, but this influence is limited to a specific logic value.

2.2.3 Delay Fault Model

For a logic circuit to operate without faults, it's essential not only to execute the logic function accurately but also to transmit the correct logic signals within a defined time constraint. A delay fault arises when there is an excessive delay along a path, causing the overall propagation time to exceed the specified limit. The prevalence of delay faults has increased as feature sizes have decreased [40, 41].

Various delay fault models exist. In the gate-delay fault and transition fault models, a delay fault occurs when the duration it takes for a signal transition from the gate input to its output surpasses the designated. The alternative model is the path-delay fault model, which takes into account the total propagation delay along a signal pathway within the CUT, which is essentially the sum of all the gate delays along that path. Consequently, the path-delay fault model proves to be a more pragmatic choice for testing compared to the gate-delay fault (or transition fault) model [42].

2.2.4 Fault Collapse Model

This model represents faults that may propagate and cause multiple errors or failures in the circuit, often due to fault interactions or complex fault effects.

2.2.5 Transition Fault Model

This model focuses on faults that affect the timing and transition behavior of signals. It considers faults that cause a delay or change in the transition of a signal.

2.2.6 Open Fault Model

Open faults occur when a circuit connection is broken or interrupted, resulting in the loss of signal propagation.

2.3 Reviews of VLSI Test Technologies

Figure 2.1 provides a comprehensive depiction of circuit testing procedures [43]. To evaluate a circuit with m inputs and n outputs, a series of test patterns, referred to as test vectors, is employed to assess the CUT, and the resulting output responses are gathered. Each test vector consists of m bits, corresponding to the inputs applied to the circuit. The collected output responses are then compared to the output responses of an error-free circuit, known as the *Golden Response*. If the output responses align with the *Golden Response*, the circuit is deemed faultless. Conversely, if any discrepancies arise, it indicates the presence of faults within the circuit.

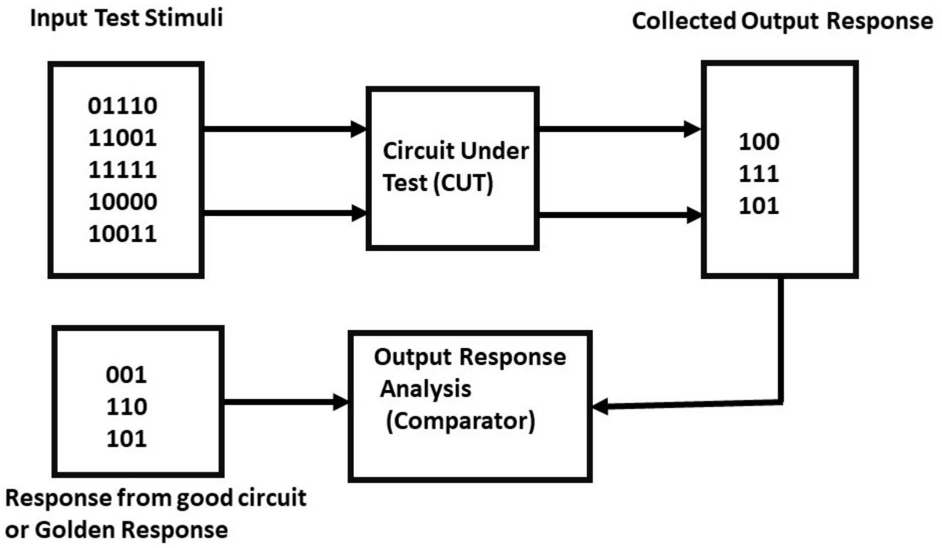


Figure 2.1: *Basic Testing Outlook [44]*

This testing approach is commonly referred to as *exhaustive testing*, where all possible combinations of test patterns are considered. In this context, the number of test patterns possible is maximized at 2^n for n outputs. Ideally, all 2^n test vectors should be applied to the CUT to thoroughly assess its functionality. However, implementing exhaustive testing by applying all 2^n potential input patterns to an n-input circuit becomes impractical, particularly when n is sizable. Despite the comprehensive nature of this testing method, it does not guarantee that all feasible states have been traversed and evaluated [43]. The example discussed, which entails applying all conceivable input test patterns to an n-input combinational logic circuit, highlights the concept of *functional testing*. Functional testing involves testing each entry in the circuit's truth table to ensure that it produces the anticipated responses. However, a more practical and efficient testing approach is *structural testing*. In structural testing, specific test patterns are chosen based on the circuit's structural information and a set of fault models. This approach focuses on targeting particular faults, thereby reducing the total number of required test patterns. Structural testing saves time and enhances test efficiency by eliminating the necessity of testing all potential input combinations. Instead, test vectors are designed to identify specific faults within the circuit. By concentrating on these targeted faults, structural testing streamlines the testing process and enhances the probability of identifying potential issues or defects. By combining circuit structural information with fault models, structural testing provides a more precise and effective approach to validate the accuracy and reliability of a circuit. It enables an efficient allocation of testing resources and enables the identification and rectification of specific faults, thereby improving the overall quality of the circuit.

VLSI testing comprises two procedures: test generation and test application. The objective of test generation is to create test patterns for effective testing, while test application involves implementing these test patterns on the CUT and assessing the resulting output responses. Test application can be carried out using either ATE or on-chip test facilities. A concise overview of the advancement in VLSI test technology is given below.

2.3.1 ATE

ATE are computer-controlled machines used to ICs and *Printed Circuit Board* (PCB) during production. They apply test patterns to components and compare the output to known good responses. ATE in the early 1980s had much higher resolution capabilities than what components needed. For instance, in 1985, when testing an 8-MHz 286 microprocessor, ATE could achieve 1-ns precision in controlling input signal transitions, known as edge placement, with minimal yield loss due to tester tolerances. However, when testing 700-MHz Pentium III microprocessors later on, ATE could only achieve a 100-ps edge placement accuracy. This means that despite a hundredfold increase in the speed of the CUT, the tester's accuracy only increased tenfold.

2.3.2 ATPG

In the 1960s, the stuck-at fault model revolutionized structural testing. The D-algorithm, an early ATPG method [45], and later the PODEM algorithm [46], enhanced ATPG efficiency. ATPG evolved with research and commercial tools like FAN [47]. ISCAS introduced benchmark circuits in 1985 [48], aiding ATPG research globally, followed by sequential benchmarks in 1989 [49].

2.3.3 Fault Simulation

Fault simulators emulate circuit faults to assess test vector effectiveness. Parallel fault simulation utilizes computer bit-parallelism, while concurrent fault simulation operates efficiently with event-driven emulation, commonly used for analog and mixed-signal circuits at the transistor level with tools like HSPICE [50].

2.3.4 DFT

DFT emerged in the 1970s as a way to streamline the construction of test vectors and improve test integration during the design phase. DFT encompasses various techniques, including ad hoc DFT, *Level-Sensitive Scan Design* (LSSD) known as scan design, and BIST [51]. Scan design, proposed in 1977, stands out as a crucial DFT approach [52].

2.3.5 BIST

Around 1980, the concept of BIST was introduced as a way to incorporate a *Test-Pattern Generator* (TPG) and an *Output Response Analyzer* (ORA) directly within VLSI devices, enabling internal testing within the IC itself. Since the test circuitry is embedded within the IC, BIST can be applied at various testing stages, spanning from wafer-level to system-level testing [53], [54].

2.3.6 Boundary Scan Testing

The boundary scan standard for core-based testing was initially proposed by the JTAG and later approved as IEEE Standard 1149.1 [55]. Originally used for testing ICs on a system board, the shift towards SoC. This shift created an analogy between the relationship of chips to the board and cores to the SoC. To facilitate efficient testing of embedded cores and corresponding circuitry in SoCs, a similar testing approach to the boundary scan standard 1149.1 was adopted and named IEEE 1500 [56], [57].

2.4 SoC

SoC refers to an IC that incorporates multiple components and functionalities of an electronic system onto a single chip. It is a complete system that integrates the core processing unit, memory, peripherals, and other necessary components onto a single chip. SoC devices are designed to provide a high level of integration and performance while minimizing power consumption and physical footprint. They are commonly used in various electronic devices, such as smartphones, tablets, smart appliances, automotive systems, and IoT devices.

2.5 Core Based SoC Design

The internal structure of a SoC can vary depending on the specific design and requirements. However, there are common components and structures that are typically found in an SoC.

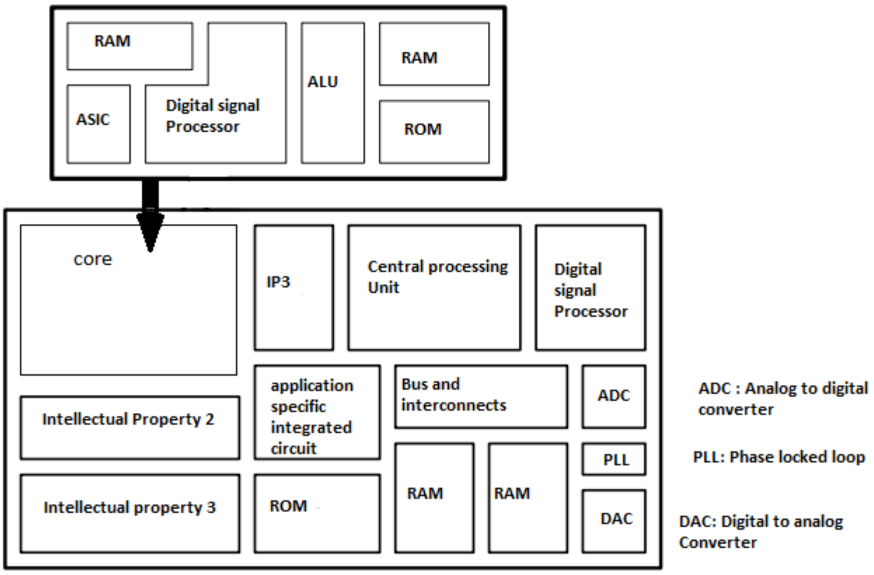


Figure 2.2: *Core Based SoC Design [13]*

Here are some key elements of the internal structure of an example of SoC shown in Figure 2.2:

1. **Processor Cores:** SoCs often include one or more processor cores, such as CPU or specialized cores like GPUs or DSPs. These cores execute instructions and perform computations for various tasks.

2. **Memory Subsystem:** The memory subsystem consists of different types of memory components, including caches, RAM, and non-volatile memory (such as Flash memory). It stores instructions, data, and temporary storage for efficient processing.

3. **Intellectual Property Cores:** IP cores refer to pre-designed and pre-verified components or functional blocks that are licensed for integration into SoC designs. These IP cores are developed by third-party IP vendors

or in-house design teams and provide specific functionalities or subsystems that can be integrated into an SoC. IP cores offer several advantages in SoC design, including faster time-to-market, reduced development costs, and increased design reuse. Instead of designing complex components from scratch, designers can leverage existing IP cores to accelerate the development process and focus on the unique aspects of their SoC design.

4. **Analog and Mixed-Signal Cores:** Analog and mixed-signal IP cores include components like *Analog-to-Digital Converters* (ADC), *Digital to Analog Converters* (DAC),*Phase Locked Loop* (PLL), voltage regulators, and analog front-ends. These cores enable analog and mixed-signal processing and interface with the external world.

5. **DSP Cores:** DSP cores are specialized processors optimized for digital signal processing tasks, such as audio and video processing, image recognition, and telecommunications.

6. **ASICs:** ASICs are integrated circuits that are custom-designed for a specific application or a specific set of functions. Unlike general-purpose microprocessors or programmable logic devices, ASICs are tailored to meet the specific requirements of a particular application, offering optimized performance, power efficiency, and cost-effectiveness.

2.6 SoC Test Architecture

2.6.1 IEEE 1500 std. for SoC Testing

The key aspect of the 1500 standard is its inclusion of a wrapper surrounding the boundary I/O terminals of individual cores. This wrapper ensures standardization of the core's test interface and enables the execution of test commands. Figure 2.3 provides an overview of the IEEE 1500 architecture, illustrating a system with N cores, each enclosed by an IEEE 1500 wrapper. [58]

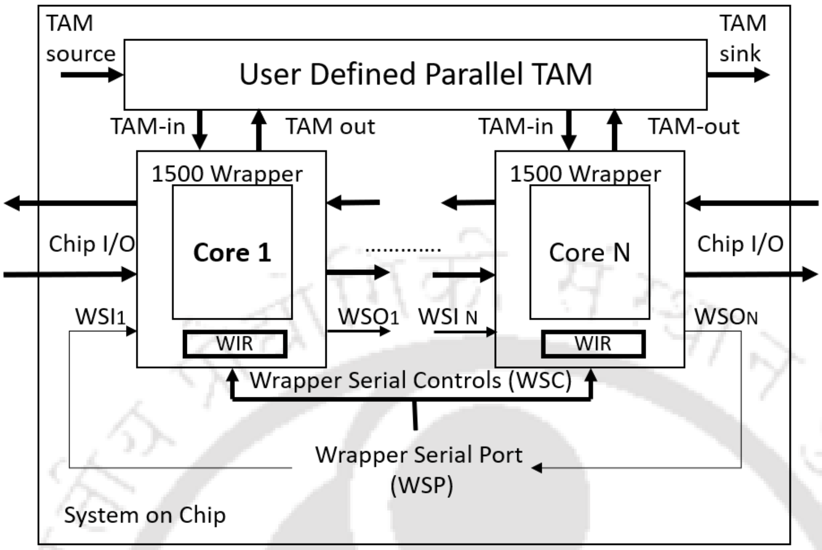


Figure 2.3: *IEEE 1500 Std. Test Architecture*

The IEEE 1500 architecture shown in Figure 2.3 consists of three main components:

2.6.2 Core Wrapper

The core wrapper is an interface between the core and the TAM. It encapsulates the core and provides standardized test and control signals to enable testing and diagnosis. The core wrapper contains several test logic modules, such as *Test Data Input* (TDI), *Test Data Output* (TDO), *Test Mode Select* (TMS), and *Test Clock* (TCK). It also includes a *Test Control Register* (TCR) to manage the test operations [59].

2.6.3 WIR

The *Wrapper Instruction Register* (WIR) is a register within the core wrapper of an embedded core that stores and controls instructions for testing and configuration purposes. It serves as a communication channel between the external TAM and the core's internal logic. The WIR allows the TAM to send instructions to the core wrapper, which are then interpreted and executed to control various aspects

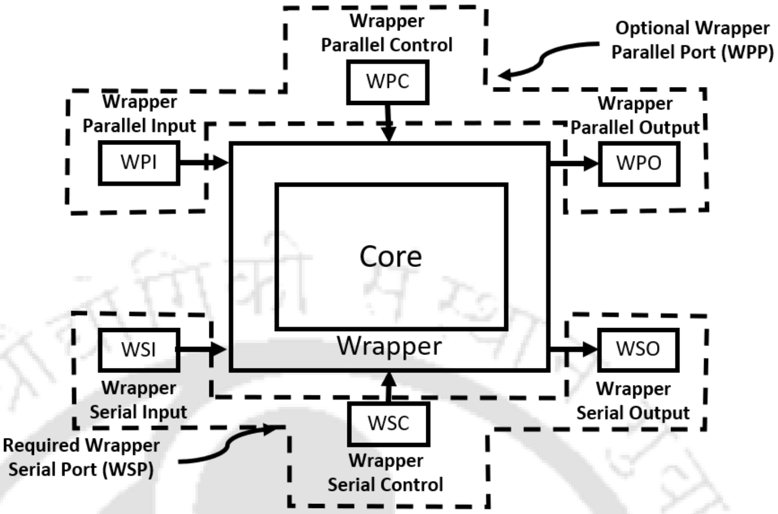


Figure 2.4: *Test Interface of Core Wrapper [59]*

of the testing process, such as setting the test mode, configuring the core, initiating test patterns, and managing test resources. By utilizing the WIR, designers can implement standardized test control and configuration mechanisms, promoting interoperability and enabling efficient testing and configuration of embedded cores within integrated circuits [60].

2.6.4 Test Access Mechanism (TAM)

IEEE 1500 incorporates both serial and parallel TAM, which offer flexible options for accessing and controlling the internal circuitry of the SoC during testing. These TAMs provide standardized interfaces that facilitate efficient testing and seamless integration of various test components. The TAM infrastructure includes a scan path that allows for the transfer of test patterns into and out of the core wrapper, enabling effective test and diagnosis operations. Additionally, the TAM incorporates a *Test Mode Controller* (TMC) responsible for coordinating the test operations and controlling the behavior of the core during the testing process [61].

Test interface of a core wrapper

The test interface of a core wrapper as shown in Figure 2.4 refers to the standardized interface that allows for testing and communication with the core within an IEEE 1500 architecture. The core wrapper acts as a boundary around the core, providing a well-defined interface for accessing and controlling the core during testing operations. The test interface typically includes elements such as test access ports, scan chains, and control signals, which enable the transfer of test patterns, test data, and control commands into and out of the core. By utilizing the test interface of the core wrapper, test engineers can effectively apply test stimuli, collect test responses, and diagnose the behavior and performance of the core during the testing process.

Serial Test Mode

1. ***Wrapper Serial Port* (WSP):** It serves as a communication channel between the external test equipment and the core wrapper, facilitating the transfer of test patterns, commands, and data. The WSP typically consists of *Wrapper Serial Input* (WSI) and *Wrapper Serial Output* (WSO) lines, which allow for the serial shifting of test patterns into and out of the core wrapper.

   (a) **WSI:** It is a dedicated input line within the WSP that allows for the serial reception of test patterns, commands, and data into the core wrapper during the testing process. The WSI serves as an interface through which the external test equipment can send serial data to the core wrapper, enabling the execution of various test operations and configurations.

   (b) **WSO:** It provides a dedicated output line within the WSP for the serial transmission of test results, responses, and other data from the core wrapper to the external test equipment during the testing process. The WSO serves as an interface through which the core wrapper communicates the test outcomes and other relevant information to the test equipment in a serial format

(c) ***Wrapper Serial Control* (WSC):** The WSC consists of control signals or commands that allow the test equipment to instruct the core wrapper. These signals initiate test modes, configure tests, control power modes, enable/disable test features, and perform other operations relevant to SoC testing. The WSC ensures a standardized interface for communication between the test equipment and the core wrapper, facilitating consistent and efficient control of the testing process.

Parallel Test Mode

1. ***Wrapper Parallel Port* (WPP):** The WPP is a standardized parallel interface that facilitates communication between the test equipment and the core wrapper during SoC testing. It enables high-speed transfer of control signals, test patterns, and data, enhancing the efficiency of testing and configuration processes. The WPP plays a vital role in optimizing the overall testing process for the SoC.

    (a) ***Wrapper Parallel Input* (WPI):** It functions as an input interface, enabling the core wrapper to receive control signals, test patterns, and data in parallel from the external test equipment during testing. The WPI ensures efficient transfer of parallel data, allowing for simultaneous and high-speed input of multiple signals or patterns into the core wrapper.

    (b) ***Wrapper Parallel Output* (WPO):** It functions as an output interface that enables the core wrapper to transmit control signals, test patterns, and data in parallel to the external test equipment during testing. The WPO ensures efficient parallel data transfer, enabling simultaneous and high-speed output of multiple signals or patterns from the core wrapper.

(c) ***Wrapper Parallel Control* (WPC):** It is tasked with overseeing and coordinating the control signals that regulate the behavior of the core wrapper throughout the testing phase. Acting as a standardized interface, the WPC enables the parallel transmission of control commands from the external test equipment to the core wrapper.

2.7 Challenges in SoC Testing

In SoC testing, cores from different vendors, such as soft cores, hard cores, or firm cores, may be integrated. To effectively test these cores after integration, it is crucial to have a design and test standard in place. There are various challenges in SoC testing.

1. **Mixing Technologies:** SoCs comprise various cores (e.g., logic, processor, memory, analog) that require post-manufacturing testing. Developing tests for all cores individually is nearly impossible for system integrators. To address this, core providers must offer assistance, and a standardized communication channel between providers and integrators is necessary for effective collaboration and comprehensive testing.

2. **Deeply Embedded Cores:** Deeply embedded cores in a chip necessitate a TAM for efficient access and testing. To facilitate manageable system integration, it is desirable for the integrated cores to have a plug-and-play feature under the TAM, enabling smooth testing and integration processes.

3. **Hierarchical Cores:** Hierarchical cores within a core may require a hierarchical test structure to ensure comprehensive testing. A TAM solely for the top-level cores is insufficient. It is necessary to have an efficient hierarchical test structure that can test cores at lower levels. Additionally, having a plug-and-play feature for cores at any hierarchical level simplifies integration work and enhances flexibility [62].

4. **Different Core Providers:** In SoC integration, cores from different vendors (soft, hard, or firm cores) are utilized. It is crucial for the system integrator to understand how to test these integrated cores. To facilitate seamless integration, the establishment of a design and test standard is essential. This standard ensures consistency and provides guidelines for effectively testing the cores after integration.

5. **IP Protection/Test Reuse:** To address IP protection concerns, core developers often restrict access to detailed internal structural information of a core. Therefore, reusing tests provided by the core developer with minimal or no modifications is desirable. This necessitates the establishment of a standard core test interface and protocol, enabling efficient test reuse and addressing the issue of limited access to internal core information.

6. **Higher Performance Core I/Os than SoC Pins:** When the clock rate inside a core exceeds the capability of the SoC pins, conducting at-speed testing becomes challenging. External testers often cannot provide test clocks that support high-speed testing, even with isolated core access through a TAM. Increasing the test clock rate using a dedicated phase-lock loop would complicate the design and incur high test costs. In such cases, employing normal functional units within the chip to create an at-speed test environment appears to be the most viable approach for achieving efficient, effective, and cost-effective testing.

7. **Expensive and Inefficient ATE:** Testing digital, analog, and memory devices require different specifications from ATE. Relying solely on external ATE for SoC testing would require expensive equipment capable of generating and examining diverse test signals. However, integrating test control and test data generation mechanisms within the chip can reduce reliance on external ATE and potentially lower test costs. This approach allows for more efficient utilization of resources and reduces the overall expenses

associated with SOC testing.

8. **Large Test Data Volume (TDV):** The high demand for test vectors in SoC testing is driven by modern SoCs incorporating numerous components, complex interconnections, advanced designs with multiple power domains and interfaces, and the need to test for different fault models. Testing under real-world scenarios, adhering to strict reliability standards, and accounting for manufacturing variations further increase the number of required test vectors. In summary, the combination of increased integration density, complex designs, diverse fault models, real-world scenarios, high reliability standards, and manufacturing variability all contribute to the need for a substantial number of test vectors in SoC testing.

9. **Long Test Application Time (TAT):** Sequential testing of cores in an SoC results in long test application times, exacerbating the high cost of ATE and potentially causing significant delays in time-to-market, leading to market loss. To mitigate these issues, parallel testing or test scheduling techniques are essential. By conducting tests in parallel or implementing efficient test scheduling, test time can be reduced, minimizing the adverse effects on cost and time-to-market, and ensuring timely product releases with minimal market impact.

10. **Large Test Power Consumption (TP):** While low power design has received significant attention, test power issues have gained recent recognition. Parallel testing is desired to minimize test time, but excessive test power can lead to incorrect results or device damage. Careful planning of the test schedule is necessary to avoid violating power constraints and limits, ensuring accurate testing without compromising the integrity or safety of the devices under test.

2.8 Strategies to Reduce the TDV

Testing SoC designs can be a complex and challenging task due to the increasing complexity and integration of components on a single chip. Reducing TDV in SoC testing is essential to improve efficiency, reduce costs, and enhance the overall testing process. Here are the main strategies to achieve these objectives:

2.8.1 Scan Chain Optimization

Optimize the scan chain architecture to reduce test data volume. Techniques like test data compression within scan chains can lead to significant TDV reduction. Scan chains are a key part of SoC testing. They allow the values of registers in a design to be scanned in and out, which can be used to test the connections between the registers, and can also be used to load test data into the registers. However, scan chains can also increase the TDV. This is because each register in the scan chain must be scanned in and out, even if it is not being tested. [63] There are a number of scan chain optimization techniques that can be used to reduce the test data volume. These techniques include:

1. **Scan Chain Compression:** This technique uses a technique called run-length encoding to compress the scan data. This can significantly reduce the TDV, while still maintaining the ability to test all of the registers in the scan chain.

2. **Scan Chain Reordering:** This technique reorders the registers in the scan chain so that the registers that are not being tested are scanned in and out less often. This can also significantly reduce the test data volume.

3. **Scan Chain Merging:** This technique merges multiple scan chains into a single scan chain. This can reduce the number of scan cycles required to test the design, which can also reduce the test data volume.

These are just a few of the scan chain optimization techniques that can be used to reduce the test data volume in SoC testing. By using these techniques, it is possible to significantly reduce the test data volume, while still maintaining the ability to test the design thoroughly.

2.8.2 Test Compression Techniques

The test data compression method is explained in Figure 2.5. The implementation of test data compression enables the tester to apply a precise and deterministic test set to the CUT while substantially reducing the overall amount of test data needed for the entire testing process. *Test Data Compression* (TDC) and *Test Response Compression* (TRC) techniques are instrumental in minimizing the volume of test data required for testing. By compressing and storing the test data in the tester, these compression methods significantly reduce the TDV. The compressed test data is then transmitted from the tester to the circuit for testing. To achieve this compression, additional on-chip hardware is utilized to decompress the test stimulus, which is subsequently applied to the scan chains on the circuit. Furthermore, this hardware also compresses the collected response at the output, ensuring efficient data transmission. It is essential for the test data compressor to achieve compression without any loss, ensuring that all test bits,

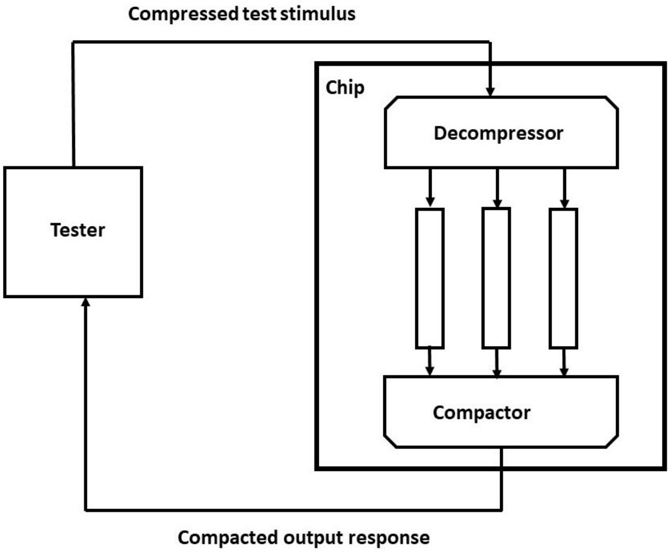


Figure 2.5: *Test Data Compression [64]*

including 0s, 1s, and don't care bits, are accurately reproduced after decompression. This requirement ensures that the compression process does not compromise the fault coverage during testing. Statistical coding is an alternative approach used in test data compression that involves categorizing test vectors into n-bit symbols and assigning variable-length codes based on the frequency of occurrence of each symbol. In [65], a coding-based compression method using selective Huffman coding is proposed to achieve efficient data compression. However, it is important to highlight that one drawback of using Huffman coding is that the decoder size grows exponentially with the size of the symbols, which may impact the overall compression efficiency. Besides statistical coding, various other compression strategies have been explored to reduce the size of test vectors in SoC testing. Constructive coding methods have been introduced in the research of Wang [66] and Reda [67], which aim to optimize the coding process and reduce test data volume. Additionally, compression of test vectors can be accomplished through linear decompression techniques. In this approach, the test vector Y is generated if a solution to a system represented by linear equations AX=Y exists. Several methods involving combinational linear decompressors have been defined in different research studies conducted by Bayraktaroglu [68], Mitra [69], and Krishna [70], providing effective ways to decompress test vectors on-chip. Furthermore, several other methods have been proposed to address the challenge of reducing the size of test vectors. In the works of Wohl [71], Volkerink [72], Krishna [73], Krishna [74], and Lee [75], various innovative techniques are introduced to optimize test vector size while maintaining the required test coverage. These methods have demonstrated to be optimal and effective for achieving significant reductions in the size of test vectors, thereby enhancing the efficiency and performance of SoC testing processes.

2.8.3 Test Data Partitioning

One strategy to decrease the overall TDV in SoC testing involves segmenting the test data into smaller parts and efficiently distributing them. This reduces the amount of test data that needs to be stored, transmitted, and applied during testing. Code-based methods are used to encode the test vectors using compression codes. In these methods, the original data is divided into symbols, which are then replaced with corresponding codes. In a run-length based encoding proposed in [76], 0s are encoded using fixed-length codes, and a cylindrical scan architecture is employed to prioritize runs of 0s, enhancing the efficiency of run-length coding. The authors of [77] discuss a test resource partitioning approach that utilizes run-length encoding-based compression to reduce the TDV. Various methods based on Golomb codes are proposed to encode runs of 0s with variable-length code words [78], [79], [80], [81]. Authors in [78], [79], and [80] present test vector compression techniques and decompression mechanisms based on Golomb codes. In [81], Chandra and Chakrabarty propose a method based on Golomb codes that perform encoding of runs of 0s with variable-length codes. Though these variable-length codes are helpful for encoding longer runs of 0s or 1s, it necessitates an efficient synchronizing system between ATE and CUT. In addition, [82] and [83] discuss selective Huffman coding and complementary Huffman coding-based compression methods. Another compression method, variable prefix dual run-length code, is discussed in [84]. In [85], a test vector reordering method is presented, which effectively increases fault coverage while minimizing the number of test vectors required to cover all faults, thus reducing the test data volume. Further improvements in compression can be achieved by utilizing *Frequency Directed Run Length* (FDR) encoding [86] and Huffman encoding [87], [82], [83]. Another form of compression, dictionary coding, involves partitioning test vectors into n-bit symbols and storing them using a dictionary. While [88] uses n scan chains and modifies test vectors to minimize dictionary size, the drawback of using dictionary coding lies in increased dictionary size, leading to decompressor overhead. To address this drawback, [89] proposes a method for partial dictionary coding, where frequently used test vectors are placed in the dictionary, while the remaining ones remain uncoded. Another method employing partial dictionary coding is discussed in [90].

2.8.4 Compaction Techniques

In addition to compression, compaction is another common technique used to reduce the TDV in SoC testing. Compaction methods utilize the values and positions of bits in test vectors, with test vector reordering and filling of don't care bits being the main features of such methods. An example of test vector reordering can be seen in the work of [85], where test vectors are reordered in a way that increases fault coverage while minimizing the number of required test vectors. This reordering effectively reduces the size of the TDV. Further advancements include the use of X-compactors proposed in [91] and [92]. X-compactors achieve an exponential reduction in output responses, thereby reducing the number of pins needed to collect test responses from the SoC.

2.8.5 Combination of Compression and Compaction Techniques

Some techniques in SoC testing employ a combination of compression and compaction simultaneously on the same test vectors to achieve a reduction in TDV. For example, in the work presented in [93], a run-length coding-based compression is applied after test vector reordering. Additionally, bit-stuffing is performed in a column-wise manner to generate test vectors with a maximum number of zeros. This integrated approach of reordering, bit-stuffing, and run-length coding-based compression results in a substantial reduction in TDV, optimizing the testing process. Various compaction and compression methods are discussed in several research works, including [94], [95], [96], and [97]. In the novel approach introduced in [97], a block matching algorithm is initially executed to segregate low-frequency and high-frequency clusters of test vectors. Low-frequency test vectors represent clusters that repeat less frequently, while high-frequency test vectors represent clusters that appear more frequently. The block matching algorithm effectively reorders the test vector blocks to place related blocks subsequently, enhancing the compaction process. All these compression and compaction methods are well-known and proven to be effective in reducing the TDV. While compressed test data facilitates efficient transportation, it requires the implementation of compression and decompression circuits on the ATE and SoC, respectively. This introduces some area overhead. Nevertheless, the advantages of reduced TDV, such as improved test time and optimized data storage, typically outweigh the overhead introduced by the compression and decompression circuits, making it a worthwhile trade-off for efficient SoC testing.

2.9 Strategies to Reduce the TAT

The test application time is the time it takes to apply a set of test patterns to a SoC. It is a critical factor in the overall SoC testing time. There are a number of strategies that can be used to reduce the TAT.

1. **Hierarchical Scan:** This involves partitioning the SoC into smaller sub-blocks, each of which can be tested independently. This can significantly reduce the overall test application time.

2. **Test Compression:** This involves encoding the test patterns in a more efficient way, which can reduce the amount of data that needs to be transferred to the SoC during testing.

3. **Dynamic Voltage and Frequency Scaling (DVFS):** This involves adjusting the voltage and frequency of the SoC during testing, which can help to reduce the power consumption and, in turn, the test application time.

4. **Parallel Testing:** The process of testing each core individually can be time-consuming and sluggish since it involves handling a large number of

test vectors for the cores. This can lead to testing resources being engaged with the cores more than expected, resulting in significant testing delays. In some cases, these delays can be so substantial that they make testing impractical, making test application time a significant concern. To address this issue, parallel testing mechanisms for cores within the SoC are essential to reduce TAT. This involves testing multiple sub-blocks of the SoC at the same time. This can significantly reduce the overall test application time, but it requires careful planning to ensure that the different sub-blocks do not interfere with each other. Implementing parallel testing requires appropriately distributing the test bus width and employing efficient scheduling to achieve an optimal TAT. Various methods have been proposed, such as the ones discussed in [98], [99], [100], and [101].

5. **ATPG:** This can help to generate test patterns that are more efficient, which can reduce the test application time.

6. **TAM Optimization:** In [100], a technique for optimizing the wrapper and TAM is proposed, considering power constraints during test scheduling. This approach finds a trade-off between TAT and TDV by determining an appropriate TAM width. Although an *Integer Linear Programming* (ILP) model is suggested in [100] for an optimal assignment of cores to test buses to minimize TAT, it may not be efficient enough as it considers test data serialization for each core [102]. To overcome this constraint, [102] proposes a genetic algorithm-based approach for achieving optimal TAM scheduling in SoCs, effectively reducing TAT and TDV. These methods ensure efficiency and fault coverage for SoC testing. Similarly, other methods to reduce TAT and improve testing efficiency have been proposed in [103] and [104].

2.10 Strategies to Reduce TP

Parallel testing of cores can reduce the TAT, but it comes at the cost of increased test power consumption. Hence, there exists a trade-off between TAT and TP. If the test power consumption exceeds a certain threshold, it can lead to incorrect test results and potential damage to the SoC. With modern SoCs being large in size, the TDV has also increased. The larger number of test vectors increases the probability of bit-flipping, which is directly related to test power consumption. Consequently, reducing test power consumption poses a challenging task. It is well-known that a system consumes more power in test mode [105], [106], making test power consumption a critical concern in SoC testing and development. Various terms are used to define TP:

1. **Energy:** During SoC testing, test vectors are applied to the core's input pins through the test architecture, resulting in changes from 0 to 1 or 1 to 0 in the values on core input pins or wrapper cells. The total switching activity generated during test application is proportional to the energy consumed in testing.

2. **Average Power:** Average power refers to the total power distribution over a given period. The average power is obtained by dividing the energy consumed during testing by the test time.

3. **Peak Power:** Peak power represents the maximum power value observed at any instant during testing. It indicates the thermal and electric threshold values of the SoC. If the peak power exceeds this threshold limit, there is no assurance that the SoC will function correctly.

There are several methods available to decrease test power consumption in SoC testing. One such technique is test vector reordering [107], [108], where test vectors are rearranged to minimize the Hamming distance between them. A heuristic approach to address this issue is presented in [109]. Another approach involves reordering the scan cells in the scan chain to reduce switching activities [110], [111]. These scan cell reordering methods offer full fault coverage without requiring external hardware, but they may lead to routing congestion during scan routing. To tackle peak power during test cycles, a scan chain reordering method is proposed in [112].

Some methods focus on reducing test power consumption by filling don't-care bits (X) in test vectors [113]. In [114], the don't-care bits are assigned values of 0 or 1 to minimize bit flipping in scan cells during test vector insertion. The vector filtering technique [115] involves excluding test vectors that do not detect new faults from being applied to the core's input pins. Test power consumption can also be decreased using test vector compaction methods like the one proposed in [116], which merges test vectors to reduce their size without compromising fault coverage.

Modifying the scan architecture is another approach to decrease test power consumption [117], [118]. Power-aware compression techniques, such as the Golomb code based approach in [119] or alternating run-length encoding in [120], are effective in reducing test power consumption. Other methods, like *Linear Feedback Shift Register* (LFSR) reseeding [121], [122], use logical gates to mask certain bits of the test vectors, thus reducing transition probability and power consumption.

Despite the efficiency and fault coverage of traditional methods, they may introduce area overhead, hardware overhead, and increase testing costs. With the growing complexity of modern SoCs, applying these traditional methods may become impractical and costly. To address this, researchers have proposed confidence-aware testing methods.

In summary, to efficiently reduce TDV, TAT, and test power consumption, innovative methods are necessary. The traditional approaches are effective and do not compromise fault coverage, but they can be costly and challenging to implement for large SoCs. Researchers have been exploring confidence-aware testing methods as potential solutions to address these challenges.

2.11 Confidence Aware Testing Methods

Confidence-aware testing revolves around two crucial parameters: *Essential Faults* and *Confidence Level* (*Fault Coverage*). Essential faults are critical and indispensable for the proper functioning of the SoC. To maintain testing cost within an acceptable range, the SoC is tested for essential faults using a minimal number of required test patterns. This approach is referred to as confidence-aware testing. While confidence-aware testing does not cover all faults, the percentage of total faults covered during testing is known as the confidence level. Striking a balance between testing cost and the quality of testing involves testing the SoC with fewer test patterns while ensuring coverage of all essential faults.

In [123], a method is proposed for confidence testing in two ways: one considers faults preferred by the manufacturer, and the other considers faults based on customer requirements. [124] defines the defect level in testing as the percentage of circuits that are defective and launched for customer use after testing, which is concluded to have a probability distribution rather than a single value when testing a circuit using random patterns.

In [125] and [126], the confidence degree of a specified defect level using random test patterns is proposed, and the quality of confidence-aware testing is quantified. Furthermore, confidence-aware testing has been extended to consider other parameters such as test power consumption. In [127], a confidence-based power-aware testing method is introduced, which considers desired fault coverage, detects all essential faults, and adheres to power constraints.

Other incomplete testing methods have employed heuristic approaches like particle swarm optimization and genetic algorithms [128], [129], [130]. These techniques aim to optimize the testing process while still achieving acceptable fault coverage and meeting the required constraints.

2.12 Incomplete Testing

To enhance TDV, TAT, and TP, designers have explored various approaches, including approximate computing techniques. These methods relax the need for precise computation and instead aim for acceptable levels of accuracy, leading to notable improvements in TAT and TP performance without a substantial cost increase. For instance, the approximate *Multiply-Accumulate* (MAC) unit, proposed in [131], is an approximate circuit that utilizes the concept of approximate computing. It employs an approximate hybrid redundant adder for internal multiplication and addition operations, resulting in reduced hardware size and TP with a minor trade off in computation error. Another method, introduced in [132], is the MAC arithmetic architecture, which decreases the number of intermediate partial components generated during arithmetic operations by a factor of 2, leading to enhanced speed and reduced hardware cost. As discussed in [131] and [132], these techniques leverage the idea that certain systems can tolerate output errors to some extent, offering significant benefits in terms of reduced hardware area and TP.

An approximate aware testing method for approximate circuits is proposed in [133] and [134]. It employs a fault classification algorithm that categorizes faults into two groups: approximation-redundant faults and non-approximation faults. However, this method has limitations as ATPG tools prioritize the shortest propagation paths of faults to the primary outputs, leading to incorrect classification of non-approximation faults that propagate to multiple primary outputs as approximation-redundant faults. To address this limitation, another ATPG methodology for approximate circuits is introduced in [135]. This methodology is based on boolean satisfiability and takes into account the quality of the output as well as the difference between non-approximation faults and approximation-redundant faults.

The adoption of approximate testing proves beneficial in various fault-tolerant applications, such as audio, video, graphics, and wireless communications, as mentioned in [136]. For instance, in [137], a mobile device receives streaming H.264 video over a WCDMA wireless channel, and the video compression algorithms take advantage of the temporal correlation of image sequences, allowing parts of the current frame to be borrowed from previously decoded frames stored in the *Decoded Picture Buffer* (DPB). To reduce TP, H.264 uses spatial redundancy and allows some errors to occur at the chip level. Similarly, in [138], a wireless communication application processes data through channel decoders, correcting errors and generating an error-free stream depending on the application. Redundancy is inherent in data stream communication in wireless communication, and error-tolerant data buffering memories within an SoC are crucial for the majority of memory in wireless applications. Various applications, including Multi-Level Cells, STT-RAM [139], neural networks [140], and convolutional neural networks [141], are also suitable for approximate testing at both the circuit and system levels. Utilizing approximate circuits can expedite computation and enhance testing metrics. However, designing such circuits is a demanding and intricate task that requires a comprehensive understanding of the circuit. Additionally, testing these circuits is complex, as specialized techniques are needed to verify that the outcomes meet the requirements of the intended application.

In this thesis, we present a novel testing method designed specifically for SoC architectures. Our approach aims to test the SoC incompletely while still allowing for error tolerance in the output, all without necessitating any modifications to the circuit's behavior. Unlike traditional approximate circuits, which may compromise the accuracy of results, our proposed method maintains the integrity of the circuit's functionality. As the complexity of SoC designs continues to escalate, traditional testing methods have struggled to keep up with the increasing demands for improved testing efficiency. Full testing coverage for such intricate systems is often impractical due to excessive time and resource requirements. Additionally, approximate circuit designs have emerged as a potential solution to reduce test time, but they come at the cost of accuracy, limiting their applicability in safety-critical applications. Our research focuses on developing an innovative incomplete testing method that addresses the limitations of conventional testing approaches and avoids the inherent compromises introduced by approximate circuits. The key characteristics of our method include selectively targeting critical components of the SoC to significantly reduce testing time and resource overhead, incorporating error tolerance into the testing process to allow for minor discrepancies in the output while maintaining overall functionality, and ensuring the method does not require any alterations to the SoC's behavior, preserving the original design's integrity and ensuring the system's robustness. To assess the effectiveness of our proposed testing method, we conducted extensive simulations on various SoC designs with diverse complexities, evaluating metrics such as TDV, TAT, and TP. Furthermore, we analyzed the quality of testing achieved using our method and compared it to traditional full testing approaches and approximate circuit techniques. Our research demonstrates that the incomplete testing method, coupled with error tolerance, offers a compelling alternative to conventional testing techniques and approximate circuits. It significantly reduces TDV, TAT, and TP while still providing satisfactory testing quality, making it an attractive choice for SoC testing, especially in scenarios where comprehensive testing is infeasible or excessively resource-intensive.

Chapter 3

Incomplete Testing of SoC : Heuristic Approach

3.1 Introduction

In the current landscape, SoC designs are under increasing pressure to accommodate a broad spectrum of applications in order to meet the diverse demands of customers. As the number and complexity of applications continue to grow, SoCs are incorporating an expanding array of cores to effectively handle these functionalities. Recent advancements in manufacturing technology have facilitated the integration of a large number of cores into SoCs, presenting both opportunities and challenges.

The incorporation of a substantial number of cores in modern SoCs requires comprehensive testing to ensure their proper functionality. However, the testing process becomes intricate when dealing with a large number of cores, necessitating a significant test vector set with an increased number of test bits. Moreover, the transportation of extensive test data to and from the SoC pins to ATE becomes a critical consideration.

Traditional methods employed for testing these large-scale SoCs with extensive TDV tend to be slow and economically burdensome. Therefore, this chapter introduces an innovative approach to testing large SoCs, utilizing an incomplete testing method. Instead of conducting exhaustive testing on the entire SoC, this approach involves partial testing without compromising the examination of significant faults.

By adopting this approach, the testing process is optimized in terms of TDV. Simultaneously, it ensures that there is no compromise fault coverage for significant faults. The methodology makes a minor trade-off in fault coverage while optimizing the TDV.

3.2 Problem Statement

Given a SoC with K number of cores, wherein each core possesses various inputs and outputs, a Test Vector Set T_c is provided for each core. Consequently, K sets of test vectors are available, with each set comprehensively addressing all faults within the corresponding core during testing. For every core, a list of essential faults is specified as $E_1, E_2, E_3, \ldots, E_n$, and the maximum permissible compromise in fault coverage is given as F_c%. Derive an optimized set of test vectors that facilitates incomplete testing of cores while ensuring coverage of all the essential faults within the respective core.

3.3 Proposed Method

Suppose a given *Core* comprises n input pins, signifying that each test vector consists of n bits, a test vector set (T_c) is provided for this core. The T_c contains n_t test vectors, and each vector is composed of n bits. Therefore, a test vector set is represented as a table of n_t rows and n columns, where n_t is the number of test vectors required to test the core, and n is the number of input pins of the core. Each column position in T_c is corresponding to one of the input pins of the core. The total number of test bits, referred to as *Test Data Volume* (TDV), is determined by the product of the number of test vectors and the number of bits per vector. This TDV is expressed as $n_t \times n$.

Given test vector set (T_c) provides comprehensive coverage of all faults, with *Full Fault Coverage* (F_f%) and a specified maximum allowable compromise in fault coverage as F_c. To achieve this compromise, specific input pins of cores are intentionally left untested, resulting in the removal of corresponding test bits from each test vector in T_c, while ensuring the testing of essential faults $E_1, E_2, E_3, \ldots, E_n$. The primary objective is to determine the maximum number of input pins to be left untested and the corresponding test bits for these pins. This omission, applied to every test vector in the set, aims to achieve a reduction of fault coverage to $(F_f - F_c)\%$. These omitted bits in each test vector are termed *Least Significant Test Bits* (LSTBs), because after removal of these bits from each test vectors, fault coverage will be affected by *least* amount. If n_{lstb} LSTBs are omitted from each test vector, then $n_t \times n_{lstb}$ bits are reduced at the cost of a $F_c\%$ loss in fault coverage. The total number of possible combinations of LSTB positions, where k columns are chosen out of n columns, is given by ${}^{n}P_k = \frac{n!}{(n-k)!}$. In most practical cases of SoC cores, n is of the orders of tens, which implies a huge number of combinations and so, determining the optimal set is computationally prohibitive. Due to the computational complexity of determining the optimal set, an efficient method is required.

3.3.1 Algorithm 1: LSTB Detector

Here a heuristic approach based on *Binary Particle Swarm Optimization* (PSO) is proposed in *Algorithm 1* titled as *"PSO based LSTB Detector"* in this chapter. This method identifies the optimal LSTB positions in a core's test vector set, where LSTBs are bits associated with positions that, when removed, lead to a reduced fault coverage of $(F_f - F_c)\%$. If there are n_t test vectors, each having n bits, the total number of bits in the test set will be $n_t \times n$. Removing n_{lstb} *LSTB*s from each test vector results in a reduction of $n_t \times n_{lstb}$ bits at the cost of a $F_c\%$ loss in fault coverage. This Algorithm can is explained by *Example 3.1.*

ALGORITHM 1: PSO Based LSTBs Detector

```
Input: Set of Completely Specified Test Vectors $T_c$ for a core, Desirable
       fault coverage $F_t$
Output: Set of Completely Specified Test Vectors $T_{lsb}$ having less bit
        length after removal of LSTBs from each test vector
Calculate number of bits (N) in each test vector .
Take population size ($N_P$)= 50
Take maximum iterations ($T_{max}$)=50
$P_{id} = 0.\ P_{gd} = 0.\ V_{id} = 0\ count = 0$
for Each of $N_P$ particle do
    for i:=1 to N do
        $x_i$= Rand(0,1);;        /* Rand(0,1)generate random number
         between 0 and 1 */
        if $x_i$< 0.1 then
            $x_i$=0;
        else
            $x_i$=1
for  j:= 1 to $T_{max}$ do
    Run PSO
    for Each of $N_P$ particles do
        for i:=i to N do
            $V_i(t) = V_i(t-1) + \phi(P_i - x_i) + \phi(P_g - x_i)$ ;   /* Calculating
             velocity of each particle */
            if $(rand() < S(V_i))$ then
                $x_i$=1;       /* Update every bit in each particle */
            else
                $x_i$=0;
        Calculate fitness function of particle;
        Find best particle;
    if Best particle[j]= Best particle[j-1] then
        count=count+1;
    else
        count=0;
    ;  /* if Best particle of current iteration is same as best
     particle of previous iteration then increase the count by
     1 */
    if count=$X_{terminate}$ then
        Modify $T_c$; ;    /* Modify each test vector of $T_c$ according
         to best particle structure.  */
        Exit;
    ;               /* $X_{terminate} = 10$ i.e If Best particle repeated
     continuously 10 times then terminate the program */
Modify $T_c$;
Exit;
```

3.3.2 Complexity of Algorithm 1

The complexity of our algorithm is determined by $O(T_{max} \times N_P)$, where T_{max} represents the maximum number of iterations, and N_P denotes the population size of particles. Both T_{max} and N_P are dependent on the specific case, which corresponds to the specifications of the SoC core and LSTBs used.

3.4 Example

Example 3.1. Suppose in Figure 3.1, consider a core with 12 *Input Pins* and 4 *Output Pins.* The associated test vector set, illustrated in Figure 3.2, comprises 5 rows and 12 columns. The bits in columns $b_{11}, b_{10}, b_9, b_8, b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$ are applied to the core inputs $I_{11}, I_{10}, I_9, I_8, I_7, I_6, I_5, I_4, I_3, I_2, I_1, I_0$. Each column in the test vector set corresponds to the input pins of the SoC.

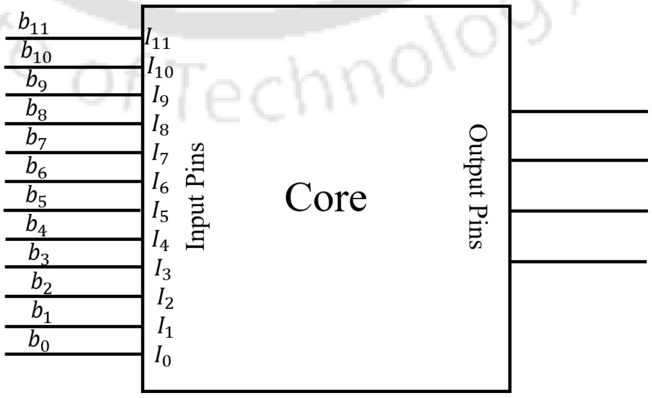


Figure 3.1: *Core with I/O*

b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	1	0	1	1	1
1	0	1	0	1	1	1	1	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1

Figure 3.2: *Test Vector Set (T_c) for a Core*

The test vector set T_c is covering all faults, achieving full fault coverage at 100%. Suppose the maximum allowable compromise in fault coverage F_c is given as 1%, essential faults are specified as E_1, E_2, $n = 12$, $n_t = 5$, and $F_f = 100\%$, now the primary objective is to determine the maximum number of columns in T_c to be removed while maintaining fault coverage at (100 - 1)% = 99%, and concurrently testing for essential faults. To address this, Algorithm 1 "PSO-based LSTB Detector" is utilized as follows.

1. **Binary PSO**

   Binary PSO is a method used for optimizing continuous non-linear functions, initially proposed by James Kennedy and Russell Eberhart in 1995 [142]. It is a variant of PSO designed to operate on binary variables, providing a specialized approach for solving problems with binary representations [143].

2. **Particle Structure**

   Let's consider the test vector set T_c given in *Example* above.

b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	1	0	1	1	1
1	0	1	0	1	1	1	1	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1

Figure 3.3: *Test Vector Set T_c for the Core in Example*

In proposed PSO model, a particle is represented as an N-bit array named P containing binary values, where N is the bit length of a test vector. Each index in the array $P[11], \ldots, P[0]$ corresponds to a columns $b_{11} \ldots b_0$ respectively in the T_c. The *Particle Structure* will be as follows.

P[11]	P[10]	P[9]	P[8]	P[7]	P[6]	P[5]	P[4]	P[3]	P[2]	P[1]	P[0]

This particle will be initially filled with randomly assigned bits, either 0 or 1 as shown below.

P[11]	P[10]	P[9]	P[8]	P[7]	P[6]	P[5]	P[4]	P[3]	P[2]	P[1]	P[0]
0	1	0	0	0	1	0	1	1	0	1	0

If the value at the i-th index of particle $P[i]$ is '1', it indicates that the bits at the i-th column b_i of T_c will remain unchanged. Conversely, if the i-th index of particle $P[i]$ is '0', it signifies that the bits at the i-th column b_i of T_c will be removed from each row of T_c. For example, if there is a '0' at the 5-th index $P[5]$ in the particle, it means that the bits present at column b_5 will be omitted from all the test vectors. Conversely, if there is a '1' at the 1-st index $P[1]$ in our particle, it means that the bits at column b_1 in all the test vectors will remain unchanged.

3. **Initial Population and Position**

   The population size comprises 50 particles. Since we can only tolerate a reduction in fault coverage from 1 to $F_c\%$ (where F_c is typically less than 10), it is more efficient to initialize the search space with a greater number of '1's and '0's. Having more '1's will prevent a significant reduction in fault coverage. In Algorithm 1, lines 5 to 11 handle the initialization of particles. As the process advances, the bit x_i of each particle is updated using the equations of *Binary* PSO [143].

$$V_i(t) = V_i(t-1) + \phi(P_i - x_i) + \phi(P_g - x_i) \tag{3.1}$$

$$x_i = \begin{cases} 1, & if(rand() < S(V_i)) \\ 0, & else \end{cases} \tag{3.2}$$

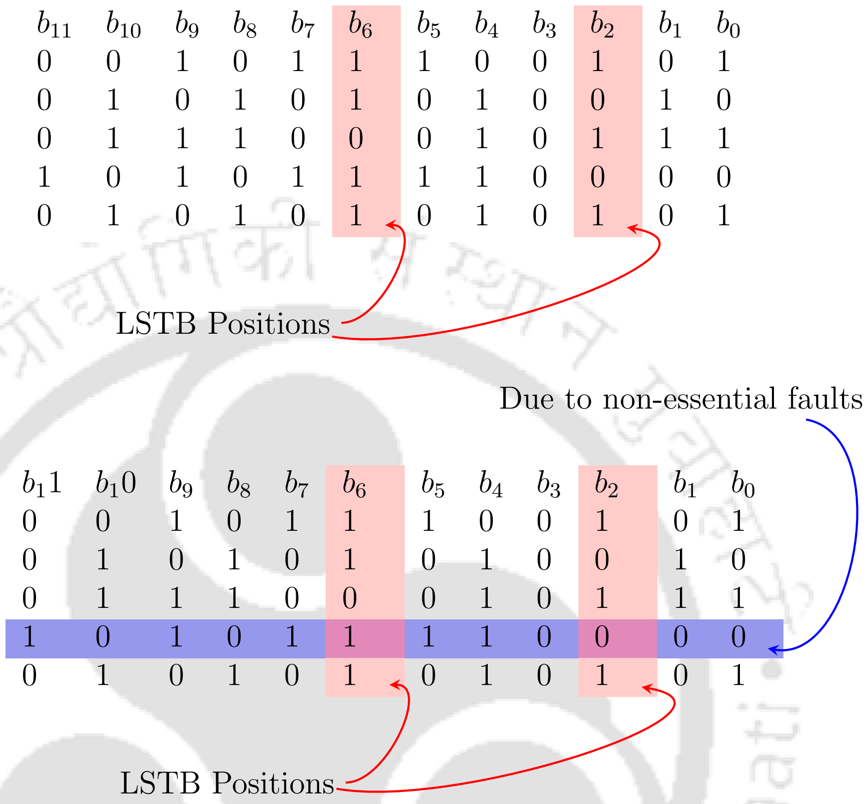
   where V_i is the rate of change of the i-th particle. Each particle i maintains a record of the position of its previous best performance in a vector called P_i. Variable g is assigned the value of the index of the particle with the best performance so far in the neighborhood. ϕ is a random positive number generated for each i. Function $S(V_i)$ is a sigmoid limiting transformation and $rand()$ is a quasi-random number selected from a uniform distribution in [0,1].

4. **Fitness Function of a Particle**

   Our objective is to attain a particular confidence level. For a given *Example*, the maximum allowable reduction in fault coverage is set at 1%. This implies that the fault coverage must not fall below 99%. Consequently, the fitness function of the particle is established as the nearest real number greater than the desired fault coverage, which is 99% in the provided example. Therefore, the fitness function ensures a fault coverage of 99% while guaranteeing the testing of essential faults E1 and E2.

5. **Modified *Test Vector Set* ($T_{Incomplete}$) for Incomplete Testing**

   In Figure 3.4, the modification of the test vector set T_c is evident in both



b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	1	0	1	1	1
1	0	1	0	1	1	1	1	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1

$b_1 1$	$b_1 0$	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	1	0	1	1	1
1	0	1	0	1	1	1	1	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1

Figure 3.4: *Test Vector Modifications*

vertical and horizontal dimensions. Specifically, for column-wise adjustments, as delineated in line 30 of Algorithm 1, subsequent to obtaining the best particle structure, modifications in T_c are executed. If a ‘0’ is located in the i-th position of the best particle, the bits at the corresponding i-th column in each test vector of T_c is omitted. Conversely, if a ‘1’ is found in the i-th position of the best particle, the bits at the i-th column in each test vector of T_c remains unchanged.

In contrast, for row-wise adjustments, compromises in fault coverage result in some faults remaining untested, leading to the omission of certain test vectors intended for testing those faults.

Utilizing the mentioned *“PSO-Based LSTBs Detector”* a compact set of test vectors has been derived for each core. Additional information on these modified test vector set will be elaborated upon in the experimental results.

3.5 Experimental Results

Test patterns for the ISCAS 85 [144] [145] and ISCAS 89 [146] benchmark circuits were produced using the Atalanta-M [147] tool. The computation of fault coverage was executed employing the *HOPE* [148] simulator. Our proposed technique for minimizing test vectors, which is rooted in the concept of PSO, was instantiated through coding in the C++ programming language. Experiments have been conducted for both combinational cores and sequential cores. Two distinct SoCs S1 and S2 configurations are introduced: one exclusively comprises combinational cores, while the other incorporates a mix of sequential and combinational cores. This setup allows us to assess the effectiveness of incomplete testing specifically for sequential cores.

3.5.1 SoC Architecture with Combinational Cores

Figure 3.5 provides a schematic depiction of the SoC architecture, featuring six distinct combinational cores c17, c432, c880, c1355, c2670, and c6288 and incorporating a pair of test buses. In the context of our experimental procedures, we employ benchmark circuits tailored for each individual core. These benchmark circuits are utilized to evaluate the performance of the cores under specific test conditions. *Modified Test Vector Set for Incomplete Testing* ($T_{Incomplete}$), derived from the *LSTB Detector*, are extracted and subsequently transmitted to these cores via the two designated test buses. These test buses serve as the essential conduits for transferring test data to the input pins of the respective cores, enabling comprehensive testing and evaluation.

The ISCAS benchmark circuits for each individual core are utilized as inputs for the ATPG tool, specifically Atalanta-M. This tool is employed to generate test vectors, which are then utilized for both complete and incomplete testing

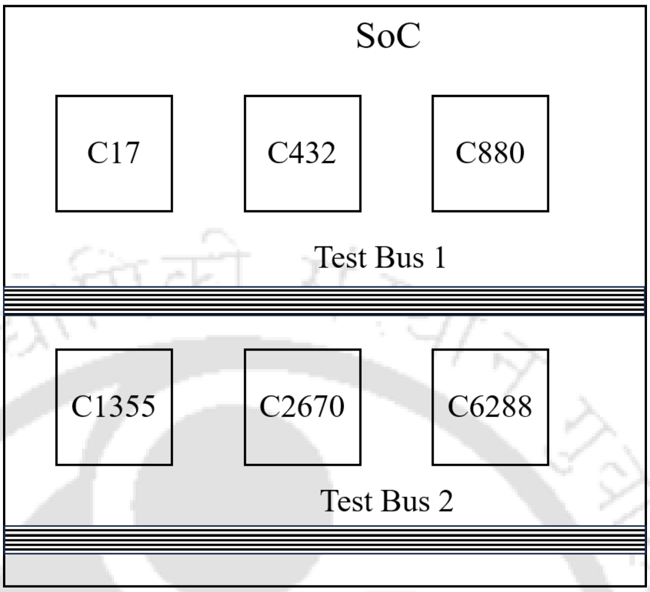


Figure 3.5: *Architecture of SoC S1.*

scenarios. These generated test vectors are subsequently applied to the HOPE simulator to estimate the *Fault Coverage* (FC). In the forthcoming section, Table 3.1 presents a comprehensive analysis of how these procedures impact the TDV and FC for both *Complete and Incomplete Testing.*

3.5.2 Results for TDV and FC

Table 3.1: *TDV and FC comparison for complete and incomplete testing*

Core	FC % for Complete Testing	TDV for Complete Testing	FC % for Incomplete Testing	TDV for Incomplete Testing	Reduction in FC%	% Saving in TDV
C17	100	30	98	25	2	16.67
C432	98.35	1872	96.37	1542	1.98	17.63
C880	95.05	2820	93.06	2312	1.99	18.01
C1355	99.45	3526	97.49	2970	1.96	15.77
C2670	100	23999	98	20436	2	14.85
C6288	97.87	1024	95.89	860	1.98	16.02

In Table 3.1, there are several columns providing key metrics for different processor cores. The column named "Core" lists the core names being analyzed, including C17, C432, C880, C1355, C2670, and C6288. The column named "FC % for Complete Testing" indicates the percentage of fault coverage achieved when employing complete testing strategies for each respective core. It measures how effectively complete testing identifies potential faults within the core. The "TDV for Complete Testing" column shows the test data volume required for comprehensive testing of each core, reflecting the amount of test data needed under complete testing conditions. The "FC % for Incomplete Testing" column displays the fault coverage percentage when using incomplete testing approaches for each core, assessing the effectiveness of these methods. The "TDV for Incomplete Testing" column reveals the Test Data Volume needed for incomplete testing of each core. Additionally, the "Reduction in FC%" column calculates the percentage difference between Fault Coverage achieved with complete and incomplete testing, while the "% Saving in TDV" column calculates the percentage reduction in TDV when using *Incomplete Testing* compared to *Complete Testing*. Each row corresponds to a specific core, presenting these metrics for both testing approaches, enabling comparisons between the two methods. For example, core C17 achieves 100% FC with *Complete Testing* but experiences a 2% reduction when using *Incomplete Testing*, resulting in a 16.67% saving in TDV. Similar interpretations can be applied to the other rows and columns within the table.

3.5.3 SoC Architecture with Sequential Cores

The architecture illustrated in Figure 3.6, referred to as the SoC S2 design, is employed in all of our experiments. This SoC comprises a total of 8 cores, consisting of 3 combinational cores and 5 sequential cores.

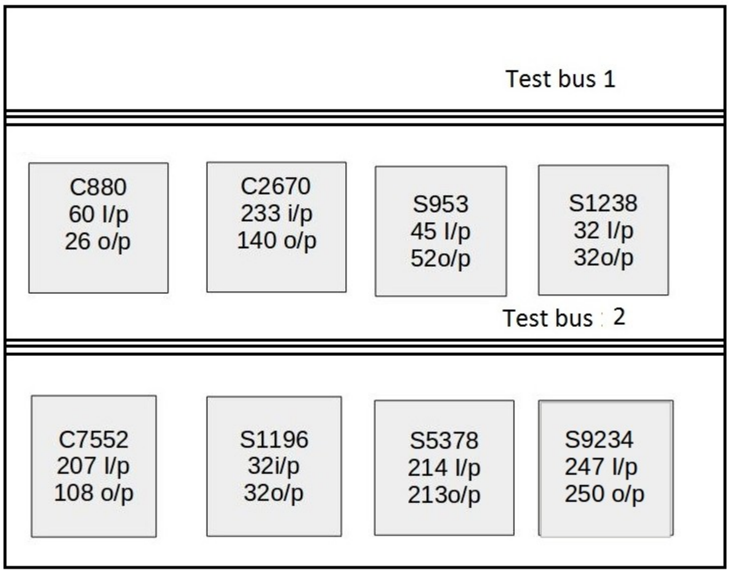


Figure 3.6: *Architecture of SoC S2.*

Table 3.2: *Comparison in Fault coverage with original and modified test vector set*

Cores	Bits	FC %	Bits - LSBs	FC ↓ %	FC ↓ (%)	Bits ↓(%)
c880	60	100.00	53	95.01	4.98	11.67
c2670	233	95.74	206	90.97	4.98	11.59
c7552	207	98.25	188	93.35	4.99	9.18
s953	45	100.00	31	95.09	4.91	31.11
s1196	32	100.00	27	95.01	4.99	15.62
s1238	32	94.91	26	90.19	4.98	18.75
s5378	214	99.12	192	94.18	4.98	10.28
s9234	247	93.48	235	88.81	4.98	4.85
SoC	1070	97.69	958	92.83	4.97	14.13

In Table 3.2, rows 2 through 9 provide the experimental outcomes for the individual cores of the SoC, specifically c880, c2670, c7552, s953, s1196, s1238, s5378, and s9234. Row 10, on the other hand, displays the average experimental results for the entire SoC S2. The first column denotes the names of the respective cores, while columns 2 and 3 detail the number of input bits encompassed by the test vectors for the core c880, in addition to the fault coverage observed prior to the implementation of the proposed testing algorithm. Column 4 reveals the reduction in the number of bits within each test vector for core c880, following the application of Algorithm 1. Concurrently, column 5 exhibits the corresponding fault coverage pertaining to these modified test vectors. It is important to note that a deliberate compromise of approximately 5% in fault coverage is introduced, as depicted in column 6. Lastly, column 7 visually elucidates the reduction in the test bit count for each core as a direct result of our proposed methodology for optimizing test vectors. This comprehensive table encapsulates similar results for various cores of the SoC throughout its entirety. In Row 10, Column 6, we present the average compromise in fault coverage for the entire SoC. This value represents the collective impact of our proposed methodology on fault coverage across all cores of the SoC. Meanwhile, in Row 10, Column 7, we provide the average reduction in the number of bits within the test vectors for the whole SoC. This value signifies the average reduction achieved in test vector size across all cores due to the application of our optimization approach.

Table 3.3: *Reduction in TDV*

Core	No.of Patterns	Bits	TDV	Bits- LSBs	↓ TDV	ΔTDV
c880	54	60	3240	53	2862	378
c2670	103	233	23999	206	21218	2781
c7552	207	207	42849	188	38916	3933
s953	87	45	3915	31	2697	1218
s1196	142	32	4544	27	3834	710
s1238	150	32	4800	26	3900	900
s5378	254	214	54356	192	48768	5588
s9234	357	247	88179	235	83895	4284

In Table 3.3, we have computed the TDV for each core, calculated as the product of the number of patterns and the number of input bits. Columns 3 and 4 show the count of input bits and the corresponding TDV for each core. Similarly, Columns 5 and 6 reveal the reduced input bits and the resulting TDV for each core after applying the proposed test vector minimization technique. The difference between Column 6 and Column 4, labeled as ΔTDV, signifies the total bit savings achieved for each core. For example, in the case of s5378, the total bit savings amount to 5588. This substantial reduction in total bits translates to reduced memory requirements for storing test patterns in ATE and a diminished quantity of bits transferred to the core via the TAM bus. For more clear comparison we have shown variation in TDV in Figure 3.7. where light shaded bar is representing TDV for core without reducing LSTBs, while dark shaded bar is representing TDV when LSTBs has been removed from each pattern.

Table 3.4 presents an analysis of the percentage reduction in fault coverage and the savings in test vector bits achieved at the expense of fault coverage. In row 2, column 3, there is a compromise in fault coverage of 1.99%, accompanied by a 7.28% reduction in test vector bits. Likewise, the remaining rows and columns provide a similar analysis, illustrating the trade-offs between fault coverage compromise and TDV savings.

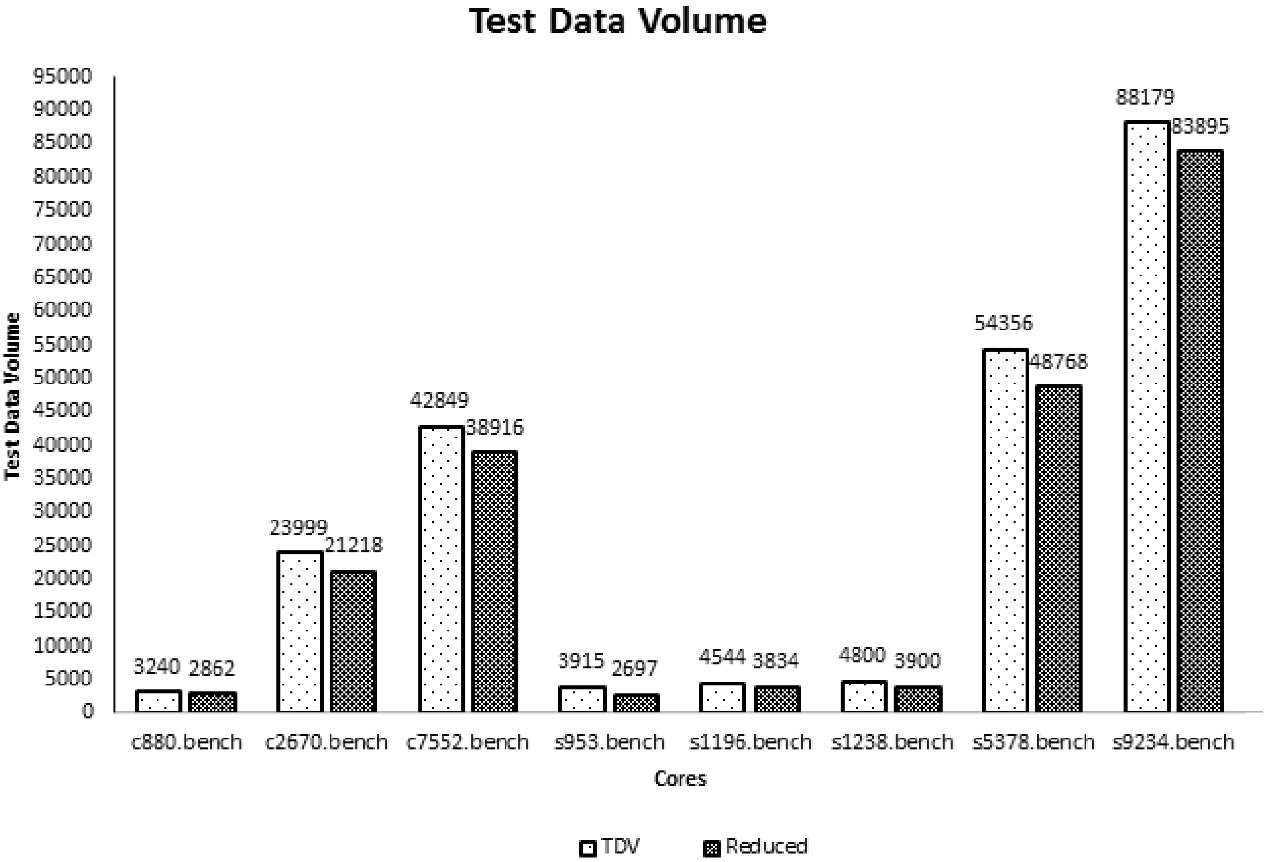


Figure 3.7: *TDV Characteristics*

Table 3.4: *Trade-off between Fault Coverage and TDV*

SNo.	Testing	FC ↓ (%)	Bits ↓(%)
1	SoC	1.99	7.28
2	SoC	3.98	12.15
3	SoC	5.99	18.12
4	SoC	7.96	24.47
5	SoC	9.98	30.12
6	SoC	11.99	36.59
7	SoC	13.99	42.24
8	SoC	15.97	45.33

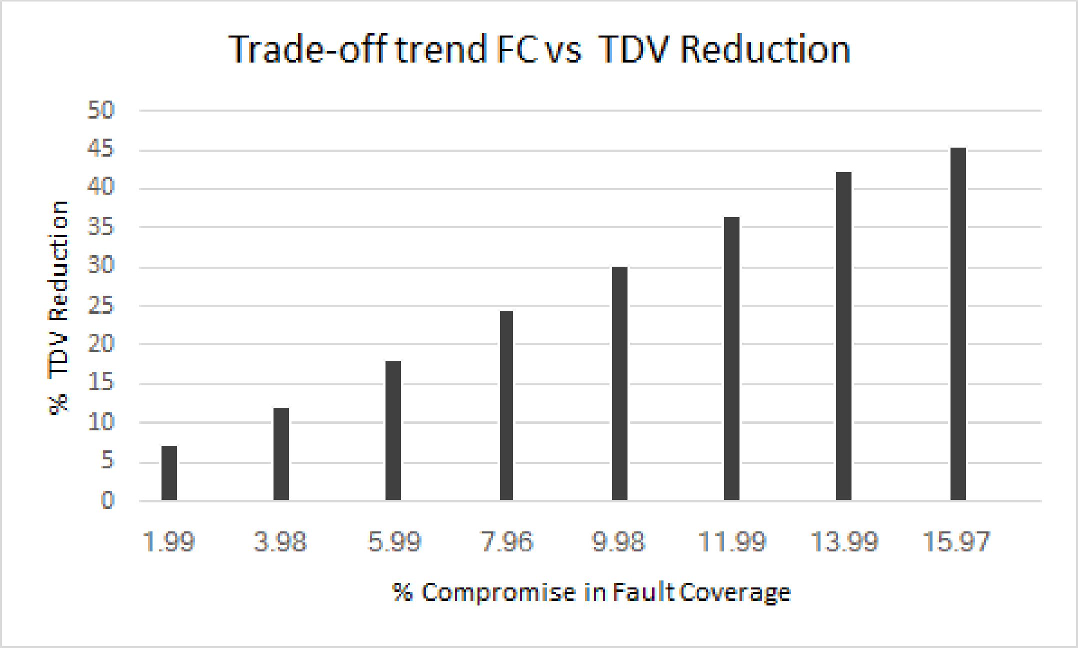


Figure 3.8: *Characteristics in Trade-off between FC and TDV reduction*

Figure 3.8 illustrates the trend in the trade-off relationship between fault coverage and test data volume bits. On the X-axis, we have the degree of compromise in fault coverage, while the Y-axis represents the savings in TDV resulting from this compromise in fault coverage.

3.5.4 Comparison of our Method with Others Methods

To the best of our knowledge, there has not been prior research specifically addressing incomplete testing, which is the central focus of our study. Previous investigations have predominantly relied on compression and compaction techniques to reduce TDV. However, in Table 3.5 and Table 3.6, we have compared our approach with results from compression techniques documented in [149]. It is important to note that while these compression techniques may excel in lowering TDV, they come with significant trade-offs. Implementing compression and decompression circuits or software introduces complexity and additional resource demands into the testing process. Furthermore, the compression and decompression procedures introduce extra time overhead due to encoding and decoding operations, which may not be suitable for time-sensitive applications. In contrast, our proposed method offers a cost-efficient and streamlined solution for TDV reduction without requiring any hardware or software overhead.

Table 3.5: *Comparison for reduction in TDV with other compression techniques*

	(Huffman)	(LZW)	(ACB)	Hybrid test	Reduction in TDV% (Our schme)
c880	63.96	62.63	56.85	69.78	11.67
c2670	62.73	62.29	58.21	66.03	11.59
c7552	56.71	55.83	51.74	59.53	9.18

Table 3.6: *Comparison for reduction in TDV with compression techniques*

	Overhead	Cost	Effective	Time
Compression Techniques	Yes	High	More	High
Incomplete testing (our method)	No	Low	Less	Low

3.6 Conclusion and Future Work

In conclusion, this chapter presented an *Incomplete Testing* technique to minimize TDV and the number of input bits for cores while accepting a certain compromise in FC. The results demonstrate that, with a 5% compromise in FC, the input bits for each core are reduced by an average of approximately 14%. This approach is particularly beneficial for testing larger SoC designs that consist of a significant number of cores, where exhaustive testing becomes impractical. The proposed method offers a cost-effective and efficient solution for incomplete testing of SoCs without requiring any hardware overhead.

In the future, this work can be extended to address the reduction of TAT and the optimization of TP. By incorporating further optimization techniques, it may be possible to achieve not only reduced TDV but also shorter TAT and TP. These advancements would contribute to enhancing the overall testing process and further optimize the testing parameters of complex SoCs.

Chapter 4

TAT Aware Incomplete Testing

4.1 Introduction

In the preceding chapter, a *Heuristic Incomplete Testing Methodology* for SoC is proposed. Our primary objective remains achieving high-quality SoCs while concurrently minimizing testing parameter TDV.

It's worth noting that in addition to TDV, TAT plays a crucial role. When it comes to testing larger SoCs using traditional methods, accuracy is assured, but sometimes the time required exceeds expectations, making it impractical. The test duration for larger SoCs is influenced by several factors, including the growing number of cores in modern SoCs and the transmission of test vectors from external pins to these cores. This transfer of test data occurs either from external pins to the cores or vice versa through test buses. To optimize the speed of test data access, it is essential to efficiently allocate cores to test buses, considering the various possible combinations. Selecting the optimal core and test bus allocation is pivotal for achieving swift test data access.

Conventional methods that take TAT into account have shown accuracy, but they often come with downsides such as excessive stringency, sluggishness, and high costs, particularly when applied to larger SoCs. In this study, we propose an efficient method for core-to-bus allocations, with a specific focus on optimizing TAT. Furthermore, for incomplete testing of SoCs, a set of equations is required to compute TAT, and some adjustments are necessary to adapt these equations to the context of incomplete testing. In summary, this study proposes an efficient method for core-to-bus allocation. Additionally, it presents modified equations to compute TAT for incomplete testing. This approach addresses the challenges of testing advanced SoC designs, particularly in deep sub-micron technology.

4.2 Incomplete Testing

Chapter 3 introduces a method for incomplete testing, presenting a "PSO-based LSTB Detector" designed to identify and sustain the required fault coverage. After detecting these LSTBs, they are excluded from the full test vector set to create a specific test vector set for further incomplete testing.

Within the context of a "PSO-based LSTB Detector", it is possible to incorporate a fitness function pertaining to TAT as shown in Figure 4.1. After the elimination of LSTBs, a modified set of test vectors is obtained, which is intended for incomplete testing. However, it is crucial to note that before TAT can be calculated and applied as the fitness function for incomplete testing, an optimal assignment of the SoC core to the test bus must be established in order to minimize TAT. The subsequent section outlines a method designed to achieve this optimal assignment, followed by subsequent sections that delve into the concept of TAT-aware incomplete testing and proposed modifications in TAT equations resulting savings in TAT.

4.3 Problem Statement

Consider a SoC provided with N_C cores and N_B test buses, each featuring distinct widths denoted as $w_1, w_2, ..., w_{N_B}$. Propose a method for the assignment of cores to test buses in order to optimize the TAT.

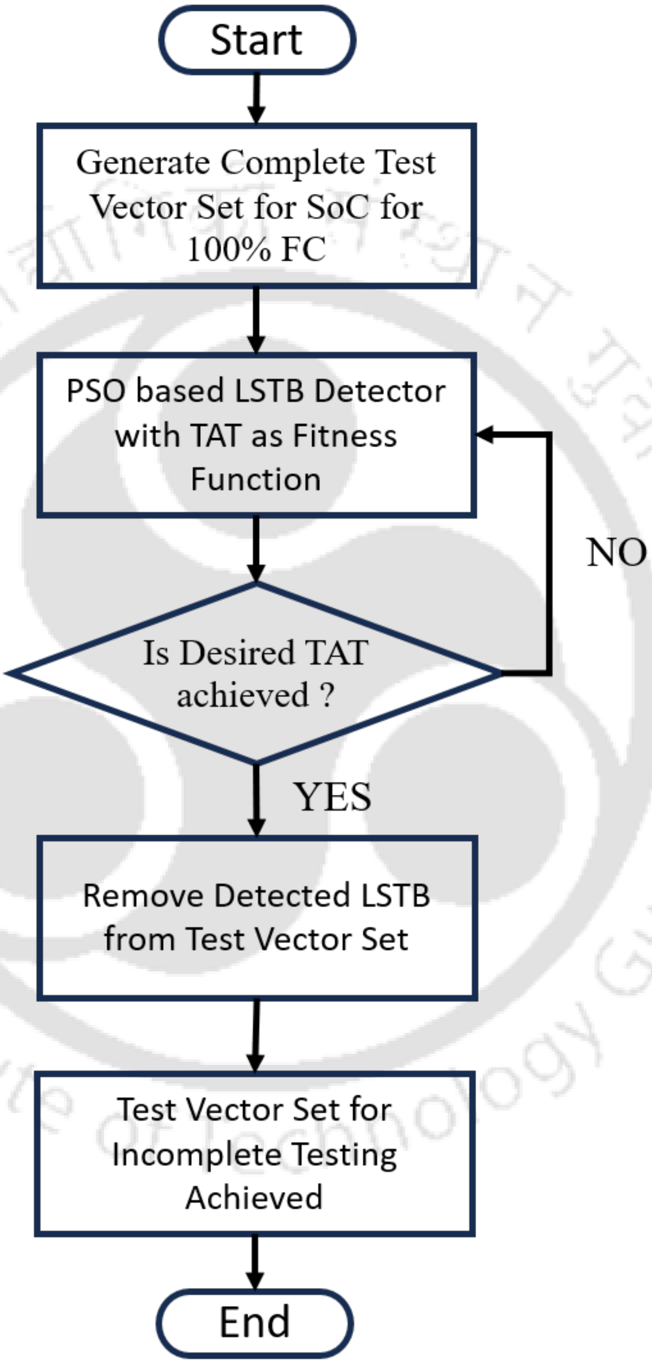


Figure 4.1: *Flow Diagram for TAT Aware Incomplete Testing of SoC*

4.4 Proposed Method

The assignment of cores to test buses plays a crucial role in the TAT calculation. If cores are allocated in such a way that minimizes TAT, then the overall TAT for the entire SoC will also be minimized. Consequently, there is a need for a method that can optimize the distribution of cores to test buses to minimize TAT for each core and the entire SoC.

Various combinations of core-to-test-bus assignments are possible, especially in larger SoCs. Using a brute force approach to search for the ideal core-to-test-bus assignments for all cores and the entire SoC would be time-consuming and impractical some times. So a heuristic-driven technique is proposed here to efficiently allocate the available test buses to the cores. The primary objective is to attain a well-balanced distribution that reduces the TAT while making sure that resources are utilized effectively. This heuristic method factors in the width of each test bus and the computational workload of individual cores, resulting in a practical and efficient strategy for assigning test buses. To assess the effectiveness of this proposed method, an integer PSO algorithm has been developed to compute the TAT for the designated test bus assignment. The integer PSO method systematically explores potential solutions to identify the optimal test bus assignment that minimizes the TAT, taking into consideration test constraints.

The proposed PSO model is given as following.

4.4.1 Particle Structure

In the integer PSO model, a particle is represented as an N_C-bit array, where N_C is the number of cores embedded in the SoC. Each element in the array is filled with an integer value ranging from 1 to N_B, where N_B represents the number of test buses available. It can be understood by example given below.

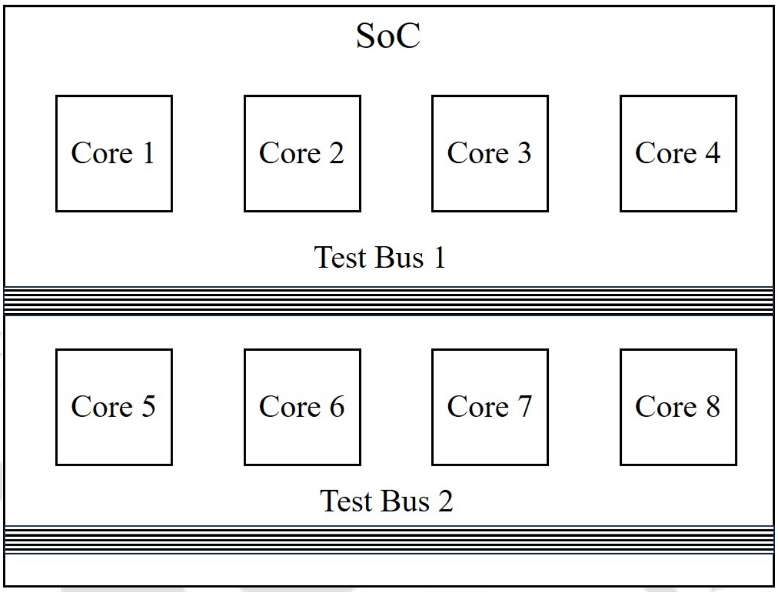


Figure 4.2: *Sample SoC*

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
2	**1**	**1**	**1**	**2**	**2**	**1**	**1**

Figure 4.3: *Particle Structure*

4.4.2 Calculation of TAT

Assume that an SoC is depicted in Figure 4.2. Within this SoC, there are eight cores, specifically labeled as core 1, core 2, core 3, core 4, core 5, core 6, core 7, and core 8, along with two test buses given as Test Bus 1 and Test Bus 2.

In this particular example, the particle arrangement signifies how test buses are distributed among the cores within the SoC. Each rectangular section in the array represents a core, and the number enclosed within each block corresponds to the assigned test buses. In Figure 4.3, the presence of a '2' in the first block p_1 of the array signifies that test bus 2 has been assigned to core 1. Likewise, test bus 2 is designated for cores 1, 5, and 6, while test bus 1 is distributed among cores 2, 3, 4, 7, and 8.

4.4.3 Population Size

In the proposed integer PSO approach, the initial population size is set to 50 for the experiments. The particles in the population are initialized randomly, with each element of the particle having a value between 1 and N_B, where N_B is the number of test buses.

As the PSO process proceeds, the bits x_i of each particle are updated using the following equations based on Kennedy and Eberhart's discrete PSO [143]:

$$\begin{aligned} V_{id} &= V_{id} + \phi(P_{id} - x_i) + \phi(P_{gd} - x_i) \\ &\begin{cases} x_i = 1, & \text{if } (rand() < S(V_{id})) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{4.1}$$

where

V_{id} represents the velocity of the i-th bit of the d-th particle in the population.

ϕ is a random parameter used to control the velocity update.

P_{id} represents the best position (bit value) of the i-th bit for the d-th particle, which is its personal best.

P_{gd} represents the best position (bit value) of the i-th bit among all particles in the population, which is the global best.

x_i denotes the current value of the i-th bit for the particle.

$rand()$ generates a random value between 0 and 1.

$S(V_{id})$ is a sigmoid function that maps the velocity to a probability of updating the bit to 1. If the sigmoid result is greater than a random value between 0 and 1, the bit is set to 1; otherwise, it is set to 0.

These equations control how each particle's bits are updated during the optimization process, aiming to find an optimal allocation of test buses to cores that minimizes the TAT.

4.4.4 Fitness Function

The fitness function assesses the TAT for every core's assignment to a test bus, taking into account the most unfavorable situation of linking test bus lines to core pins. It subsequently totals these TAT values for all the cores. The objective is to reduce the overall TAT by optimizing the allocation of cores to test buses through the PSO optimization procedure.

4.5 Incomplete Testing and Reduction in TAT

TAT is the total time taken to test an SoC. To calculate the TAT for the entire SoC, we first determine the TAT for each individual core based on its assigned test bus. These individual cores TAT values are then aggregated.

4.5.1 Problem Statement

Consider a SoC provided with N_C cores and N_B test buses, each featuring distinct widths denoted as $w_1, w_2, ..., w_{N_B}$. Propose modified equations to compute TAT for incomplete testing and analyze the impact of incomplete testing on minimizing TAT.

4.5.2 Proposed Modified TAT Equations for Incomplete Testing

Considering, the most pessimistic scenario outlined in [100], wherein all the $w_j - 1$ lines within a test bus (j) are linked in parallel to $w_j - 1$ core pins, while the final line in the test bus connects serially to $\psi_i - w_j + 1$ core pins. In this context, ψ_i signifies the greater value between the number of inputs (m_i) and outputs (n_i) for core i.

Standard Equations to Compute TAT

For the computation of TAT (T_{ij}) associated with a core (i) assigned to a test bus (j), the following formulas from references [150] and [102] are employed:

$$T_{ij} = \begin{cases} t_i, & \text{if } (\psi_i \leq w_j) \\ t_i * (\psi_i - w_j + 1), & \text{if } (\psi_i > w_j) \end{cases} \tag{4.2}$$

where

t_i is the test time for core i.

For Combinational Cores:

$$t_i = p_i \tag{4.3}$$

For Sequential Cores :

$$t_i = (p_i + 1) * f_i / N_i + p_i \tag{4.4}$$

where

p_i is the number of test patterns

f_i is the number of clock cycles required for testing core i

N_i is the number of clock cycles between two consecutive patterns for i^{th} core.

Proposed Modified Equations to Compute TAT

It is already defined that ψ_i denotes the greater value between the number of inputs (m_i) and outputs (n_i) for core i.

$$\psi_i = \begin{cases} m_i, & \text{if}(m_i > n_i) \\ n_i, & \text{if}(m_i < n_i) \end{cases} \tag{4.5}$$

In the context of incomplete testing, if we remove "L" number of LSTBs from the input bits of test vectors, the number of input bits will change to $m_i - L$, while the output remains unchanged. The value of ψ_i depends on the values of m_i and n_i If m_i is greater than n_i, then ψ_i is equal to $m_i - L$ and if m_i is less than n_i, then ψ_i is equal to n_i. Formally it can be represented by following equations.

$$\psi_i = \begin{cases} m_i - L, & \text{if}(m_i > n_i) \\ n_i, & \text{if}(m_i < n_i) \end{cases} \tag{4.6}$$

This equation of ψ_i is included in standard TAT Equation 4.2. Hence the final modified equation to compute the TAT will be

$$T_{ij_{incomplete}} = \begin{cases} t_i, & \text{if}(\psi_i \leq w_j) \\ t_i * ((m_i - L) - w_j + 1), & \text{if}(\psi_i > w_j) and (m_i > n_i) \\ t_i * (n_i - w_j + 1), & \text{if}(\psi_i > w_j) and (m_i < n_i) \end{cases} \tag{4.7}$$

where

> $T_{ij_{incomplete}}$ signifies the TAT associated with a collection of test patterns used for partial SoC testing, and this TAT is contingent upon the values of t_i, ψ_i, and w_j.
>
> If $\psi_i \leq w_j$, then $T_{ij_{incomplete}}$ is equal to t_i.
>
> If $\psi_i > w_j$ and $m_i > n_i$, then $T_{ij_{incomplete}}$ is given by $t_i * ((m_i - L) - w_j + 1)$.
>
> If $\psi_i > w_j$ and $m_i < n_i$, then $T_{ij_{incomplete}}$ is given by $t_i * (n_i - w_j + 1)$.

Projected Reduction in TAT for Incomplete Testing

The exact savings in TAT can be determined by calculating the difference between the proposed modified TAT $T_{ij_{incomplete}}$ as illustrated in equation 4.7 and the standard TAT as depicted in equation 4.2. This savings is denoted as (S_{ij}) and can be formally expressed in the following equation 4.8.

$$S_{ij} = \begin{cases} Zero, & \text{if}(\psi_i \leq w_j) \\ t_i * ((L) - w_j + 1), & \text{if}(\psi_i > w_j) and (m_i > n_i) \\ Zero, & \text{if}(\psi_i > w_j) and (m_i < n_i) \end{cases} \tag{4.8}$$

where

S_{ij} depends on the values of t_i, L, and w_j.

If $\psi_i \leq w_j$, the net reduction is “Zero,” meaning there is no reduction in TAT.

If $\psi_i > w_j$ and $m_i > n_i$, the net reduction is given by $t_i * ((L) - w_j + 1)$.

If $\psi_i > w_j$ and $m_i < n_i$, the net reduction is “Zero,” meaning there is no reduction in TAT.

4.6 Experimental Results

In experimental study involved the generation of test patterns for benchmark circuits from ISCAS 85 and ISCAS 89 using the Atalanta-M tool. The assessment of fault coverage was carried out through the utilization of the *HOPE* simulator. Our proposed test vector minimization technique, which is based on PSO, was implemented using the C++ programming language.

The experiments were conducted on three distinct SoCs denoted as S1, S2, and S3. SoC S1 primarily consists of combinational cores, while SoC S2 combines both combinational and sequential cores. SoC S3, on the other hand, serves as a reference for comparing the results with conventional testing methods. This comprehensive approach allowed for the evaluation of the effectiveness of the PSO-based test vector minimization technique across different SoC architectures, providing valuable insights into its performance and potential benefits in comparison to traditional testing methodologies.

4.6.1 SoC with Combinational Cores

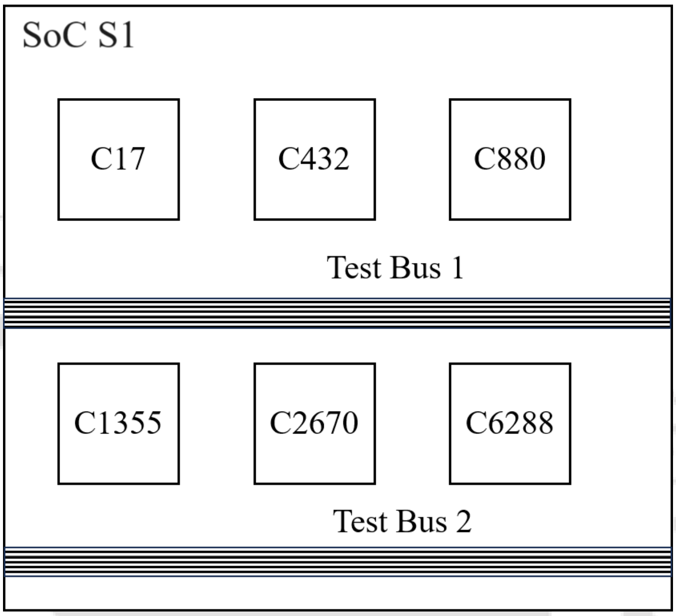


Figure 4.4: *SoC S1*

Table 4.1 presents our findings regarding the scheduling of cores within SoC S1 shown in Figure 4.4, along with the calculated TAT for various combinations of test bus width distributions. During this analysis, a modified test vector set was employed for incomplete testing.

In the second row and first column of Table 4.1, we observe a total bus width of 48. Moving on to the second column (row 2), it illustrates the distribution of bus width between the two cores, denoted as "1,47," signifying that test bus 1 possesses a bus width of 1, while test bus 2 holds a width of 47.

Furthermore, in the third column (row 2), we encounter the optimal assignment of test buses to the cores, represented as "111121." This allocation indicates that test bus 1 is assigned to cores 1, 2, 3, 4, and 6, while test bus 2 is exclusively assigned to core 5.

Lastly, in column 5 of the same row, we find the TAT value for complete testing without the removal of LSTBs. These insights collectively contribute to our comprehensive understanding of the test bus allocation and its impact on the overall testing process for SoC S1.

Table 4.1: *TAT for complete testing*

Bus width	Bus width distribution	Optimal assignment	TAT for complete testing (cycles)
48	1,47	111121	19261
48	4,44	111121	19570
48	8,40	111121	19982
48	13,35	111121	20497
48	16,32	111121	20806
48	20,28	222212	21218
48	24,24	111121	21630

Similarly Table 4.2 represent experimental results for the incomplete testing on SoC S1.

Table 4.2: *TAT for incomplete testing*

Bus width	Bus width distribution	Optimal assignment	TAT for incomplete testing (cycles)
48	1,47	222122	7310
48	4,44	222122	7055
48	8,40	122121	7669
48	13,35	221122	8215
48	16,32	112122	8326
48	20,28	222212	8652
48	24,24	211121	8366

Table 4.3 presents a comparison of results for TAT between complete and incomplete testing, revealing a noteworthy reduction in TAT. In the case of a bus width distribution of (4,44), the savings reach as high as 63.94%. Similarly, for various other distributions, the savings remain substantial, averaging around 60%, which is a significant outcome.

Table 4.3: *Comparison between complete testing and incomplete testing*

Bus width	Bus width	TAT for complete testing (cycles)	TAT for incomplete testing (cycles)	Saving %
48	1,47	19261	7310	62.04766
48	4,44	19570	7055	63.94992
48	8,40	19982	7669	61.62046
48	13,35	20497	8215	59.92096
48	16,32	20806	8326	59.9827
48	20,28	21218	8652	59.2233
48	24,24	21630	8366	61.32224

4.6.2 SoC with Sequential Cores and Combinational Cores

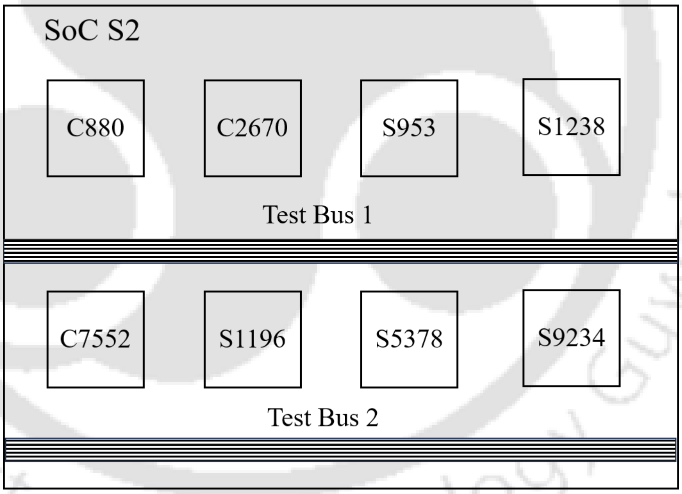


Figure 4.5: *SoC S2*

The experiments are carried out using the SoC denoted as S2, which is depicted in Figure 4.5. SoC S2 is comprised of 3 combinational cores, namely c880, c2670, and c7552, as well as 5 sequential cores, which are s953, s1196, s1238, s5378, and s9234, in addition to two test buses. The number of inputs for the cores c880, c2670, and c7552 are 60, 233, and 207, respectively, while the inputs for the cores s953, s1196, s1238, s5378, and s9234 are 45, 32, 32, 214, and 247, respectively.

Table 4.4: *Optimal Assignment and Reduction in TAT for 5% compromise in FC*

Compromise in Fault Coverage 5 %							
SoC	Bus Width	Dist.	Opt Assignment Complete Testing	TAT Complete Testing (Cycles)	Opt Assignment Incomplete Testing	TAT Incomplete Testing (Cycles)	TAT↓(%)
S1	48	1,47	1 2 2 2 2 2 2 1	96074	2 2 2 2 2 2 2 1	89250	7.1
S1	48	4,44	1 2 2 1 2 2 2 1	97244	2 2 2 2 2 2 2 1	88179	9.32
S1	48	8,40	1 2 2 1 1 2 2 1	99358	2 2 2 2 2 2 2 1	89085	10.34
S1	48	13,35	1 2 1 1 2 2 1 2	97901	1 2 2 1 2 2 2 1	90660	7.4
S1	48	16,32	1 2 1 1 2 1 1 2	99131	1 2 2 1 1 2 2 1	91568	7.63
S1	48	18,30	1 2 1 1 1 1 1 2	99909	1 2 2 1 1 2 2 1	92696	7.22
S1	48	20,28	1 2 1 1 1 1 1 2	100829	1 2 2 1 1 2 2 1	93824	6.95
S1	48	22,26	1 2 1 1 1 1 1 2	101749	1 2 2 1 1 2 2 1	94952	6.68

The allocation of cores to test buses plays a pivotal role in the calculation of TAT. Therefore, the method proposed in Section 4.4 is employed to optimize the distribution of cores across test buses, aiming to minimize the TAT for each core and the entire SoC. Both test buses are allocated to 8 cores in order to minimize the overall TAT for the entire SoC. The primary goal is to achieve a well-balanced distribution that reduces TAT while ensuring efficient resource utilization. The integer Particle Swarm Optimization method systematically explores potential solutions to identify the optimal assignment of test buses that minimizes TAT. These experiments encompass various compromise in FC and test bus widths.

Table 4.4 comprises 8 columns and 10 rows. Row 1 shows the compromise in fault coverage for incomplete testing which is 5% for the first table. Row 2 describes the parameters. In column 2, the total bus width is specified as 48. Column 3 displays the distribution of the total bus width to both test buses. For instance, row 2, column 3 shows that the total bus width of 48 is distributed between the two test buses, with the first test bus having a width of 1 and the second test bus having a width of 47.

Row 3, column 4 delineates the core-to-test bus assignment, denoted as "1 2 2 2 2 2 1" indicating that test bus 1 is assigned to core 1 and core 8, while test bus 2 is assigned to core 2, core 3, core 4, core 5, core 6, and core 7. Column 5 indicates the TAT in cycles, and for row 3, column 5, the TAT is recorded as 96074 cycles.

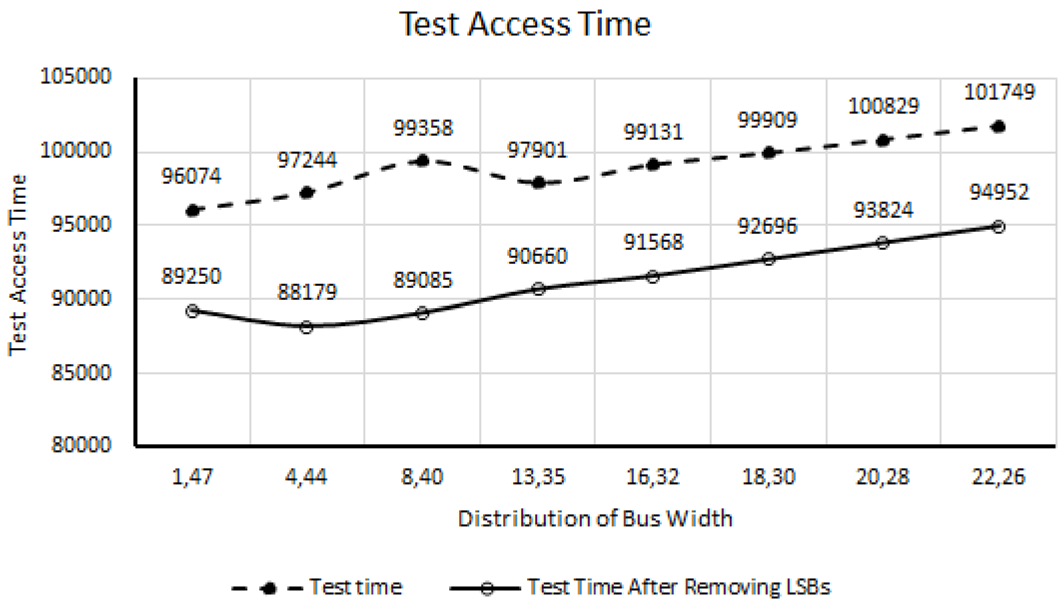


Figure 4.6: *TAT Characteristics*

Similarly, row 3, column 6 displays the optimal assignment for incomplete testing, where LSTBs are removed from the test vectors. Row 3, column 7 reveals the TAT cycles for incomplete testing, which amounts to 89250 cycles, representing a 7.1% reduction compared to the TAT for complete testing of the SoC.

Column 8 shows the percentage savings achieved through our optimal assignment in incomplete testing. When the test bus distribution is set at 8, the maximum saving in TAT is observed at 40, amounting to 10.34%. As depicted in Figure 4.6, it is evident that the TAT has been consistently reduced across various distributions and optimal assignments, leading to a remarkable enhancement in performance. In Figure 4.6, the dotted line represents the TAT curve for SoC S2, where each core is equipped with all the input bits. Conversely, the continuous curve illustrates the TAT curve for SoC S2 when utilizing modified test patterns specifically designed for incomplete testing. Graph in this figure clearly shows the reduction in TAT. The results clearly demonstrate the effectiveness of our approach in achieving reduced TAT values and optimizing the performance of SoC S2.

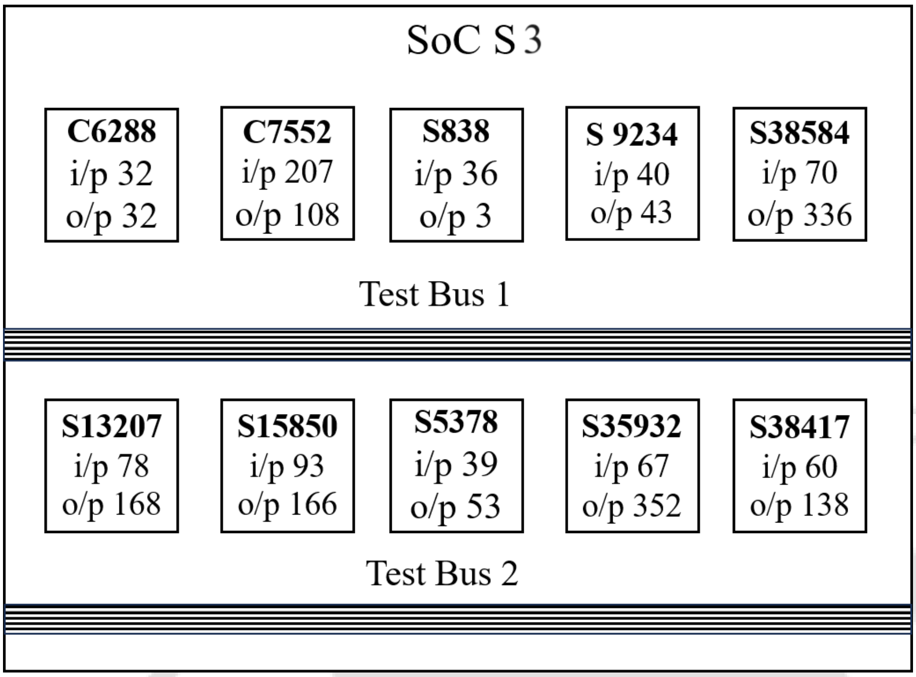


Figure 4.7: *SoC S3*

4.6.3 Comparison with Other Methods

To make a comparison with the conventional methods, experiments are conducted on the SoC S3 as described in [102], as illustrated in Figure 4.7. This comparison involved utilizing all the bits found in the test vectors. The detailed comparative analysis can be found in Table 4.5.

SoC S3 consists of 2 combinational cores, namely c6288 and c7552, along with 6 sequential cores, namely s838, s9234, s38584, s13207, s15850, s5378, s35932, and s38417, in addition to two test buses. Table 4.5 consists of 8 columns and 10 rows. The first row provides information on the fault coverage compromise, which is 5% in the initial table. Row 1 serves as an introduction to the parameters being discussed. In column 2, the total bus width is explicitly stated as 48. Column 3 demonstrates how this total bus width is divided between the two test buses. For example, in row 3, column 3, it is shown that the total bus width of 48 is divided between the two test buses, with the first test bus having a width of 4 and the

Table 4.5: *Comparison between proposed method and other method*

SoC	Width	Distribution	Other Method		Proposed Incomplete Method		Redcution in TAT %
			Op. Assignments	TAT (Cycles)	Op. Assignment	TAT (Cycles)	
S2	48	1,47	2,2,2,2,1,2,2,2,1,2	1966608	2,2,2,2,1,1,2,2,1,2	1780617	9.46
S2	48	4,44	1,1,2,2,1,2,2,2,1,2	2018027	1,1,1,2,1,2,2,2,1,2	1755330	13.02
S2	48	8,40	1,1,1,2,1,2,2,2,1,2	2117313	1,2,1,2,1,2,2,2,1,1	1789716	15.47
S2	48	13,35	1,2,2,2,1,2,2,2,2,1	2116179	2,1,1,1,1,2,2,1,1,1	1795752	15.14
S2	48	16,32	1,1,1,2,1,2,2,2,2,1	2170347	1,2,1,2,1,2,1,2,1,2	1699795	21.68
S2	48	18,30	1,1,2,2,1,2,1,2,2,2	2174091	2,1,1,2,1,2,1,1,1,1	1889754	13.08
S2	48	20,28	2,1,2,1,1,2,2,2,2,1	2204425	1,2,1,1,1,2,1,2,2,2	1876761	14.86
S2	48	22,26	2,1,1,1,1,2,2,2,2,1	2212916	1,1,1,1,1,2,2,1,2,1	1980979	10.48
S2	48	24,24	1,1,1,1,2,1,2,2,1,1	2213412	2,1,1,2,1,1,1,1,1,2	1879809	15.07

second test bus having a width of 44.

Row 3, column 4 outlines the assignment of cores to test buses, represented as “1 1 2 2 1 2 2 2 1 2” indicating that test bus 1 is assigned to cores 1, 2, 5, and 9, while test bus 2 is assigned to cores 3, 4, 6, 7, and 8. Column 5 provides the TAT in cycles, and in row 3, column 5, the TAT is documented as 2018027 cycles.

Likewise, row 3, column 6 illustrates the optimal assignment for incomplete testing, where LSTBs are excluded from the test vectors. Row 3, column 7 displays the TAT cycles for incomplete testing, amounting to 1755330 cycles, representing a 13.02% reduction compared to the TAT for complete testing of the SoC.

Column 8 presents the percentage savings achieved through our optimal assignment in incomplete testing. When the test bus distribution is set to 4, the maximum savings in TAT are observed at 44, constituting a 21.68% reduction.

4.7 Conclusion and Future Work

In this chapter, we carried out an examination of the TAT in the context of incomplete testing. Our research findings revealed that by accepting a 5% compromise on fault coverage, we were able to achieve a reduction of 10% in TAT. Subsequently, we compared these results with the outcomes of existing methods for large SoC designs, where we also observed a significant reduction in TAT.

The balance between maintaining fault coverage and realizing resource savings proves to be advantageous, particularly in the realm of testing larger and more complex SoCs.

Chapter 5

TP Aware Incomplete Testing

5.1 Introduction

Chapter 3 introduces a *Heuristic Incomplete Testing Methodology* for optimizing TDV, followed by Chapter 4 where *Incomplete Testing* is extended to optimize TAT and assign cores to test buses, aiming to minimize TAT. Further modifications to TAT equations are made for *Incomplete Testing* of SoCs. Traditional methods that consider TP and TAT constraints are known for their accuracy but often suffer from excessive stringency, sluggishness, and high costs, especially when applied to larger SoCs.

This chapter extends the *Incomplete Testing* technique to minimize *Test Power Consumption* (TP). Additionally, a customized TAM architecture is introduced to support the incomplete testing approach, playing a crucial role in facilitating access to various SoC components for testing purposes. Optimizing the TAM design is crucial for enabling efficient and effective *Incomplete Testing* while concurrently reducing TP.

In summary, this chapter offers a fresh perspective on SoC testing, with a specific focus on TP. By combining incomplete testing with a well-optimized TAM architecture, it presents a practical and efficient solution for testing modern, densely integrated SoCs. This proposed approach holds significant potential in addressing the challenges associated with testing advanced SoC designs, particularly in deep sub-micron technology.

5.2 Incomplete Testing

Chapter 3 introduces a method for incomplete testing, presenting a "PSO-based LSTB Detector" aimed at identifying and preserving the necessary fault coverage. These detected LSTBs are then removed from the complete *Test Vector Set*, resulting in a specific test vector set tailored for subsequent *Incomplete Testing*. The utilization of a "PSO-based LSTB Detector" allows for the incorporation of a fitness function related to TP, as illustrated in Figure 5.1. Once the LSTBs are excluded, a modified set of test vectors is obtained for the purpose of incomplete testing. It's important to note that the calculation and application of TP as the fitness function for incomplete testing require certain equations, which are detailed in the subsequent section.

Furthermore, the chapter proposes a TAM architecture to support incomplete testing and achieve savings in terms of TP. This integrated approach offers a systematic and efficient way to identify and address test vectors with limited fault coverage, ultimately contributing to more streamlined testing procedures in the context of the PSO-based LSTB detector and TAM architecture.

5.3 Calculation of TP

The concept of assessing TP can be more clearly illustrated through the example presented in Figure 5.2. In this example, the procedure involves the sequential application of test vectors (0, 1, 0) and (1, 1, 0), resulting in a total of 5 transitions, which are represented as circles. These transitions occur when there are changes in signal values between consecutive elements in the test vectors. Following the insertion of the test vector (1, 1, 0), the wrapper cells w1, w2, and w3 assume the values 1, 1, and 0, respectively. These wrapper cells act as an interface between the core terminals and the test input/output. Subsequently,

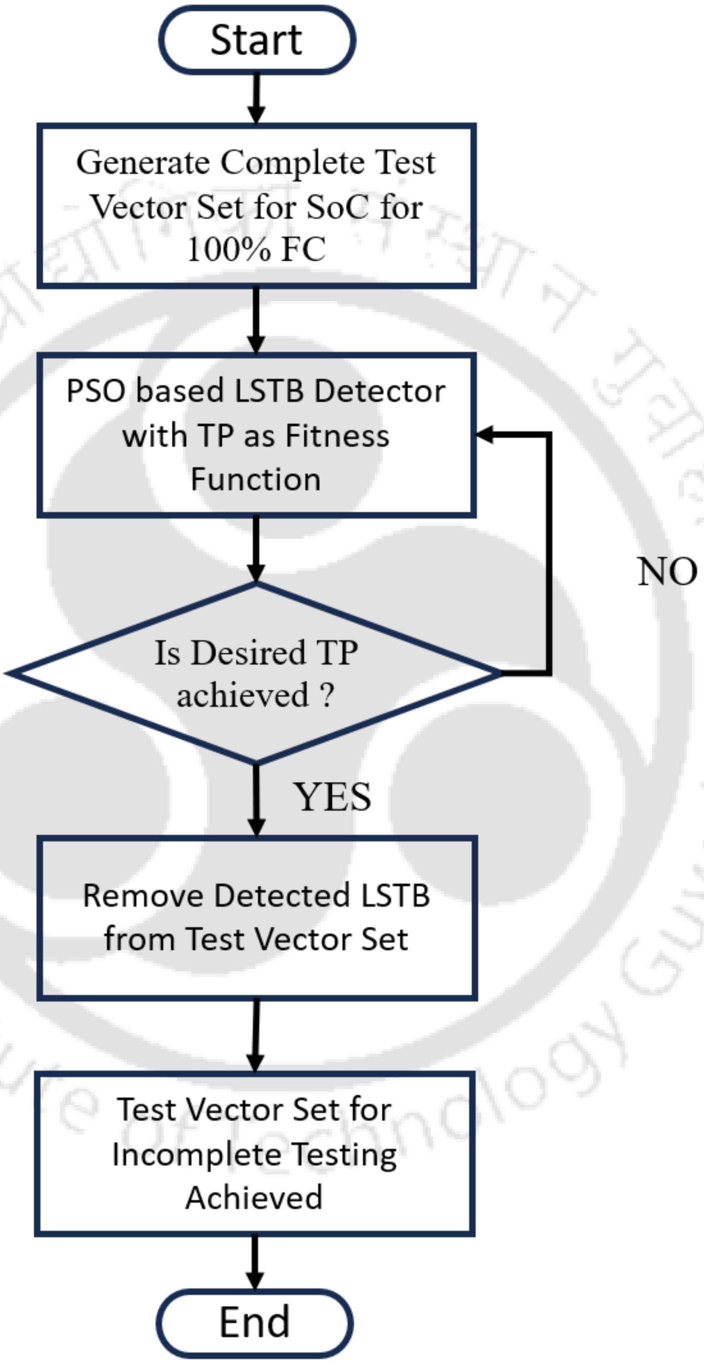


Figure 5.1: *Flow Diagram for Incomplete Testing of SoC*

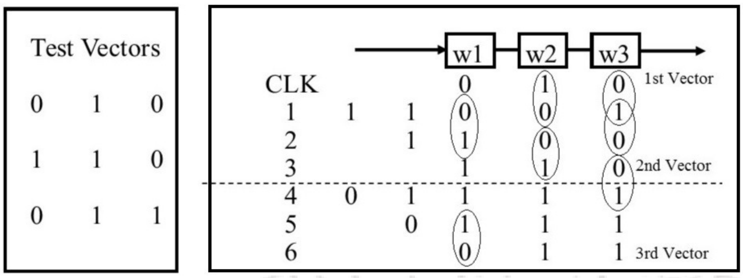


Figure 5.2: *Computation of transitions in wrapper cells* [151]

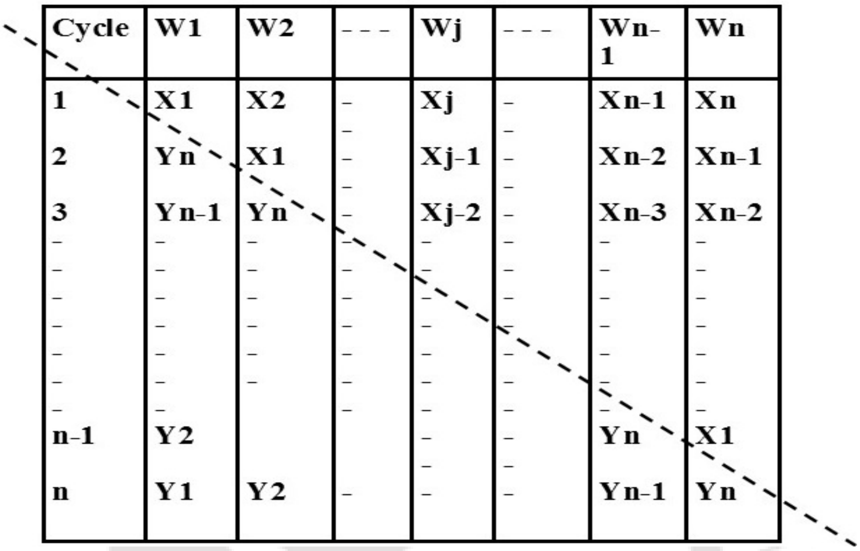
when the next test vector (0, 1, 1) is shifted into the wrapper chain, it leads to 2 additional transitions. This entire process demonstrates how transitions and the values of wrapper cells are computed based on the specific test vectors employed.

5.3.1 Standard Equations to Compute TP

This study primarily emphasizes the input wrapper chain, with a specific exclusion of the internal scan flip-flops and output wrapper cells. The test vectors are applied to the input wrapper chain using various modes, including parallel, sequential, or a combination of both parallel and serial modes.

Parallel Transition:

For the bits that are applied in parallel to the wrapper chain, the bit flips are directly calculated by comparing the change in previously stored values in flip-flops after shifting the next test vector. This means that the new values $(y_1, y_2, y_3, ..., y_n)$ of the test vector are applied to the existing values $(x_1, x_2, x_3, ..., x_n)$ of the wrapper chain in parallel. The bit flips are simply the result of the EX-OR operation between the old and new values of each corresponding bit.



Cycle	W1	W2	- - -	Wj	- - -	Wn-1	Wn
1	X1	X2	-	Xj	-	Xn-1	Xn
2	Yn	X1	-	Xj-1	-	Xn-2	Xn-1
3	Yn-1	Yn	-	Xj-2	-	Xn-3	Xn-2
-	-	-	-	-	-	-	-
n-1	Y2			-	-	Yn	X1
n	Y1	Y2	-	-	-	Yn-1	Yn

Figure 5.3: *Transitions while shifting test vectors* [151]

Serial Transition:

To calculate transitions for the bits that are shifted serially into the wrapper chain, the transition matrix equations, as detailed in the reference [151], are employed. These equations consider the changes in bit values between test vectors during the serial shifting process. The transition matrix equations offer a systematic method for determining the number of bit flips that occur when test vectors are serially shifted.

The equations required for computing TP are derived in the work referenced in [151]. Let's consider an SoC with a wrapper chain containing initial values filled by the input test vector $X_1, X_2, \ldots, X_n$, and the next vector to be shifted in serially is represented as $Y_1, Y_2, \ldots, Y_n$, as illustrated in Figure 5.3. It is assumed that the number of wrapper cells (flip-flops) in the wrapper chain is the same as the number of bits to be shifted. To determine the number of transitions, an $n \times n$

transition matrix, denoted as T(X, Y), is defined according to the reference [151].

$$T(X,Y) = \begin{pmatrix} t_{11} & ... & t_{1n} \\ : & & : \\ t_{n1} & ... & t_{nn} \end{pmatrix} \tag{5.1}$$

where

t_{ij}= 1 if transition occurs in j^{th} wrapper cell in i^{th} clock cycle.

$t_{ij} = 0$ Otherwise.

.

Finally total number of transitions (tr_{total}) can be calculated by summing all the components of transition matrix and is shown in equation 4.10 as follows.

$$tr_{total} = \sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij} \tag{5.2}$$

where

$t_{11} = X_1 \oplus Y_n$ which shows that diagonal elements in Figure 4.5 are identical.

$t_{1j} = X_{j-1} \oplus X_j \qquad if(1 < j \leq n)$

$t_{i1} = Y_{n-i+2} \oplus Y_{n-i+1} \qquad if(1 < i \leq n)$

$t_{ij} = t_{(i-1)(j-1)} \qquad if(1 < i, j \leq n)$ This shows that

Symbol $\oplus$ represents the EX-OR operation, which is applied to elements in the test vectors when calculating transitions.

In summary, when applying test vectors in parallel, bit flips are directly calculated based on the EX-OR between old and new values. For test vectors shifted serially, the transitions are calculated using the transition matrix equation 4.10 to account for the changes in signal values during the shifting process.

5.4 Problem Statement

Considering an SoC equipped with N_C *cores and* N_B *test buses, each having different widths represented by* $w_1, w_2, ..., w_{N_B}$*. Propose a TAM and assess how incomplete testing affects TP coverage.*

5.5 Proposed TAM Architecture

The 1500 standard addresses Design for Testability (DFT) and communication challenges in core-based testing. It enables reuse of DFT and patterns, integrating DFT logic, and improving communication between core providers and users. It uses TAMs for data transfer and adapts bandwidth with core wrappers. The standard employs IEEE Std. 1450.6 for communication, accommodating various DFT strategies. The IEEE 1500 group aims to optimize core-based testing at the SoC level, supporting diverse testing needs with standardized core wrapper components and *Core Test Language* (CTL) requirements.

5.5.1 The IEEE 1500 Wrapper Architecture

A wrapper is DFT logic added to a core for interfacing with the SoC-level TAM after core integration. It adapts bandwidth between TAM and core, isolates the core and surrounding logic, and supports various test data and control mechanisms for different core testing needs.There are 1500 wrapper hardware components supporting various functions for embedded core testing. Figure 5.4 illustrates these components.

The WIR plays a crucial role in initiating various test operations within the 1500 wrapper. It accomplishes this by configuring the wrapped core into the specific mode required for each test. The type of test to be performed is determined by an instruction loaded into the WIR, which is then decoded by pre-designed circuitry. This decoding process generates various control signals that set up the wrapper components for the intended test.

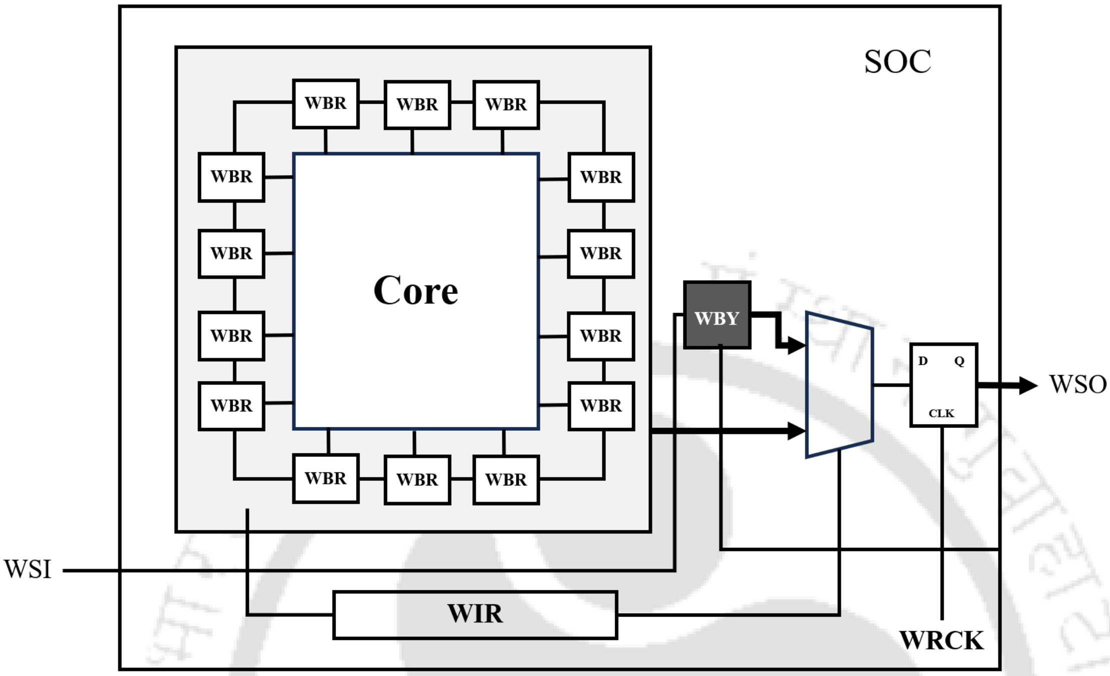


Figure 5.4: *IEEE 1500 wrapper core design* [152]

The *Wrapper Boundary Register* (WBR), on the other hand, serves as a data register within the wrapper, facilitating access to the core's input and output terminals. The WBR is essentially a collection of 1500 compliant wrapper cells, with each cell corresponding to a unidirectional core terminal.

It's important to note that the 1500 architecture allows for the creation of user-defined wrapper data registers in addition to the WBR, without replacing it.

The plug-and-play nature of the 1500 wrapper simplifies the process of connecting multiple 1500 wrapped cores at the SoC level, making it a convenient and efficient solution.

As a result of these features and capabilities, the 1500 wrapper provides a robust and versatile framework for testing and integrating multiple cores within an SoC.

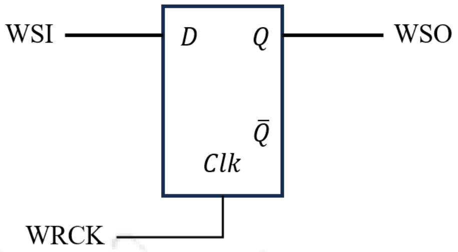


Figure 5.5: *WBY internal structure*
[152]

5.5.2 WBY

WBY is a critical element of the 1500 wrapper, designed specifically for bypassing other WDRs. It plays a vital role within the 1500 wrapper architecture. When connecting multiple 1500 wrapped cores in a serial manner, it is possible that the test data intended for a particular core may have to pass through one or more other cores before reaching its destination. If we did not have the *Wrapper Bypass Register* (WBY), this would mean that the data would have to be sequentially shifted through every WBR chain that lies along the path from the SoC inputs to the target wrapper. In such a situation, the WBR would be the only mandatory data register within the wrapper. This could result in a bottleneck and longer test times directly proportional to the number of WBR chains that need to be traversed. The purpose of the WBY is to offer a shorter path through each 1500 wrapper, thus avoiding the need to unnecessarily navigate through lengthy data registers.

The WSI is connected to the D port of the example WBY, while WRCK is connected to the CLK port. The output of this connection must feed into WSO. These three WSP signal connections are depicted in Figure 5.5.

The 1500 wrapper clock (WRCK) serves as a specialized terminal within the WSP, offering clock functionality to all 1500 wrapper elements. This dedicated wrapper clock enables the simultaneous execution of test wrapper tasks, like the preload operation, alongside the core's functional operations.

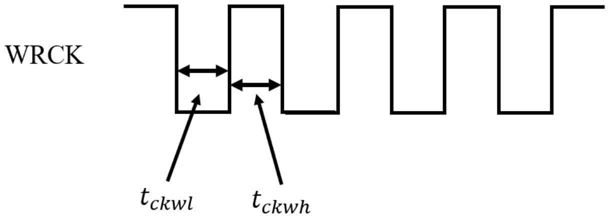


Figure 5.6: *WRCK waveform*
[152]

The 1500 Wrapper Clock (WRCK) operates as a specialized terminal within the WSP, providing clock functionality to all components within the 1500 wrapper. This dedicated clock allows for the concurrent execution of test wrapper operations, such as the preload operation, alongside the core's functional operations. Figure 5.6 illustrates the timing parameters related to the WRCK signal, including t_{ckwh} (high pulse width of WRCK) and t_{ckwl} (low pulse width of WRCK).

5.5.3 WS_BYPASS Instruction

The 1500 standard comprises three mandatory instructions, one of which is WS_BYPASS. This instruction is essential and serves two distinct purposes. Firstly, it allows the core to function in its normal or functional mode by bypassing the WBR. During this instruction, the path from WSI to WSO through WBY is typically not used. Secondly, WS_BYPASS is employed to bypass the WBR, reducing the length of the *Wrapper Data Register* (WDR) during a test mode.

WS_BYPASS is considered the default instruction and is automatically set in the WIR during the core's reset using WRSTN. When WRSTN is disabled, all signals transmitted from the WIR to the core, WBR, and WBY must transition the core and WBR into functional mode while routing the WBY into the path from WSI to WSO.

5.5.4 Proposed TAM for Incomplete Testing

Based on the preceding discussion, it can be inferred that the WBY operates in two distinct modes:

1. **Bypass Mode:** In this mode, the WBY allows the WS_BYPASS instruction to pass through the WIR, permitting input data to flow transparently without any alterations. Essentially, it acts as a direct conduit for data transfer within the chain.

2. **Storage Mode:** Conversely, the WBYs have the capability to store input data and facilitate its movement within the chain. This mode enables serial data shifting, enabling controlled data movement throughout the chain.

During the process of incomplete testing for the SoC, specific bits are intentionally excluded, and particular input pins of the cores are left untested, in accordance with the customized testing requirements established by application-specific users. Each core may undergo varying degrees of incomplete testing, resulting in the inability to remove the WBR attached to untested pins. To address this requirement, we propose the implementation of a TAM architecture that enables programming instructions to either disable a WBR for the corresponding pin or allow data to bypass that input pin. To realize this functionality, we propose a modification in IEEE 1500 TAM where the WBRs associated with the input pins within the core's wrapper will be replaced with WBYs, as illustrated in Figure 5.7. The operation and control of these WBY elements are facilitated through the utilization of the WS_BYPASS instruction specified in IEEE 1500, in conjunction with the WRCK clock signal. A specific program within the context of the Test CTL can be formulated to cater to distinct test prerequisites, enabling the precise manipulation and control of the WBYs in accordance with the specified requirements. This concept can be elucidated further through the following illustrative example in Figure 5.10.

In Figure 5.10 there is one core with 14 input pins and a test bus with 4 bus

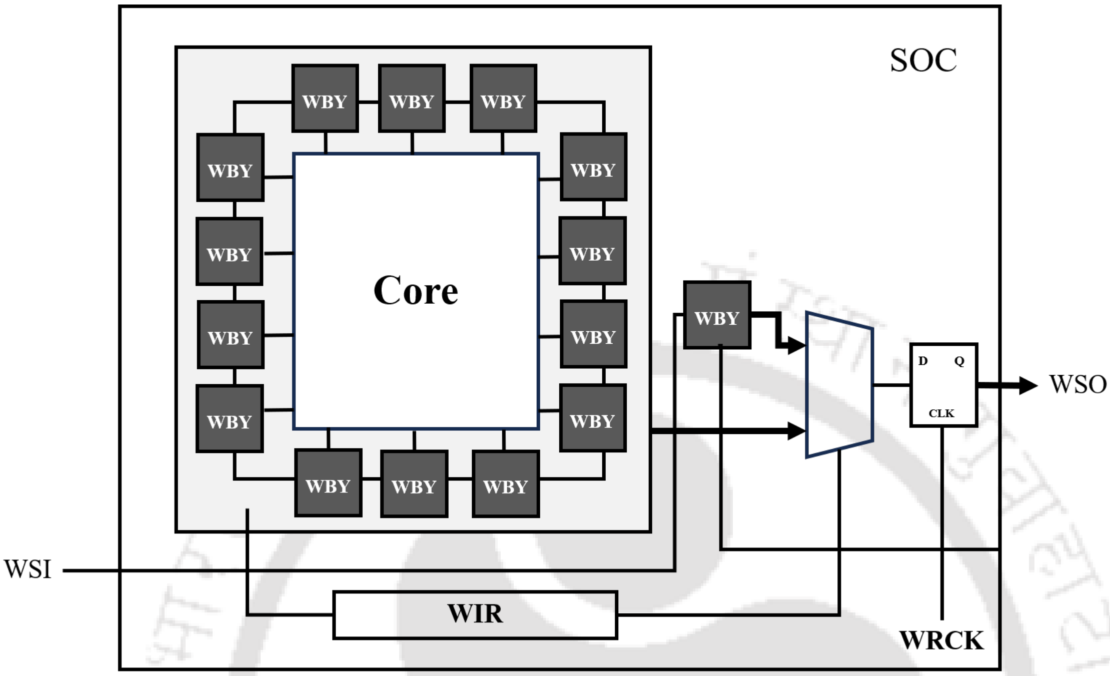


Figure 5.7: *IEEE 1500 Design for Incomplete Testing*

lines. Consider a pessimistic scenario, all but one of the bus lines are connected in parallel to the input terminals of the core. so out of four lines 3 bus lines are connected in parallel to input pins P1, P2 and P3 and one bus line is connected serially to all the other input pins P4 to P14. However, the last bus line is connected serially to the remaining input terminals of the core. This configuration for the proposed TAM can be visually represented in Figure 5.6. In this illustration, a core features 14 input pins, which are connected to WBYs $W_0, W_1, \ldots, W_{14}$. Additionally, a test bus is depicted, and this wrapper chain allows the test bus access to the input pins of the SoC.

Consider the scenario where a test vector denoted as $b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{13}, b_{14}$ is being applied to the input pins of cores P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, and P14 through the use of WBRs W1, W2, W3, W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, and W14, as depicted in Figure 5.8.

In the proposed TAM model tailored for incomplete testing, all the WBRs

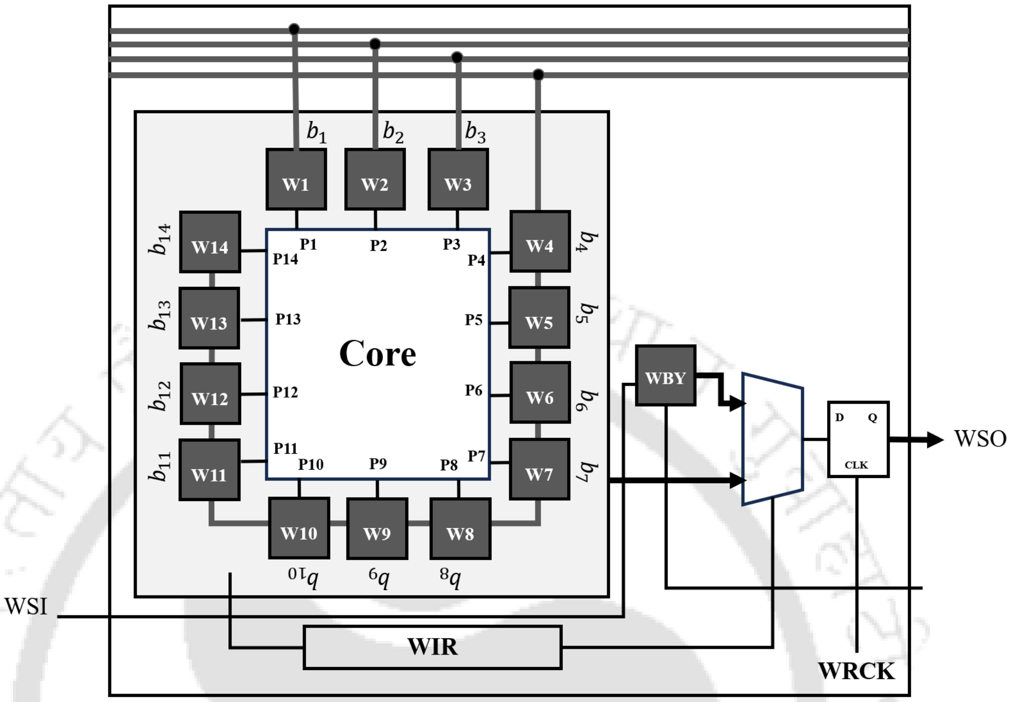


Figure 5.8: *TAM architecture with WBY Normal Mode*

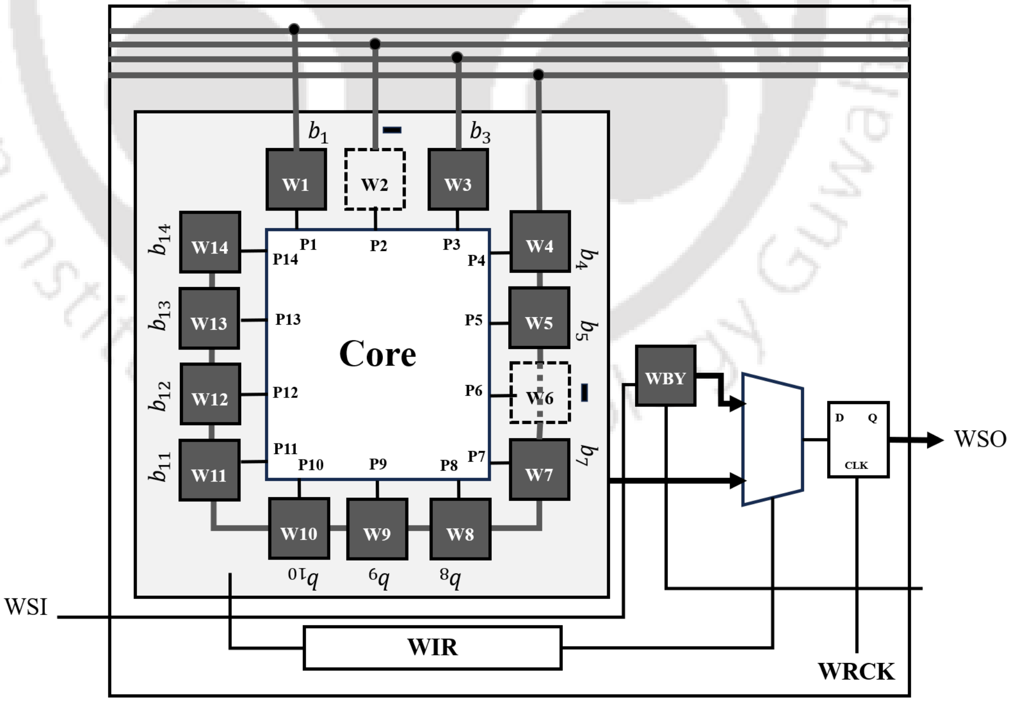


Figure 5.9: *TAM architecture with WBY Bypass Mode*

Figure 5.10: *Proposed TAM architecture*

have been replaced with WBYs, as clearly depicted in Figure 5.9. In the context of incomplete testing, consider a test vector $b_1, -, b_3, b_4, b_5, -, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{13}$, b_{14}, which is derived by omitting the LSTBs. This test vector is then applied to the input pins of the cores, as shown in Figure 5.9. When test bits are applied to the core's input pins, these WBYs operate in storage mode. However, in case of incomplete testing where certain bits in corresponding columns in all test vectors within the test set will be removed, the corresponding WBY for that specific pin switches to Bypass mode using *Wrapper Clock Terminal* (WRCK) and WS_BYPASS instruction in CTL. As the test vector is inserted sequentially, these transparent WBYs actively redirect the test bit to bypass the affected test pins, ensuring that the test bit proceeds to the next WBY in the sequence, thereby effectively maintaining the integrity of the testing process.

To be more specific, in the illustrated example of incomplete testing in Figure 5.9, no test bit will be applied to the input pins of P2 and P6 in the core. Consequently, WBYs W2 and W6 will operate in Bypass mode, allowing them to remain transparent to input bits. All other WBYs, except W2 and W6, will continue operating in storage mode.

5.5.5 Incomplete Testing and Reduction in TP

TP, which measures the number of transitions between 1 and 0 and vice versa, plays a pivotal role in evaluating power consumption during the testing phase of SoC devices. It relies on counting transitions occurring in different aspects of the testing process, encompassing scan chains, wrapper cells, and the manipulation of test vectors at input and output pins. The significance of TP lies in its direct correlation with power consumption during testing, making it a crucial metric for power-constrained devices and environments. Reducing TP is essential to minimize power consumption, emphasizing the need to optimize test vectors and testing methodologies for both efficient testing and reduced power consumption. This reduction in TP is achieved through two primary mechanisms in incomplete testing.

1. **Reduced TP in Parallel Shifting of Test Vector** Incomplete testing involves the application of modified test vectors to the input pins of cores through WBRs, which contain fewer test bits. During the parallel shifting of test vectors, each bit travels through one WBR and is applied to a single input pin. In the context of modified test vectors for incomplete testing, if a particular bit is excluded as a LSTB, it results in the reduction of one transition associated with that bit. This reduction in transitions contributes to an overall decrease in transitions, subsequently leading to a reduction in TP.

2. **Reduced TP in Sequential Shifting of Test Vector** Incomplete testing entails the use of modified test vectors on core input pins through WBRs, which contain a reduced number of test bits. During the sequential shifting of test vectors, individual bits traverse through multiple WBRs and are applied to any of input pins. In the context of modified test vectors for incomplete testing, if a particular bit is omitted as a LSTB, it leads to the reduction of transitions across multiple WBRs through which that bit travels. These reductions in transitions collectively contribute to an overall decrease in transitions, subsequently resulting in a reduction in TP.

Therefore, it can be inferred that this approach emphasizes the importance of incomplete testing methods to reduce power consumption during testing by omitting test bits as per TP constraints.

5.6 Experimental Results

All the experiments are performed on SoC using the ISCAS'85 and ISCAS'89 benchmark circuits as the foundation [153]. The programming framework for these experiments was established through the utilization of the C++ language. Test patterns were meticulously generated using the ATALANTA-M tool [147], and the ensuing fault simulations were executed with precision employing the HOPE simulator [148]. The experiments described herein are performed exclusively within the defined SoC framework, as depicted in Figure 5.11. This specific SoC configuration comprised six discrete cores, designated as c17, c432, c880, c2670, c6288, and c1355, with two test buses. The cumulative bus width was partitioned, resulting in a combined allocation of 48.

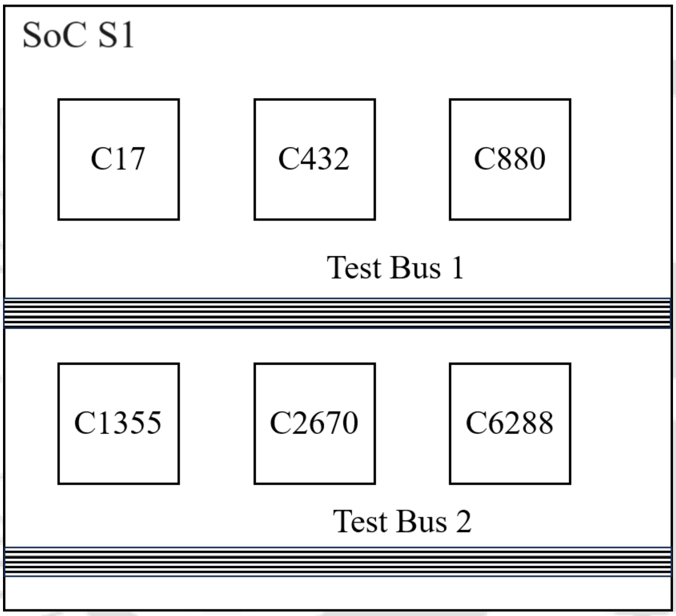


Figure 5.11: *Equations to calculate transitions in wrapper chain after insertion of two test vector sequentially*

5.6.1 Results for TP for SoC with Combinational Cores

The TP was computed through the utilization of the test pattern generated via our proposed method. Within SoC, as depicted in Figure 5.11, there exist two test buses. Various random distributions of bus width were applied to both of these test buses, alongside an optimal core-to-bus scheduling to optimize TP.

In Table 5.1, the bus width distribution is illustrated as (1,47), signifying that test bus 1 possesses a bus width of 1, whereas test bus 2 has a bus width of 47, thereby yielding a combined bus width of 48. The core schedule chosen

Table 5.1: *TP for Bus Width Distribution of (1,47)*

Core scheduling (2,1,2,2,2,2)			
Core	TP for Complete Testing	TP for Incomplete Testing	TP Saving in %
c17.test	14	13	7.14
c432.test	32246	30163	6.45
c880.test	6445	5797	10.05
c1355.test	1768	1648	6.78
c2670.test	1741799	1313664	24.58
c6288.test	483	480	0.62
TP for SoC S1	1782107	13,51,765	24.14

was (2,1,2,2,2,2), designating test bus 1 to core 2 (C880), while the remaining cores were assigned to bus 2. Notably, a 24% reduction in TP for the SoC was observed when employing the bus width distribution of (1,47) as shown in Table 5.1. Similarly, TP was assessed for varying bus width distributions and core scheduling configurations.

Table 5.2 offers a comprehensive comparison of TP results for various test configurations within a SoC testing scenario, including core scheduling (2,1,1,2,2,1), individual core performance under complete and incomplete testing, and the corresponding TP savings expressed as percentages. Specifically, it outlines TP outcomes for cores c17.test, c432.test, c880.test, c1355.test, c2670.test, and c6288.test, revealing the TP savings percentages for each. Additionally, it summarizes the overall SoC performance, indicating that incomplete testing results in a TP savings of approximately 20.23% compared to complete testing. In essence, the table provides a comprehensive view of TP results, aiding in the evaluation of testing strategies for SoC cores. This Table 5.3 presents a comparison of TP results for different test configurations within a SoC testing scenario, employing a specific core scheduling strategy (2,2,1,1,2,1). It provides data for individual cores, including c17.test, c432.test, c880.test, c1355.test, c2670.test, and c6288.test, showcasing the TP achieved under both complete and incom-

Table 5.2: *TP for Bus Width Distribution of (4,44)*

Core scheduling (2,1,1,2,2,1)			
Core	TP for Complete Testing	TP for Incomplete Testing	TP saving in %
c17.test	14	13	7.14
c432.test	27372	25407	7.17
c880.test	86969	74607	14.21
c1355.test	1768	1648	6.78
c2670.test	1793095	1409731	21.38
c6288.test	13915	12634	9.20
TP for SoC S1	1910771	1524040	20.23

Table 5.3: *TP for Bus Width Distribution of (40,8)*

Core scheduling (2,2,1,1,2,1)			
Core	TP for Ccomplete Testing	TP for Incomplete Testing	TP saving in %
c17.test	14	13	7.14
c432.test	914	851	6.89
c880.test	75569	64263	14.96
c1355.test	50000	47160	5.68
c2670.test	1874035	1434198	23.47
c6288.test	10311	9500	7.86
TP for SoC S1	1999537	15,55,985	22.18

plete testing methods, along with the corresponding TP savings expressed in percentages. Notably, it highlights that for the entire SOC, complete testing yields a TP of 1,999,537, whereas incomplete testing results in a TP of 1,555,985, indicating a TP savings of approximately 22.18%. This table serves as a comprehensive reference for evaluating the effectiveness of different testing strategies for SoC cores. This Table 5.4 provides a comparison of TP results for various test configurations in a SoC testing context, specifically utilizing the core scheduling strategy (1,1,2,1,2,1). The table includes data for individual cores, such as c17.test, c432.test, c880.test, c1355.test, c2670.test, and c6288.test, indicating the

Table 5.4: *TP for Bus Width Distribution of (13,35)*

Core scheduling (1,1,2,1,2,1)			
Core	Tp for Complete Testing	TP for Incomplete Testing	TP saving %
c17.test	14	13	7.14
c432.test	14294	13721	4.00
c880.test	19635	17025	13.29
c1355.test	37158	34956	5.92
c2670.test	1976907	1588444	19.65
c6288.test	6621	6182	6.63
TP for SoC S1	2052019	1660341	19.08

TP achieved under both complete and incomplete testing scenarios, along with the corresponding TP savings percentages. Notably, it demonstrates that for the entire SoC, complete testing results in a TP of 2,052,019, while incomplete testing yields a TP of 1,660,341, representing a TP savings of approximately 19.08%. This table serves as a valuable reference for assessing the effectiveness of different testing strategies for SoC cores.

This Table 5.5 presents a comparison of TP results for different test configurations in a SoC testing context with a specific core scheduling strategy (1,2,2,1,1,1). The table includes data for individual cores, such as c17.test, c432.test, c880.test, c1355.test, c2670.test, and c6288.test, indicating the TP achieved under both complete and incomplete testing scenarios, along with the corresponding TP savings percentages. Notably, it shows that for the entire SoC, complete testing results in a TP of 2,394,559, whereas incomplete testing yields a TP of 1,717,514, representing a substantial TP savings of approximately 28.27%. This table provides valuable insights into the performance differences between complete and incomplete testing strategies for SoC cores.

This Table 5.6 provides a comparison of TP results for different test configurations within a SoC testing scenario, utilizing the core scheduling strategy (2,1,2,1,2,1). It includes data for individual cores, such as c17.test, c432.test,

Table 5.5: *TP for Bus Width Distribution of (16,32)*

Core scheduling (1,2,2,1,1,1)			
Core	TP for Complete Testing	TP for Incomplete Testing	TP Saving %
c17.test	14	13	7.14
c432.test	1426	1331	6.66
c880.test	24269	20573	15.22
c1355.test	29614	28248	4.61
c2670.test	2338079	1662841	28.88
c6288.test	4853	4508	7.10
TP for SoC S1	2394559	1717514	28.27

Table 5.6: *TP for Bus Width Distribution (18,30)*

Core scheduling (2,1,2,1,2,1)			
Core	TP for Complete Testing	TP for Incomplete Testing	TP saving in %
c17.test	14	13	7.14
c432.test	7558	7429	1.70
c880.test	31225	26655	14.63
c1355.test	21818	20542	5.84
c2670.test	2115589	1655448	21.75
c6288.test	3069	2808	8.50
TP for SoC S1	2174703	1712895	21.23

c880.test, c1355.test, c2670.test, and c6288.test, showing the TP achieved under both complete and incomplete testing conditions, along with the corresponding TP savings percentages. Notably, it indicates that for the entire SoC, complete testing results in a TP of 2,174,703, while incomplete testing yields a TP of 1,712,895, resulting in a TP savings of approximately 21.23%. This table provides valuable insights into the performance differences between complete and incomplete testing strategies for SoC cores.

This Table 5.7 presents data on bus width distribution (28,20) and TP for a specific core scheduling strategy (1,1,1,2,2,2) in a SoC testing scenario. It pro-

Table 5.7: *TP for Bus Width Distribution of (28,20)*

Core scheduling (1,1,1,2,2,2)			
Core	TP for Complete Testing	TP for Incomplete Testing	TP saving %
c17.test	14	13	7.14
c432.test	6118	5759	5.86
c880.test	42549	36179	14.97
c1355.test	12006	11270	6.13
c2670.test	2153497	1706000	20.78
c6288.test	1215	1160	4.52
TP for SoC S1	2209029	1760381	20.30

vides information for individual cores, including c17.test, c432.test, c880.test, c1355.test, c2670.test, and c6288.test, indicating their test power consumption under both complete and incomplete testing conditions, as well as the corresponding test power savings expressed in percentages. Notably, it highlights that for the entire SoC, complete testing results in a test power consumption of 2,209,029, while incomplete testing yields a consumption of 1,760,381, resulting in a test power savings of approximately 20.30%. This table offers insights into the test power efficiency of different cores within the SoC and the overall impact of incomplete testing on power consumption.

Tables 5.1 to 5.7 provide an assessment of TP savings for each of the six individual cores, ranging from c17 to c6288. Table 5.8 consolidates the findings from Tables 5.1 to 5.7 and conducts a comprehensive analysis for the entire SoC. TP values are computed across different bus width distributions, spanning from (1,47) to (28,20) for SoC S1. In Table 5.8, the second row and second column represent a bus width configuration of (1,47). In this configuration, the TP for complete testing is recorded as 1,782,107, while for incomplete testing, it stands at 1,351,765. The fifth column in the second row concludes that the TP savings amount to 24.15%. Throughout the table, similar columns and rows provide information on TP savings for various bus width distributions, with the highest

Table 5.8: *Analysis of whole SoC S1 for TP Saving*

SoC	Bus Width Distributions	TP for Complete Testing	TP for Incomplete Testing	TP Saving %
S1	1,47	1782107	1351765	24.15
S1	4,44	1910771	1524040	20.24
S1	40,8	1999537	1555985	22.18
S1	13,35	2052019	1660341	19.09
S1	16,32	2394559	1717514	28.27
S1	18,30	2174703	1712895	21.24
S1	20,28	2209029	1760381	20.31

observed TP savings being 28.27% for the (16,32) bus width distribution.

In Figure 5.12, a visual representation illustrates the connection between TP savings and the distribution of bus width. This graph clearly demonstrates that the TP value is influenced by the choice of bus width distribution within the same SoC. Notably, the graph highlights that the highest TP savings are achieved when utilizing a bus width distribution of (16,32). In Figure 5.13, a comparison is presented between complete testing and incomplete testing. It is clear from the data that incomplete testing consistently results in TP savings across various bus width distributions and core-to-bus scheduling configurations. The trend observed indicates a continual increase in TP for all the specified scenarios.

5.6.2 Results for TP for SoC with Sequential and Combinational Cores

In the SoC S1, all six cores are combinational cores. To evaluate the effectiveness of our approach, we will analyze sequential cores in the following section. The experiments performed in this research are solely conducted within the designated SoC framework, as depicted in Figure 5.14. This specific SoC configuration comprises a total of eight individual cores, including three combinational cores (c880, c2670, and c7552) and five sequential cores (s953, s1238, s1196, s5378, and s9234). Additionally, it features two test buses, and the total bus width is partitioned, leading to a combined allocation of 48.

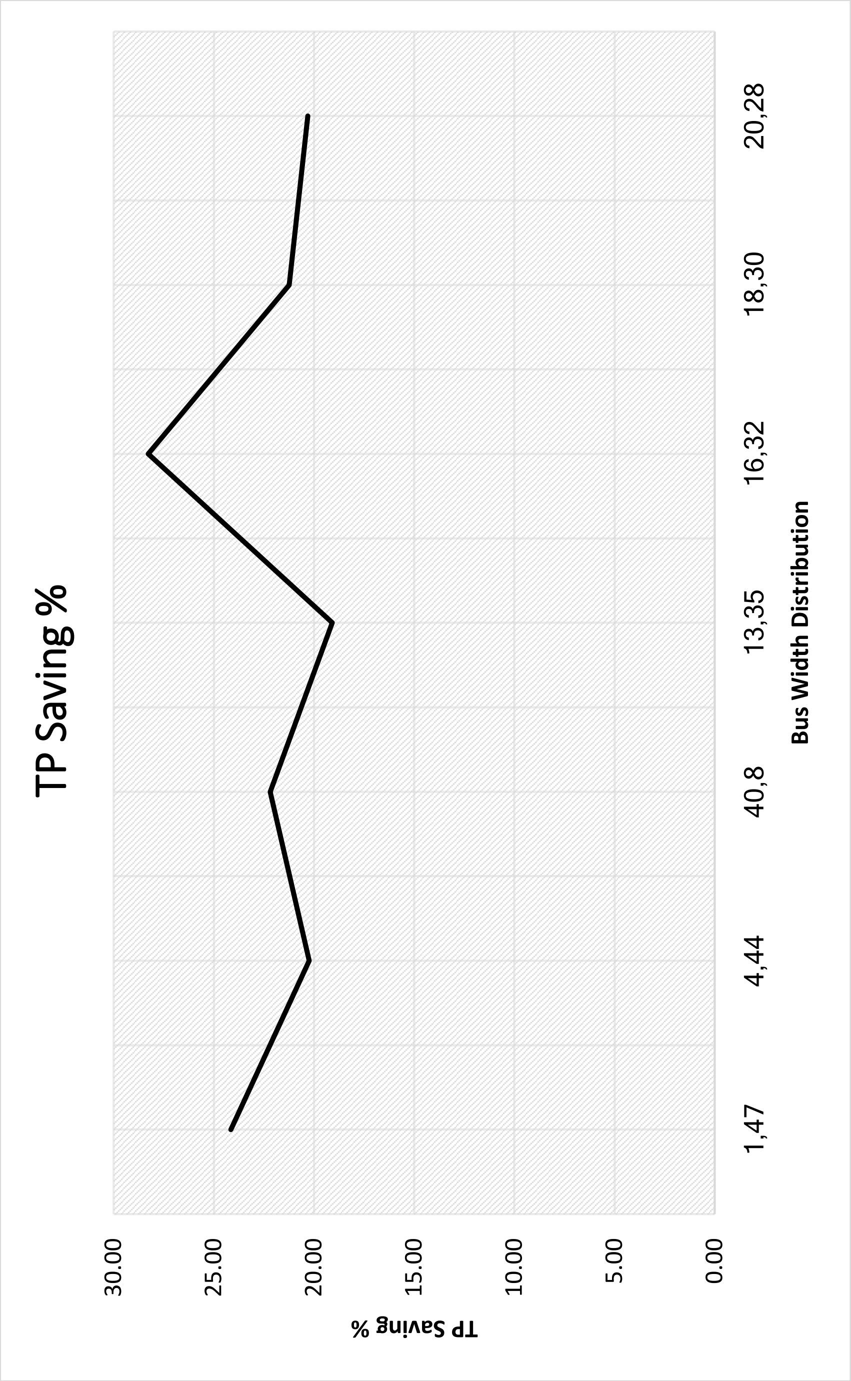


Figure 5.12: *Bus Width and TP saving in %*

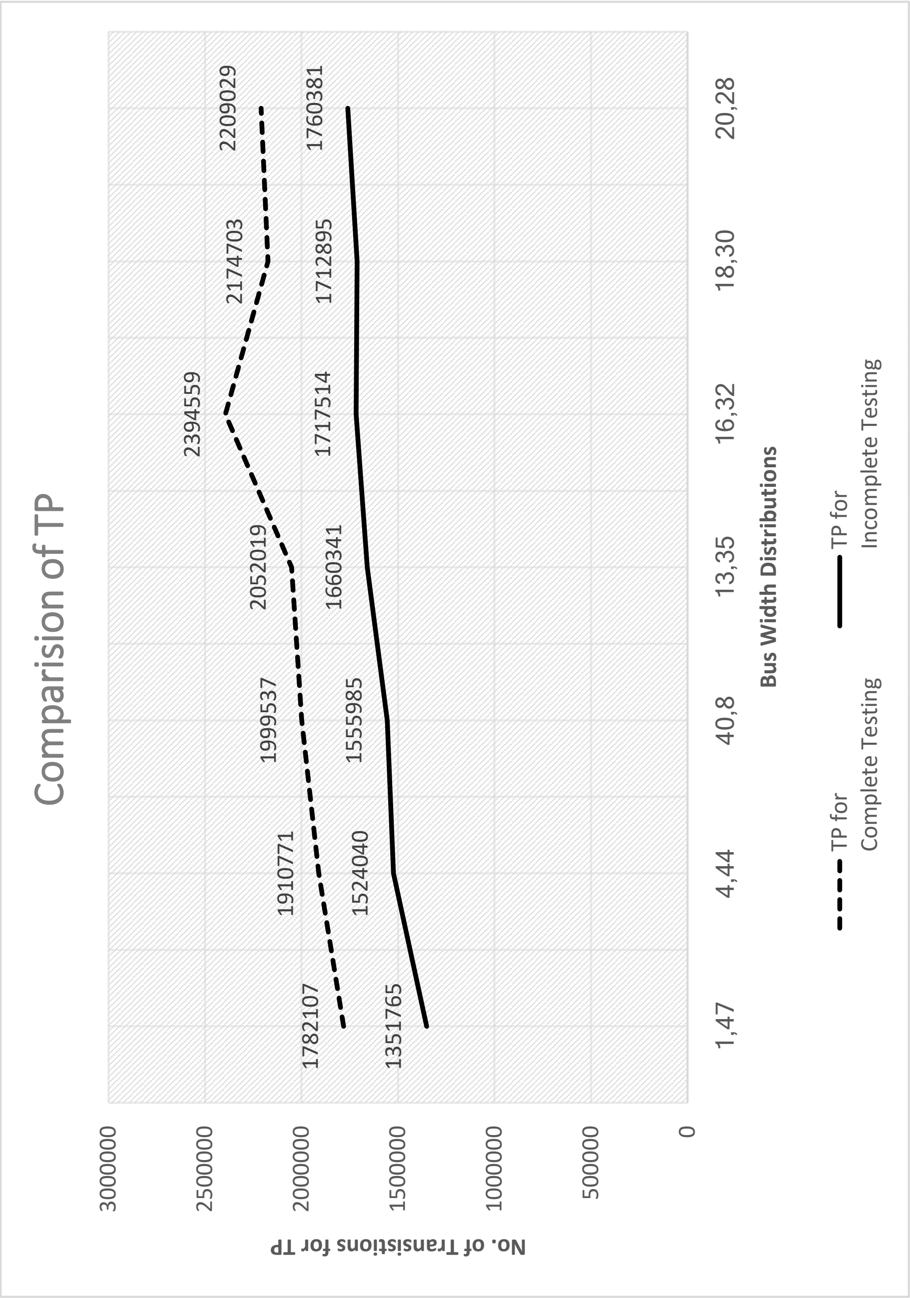


Figure 5.13: *SoC S2*

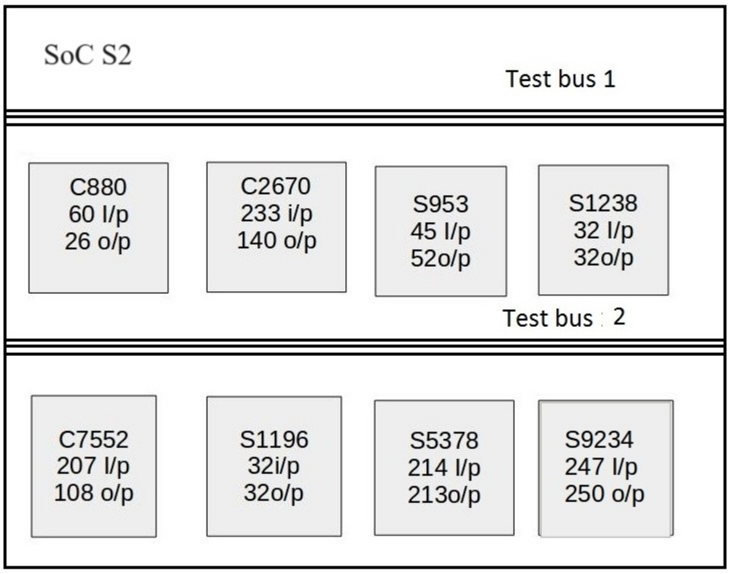


Figure 5.14: *SoC S2*

This Table 5.9 provides a detailed analysis of TP performance for a SoC labeled as "S2" under various Bus Width Distribution configurations, all maintaining a constant bus width of 48 bits. The "Distribution" column specifies the ratio of bits allocated to the primary bus versus the secondary bus. For each configuration, it presents the TP achieved under both Complete Testing and Incomplete Testing conditions. The "TP Saving %" column calculates the percentage difference between the TP achieved under complete testing and incomplete testing, representing the TP savings obtained by using an incomplete testing strategy.

For instance, when the bus distribution is (1,47) (1 bit allocated to the test bus 1 and 47 bits to test bus 2), the TP for complete testing is 9,461,414, while for incomplete testing, it is 7,978,533, resulting in a TP savings of approximately 15.67%. As the bus distribution ratio changes, the TP savings percentage also varies, ranging from 15.67% to 16.65% for different configurations. This table helps in understanding how altering the bus width distribution affects the trade-off between TP and testing efficiency in the SoC, where incomplete testing can lead to significant TP savings while maintaining acceptable testing quality.

The results clearly indicate that for SoC S2, an average power saving of 16%

Table 5.9: *TP Analysis for SoC with various Bus Width Distribution*

SoC	Bus	Distribution	TP for Complete Testing	TP for Incomplete Testing	TP Saving %
S2	48	1,47	9461414	7978533	15.67
S2	48	4,44	9503124	7988551	15.94
S2	48	8,40	9530114	7995703	16.1
S2	48	13,35	9577224	7983027	16.65
S2	48	16,32	9591798	7996651	16.63
S2	48	18,30	9602162	8007201	16.61
S2	48	20,28	9605616	8015709	16.55
S2	48	22,26	9611600	8026815	16.49

has been achieved for each distribution and optimal assignment. Figure 5.15 illustrates the characteristic curve, where the dotted curve represents the TP for complete testing of the SoC, and the continuous curve represents the TP for testing the SoC with modified test patterns after the removal of LSTBs.

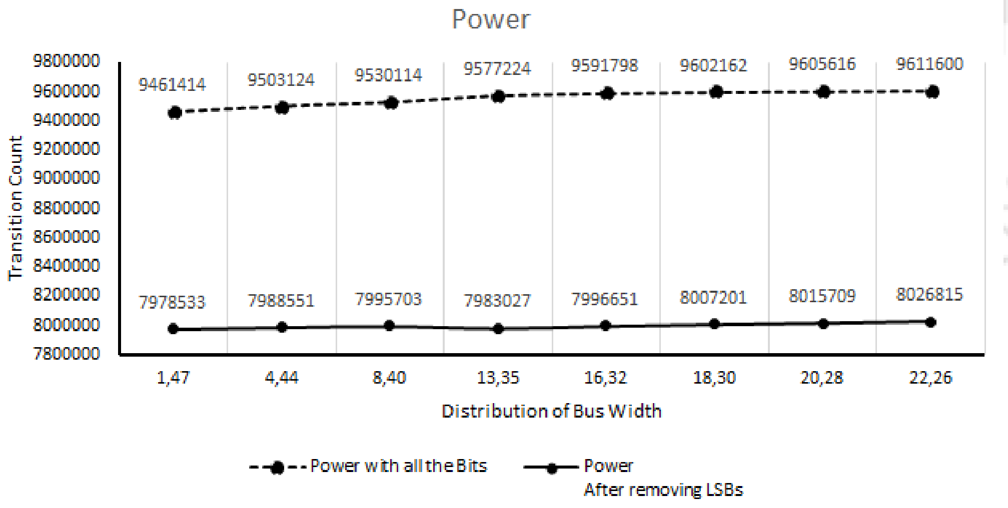


Figure 5.15: *TP Characteristics*

5.7 Conclusion and Future Work

In this chapter, a novel approach to incomplete testing that incorporates TP is presented. A modified test vector set, specifically tailored for incomplete testing, is introduced. Additionally, a design for a TAM that utilizes the IEEE 1500 test architecture to support this approach in general incomplete testing scenarios is proposed. The TP-aware incomplete testing method, characterized by the optimal assignment of cores to test buses, attains significant power savings, with only a slight compromise in fault coverage, as evidenced by experimental results showcasing a remarkable reduction in TP utilization. Looking forward, there is potential to develop alternative incomplete testing methods for application to a wider range of electronic devices, such as microfluidic chips and 3D hierarchical chips. It's important to note that the method primarily addresses Stuck-at Faults, and future research may extend its applicability to other fault types, including bridge faults and delay faults.

Chapter 6

Incomplete Testing Based on Approximation and its Impact on TP and TAT

6.1 Introduction

In the preceding chapters, diverse methodologies were investigated to enable the practice of *Incomplete Testing* of SoCs, acknowledging a certain compromise in testing quality. *Chapter 3* introduces a heuristic approach specifically designed for *Incomplete Testing* of SoCs.

Expanding upon the insights gained in *Chapter 3*, *Chapter 4* not only presents a heuristic approach for *Incomplete Testing* but also further refines this approach to optimize the TAT. The refinements involve modifications to the TAT equations, with the overarching goal of streamlining and enhancing the testing process while ensuring an acceptable level of test quality.

Advancing the thesis, *Chapter 5* extends the heuristic approach established in *Chapter 4* to optimize TP. This chapter introduces a modified TAM architecture specifically tailored for *Incomplete Testing*.

In summary, the thesis unfolds from the exploration of heuristic testing techniques for *Incomplete Testing* in *Chapter 3* to the development and refinement of this approach for TAT optimization in *Chapter 4*. *Chapter 5* then takes this optimized approach to further optimize TP, presenting a modified TAM architecture. This sequential progression illustrates a systematic and comprehensive exploration of testing strategies for SoCs, carefully considering trade-offs between testing quality and parameters such as TDV, TAT and TP.

In this chapter, a different approach to conduct *Incomplete Testing* of SoCs is introduced, drawing inspiration from approximate circuits. The chapter also analyses of its impact on testing metrics such as TDV, TAT, and TP.

6.2 Problem Formulation

Given a SoC, generate reduced test vectors set for incomplete testing of SoC to reduce TDV, TAT, TP and maximize the fault coverage.

6.3 Proposed Method

To understand the proposed method, focus shifts to Figure 6.1, which illustrates the multiplication of two 3-bit binary numbers.

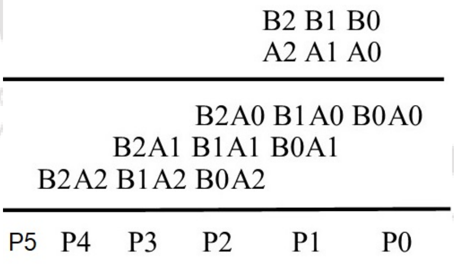


Figure 6.1: *Multiplication of two 3 bit numbers*

Figure 6.1 displays two 3-bit binary numbers denoted as A2, A1, A0 and B2, B1, B0, where A0 and B0 represent the *Least Significant Bits* (LSBs). The multiplication of these numbers yields a six-bit output, namely P5 to P0, with P0 signifying the LSB and P4 the *Most Significant Bits* (MSBs). Each bit in

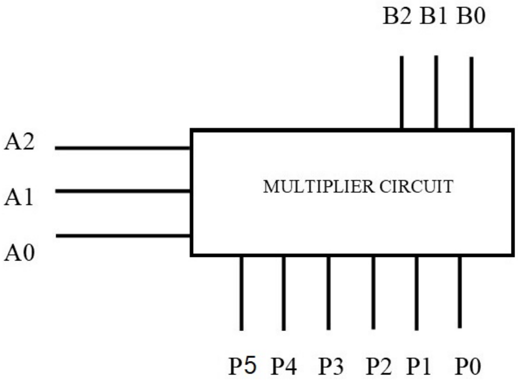


Figure 6.2: *3 bit multiplier*

the output holds a specific weight. The diagram demonstrates the use of adders in implementing a multiplier circuit. When the two 3-bit numbers, A2, A1, A0 and B2, B1, B0, are multiplied, individual products are generated and summed to obtain the final result. For instance, P0 represents the product of B0 and A0. Similarly, P1 is the sum of B1·A0 and B0·A1, P2 is the sum of B2·A0, B1·A1, and B0·A2. P3 is the sum of B1·A2 and B2·A1, and finally, the MSB P4 represents B2·A2.

Figure 6.2 illustrates the implementation of a 3-bit multiplier circuit. The binary numbers B2, B1, and B0, along with A2, A1, and A0, are fed into the input pins of the circuit. The multiplication result is then presented at the output pins as P4, P3, P2, P1, and P0.

In Figure 6.3, there is an example provided where two 3-bit binary numbers, 110 and 101, are multiplied, leading to an output of 11110.

Suppose we allow for a tolerance or error in the LSB P0 since it carries the smallest weight, and any error in this bit would have the least impact on the overall output value.

Let's consider the scenario where errors are allowed in output bits P0, P1, and P2, resulting in the reversal of these bits and producing a value of 001 instead of 110. Consequently, the final output of the multiplier becomes 11001 instead of

```
   110      (6)
   101      (5)
 -------
   110
  000
 110
 -------
 11110      (30)
```

Figure 6.3: *Example of multiplication of 3 bit numbers*

11110, with a value of 25 instead of 30. The percentage error can be calculated as $((30 - 25) \div 30) \times 100 = 16.66\%$.

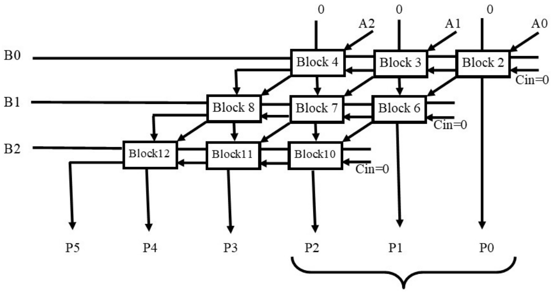
The multiplier circuit depicted in Figure 6.4(a) consists of multiple blocks, each containing an internal circuit shown in Figure 6.4(b) comprising an AND gate and a full adder. The output pins of the multiplier are labeled as P5, P4, P3, P2, P1, and P0.

To ensure complete testing, all nine blocks related to the output bits must be tested after implementation, which would require nine units of power if each block takes one unit of power to test.

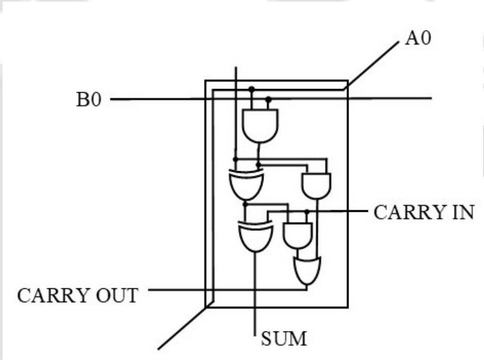
However, if errors in output bits P2, P1, and P0 can be tolerated, the corresponding blocks or circuit elements can be left untested, thereby reducing TDV, TAT, and TP. Figure 6.4(c) illustrates that components in block 2, responsible for output P0, can be left untested if errors on this output pin are tolerable. Consequently, the EX-OR gate inside block 2 shown in Figure 6.4(d) will not be tested.

For output P1, blocks 3 and 6 need to be tested, but the EX-OR gate inside block 6 can remain untested since it only affects output P1 and not the other outputs.

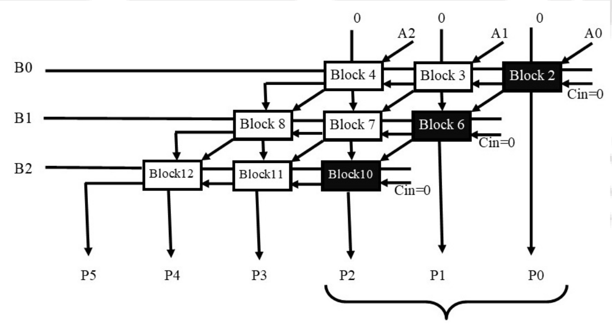
Similarly, for output P2, blocks 4 and 7 are fully tested, but the EX-OR gate in block 10, responsible for the value on output P1, will not be tested.



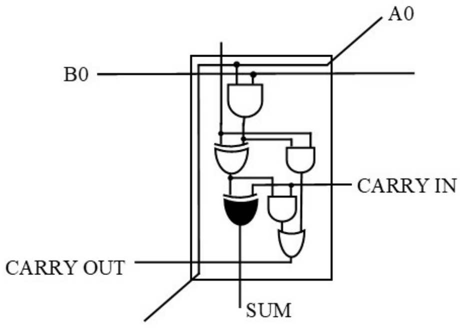
(a) Multiplication circuit having blocks



(b) Internal circuit of one block of fig 3a.



(c) Multiplication circuit having blocks incompletely tested



(d) Internal circuit of block incompletely tested

Figure 6.4: *Multiplier circuit*

By adopting this approach, SoC can be incompletely tested for the least significant outputs while still achieving acceptable accuracy for the intended application.

The proposed method, which can be easily understood using the example of circuit c17, is outlined below.

6.3.1 Step 1:

Construct the circuit shown in Figure 6.6 based on the provided benchmark circuit description in Figure 6.5 This circuit comprises five input lines, two output lines, and six NAND gates.

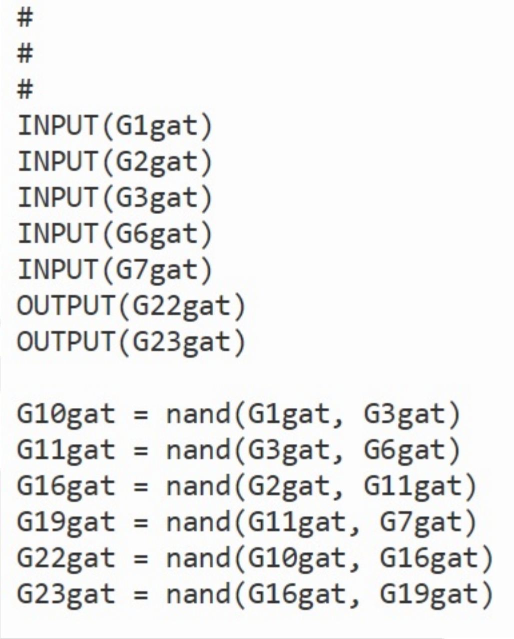
6.3.2 Step 2:

Create a graphical representation of the circuit in Figure 6.6 using input pins, output pins, and gates. Establish five input nodes: I_2, I_3, I_6, I_7, and I_1, which are connected to input lines INPUT(G2gat), INPUT(G3gat), INPUT(G6gat), INPUT(G7gat), and INPUT(G1gat), respectively. Additionally, create two output nodes: O_{23} and O_{22}, connected to output lines OUTPUT(G23gat) and OUTPUT(G22gat). Furthermore, include six nodes: $G_{23}, G_{22}, G_{10}, G_{19}, G_{16}$, and G_{11}, representing the six NAND gates.

6.3.3 Step 3:

We now have an equivalent graph of the c17 circuit, displayed in Figure 6.7. Our objective is to trace a backtrack path from the fault tolerable output back to the input while ensuring that the gates and connections on this path do not affect other components of the circuit. By doing so, we can ensure the accuracy of all other outputs except for the output pin O_{23}. Assuming O_{23} can tolerate errors, there is no need to test it. To obtain this backtrack path, we follow Algorithm 2 and Algorithm 3.

Algorithm 2 takes the equivalent graph of the circuit as its input and generates



```
#
#
#
INPUT(G1gat)
INPUT(G2gat)
INPUT(G3gat)
INPUT(G6gat)
INPUT(G7gat)
OUTPUT(G22gat)
OUTPUT(G23gat)

G10gat = nand(G1gat, G3gat)
G11gat = nand(G3gat, G6gat)
G16gat = nand(G2gat, G11gat)
G19gat = nand(G11gat, G7gat)
G22gat = nand(G10gat, G16gat)
G23gat = nand(G16gat, G19gat)
```

Figure 6.5: *Benchmark circuit of c17*

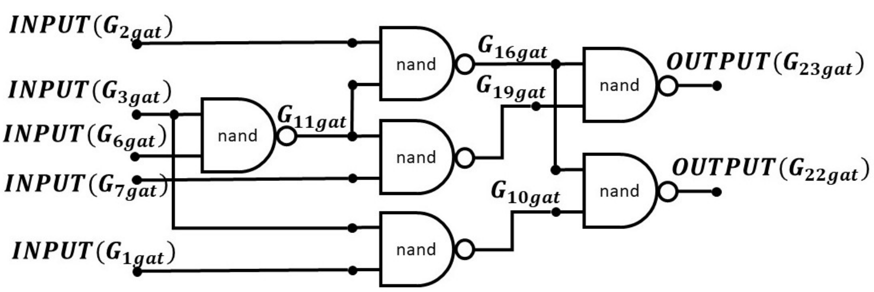


Figure 6.6: *c17 circuit*

ALGORITHM 2: Algorithm for finding GATES not to be tested

```
Input  : Circuit graph with $N$ number of nodes, $M$ number of output
         pins not to be tested
Output: GATES not to be tested
for $node = 1$ to $N$ do
    outdegree(node);
for $out\_pin = 1$ to $M$ do
    backtrack(out_pin);
```

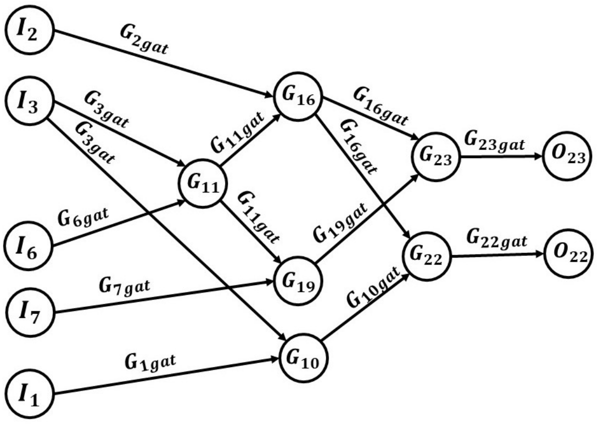


Figure 6.7: *Equivalent graph of c17 circuit*

ALGORITHM 3: Backtrack (i)

```
Input  : output pin $i$
Output: To find GATES on the path from output pin $i$ to one of the
        input pins
if $outdegree(parents(i)) > 1 \,||\, parents(i) = input\ pin$ then
    add $i$ to the result;
    break;
else
    Backtrack (parents($i$));
```

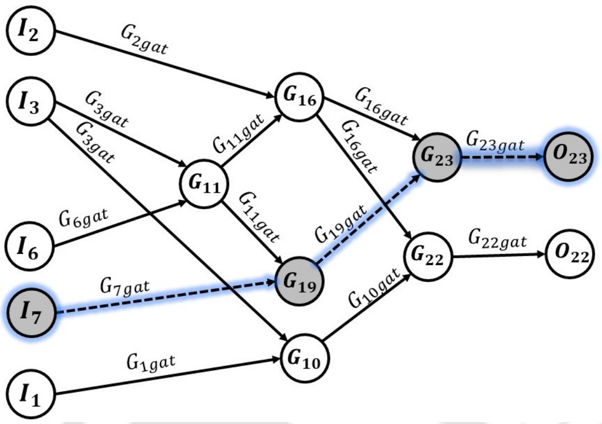


Figure 6.8: *Backtrack path for c17 circuit*

a pathway from the output pin to the input pin. O_{23} represents the output pin of least significance, allowing it to tolerate errors. For all the circuit components that depend on O_{23} and do not require precision on this particular output pin, testing is unnecessary. The value of O_{23} is determined by the gate G_{23}, which, in turn, relies on gates G_{16} and G_{19}. Since G_{19} does not impact any other component, it remains untested, but G_{16} must be tested, as errors in it would affect O_{22}. Additionally, G_{11} cannot be left untested either, as any errors in it would impact G_{16}, and consequently, O_{22}. However, input pin I_7 can be left untested since it has no effect on any other component apart from G_{19}. The pathway obtained through Algorithm 2, shown as $O_{23} \rightarrow G_{23} \rightarrow G_{19} \rightarrow I_7$ in Figure 6.8, identifies the hardware components that will not undergo testing. This partial testing approach will influence testing parameters such as FC, TDV, TAT, and TP.

6.4 Impact on Fault Coverage (FC)

After acquiring the path, we eliminate the components along it and conduct a fault simulation.

As seen in Figure 6.9, the NAND gates G_{23} and G_{19}, as well as the output pin O_{23} and input pin I_7, all located on the path $O_{23} \rightarrow G_{23} \rightarrow G_{19} \rightarrow I_7$, will

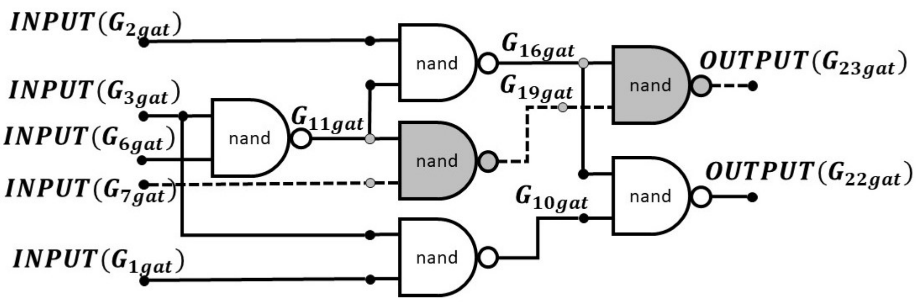
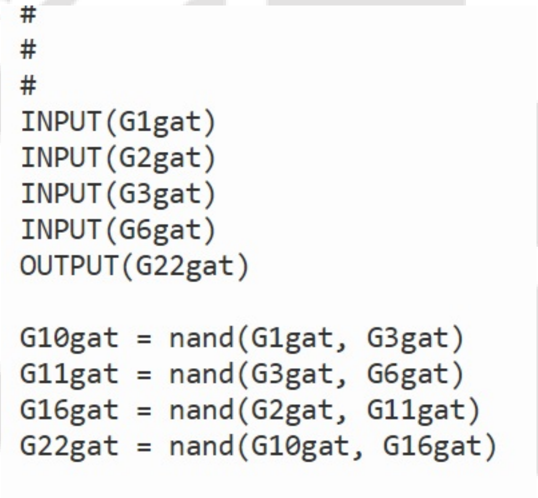


Figure 6.9: *Equivalent graph of c17 circuit for testing*



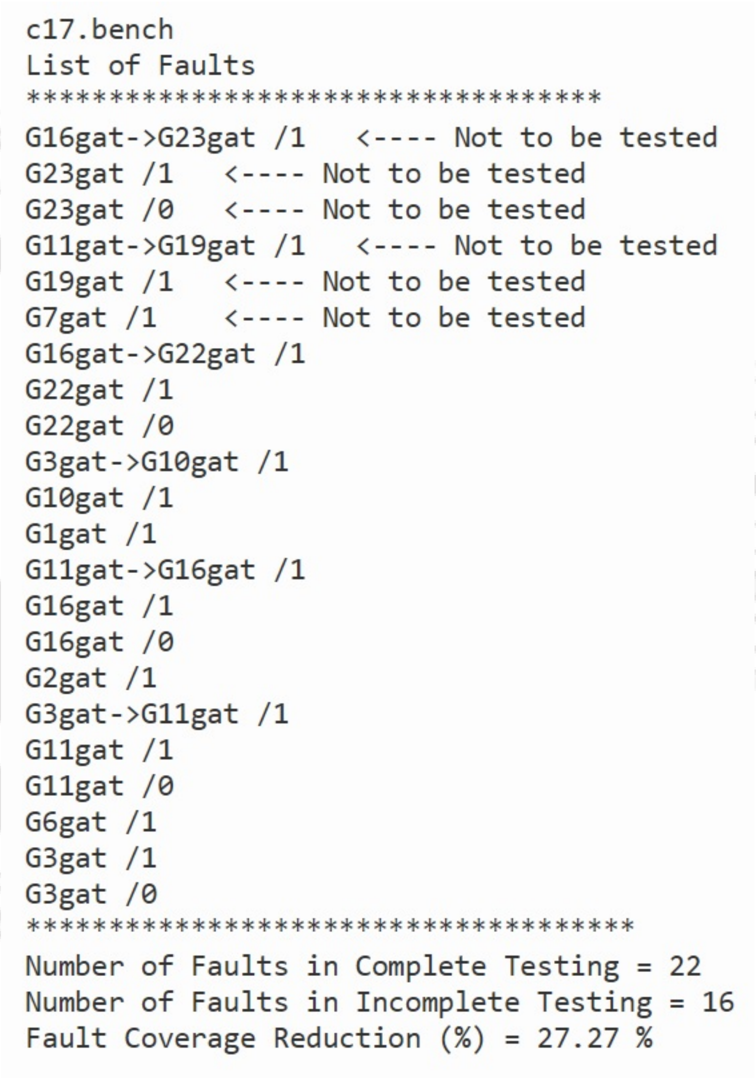
```
#
#
#
INPUT(G1gat)
INPUT(G2gat)
INPUT(G3gat)
INPUT(G6gat)
OUTPUT(G22gat)

G10gat = nand(G1gat, G3gat)
G11gat = nand(G3gat, G6gat)
G16gat = nand(G2gat, G11gat)
G22gat = nand(G10gat, G16gat)
```

Figure 6.10: *c17 benchmark circuit incomplete testing*

not undergo testing.

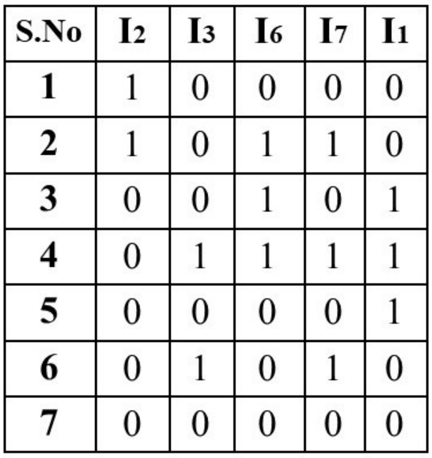
Figure 6.10, shows the the circuit description of modified circuit for incomplete testing after removing components located on the backtrack path.

In Figure 6.11, a list of multiple stuck-at-faults is provided. The total number of faults for complete testing amounts to 22 and 6 of them related to the excluded components, such as $G16gat \rightarrow G23gat/1$, $G23gat/1$, $G23gat/0$, $G11gat \rightarrow G19gat/1$, $G19gat/1$, and $G7gat/1$, will remain untested. Therefore, after excluding these faults, the incomplete testing will account for a total of 16 faults,



```
c17.bench
List of Faults
**********************************
G16gat->G23gat /1   <---- Not to be tested
G23gat /1   <---- Not to be tested
G23gat /0   <---- Not to be tested
G11gat->G19gat /1   <---- Not to be tested
G19gat /1   <---- Not to be tested
G7gat /1    <---- Not to be tested
G16gat->G22gat /1
G22gat /1
G22gat /0
G3gat->G10gat /1
G10gat /1
G1gat /1
G11gat->G16gat /1
G16gat /1
G16gat /0
G2gat /1
G3gat->G11gat /1
G11gat /1
G11gat /0
G6gat /1
G3gat /1
G3gat /0
************************************
Number of Faults in Complete Testing = 22
Number of Faults in Incomplete Testing = 16
Fault Coverage Reduction (%) = 27.27 %
```

Figure 6.11: *c17 benchmark circuit incomplete testing*



S.No	I2	I3	I6	I7	I1
1	1	0	0	0	0
2	1	0	1	1	0
3	0	0	1	0	1
4	0	1	1	1	1
5	0	0	0	0	1
6	0	1	0	1	0
7	0	0	0	0	0

Figure 6.12: *Test vector set for c17 benchmark circuit*

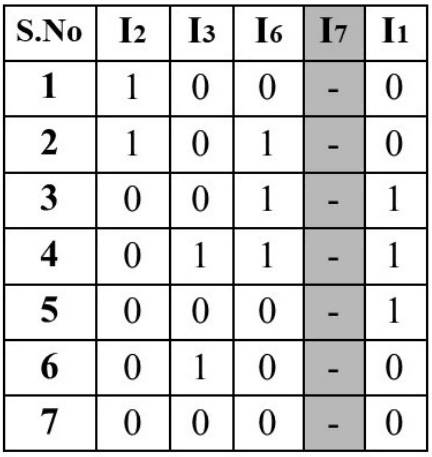
resulting in a fault coverage reduction of 27.27 %. Similarly, for all the cores, a slight compromise with the fault coverage will occur.

6.5 Impact on TDV

TDV will be reduced in two ways. There are two possible cases.

Case 1: When Backtrack Path Does Not Terminate On Any Input Node.

Skipping the testing of circuit elements found on the identified path leads to a reduction in the number of necessary test vectors. For instance, when testing the c17 circuit initially, we required seven test patterns to address all potential faults, as illustrated in Figure 6.12. However, employing our incomplete testing method allows us to bring down the number of test vectors to five, as depicted in Figure 6.13.



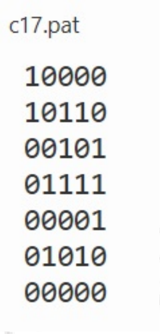
S.No	I2	I3	I6	I7	I1
1	1	0	0	-	0
2	1	0	1	-	0
3	0	0	1	-	1
4	0	1	1	-	1
5	0	0	0	-	1
6	0	1	0	-	0
7	0	0	0	-	0

Figure 6.13: *Modified test vector set for c17 benchmark circuit*

Case 2: When Backtrack Path Terminates on Any Input Node.

The algorithm identifies a path from an output pin to an input pin, indicating that this input pin only affects that specific output pin and not others. Therefore, we can skip testing that output pin by not applying a test bit to its corresponding input pin. This method reduces the number of test vectors and number of bits in each test vector. For instance, let's consider the c17 circuit with five input pins, which originally required seven test vectors, each consisting of five bits. However, by applying our incomplete testing method, we have managed to reduce the number of test vectors to five, with four bits per vector. In Figure 6.12, you can see the set of test vectors we generated for testing the input pins (I_1, I_2, I_3, I_6, and I_7) of the c17 circuit. This reduction in test vectors was achieved by excluding the test bit corresponding to input pin I_7 from all the test vectors found on the backtrack path obtained from Algorithm 2. The reason behind this exclusion is that input pin I_7 is omitted from the circuit during incomplete testing, as demonstrated in Figure 6.13.

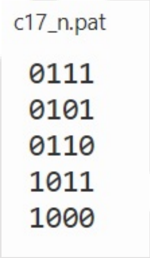
To elaborate further, there are two approaches to measure the TDV. For complete testing (as shown in Figure 6.14), the c17 circuit requires 7 test vectors,



c17.pat

```
10000
10110
00101
01111
00001
01010
00000
```

Figure 6.14: *Test patterns for c17 benchmark circuit in complete testing*



c17_n.pat

```
0111
0101
0110
1011
1000
```

Figure 6.15: *Test patterns for c17 benchmark circuit in incomplete testing*

each comprising 5 bits, resulting in a TDV of $7 \times 5 = 35$ bits.

In contrast, for incomplete testing (as depicted in Figure 6.15), the c17 circuit needs only 5 test vectors, each with 4 bits, leading to a TDV of $5 \times 4 = 20$ bits. By adopting the incomplete test strategy, substantial TDV savings can be achieved, which can be calculated as ((35-20) ÷ 35) x 100 = 42.85%.

6.6 Impact on TAT

6.6.1 Computation of TAT

To determine the TAT, optimal assignment of test buses to cores is done by the method proposed in Chapter 4 section 4.4.

6.6.2 Reduction in TAT

TAT will be reduced in two ways in incomplete testing.

When Backtrack Path Does Not Terminate On Any Input Node.

Suppose p_r number of test pattern removed from test vector set in incomplete testing then reduced TAT will calculated as follows:

$$T_{ij} = \begin{cases} t_i, & \text{if}(\psi_i \leq w_j) \\ t_i \times (\psi_i - w_j + 1), & \text{if}(\psi_i > w_j) \end{cases} \tag{6.1}$$

where $\psi_i = MAX(m_i, n_i)$

m_i=number of inputs of core i, $0 < i \leq N_c$

n_i=number of outputs of core i, $0 < i \leq N_c$

$$t_i = \begin{cases} p_i - p_r, & \text{for combinational core} \\ ((p_i - p_r) + 1) \times f_i/N_i + (p_i - p_r), & \text{for cores with internal scan chain} \end{cases} \tag{6.2}$$

where core i contains f_i flip-flops and N_i internal scan chains.

p_i is the number of test patterns.

When Backtrack Path Terminates On Any Input Node.

The Algorithm 1 identifies a path from an output pin to an input pin, indicating that this input pin only affects that specific output pin and not others. Therefore, we can skip testing that output pin by not applying a test bit to its corresponding input pin. Suppose there are L number of bit positions where test bits will be removed from all the test vector then

$$\psi_i = \begin{cases} m_i - L, & \text{if}(m_i > n_i) \\ n_i, & \text{if}(m_i < n_i) \end{cases} \tag{6.3}$$

After removing bits from the L designated bit positions in all test vectors, reduced TAT will be calculated as follows :

$$T_{ij} = \begin{cases} t_i, & \text{if}(\psi_i \leq w_j) \\ t_i \times ((m_i - L) - w_j + 1), & \text{if}(\psi_i > w_j) and (m_i > n_i) \\ t_i \times (n_i - w_j + 1), & \text{if}(\psi_i > w_j) and (m_i < n_i) \end{cases} \tag{6.4}$$

Net reduction in TAT after incomplete testing will be calculated as follows:

$$S_{ij} = \begin{cases} Zero, & \text{if}(\psi_i \leq w_j) \\ t_i \times ((L) - w_j + 1), & \text{if}(\psi_i > w_j) and (m_i > n_i) \\ Zero, & \text{if}(\psi_i > w_j) and (m_i < n_i) \end{cases} \tag{6.5}$$

6.7 Impact on TP

6.7.1 Reduction In TP In Incomplete Testing

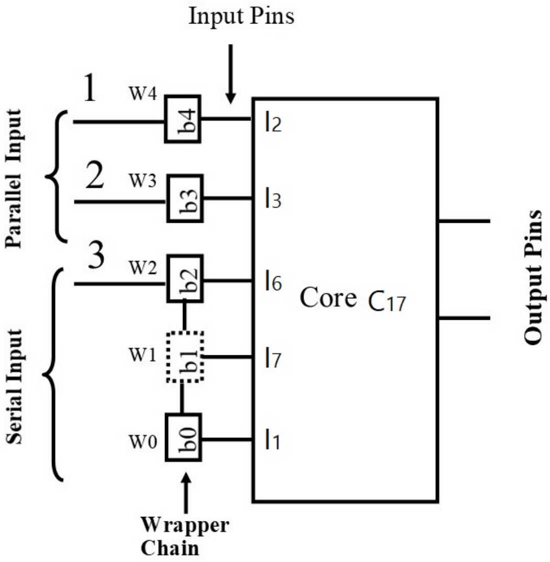


Figure 6.16: *Modified TAM architecture for incomplete testing*

Consider the example depicted in Figure 6.16, where input pins I_2, I_3, I_6, I_7, and I_1 of core c17 are fed with test vectors b_4, b_3, b_2, b_1, and b_0 respectively. Test vector bits b_4 and b_3 are transported through test bus wires 1 and 2 and applied to input pins I_2 and I_3 via WBRs W_4 and W_3 in parallel, while test vector bits b_2, b_1, and b_0 are transported on test bus wire 3 and applied to input pins I_6, I_1, and I_7 through WBRs W_2, W_1, and W_0 sequentially. As a result, when test vectors are applied one after another, the transitions in test bit values will be calculated in two ways: the transition in W_4 and W_3 will be calculated by hamming distance due to parallel insertion, and the transition in WBRs W_2, W_1, and W_0 will be calculated by equation 6.9 due to serial insertion.

$$tr_{total} = \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij} \tag{6.6}$$

This will result in a reduction of TP consumption in two ways. Firstly, in incomplete testing, a smaller number of test vectors will lead to fewer bit flips, ultimately reducing TP. Secondly, in incomplete testing, pin I_7 will not be tested, and test WBR W_1 will become transparent to input bits. Consequently, all the bit flips in WBR W_1 corresponding to input pin I_7 will be eliminated straightforwardly. Fewer transitions will reduce the TP.

6.8 Experimental Results

To perform the experiments, the SoC was employed, and the ISCAS'85 and ISCAS'89 benchmark circuits were used for testing purposes [153]. The experiment program was implemented in C++, while the test patterns were generated using ATALANTA [147]. Furthermore, we made use of the HOPE simulator for conducting fault simulations [148]. All the experiments were executed on the SoC illustrated in Figure 6.17, which incorporates eight cores (c432, c880, c2670, c6288, s27, s298, s444, s520) and two test buses. The total bus width of 48 is distributed between both buses.

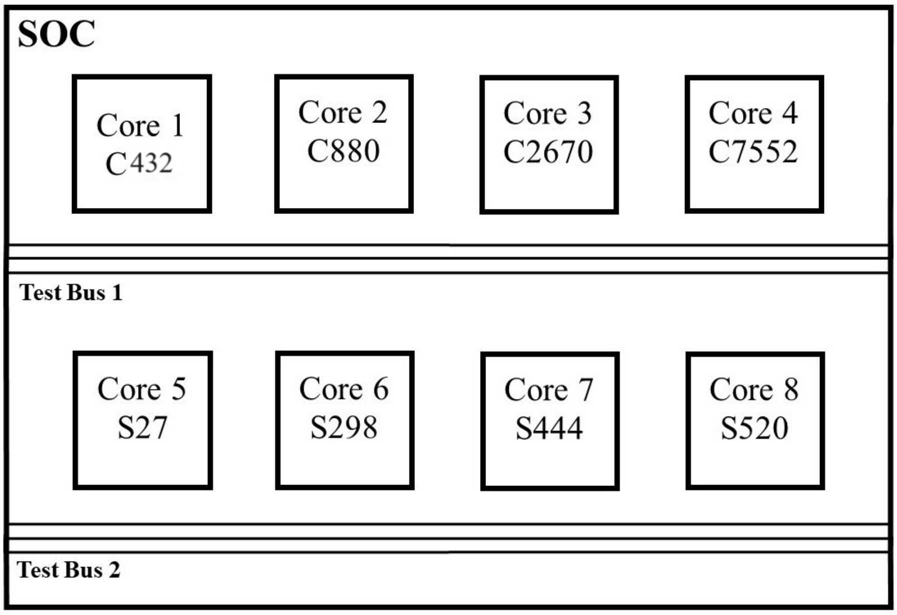


Figure 6.17: *SOC used for experiments*

6.8.1 Results for FC and TDV

The results for FC and TDV are detailed in Table 6.1. We generated test patterns and fault lists for all cores and employed a backtracking approach for comprehensive testing. This approach involved excluding hardware components associated with the backtrack path, as explained in section 6.4, and removing faults corresponding to the excluded hardware. The total number of faults for both complete and incomplete testing is displayed in columns 2 and 4 of Table 6.1, respectively, while the fault reduction is depicted in column 6. TDV can be calculated as the product of the number of test vectors and the number of bits in a test vector. The total TDV for complete and incomplete testing is presented in columns 3 and 5, respectively, with the reduction in TDV displayed in column 7. The TDV comparison between complete and incomplete testing is summarized in Table 6.1. For cores c880, c2670, c432, c6288, s27, s298, s444, and s520, the TDV reduction amounts to 27.78%, 45.87%, 52.17%, 40.96%, 33.33%, 39.29%, 43.33%, and 48.77%, respectively, while the fault coverage is compromised by up to 0.76%,

Table 6.1: *Results for FC and TDV*

Cores	No of faults in complete testing	TDV for complete testing	No. of faults in incomplete testing	TDV for incomplete testing	Fault reduction (%)	TDV reduction (%)
c432	524	2592	520	1872	0.76	27.78
c880	942	6000	909	3248	3.50	45.87
c2670	2747	45435	2695	21733	1.89	52.17
c7552	7550	74313	7484	43878	0.87	40.96
s27	32	63	26	42	18.75	33.33
s298	308	952	271	578	12.01	39.29
s444	474	1320	470	748	0.84	43.33
s520	555	2688	471	1377	15.14	48.77

3.50%, 1.89%, 0.87%, 18.75%, 12.01%, 0.84%, and 15.14%, respectively.

6.8.2 Results for TAT

Table 6.2 showcases the TAT for the ideal allocation of cores to the test buses within the SoC. We applied a PSO-based scheduling algorithm, as detailed in Chapter 4, to achieve this optimal allocation. The test bus maintains a width of 48, and we conducted experiments with various bus width distributions: (1,47), (4,44), (28,20), (24,24), (32,16), and (40,8), as presented in the first column of Table 6.2. Columns 2 and 4 illustrate the optimal core-to-bus assignments, while columns 3 and 5 depict the TAT for complete and incomplete testing, respectively. The reduction in TAT is displayed in column 6.

For instance, in the first row of Table 6.2, the bus width distribution between two test buses is (1,47). The optimal core-to-test bus allocation is represented as 11122121, signifying that test bus 1 serves cores 1, 2, 3, 6, and 8, while test bus 2 serves cores 4, 5, and 7. This optimal allocation results in a minimized TAT for the SoC, amounting to 57863 cycles for complete testing and 30074 cycles for incomplete testing. The TAT can be calculated in real-time based on the system's frequency. The corresponding reduction in TAT for this optimal allocation is shown in row 1, column 6, at 48.03% for the SoC. Similarly, TAT is computed for other combinations of test bus width distributions, and for each distribution, the savings average around 48%, which is a substantial improvement.

Table 6.2: *Results for TAT*

Buswidth distribution	Optimal allocation	TAT for SOC (complete test)	Optimal allocation	TAT for SOC (incomplete test)	Reduction in TAT (%)
(1, 47)	11122121	57863	11122121	30074	48.03
(4, 44)	11122111	58885	11122111	30679	47.90
(28, 20)	22212222	64620	22212222	34080	47.26
(24, 24)	22212222	66056	22212222	34932	47.12
(32, 16)	22212222	63184	22212222	33228	47.41
(40, 8)	22212222	60312	22212222	31524	47.73

Table 6.3: *Results for TP for complete testing*

Bus width distribution	(1, 47)	(4, 44)	(28, 20)	(24, 24)	(32, 16)	(40, 8)
Optimal allocation	11122121	11122111	22212222	22212222	22212222	22212222
Cores	TP	TP	TP	TP	TP	TP
c432	44779	37605	50050	50050	1945	50050
c880	187377	187377	55698	69008	187377	138426
c2670	5090603	3417327	5178381	5178381	4464549	5178381
c7552	7475138	7475138	6068577	5809305	7475138	6831883
s27	223	73	341	341	33	341
s298	10303	10303	463	463	10303	3019
s444	651	12605	18881	18881	2699	18881
s526	38519	38519	1257	1257	38519	1257
SOC	12847593	11178947	11373648	11127686	12180563	12222238

Table 6.4: *Results for TP for incomplete testing*

Bus width distribution	(1, 47)	(4, 44)	(28, 20)	(24, 24)	(32, 16)	(40, 8)
Optimal allocation	11122121	11122111	22212222	22212222	22212222	22212222
Cores	TP	TP	TP	TP	TP	TP
c432.pat	31978	27402	36088	36088	1456	36088
c880.pat	97835	97835	28209	35299	97835	71851
c2670.pat	2154776	1395630	2196792	2196792	1884964	2196792
c7552.pat	4302353	4302353	3474765	3322563	4302353	3923667
s27.pat	112	30	199	199	16	199
s298.pat	6549	6549	277	277	6549	2037
s444.pat	346	6286	9949	9949	1160	9949
s526.pat	22443	22443	663	963	22443	663
SOC	6616392	5858528	5746942	5602130	6316776	6241246

Table 6.5: *Results for comparison of TP*

Bus width Distribution	Optimal allocation	TP for SOC (complete test)	Optimal allocation	TP for SOC (incomplete test)	Reduction in TP (%)
(1, 47)	11122121	12847593	11122121	6616392	48.50
(4, 44)	11122111	11178947	11122111	5858528	47.59
(28, 20)	22212222	11373648	22212222	5746942	49.47
(24, 24)	22212222	11127686	22212222	5602130	49.66
(32, 16)	22212222	12180563	22212222	6316776	48.14
(40, 8)	22212222	12222238	22212222	6241246	48.94

6.8.3 Results For TP

TP For Complete Testing

Table 6.3 displays TP outcomes derived from experiments executed to determine TP for full testing using distinct random bus width distributions and optimal core-to-bus scheduling, as expounded in section 6.7. In the second column, TP is computed for all eight cores and the SOC, with a bus width distribution of (1,47) and core-to-bus allocation of 11122121. The TP figures for cores c880, c2670, c432, c6288, s27, s298, s444, and s520 stand at 44779, 187377, 5090603, 7475138, 223, 10303, 651, and 38519, correspondingly. For complete SOC testing, the TP results are 12847593, 11178947, 11373648, 11127686, 12180563, and 12222238 for bus width distributions of (1,47), (4,44), (28,20), (24,24), (32,16), and (40,8), in that order.

TP for Incomplete Testing

The outcomes of experiments aimed at computing TP for partial testing using diverse bus width distributions and optimal core-to-bus scheduling are displayed in Table 6.4. The calculations were executed following the procedure outlined in section 6.7.1. In the second column of the table, TP is exhibited for all eight cores and the SoC, with a bus width distribution of (1, 47) and core-to-bus allocation of 11122121. The TP values for individual cores, such as c880, c2670, c432, c6288, s27, s298, s444, and s520, stand at 31978, 97835, 2154776, 4302353, 112, 6549, 346, and 22443, correspondingly. For partial SoC testing, the TP results are 6616392, 5858528, 5746942, 5602130, 6316776, and 6241246 for various bus width distributions, specifically, (1, 47), (4, 44), (28, 20), (24, 24), (32, 16), and (40, 8), in that order.

Comparison of TP for Complete and Incomplete Testing

The TP results for both full and partial testing are showcased in Table 6.5. Column 1 and 2 exhibit the bus width distribution and optimal core-to-bus allocation, respectively. Column 3 and 5 depict the TP of the SoC for complete and partial testing, respectively. Finally, column 6 demonstrates the reduction in TP. For instance, the first row of Table 6.5 demonstrates a bus width distribution of (1,47), with the optimal core-to-bus allocation being 11122121. The TP for this optimal distribution is 12847593 for full testing and 6616392 for partial testing. In the first row, column 6 of Table 6.5, it signifies a TP reduction of 48.50% for the SoC. Correspondingly, TP is computed for other combinations of test bus width distributions. For the bus width distribution of (24,24), the savings amount to 49.66%. On average, the savings for other distributions are also approximately 49%, which is exceedingly substantial.

6.9 Conclusion and Future work

In conclusion, the proposed method represents a novel and advantageous approach for fault-tolerant applications spanning audio, video, graphics, wireless communications, multi-level cell STT-RAM, and neural networks. Particularly valuable in the context of large SoC devices where comprehensive testing faces limitations, incomplete testing emerges as a practical means to optimize testing parameters while only slightly compromising fault coverage. This approach not only accelerates testing processes but also proves cost-effective, potentially expanding customer accessibility to these devices. Our research demonstrates significant reductions of up to 52% in TDV, 48% in TAT, and 50% in TP with a trade-off of 1% to 10% in FC. Future avenues for exploration encompass extending the method to various fault types, including bridge and delay faults, as well as adapting it for 3D hierarchical SoCs. Moreover, tailored incomplete testing approaches for specific applications like image and digital signal processing, wireless communication, and other error-tolerant scenarios hold promise for further research and development.

Chapter 7

ATPG for Incomplete Testing of SoC having Bridge Faults

In the preceding chapters, the focus has been on incomplete testing of SoC devices, particularly with regard to *Stuck-at Faults*. However, SoC testing involves a range of fault types, including *Stuck-at Faults*, *Bridging Faults*, transition faults, and path delay faults. *Stuck-at Faults* manifest when a signal or circuit element remains at a constant value, while *Bridging Faults* occur due to unintentional connections between different nets or nodes, potentially causing signal interference and malfunctions.

Detecting and addressing these faults is vital for ensuring the dependable and optimal performance of SoC devices. *Bridging Faults*, also referred to as short circuits or cross-talk faults, pose a specific challenge. They can arise from manufacturing defects, material impurities, or environmental factors, making them challenging to identify and replicate during testing.

In this chapter, the *Incomplete Testing* approach is expanded to tackle *Bridging Faults*. By broadening the testing strategies to encompass these intricate defects, the goal is to improve the overall quality and dependability of SoC devices. Effective testing methods for *Bridging Faults* are essential for delivering robust and high-performance SoCs for diverse applications.

7.1 Problem Formulation

Given a SoC, generate reduced test vectors for Incomplete Testing of SoC for Bridging Faults.

7.2 Proposed Scheme : Boolean Satisfiability Method for Incomplete Testing of SoC

To generate a test pattern for a specific fault in a circuit, the first step is to create a logical formula that represents the circuit's behavior and includes the fault condition. This logical formula defines how the circuit components are connected and how the fault affects its operation. Once the formula is established, the *Boolean Satisfiability Algorithm* is applied to find variable assignments that satisfy the formula.

In the circuit, every gate aligns with an individual logical formula delineating its operation. This formula encapsulates the manner in which the inputs to the gate are combined to generate the output. Formulating expressions for each component in the circuit and taking the fault condition into account allows for the systematic determination of the test pattern that triggers the fault and discerns its impact on the circuit's behavior.

For instance, consider the circuit illustrated in Figure 7.1. To generate a test pattern for a particular fault associated with this circuit, we would construct logical formulas for each gate and then incorporate the fault condition into the overall formula. The *Boolean Satisfiability* algorithm would then be utilized to find suitable variable assignments that satisfy the formula and reveal the desired test pattern for the specific fault. This process helps ensure the proper functioning and reliability of the circuit under various conditions and potential faults.

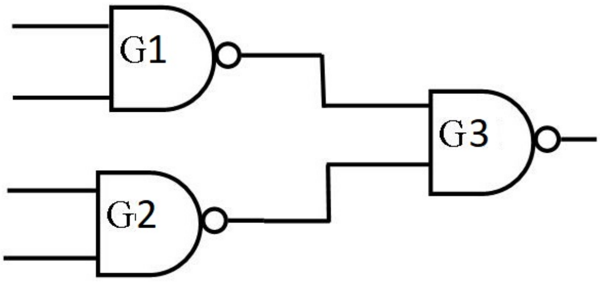


Figure 7.1

7.2.1 CNF

The *Conjunctive Normal Form* (CNF) is a specific representation used in propositional logic. It expresses a logical formula as a *Conjunction* (AND) of clauses, where each clause is a *Disjunction* (OR) of literals. In other words, it is a product of sums form. A logical formula is said to be in CNF if it is written in the form:

$$(L_{11} \vee L_{12} \vee \ldots \vee L_{1n}) \wedge (L_{21} \vee L_{22} \vee \ldots \vee L_{2m}) \wedge \ldots \wedge (L_{k1} \vee L_{k2} \vee \ldots \vee L_{kp})$$

where each L_{ij} is a literal, which can be either a variable or its negation. For example, the logical expression $(a \wedge \neg b) \vee (c \wedge d \wedge \neg e)$ can be written in CNF as $(a \vee \neg b) \wedge (c \vee d \vee \neg e)$.

For a NAND gate with inputs a and b, and output c, the logical function is $c = \overline{a \cdot b}$. In terms of logical equivalences, a logical sentence $P = Q$ is equivalent to saying $P \implies Q$ and $Q \implies P$. Thus, the output of the NAND gate, $c = \overline{a \cdot b}$, is equivalent to the expressions $c \implies \overline{a \cdot b}$ and $\overline{a \cdot b} \implies c$. By applying the propositional logic rule $P \implies Q$ is equivalent to $\overline{P} + Q$, we can write the CNF or the product of sums for the logical function as $(\bar{c} + \overline{a \cdot b}) \cdot (\overline{\overline{a \cdot b}} + c)$, which can be further simplified as $(\bar{c} + \bar{a} + \bar{b}) \cdot (c + a) \cdot (c + b)$. So in conclusion the logical function $c = \overline{a \cdot b}$ is $(\bar{c} + \bar{a} + \bar{b}) \cdot (c + a) \cdot (c + b)$. This expression is a *Conjunction* of clauses, where each clause represents a combination of literals that make the logical formula true.

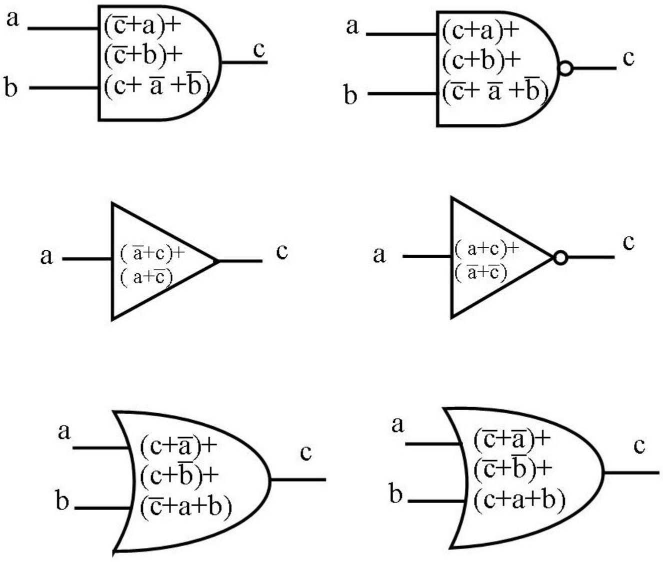


Figure 7.2: *Various Gates having CNF formula*

7.2.2 CNF Formulation for Original Circuit

In Figure 7.2, it is demonstrated that a CNF formula can be created for each logic gate, with variables representing the inputs and outputs of these gates. In this section, we have specifically examined a benchmark circuit called C17 from ISCAS 89, as depicted in Figure 7.3.

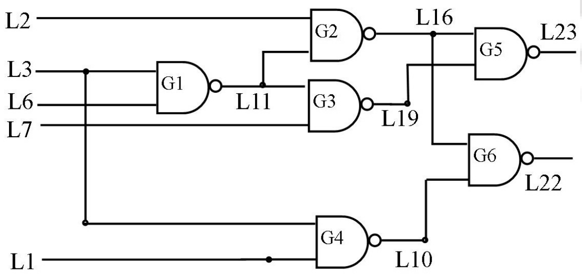


Figure 7.3: *C17 benchmark circuit*

CNF formula for C17 will be $(\overline{L23} + \overline{L16} + \overline{L19}).(L23 + L16).(L23 + L19)$. $(\overline{L16}+\overline{L11}+\overline{L2}).(L16+L11).(L16+L2)$. $(\overline{L19}+\overline{L11}+\overline{L7}).(L19+L11).(L19+L7)$.

$(\overline{L11}+\overline{L6}+\overline{L3}).(L11+L6).(L11+L3).\ (\overline{L22}+\overline{L16}+\overline{L10}).(L22+L10).(L22+L16)$ $.\ (\overline{L10}+\overline{L1}+\overline{L3}).(L10+L1).(L10+L3)$

The construction of a CNF formula for each logic gate involves backtracking from the output to the input. For every node, a CNF formula is derived, and each CNF formula must be satisfied independently for all its input and output variables. To obtain the CNF formula for the entire circuit, we take the AND of each node visited during backtracking. As each node's CNF formula is individually satisfiable, the conjunction of CNF formulas for all nodes must be boolean satisfiable, meaning it is possible to find variable assignments that satisfy the entire CNF formula of the circuit.

7.2.3 Introduction of Bridging Faults in Original Circuit

In modern circuits, the complexity and packing density have significantly increased, allowing a higher number of components to be integrated onto a single chip. Consequently, bridging faults have become a major concern in such circuit designs. A bridging fault refers to an unintended short circuit between two wires. The type of bridging fault depends on the specific technology employed in the circuit.

In circuits using *Transistor-Transistor Logic* (TTL) technology, the bridging fault is known as wired-AND, where both wires connected by the short circuit will have a logical value of 0. On the other hand, in circuits using *Emitter-Coupled Logic*(ECL) technology, the bridging fault is referred to as wired-OR, and both connected wires will have a logical value of 1. For our research, we focused solely on non-feedback bridging faults in combinational circuits.

A bridging fault can be denoted as $L_i/a_i/L_j$, where L_i and L_j represent the dominated and dominating lines, respectively. b_i signifies the logical value of 0 or 1 by which line L_j dominates line L_i. In a circuit with n lines, there can be $\binom{n}{2}$ possible shorts between two lines.

To detect these faults, we assign a value $b_i{}'$ to the dominated line L_i and b_i

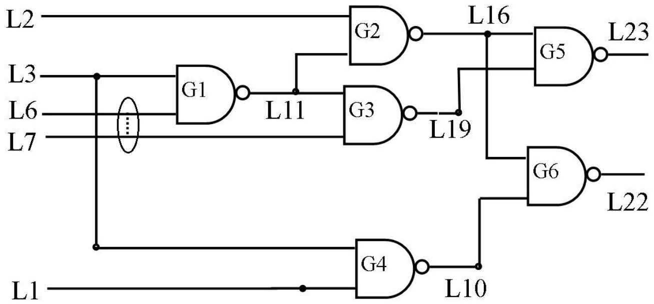


Figure 7.4: *Bridging Fault between L6 and L7*

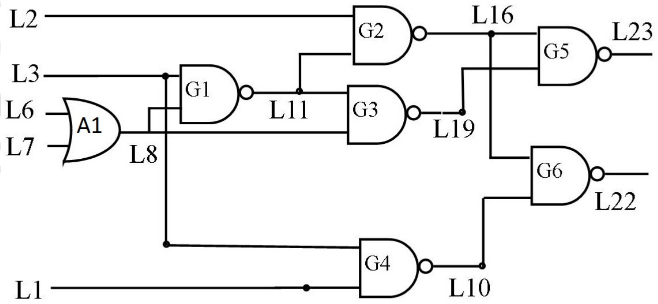


Figure 7.5: *Testing of Bridging Fault*

to the dominating line L_j. Then, we propagate the faulty and non-faulty values from L_i to the output. As a result, the test vector assigns the value a_i to line L_j and detects that line L_i is stuck at a_i.

As an example, in our circuit C17, we consider ECL technology. Thus, shorting between lines L6 and L7 results in a wired-OR fault. We introduce a bridging fault by adding OR gate A1 between lines L6 and L7, and its output will be inputted to gates G1 and G3.

The CNF formulation will be updated to account for the short between lines L6 and L7, as illustrated in Figure 7.4. The CNF for the modified circuit, shown in Figure 7.5, is as follows: $(\overline{L23}+\overline{L16}+\overline{L19}).(L23+L16).(L23+L19)$. $(\overline{L16}+\overline{L11}+\overline{L2}).(L16+L11).(L16+L2)$. $(\overline{L19}+\overline{L11}+\overline{L8}).(L19+L11).(L19+L8)$. $(\overline{L11}+\overline{L8}+\overline{L3}).(L11+L8).(L11+L3)$. $(\overline{L22}+\overline{L16}+\overline{L10}).(L22+L10).(L22+L16)$. $(\overline{L10}+\overline{L1}+\overline{L3}).(L10+L1).(L10+L3)$. $(\overline{L8}+L1+L3).(L8+\overline{L6}).(L8+\overline{L3})$. By using boolean satisfiability algorithms we shall assign the values to the variables of the CNF equation.

7.2.4 Reduction of Boolean Satisfiability Equation for Incomplete Testing

Now *Algorithm 1* in *Chapter 6* supports an incomplete method, where the test engineer determines the least significant output based on the applications required by users. To apply the algorithm, the circuit is first converted into an equivalent graph, where each gate represents a node, and the wires are represented as edges in the graph. The algorithm takes this graph and the least significant outputs as input. It then performs backtracking from the least significant output to the input.

During the backtracking process, nodes with out-degree 1 are identified, meaning these nodes do not affect other nodes. The algorithm selects a path in which each node has an out-degree less than 2. The path may not necessarily terminate at the input node; even a short path not reaching the input is considered. The bridging faults associated with these paths are left untested.

For the remaining bridging faults, the algorithm tests the ones between the lines associated with nodes having higher out-degrees, as nodes with higher out-degrees affect more nodes in the graph. For example, in Figure 7.6, the equivalent graph of circuit C17 is shown. Output 22 is considered the least significant. After applying the algorithm, a path $I1 \rightarrow G4 \rightarrow G6 \rightarrow O22$ is obtained, where each node has an out-degree less than 2. *Bridging Faults* associated with lines on this path are ignored, and faults like L1/a'/E-1, L10/a'/E, and L22/a'/E, where a' is the logical binary value and E is any other line in the circuit, are not tested.

For the remaining circuit, bridging faults among the lines associated with nodes I3, G1, and G2 having the highest out-degree of 2 are tested first. In the case of circuit C17, bridging faults like L6/a'/L, L3/a'/L, L11/a'/L, L2/a'/L, L16/a'/L, L23/a'/L will be tested first, where L represents any other lines in the circuit. These faults are more significant and have a larger impact on the circuit than others. Finally, other bridging faults on the lines associated with nodes are tested in decreasing order of out-degree according to the required fault coverage. After eliminating the path found from our algorithm the graph is shown in Figure 7.7 and the CNF of our circuit will be reduced to $(\overline{L23}+\overline{L16}+\overline{L19}).(L23+L16).(L23+L19)$. $(\overline{L16}+\overline{L11}+\overline{L2}).(L16+L11).(L16+L2)$. $(\overline{L19}+\overline{L11}+\overline{L7}).(L19+L11).(L19+L7)$. $(\overline{L11}+\overline{L6}+\overline{L3}).(L11+L6).(L11+L3)$ which will have less number of variables in it. Hence assigning values to these variables will take less testing time.

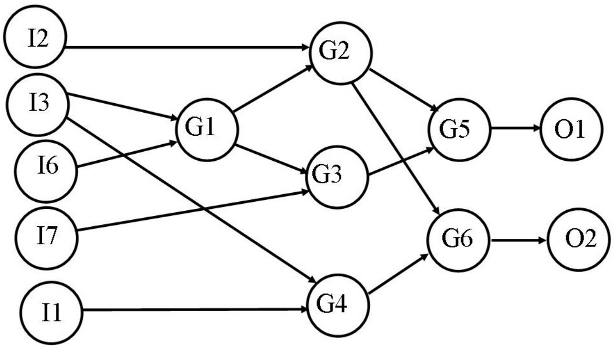


Figure 7.6: *Equivalent graph of C17*

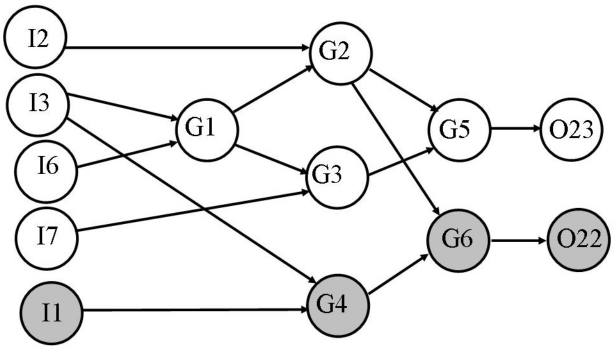


Figure 7.7: *Path from output to input not to be tested*

ALGORITHM 4: Algorithm for incomplete testing considering bridging faults

```
Input: Equivalent graph of the circuit with N nodes, L edges, M output
       left untested
Output: Bridging faults not to be tested
for nodes $i = 1$ to $N$ do
    outdegree (i);
for Output $i = 1$ to $M$ do
    backtrack (i);                                    // Algorithm 2
don't test the bridging faults associated to path obtained from backtrack
  algorithm.
sort the nodes in order of decreasing out degree.
Test the bridging faults associated with components having higher out
  degree to the lower out degree.
```

ALGORITHM 5: backtrack (m)

```
Input: Output m
Output: Gates on the path from output pin i to one of the input pins
if outdegree (parent(i)) $> 1$ || parent(i) $=$ input then
    add m to the result;
    break;
else
    backtrack(parent(m));
```

7.2.5 Satisfying the Formula

To identify bridging faults in a circuit and generate corresponding test patterns, a miter circuit is employed as illustrated in Figure 7.8. This circuit introduces faults and compares the outputs of the faulty and fault-free circuits using an EX-OR gate. To assess bridging faults, only input values that result in output discrepancies between the two circuits are necessary. The process involves constructing a CNF formula that includes XOR operations.

The final CNF formula for fault detection combines the CNF formulas for the faulty and non-faulty circuits, along with those for additional XOR gates. While several clauses in the CNF formula are shared between these circuits, these common clauses are not duplicated in the ultimate CNF equation. If the miter circuit's output is 1 for specific input assignments, it signifies discordance between the outputs of the faulty and non-faulty circuits, indicating fault detection. The corresponding assignment becomes the test vector for that specific bridging fault. If the final CNF equation is satisfiable for particular input assignments, the bridging fault is deemed detectable; otherwise, it is considered undetectable.

It's worth noting that solving *Boolean Satisfiability* problems, such as this, is challenging and falls under the NP-hard problem category. While various SAT solver tools are available, tackling larger circuits with numerous variables can lead to exponential problem-solving times, rendering solutions infeasible. To address this, the following section introduces a particle swarm-based heuristic approach for solving the satisfiability equation.

After eliminating the path found from our algorithm the CNF of our circuit will be reduced to $(\overline{L23} + \overline{L16} + \overline{L19}).(L23 + L16).(L23 + L19)\ .\ (\overline{L16} + \overline{L11} + \overline{L2}).(L16 + L11).(L16 + L2)\ .\ (\overline{L19} + \overline{L11} + \overline{L7}).(L19 + L11).(L19 + L7)\ .\ (\overline{L11} + \overline{L6} + \overline{L3}).(L11 + L6).(L11 + L3)$.

which will have less number of variables in it. Hence assigning values to these variables will take less time.

7.2.6 Heuristic Approach using Particle Swarm Optimization

A PSO based optimization technique is proposed for assigning values to variables.

Particle Structure

If we have n input variables in our circuit named as $I_1, I_2, I_3, I_4, ..., I_n$ then our particle will be n bit array where each bit can have binary value a which is either 0 or 1.

I_1	I_2	I_3	I_4	. . . .	In
a	a	a	a	.....	a

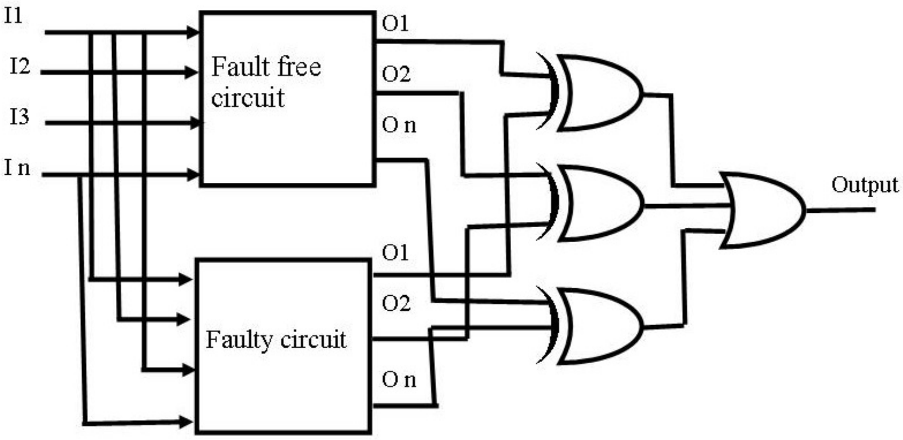


Figure 7.8: *Miter Circuit*

Suppose we have a circuit c17 where the number of inputs are 5 then our particle structure will be like

1	1	0	1	1

Initial Population and Position

For the larger circuits having more inputs, the population size has been taken as 60. Initialization of particles are random as we don't know about the circuit behavior. Initially array will have assignment of 0 and 1 randomly. Further assignment to each bit I_n will be modified as per the following equation [143] .

$$V_{id} = V_{id} + \phi(P_{id} - x_i) + \phi(P_{gd} - x_i) \quad (7.1)$$

$$where \quad (7.2)$$

$$(7.3)$$

$$\begin{cases} x_i = 1, & \text{if}(rand() < S(V_{id})) \\ x_i = 0, Otherwise & \end{cases}$$

Fitness Function of a Particle

To identify a bridging fault, we must allocate binary values of 0 or 1 to variables in such a way that the *Boolean Equation* of the *Miter Circuit*, represented in CNF, evaluates to 1. Our PSO algorithm's objective is to achieve a CNF equation for the miter circuit that equals 1. In cases where PSO fails to discover a suitable assignment resulting in CNF = 1, a traditional satisfiability solver becomes necessary. If a valid assignment is found, it implies the fault is not detectable. To streamline testing with incomplete information, a simplified CNF formulation will be employed, effectively minimizing the time and size associated with each test vector.

7.3 Experimental Results

The experiments were carried out using ISCAS 85 combinational circuits. A C++ program was employed to modify the benchmark files of these circuits. These modified benchmark files, representing both faulty and non-faulty circuit versions, were used as input for the ABC system. The ABC system then generated the CNF representation of the miter circuit, which required a SAT solver for satisfiability checking. In this study, we adopted a heuristic approach utilizing PSO. The PSO-based technique was implemented using C++. The program takes the CNF equation and input lines as inputs and provides variable assignments for the equation.

Due to the size of the circuits, testing all possible bridging faults is impractical. Therefore, random faults were chosen for experimentation. The results were compared against the findings of a previously published paper [161] in the same domain. Specifically, our experiments were confined to wired-AND and wired-OR bridging faults. The entire set of experiments focused on the SoC illustrated in

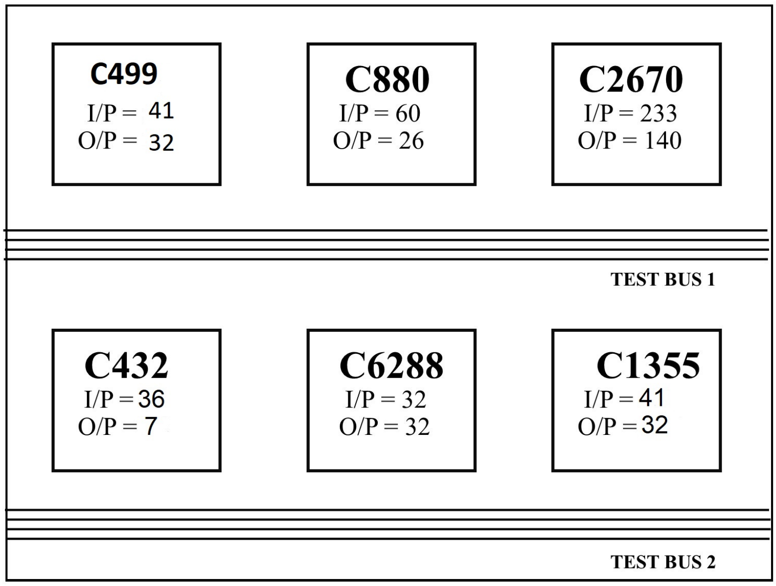


Figure 7.9: *SoC for experiments*

Figure 7.9.

For our experimental investigation, we chose a SoC comprising 6 cores from the ISCAS 85 dataset: C499, C880, C2670, C432, C6288, and C1355, along with 2 test buses. The individual characteristics of these cores are as follows: C499 with 41 input and 32 output pins, C880 with 60 input and 26 output pins, C2670 with 233 input and 140 output pins, C432 with 36 inputs and 7 output pins, C6288 with 82 input and 32 output pins, and C1355 with 81 input and 32 output pins. To apply test patterns to these cores, designated test buses were employed. The scheduling of these two buses to the cores within the SoC was optimized to ensure efficient test application, a process that can be enhanced using various scheduling algorithms.

Our experimentation concentrated exclusively on non-feedback wire-AND and wire-OR bridging faults. The findings from these tests are presented in Table 7.1, which outlines the outcomes for wire-AND bridging faults. The first column displays the core names, while the second column shows the injected faults in

Table 7.1: *Comparison of our method with the existing methods for wired-AND bridging faults*

Cores	Number of injected faults	Complete Testing		Incomplete testing	
		Fault coverage percentage	Test Vectors	Fault coverage percentage	Test Vectors
C499	499	98.39	489	92.23	446
C880	880	100	719	95.45	669
C2670	2670	99.73	2394	91.29	2134
C432	432	98.61	221	94.54	167
C6288	6288	100	6087	93.11	5986
C1355	1355	99.55	703	90.80	630

Table 7.2: *Comparison of our method with the existing methods for wired-OR bridging faults*

Cores	Number of injected faults	Complete Testing		Incomplete testing	
		Fault coverage percentage	Test Vectors	Fault coverage percentage	Test Vectors
C499	499	98.39	488	92.65	402
C880	880	100	776	96.46	745
C2670	2670	99.51	2424	93.21	2263
C432	432	100	227	94.26	198
C6288	6288	99.96	6080	94.87	5743
C1355	1355	99.77	688	93.27	607

the circuits. A comparison of fault coverage percentage and test data volume was conducted between existing methods and our newly proposed approach. The results indicate that for core C499, fault coverage reached 98.3%, accompanied by 489 test vectors. In the context of incomplete testing, fault coverage reduced to 92.23%, but the number of test patterns was also diminished to 446. Similarly, for C880, traditional methods achieved full fault coverage of 100%, utilizing 719 test vectors. In contrast, our method saw fault coverage drop to 95.45%, along with a reduction in test vectors to 669.

While there was a slight compromise in fault coverage across other cores, a substantial decrease in TDV was observed. This comes with a trade-off in product quality, as test vector numbers are significantly reduced. This reduction in test vectors not only curtails testing time and complexity but also holds potential to reduce overall testing costs.

In addition, Table 7.2 portrays our experiments concerning wire-OR bridging faults, using the same SoC. The injected fault quantities remain consistent with those in Table 7.1. Here, fault coverage percentage for C4994 achieved 98.3% through traditional methods, while our approach saw coverage decrease to 92.65%. For C499, the test vectors decreased from 488 to 402. Similarly, for C880, the traditional method led to fault coverage of 100%, while our incomplete testing yielded 96.46% FC and a reduction in test vectors from 776 to 745.

Collectively, these two tables highlight that our incomplete testing strategy results in some compromise to product quality, but this is offset by a substantial reduction in test vectors and TDV.

7.4 Conclusion and Future Work

The conclusions drawn from the wire-AND *Bridging Fault* tests highlight the trade-offs observed between fault coverage and TDV. Our methodology demonstrated an impressive fault coverage of 98.3% for core C499, utilizing 489 test vectors, whereas incomplete testing resulted in a slightly reduced coverage of

92.23% with 446 test patterns. Similarly, in the case of core C880, traditional methods achieved full fault coverage of 100% employing 719 test vectors, whereas our approach maintained a commendable 95.45% coverage with only 669 vectors. While there was a compromise in fault coverage for certain cores, this was counterbalanced by a significant reduction in TDV, leading to more streamlined testing, reduced complexity, and potential cost savings. Furthermore, experiments involving wire-OR bridging faults underscored the effectiveness of our approach, maintaining satisfactory fault coverage while substantially decreasing the number of required test vectors.

Chapter 8

Conclusions and Future Perspectives

8.1 Summarization

VLSI testing constitutes a crucial phase in product development, encompassing a broad realm of research. Products based on SoCs are ubiquitously employed in our daily routines. SoCs encompass diverse cores, I/O pins, test buses, and scan chains. The examination of these SoCs involves the application of assorted test patterns to their pins. Nonetheless, this process is accompanied by various challenges.

The primary challenge pertains to the size of test patterns, referred to as TDV. If these patterns are substantial in size, managing the associated test data becomes intricate, necessitating supplementary circuitry for storage. Another testing challenge involves an elongated TAT. The process of applying extensive test data to SoC pins consumes more time, consequently elongating the overall TAT. Another significant obstacle is the imperative to curtail TP. This power usage during testing is directly correlated to the frequency of transitions in test bits between 0 and 1, or vice versa.

Various strategies exist to address these testing predicaments. Progress in manufacturing technology has led to escalated integration density of SoCs, involving a higher component count per unit area. Consequently, traditional techniques have become prohibitively expensive, sluggish, and financially demanding.

To counteract these limitations, the notion of *Incomplete Testing* has emerged. The principal aim of this dissertation is to implement incomplete testing for SoC. This approach optimizes TDV, diminishes TAT, and mitigates TP while compromising with the FC to the certain acceptable level.

Chapter 1 introduces the fundamentals of SoC test architecture, outlines the challenges encountered in VLSI testing of SoCs, and elucidates the thesis objectives.

Chapter 2 presents an in-depth literature survey. The survey reveals that prevailing conventional methods predominantly rely on compression and compaction techniques. Various compression and compaction methods are elaborated upon. Additionally, power models aimed at reducing TP are discussed, alongside a discussion on a crucial parameter, TAT. Prominent TAT reduction strategies are also covered. As the complexity of SoC designs continues to evolve, their testing importance. The chapter surveys the domain of SoC testing to explore existing research, methodologies, and advancements. It offers insights into the challenges and solutions researchers have devised to cater to SoC testing requirements. This chapter provides a comprehensive foundation, enabling identification of gaps in existing literature and proposing innovative approaches to enhance SoC testing efficiency.

Chapter 3 concludes by introducing a heuristic technique to minimize TDV and input bits for cores, accepting a certain fault coverage compromise. Results exhibit that, with a 2% to 5% compromise in fault coverage, input bits for each core are reduced by an average of around 18%. This technique proves particularly beneficial for testing extensive SoC designs contains numerous cores, where exhaustive testing is impractical. The method provides a cost-effective and efficient approach for *Incomplete Testing* of SoCs reduces TDV on the cost of the quality of testing.

Chapter 4 proposed the incomplete testing and analyzed the impact of *Incomplete Testing* on TAT reduction. A method is proposed for optimal assignments of cores to test buses for incomplete testing. TAT equations are modified for incomplete testing. Research findings demonstrate that a 5% fault coverage compromise can reduce Input Bits per core by a substantial 14%. Additionally, TAT sees an 8% reduction.

Chapter 5 introduces a novel approach to incomplete testing that incorporates TP is presented. A modified test vector set, specifically tailored for incomplete testing, is introduced. Additionally, a design for a TAM that utilizes the IEEE 1500 test architecture to support this approach in general incomplete testing scenarios is proposed. The TP-aware incomplete testing method, characterized by the optimal assignment of cores to test buses, attains significant power savings, with only a slight compromise in fault coverage, as evidenced by experimental results showcasing a remarkable reduction in TP utilization.

Chapter 6 Proposed another novel method for *Incomplete Testing* of SoC. The strategy is particularly advantageous for large SoC devices where extensive testing may be impractical due to TAT and TP. *Incomplete Testing* optimizes parameters like TDV, TAT, and TP with minimal fault coverage compromise, leading to enhanced efficiency, cost-effectiveness, and broader customer accessibility to devices. Research indicates that incomplete testing can reduce TDV, TAT, and TP by up to 52%, 48%, and 50%, respectively, with a trade-off of 1% to 10% in fault coverage.

Chapter 7 concludes the findings from the wire-AND *Bridging Fault* tests revealed trade-offs between fault coverage and TDV. Our approach achieved a high fault coverage of 98.3% for core C499 with 489 test vectors, while *Incomplete Testing* lowered coverage to 92.23% with 446 test patterns. Similarly, for core C880, conventional methods yielded full fault coverage of 100% using 719 test vectors, whereas our approach achieved 95.45% coverage with 669 vectors. This compromise in fault coverage across some cores was offset by a notable reduction in test data volume, offering streamlined testing, decreased complexity, and potential cost savings. Experiments involving wire-OR bridging faults further demonstrated the effectiveness of our approach, maintaining satisfactory fault coverage while significantly reducing the number of test vectors.

8.2 Future Works

In the future, there are several avenues for extending our work to enhance testing efficiency. Our proposed *Incomplete Testing* approach has demonstrated its efficiency and speed, particularly in scenarios involving larger TDV, where the application of test vectors to the core consumes substantial time. Traditional testing methods for extensive SoC designs can often lead to unexpectedly long testing duration. Our proposed *Incomplete Testing* method offers significant reductions in TAT, with up to 63% time savings observed in certain cases, and an average TAT reduction of around 60%. This approach also contributes to reduced TP, with potential savings of up to 24% in specific instances. It is important to note that the methods discussed in the preceding chapters are tailored for *Stuck-at Faults* only. However, other fault types such as *Bridging Faults* and *Delay Faults* are also critical in the testing domain. An *Incomplete Testing* technique targeting *Bridging Faults*, which not only improves TDV but also provides valuable insights for further exploration. Looking ahead, there are several promising directions for future research:

1. Integration of thermal-aware techniques with *Incomplete Testing* approaches to address potential thermal challenges during testing.

2. Focus on test scheduling strategies specifically designed for cores in a 3D SoC environment.

3. Development of commercial software for *Incomplete Testing*, extending the benefits of our method to real-world applications.

4. Broadening the scope of fault coverage to include other fault types such as *Delay Faults*, enhancing the comprehensiveness of the testing approach.

5. Incorporation of power estimation techniques that consider the impact of recent manufacturing technologies on both leakage and dynamic power consumption.

6. Exploration of the application of *Incomplete Testing* concepts in other testing domains, such as digital bio-fluidic chip testing.

By venturing into these areas, this dissertation can further refine and expand the capabilities of *Incomplete Testing* methodologies, leading to more efficient and effective testing practices across a broader spectrum of fault scenarios and testing environments.

References

[1] M. Abramovichi, M. A. Breuer, and A. D. Friedman, "Digital testing and testable design," *Computer Science Press, New York*, vol. 10, pp. 379–388, 1990. [Pg.1]

[2] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits.* Springer Science & Business Media, 2004, vol. 17. [Pg.1]

[3] G. Martin and H. Chang, "System-on-chip design," in *ASICON 2001. 2001 4th International Conference on ASIC Proceedings (Cat. No. 01TH8549).* IEEE, 2001, pp. 12–17. [Pg.3]

[4] system on chip. [Online]. Available: https://en.wikipedia.org/wiki/System-on-a-chip [Pg.3]

[5] [Online]. Available: https://www.apple.com/in/newsroom/2020/10/all-new-ipad-air-with-advanced-a14-bionic-chip-available-to-order-starting-today/ [Pg.3]

[6] [Online]. Available: https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-888-5g-mobile-platform [Pg.3]

[7] [Online]. Available: https://semiconductor.samsung.com/processor/mobile-processor/exynos-2100/ [Pg.3]

[8] [Online]. Available: https://www.amd.com/en/processors/ryzen-5000-series [Pg.3]

[9] [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/series/202986/11th-generation-intel-core-i7-processors.html [Pg.3]

[10] [Online]. Available: https://developer.nvidia.com/content/tegra-x1 [Pg.3]

[11] [Online]. Available: https://www.tesla.com/support/autopilot [Pg.3]

[12] [Online]. Available: https://cloud.google.com/tpu/docs/intro-to-tpu [Pg.3]

[13] C.-W. Wu and X. Wen, *VLSI Test Principles and Architectures.* Morgan Kaufmann, 2006. [Pg.vii], [Pg.3], [Pg.5], [Pg.7], [Pg.17], [Pg.18], [Pg.26]

[14] T. W. Williams and K. P. Parker, "Design for testability—a survey," *Proceedings of the IEEE*, vol. 71, no. 1, pp. 98–112, 1983. [Pg.4]

[15] T. Willians and K. Parker, "Design for testability-a survey," *IEEE Transactions on Computers*, vol. 31, no. 1, pp. 98–112, 1982. [Pg.4]

[16] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," in *Test Conference, 1998. Proceedings., International.* IEEE, 1998, pp. 130–143. [Pg.4]

[17] L. Mostardini, L. Bacciarelli, L. Fanucci, L. Bertini, M. Tonarelli, and M. De Marinis, "Fpga-based low-cost automatic test equipment for digital integrated circuits," in *2009 IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications.* IEEE, 2009, pp. 32–37. [Pg.5]

[18] M. Radu, "Testing digital circuits using a mixed-signal automatic test equipment," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics.* IEEE, 2014, pp. 1–4. [Pg.5]

[19] A. Balasubramanian, B. Bhuva, L. Massengill, B. Narasimham, R. Shuler, T. Loveless, and W. T. Holman, "A built-in self-test (bist) technique for single-event testing in digital circuits," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3130–3135, 2008. [Pg.5]

[20] N. Das, P. Roy, and H. Rahaman, "Built-in-self-test technique for diagnosis of delay faults in cluster-based field programmable gate arrays," *IET Computers & Digital Techniques*, vol. 7, no. 5, pp. 210–220, 2013. [Pg.5]

[21] M. Nicolaidis and L. Anghel, "Concurrent checking for vlsi," *Microelectronic Engineering*, vol. 49, no. 1-2, pp. 139–156, 1999. [Pg.5]

[22] M. Nicolaidis and Y. Zorian, "On-line testing for vlsi—a compendium of approaches," *Journal of Electronic Testing*, vol. 12, pp. 7–20, 1998. [Pg.5]

[23] B. Sen, J. Das, and B. K. Sikdar, "A dft methodology targeting online testing of reversible circuit," in *2012 international conference on devices, circuits and systems (ICDCS).* IEEE, 2012, pp. 689–693. [Pg.5]

[24] Y. Zorian, "Test requirements for embedded core-based systems and ieee p1500," in *Test Conference, 1997. Proceedings., International.* IEEE, 1997, pp. 191–199. [Pg.7]

[25] E. J. Marinissen and Y. Zorian, "Challenges in testing core-based system ics," *Communications Magazine, IEEE*, vol. 37, no. 6, pp. 104–109, 1999. [Pg.7]

[26] Y. Zorian, "Today's soc test challenges," in *IEEE International Conference on Test, 2005.* IEEE, 2005, pp. 2–pp. [Pg.7]

[27] S. Goel, E. J. Marinissen, A. Sehgal, and K. Chakrabarty, "Testing of socs with hierarchical cores: common fallacies, test access optimization, and test scheduling," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 409–423, 2008. [Pg.7]

[28] M. Goetz, "System on chip design methodology applied to system in package architecture," in *52nd Electronic Components and Technology Conference 2002.(Cat. No. 02CH37345).* IEEE, 2002, pp. 254–258. [Pg.8]

[29] K. Chakrabarty, "Low-cost modular testing and test resource partitioning for socs," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 3, pp. 427–441, 2005. [Pg.8]

[30] V. Iyengar, A. Chandra, S. Schweizer, and K. Chakrabarty, "A unified approach for soc testing using test data compression and tam optimization," in *2003 Design, Automation and Test in Europe Conference and Exhibition.* IEEE, 2003, pp. 1188–1189. [Pg.8]

[31] E. J. Marinissen and Y. Zorian, "Challenges in testing core-based system ics," *IEEE Communications Magazine*, vol. 37, no. 6, pp. 104–109, 1999. [Pg.8]

[32] P. Girard, N. Nicolici, and X. Wen, *Power-aware testing and test strategies for low power devices.* Springer Science & Business Media, 2010. [Pg.9]

[33] M. Renovell and H.-J. Wunderlich, "In praise of vlsi test principles and architectures: Design for testability." [Pg.17], [Pg.18]

[34] G. Huertas, J. L. Huertas, and E. Lora-Tamayo, "Very-large-scale integration of electronic circuits." [Pg.17], [Pg.18]

[35] D. F. Barbe, *Very large scale integration (VLSI): fundamentals and applications.* Springer Science & Business Media, 2013, vol. 5. [Pg.18]

[36] Very large scale integration. [Online]. Available: https://en.wikipedia.org/wiki/Very-large-scale-integration/ [Pg.18]

[37] T. W. Williams and N. Brown, “Defect level as a function of fault coverage,” *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 987–988, 1981. [Pg.18]

[38] A. D. Friedman, M. Abramovici, and M. A. Breuer, *Digital Systems Testing and Testable Design.* Jaico, 2007. [Pg.19]

[39] J. M. Emmert, C. E. Stroud, and J. R. Bailey, “A new bridging fault model for more accurate fault behavior,” in *2000 IEEE Autotestcon Proceedings. IEEE Systems Readiness Technology Conference. Future Sustainment for Military Aerospace (Cat. No. 00CH37057).* IEEE, 2000, pp. 481–485. [Pg.20]

[40] A. K. Majhi and V. D. Agrawal, “Delay fault models and coverage,” in *Proceedings Eleventh International Conference on VLSI Design.* IEEE, 1998, pp. 364–369. [Pg.21]

[41] G. L. Smith, “Model for delay faults based upon paths.” in *ITC*, vol. 85. Citeseer, 1985, pp. 342–349. [Pg.21]

[42] I. Pomeranz and S. M. Reddy, “Transition path delay faults: A new path delay fault model for small and large delay defects,” *IEEE transactions on very large scale integration (VLSI) systems*, vol. 16, no. 1, pp. 98–107, 2007. [Pg.21]

[43] P. C. Maxwell, R. C. Aitken, and L. M. Huisman, “The effect on quality of non-uniform fault coverage and fault probability,” in *Proceedings., international test conference.* IEEE, 1994, pp. 739–746. [Pg.22], [Pg.23]

[44] R. Rinitha and R. Ponni, “Testing in vlsi: A survey,” in *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*. IEEE, 2016, pp. 1–6. [Pg.vii], [Pg.22]

[45] J. P. Roth, “Diagnosis of automata failures: A calculus and a method,” *IBM journal of Research and Development*, vol. 10, no. 4, pp. 278–291, 1966. [Pg.24]

[46] Goel, “An implicit enumeration algorithm to generate tests for combinational logic circuits,” *IEEE transactions on Computers*, vol. 100, no. 3, pp. 215–222, 1981. [Pg.24]

[47] Fujiwara and Shimono, “On the acceleration of test generation algorithms,” *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1137–1144, 1983. [Pg.24]

[48] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1989, pp. 1929–1934. [Pg.24]

[49] F. Brglez, D. Bryan, and K. Koiminski, “Combinational profiles of sequential benchmark circuits zyxwvutsrqponm.” [Pg.24]

[50] [Online]. Available: https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-hspice.html [Pg.24]

[51] E. J. McCluskey, *Logic design principles with emphasis on testable semicustom circuits.* Prentice-Hall, Inc., 1986. [Pg.24]

[52] T. W. Williams and K. P. Parker, “Design for testability-a survey,” *IEEE Transactions on computers*, vol. 31, no. 01, pp. 2–15, 1982. [Pg.24]

[53] P. Bardel, “Self-testing of multichip logic modules,” in *Proc. International Test Conference*, 1982, pp. 200–204. [Pg.25]

[54] C. E. Stroud, *A designer's guide to built-in self-test.* Springer Science & Business Media, 2005, vol. 19. [Pg.25]

[55] I. S. Committee *et al.*, "Ieee std. 1149.1-2001 ieee standard test access port and boundary scan architecture." [Pg.25]

[56] "Ieee standard testability method for embedded core-based integrated circuits," *IEEE Std 1500-2005*, pp. 1–136, 2005. [Pg.25]

[57] "Ieee standard for boundary-scan testing of advanced digital networks," *IEEE Std 1149.6-2003*, pp. 1–140, 2003. [Pg.25]

[58] [Online]. Available: https://grouper.ieee.org/groups/1500/publicationmy.htm [Pg.27]

[59] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper design for embedded core test," in *Proceedings International Test Conference 2000 (IEEE Cat. No. 00CH37159).* IEEE, 2000, pp. 911–920. [Pg.vii], [Pg.28], [Pg.29]

[60] W. Chao, W. Hong, and Y. Shiyuan, "A p1500-compliant wrapper and tam controller co-design scheme," in *2005 6th International Conference on ASIC*, vol. 2. IEEE, 2005, pp. 709–713. [Pg.29]

[61] J. Pouget, E. Larsson, Z. Peng, M.-L. Flottes, and B. Rouzeyre, "An efficient approach to soc wrapper design, tam configuration and test scheduling," in *The Eighth IEEE European Test Workshop, 2003. Proceedings.* IEEE, 2003, pp. 51–56. [Pg.29]

[62] E. J. Marinissen, R. Kapur, and Y. Zorian, "On using ieee p1500 sect for test plug-n-play," in *Proceedings International Test Conference 2000 (IEEE Cat. No. 00CH37159).* IEEE, 2000, pp. 770–777. [Pg.32]

[63] H. Chen, Z. Qi, L. Wang, and C. Xu, "A scan chain optimization method for diagnosis," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, 2015, pp. 613–620. [Pg.35]

[64] N. A. Touba, “Survey of test vector compression techniques,” *IEEE Design & test of computers*, vol. 23, no. 4, pp. 294–303, 2006. [Pg.vii], [Pg.36]

[65] A. Jas, J. Ghosh-Dastidar, M.-E. Ng, and N. A. Touba, “An efficient test vector compression scheme using selective huffman coding,” *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 22, no. 6, pp. 797–806, 2003. [Pg.37]

[66] Z. Wang and K. Chakrabarty, “Test data compression for ip embedded cores using selective encoding of scan slices,” in *IEEE International Conference on Test, 2005.* IEEE, 2005, pp. 10–pp. [Pg.37]

[67] S. Reda and A. Orailoglu, “Reducing test application time through test data mutation encoding,” in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition.* IEEE, 2002, pp. 387–393. [Pg.37]

[68] I. Bayraktaroglu and A. Orailoglu, “Concurrent application of compaction and compression for test time and data volume reduction in scan designs,” *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1480–1489, 2003. [Pg.37]

[69] S. Mitra and K. S. Kim, “Xpand: An efficient test stimulus compression technique,” *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 163–173, 2006. [Pg.37]

[70] C. Krishna and N. A. Touba, “Adjustable width linear combinational scan vector decompression,” in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No. 03CH37486).* IEEE, 2003, pp. 863–866. [Pg.37]

[71] P. Wohl, J. A. Waicukauski, S. Patel, and M. B. Amin, “Efficient compression and application of deterministic patterns in a logic bist architecture,” in *Proceedings 2003. Design Automation Conference (IEEE Cat. No. 03CH37451).* IEEE, 2003, pp. 566–569. [Pg.37]

[72] E. H. Volkerink and S. Mitra, “Efficient seed utilization for reseeding based compression [logic testing],” in *Proceedings. 21st VLSI Test Symposium, 2003.* IEEE, 2003, pp. 232–237. [Pg.37]

[73] C. Krishna and N. A. Touba, “Reducing test data volume using lfsr reseeding with seed compression,” in *Proceedings. International Test Conference.* IEEE, 2002, pp. 321–330. [Pg.37]

[74] C. Krishna, A. Jas, and N. A. Touba, “Test vector encoding using partial lfsr reseeding,” in *Proceedings International Test Conference 2001 (Cat. No. 01CH37260).* IEEE, 2001, pp. 885–893. [Pg.37]

[75] J. Lee and N. A. Touba, “Combining linear and nonlinear test vector compression using correlation-based rectangular encoding,” in *24th IEEE VLSI Test Symposium.* IEEE, 2006, pp. 6–pp. [Pg.37]

[76] A. Jas and N. A. Touba, “Test vector decompression via cyclical scan chains and its application to testing core-based designs,” in *Test Conference, 1998. Proceedings., International.* IEEE, 1998, pp. 458–464. [Pg.38]

[77] A.Chandra and K. Chakrabarty, “Reduction of soc test data volume, scan power and testing time using alternating run-length codes,” in *Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)*, June 2002, pp. 673–678. [Pg.38]

[78] A. Chandra and K. Chakrabarty, “Test data compression for system-on-a-chip using golomb codes,” in *Proceedings 18th IEEE VLSI Test Symposium*, April 2000, pp. 113–120. [Pg.38]

[79] A.Chandra and K. Chakrabarty, “Test data compression and decompression based on internal scan chains and golomb coding,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 6, pp. 715–722, June 2002. [Pg.38]

[80] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on golomb codes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 355–368, March 2001. [Pg.38]

[81] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on golomb codes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 355–368, 2001. [Pg.38]

[82] A. Jas, J. Ghosh-Dastidar, Mom-Eng Ng, and N. A. Touba, "An efficient test vector compression scheme using selective huffman coding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 797–806, June 2003. [Pg.38]

[83] S. Lu, H. Chuang, G. Lai, B. Lai, and Y. Huang, "Efficient test pattern compression techniques based on complementary huffman coding," in *2009 IEEE Circuits and Systems International Conference on Testing and Diagnosis*, April 2009, pp. 1–4. [Pg.38]

[84] Y. Yu, Z. Yang, and X. Peng, "Test data compression based on variable prefix dual-run-length code," in *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, May 2012, pp. 2537–2542. [Pg.38]

[85] Xijiang Lin, J. Rajski, I. Pomeranz, and S. M. Reddy, "On static test compaction and test pattern ordering for scan designs," in *Proceedings International Test Conference 2001 (Cat. No.01CH37260)*, Nov 2001, pp. 1088–1097. [Pg.38], [Pg.39]

[86] A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (fdr)

codes," *IEEE transactions on computers*, vol. 52, no. 8, pp. 1076–1088, 2003. [Pg.38]

[87] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici, "Variable-length input huffman coding for system-on-a-chip test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 783–796, 2003. [Pg.38]

[88] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain designs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 4, pp. 460–469, 2003. [Pg.38]

[89] L. Li, K. Chakrabarty, and N. A. Touba, "Test data compression using dictionaries with selective entries and fixed-length indices," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 4, pp. 470–490, 2003. [Pg.39]

[90] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections," in *2004 International Conference on Test*. IEEE, 2004, pp. 926–935. [Pg.39]

[91] S. Mitra and Kee Sup Kim, "X-compact: an efficient response compaction technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 3, pp. 421–432, March 2004. [Pg.39]

[92] S.Mitra and Kee Sup Kim, "X-compact: an efficient response compaction technique for test cost reduction," in *Proceedings. International Test Conference*, Oct 2002, pp. 311–320. [Pg.39]

[93] U. S. Mehla, K. S. Dasgupta, and N. M. Devashrayee, "Hamming distance based reordering and columnwise bit stuffing with difference vector: A better scheme for test data compression with run length based codes," in *2010 23rd International Conference on VLSI Design*, Jan 2010, pp. 33–38. [Pg.39]

[94] L. Li, K. Chakrabarty, and N. Touba, "Test data compression using dictionaries with selective entries and fixed-length indices," *ACM Trans. Design Autom. Electr. Syst.*, vol. 8, pp. 470–490, 10 2003. [Pg.39]

[95] T.-B. Wu, H.-Z. Liu, and P.-X. Liu, "Efficient test compression technique for soc based on block merging and eight coding," *J. Electron. Test.*, vol. 29, no. 6, pp. 849–859, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10836-013-5415-7 [Pg.39]

[96] S. Alampally, R. T. Venkatesh, P. Shanmugasundaram, R. A. Parekhji, and V. D. Agrawal, "An efficient test data reduction technique through dynamic pattern mixing across multiple fault models," in *29th VLSI Test Symposium*, May 2011, pp. 285–290. [Pg.39]

[97] S. N. Biswas, S. R. Das, and E. M. Petriu, "On system-on-chip testing using hybrid test vector compression," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 11, pp. 2611–2619, Nov 2014. [Pg.39]

[98] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test wrapper and test access mechanism co-optimization for system-on-chip," *Journal of Electronic Testing*, vol. 18, no. 2, pp. 213–230, 2002. [Pg.41]

[99] K. Chakrabarty, "Optimal test access architectures for system-on-a-chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 6, no. 1, pp. 26–49, 2001. [Pg.41]

[100] K.Chakrabarty, "Design of system-on-a-chip test access architectures using integer linear programming," in *VLSI Test Symposium, 2000. Proceedings. 18th IEEE.* IEEE, 2000, pp. 127–134. [Pg.41], [Pg.75]

[101] K. chakrabarty, "Test scheduling for core-based systems using mixed-integer linear programming," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 10, pp. 1163–1174, 2000. [Pg.41]

[102] Z. S. Ebadi and A. Ivanov, “Design of an optimal test access architecture using a genetic algorithm,” in *Test Symposium, 2001. Proceedings. 10th Asian.* IEEE, 2001, pp. 205–210. [Pg.41], [Pg.76], [Pg.84]

[103] K. Chakrabarty, “Low-cost modular testing and test resource partitioning for socs,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 3, pp. 427–441, May 2005. [Pg.41]

[104] S. Samii, M. Selkala, E. Larsson, K. Chakrabarty, and Z. Peng, “Cycle-accurate test power modeling and its application to soc test architecture design and scheduling,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 973–977, 2008. [Pg.41]

[105] P. Girard, “Survey of low-power testing of vlsi circuits,” *IEEE Design & test of computers*, vol. 19, no. 3, pp. 82–92, 2002. [Pg.42]

[106] P. Basker and A. Arulmurugan, “Survey of low power testing of vlsi circuits,” in *2012 International Conference on Computer Communication and Informatics.* IEEE, 2012, pp. 1–7. [Pg.42]

[107] S. Chakravarty and V. P. Dabholkar, “Two techniques for minimizing power dissipation in scan circuits during test application,” in *Proceedings of IEEE 3rd Asian Test Symposium (ATS).* IEEE, 1994, pp. 324–329. [Pg.42]

[108] G. Vellingiri and R. Jayabalan, “An improved low transition test pattern generator for low power applications,” *Design Automation for Embedded Systems*, vol. 21, no. 3-4, pp. 247–263, 2017. [Pg.42]

[109] V. Dabholkar, S. Chakravarty, I. Pomeranz, and S. Reddy, “Techniques for minimizing power dissipation in scan and combinational circuits during test application,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1325–1333, 1998. [Pg.42]

[110] S. Ghosh, S. Basu, and N. A. Touba, “Joint minimization of power and area in scan testing by scan cell reordering,” in *IEEE Computer Society Annual Symposium on VLSI, 2003. Proceedings.* IEEE, 2003, pp. 246–249. [Pg.43]

[111] ——, “Joint minimization of power and area in scan testing by scan cell reordering technical report: Ut-cerc-tr-nat02-1.” [Pg.43]

[112] N. Badereddine, P. Girard, S. Pravossoudovitch, A. Virazel, and C. Landrault, “Scan cell reordering for peak power reduction during scan test cycles,” in *Vlsi-Soc: From Systems To Silicon.* Springer, 2007, pp. 267–281. [Pg.43]

[113] T. Hiraide, K. O. Boateng, H. Konishi, K. Itaya, M. Emori, H. Yamanaka, and T. Mochiyama, “Bist-aided scan test-a new method for test cost reduction,” in *Proceedings. 21st VLSI Test Symposium, 2003.* IEEE, 2003, pp. 359–364. [Pg.43]

[114] S. Wang and W. Wei, “A technique to reduce peak current and average power dissipation in scan designs by limited capture,” in *2007 Asia and South Pacific Design Automation Conference.* IEEE, 2007, pp. 810–816. [Pg.43]

[115] S. Manich, A. Gabarro, M. e. Lopez, J. Figueras, P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, P. Teixeira, and M. Santos, “Low power bist by filtering non-detecting vectors,” *Journal of Electronic Testing*, vol. 16, no. 3, pp. 193–202, 2000. [Pg.43]

[116] P.-H. Wu, T.-T. Chen, W.-L. Li, and J.-C. Rau, “An efficient test-data compaction for low power vlsi testing,” in *2008 IEEE International Conference on Electro/Information Technology.* IEEE, 2008, pp. 237–241. [Pg.43]

[117] L. Whetsel, “Adapting scan architectures for low power operation,” in *Proceedings International Test Conference 2000 (IEEE Cat. No. 00CH37159).* IEEE, 2000, pp. 863–872. [Pg.43]

[118] P. Rosinger, B. M. Al-Hashimi, and N. Nicolici, “Scan architecture with mutually exclusive scan segment activation for shift-and capture-power reduction,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1142–1153, 2004. [Pg.43]

[119] A. Chandra and K. Chakrabarty, “Combining low-power scan testing and test data compression for system-on-a-chip,” in *Design Automation Conference, 2001. Proceedings.* IEEE, 2001, pp. 166–169. [Pg.43]

[120] ——, “Reduction of soc test data volume, scan power and testing time using alternating run-length codes,” in *Proceedings of the 39th annual Design Automation Conference.* ACM, 2002, pp. 673–678. [Pg.43]

[121] P. M. Rosinger, B. M. Al-Hashimi, and N. Nicolici, “Low power mixed-mode bist based on mask pattern generation using dual lfsr re-seeding,” in *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors.* IEEE, 2002, pp. 474–479. [Pg.43]

[122] J. Lee and N. A. Touba, “Low power test data compression based on lfsr reseeding,” in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* IEEE, 2004, pp. 180–185. [Pg.43]

[123] W.-B. Jone, P. Gondalia, and A. Gutjahr, “Realizing a high measure of confidence for defect level analysis of random testing [vlsi],” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 3, pp. 446–450, 1995. [Pg.44]

[124] P. Gondalia, A. Gutjahr, and W.-B. Jone, “Realizing a high measure of confidence for defect level analysis of random testing,” in *Proceedings of IEEE International Test Conference-(ITC).* IEEE, 1993, pp. 478–487. [Pg.44]

[125] W.-B. Jone and K. Tsai, "Confidence analysis for defect-level estimation of vlsi random testing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 3, no. 3, pp. 389–407, 1998. [Pg.44]

[126] L. Cheng, P. Gupta, and L. He, "On confidence in characterization and application of variation models," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2010, pp. 751–756. [Pg.44]

[127] T. K. Maiti, S. Kundu, A. Dutta, and S. Chattopadhyay, "Confidence based power aware testing," in *2012 International Symposium on Electronic System Design (ISED)*. IEEE, 2012, pp. 62–66. [Pg.44]

[128] C. Giri, S. Sarkar, and S. Chattopadhyay, "A genetic algorithm based heuristic technique for power constrained test scheduling in core-based socs," in *2007 IFIP International Conference on Very Large Scale Integration*, Oct 2007, pp. 320–323. [Pg.44]

[129] C. Giri, D. K. R. Tipparthi, and S. Chattopadhyay, "Genetic algorithm based approach for hierarchical soc test scheduling," in *2007 International Conference on Computing: Theory and Applications (ICCTA'07)*, March 2007, pp. 141–145. [Pg.44]

[130] S. Chattopadhyay and K. S. Reddy, "Genetic algorithm based test scheduling and test access mechanism design for system-on-chips," in *16th International Conference on VLSI Design, 2003. Proceedings.*, Jan 2003, pp. 341–346. [Pg.44]

[131] S. Dutt, A. Chauhan, R. Bhadoriya, S. Nandi, and G. Trivedi, "A high-performance energy-efficient hybrid redundant mac for error-resilient applications," in *2015 28th International Conference on VLSI Design*. IEEE, 2015, pp. 351–356. [Pg.11], [Pg.45]

[132] V. Rao and B. Nowrouxian, "A novel high-speed parallel multiply-accumulate arithmetic architecture employing modified radix-4 signed-

binary recoding," in *Circuits and Systems, 1996., IEEE 39th Midwest symposium on*, vol. 1. IEEE, 1996, pp. 57–60. [Pg.10], [Pg.11], [Pg.45]

[133] A. Chandrasekharan, S. Eggersglüß, D. Große, and R. Drechsler, "Approximation-aware testing for approximate circuits," in *2018 23rd Asia and South Pacific design automation conference (ASP-DAC)*. IEEE, 2018, pp. 239–244. [Pg.45]

[134] A. Chandrasekharan, D. Große, and R. Drechsler, *Design automation techniques for approximation circuits*. Springer, 2018. [Pg.45]

[135] A. Gebregiorgis and M. B. Tahoori, "Test pattern generation for approximate circuits based on boolean satisfiability," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1028–1033. [Pg.45]

[136] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 957–960. [Pg.46]

[137] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003. [Pg.46]

[138] A. M. Eltawil and F. J. Kurdahi, "Improving effective yield through error tolerant system design," in *2005 12th IEEE International Conference on Electronics, Circuits and Systems*. IEEE, 2005, pp. 1–4. [Pg.46]

[139] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, and J. Henkel, "Approximation-aware multi-level cells stt-ram cache architecture," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. IEEE, 2015, pp. 79–88. [Pg.46]

[140] C. Torres-Huitzil and B. Girau, “Fault and error tolerance in neural networks: A review,” *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017. [Pg.46]

[141] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, “Recent advances in convolutional neural networks,” *Pattern recognition*, vol. 77, pp. 354–377, 2018. [Pg.46]

[142] J. Kenndy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948. [Pg.54]

[143] J. Kennedy and R. C. Eberhart, “A discrete binary version of the particle swarm algorithm,” in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, vol. 5. IEEE, 1997, pp. 4104–4108. [Pg.54], [Pg.56], [Pg.74], [Pg.149]

[144] [Online]. Available: http://ddd.fit.cvut.cz/prj/Benchmarks/ [Pg.58]

[145] [Online]. Available: http://www.pld.ttu.ee/~maksim/benchmarks/iscas85/verilog/ [Pg.58]

[146] [Online]. Available: http://www.pld.ttu.ee/~maksim/benchmarks/iscas89/verilog/ [Pg.58]

[147] [Online]. Available: http://ddd.fit.cvut.cz/prj/Atalanta-M/ [Pg.58], [Pg.101], [Pg.131]

[148] [Online]. Available: http://ddd.fit.cvut.cz/index.php?page=download [Pg.58], [Pg.102], [Pg.131]

[149] S. N. Biswas, S. R. Das, and E. M. Petriu, “On system-on-chip testing using hybrid test vector compression.” *IEEE Trans. Instrumentation and Measurement*, vol. 63, no. 11, pp. 2611–2619, 2014. [Pg.66]

[150] J. Aerts and E. J. Marinissen, "Scan chain design for test time reduction in core-based ics," in *Test Conference, 1998. Proceedings., International*. IEEE, 1998, pp. 448–457. [Pg.76]

[151] S. Samii, E. Larsson, K. Chakrabarty, and Z. Peng, "Cycle-accurate test power modeling and its application to soc test scheduling," in *Test Conference, 2006. ITC'06. IEEE International*. IEEE, 2006, pp. 1–10. [Pg.90], [Pg.91], [Pg.92]

[152] F. Da Silva, T. McLaurin, and T. Waayers, *The Core Test Wrapper Handbook: Rationale and Application of IEEE Std. 1500TM*. Springer Science & Business Media, 2006, vol. 35. [Pg.94], [Pg.95], [Pg.96]

[153] [Online]. Available: http://ddd.fit.cvut.cz/prj/Benchmarks/ [Pg.101], [Pg.131]

[154] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–9. [Pg.10]

[155] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2012. [Pg.11]

[156] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1250–1255. [Pg.11]

[157] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 820–825. [Pg.11]

[158] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*. IEEE, 2011, pp. 346–351. [Pg.11]

[159] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *2010 IEEE international conference of electron devices and solid-state circuits (EDSSC)*. IEEE, 2010, pp. 1–4. [Pg.11]

[160] M. B. Sullivan and E. E. Swartzlander, "Truncated error correction for flexible approximate multiplication," in *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. IEEE, 2012, pp. 355–359. [Pg.11]

[161] H. Mokhtarnia, S. E. Borujeni, and M. S. Ehsani, "Automatic test pattern generation through boolean satisfiability for testing bridging faults," *Journal of Circuits, Systems and Computers*, vol. 28, no. 14, p. 1950240, 2019. [Pg.150]

[162] K. M. Singh, J. Deka, and S. Biswas, "Incomplete testing of soc," *Journal of Electronic Testing*, pp. 1–16, 2023. [Pg.182]

[163] K. Mrityunjay, S. Biswas, and J. K. Deka, "Atpg for incomplete testing of soc and power aware tam architecture," in *2018 15th IEEE India Council International Conference (INDICON)*. IEEE, 2018, pp. 1–6. [Pg.182]

[164] K. M. Singh, S. Biswas, and J. K. Deka, "Atpg for incomplete testing of soc considering bridging faults," in *TENCON 2021-2021 IEEE Region 10 Conference (TENCON)*. IEEE, 2021, pp. 323–328. [Pg.182]

[165] A. K. Yadav, K. M. Singh, and S. Biswas, "Adaptive bfs based fault tolerant routing algorithm for network on chip," in *2020 IEEE REGION 10 CONFERENCE (TENCON)*. IEEE, 2020, pp. 170–175. [Pg.182]

Biography of the Author

Personal Details

Name : **Kunwer Mrityunjay Singh**

Institute : Indian Institute of Technology, Guwahati
Assam, India Pincode 781039

Mail : `mrityunjay.jkit@gmail.com`,
`kunwer@iitg.ac.in`

Contact : 8876223463

Websites : `https://iitg.ac.in/stud/kunwer/`
`https://www.kunwermrityunjaysingh.com/`

Professional Experience

Lecturer Electronics Engineering [**Year 2016 - Present**]

Department of Technical Education

Government of Uttar Pradesh, India

Academic Details

(M.Tech + Ph.D.) Dual Degree

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati, Assam, India

B.Tech

(Electronics and Communication Engineering)

University of Allahabad, U.P., India

B.Sc.

(Physics, Chemistry, Maths)

Ewing Christian College,

University of Allahabad, U.P., India

Publication

Journal

1. K. M. Singh, J. Deka, and S. Biswas, Incomplete testing of soc," Journal of Electronic Testing, pp. 116, 2023. [Pg.182] [162]

Conference

1. K. Mrityunjay, S. Biswas, and J. K. Deka, Atpg for incomplete testing of soc and power aware tam architecture," in 2018 15th IEEE India Council International Conference (INDICON). IEEE, 2018, pp. 16. [Pg.182] [163]
2. K. M. Singh, S. Biswas, and J. K. Deka, Atpg for incomplete testing of soc considering bridging faults," in TENCON 2021-2021 IEEE Region 10 Conference (TENCON). IEEE, 2021, pp. 323328. [Pg.182] [164]
3. A. K. Yadav, K. M. Singh, and S. Biswas, Adaptive bfs based fault tolerant routing algorithm for network on chip," in 2020 IEEE REGION 10 CONFERENCE (TENCON). IEEE, 2020, pp. 170175. [Pg.182] [165]