Design and Testing of Digital VLSI Circuits using Approximate Computing

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
Sisir Kumar Jena

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

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Oct, 2023

Abstract

Several studies on the applications of Recognition, Mining, and Synthesis (RMS) have been undertaken in recent years. The tasks executed by these applications don't require a golden answer or an outstanding numerical result. Instead, they must deliver products that are acceptable or sufficient in quality. These workloads have *inherent application resilience* or the capacity to deliver acceptable results even if a significant portion of their computations are executed in an imprecise or approximate manner. Intrinsic application resilience adds a whole new level to the optimization of computing platforms. However, the belief that every computation must be conducted with the same stringent idea of accuracy continues to govern the design of computing systems. With unrelenting demand for computing performance on one side and the power requirement from technology scaling on the other, it's essential to delve into a new source of efficiency. *Approximate Computing (AxC)* is a new design method that takes advantage of the flexibility given by intrinsic application resilience to optimise hardware or software implementations that are more energy or performance efficient. Several AxC techniques have been effectively developed for system architecture, software, storage elements, arithmetic circuits, and simulation in the last decade. In this thesis, we focus on Approximate Arithmetic Circuits, particularly *Approximate Adder*, which are the result of applying AxC techniques at the hardware level, and *Approximate Testing*, which is the process of approximating the conventional test procedure.

Recent techniques in approximate adder design revolve around two important principles: (1) reducing the carry chain and (2) tolerating

inaccuracy at Least Significant Bits (LSBs). Using the first principle, a given n-bit adder is divided into a number of blocks to shorten the carry chain, and no modification is made to the Full Adders (FAs), which is the fundamental unit of an adder circuit. The next category of adders is based on the second principle. The approximation is achieved through employing Approximate Full Adder (AFA) in the LSB part of the adder circuit. The basic design principle is to partition the given n-bit adder into two segments: an inaccurate (inexact) segment and an accurate segment. The former consists of AFAs accepting the inputs from LSBs. The latter consists of conventional FAs receiving inputs from MSBs. The approximate adder design techniques discussed above are primarily applied to Ripple Carry Adder (RCA) circuits, where a carry-chain or a fundamental block such as FA is present in the design. However, complex adder designs such as Kogge Stone Adder (KSA), where no fundamental blocks or carry chain is available, require unique treatments to generate an approximation version. This thesis focuses on designing such adders using the Significance-based gate-level pruning (SGLP) technique. With this approach, a non-significant gate is identified and is removed from the actual architecture to achieve approximation. Following SGLP, the accuracy of the adder can be controlled using the error threshold provided by the designer.

Circuits that produce acceptable results can be used in applications like RMS that have error resilience qualities. To put it another way, a circuit that has a fault but nevertheless produces a decent outcome can be used in error-tolerant applications. These circuits are referred to as Acceptable Integrated Circuits (AcICs). However, when using the traditional testing process, we discovered no technique for identifying AcICs through testing. We can't overlook the fact that the faulty circuit discovered via traditional testing may generate error-free output for the majority of test patterns. This thesis proposes techniques to approximate the traditional test flow architecture (Approximate Testing) for distinguishing AcICs from rejected circuits.

The key idea is to classify the faults as benign or malignant (i.e., acceptable or unacceptable, respectively) based on an error threshold (i.e., the maximum tolerable amount of error). This classification provides two sets of faults (i.e., acceptable and unacceptable). Then, we employ an Automatic Test Pattern Generation (ATPG) system that is aware of the classification and generates test patterns only for unacceptable faults while minimizing detection of acceptable faults. The proposed approach has the considerable benefit of increasing yield. According to the proposed yield model, the effective yield gain will be between 10-20% on average.

Declaration

I certify that:

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

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 (data, theoretical analysis, figures, 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.

Sisir Kumar Jena

Copyright

Signature of Author...

Sisir Kumar Jena

Certificate

This is to certify that this thesis entitled **"Design and Testing of Digital VLSI Circuits using Approximate Computing"** being submitted by **Sisir Kumar Jena**, to 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.

....

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

....

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

Dedicated to

Late Baikuntha Nath Jena

My Father, whose blessing, love and inspiration paved my path of success

Acknowledgments

A great many 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. Santosh Biswas and Prof. Jatindra Kumar Deka 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 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 advisor, I would like to thank the rest of my thesis committee: Prof. Purandar Bhaduri, Dr. Aryabartta Sahu, and Dr. Ashish Anand, for their insightful comments and encouragement. Their comments and suggestions helped me to widen my research from various perspectives.

I 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 eduaction. I am thankful to all faculty and staff of Dept. 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.

I am thankful to my friends for supporting and motivating to overcome any problems either in work and otherwise. The countless discussions, sharing ideas has improved our research. I am also grateful to all my seniors, friends and juniors especially Pranav, Partha, Chinmaya, Debabrata, Swagat Abhijeet, Madhurima, Nilakshi, Shrestha, Subrata, Arunangshu, Palash, Aparajita, Vasudevan, Bhale, and many others for their unconditional help and support. You made my life at IIT Guwahati a memorable.

Most importantly, none of this would have been possible without the love and patience of my family. I want to thank my parents, Lipun and Duggu for being a constant source of love, concern, support, and strength all these years.

Contents

List of Figures

List of Algorithms

List of Tables

List of Acronyms

AcIC	Acceptable Integrated Circuit
AFA	Approximate Full Adder
AMA	Approximate Mirror Adder
ATPG	Automatic Test Pattern Generation
AxBA	Approximate Block Adder
AxC	Approximate Computing
ACSA	Approximate Carry Skip Adder
AxIC	Approximate Integrated Circuit
AxPA	Approximate Pruning Adder
AxSA	Approximate Segment Adder
BKA	Brent-Kung Adder
CLA	Carry Lookahead Adder
CUT	Circuit Under Test
CNN	Convolutional Neural Networks
CSPA	Carry Speculative Adder
DNN	Deep Neural Networks
DSP	Digital Signal Processing
ETAII	Error Tolerant Adder Type II

ExIC	Exact Integrated Circuit
FaIC	Faulty Integrated Circuit
FA	Full Adder
HCA	Han-Carlson Adder
IoT	Internet of Things
KSA	Kogge Stone Adder
LFA	Ladner-Fischer Adder
LSB	Least Significant Bit
LOA	Lower-part-OR Adder
ML	Machine Learning
MA	Mirror Adder
MSB	Most Significant Bit
RCA	Ripple Carry Adder
RMS	Recognition, Mining, and Synthesis
SBER	Significance based Error-rate
SGLP	Significance-based gate-level pruning
SCSA	Speculative Carry Select Addition
TDV	Test Data Volume
VLSA	Variable Latency Speculative Addition
WCE	Worst Case Error

Chapter 1

Introduction

In recent years, application domains like the Internet of Things (IoT), Machine Learning (ML), and Big Data Analytics require intensive computations. Industries started building denser chips to meet these applications' computational needs and performance requirements. However, keeping a constant power requirement and fulfilling the computational need is not possible at the same time. Large computations require massive power. The advancement in technology scaling pushes the operating voltage beyond its threshold, leading to a thermal violation and conceivably damaging the chip. The existing cooling solutions such as dark silicon [1], Thermal Design Power [2], Dynamic Voltage and Frequency Scaling (DVFS) [3], Power Gating (PG) [4], near threshold computing [5], and adaptive scheduling [6] limits the power requirement but fails to achieve the rate of performance improvement. Therefore, it is necessary to substantially or entirely stray from the existing architecture and devise a new solution to fulfill the computational need and achieve performance improvement.

The Approximate Computing (AxC) paradigm is emerging as a new architecture to deliver better solutions in terms of energy gain, improved performance by trading off the accuracy of the result [7]. Approximate computing technique is driven by several error-resilient applications like multimedia, signal processing, machine learning, and robotics. Based on the input data and nature of compu-

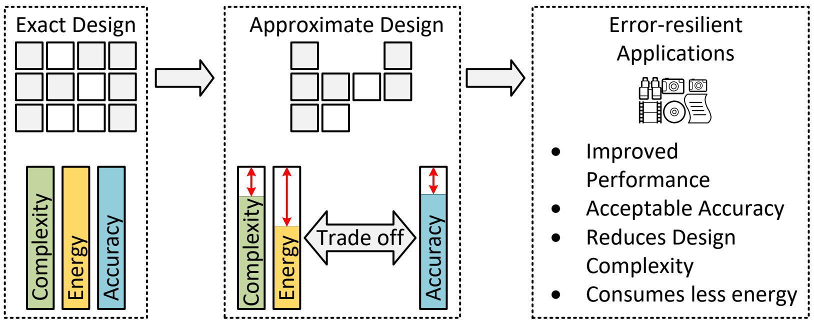


Figure 1.1: *Approximate computing philosophy*

tations, these applications do not demand exactness in the output. There are several factors involved as a reason for this error resiliency [8]. (1) Users' perceptual limits, in which a slight inaccuracy in visual data goes unnoticed by users due to their psycho-visual limitations. (2) There is no such thing as a "golden solution" when numerous results for a given input are equally acceptable. (3) Applications are said to be resilient to input noise when it produces acceptable outputs, even in the presence of natural input noises. AxC is one such paradigm that takes advantage of an application's error resistance to improve the system's overall resource efficiency. Figure 1.1 shows the philosophy of AxC. An exact design always produces accurate results, consumes sufficient energy, and is complex. According to AxC's philosophy, managing with a slight reduction in the result's accuracy saves a lot of energy and reduces complexity. AxC techniques provide solutions to build less complex designs, which creates a trade-off between the accuracy of the result with complexity and energy efficiency. Error resilient applications can be built using these approximate circuits to have several benefits like (1) performance improvement, (2) reduction in design complexity, and (3) consuming less energy during operations.

This chapter presents some background information about the work discussed in this thesis. The discussion begins with a brief overview of the areas in which approximate computing techniques are used and a review of several features of

Table 1.1: *Categorization of Approximate Computing Techniques*

Approximate Computing Research	**Approximate Computing Techniques**
Approximate Arithmetic Circuits	Approximate adder [9–12] Approximate multiplier [13–15] Approximate divider [16,17]
Approximate Storage	Voltage overscaling [18] Refresh rate reduction [19,20] Inexact read/write [21,22]
Approximate systems	Approximate accelerator [23,24] Programmable processors [25]
Software Approximation	Loop perforation and Loop Unrolling [26,27] Precision scaling [28,29] Memoization [30,31]
Hardware/Software Codesign	Frameworks [32] Compilers [33]
Approximate Computing for Security	Bitcoin mining [34] Information Hiding [35,36] Post Quantum Cryptography [37,38]
Approximate AI System Design	Cross layer approximation [39,40]

AxC. Further, it discusses Approximate Integrated Circuit (AxIC) in particular, which is the focus of our research. The testing aspect of digital VLSI circuits is also discussed and how the approximation computing paradigm aids in their improvement. In other words, the discussion focuses on two independent research areas of approximate computing in this thesis: *approximate arithmetic circuit design* and *approximating test methodologies* for hardware integrated circuits.

1.1 Approximate Computing Research Areas

Several approximation techniques have been effectively developed for system architecture, software, storage elements, arithmetic circuits, and simulation in the last decade. Table 1.1 shows seven different research areas of approximate computing techniques. The main motive of this classification is to show the vast application area of approximate computing techniques. This section provides a brief overview of these classifications.

1. *Approximate Arithmetic Circuit Design*: In this, the researchers propose a

simplified design of existing arithmetic circuits (such as addition, multiplication, or division) that produce approximate results close enough to the accurate result [9–17]. The main objective of modifying the circuit is to reduce energy consumption during operation. Other benefits include the gain in area and delay.

2. *Approximate Storage*: The accelerating power consumption and performance cost caused by unavoidable memory refresh instructions are the driving force behind approximate storage. Several approximation techniques like voltage overscaling [18], refresh rate reduction [19, 20], inexact read/write [21, 22] was proposed to reduce a significant portion of the system's power consumption.

3. *Approximate Systems*:To achieve approximation at the system level, one can design an approximate accelerator or programmable processor. Works in [23, 24] focus on the approximate accelerator, and [25] concentrate on programmable processor designs.

4. *Software Approximation*: In software, the approximation is realized using techniques such as loop perforation [26, 27], precision scaling [28, 29], and memoization [30, 31]. Loop perforation is achieved by skipping some iterations of a loop, whereas memoization replaces the computation with previously computed results.

5. *Hardware/Software Codesign*: The majority of approximation computing research concentrates on either software or hardware. In literature, there are several articles that show a coordinated architecture using both software and hardware to generate effective, high-performance, and reliable outcomes. A hardware/software codesign method in [32] and [33] is proposed for frameworks and compilers, respectively.

6. *Approximate Computing for Security*: Several approximation techniques are developed to strengthen cryptographic and hardware security to secure

a system [41]. Techniques such as bitcoin mining [34] and post-quantum cryptography [37, 38] benefit traditional cryptographic primitives. On the other hand, techniques such as information hiding [35,36] benefits hardware security.

7. *Approximate AI System Design*: The use of approximation computing to Deep Neural Networks (DNN) is ideal since DNN have a lot of redundancy and can tolerate errors. Currently, several articles focus on designing a complete AI system build using cross-layer approximation techniques [39, 40]. Some works are also used the approximation technique to reduce the computations involved in Convolutional Neural Networks (CNN) architectures [42].

Besides the categories explained above, one major challenge is automatically synthesizing approximate circuits without depending on the designer's skill. It is also essential to develop techniques that synthesize approximate circuits from the given high-level descriptions, such as C or behavioral Verilog. In [43], the author reviews these techniques in detail. *Approximating circuit testing* is another intriguing topic that has recently been investigated. Works in literature [44–46] investigates a survey of testing procedures for approximation integrated circuits.

This thesis explores two research areas where the performance of existing methodologies is improved by employing approximation techniques: (1) Approximate Adder Design (2) Approximating Testing. There have been several approximation techniques proposed so far in approximate adder design. However, those are ad-hoc and are not systematic. There is a need to improve the approximate adder circuits' design approach. Therefore, the first contribution of this thesis focuses on proposing an efficient, systematic design procedure to produce an approximate adder. The proposed technique uses the idea of probabilistic pruning, where controlling the energy-accuracy trade-off is much easier compared to previous approaches. The process is more elegant and easier to implement because of its iterative nature. The technique removes a circuit element in every iteration and calculates its output deviation. The method stopped when the variation in the output reached to required threshold. The other contribution of this thesis is approximating the test flow architecture. The idea of approximating the test flow architecture relies on two key questions: (1) Can we test an IC approximately? - testing an IC that avoids 100% fault coverage (2) Is there a need to modify the approximate test flow architecture to test an AxIC?. This dissertation discusses the solution to the above questions and proposes several modifications to the traditional test method.

1.2 Motivation and Research Focus

From the discussion in Section 1.1, it is pretty evident that AxC is now a widely used method to improve performance in several areas. This section provides a detailed explanation of the primary focus of this thesis. Mainly, it explores two dimensions of approximation: (1) Digital VLSI Circuit Design and (2) Digital VLSI Circuit Testing. Figure 1.2 shows an elaborated description of our research focus. The significant contribution of this thesis is to propose approximation techniques for circuit design and testing to improve their performances.

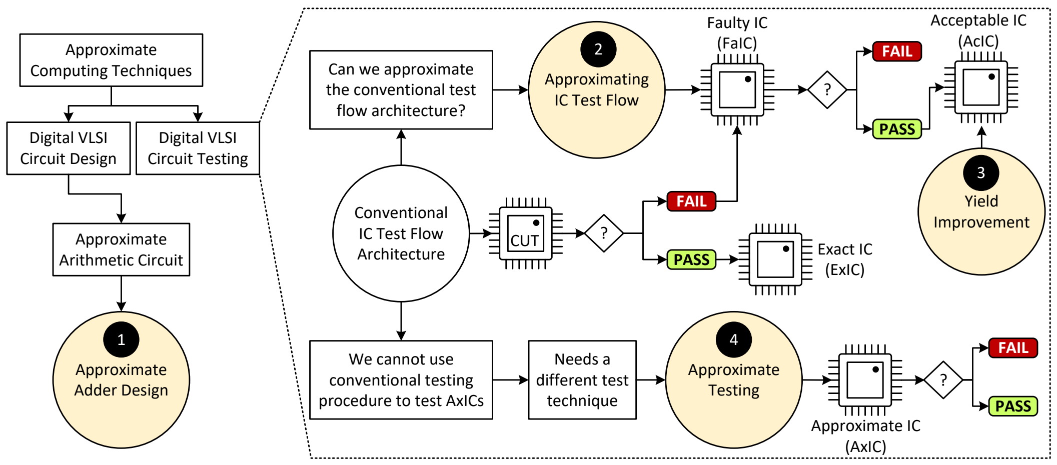

Approximate Computing Techniques
Digital VLSI Circuit Design
Digital VLSI Circuit Testing
Approximate Arithmetic Circuit
1
Approximate Adder Design
Can we approximate the conventional test flow architecture?
2
Approximating IC Test Flow
Faulty IC (FaIC)
?
FAIL
PASS
Acceptable IC (AcIC)
3
Yield Improvement
Conventional IC Test Flow Architecture
CUT
?
FAIL
PASS
Exact IC (ExIC)
We cannot use conventional testing procedure to test AxICs
Needs a different test technique
4
Approximate Testing
Approximate IC (AxIC)
?
FAIL
PASS

Figure 1.2: *Research Focus*

Arithmetic circuits such as adders, multipliers, dividers, and subtractors are the basic building blocks of any digital computing system. The basic building blocks of any digital system constitute mostly standard arithmetic operations such as adder, subtractor, multiplier, and divider. Figure 1.3 (a) shows that typical benchmarks contain 25% of such operations. However, 90% of ALU energy consumption occurs due to this 25% of arithmetic operations. Figure 1.3 (b) shows the ALU energy consumption percentage of several benchmarks. Therefore, redesigning these arithmetic circuits using approximate computing techniques helps reduce energy consumption.

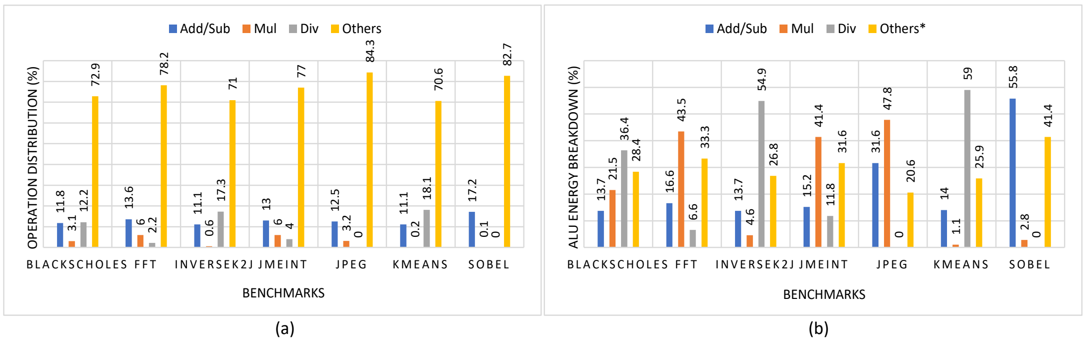


Figure 1.3: *ALU Energy Consumption with respect to basic Basic Arithmetic Operations (a) Number of Operations in Benchmarks (b) ALU Energy Consumption [47]*

The addition is the most commonly utilized operation, whether on a general-purpose system or a complex Digital Signal Processing (DSP) unit. A complex system's performance is significantly improved by an efficient adder design. It is no surprise that adders have gotten a lot of attention from researchers, and as a result, computer architects have several adder designs to choose from. Of course, they want to employ the best adder. The major challenge that was always a problem with the adder was the area, power, and delay. First, let us talk about two incompatible terminologies: delay and area, enhancing one need sacrificing the other. For instance, consider a RCA; n FAs are connected in series. In the worst case, the carry of the rightmost FA may propagate to the extreme left FA. Therefore, the delay of an n-bit RCA is almost $O(n)$. RCA is the simplest adder that takes less area than the adder, such as KSA. KSA is faster with logarithmic delay than RCA but takes a large area ($O(n \log_2 n)$). Similarly, when we consider power and area, they are proportional to each other. In summary, delay and power consumption also increase when bit-width increases [48]. One solution to this is to use an approximate computing technique to redesign the conventional adders, that improve performance. As discussed above, the approximate computing technique is driven by error-tolerant applications. Hence, due to the error-tolerant nature of several applications and the idea of approximate computing, several adder designs have been proposed that consume less energy, area, and delay. On the other hand, these adders compromise on the accuracy of the result. However, that does not impact the overall performance because they are assumed to be used in error-tolerant applications. Previous techniques on designing approximate adders are mainly grouped into three categories. The first category is reducing the carry-chain, which is the most common method followed in [11, 49–52]. With the second category, the designer modifies the fundamental FA block and deploys it to the Least Significant Bit (LSB) bit of the conventional adder. It is assumed that errors in the LSB bit do not produce a catastrophic result [53–55]. The third category is based on the probabilistic pruning technique [56, 57].

The first contribution of this thesis marked ❶ in Figure 1.2 presents an approximation technique to reduce the size of the adder circuit through the gate-level pruning method. The non-significant gates have been identified and removed from the original adder circuit to reduce the size. Due to which the power and delay of the circuit get reduced.

The second contribution of the thesis is on modifying the conventional test flow architecture. There are multiple benefits as well as a necessity for this modification. Before the discussion proceeds, let us introduce three circuit terminologies: (1) *Exact Integrated Circuits (ExICs)*, (2) *Approximate Integrated Circuits (AxICs)*, and (3) *Acceptable Integrated Circuits (AcICs)*. ExICs are conventional circuits that follow state-of-the-art circuit design and testing techniques.

Table 1.2: *A Comparison table that describes about ExICs, AxICs, and AcICs*

Headings	ExICs	AxICs	AcICs
Definition	These are conventional circuits and follow the standard design procedure, pass all tests, and finally produce the IC's semantics.	Analyze and redesign any conventional IC to improve performance by reducing area, energy consumption, and delay while sacrificing the accuracy of the result.	Refers to the ExICs that are rejected during testing and found to be acceptable because the circuit's error is not catastrophic.
Design Procedure	Follows the conventional design procedure.	It was redesigned from an existing architecture by following a traditional design technique.	No design procedure is needed because these circuits are the result of retesting techniques.
Testing Technique	Follows the conventional testing technique.	Requires specialized test techniques [44] because errors are intentional in these circuits. Therefore testing procedure must ignore these errors during testing.	Requires new test techniques to identify acceptable rejected ICs. Some literature refers to this as threshold testing.

After completing the test application process, the resultant circuit that passes all tests is called ExIC. Note that the conventional circuits that fail during the manufacturing test are called *Faulty ICs (FaICs)*. The second category (AxICs) is redesigned from the existing fundamental circuits with an objective to reduce design complexity in terms of area, power, and delay. These circuits produce an incorrect result; however, the results are acceptable. The third category, i.e., AcICs, is identified by retesting FaICs. Like, ExICs these circuits produce an incorrect output but are acceptable. A brief comparison of these three circuits is given in Table 1.2.

To understand it further, consider a 2-bit exact multiplier circuit (ExIC) shown in Figure 1.4 (a) and its corresponding AxIC in Figure 1.4 (b) [58]. Figure 1.4 (c) shows the same circuit with a stuck-at-0/1 fault at net i, and Figure 1.4 (d) shows the circuit with stuck-at-0/1 fault at net s. Now, consider testing these circuits by using a single Stuck-at fault model. Before that, look at the truth

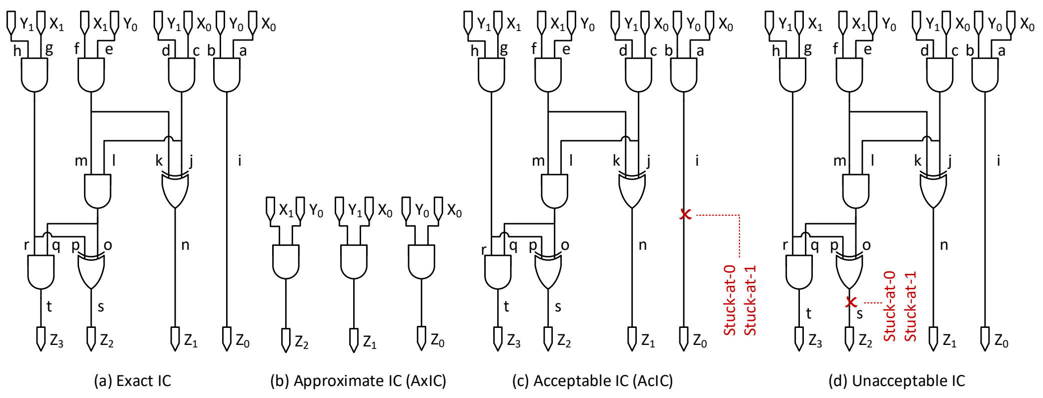


Figure 1.4: *Circuit Categories (2-bit Multiplier Circuit)*

table analysis of all these circuits shown in Table 1.3. All the inputs and outputs are represented with their integer equivalent. The first (X) and second (Y) column shows the integer equivalent of all the input patterns. The third and fourth column shows the output produced by the exact circuit and the approximate circuit, respectively. A predefined threshold value $\delta = 2$, originally decided by [58], is considered here to analyze the output produced by the AxIC. A *threshold* is defined as the absolute integer amount by which the output of a circuit may deviate and considered acceptable. As shown in column 4, the output of AxIC deviates from its correct result at four places marked as a $\oplus$ symbol. However, the absolute difference is 2 in all cases, which is within the threshold (δ) defined above. Hence, the outputs are acceptable. We use the $\oplus$ symbol to indicate an output which is wrong but acceptable, and the ✓symbol indicates the correct output. The fifth and sixth column shows the output of the acceptable circuit (AcIC) in the presence of a stuck-at-0 (Sa0@i) and stuck-at-1 (Sa1@i) fault at net i, respectively. Here we can notice the output at several places is wrong but acceptable, annotated with a $\oplus$ symbol. That is why the circuit is known as an acceptable circuit. According to the definition defined above, an AcIC may not produce the correct result but still be usable in fault-tolerant applications like image processing. Similarly, the seventh and eighth column shows the output of the unacceptable circuit (UnAcIC) in the presence of a stuck-at-0 and stuck-at-1 fault at net s, respectively. Here we can mark the output at several places are wrong, annotated with a $\times$ symbol, and the deviation in the output is more than the threshold defined above. That is why the circuit is known as an unacceptable circuit. A $\times$ symbol here indicates a wrong and unacceptable result.

A conventional IC test procedure is used to determine whether the Circuit Under Test (CUT) behaves as per the specification or not [59]. The inputs that are supplied during the test procedure is known as *test vector* or *test patterns*, and the entire collection of test patterns is called a *test set*. After applying the test patterns, the CUT's response is recorded and compared with the golden response using a test response analyzer. If the response matches, the CUT is considered"pass"; otherwise, it is faulty. The *golden response* is referred to as the expected response of the CUT. The circuit's quality depends on the test patterns or the test set. A *test-generation method* is a process that helps the tester to generate the test patterns. A brief description of the conventional test flow architecture is discussed in Chapter 2.

One fundamental question is; is it true that the FaICs are not usable at all? A conventional test flow architecture follows a sequential application of test patterns. Test patterns are applied one after another, and the result is checked for correctness. The test flow stops when it produces a wrong result without applying the remaining test patterns. However, there is a possibility that the CUT may produce the correct output for the leftover test patterns. Again, the pattern for which the CUT produces a wrong result may not have much impact on the overall output. The question creates a motivation to retest the FaICs and find how many of them are acceptable.

Retesting a FaIC requires a new test flow architecture because the conventional test technique is not designed to ignore a fault present in CUT. This raises to think, "can we approximate the conventional test flow architecture?". Using the term "approximation" here is to indicate the relaxation requirement of the retesting process. Rather than testing for all faults present in a circuit, we can

Table 1.3: *Truth Table Analysis of 2-bit Multiplier Circuit*

X	Y	ExIC	AxIC	AcIC		UnAcIC	
				Sa0@i	Sa1@i	Sa0@s	Sa1@s
0	0	0	0✓	0	1⊕	0✓	4×
0	1	0	0✓	0	1⊕	0✓	4×
0	2	0	0✓	0	1⊕	0✓	4×
0	3	0	0✓	0	1⊕	0✓	4×
1	0	0	0✓	0	1⊕	0✓	4×
1	1	1	1✓	0⊕	1✓	1✓	5×
1	2	2	2✓	0	3⊕	2✓	6×
1	3	3	3✓	2⊕	3✓	3✓	7×
2	0	0	0✓	0	1⊕	0✓	4×
2	1	2	0⊕	2	3⊕	2✓	6×
2	2	4	4✓	4	5⊕	0×	4✓
2	3	6	4⊕	6	7⊕	2×	6✓
3	0	0	0✓	0	1⊕	0✓	4×
3	1	3	1⊕	2⊕	3✓	3✓	7×
3	2	6	6✓	6	7⊕	2×	6✓
3	3	9	7⊕	8⊕	9✓	9✓	13×

Absolute Error Threshold $|\delta| = 2$

✓: Correct ⊕: Wrong but Acceptable ×: Not Acceptable

ExIC: Exact IC AxIC: Approximate IC

AcIC: Acceptable IC UnAcIC: Unacceptable IC ignore some. The approach of relaxing the test flow is known as *approximate testing* [60]. A fault modal is decided in a conventional test procedure, and then test patterns are generated using an Automatic Test Pattern Generation (ATPG) algorithm for all faults. Approximate testing relaxes in terms of generating test patterns. It generates a test pattern only for those faults whose effect is disastrous. The major challenge is to develop techniques that help identify those catastrophic faults and again generate test patterns using a new or an old ATPG. In this thesis, our first contribution towards circuit testing revolves around the challenges described above and is marked as ❷ in Figure 1.2. A similar challenge is also posed by approximate circuit design and manufacturing processes. The test engineers must be careful while testing an AxIC because distinguishing actual defects (either caused during design or manufacturing) from what is being approximated becomes more challenging, as design/manufactured defects may result in very similar variations in results. Therefore, a different test procedure is needed to test AxICs, shown in Figure 1.2 marked ❹.

Several rejected circuits (faulty ICs) will be found acceptable by approximating the test technique. Due to which there is an improvement in the yield occurs in the entire manufacturing process. Therefore, there is a requirement to propose a yield model to analyze the yield gain. The second contribution to circuit testing is to design a yield model marked ❸ in Figure 1.2.

1.3 Scope and Objective of the Thesis

The primary objective of the thesis is to explore the usage of approximate computing techniques in the area of digital VLSI design and testing. Mainly, the study focuses on two areas: (1) proposing a universal approximation technique to design an approximate adder and (2) approximating the conventional test flow architecture to improve the manufacturing yield.

To achieve our first objective, we learned about the internal architecture of several arithmetic adder circuits. We explored different proposed methods that can be used to generate an approximate version of the given adder. Note that arithmetic adder circuits are the fundamental building block of any kind of circuit. Several adders exist like RCA, KSA, Brent-Kung Adder (BKA), and Ladner-Fischer Adder (LFA), for which researchers propose approximation techniques to generate approximate adders. Techniques proposed so far are adder specific, and the methods developed for one type of adder may not help generate approximate adders of other types. For instance, a technique developed for generating an approximate adder from a given RCA circuit may not help generate an approximate adder from a given KSA circuit. Therefore, a universal approximation technique is required that can help generate an approximate adder for any existing adders.

The second objective builds around two fundamental questions based on the error-resilience property of several applications.

1. Can we approximate the conventional test flow architecture to test a circuit approximately?

2. Can we reuse the failed circuits (rejected during conventional testing) in error-tolerant applications if that circuit produces an approximate result?

We explored the process flow of conventional testing and fault modeling to achieve the second objective. Due to the decrease in the feature size, several transistors are fabricated in a small area, increasing the circuit's complexity. The circuit's complexity leads to several defects, and the test engineers must test it adequately to ensure its correct functionality. However, rigorous testing is also not useful for yield enhancement. Hence, relaxing the test process sometimes helps in improving the yield. Therefore, a different testing methodology is required to relax the test process with the objective of reducing test application time and energy consumption and, most importantly, improving the yield.

1.4 Major Contribution of the Thesis

As part of the research work on approximate computing, the contributions explore two research areas: (1) approximate adder design and (2) approximating digital VLSI test flow architecture - approximate testing. The following sections discuss these contributions.

1.4.1 Contribution on Approximate Adder Design

Approximate adder design is the most explored area in the approximate computing paradigm. A basic adder is a collection of FAs connected in series. The addition starts from the rightmost FA produces a sum and a carry (if any). The carry propagates through the series of connections from LSB to MSB and finally delivers the result. Most ideas proposed so far are either based on reducing the carry propagation or ignoring the output impact of LSB bits. However, these techniques are only applicable to traditional circuits like RCA. Further, this technique makes no area reduction possible though it produces an approximate result. The first contribution of this thesis is to propose a systematic approach to design an approximate adder. The following section explains the first contribution.

1.4.1.1 Systematic Design of Approximate Adder Using Significance Based Gate-Level Pruning (SGLP)

It is well known that approximation techniques are applied to conventional adders to improve their performance. Existing conventional adders are either built using a chain of FAs such as RCA or follows a complex architecture like KSA. Techniques developed to design approximate adders are mainly applied to RCA circuits. The approaches that are proposed to approximate the RCA circuit are not suitable for complex architecture like KSA [11, 49–55]. Therefore there is a requirement to propose a technique to approximate the complex architectures like KSA, Brent-Kung Adder (BKA), Ladner-Fischer Adder (LFA), and Han-Carlson Adder (HCA). Unfortunately, a single contribution exists in the literature towards approximating these complex architectures [56]. Our first contribution is based on Significance-based gate-level pruning (SGLP) technique where a non-significant gate is identified and is removed from the actual architecture to achieve approximation. Using SGLP, the result's accuracy of an adder circuit is restrained and can be kept below the given threshold. Here threshold refers to the allowable range of the deviated output from the original one. Interestingly, our approach is applicable to approximate both simple and complex architectures. To approximate an RCA, the procedure follows a two-step approach. First, it prunes the non-significant gates of the fundamental FA block to get an Approximate Full Adder (AFA) and then use those AFAs to generate the LSB bit of the RCA circuit. Similarly to approximate complex architectures like KSA, we follow an identical approach described in [56]. However, the technique proposed in [56] is not suitable for approximating simple architectures like RCA.

Not only design but testing is also plays a significant role in overall circuit performance and time to market. Therefore, modifying conventional test flow architecture is a requirement for testing AxICs. Another benefit of altering the test architecture is to test traditional ICs approximately. Thus the following contribution is focused on proposing approximate test techniques.

1.4.2 Contribution on Approximate Testing

Contribution towards approximate testing begins with the proposal of avoiding 100% fault coverage. Rather than testing all faults, the idea is to test the malignant defects whose impact is catastrophic. In other words, avoid generating the test patterns for all identified faults that automatically reduce the number of test patterns. Further, it also took less time to apply these test patterns. The following section discusses the contributions towards approximate testing.

1.4.2.1 Approximate Testing of Digital VLSI Circuits using Error Significance based Fault Analysis

A conventional test procedure follows a unique simple rule of achieving 100% fault coverage during a circuit test. Therefore, it is mandatory to generate test patterns for all faults identified earlier during test generation. That means to attend a high test quality, a massive amount of test patterns are generated, which increases the Test Data Volume (TDV). Applying such voluminous test patterns requires more time and thus consumes more power. Therefore several works in the literature focus on reducing the test data volume through compression techniques. This contribution introduces the term "approximate testing, " which relaxes the requirement of 100% fault coverage. The basic idea is to identify faults having a catastrophic effect and generate test patterns for them only. The remaining faults are left untested, and no test patterns are generated.

1.4.2.2 Retesting of Rejected Circuits using Approximation Technique

Error-resilient applications like image processing, machine learning, and speech recognition can use circuits that may not deliver the exact result. To put it another way, a circuit that has a fault but nevertheless produces a decent outcome can be used in error-tolerant applications. These circuits are referred to as Acceptable Circuits (AcICs). However, when using the traditional testing process, we discovered no technique for identifying AcICs through testing. Traditional test procedure checks whether the circuit is good or bad; it never checks for acceptability. However, some bad (faulty) circuits may produce an erroneous result for k input patterns out of n input patterns, and k may be much less than n. Again, it may happen that the k input pattern for which the circuit yields erroneous output may not be catastrophic. Therefore, we have proposed a technique to identify the circuit's acceptability in this contribution. The idea is to continue applying the test patterns even if it produces the wrong output and records the deviation from the actual output. Finally, quantify this deviation and compare it with the allowable threshold decided previously. If the variation is within the given threshold, the circuit is acceptable; otherwise, it is rejected.

1.4.2.3 Retesting Defective Circuits to Allow Acceptable Faults for Yield Enhancement

We extend the idea to identify AcICs from the rejected ones using fault classification in this contribution. The technique uses Hamming distance as a measure to quantify the fault. More generally, it calculates the fault pay-off of each fault. Fault pay-off is a numerical quantity based on Hamming distance. It is found that a fault having the largest fault pay-off affects more than 50% of the output line of the CUT. The technique assumes the fault is catastrophic (unacceptable) if it alters more than 50% of output lines. The retesting phase checks for the existence of all such faults (unacceptable faults). If no such faults exist, the circuit is considered as an acceptable circuit; otherwise, it is rejected. The effectiveness of this technique is measured through experiments conducted using several benchmark circuits from ISCAS85, ISCAS89, and ITC'99. The efficacy is also studied on System on Chips (SOCs), and it is found that 30-40% of faults are classified as acceptable. The proposed approach has the considerable benefit of increasing yield. According to the proposed yield model, the effective yield gain will be between 10-20% on average.

1.5 Organization of the Thesis

This thesis is divided into five chapters based on the contributions mentioned above. The following is a list of the remaining chapters in this thesis.

Chapter 2: Research Background This chapter starts with a discussion of approximate adder design philosophy. Further, it presents a brief survey of different approximate adder circuits. This chapter also explains the principle behind approximating the conventional test flow architecture. Then it discusses previously proposed techniques on approximating test techniques to test traditional ICs and AxICs.

Chapter 3: Approximate Adder Design This chapter explains our proposed SGLP technique to design an approximate adder, starting with the proposal to design an AFA and then to construct an n-bit adder circuit using the AFA in Most Significant Bit (MSB) bits. Finally, the chapter ends with an experimental discussion on image processing.

Chapter 4: Approximate Testing: Approximating Conventional Test Flow Architecture This chapter discusses approximate test techniques, which are divided into two sections. In the first section, we present the notion of approximate testing through fault analysis to reduce the number of test patterns required to test a conventional circuit. The second section presents the process of identifying acceptable circuits from rejected ones using approximate testing. Further, section two also introduces a yield model for approximate testing.

Chapter 5: Conclusion and Future Work The study covered in this thesis comes to a close in this chapter. We highlight some of the most critical design issues and future aspects of approximate adders. In addition to the thesis's contributions and benefits, we also describe the future element of approximate testing of AxICs.

Chapter 2

Research Background

This chapter presents a brief overview of approximate circuit design methodologies, particularly approximate adder designs. It also provides the introductory description of conventional test flow architecture and the testing process in the context of approximate computing. It first discuss the design principles behind approximate adder circuits and then presents a survey of design approaches used in designing approximate adder circuits. Further, it describes the fundamental principle of digital testing and explains the process of approximating test techniques available to test approximate circuits.

2.1 Approximate Adder Design Philosophy

Before reviewing the several approximate adder design techniques, it is necessary to understand its principle. The basic adders like RCA and Carry Lookahead Adder (CLA) are mainly studied for approximation. Other adder circuits such as KSA, the LFA, the BKA, and the HCA have also been examined to design approximate versions. The methods developed so far in designing approximate adders depend on a particular type of adder and are not generic. In other words, the method/technique developed for RCA may not be applicable for KSA. In [48], we find the architecture and other characteristics of each adder circuits described above.

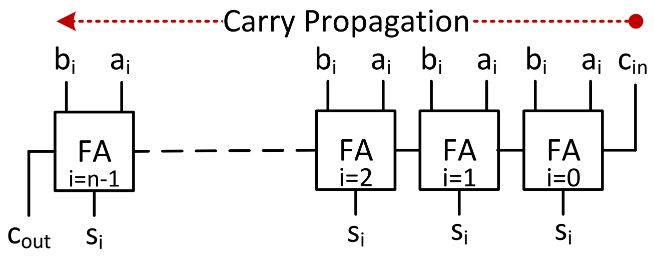


Figure 2.1: *Ripple Carry Adder*

The basic building block of the RCA circuit is the FA. An n-bit RCA is a collection of n FAs connected in series. Each FA takes three inputs: a_i, b_i, and c_{in}, where c_{in} is the carry-in that comes from its previous FA, and two outputs: s_i and c_{out}. Here, a_i and b_i is the ith input, and s_i is the sum of ith FA. The formulation for s_i and c_{out} is $s_i = a_i \oplus b_i \oplus c_{in}$ and $c_{out} = a_i b_i + (a_i \oplus b_i)c_{in} = a_i b_i + a_i c_{in} + b_i c_{in}$, respectively. The carry-in of the 1st FA (rightmost) is 0, and for RCA, the carry is propagated from the rightmost FA towards the FA present at extreme left. An FA cannot start its operation unless it receives the carry-in from its previous one. Therefore, in an n-bit RCA, the worst-case *delay* will be $O(n)$. In other words, the critical path of RCA depends on the length of the carry propagation. However, in most cases, an n-bit adder's carry chain is not n-bit; instead, k-bit is sufficient to predict the carry, where $k < n$ [9, 49]. This creates the opportunity for a designer to think about approximating the RCA circuits and reducing the delay caused due to the carry propagation. The first design philosophy is evolved from the above description and is quoted below.

Design Philosophy-1: The carry propagation chain along the critical path is primarily responsible for the delay in the RCA circuit. If carry propagation can be abolished or reduced, speed, performance, and power consumption can be significantly improved.

The second design philosophy has come from the fact that error in the LSB bit does not affect the overall output of an adder. For instance, consider an 8-bit

RCA shown in Figure 2.2.

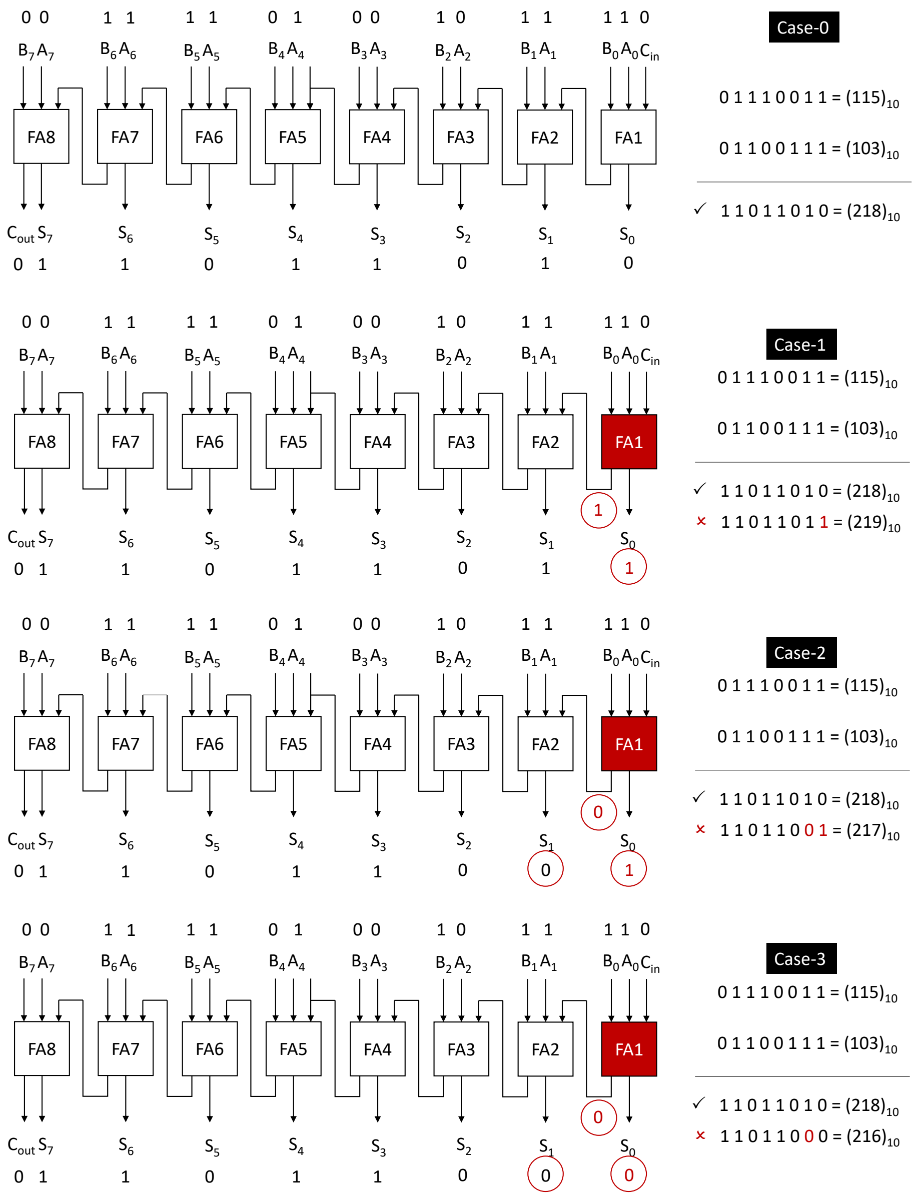


Figure 2.2: *8-bit RCA Circuit Showing Error at LSB bit [61]*

Case-0 shows the correct adder circuit and produces the correct output. The remaining cases (case-1 to 3) show the erroneous output produced due to an error that occurred in FA1. In other words, three cases are possible if an error occurs at FA1; the affected FA (FA1) might produce (1) an incorrect sum but correct carry, (2) incorrect sum and incorrect carry, and (3) correct sum but incorrect carry. Figure 2.2 also shows a numerical example that shows the deviations in the outputs. Carefully analyzing different results, it is found that the errors can be ignorable if it is due to a fault present in the LSB bit.

Design Philosophy-2: Erroneous output produced due to the error present in the LSB bit does not affect the overall output much. In other words, making modifications on FAs present in the LSB to gain in area, power or delay does not affect the overall result of the adder circuit.

2.2 Approximate Adder Design Techniques

Section 2.1 introduces two design philosophies for approximate adders. The first philosophy explains the effect of reducing the critical path, and the second one shows the impact of the LSB bit on the overall result. Based on the above two philosophies, the approximate adders designed so far are classified into three categories: (1) Approximate Block Adder (AxBA) (2) Approximate Segment Adder (AxSA) (3) Approximate Pruning Adder (AxPA). This section describes the basic principle behind the categorization, and later, it provides an extensive survey of previously published works on approximate adders under each category.

2.2.1 Approximate Block Adders (AxBA)

In this category, the adder is split into separate blocks and is based on the first design philosophy described in Section 2.1. The approximation is achieved by shortening the carry propagation chain. The result of a block is determined on its carry-in bit, which is estimated based on previous LSB inputs. No modification is done on any FAs, which are the fundamental blocks of any adder circuit. Techniques proposed in approximate adder circuits like Variable Latency Speculative Addition (VLSA) [49], Error Tolerant Adder Type II (ETAII) [50], Speculative Carry Select Addition (SCSA) [11], Approximate Carry Skip Adder (ACSA) [51], Carry Speculative Adder (CSPA) [52], and Efficient Carry Speculative Approximate Adder (EFCSA) [62] are coming under this category.

2.2.1.1 Variable Latency Speculative Addition (VLSA)

In [49], the author constructs an incredibly fast unreliable adder that provides correct results for most input combinations and is referred to as Almost Correct Adder (ACA). The sum (s_i) and carry (s_i) at bit position i are calculated using the formula given below:

$$c_i = \begin{cases} 0, & \text{if } k_i = 1, \\ 1, & \text{if } g_i = 1, \\ c_{i-1}, & \text{otherwise } (p_i = 1) \end{cases}$$
$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

where, the generate, propagate, and kill signals will be defined as $g_i = a_i b_i$, $p_i = a_i \oplus b_i$, and $k_i = \overline{a_i + b_i}$, respectively. The longest propagate sequence is approximately $logn$. The VLSA is constructed using ACA as a component and yields the result of the ACA as the output. Along with that, VLSA also produces a signal denoting whether the result is correct or not. If an error is found, then after two cycles, it yields the correct output. The major drawback of this technique is the requirement of an error-correction circuit and some extra clock cycles needed for the correct result. Furthermore, the area overhead (due to error detection and correction circuitry) can not be negligible in practice.

2.2.1.2 Error Tolerant Adder Type II (ETAII)

In this technique [50], the author has split the entire carry propagation chain into a number of smaller chains. The carry propagation occurs in these smaller chains concurrently. Figure 2.3 shows the architecture of ETAII. A given N-bit adder is divided into M blocks, and each block consists of N/M bits. Each block contains two different circuitries: *carry generator* and *sum generator*. Different from conventional adder circuits, in ETAII, the carry generator doesn't take carry signals from the previous blocks. However, the sum generator receives the carry-in signal from the previous block.

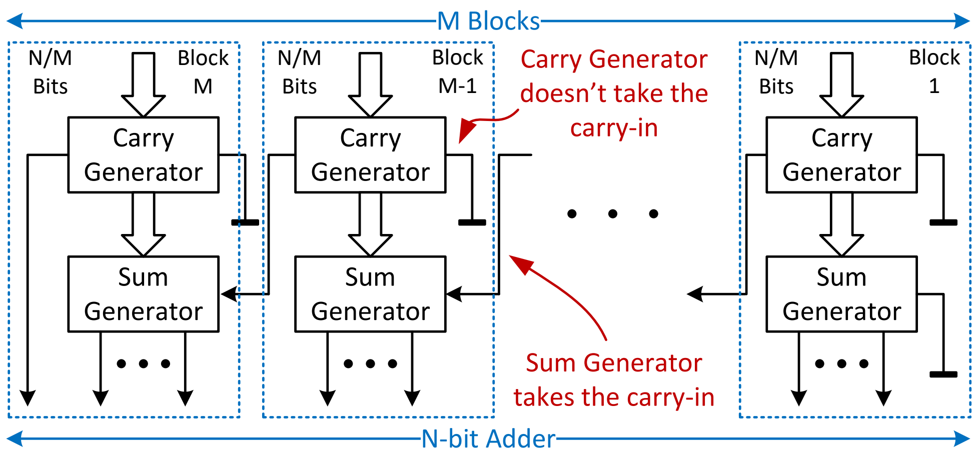


Figure 2.3: *Block diagram of ETAII [50]*

Due to this block-level architecture, the speed and performance of ETAII are remarkably enhanced. However, the power consumption of this architecture remains the same as conventional adder circuits.

2.2.1.3 Speculative Carry Select Addition (SCSA)

The SCSA of [11] is very similar to ETAII and is based on the observation that *the critical path is rarely activated in traditional adders.* Like ETAII, the key idea is to divide the carry propagation chain into blocks of the same size known as *window*. The general structure is shown in Figure 2.4. For a given N-bit adder with window size k, the number of windows (m) will be calculated as $m = \left\lceil \frac{N}{k} \right\rceil$. A window's carry-in is speculated form all k-bits of its previous window. Thus, it reduces the delay of the carry propagation.

For a given N-bit adder, the sum and carry will be calculated as using the following:

$s_i = p_i \oplus c_{i-1}$ and $c_i = g_i + p_i c_{i-1}$

where i represent the bit position, g_i represents the generate signal and p_i

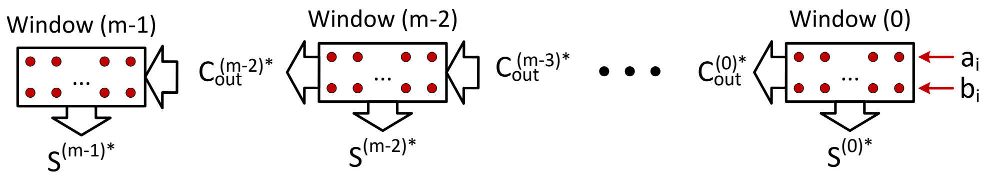


Figure 2.4: *Block diagram of SCSA [11]*

represents the propagate signal. The g_i and p_i signal of ith bit is defined as follows:

$p_i = a_i \oplus b_i$ and $g_i = a_i b_i$

The most extended carry chain length will be the number of successive propagate signals p_i with value 1. Because one input is only coupled to one block adder, the primary input fan out and the area are minimised when compared to VLSA.

2.2.1.4 Approximate Carry Skip Adder (ACSA)

This paper follows the same approach of dividing the given adder into smaller blocks and introducing a carry skip logic between blocks. The carry skip scheme helps predict an accurate carry in a parallel manner. The technique also includes an error magnitude reduction strategy that takes no extra clock cycle to calculate the error-free result. Figure 2.5 (a) shows the block diagram of ACSA. Each block consists of k-bit adder and k-bit carry generator. The total number of blocks (m) will be calculated as $m = \left\lceil \frac{N}{k} \right\rceil$, where N represents the given N-bit adder. Each k-bit adder is implemented through traditional adder circuits like RCA or CLA. All block's carry generators with their k inputs execute simultaneously and generate the carry-out. Similarly, all block's adders take the speculated carry-in (generated using two preceding k-bit carry generators with a multiplexer) and produce the partial sum.

The carry prediction scheme of the proposed technique is shown in Figure 2.5

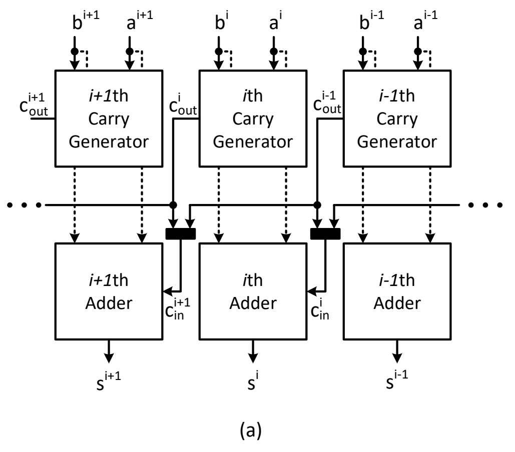


Figure 2.5: *(a) Block diagram of ACSA (b) Carry Prediction [51]*

(b). It depends on the propagate signal p. The carry-out of $i-1$th block is used only when the ith block's propagate signals are true. The propagate, generate, and kill is calculated as $g_i = a_i b_i$, $k_i = \bar{a_i}\bar{b_i}$, and $p_i = a_i \oplus b_i$, respectively. The carry signal is calculated using the following.

$$c_i = \begin{cases} 1, & \text{if } g_i = 1, \\ 0, & \text{if } k_i = 1, \\ c_{i-1}, & \text{if } p_i = 1 \end{cases}$$

2.2.1.5 Carry Speculative Adder (CSPA)

Like other speculative adders discussed so far, the CSPA is also a unique high-performance, low-power approximation adder that splits an N-bit adder into multiple smaller block adders that can run in parallel. Every block adder contains a carry predictor unit to predict the carry for each block. It uses the closest MSBs as input to predict the carry. Therefore, the critical path time is minimized because the carry predictor uses fewer bits. The overall architecture of a single block is shown in Figure 2.6. The technique divides the given adder into multiple blocks of size x-bits each. Therefore there will be $m = \left\lceil \frac{N}{x} \right\rceil$ number of blocks possible. Each block consists of three significant parts: a multiplexer, a sum generator, and a carry generator. Every block uses a carry predictor to predict the carry. The sum generator of every block generates the partial sum.

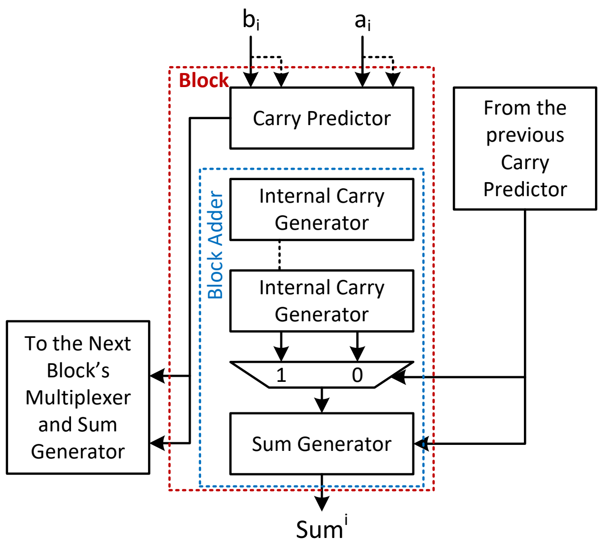


Figure 2.6: *A Single Block of CSPA [52]*

There are two basic differences between CSPA and the previously described techniques. The first difference is the use of two carry generators to generate the internal carry signal, which is later used by the sum generator to generate the sum. The second one is the use of a modified full adder as the internal adders. Compared to the traditional ones, the modified full adder uses fewer cycles.

2.2.1.6 Efficient Carry Speculative Approximate Adder (EFCSA)

This paper [62] proposes a novel block-based reconfigurable carry speculative approximate adder (EFCSA). It reduces the carry chain by dividing the adder into fragments known as summation blocks and inputs into sub-inputs. An N-bit approximate adder consists of [N/k] k-bit summation blocks. A summation block contains two components, (1) summation logic (SL) and (2) carry prediction logic (CPL). The sub-inputs are fed into the SL that generates the sum in a non-blocking parallel manner. The carry input of a summation block is predicted from its previous CPL. The unique feature of the EFCSA adder is that it limits the carry chain and uses a layered approximation technique for each summation block, enabling faster calculations.

2.2.2 Approximate Segment Adders (AxSA)

These adders follow the second design philosophy described in Section 2.1. The approximation is achieved through employing inexact full adders in the LSB part of the adder circuit. In other words, the basic building block of these categories of adders is the inexact Full Adders, commonly known as Approximate Full Adders (AFAs) in most of the literature. The basic design principle of AxSA is to partition the given n-bit adder into two segments: an inaccurate (inexact) segment and an accurate segment, as shown in Figure 2.7. The former consists of AFAs accepting the inputs from LSBs. The latter consists of conventional Full Adders (FAs) receiving inputs from MSBs. Compared to AxBA, AxSAs are more power-efficient and consumes lesser area [63]. Techniques proposed like Gate-Level Simplification [53], Lower-part-OR Adder (LOA) [54], and Approximate Mirror Adder (AMA) [55] are coming under this category. Adder proposed in [64], error reduced carry prediction approximate adder (ERCPAA) [65] slightly modifies the above idea to enhance the performance of approximation.

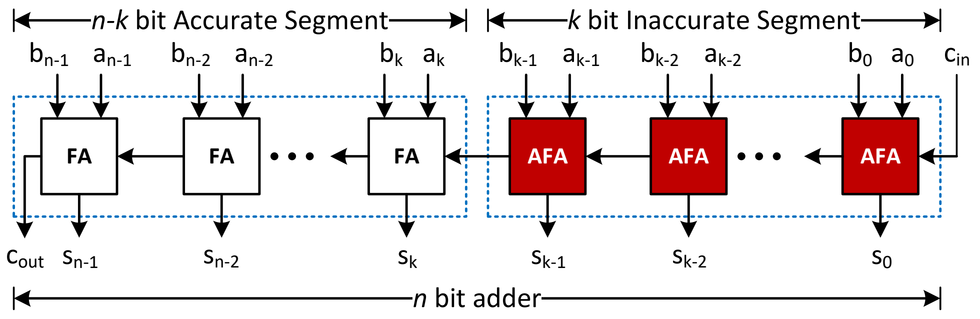
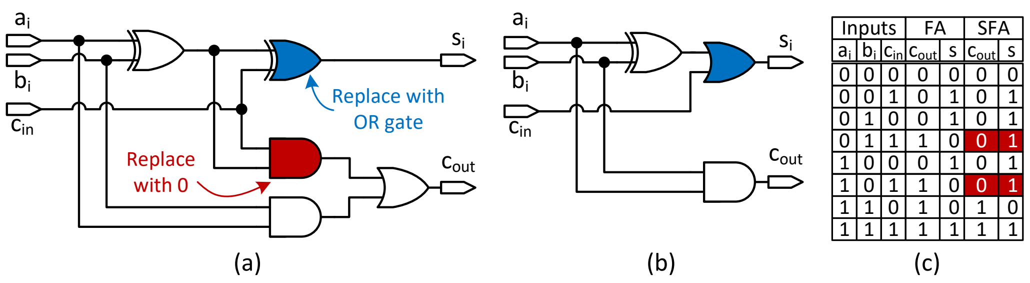


Figure 2.7: *Basic Design Principle of AxSA*

2.2.2.1 Gate-Level Simplification

A gate-level simplification of conventional FA circuit was proposed in [53] that uses a re-design technique to reduce the complexity. The basic idea is to cut the critical path by simplifying on-path gates. For instance, if a critical path contains an OR gate, it can be replaced with 1, and an AND gate with 0. Figure 2.8 (a) shows a FA circuit with an indication of what needs to be done to cut the critical path. A gate-level simplification of the given FA is done by substituting the outcome of the AND gate with 0. The technique also removes the XOR gate with an OR gate. Figure 2.8 (b) shows this modified design called Simplified Full Adder (SFA).



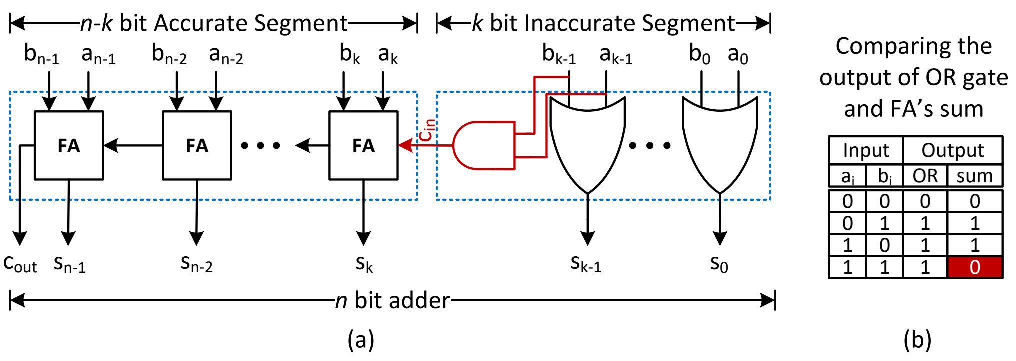
Inputs			FA		SFA	
a_i	b_i	c_{in}	c_{out}	s	c_{out}	s
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	1
1	0	1	1	0	0	1
1	1	0	1	0	1	0
1	1	1	1	1	1	1

Figure 2.8: *Gate Level simplification of FA [53]*

Modifying the original FA using this technique leads to some amount of error in the output. Figure 2.8 (c) shows the truth table that compares the result of SFA with FA. The output of SFA differs in two places from the original one.

2.2.2.2 Lower-part-OR Adder (LOA)

The Lower-part-OR Adder (LOA) divides a given n-bit adder into two segments, not necessarily equal [54]. The first segment is known as k-bit inaccurate segment. All conventional FAs will be replaced with OR gates in this segment, as shown in Figure 2.9 (a). The second segment that constitutes the remaining bits is called the $(n - k)$ bit accurate segment. The accurate segment computes the precise result by taking inputs from the MSBs. The carry-in requirement for the accurate


n-k bit Accurate Segment
k bit Inaccurate Segment
b_{n-1} a_{n-1} b_{n-2} a_{n-2} b_k a_k b_{k-1} a_{k-1} b_0 a_0
FA
C_{in}
C_{out} S_{n-1} S_{n-2} S_k S_{k-1} S_0
n bit adder
(a)
Comparing the output of OR gate and FA's sum
(b)

Input		Output	
a_i	b_i	OR	sum
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

Figure 2.9: *Lower Part OR Adder Architecture [54]*

segment is fulfilled by using an extra AND gate connected to $(k-1)$th input bit of the inaccurate part.

The objective of replacing the FAs with OR gates is simple: compare the sum and OR gates output shown in 2.9 (b). Also, the overhead of carry-propagation is removed in the inaccurate segment and the number of gates reduces to 1 only. The error probability will increase with an increase in the lower-part length.

2.2.2.3 Approximate Mirror Adder (AMA)

A Mirror Adder (MA) is an economical implementation of a traditional FA at the transistor level [66]. It consists of 24 transistors. Carefully removing transistors leads to producing an Approximate Mirror Adder (AMA). Arbitrary removal of transistors may cause an adverse effect, and the circuit may be unusable. Therefore, we should ensure two things: (1) No input combinations result in short circuits or open circuits, and (2) The simplified MA (after removal of transistors) must not produce too many errors. In [55], the author proposes four simplified versions of MA with lesser transistor count; attain a lower circuit complexity and thus power.

The first approximation (AMA1) design reduces the number of transistors to 16, producing one error in cout and two errors in sum. Similarly, for the remaining approximate designs (AMA2, AMA3, AMA4), the number transistor

Table 2.1: *Truth Table Comparisons of AMAs*

Inputs			MA		AMA1		AMA2		AMA3		AMA4	
a_i	b_i	c_{in}	sum	c_{out}	sum	c_{out}	sum	c_{out}	sum	c_{out}	sum	c_{out}
0	0	0	0	0	0	0	**1**	0	**1**	0	0	0
0	0	1	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	**0**	**1**	1	0	**0**	**1**	**0**	0
0	1	1	0	1	0	1	0	1	0	1	**1**	**0**
1	0	0	1	0	**0**	0	1	0	1	0	**0**	**1**
1	0	1	0	1	0	1	0	1	0	1	0	1
1	1	0	0	1	0	1	0	1	0	1	0	1
1	1	1	1	1	1	1	**0**	1	**0**	1	1	1
Transistor Count			24		16		14		11		11	

count is reduced to 14, 11, and 11, respectively. Table 2.1 shows the errors of all approximation designs with highlighted blocks.

In [67], the author proposes another simple and efficient design based on traditional Mirror Adder (MA) named Light-weight Configurable Approximate Adder (LCAA) with two extra transistors. A distinguishing feature of LCAA is to dynamically switch between the accurate and approximate mode.

2.2.3 Approximate Pruning Adders (AxPA)

The approximate adder design techniques discussed so far are primarily applied to RCA circuits, where a carry-chain or a fundamental block such as FA is present in the design. Either the method reduces the carry-chain or re-designs the FA block to achieve approximation. But, what about the adder circuits like KSA, the LFA, the Brent–Kung adder BKA, where no fundamental blocks or carry chain is available. These adders require unique treatments to generate an approximation version of it. In [56, 57], the author proposes one such technique based on probabilistic pruning [68]. *Probabilistic pruning* is a circuit design approach in which it is possible to systematically prune or eliminate components and their associated wires along the circuit path. The error tolerance of the application determines the amount of pruning.

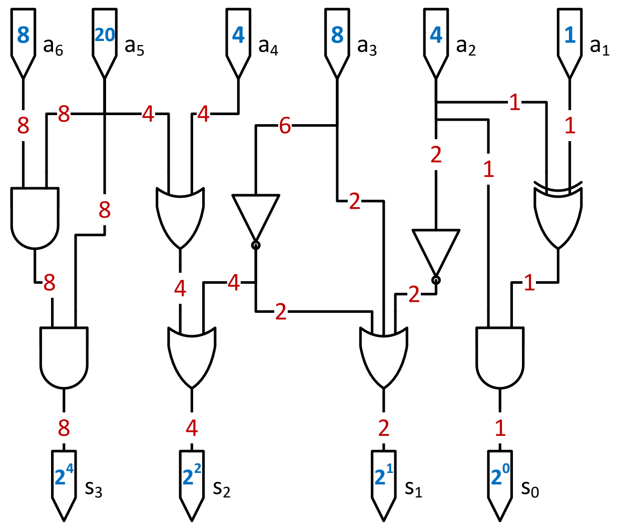


Figure 2.10: *Gate-Level Netlist Example and Significance [57]*

Figure 2.10 shows a simple gate-level netlist. To prune a gate from this circuit depends on two parameters, combined called Significance Activity Product (SAP): significance and activity (toggle count). The gate that has the lowest SAP is pruned first. The toggle count of each wire can be obtained from the *.SAIF* file (Switching Activity Interchange Format). The significance calculation is obtained by assigning weighted significance to each output bit of the circuit. In Figure 2.10, 2^0 is assigned to s_0, and it is two times higher as we move from LSB to MSB. Therefore, the weighted significance of s_1, s_2, s_3 will be 2^1, 2^2, and 2^3, respectively. A reverse topological traversal is performed to compute each nodes significance using the following formula:

$$\sigma_i = \sum \sigma_{desc(i)}$$

where σ_i is the significance of the node i and $\sigma_{desc(i)}$ is the significance of the direct descendants of node i. An implementation example of the technique is described in [56] for adder circuits.

2.3 Approximate Testing Philosophy

This section presents a literature review on test design for approximate circuits and approximating test flow architecture. First, it introduces the flow of digital VLSI testing and explains several associated terminologies. Then it presents the motivation behind approximate testing and gives the idea of approximating the test flow architecture for testing an approximate circuit.

2.3.1 Digital VLSI Testing

A system *failure* occurs when a system fails to produce the desired result. In other words, the system fails to do what it has to do. A failure is caused by an *error* that is nothing but a system's state. We can say that the system is in an erroneous state when it differs from the state in which it should be to produce the expected service. A *fault* causes an error, and a circuit error can result in a system failure [69].

Testing is an indispensable part of a typical digital system manufacturing process. It ensures that all fabricated chips meet the desired specification. A typical test flow architecture is shown in Figure 2.11, where CUT is referred to as the Circuit Under Test [59]. The inputs that are supplied during the test procedure is known as *test vectors* or *test patterns*, and the entire collection of test patterns is called a *test set*. After applying the test patterns, the CUT's response is recorded and compared with the golden response using a test response analyzer. If the response matches, the CUT is considered "pass"; otherwise, it is faulty. The *golden response* is referred to as the expected response of the CUT. From the test application process, it is evident that the circuit's quality depends on the test patterns prepared ahead of time by the *test-generation method.*

Testing a CUT with n inputs and m outputs requires 2^n possible input patterns, and if we do so to test the circuit, it is called *exhaustive testing.* However, the exhaustive testing approach is not practical when n is large. The basic idea of *functional testing* resulted from the concept of exhaustive testing. In other

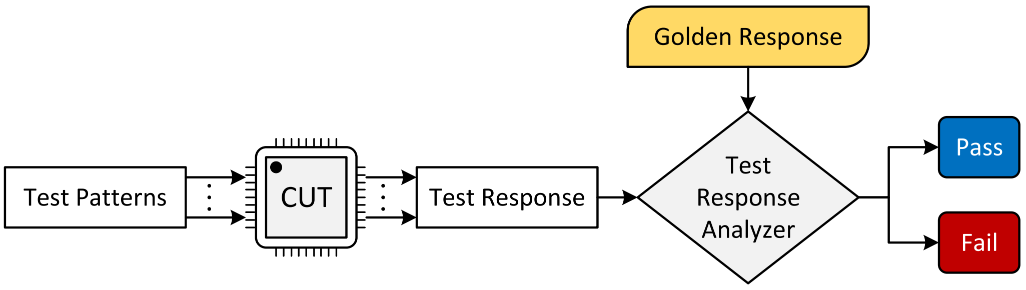


Figure 2.11: *Conventional Test Flow Architecture [59]*

words, with functional testing, a tester applies all possible input patterns (2^n) and tests the system for its correctness.

Instead of using functional testing, a more practical approach is followed, known as *structural testing* [70]. A specific set of test patterns is chosen with this technique based on the circuit's structural information and a *fault model*. Structural testing uses a subset of test patterns that saves time in applying it and improves test efficiency compared to functional testing. However, structural testing does not guarantee all manufacturing defect detection because the test vectors that we use are generated for a specific *fault model*. Therefore, *fault coverage* (fc) is defined as the ratio between the number of detected faults (df) and the total number of possible faults (tf) ($fc = \frac{df}{tf}$).

The purpose of *test generation* is to come up with a collection of test vectors that can identify all of the faults that are relevant to that circuit. Because a set of test vectors can normally discover a large number of faults in a circuit, *fault simulation* is commonly used to assess the fault coverage produced by that set of test vectors. As a result, *fault models* are required for both fault simulation and the development of tests.

2.3.1.1 Fault Models

The technique of modeling defects at higher levels of abstraction in the design hierarchy is known as fault modeling. An exemplary fault model satisfies two criteria: (1) it must represent the defect's behavior accurately and (2) be computationally efficient in fault simulation and test vector generation. Several fault models have been proposed in the literature, and the most well-known and commonly used fault model is the *Single Stack-At fault (s-a) model* [70]. Though there exist some other fault models, such as delay faults, bridging faults, and n-detect faults, our focus only revolves around the s-a fault model.

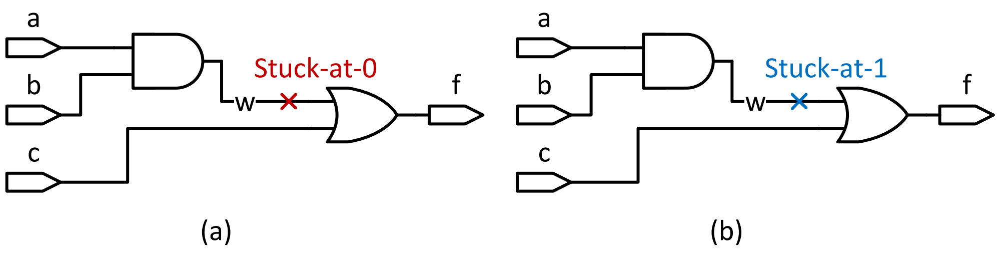


Figure 2.12: *Stuck at Fault model example (a) Line w is stuck to 0 (b) Line w is stuck to 1*

In the s-a model, a line in a logical circuit is either stuck to 0 or 1, referred to as *stuck-at-0 (sa0)* or *stuck-at-1 (sa1)*, respectively. Figure 2.12 shows a circuit with sa0 and sa1 faults for illustration purposes. A circuit with k lines can have $2k$ single stuck-at faults. Some similar faults, on the other hand, maybe removed during implementation. *Equivalent faults* produce identical defective behavior for all input vectors. The number of single stuck-at faults is reduced due to the removal of equivalent faults. The process of detecting equivalent faults and removing them to achieve a reduction in the entire set is known as *fault collapsing*. Fault collapsing helps both test generation and fault simulation times.

2.3.1.2 Test Generation

Test generation is a process to generate a compelling set of test patterns for a given fault model to accomplish a high fault coverage. It is possible to generate an appropriate reduced set of test vectors manually, automatically, or both. The goal of *Automatic Test Pattern Generation*, usually abbreviated as ATPG, is to produce relevant test patterns to test the internal structure of a digital circuit. D-algorithm, PODEM, and FAN are the most commonly used ATPG algorithms found in industries. Most ATPG algorithms follow the *path sensitization method* consists of three steps:

1. *Fault Sensitization (fault activation or fault excitation)*: It is the process to activate the stuck-at fault by forcing the fault value to its opposite value. For instance, in Figure 2.12 (a), line w shows a stuck-at-0 fault, and to sensitize it, we should assign 1 forcibly.

2. *Fault Propagation (path sensitization)*: It is the process of propagating the fault effect along the path to the primary output of the circuit.

3. *Line Justification*: It is the process of backward tracing to the circuit's primary inputs and set it appropriately so that the fault effect correctly propagates to the observable output.

The algorithms described above are suggested for testing combinational circuits and will not effectively work for sequential circuits. Although ATPG algorithms for sequential circuits exist, they are typically resource-intensive and inefficient. The key challenge with sequential ATPG is controlling and observing the circuit's internal state, i.e. the values of all flip-flops. Consequently, *Design for Test* (DFT) comes to the rescue that proposes several design techniques striving to enhance the testability of the target design. One such technique is called *scan-chain design*, which provides a solution to increase the observability and controllability of flip-flops.

Figure 2.13 shows the basic concept of scan design. First, all the flip-flops of the circuit are converted to an equivalent muxed flip-flop that has input data pins and a scan-in pin. A select line will control the switching between the scan state and the normal functional test. Figure 2.13 (a) shows the modified flip-flop. A scan design is shown in Figure 2.13 (b) with their connection to form a scan chain. The scan chain is activated during the test mode, and desired values are

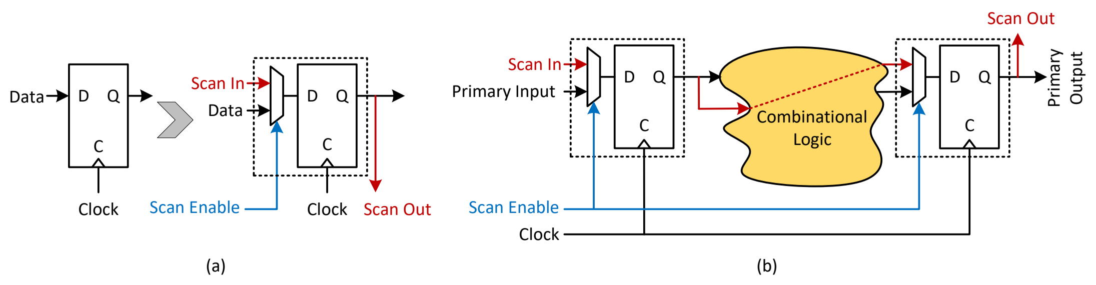


Figure 2.13: *Scan Chain Design Technique (a) Converting Traditional Flip-flop to Scan Flip-flop (b) Scan Design*

pushed to the flip-flips sequentially in each clock cycle. In the next step, the scan mode is disabled, and the circuit operates in functional mode; finally, scan mode is activated again to push the value to the primary output.

2.4 Literature Review of Testing Techniques in the Context of Approximate Computing

This section present an overview of all research articles published in the context of approximate testing. In [60], the author introduces the term Approximate Test (AxT). The author assumes that an approximate system can be built using approximate circuits that produce an inexact result. In other words, an approximate system does not require to be built using defect-free circuits. We can ease the test and reliability limitations of the manufactured ICs under this assumption. Testing solely for a subset of faults rather than all possible faults is one technique to attain this goal. We can cut production costs by reducing the number of test patterns and, as a result, the test duration. The proposal presented in this paper is primarily focuses on relaxing conventional test procedures and thus will never assume a 100% fault coverage, which is against the state-of-the-art. However, the technique is useful where a strict test is not required. A similar process also exists in the literature known as *Threshold Testing* (TT) [71]. It is based on the idea of how much error a design can tolerate and still produce the acceptable output. The error for which the circuit has an acceptable outcome called its *error threshold*. In [72], the author proposes three well-known measures to quantify the error threshold used by all approximate/threshold-based testing techniques: (1) *Error Significance*- the maximum deviation between a circuit's output and the matching error-free output, (2) *Error Rate*- the percentage of test patterns that yield incorrect results, and (3) *Error Accumulation*- error rate variation over time. Testing AxICs also posed several issues and challenges [73, 74]. Testers can not use conventional test techniques to test AxICs because of their inherent error tolerance. Based on the above discussions, test techniques in the context of approximate computing are grouped into three categories: (1) Approximating Conventional Test Flow Architecture, (2) Threshold Testing, and (3) Testing Techniques for Approximate Circuits (AxICs). The following sections will provide a brief overview of several proposed techniques in each category.

2.4.1 Approximating Conventional Test Flow Architecture

In [60], the author proposes a technique to approximate the conventional test flow architecture to test a digital circuit approximately. The overall idea is to generate test patterns for a subset of faults instead of all faults. The approach identifies the fault subset through functional and structural analysis of the given netlist. If prior knowledge of the application is known, functional analysis performs better. For instance, functional analysis works well if the given netlist is an adder circuit and is decided to be used in image processing applications. In structural analysis, each primary output is analyzed based on the fact that the primary output's susceptibility depends on the number of nodes present in its fan-in logic core.

Figure 2.14 shows an example to estimate the susceptibility of each cone. Each gate in the given circuit is assigned a weight (Wt), the number of input and output of that gate. To find the weight of each cone, we can add all the gate's weight which is 15 for 1st fan-in cone and 10 for the 2nd. In the next step, the fault list is generated using the result of structural and functional analysis. The

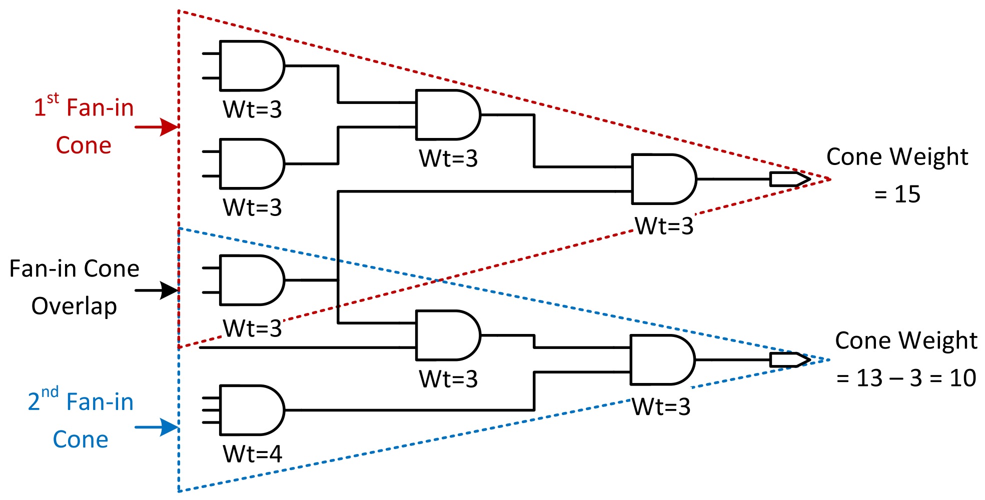


Figure 2.14: *Structural Analysis of Primary Output [60]*

final step generates test patterns for the previously identified fault list. Only a subset of fault is identified and tested by this method, reducing the test data volume, test application time, and energy.

In [75], the author examines the error tolerance nature of the target circuit to relax the functional test requirements. The proposed method in this paper is called approximate functional testing.

2.4.2 Threshold Testing

The idea of Threshold testing is first introduced in [71]. An IC tested using this technique may contain faults but produce output below the identified error threshold. As discussed in Section 2.4, the error threshold is estimated using three quantification measures: error-significance, error-rate, and error-accumulation. We classify previously published literature on threshold testing into three groups based on these quantification measures. The first group uses error-rate (error-rate testing) [76–78], the second uses error-significance [79–81], and the third uses both [53, 82]. No work is found that uses error-accumulation as its measure.

2.4.2.1 Error-rate based Threshold Testing (Error-rate Testing)

In [76], the author proposes a test methodology based on error-rate estimation. First, the method uses the sampling method to find the error-rate of each fault and define a system error-rate. A set of faults is identified based on these two values, which are ignored during the testing. System error-rate is determined using error-rate of each fault and its occurrence rate. Figure 2.15 (a) shows the process flow of this technique which is sometimes referred to as fault-based testing. Identifying the unacceptable fault list requires knowledge of the acceptable error-rate. After estimating the error rate of each fault, it is compared with the acceptable error-rate, and the unacceptable fault list is prepared. Finally, test patterns are generated for them.

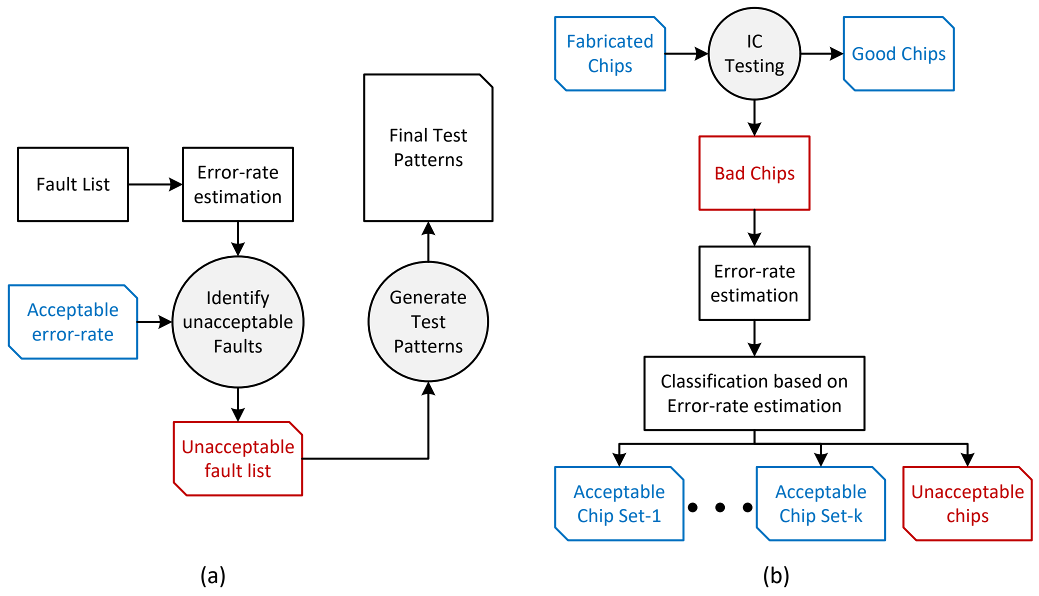


Figure 2.15: *(a) Fault-based test using error-rate [76] (b) Error-rate based Test Methodology [77]*

Another error-oriented test methodology is proposed in [77], which is based on error-rate estimation and supports product grading. The overall outline is shown in Figure 2.15 (b). The bad chips resulting from traditional testing are used in this technique, and their error rate is estimated. Finally, the chips are classified based on their error estimation. These bad chips are now considered acceptable with some percentage of errors in them. The next error-rate testing approach, known as multi-vector testing, was proposed in [78]. Rather than one test vector being generated for each fault as in traditional testing, many test vectors are generated and added to a test set as a single session. If all of the vectors in a session fail to give a proper output, then the CUT is discarded.

2.4.2.2 Error-significance based Threshold Testing

Threshold testing using error significance as error threshold is proposed in [79,80]. Each CUT is tested using a specifically created threshold test set, as illustrated in Figure 2.16. A threshold test pattern is generated for each unacceptable fault and forms the threshold test set. The process first applies the test pattern to the chip and computes the error. If the error is within the threshold, the chip is acceptable and rejected otherwise. A fault in the circuit is considered acceptable if it does not generate an unacceptable error on any output bus for any vector. Otherwise, the fault is referred to as an unacceptable fault.

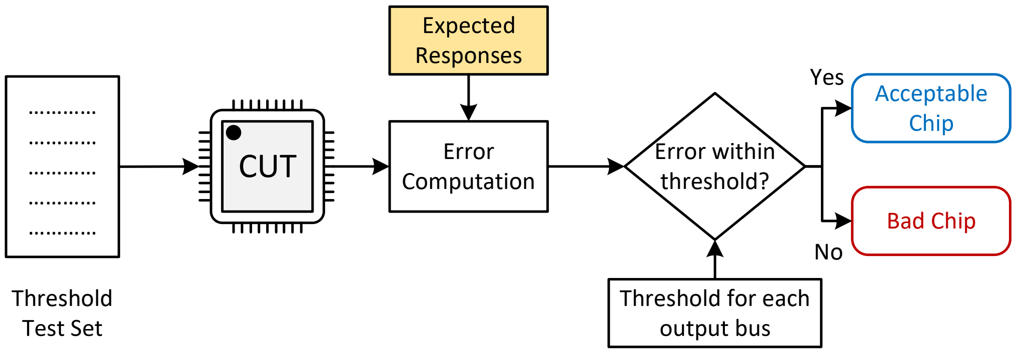


Figure 2.16: *(a) Threshold testing using error significance as error threshold [79]*

Works in [81] proposes another error-significance based threshold testing method that employs standard ATPG rather than specialized ATPG. This work proposes two test generation models: the difference model and the acceptable fault identification model. The difference model is used to create threshold test patterns that can be utilized to identify acceptable and unacceptable faults. However, this model required about twice the amount of hardware as the original circuit, as well as a significant amount of work. As a result, the author suggested an acceptable fault identification model that uses a masking technique to detect all unacceptable faults while only identifying a portion of acceptable defects.

2.4.2.3 Threshold Testing Using Both Error-Rate and Error-Significance

Some chips' acceptance is determined by the fraction of input patterns with error-significance greater than a threshold. In other words, the acceptance is not dependent on a test pattern whose error-significance is greater than a threshold. In these situations, both error-rate, and error-significance is considered as the quantification measure. In [82], a similar idea is presented where Significance based Error-rate (SBER) is used. Consider a circuit C with n input pins, and the given error-significance threshold is T. Let the circuit contain a static fault f, then the SBER of $C(f)$ is defined as the fraction of all 2^n patterns whose error-significance is greater than T. Similarly, in [53] the author uses the product of error-significance and error-rate as the metric for error calculation. As shown in Figure 2.17, the acceptance criterion is based on a predefined threshold T specified earlier by the application: $errorSignificance \times errorRate \leq T$.

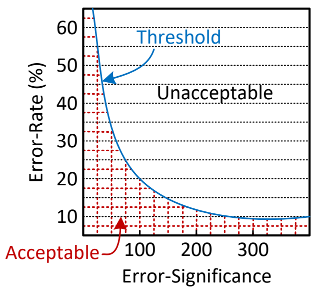


Figure 2.17: *Acceptance Threshold for Error-rate and Error-significance [53]*

2.4.3 Testing Techniques for Approximate Circuits (AxICs)

We know that in an AxIC, the errors are intentional. Therefore, we can not use conventional test techniques to test these ICs. In this section, our focus is on test aspects of AxICs. Works of literature found in this area are basically or can be grouped into two categories. The first category explains the relative challenges in testing AxICs [45, 73, 74], and the second is actually proposing some techniques to test the AxICs [46, 75, 83–86]. Our focus is on the second category.

Works in [75] introduce the notion of testing approximation integrated circuits. The author presents a conceptual framework for performing approximation functional testing based on the application's error-tolerance feature. To elucidate the suggested method's flow, they analyse an image processing application. The entire flow is divided into six steps, starting with examining the image's quality property FSIMc (Feature SIMilarity index with chrominance information). The author's overall goal is to determine the absolute value difference between neighboring pixels (pixel pairs) and keep it if it falls inside a certain range. The recommended technique just explains an idea and does not go into detail about how to put it into practice.

In [83], the author proposes the first workable model for testing Approximate Integrated Circuits (AxIC). The technique is introduced as a preprocess to traditional ATPG, in which fault classification is performed by comparing non-approximate design with its precise matching design with an injected fault under the specified error metric limitations as shown in Figure 3. There are four essential components: a golden reference netlist G, a faulty approximate netlist G_f, an error computation network, and a fault classification network. G refers to a nonfaulty circuit, and G_f refers to the approximate circuit containing a fault. Error computation network compute the concrete error based on the given error metric. Finally, the fault classification network classifies the faults either to an approximation-redundant fault ($E \leq T$) or non-approximation fault ($E > T$). The test generation phase targets only the non-approximation fault and ignores the presence of approximation-redundant faults.

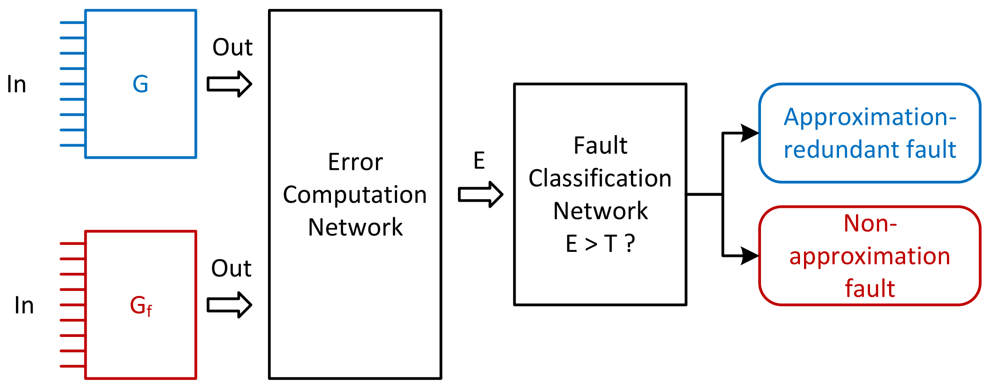

In
G
Out
In
Gf
Out
Error Computation Network
E
Fault Classification Network E > T ?
Approximation-redundant fault
Non-approximation fault

Figure 2.18: *Fault Classification Procedure for Approximate Aware Testing [83]*

An SAT-based ATPG for approximation circuit is proposed in [86]. This approach generates a testing circuit having three sub-components from the given circuit under test : (1) A faulty approximate circuit (AxIC), (2) a fault-free exact circuit (ExIC), and (3) a comparator that compares the output of ExIC and AxIC with a given error metric. Figure 2.19 shows the basic arrangement of these components.

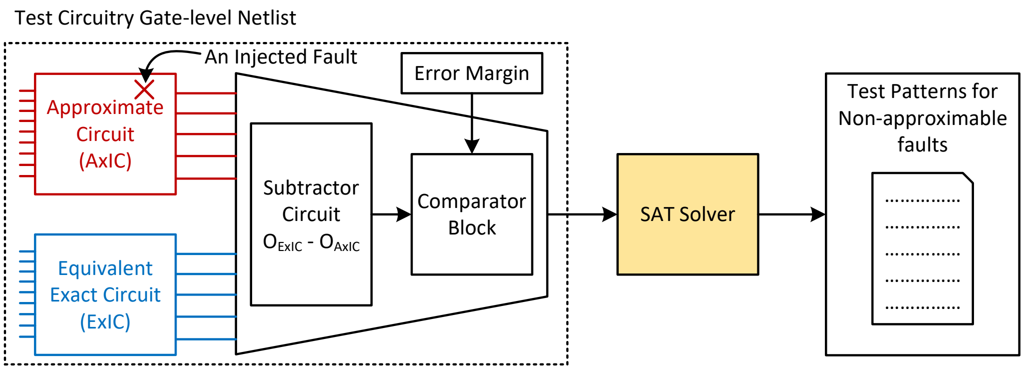

Test Circuitry Gate-level Netlist
An Injected Fault
Error Margin
Approximate Circuit (AxIC)
Equivalent Exact Circuit (ExIC)
Subtractor Circuit O_{ExIC} - O_{AxIC}
Comparator Block
SAT Solver
Test Patterns for Non-approximable faults

Figure 2.19: *SAT-based ATPG for Approximate Circuit [86]*

First, a fault is injected at the fault site of the approximate circuit. The next step generates an SAT instance of the given test circuitry. An SAT solver is employed to solve the SAT instance. The SAT solver evaluates all propagation paths starting from the injected fault location to any primary output. Then a decision is taken to categorize the fault either as approximable or non-approximable. Finally, the tool develops a collection of test patterns that can discover non-approximable defects.

Triola et al. [84] propose three different test pattern generation techniques to test AICs and compare these approaches on public benchmark suites. The first approach, Architecture Under Test (AUT), is quite similar to [83] that requires building a new circuit that embeds both the precise and approximate circuit and receives the same input. The result is analyzed by comparing it with a given application-specific threshold (deviation in actual and approximated output) for acceptability.The second approach, Fault Simulation (FS), applies ATPG directly to the approximated circuit and identifies the unacceptable faults, which require the determination of error due to the presence of the faults in the precise circuit. Finally, in the third approach, Pattern Sorting (PS), the author tries to sort the test patterns (generated in the second approach) by their non-acceptable fault coverage to achieve more pattern reduction. Methods described in this paper consider one error metric: Worst Case Error (WCE), which is defined as the maximum arithmetic difference between approximate and exact circuits.

2.5 Conclusion

This chapter discusses two research areas: (1) approximate adder design and (2) testing techniques in the context of approximate computing. We first discussed approximate adder design techniques beginning with design philosophy. The chapter also presents a survey of approximate block adders such as Variable Latency Speculative Adder (VLSA), Error Tolerant Adders (ETAs), Carry Skip, and Carry speculative adder. Similarly, approximate segment adders like Lower-part-OR Adder (LOA), Approximate Mirror Adders (AMAs), etc., are also discussed. Finally, the discussion on approximate adder design ends after discussing the idea of approximate pruning adder.

A brief description of testing techniques in the context of approximate computing is also presented in this chapter, beginning with an explanation of conventional testing of digital VLSI circuits. Works of literature presenting the idea of approximating test flow architecture are also discussed here. Other test techniques such as threshold test techniques based on error-significance error-rate are also given. The chapter ends with discussing all testing techniques developed for Approximate Circuits (AxICs).

Chapter 3

Approximate Adder Design

This chapter mainly focuses on the design of an approximate adder circuit for image processing application. Section 2.2 introduces different kinds of approximate adder design techniques. We propose an ACD technique known as Significance-based Gate-Level Pruning (SGLP) for designing adder circuits. Suppose the categories explained in section 2.2 are concerned. In that case, the SGLP method combines the feature of AxPA and AxBA, i.e., we can apply this method to obtain an approximate version of chained (adders made up of a chain of full-adders, e.g., RCA), Unchained (e.g., KSA), and full adder block. As mentioned in [56], chained adders are not suitable for the GLP approach, but with SGLP, this is possible. In summary, the SGLP approach can do the following that defines our contribution to this work.

- It is quite easy to get an FA approximation which can later be used in the lower part of a multi-adder to get a multi-bit approximate adder (Block adder category).
- We can apply SGLP directly on the chain of FA (gate-level netlist) to realize its approximate variant.
- Unlike GLP [56], SGLP can also be used to obtain the approximate version of uncut adders (e.g., KSA).

The other contribution of this work includes: (1) We introduces a way of categorizing the ACD techniques for approximate adder design. To the best of our knowledge, this is the first work that categorizes the ACD techniques. (2) We propose a systemic approach that removes gates and reduces the logic complexity at gate-level.(3) We demonstrate the benefits (in terms of power, area, and accuracy) of SGLP over previous approximate procedures and conventional adder design. (4) We built a DCT architecture using the approximate adders generated through SGLP for image compression application, and the result is outrightly acceptable.

3.1 SGLP Technique and Implementation

3.1.1 SGLP for FA Approximation

In this section, we describe the detailed procedure to generate an approximate FA block (AFA) and later part of this section describes how this AFA block is used to form a multi-bit adder circuit. We use RCA as the primary architecture upon which the AFA is implemented to build the required approximate multi-bit adder. SGLP follows a systematic approach and prune gates one by one and on every removal, we get one approximate version of the FA.

Figure 3.1 shows the overall process. There are mainly two processes: (1) Significance Assignment (SA) and (2) Prune and Truth Table Analysis (PTA). The objective of SA is to assign an integer numeral to each gate of the given FA netlist.The purpose of PTA is to prune the gate having the lowest significance and analyze the effect in the truth table for all exhaustive set of inputs. One gate removal originates one approximate FA (AFA) which is again going through the SA and PTA method to produce another AFA. By connecting AFAs, we can generate several multi-bit approximate adders.

Figure 3.2 shows the detail of the entire AFA generation started with the SA shown in Figure 3.2 (a). There are two output lines y and c_{out}, which is assigned with an initial value 2^0 and 2^1, respectively. Hence, the significance of the gates

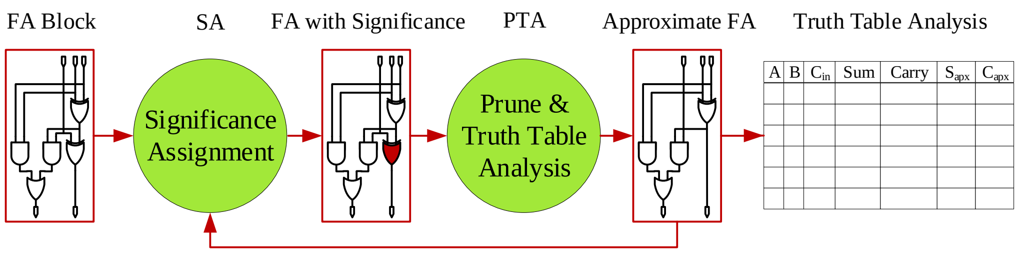


Figure 3.1: *Systematic Process of SGLP for AFA*

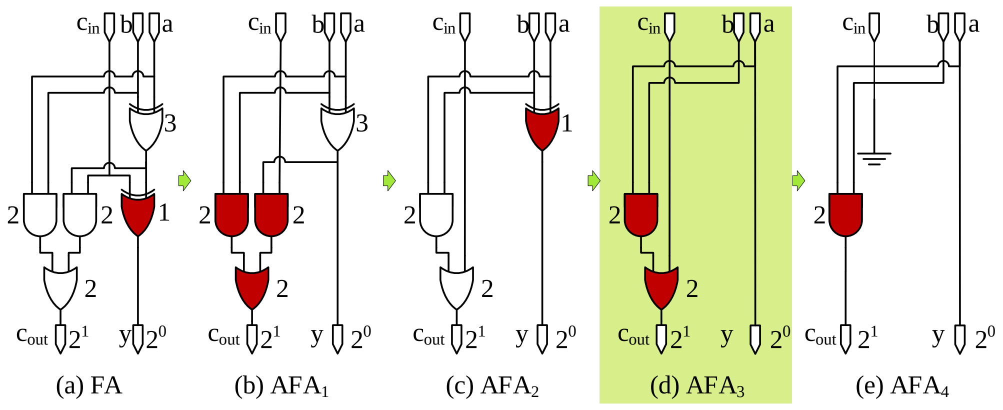


Figure 3.2: *Systematic Generation of AFAs using SGLP*

connected to y and c_{out} will be 1 and 2, respectively. The significance of the remaining gates follows a reverse topological order and is calculated by adding the significance of its descendants. Figure 3.2(a) shows the significance of each gate calculated through the above approach. We can now prune the gates one by one to obtain AFAs starting with the gate having the lowest significance. Figure 3.2 (b) shows the first AFA obtained by removing one gate from the original FA. After the removal, we recalculate the significance of the newly obtained AFA. The similar process continues to obtain the remaining AFAs. Figure 3.2 (c) - (e) shows the entire AFAs obtained through the above approach. Table 3.1 shows the corresponding truth table analysis of each AFAs. The first five columns specify the input combinations (a, b and C_{in}) and the output of a conventional FA (Sum

and C_{out}). The remaining columns show the outputs of all the AFAs (AFA_1 through AFA_4) where a tick mark (✓) is used to indicate the correct output and cross (×) to indicate incorrect output. Now, we can construct multi-bit adders using the AFAs depending on the application where it is used and accuracy of output the application needs.

Table 3.1: *Truth Table Analysis of Approximate FAs*

			FA		AFA_1		AFA_2		AFA_3		AFA_4	
a	b	C_{in}	S	C_o	S	C_o	S	C_o	S	C_o	S	C_o
0	0	0	0	0	0✓	0✓	0✓	0✓	0✓	0✓	0✓	0✓
0	0	1	1	0	0×	0✓	0×	1×	0×	1×	0×	0✓
0	1	0	1	0	1✓	0✓	1✓	0✓	0×	0✓	0×	0✓
0	1	1	0	1	1×	1✓	1×	1✓	0✓	1✓	0✓	0×
1	0	0	1	0	1✓	0✓	1✓	0✓	1✓	0✓	1✓	0✓
1	0	1	0	1	1×	1✓	1×	1✓	1×	1✓	1×	0×
1	1	0	0	1	0✓	1✓	0✓	1✓	1×	1✓	1×	1✓
1	1	1	1	1	0×	1✓	0×	1✓	1✓	1✓	1✓	1✓

3.1.2 SGLP for Uncut Adder

This section describes the detailed procedure and algorithm of Significance based Gate-Level Pruning (SGLP) method for Uncut adder. Figure 3.3 shows the overall process flow of SGLP. The SGLP method comprehensively defined as the process of removing (pruning) the netlist component (in our case, gate) such that, on execution, the netlist may produce an error but within the threshold defined by the designer. It is an iterative process (refer Figure 3.3), begins with (1) significance assignment and pruning the gate with lower significance followed by (2) average Error-Significance (ES_{avg}) calculation and finally, (3) checking whether ES_{avg} is less than or equal to the error-threshold (Δ). The process repeats until $ES_{avg} \leq \Delta$, and on completion, we get a pruned version of the netlist with less number of gates, which produce a result within the error-threshold. Algorithm 1 shows the detailed procedure of SGLP method. GN denote the given Gate-Level netlist with x_i and y_i as the input and output lines, respectively.

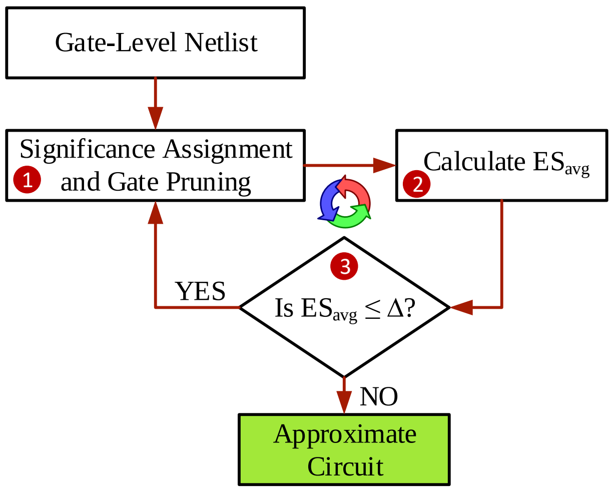


Figure 3.3: *Significance-based Gate-Level Pruning Process*

We represent the circuit under consideration (GN) as a tuple set shown in Equation 3.1:

$$GN = \Big\{ \big\langle g_i, s(g_i) \big\rangle \Big\} \tag{3.1}$$

Where: g_i = represents gate(s) in GN and $s(g_i)$ = represents significance of each gate Our algorithm generates a pruned version of GN (PGN) such that, the following conditions hold:

1. $|PGN(g_i)| \leq |GN(g_i)|$
2. $|PGN(x_i)| \equiv |GN(x_i)|$ and $|PGN(y_i)| \equiv |GN(y_i)|$
3. $ES_{avg}(PGN) \leq \Delta$

i.e., PGN has less number of gates and an equal number of input/output lines compared to GN. Lastly, the result produced is less than or equal to the error-threshold (Δ). We use the following notations in our proposed algorithm:

- T_e: Number of exhaustive test patterns of a circuit which is 2^n, where n is the number of input lines.

- T_k : Set of sample test patterns. T_k is much less than T_e.
- T_k^j: Represent a test pattern in T_k. i.e., $T_k^j \in T_k$.
- Υ: Output produced by the netlist (GN or PGN).
- R and $R^\dagger$: Set of all output produced by GN and PGN, respectively.

Our algorithm starts with the calculation of the golden result produced by the given netlist (GN). For smaller circuits, we took exhaustive test patterns (T_e), and for a larger one, we randomly chose a subset (T_k). After applying them to GN, the output (Υ) is calculated using the Equation 3.2 and stored in a set R (refer line number 1 to 3 in Algorithm 1).

$$\Upsilon = y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + ... + y_0 2^0 \tag{3.2}$$

The next task of our algorithm is to assign significance to each gate, which helps in identifying the first one to be removed. The process starts with the lowest level gates connected to output line and having no successor. It takes the form 2^m (assigns from the Least significant bit) where $m = 0, 1, 2...$, represents the number of lines present in the output. Then a reverse topological traversal is performed to assign significance to the remaining gates present in the circuit. Significance calculation is carried out using Equation 3.3 (refer line number 6 to 12 in Algorithm 1.

$$s(g_i) = \sum s\left(g_i^{descendant}\right) \tag{3.3}$$

Where:

$s\left(g_i^{descendant}\right)$ Significance of the immediate descendant of gate i

After successfully assigning the significance to each gate of the circuit the pruning process is carried out. Line number 13 and 14 of our algorithm is doing this job. The function findSmallest(GN) is used to search the entire set and finds the gate having the lowest significance.This gate is pruned from the GN

ALGORITHM 1: Significance based Gate-Level Pruning

```
Input: Gate-Level Netlist ($GN$), Error Thershold ($\Delta$), Sample Test
       Pattern ($T_k$)
Result: Pruned Gate-Level Netlist ($PGN$)
// Calculate the golden result of GN and store it in a set R
foreach $T_k^j \in T_k$ do
    $\Upsilon = y_{y-1}2^{n-1} + y_{n-2}2^{n-2} + ... + y_0 2^0$;
    $R = R \bigcup \{\Upsilon\}$;
end
repeat
    // Assign Significance to each gate
    foreach $g_i \in GN$ do
        if $g_i$ has no successor then
            $s(g_i) = 2^m, m = 0, 1, 2, ...$;
        else
            $s(g_i) = \sum s\Big(g_i^{descendant}\Big)$;
        end
    end
    // Gate Pruning: Removing the gates from the netlist
       having lowest significance
    // Find smallest tuple w.r.t. $s(g_i)$
    $\tau = findSmallest(GN)$ ;
    $PGN = GN - \tau$ ;
    // $ES_{avg}$ Calculation
    Initialization: $ES_{sum} = 0$;
    foreach $T_k^j \in T_k$ do
        $\Upsilon = y_{y-1}2^{n-1} + y_{n-2}2^{n-2} + ... + y_0 2^0$;
        $R^\dagger = R^\dagger \bigcup \{\Upsilon\}$;
    end
    $ES_{sum} = \sum_{i=1}^{|T_k|} \Big(|R_i - R_i^\dagger|\Big)$ ;
    $ES_{avg} = \frac{ES_{sum}}{|T_k|}$ ;
    $GN = PGN \setminus GN$
until $ES_{avg} \leq \Delta$;
return PGN;
```

in line number 14 to get the PGN. Again, the same set of test-pattern (T_k) is applied on PGN to calculate the result. The output produced is stored in the set $R^\dagger$ (refer line number 16 to 19 in Algorithm 1. We have two sets of results R and $R^\dagger$, obtained from GN and PGN, respectively. We subtract them element-by-element using Equation 3.4 to get our error-significance sum (ES_{sum}) in line number 20.

$$ES_{sum} = \sum_{i=1}^{|T_k|} \Big(|R_i - R_i^\dagger| \Big) \tag{3.4}$$

Line number 21 calculates the average error-significance, and in line number 23 it is compared with the Error-Threshold (Δ). The process repeats only if $ES_{avg} \leq \Delta$, else our algorithm returns the PGN shown in line number 24. In case it reiterates then, the PGN obtained in the last iteration, is treated as the GN for the current iteration (refer line number 22).

3.2 Experimental Evaluation

In this section, we describe the detailed implementation of the 16-bit adder circuit for FA approximation. Due to size limitation, we are not showing the detailed implementation of the approximate uncut adder. But we can follow the same procedure as FA approximation to get our desired result. A simple Ripple Carry Adder (RCA) is considered in our case which is implemented using Verilog HDL in Xilinx Vivado 2018.1 environment (Vivado System Generator for DSP 2018.1). Compatible version Matlab 2017b is used to run our design. A system with Core i5 processor and 8GB RAM is used to execute our experiments.

For FA approximation, we divided the entire circuit into two segments not necessarily equal. We replace the LSBs with the proposed AFAs, and the remaining MSBs uses the regular FA. We use the Design Compiler (DC) EDA tool from Synopsys with 45-nm open cell library to transform the RTL design into Gate-Level netlist. The detail of gain in terms of area, power, and delay is shown in Table 3.2. Here l_b represents the number of FA replaced with AFAs from LSB. For instance, $l_b = 4$ means we divide the entire circuit into two segments one segment contains 4 FAs from LSB, and other contains 12 FAs from MSB. We replace the 4 FAs from LSB with 4 AFAs.

Table 3.2: *Area,Power, and Delay Characteristics*

Adder	Matrix	$l_b = 2$	$l_b = 4$	$l_b = 8$	$l_b = 16$
RCA	Area	68			
	Power	0.54			
	Delay	0.8			
AFA_1	Area	66	64	61	54
	Power	0.52	0.51	0.48	0.43
	Delay	0.78	0.76	0.72	0.64
AFA_2	Area	64	62	55	41
	Power	0.50	0.46	0.42	0.32
	Delay	0.77	0.72	0.63	0.48
AFA_3	Area	59	55	46	27
	Power	0.49	0.45	0.37	0.21
	Delay	0.74	0.68	0.56	0.32
AFA_4	Area	61	54	40	14
	Power	0.47	0.42	0.30	0.10
	Delay	0.72	0.64	0.48	0.16
Area→$[nm^2]$, Power→$[mW]$, Delay→$[ns]$					

After obtaining all these approximate 16-bit Adders (AFA-16), we generate the black box for each of them using Vivado System Generator Tool. These black boxes will be further used to implement the DSP application for image processing. We have generated black boxes for four AFA-16 with $l_b = 8$ which replaces AFA_1 through AFA_4 in each of these AFA-16.

3.2.1 DCT Application

DCT is a computationally ideal component for image processing applications. For our experiment, we took 8×8 pixel blocks DCT. Several DCT architecture has been proposed in the literature [87–89]. The conventional method requires 64 multiplication and 56 addition operations which is a substantial number and hence cannot solve our goal. The scope of our work needs a multiplier-less DCT architecture. This work does not present any new DCT rather we are using an existing multiplier-less state-of-the-art architecture to test our proposed method. One such architecture is presented in [89] which is a multiplication-free transform suitable for image compression commonly referred BAS-2011 in literature. The proposed hardware architecture of BAS-2011 has total 18 addition operations represent a 1D DCT. Using two 1D DCT block along with a transpose buffer, we can realize a complete 2D-DCT transform.

3.2.1.1 FPGA Implementation

Initially, each 1D DCT is modeled by replacing the conventional adders with the proposed adder circuit (implemented as black box using system generator tool) and then linked to form a comprehensive 2D transform. The entire design is realized using Vivado System Generator tool. By this process, we get four different 2D DCT model named as DCT_{v1}, DCT_{v2}, DCT_{opt}, and DCT_{nop}. The realized models are physically built using Xilinx Virtex-6 XC6VLX240T field programmable gate array (FPGA) and connected to the host computer running Matlab Simulink version 2017b. Image processing activity is carried out by estimating the DCT of sample images acquired from each model. The transformed image is then fed into Inverse DCT function to obtain the compressed image. Finally, we calculate the quality measure PSNR (peak signal-to-noise ratio) of the original and compressed image for image degradation using Equation 3.5.

$$PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right) \tag{3.5}$$

Where MAX represents the maximum possible pixel value and MSE is the mean square error: the cumulative squared error between the original image I and obtained compressed image $\hat{I}$ using equation 3.6.

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=0}^{N}[I(i,j) - \hat{I}(i,j)]^2 \tag{3.6}$$

3.2.1.2 Image compression and Result Analysis

To show that our proposed approach does not provide any unreasonable output, we conducted the image compression experiment described in [90] and carried by [91–93]. We considered 45 512×512 grayscale images obtained from a public image library [94]. Each image is divided into 8×8 blocks and submitted to the 2D transformation similar to [92]. For a particular transformation, each block furnished 64 coefficients in the approximate transform domain. Following the standard zigzag sequence [95] reconstruction of the image is done by employing $r(1 \leq r \leq 45)$ initial coefficient to each block and zero to the remaining coefficient. Finally, we obtain the compressed image by applying the actual inverse transformation. We then compare the original image with the compressed one for image degradation using PSNR as the quality measure. Figure 3.4 shows the average PSNR plot obtained by the experiment. Analyzing the result, we conclude that the proposed testing method does not produce any unreasonable output and quite competent for image compression.

Our objective is not to compare our result with other existing DCT algorithm. The main purpose of the experiment is to determine that approximate testing of circuit generates a tolerable result. Hence we also present a visual quality evaluation of our experimental result applied to the standard Lena image for r=25. Figure 3.5 shows the effects of the experiment and supports our claim acquainted in introduction section.

3.3 Conclusion

Approximate Computing technique is a novel design paradigm provides several benefits in terms of area, power consumption, and delay. In this work, we have presented an ACD technique named as SGLP which can be used to reproduce adder circuits for chained and unchained adders. With the previously developed technique, this is not possible that shows the novelty of our work. We have tested our approach using a DCT architecture for image processing particularly image

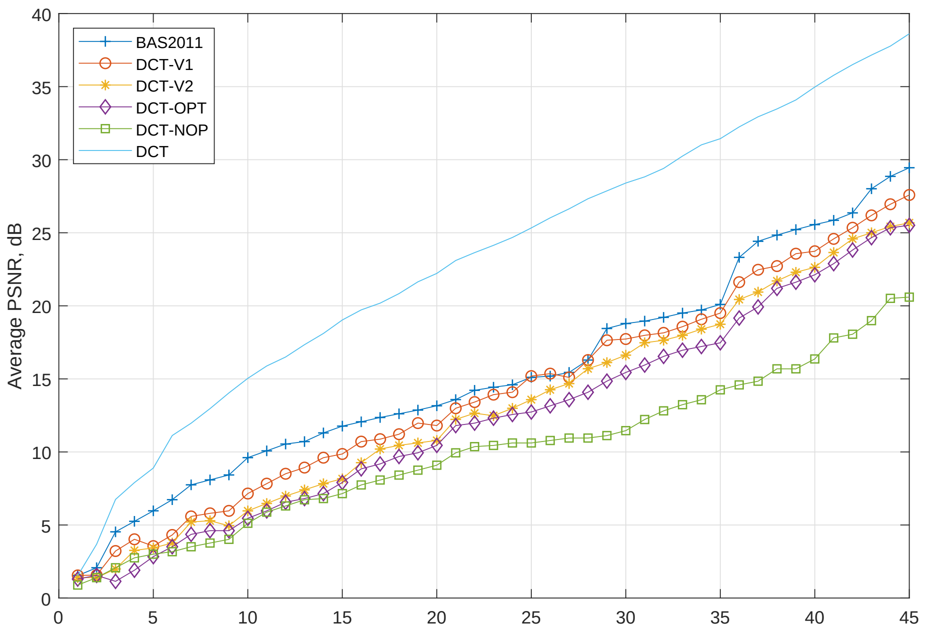


Figure 3.4: *Average PSNR for several compression ratios*

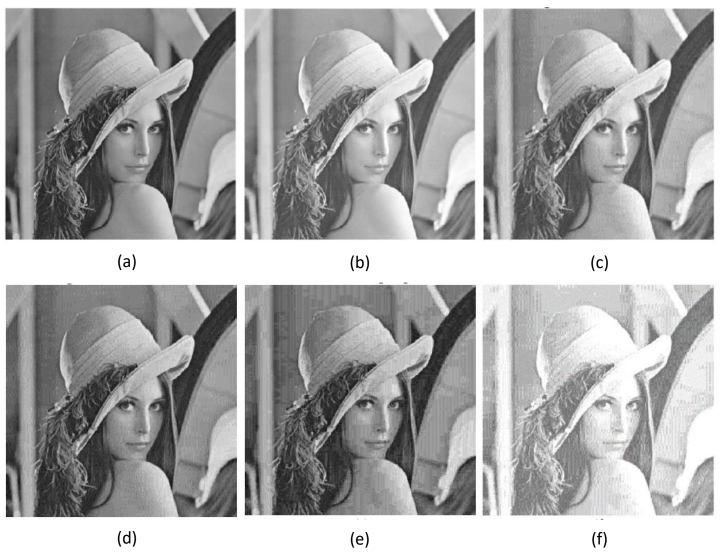


Figure 3.5: *Lena image produced with (a) DCT, (b) BAS-2011, (c-f) Proposed method* DCT_{v1}, DCT_{v2}, DCT_{opt}, DCT_{nop}

compression and found that our result is acceptable to human perception-behavior on image clarity.

Chapter 4

Approximate Testing

Approximate testing is a technique that dilutes the strictness of conventional test procedures. The broad thought belongs to the idea of *threshold testing* described in Section 2.4.2. After deciding on a fault model, instead of generating test patterns for all faults, the method generates test patterns for some critical faults. Due to this, some faults were left untested during the test application, leading to incorrect results while the circuit was actually used. However, the circuit is still acceptable because we are supposed to use these circuits in error-tolerant applications. In this chapter, we explore different techniques for approximating the test procedure.

Overall, the contribution is twofold. The first contribution explains a procedure that generates an approximate version of an original circuit and then

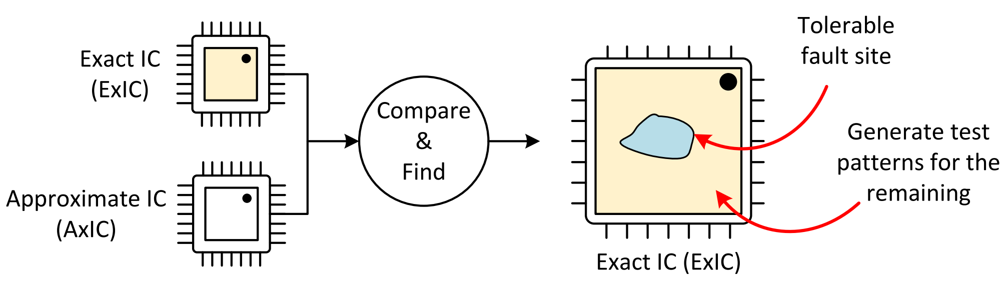


Figure 4.1: *Idea of approximate testing*

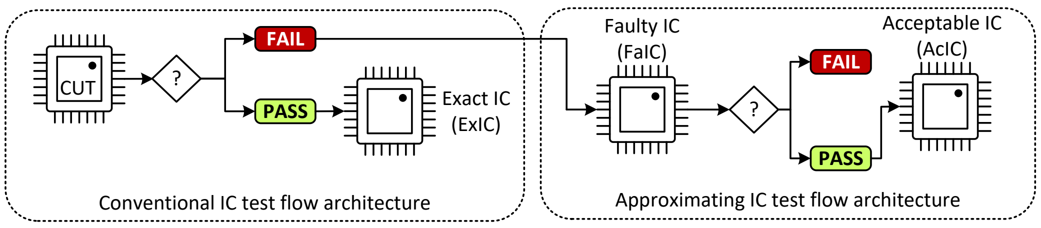

CUT
?
FAIL
PASS
Exact IC
(ExIC)
Conventional IC test flow architecture
Faulty IC
(FaIC)
?
FAIL
PASS
Acceptable IC
(AcIC)
Approximating IC test flow architecture

Figure 4.2: *Idea of approximating conventional IC test flow architecture*

compares them (original and its approximate version) to identify the tolerable fault sites in the original circuit. During the testing, faults in the tolerable fault site are intentionally overlooked, assuming that the fault effect is benign. Figure 4.1 shows the idea of the proposed approximate testing. In the second contribution, we propose an approximate retesting technique to test all faulty circuits resulting from a conventional IC test flow. The objective is to identify all acceptable circuits from the rejected ones. Figure 4.2 shows the idea of approximating the IC test flow architecture to obtain acceptable ICs.

The chapter has two sections. In Section 4.1, we present a basic idea of testing a conventional IC through the technique of approximate testing based on fault analysis. Section 4.2 proposes retesting FaICs (rejected ICs during conventional IC test flow) to identify AcICs, which obviously helps improve the manufacturing yield.

4.1 Fault Analysis based Approximate Testing

The growing design complexity increases the number of fault sites in the circuit. We require an enormous amount of test patterns to cover all these faults. The ever-increasing Test Data Volume (TDV) causes extremely high test cost and power because of extended Test Application Time (TAT) and large Automatic Test Equipment (ATE) memory requirements. In today's scenario, TDV extensively contributes to the cost of an Integrated Circuit (IC) manufacturing. One approach to reduce TDV is to use test data compression scheme [70], which is the most widely used approach. It requires a decompressor unit which is additional hardware that needs to be deployed in the system to decompress test data before applying to the Circuit Under Test (CUT). This work introduces a technique called Fault-based Approximate Testing (FAT) having the following initial deliberations:

1. Application-oriented as compared to classical testing.
2. Based on the concept of Threshold Testing where a specific error threshold (ET) is assigned depending on the application.
3. A classical testing approach is sometimes called stop on first error (SOFE) because the testing continues with the next test vector only when the captured output matches the golden response else the CUT is discarded. Our approach is not based on SOFE; the testing process continues until the output is within the ET.

The basic idea of FAT is to identify the *Approximate Fault sites* (AFS) and overlook them without testing it. In our case, each AFS is either represented with a stuck-at-0 (SA0) or stuck-at-1 (SA1) fault. Overlooking an AFS means not generating test patterns for SA0 and SA1 which leads to a reduction in the number of test patterns.The essential part of our approach is to identify the AFS. To do that, we need to quantify the adverse effect of a fault using some quantification measure (QM) described in [72].The three well-known QM defined in [72] are as follows: (i) *Error-Significance (ES)*: The maximum amount by which the output of a circuit deviates from the corresponding error-free output. (ii) *Error-rate (ER)*: Fraction of test patterns that produce erroneous output. and (iii) *Error-Accumulation (EA)*: Change in error-rate over time. For our purpose, we use ES as the QM. The average ES is calculated for each fault and compared with an ET provided by the designer of the circuit. The fault is considered tolerable if the average ES is lower than or equal to the ET else it is intolerable. Before the test generation, all intolerable fault sites have been identified using the above approach. Later, we have tested the circuit by injecting faults at the intolerable fault site (identified above) for image processing application. Ultimately, we found that even some area of the circuit remains untested does not affect the quality of the image with respect to human perception.

As compared to the traditional testing method, We can define *approximate testing as the testing of digital circuits without 100% fault coverage with an assumption that the uncovered fault, if it exists in the design, does not produce an unacceptable result.*

In summary, in this work the following contributions has been made:

- Identification of Tolerable and Intolerable fault sites in a given gate-level netlist
- Provides a way to preprocess the circuit before actual ATPG is applied.
- May contribute to yield improvement if the rejected circuit (Due to SOFE) is retested using our method and the faults for which it was rejected is present in the tolerable area of the circuit.

4.1.1 Motivation

There are applications like image, video and audio processing that have a common property called *inherent application resilience property* - producing an acceptable output despite incorrect computation made by an underlying hardware. For instance, Figure 4.3 shows two images which look almost the same but their histograms are different. The reconstructed image is obtained by subtracting 20 from each pixel value of the original image. That means the image quality will not be affected much by small changes in the pixel value. So the circuits used in image processing applications can tolerate a small amount of error. Knowing this, the image can also be readily accepted if we do not test some part of the circuit and use the circuit in image processing applications. This discussion brings a motivation to test a circuit approximately.

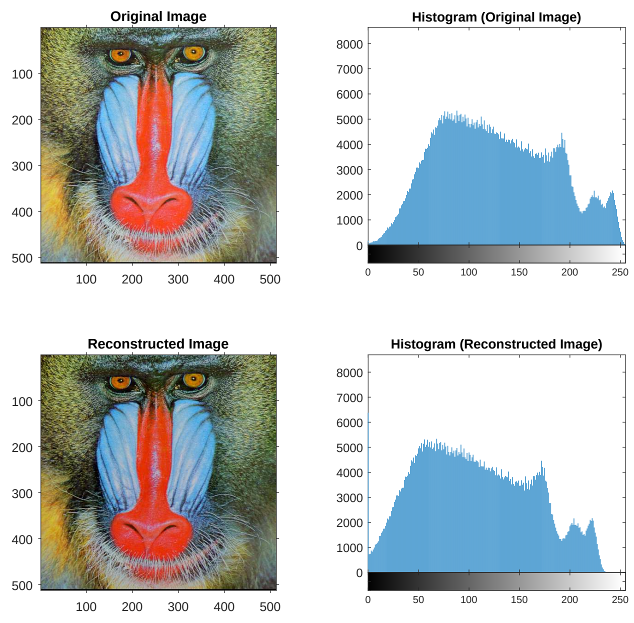


Figure 4.3: *Histogram Comparison*

Test techniques in the context of approximate computing are grouped into three categories, as discussed in section 2.4. Category 1 describes the process of approximating the conventional test flow architecture, category 2 refers to threshold testing, and the third category explains the test techniques to test AxICs. This work is based on category one. The traditional method of circuit testing is based on 100% fault coverage, but category 1 relies on the classification of the fault into tolerable and intolerable faults. The former is left untested, and test patterns are generated to test the latter. Previously, Wali et al. [60, 96] contributed towards this category, where they perform a structural analysis to determine the vulnerable circuit elements and thus generate the test pattern for those circuits. As they are not classifying the faults, they ignore the inherent resilience property of the circuit. In this work, we consider the inherent resilience property of the given circuit and classify the fault into tolerable and intolerable faults.

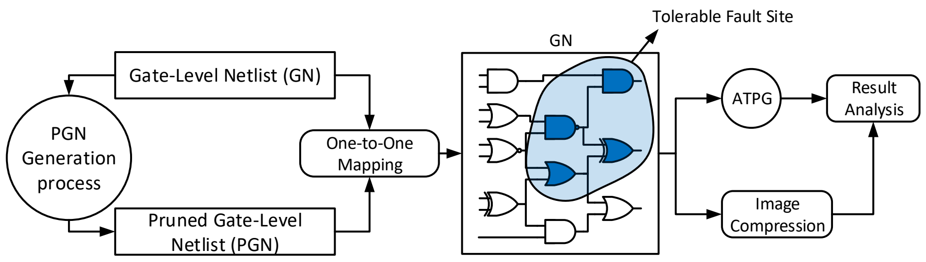


Figure 4.4: *Proposed Method*

4.1.2 Proposed Approach

Our basic approach starts with the identification of tolerable fault site present in the given gate-level netlist (GN). Figure 4.4 describes the overall procedure of our approach. The first step in our process flow is to obtain a pruned version (PGN) with less number of gates as compared to GN using the method described in Section 3.1.2. But it is not possible to generate PGN for all types of circuits using this method. Hence our approach is limited to the image processing applications. The second step is to identify tolerable fault sites in GN which is achieved by one-to-one mapping of fault site present in both GN and PGN. In the third step, test patterns are generated only for intolerable faults that reduces the fault coverage but, it does not affect the quality of the result produced by the circuit. The experimental result shows that 40-50% reduction in test pattern is achieved using our approach. Further, we conducted an image compression experiment by injecting faults at the tolerable fault sites, and the result is absolutely satisfactory with respect to human perception.

4.1.2.1 PGN Generation

Our technique requires a corresponding approximate circuit of the given gate-level netlist. PGN generation process follows gate level pruning technique described in Section 3.1.2. We use Error-Significance (ES) and Average-Error-Significance (ES_{avg}) as the criteria for pruning a node (gate). The process of PGN generation starts with weight assignments and pruning followed by average ES (ES_{avg}) calculation.

Figure 4.5(a) shows a 4-bit ripple carry adder (RCA) circuit for which our approach generates the weight of each gate. Weight distribution takes the form 2^n where $n = 0, 1, 2, ...$ starting from the LSB output line and move towards MSB. There are four full adders that make the 4-bit RCA; each full adder is considered separately for weight assignment. Each carry line also takes part in the initial weight assignment process. The gate connected to primary output is called level-1 or first level and as we go up, the level number is increased. Starting with LSB (y_0) and moving towards MSB ($cout$) the output lines are assigned weight in the form 2^n. As mentioned earlier the carry lines are also assigned weight values denoted in red oval shape in the figure. The first level gates use these values as their weight. The weight of the top-level gate is calculated using Equation 3.3.

After finishing the weight allotment, the next process of our approach is to prune the gates. The gate having lowest weight is removed first followed by average ES (ES_{avg}) calculation. The average error significance (ES_{avg}) is calculated using Equation 4.1.

$$ES_{avg}(PGN) = \frac{\sum_{T_k^j \in T_k} |R_{GN} - R_{PGN}|}{|T_k|} \tag{4.1}$$

where, R_{GN} and R_{PGN} represents the result produced by GN and PGN, respectively.

The entire process is iterated until we get the PGN whose ES_{avg} is less than the ET supplied by the designer of the circuit. In every iteration one node is removed from the circuit, ES_{avg} is calculated, compared with ET and the process is repeated until $ES_{avg} < ET$.

Example: The gate having the lowest weight is pruned first. Consider the example circuit shown in Figure 4.5(a). The gate shown in black is having the lowest significance and recognized as a candidate to be pruned from the circuit. Figure 4.5(b) shows the netlist after removal of 12 gates from the original one. Let

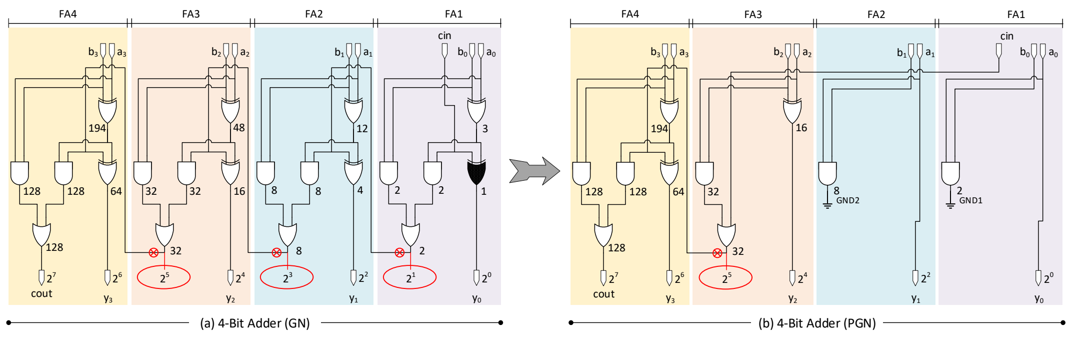


Figure 4.5: *Example*

$(a_3, a_2, a_1, a_0) = (1011)$ and $(b_3, b_2, b_1, b_0) = (1111)$ be the inputs applied to GN and the corresponding output produced is $(cout, y_3, y_2, y_1, y_0) = (11010)$ which is the actual addition result. Result R_{GN} of the circuit is obtained using Equation 3.2. The same input and the equation is utilized to calculate R_{PGN}.

$$R_{GN} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 26$$

$$R_{PGN} = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 23$$

The absolute difference between these two output represents the Error-significance of PGN and is calculated as follows:

$$ES(PGN) = |R_{GN} - R_{PGN}| = |26 - 23| = 3$$

Depending on the time and the size of the circuit the tester needs to decide and calculate ES with an exhaustive test pattern (T_e) or a set of sample test patterns (T_k). For our example, we took (T_e) because our circuit is small and Table 4.1 shows the analyzed result. We define the threshold value as 2. In total, 136 input patterns are applied. It is interesting to note that by removing 12 gates from the original netlist (RCA 4-bit with 20 gates) we get the PGN (highlighted in color yellow). Out of 136 input patterns, the circuit does not produce any error for 28 patterns. For the remaining, the circuit produces approximate output within the threshold.

Table 4.1: *Analysis of the result obtained for RCA 4-bit*

Criteria	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	PGN	R_{10}
# Gate Removed (out of 20)	1	2	3	5	6	7	8	10	11	12	13
# Test Pattern for which no error produced	136	136	64	64	64	28	28	28	28	28	28
Error Percentage (%)	0	0	52.94	52.94	52.94	52.94	79.41	79.41	79.41	79.41	92.64
Average ES	0	0	0.52	0.52	0.52	0.52	1.35	1.64	1.64	1.64	3.17

4.1.2.2 One-to-One mapping

We have two circuits (i) Original circuit (GN) (ii) Pruned gate-level netlist (PGN). Because PGN is the pruned version of GN, it has less number of gates and less number of fault sites as compared to GN. Each fault site represents either a stuck-at-0 or stuck-at-1 fault. The set P_{fs} and G_{fs} denote the fault set corresponding to PGN and GN, respectively and a fault p_i belongs to P_{fs} corresponds to the fault g_j that belongs to G_{fs} and vice versa. This relation is represented using a function shown in figure 4.6 and explained as follows:

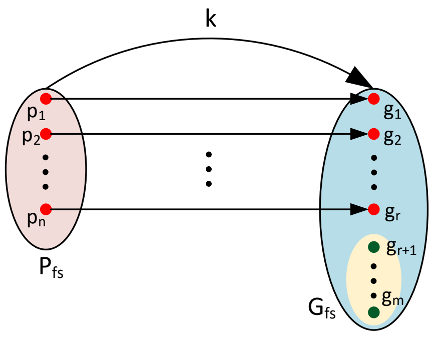


Figure 4.6: *Function representing the relationship between PGN and GN*

Let P_{fs} and G_{fs} be two nonempty sets. A function $k : P_{fs} \longrightarrow G_{fs}$ is defined as a relation between $p_i \in P_{fs}$ and $g_j \in G_{fs}$ when both have same fault location (in their respective circuits), as well as they, refer to the same type of fault (s-a-0 OR s-a-1). Figure 4.6 represents the function k from P_{fs} to G_{fs}. The domain of k is the set $P_{fs} = \{p_1, p_2, \ldots, p_n\}$ and the codomain is the set $G_{fs} = \{g_1, g_2, \ldots, g_m\}$. The range of k is the set $\{g_1, g_2, \ldots, g_r\}$, because $\{g_{r+1} \ldots g_m\}$ is not related to any element in P_{fs}.

A subset of elements $\{g_{r+1} \ldots g_m\}$ in the set G_{fs} is not mapped as it is an onto function. Those elements (faults) though present in the original circuit (GN) are treated as the candidate faults whose presence does not affect the output of GN. During testing, these faults can be left untested and treated as tolerable. The remaining elements $\{g_1, g_2, \ldots, g_r\}$ which has a one-to-one mapping with the element of P_{fs} has to be tested explicitly and are called intolerable faults.

Return to Example Circuit: RCA 4-bit We took the same example to further explain the classification of faults. Figure 4.7 (a) shows the original RCA 4-bit netlist and 4.7(b) shows the optimal pruned netlist with its fault site marked with red dot. Each red dot corresponds to either stuck-at-0 or stuck-at-1 fault. There are 61 fault sites corresponding to 122 faults ($61 \times 2 = 122$) in the original circuit. Similarly, 25 fault sites correspond to 50 faults in the pruned RCA 4-bit circuit. Hence, we get approximately a 41% reduction in the number of faults.

4.1.3 Experimental Result

Our experimental result is two-fold. (i) Test pattern reduction analysis: We tested a wide range of Arithmetic circuits as well as ISCAS-85 benchmark circuits and summarized the results. (ii) Image compression: We use the arithmetic circuits that are tested using our approach to see the impact of image compression.

4.1.3.1 Test Pattern Reduction Analysis

Identification of tolerable fault site is implemented using C++ program where the input to the program is the gate-level netlist (GN). The simulation process is executed on a machine with an Intel I5 processor and 8GB of RAM. Before the ATPG generation process, we divided the circuit and identified the region of fault tolerance. After that, the normal ATPG flow is carried out. We have taken two types of circuits for the experiment (i) Standard arithmetic circuits that are collected from [97], act as the perfect candidate for our approach (ii) ISCAS-85 benchmark circuits [98] that are fit to our proposed method.

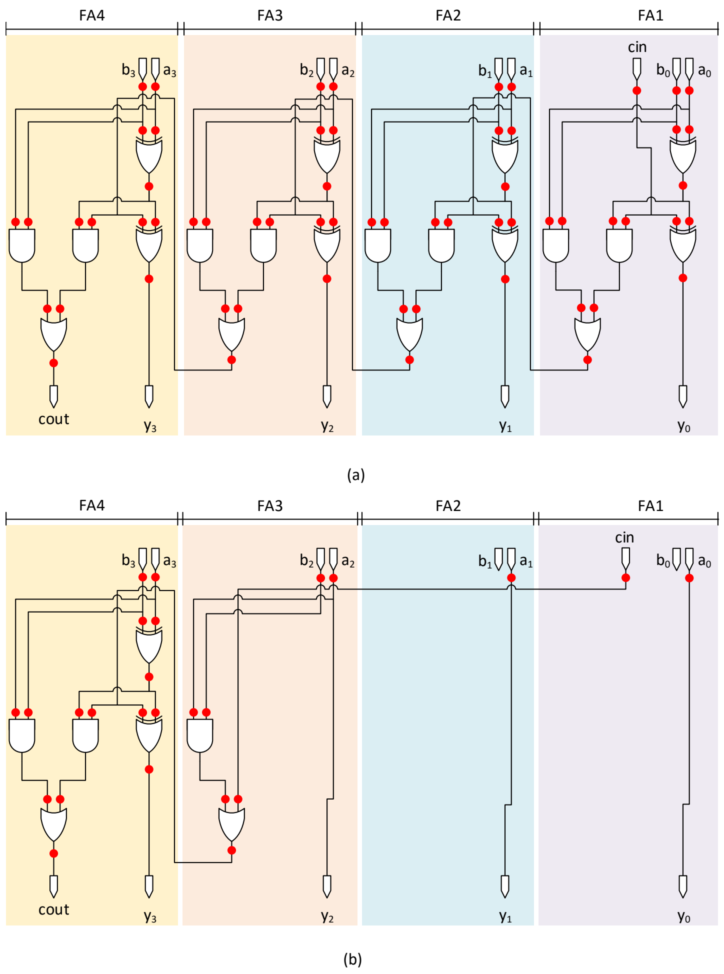


Figure 4.7: *Fault site shown in red dot*

Table 4.2 summarizes the result produced by the experiment. The 2nd and 3rd column of the table indicates the number of primary input/output and the number gates of each circuit, respectively. The 4th, 5th and 6th columns show the total number of faults ($\#F$) that include fault equivalence and fault dominance, tolerable faults ($\#T_f$) identified using our proposed approach and targetted fault percentage (F_{tar}) which is calculated using the Equation 4.2, respectively.

$$F_{tar} = \frac{F - T_f}{F} \tag{4.2}$$

Table 4.2: *Fault-based Test Technique Results for Benchmark Circuits*

Circuit	#PI/#PO	gates	$\#F$	$\#T_F$	F_{tar}	T_v	RT_v	TVR %
Ripple Carry Adder	65/33	160	1024	296	71.09%	242	104	42.97%
Carry-Skip Adder	65/33	160	924	186	79.87%	184	96	52.17%
Carry Look-Ahead Adder	65/33	160	873	225	74.22%	216	89	41.20%
Kogge Stone Adder	64/33	839	1789	534	70.15%	313	134	42.81%
Han Carlson Adder	64/33	655	1415	410	71.02%	201	65	32.33%
Brent Kung Adder	64/33	745	1278	325	74.56%	317	156	49.21%
Ladner-Fisher Adder	64/33	545	1178	125	89.38%	210	96	45.71%
4-Operand Adder	64/18	614	1434	556	60.59%	292	162	55.47%
Wallace Multiplier	16/16	641	1641	285	82.63%	339	123	36.28%
Dadda Multiplier	16/16	641	1641	263	83.97%	275	52	18.90%
Array Multiplier	16/16	610	1585	321	79.74%	316	124	39.24%
ISCAS-85 Benchmark Circuits								
c880 (8-bit ALU)	60/26	383	1074	293	72.71%	15	6	40.00%
c2670 (12-bit ALU and controller)	233/140	1193	1950	587	69.89%	44	19	43.18%
c3540 (8-bit ALU)	50/22	1669	2657	394	85.17%	63	31	49.20%
c5315 (9-bit ALU)	178/123	2307	4224	1341	68.25%	136	63	46.32%
c6288 (16×16 multiplier)	32/32	2406	2314	541	76.62%	94	39	41.48%
c7552 (32-bit adder/comparator)	207/108	3512	4490	1184	73.63%	148	71	47.97%

Here, targetted fault percentage refers to the percentage of faults considered for testing. Finally, the last three columns specify the total number of test vectors (T_v), reduction in test vector (RT_v) after ATPG is applied, and test vector reduction percentage, respectively. The analysis result shows that we have a reduced amount of targetted fault compared to conventional testing. Therefore, our approach provides a quite good reduction in the test vector. On average 40-50% reduction in the test vector is achieved by this method.

4.1.3.2 Image Compression

We assess our technique by implementing three 16-bit adder circuits (Ripple Carry Adder, Kogge Stone Adder, Han Carlson Adder) used to build a DCT (Discrete Cosine Transform) for image compression. All experiments are executed using a system having a Core i5 processor with 8GB RAM. We realize all the adder circuits through Verilog code written in Xilinx Vivado 2018.1 design suite. The implemented designs are extracted through the Vivado System Generator Tool, and then we run it in Matlab 2017b. The approximated regions are identified using our proposed approach, and then faults are injected into that region. We took 100 input pairs with a different threshold that varies from 3 to 7. We did not follow any specific rule to define the range of numbers rather we obtained five different ranges (1-1000, 1-20000,10000-20000, 7000-15000, 5000-15000) of numbers. For each realized adder circuit, we generate the black box using the system generator tool with an objective to utilize it to perform the image processing experiment. Before processing an image, we replace all the standard adder circuit with the proposed one by remodeling the 1D DCT architecture. Then we link this 1D DCT to create a comprehensive 2D transform. The outcome obtained from the above process is passed through an inverse DCT function, and the result is a compressed image. Finally, we perform an experiment using the Lena image from [99] to ensure that the stated method does not produce an output, which may not be acceptable. Figure 4.8 shows the output obtained after performing the experiment and has not violated the claim made by our approach. As per the output, we can say that even some parts of an exact circuit are left untested, it does not affect the behaviors of human perception towards image clarity. The image generated with the injected fault on RCA (Figure 4.8b) produces better visual clarity than the other two images shown in Figure 4.8 (c) and (d).

4.2 Retesting Defective Circuits using Approximation Technique

In a conventional test procedure, test patterns are run sequentially, and the process stops applying the test patterns when the CUT produces an incorrect result. This procedure is known as SOFE (Stop on First Error). But we can not ignore the fact that the faulty circuit identified above may produce error-free output for the remaining test patterns. In other words, the circuit that contains a defect but yet generates a satisfactory result can be usable. We call these circuits as Acceptable Circuits (AcICs). However, we found no technique to identify the AcICs through testing while considering the conventional testing procedure. That's why, in this work, we propose a retesting technique that will identify the AcICs from

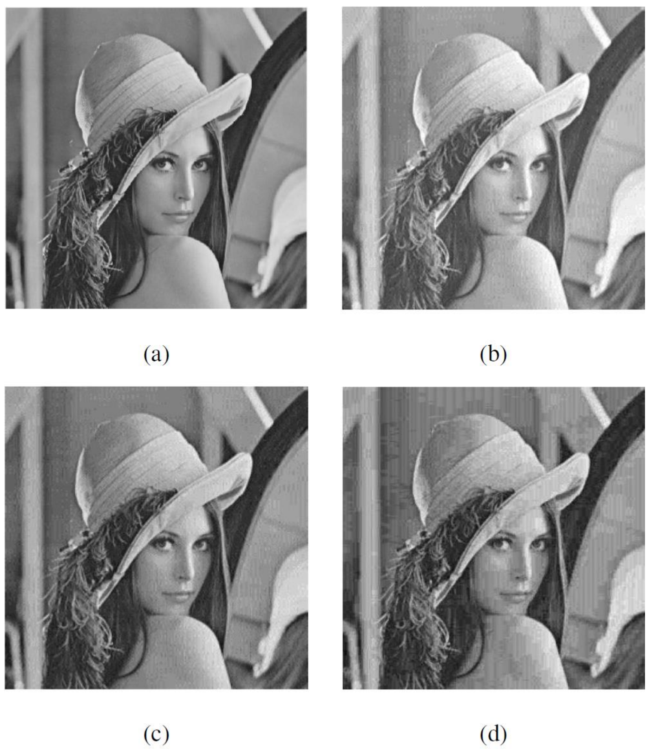


Figure 4.8: *Compressed Lena image produced using the proposed model (a) Conventional DCT (b) RCA with injected fault (c) KSA with injected fault (d) HCA with injected fault*

the faulty (rejected) circuits. Approximate testing [60] is a new approach to testing that is designed to reduce the time and cost of testing. It involves testing the product against a set of approximate test cases. These test cases are designed to be less strict than the conventional test cases. They allow for some degree of error in the product. In this work, we use the idea of approximate testing to test AcICs.

Another benefit of this work is *yield improvement*. According to the current scenario of technology scaling, manufacturing yield is measured in terms of the number of perfect chips produced [79]. However, an imperfect chip producing a good-enough result can also be considered and helps enhance the yield. Hence, identifying those acceptable circuits (AcIC) through retesting increases the *effective yield* indirectly.

Our approach uses the Hamming distance to calculate *Fault Pay-off* (F_{po}). F_{po} is a quantification measure used here to classify the faults. From the experimental analysis, we found that the location with the highest F_{po}, if it contains a fault, will likely affect more than 50% of the output lines (Hamming Distance is more than 50%). Hence, in the approximate-test phase, we retest the faulty circuits by generating the test patterns for the faults in the location having the lowest F_{po}. To show the benefit of our proposed method in terms of yield gain, we devise a yield model. According to the analysis of the yield model, we found that, on average, 10-20% of circuits are found acceptable.

4.2.1 Motivation and Analysis

According to [100], the error-tolerability of an AI system is more significant than that of a human being. A human sensor system may not react to a minor change in the pixel quality of an image, but if it contains more noise, it is unacceptable. However, an AI system may not complain even if the image contains more noise. It is illustrated using Figure 4.9 (c) where a pedestrian is still be detected by an AI system, but a human being may conclude it as an erroneous image. Hence, AI systems are more error-tolerant. From this, we can conclude that the circuits that make an AI system and fail during conventional test flow are still usable if it produces some minor error during regular operation. As per the definitions provided in Table 1.2, both AcICs and AxICs can be employed in AI systems because both can tolerate some amount of error.

Two major components are involved in identifying an AcIC. The first component is the *quantification measure* (either error-significance or error-rate, and sometimes both), and the second is a *fault model*. The analysis presented in this section is based on the following fundamental questions and our contribution includes the solutions to these questions.

- Which quantification measure correctly determines the acceptability of a circuit irrespective of its type?

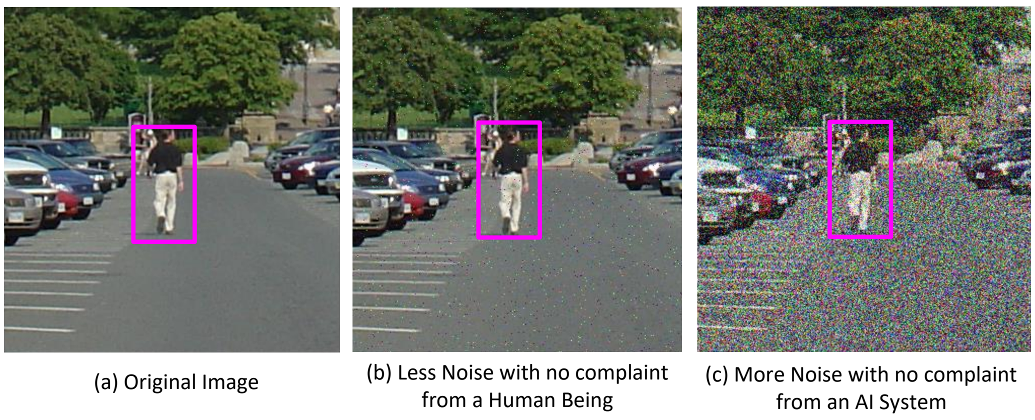

(a) Original Image
(b) Less Noise with no complaint from a Human Being
(c) More Noise with no complaint from an AI System

Figure 4.9: *Motivating Example*

- Is it required to choose a fault model, and if not, how it affects the determination of AcICs?
- Whether to choose the existing ATPG algorithm or a new algorithm is needed to determine the circuit's acceptability?

4.2.1.1 Analysis-1

Most works of literature use error-rate and ignore the requirement of a fault model [76,77,101–105]. They calculate the fraction of input patterns that produce an erroneous output and ignore the *fault's severity* (also known as the overall fault effect). The severity of a fault may reduce the effectiveness of these techniques. Let us define the fault severity here, and a theoretical definition is given in section 4.2.2.1. The severity of a fault is defined as the product of two terms: (1) The number of test patterns that excites the fault effect to the output, and (2) The addition of the Hamming distance produced by each test pattern. For instance, consider the circuit C shown in Figure 4.10 (a) with two faults, f_1 and f_2, where f_2 is more disastrous than f_1. Let t be the total number of test patterns required to test the given circuit C, out of which s test patterns generate incorrect output due to f_1, and k test patterns produce incorrect output due to f_2. Assume that,

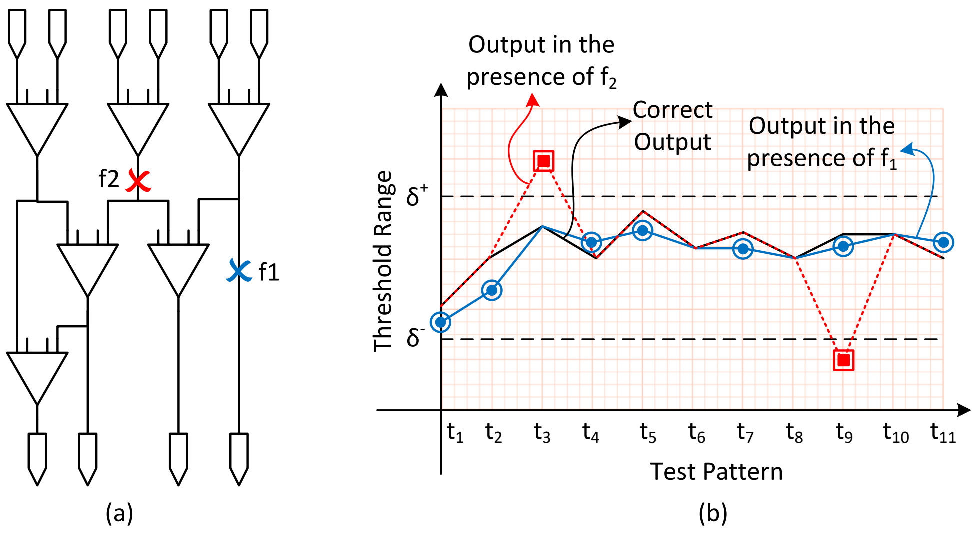
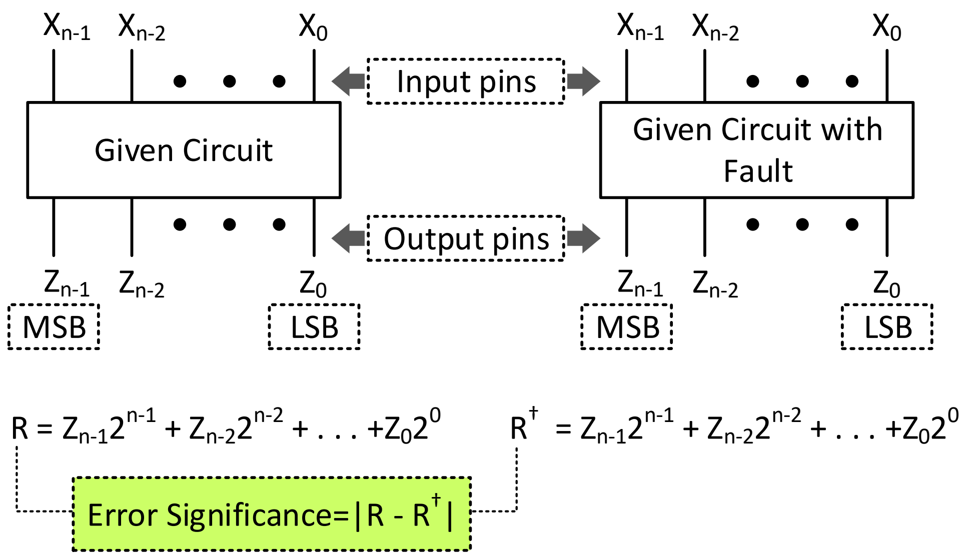

f2
f1
(a)
Output in the presence of f_2
Correct Output
Output in the presence of f_1
Threshold Range
δ^+
δ^-
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11}
Test Pattern
(b)

Figure 4.10: *Example Circuit to explain Fault Severity*

$|k| << |s|$, then according to the current technique, f_1 is considered more severe than f_2. Figure 4.10 (b) shows the scenario: where t=11, t_1 through t_{11} test patterns are applied to test the circuit. The circuit produces the wrong output 7 times ($s = 7$) due to fault f_1, 2 times ($k = 2$) due to f_2. The error rate in the presence of f_1 will be 63% and 18% for f_2. According to the techniques proposed so far, f_1 is estimated as more catastrophic than f_2. But according to our assumption, the procedure must identify f_2 as more severe than f_1. We are claiming this because of the predefined threshold indicated in Figure 4.10 (b) as δ^+ and δ^-. The incorrect output produced due to f_2 is beyond the threshold. Hence, we need a different technique based on the fault model and the severity of each fault. *The conclusion is that the current method considers only the fraction of test patterns that produce an incorrect result, but our approach is based on the fault model and the severity of each fault.*

4.2.1.2 Analysis-2

Some works of literature use error-significance as a measurement to identify the acceptability of a circuit [71, 79–81, 106, 107]. Others, like [82], considers both error-significance and error-rate (SBER). Error-significance of these circuits is defined as the numerical difference between the actual output and the erroneous output. The desired circuits' output pins are marked as MSBs and LSBs. Compared to MSBs, LSBs are assigned the lowest numerical significances. Figure 4.11 shows the error-significance calculation procedure. R represents the actual output, and $R^{\dagger}$ represents the output produced by the faulty circuit. In this approach, a predefined *threshold* is used to classify the faults. The major problem with these techniques is that we can not use it to classify faults for all types of circuits. For instance, circuits like an adder have the potentiality of identifying MSBs and LSBs. Most arithmetic circuits allow us to mark the pins as LSBs and MSBs, but several circuits (other than arithmetic circuits) exist for which distinguishing impact of MSBs and LSBs on the overall output is different. For example, let us consider a random circuit with four output lines, as shown in Figure 4.12. In Figure 4.12(a), we choose pin 5 as the LSB and pin 8 as MSB, and accordingly, we allocate the significance, 2^0 through 2^3, respectively. In Figure 4.12(b), the other way is chosen, i.e., pin 8 is referred to as LSB and pin 5 as MSB. In such a situation, the significance of the output pins vary depending on the marking of MSB and LSB. However, the significance of an output pin should be decided based on an output pin's *criticality*. Here, criticality refers to the effect of the circuit's overall output due to a wrong result produced by the considered output pin. To elaborate on the situation, assume that the criticality of pin 8 is higher than pin 5. Hence, the significance of pin 8 and 5 should be 2^3 and 2^0, respectively. Figure 4.12(a) shows this ideal situation. According to the current literature analysis, the worst-case state can occur if the technique chooses pin 5 as MSB and pin 8 as LSB irrespective of its criticality, as shown in Figure 4.12(b). Table 4.3 shows the output differences between both these cir-



X_{n-1} X_{n-2} X_0
Input pins
Given Circuit
Given Circuit with Fault
Output pins
Z_{n-1} Z_{n-2} Z_0
MSB
LSB
$R = Z_{n-1}2^{n-1} + Z_{n-2}2^{n-2} + \ldots + Z_0 2^0$
$R^{\dagger} = Z_{n-1}2^{n-1} + Z_{n-2}2^{n-2} + \ldots + Z_0 2^0$
Error Significance=$|R - R^{\dagger}|$

Figure 4.11: *Showing absolute numerical difference (error-significance) between output of original circuit and the faulty circuit*

cuits based on the calculation shown in Figure 4.11. The last column shows the absolute numerical difference between these circuits. In 75% of cases, the outputs are wrong, and in 50% of cases, the absolute numerical difference is more than 2. The analysis is presented for only a 4-output pin circuit, and if the number of output pins increases, then the situation will worsen. *So the conclusion is that we should wisely choose the MSB and LSB of output pins while considering non-arithmetic circuits. This work avoids determining the MSB and LSB bits because our approach is purely based on the fault model and its impression on the output pins.*

4.2.2 Proposed Approach

This section will discuss the proposed two-phase testing architecture to determine all the acceptable circuits (AcICs). First of all, we will describe the fundamental principle that drives our proposed approach, which also explains why our tech-

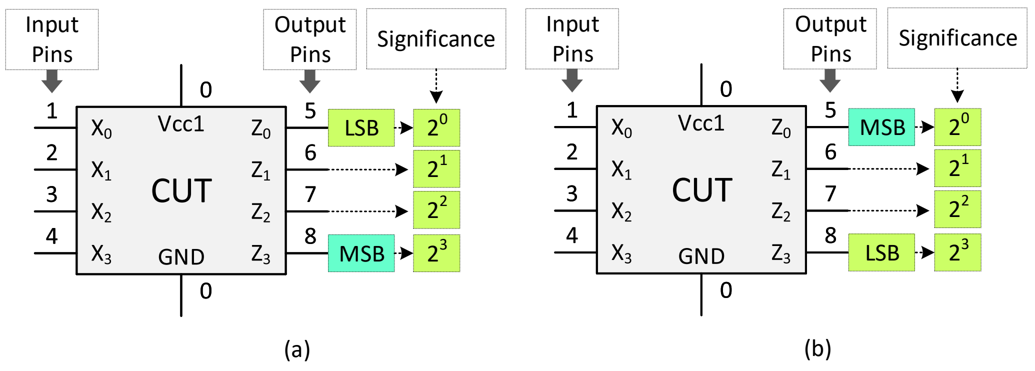


Figure 4.12: *Imaginary Circuits under Test (a) Circuit showing Z_0 as LSB and Z_3 as MSB (b) Circuit showing Z_3 as LSB and Z_0 as MSB*

Table 4.3: *Output analysis of the example circuit*

Z_3	Z_2	Z_1	Z_0	R	Z_3	Z_2	Z_1	Z_0	$R^\dagger$	$\lvert R - R^\dagger \rvert$
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	8	$\boxed{7}$
0	0	1	0	2	0	1	0	0	4	$\boxed{2}$
0	0	1	1	3	1	1	0	0	12	$\boxed{9}$
0	1	0	0	4	0	0	1	0	2	$\boxed{2}$
0	1	0	1	5	1	0	1	0	10	$\boxed{5}$
0	1	1	0	6	0	1	1	0	6	0
0	1	1	1	7	1	1	1	0	14	$\boxed{7}$
1	0	0	0	8	0	0	0	1	1	$\boxed{7}$
1	0	0	1	9	1	0	0	1	9	0
1	0	1	0	10	0	1	0	1	5	$\boxed{5}$
1	0	1	1	11	1	1	0	1	13	$\boxed{2}$
1	1	0	0	12	0	0	1	1	3	$\boxed{9}$
1	1	0	1	13	1	0	1	1	11	$\boxed{2}$
1	1	1	0	14	0	1	1	1	7	$\boxed{7}$
1	1	1	1	15	1	1	1	1	15	0

nique is better than the other technologies developed so far. Then we will describe the architecture flow of our method in subsequent sections.

4.2.2.1 Fundamental Principle

Considering a rejected circuit that fails the conventional manufacturing test procedure due to some fault in the chip, our approach is based on two fundamental principles:

- Not all input patterns propagate the fault effect to the output.
- Even if some input patterns propagate the fault effect; it may not affect all output lines.

Let the given circuit C have m input lines and n output lines. During the usage, the circuit may come across 2^m patterns for which the circuit needs to produce the correct result. Assume that a fault f_i is present in the circuit, and k input patterns drive the fault effect to the output. Hence, $2^m - k$ patterns do not propagate the fault effect, which means for the $2^m - k$ patterns, the circuit produces the correct output. Again, assume that the k input patterns that drive the fault effect, alter r output lines (r may vary for each k_i). We can quantify it by finding the *Hamming Distance* between n and $n^\dagger$, where n represents the set of all output lines and $n^\dagger$ represents the set of output line with some output lines get altered due to the fault effect. We need to calculate the value of r for each k_i using the formula $\sum_{i=1}^{k} HammingDistance(n, n^\dagger)$. Only finding the summation of Hamming distance is not sufficient to identify the catastrophic fault. Consider a situation where two faults ($f1$ and $f2$) having the same Hamming distance. When this situation arises, we check the value of k; the number of input patterns drives the fault effect to the output. For instance, at the presence of $f1$, let 3 test patterns produce the wrong output with Hamming distance 3, 2, 2 (3+2+2=7). Similarly, at the presence of $f2$, let 5 test patterns produce the wrong output with Hamming distance 1, 2, 1, 2, 1 (1+2+1+2+1=7). The summation of Hamming distance for both faults is 7. Note that due to $f2$, 5 test patterns produce the wrong output. Whereas in the presence of $f1$, 3 test patterns produce incorrect output. Hence, $f2$ is assumed to be more catastrophic than $f1$. Multiplying k (k is 3 for $f1$ and 5 for $f2$) makes it easy to identify this. Finally, we can calculate the overall fault effect (OFE) due to the presence of fault f_i using Equation 4.3.

$$OFE(f_1) = k \times \sum_{i=1}^{k} HammingDistance(n, n^{\dagger}) \tag{4.3}$$

Let us take an example to understand the overall fault effect of two faults, f_1 and f_2, that occur at different regions of the circuit $C(m = 5, n = 10)$ but not at the same time. Let at the presence of f_1, 3 test patterns produce the wrong output, which means $k = 3$. For each k, the Hamming distance will vary. Let Hamming distances for each k be 3, 2, 2, respectively. Hence, the $OFE(f_1)$ will be $3 \times (3+2+2) = 21$. Similarly, let in the presence of f_2, 4 test pattern produce wrong output with Hamming distance 1, 1, 1, 1, respectively. Then the $OFE(f_2)$ will be $4 \times (1 + 1 + 1 + 1) = 16$. From the above example, we can conclude that the presence of f_1 causes the circuit to behave more incorrectly than f_2.

Figure 4.13 shows the overall test flow architecture of our proposed technique. It contains two phases: (1) Modified Conventional Test Flow (CTF_m) and (2) Approximate Test Flow (ATF). In CTF_m, we have added a new block known as the *fault analysis block* in the design phase, which is the significant contribution of this work. First, we discuss the fault analysis technique, and then we will describe the subsequent stages.

4.2.2.2 Fault Analysis

It is the process of determining a region (net), where if a fault exists (we consider only stuck-at-fault), how many input patterns can propagate the fault effect to the output line and how many output lines are affected by the fault propagation. It serves three objectives:

- Calculate the fault pay-off (F_{po}),

- Classify the fault according to the value of F_{po}, and
- Determine the fault which acts as the primary input to the second phase.

Let us consider a given gate-level netlist $C(G, N)$, where G represents the set of gates, and N represents the set of nets (channels) connected between gates. Here we will use the terms net, line, and channel interchangeably. Theoretically, we can represent N, as $N = \{n_1, n_2, ..., n_l\}$, where, n_j represents a single net, and each fan-in branch is treated individually, which means a fault can occur at any branch. If faulty, each n_j is either affected with Stuck-at-0 (*sa0*) fault or Stuck-at-1 (*sa1*) fault. Hence, we represent n_j as a three tuple variable, $n_j \langle sa0, sa1, ff \rangle$, where $sa0$ represents Stuck-at-0, $sa1$ represents Stuck-at-1, and ff represents fault-free states. A value 1 at any tuple represents the existence of that fault at n_j, and a value 0 denotes its nonexistence. For instance, $n_j \langle 1, 0, 0 \rangle$ means that n_j is faulty and is affected by sa0. Similarly, $n_j \langle 0, 0, 1 \rangle$ means the channel is fault-free. Our objective is to identify the faulty channel whose fault effect is trivial. That is why we add another tuple to each n_j, which is known as t, and t represents whether the fault effect is trivial or non-trivial. Here, non-trivial means the fault effect is catastrophic and trivial means we can ignore the effect of this fault. If $t = 1$, the fault is trivial, whereas $t = 0$ means the fault is non-trivial. Finally, every channel is represented as $n_j \langle sa0, sa1, ff, t \rangle$. For instance, $n_j \langle 1, 0, 0, 1 \rangle$ means that n_j is faulty, affected by sa0 and is trivial.

The faults for which the effects are non-trivial are known as unacceptable faults and should be listed. The remaining faults (whose effects are trivial) can be ignored. The triviality of a fault effect is known as *fault pay-off (F_{po})*, and we calculate the F_{po} for each fault using Equation 4.4 and 4.5.

$$F_{po}^{n_i\langle 1,0,0,t\rangle} = \left| TP_{n_i\langle 1,0,0,t\rangle} \right| \times TFE_{n_i\langle 1,0,0,t\rangle} \tag{4.4}$$

$$F_{po}^{n_i\langle 0,1,0,t\rangle} = \left| TP_{n_i\langle 0,1,0,t\rangle} \right| \times TFE_{n_i\langle 0,1,0,t\rangle} \tag{4.5}$$

where $TP_{n_i\langle 1,0,0,t\rangle}$ and $TP_{n_i\langle 0,1,0,t\rangle}$ represent the set of test patterns that detect sa0 and sa1 at n_j, respectively. Total Bit-flip Error (TFE) is the number of modified bit positions (Hamming Distance) while comparing the output bits of the circuit C with the output produced in the presence of a fault. TFE is calculated by the formula given in Equation 4.6.

$$\forall TP_f^k, TFE = \sum_{i=0}^{n-1} \left(C_{Z_{n-1}} ... C_{Z_0}\right) \oplus \left(C^f_{Z_{n-1}} ... C^f_{Z_0}\right) \tag{4.6}$$

where $Z_{n-1}...Z_0$ represent the output lines of the corresponding circuits, and f represents either $\langle 1,0,0,t\rangle$ or $\langle 0,1,0,t\rangle$. k varies from 1 to the total number of test patterns that detect f. That means, TFE is calculated for each test pattern that detects f. Initially, the triviality of a fault is unknown, so the value of t is unknown for Equation 4.4, 4.5, and 4.6.

The triviality of a fault can only be calculated by analyzing the fault pay-off (F_{po}). Our approach will calculate the F_{po} for every fault and build a sorted Fault Analysis set (FAs), where each entry represents a tuple with two elements: the fault and the fault pay-off. For instance, $\left(n_j \langle 0,1,0,t\rangle, F_{po}^{n_j\langle 0,1,0,t\rangle}\right) = \left(n_2 \langle 0,1,0,t\rangle, 135\right)$ represents a stuck-at-1 fault at the net n_2 and having the fault pay-off as 135. Algorithm 2 shows the detailed steps to generate the FAs. Initially, the FAs set is empty, as shown in step-1 of Algorithm 2. Every time one iteration completes the loop mentioned in lines number 2 and 12, one fault and its pay-off get added to the FAs mentioned in lines number 10 and 20 for stuck-at-0 and stuck-at-1 fault, respectively. Line numbers 3 and 13 represent the total number of test patterns that propagate the fault effect to the output line for s-a-0 and s-a-1, respectively. When the fault effect is propagated to the output line, we get a different output than the actual. To calculate the deviation in the output, we use the concept of Hamming Distance. Line number 6 (for s-a-0) and 16 (for s-a-1) calculate the Hamming Distance using Equation 4.6. The deviation in the output is calculated for each test pattern. That is why line number 5 and 15 is a loop that iterates for s-a-0 and s-a-1, respectively. Line number 9 and

ALGORITHM 2: Fault Analysis

```
   Input: Simulated Gate-Level Netlist $C$, Test Pattern $TP$
   Result: Fault Analysis Set ($FA_s$)
   // Initially Fault Analysis Set is Empty
1  $FA_s \longleftarrow \{\phi\}$;
   // Calculating Fault Pay-off for each stuck-at-0 fault occurs
      in every net
2  for $j$=1 to $l$ do
3      $k = \left|TP_{n_i\langle 1,0,0,t\rangle}\right|$
4      $r = 0$
5      for $s$=1 to $k$ do
6          $TFE_{n_j\langle 1,0,0,t\rangle} = \sum_{i=0}^{n-1} \left(C_{Z_{n-1}}...C_{Z_0}\right) \oplus \left(C_{Z_{n-1}}^{n_j\langle 1,0,0,t\rangle}...C_{Z_0}^{n_j\langle 1,0,0,t\rangle}\right)$
7          $r = r + TFE_{n_j\langle 1,0,0,t\rangle}$
8      end
9      $F_{po}^{n_j\langle 1,0,0,t\rangle} = k \times r$
10     $FA_s = FA_s \bigcup \left\{\left(n_j\langle 1,0,0,t\rangle, F_{po}^{n_j\langle 1,0,0,t\rangle}\right)\right\}$
11 end
   // Calculating Fault Pay-off for each stuck-at-1 fault occurs
      in every net
12 for $j$=1 to $l$ do
13     $k = \left|TP_{n_i\langle 0,1,0,t\rangle}\right|$
14     $r = 0$
15     for $s$=1 to $k$ do
16         $TFE_{n_j\langle 0,1,0,t\rangle} = \sum_{i=0}^{n-1} \left(C_{Z_{n-1}}...C_{Z_0}\right) \oplus \left(C_{Z_{n-1}}^{n_j\langle 0,1,0,t\rangle}...C_{Z_0}^{n_j\langle 0,1,0,t\rangle}\right)$
17         $r = r + TFE_{n_j\langle 0,1,0,t\rangle}$
18     end
19     $F_{po}^{n_j\langle 0,1,0,t\rangle} = k \times r$
20     $FA_s = FA_s \bigcup \left\{\left(n_j\langle 0,1,0,t\rangle, F_{po}^{n_j\langle 0,1,0,t\rangle}\right)\right\}$
21 end
22 Return $FA_s$
```

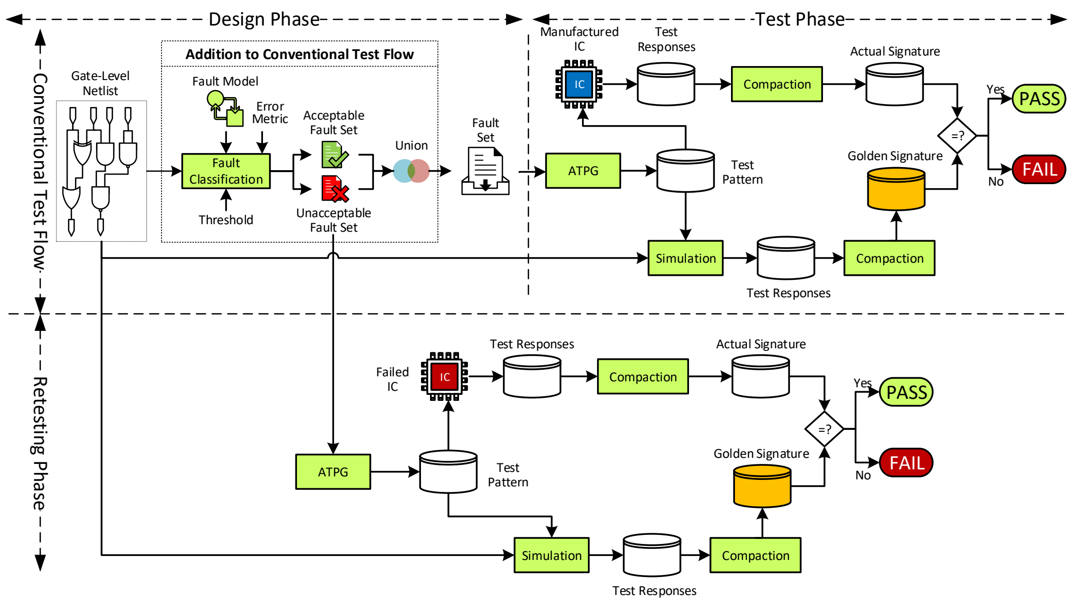


Figure 4.13: *Proposed Test Flow Architecture*

19 calculate the F_{po} for s-a-0 and s-a-1 using Equation 4.4 and 4.5, respectively. Finally, Line number 14 returns the FAs.

4.2.2.3 Fault Classification

The next step in the fault analysis phase is to classify the fault into acceptable and unacceptable ones. According to Algorithm 2 , we have a fault set FA_s that contains the fault list and its fault pay-off. Needless to say, the highest fault pay-off leads to a large variation in the circuit's output. The primary component in calculating the fault pay-off is the Hamming distance. Thus, in our classification technique, TFE plays a significant role. If half of the output bits are altered due to a fault's presence, we consider it unacceptable. Classifying fault is the main component of our fault analysis phase.

A careful selection of faults help us achieve a better result. We have the fault analysis set with all faults and their associated fault pay-offs. Our objective is to classify these faults into benign and malignant. Let us use a simple notation to represent the set FA_s, which is shown in Equation 4.7.

$$FA_s = \Big\{(n_1, F^1), (n_2, F^2), (n_3, F^3), ..., (n_i, F^i)\Big\} \tag{4.7}$$

where, n_i represents the net, and F^i represents the fault pay-off of each n_i. Our algorithm classifies it to two disjoint set, FA_s^{benign} and $FA_s^{malignant}$ such that:

$$FA_s^{benign} \bigcap FA_s^{malignant} = \emptyset \tag{4.8}$$

First, we will find F^i_{max}, the highest fault pay-off of any n_i, and add it to the malignant set. This is because of the fundamental principle explained in Section 4.2.2.1. The next step in our algorithm is to choose δ, a numeric value that defines the distance from F^i_{max}. Finally, we follow the Algorithm 3 (Line no. 5 to 11) to classify the fault as benign or malignant. In each iteration, an element is chosen from the set FA_s and compared with $F^i_{max} - \delta$. If the selected element's fault pay-off is less than or equal to $F^i_{max} - \delta$, then it is included in set $FA_s^{malignant}$, else assigned to set FA_s^{benign}. Choosing a value for delta requires the proper analysis of all the faults and its fault pay-off. We explain this process using a case study in Section 4.2.3. Algorithm 3 starts with the initialization of two empty sets FA_s^{benign} and $FA_s^{malignant}$ at the line numbers 1 and 2, respectively. Line numbers 3 and 4 find an element that belongs to FA_s with the highest fault pay-off returned from the function $findMax$ and inserts that element to the set $FA_s^{malignant}$.

4.2.2.4 Conventional Test Flow

In our proposed method (Figure 4.13), the first phase is the conventional test flow. A significant addition to this phase is fault analysis and classification, discussed in Sections 4.2.2.1 and 4.2.2.2. The rest of the components in this phase are standard and straightforward. During the *design phase*, the designer must decide on the *fault model* and generate a fault set. This fault list is tested during the testing phase, and the fault coverage is analyzed. For clarity, we are elaborating the process flow of fault analysis phase here in more detail.

ALGORITHM 3: Fault Classification

```
Input: Fault Analysis set $FA_s$ generated in Algorithm 2 and $\delta$, a
       numeric value that defines the distance from $F^i_{max}$
Result: $FA_s^{benign}$, acceptable fault set and $FA_s^{malignant}$, unacceptable
        fault set
   // Initially both sets are Empty
1  $FA_s^{benign} \longleftarrow \{\phi\}$;
2  $FA_s^{malignant} \longleftarrow \{\phi\}$;
   // Initializing the malignant fault set
3  $F^i_{max} = findMax(FA_s)$
4  $FA_s^{malignant} = FA_s^{malignant} \bigcup F^i_{max}$;
   // classify the fault as benign or malignant
5  for $F^j \in FA_s$ do
6      if $F^j \geq F^i_{max} - \delta$ then
7          $FA_s^{malignant} = FA_s^{malignant} \bigcup F^j$
8      else
9          $FA_s^{benign} = FA_s^{benign} \bigcup F^j$
10     end
11 end
12 Return $FA_s^{benign}$ and $FA_s^{malignant}$
```

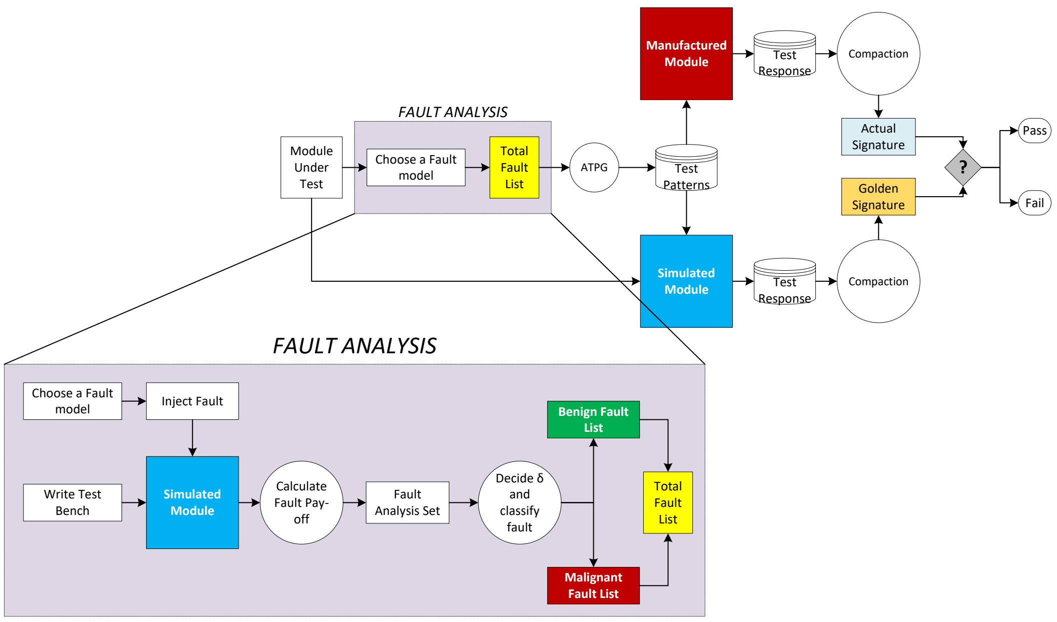


Figure 4.14: *Traditional Test Flow Architecture With Enlarged View of the Fault Analysis Steps*

Figure 4.14 shows the traditional test flow architecture along with the position of the fault analysis step. The Module Under Test (MUT) is either a monolithic circuit or represents a SOC module. First, we decide on a fault model, which is the single stuck-at-fault model in our case. Then for the entire fault list, an ATPG generates the test patterns. These test patterns are applied to the simulated module, and the *test responses* generated from this process are captured as *golden signatures* using the *compaction* technique. On the other hand, the same test pattern is applied to the *manufactured module*, and the test response is compacted to form the *actual signature*. Finally, the actual signature is compared with the golden signature to decide whether to accept (*pass*) or reject (*fail*) the manufactured unit. Figure 4.14 also shows an enlarged process flow of the fault analysis step. This step requires the simulated design of the MUT, where we can inject faults one by one. We also wrote a test bench that generates random test patterns and applied them to the simulated module. Our algorithm calculates the impact on the output for every injected fault, which is quantized as fault pay-off. We store every fault, and its pay-off in a set called fault analysis set. In our case, we get two sets of faults: (1) benign fault set and (2) malignant fault set. Before the conventional test phase starts, we combine (*union*) these two sets to form a single *fault set*.

4.2.2.5 Approximate Test Phase

The approximate test phase is an addition to the conventional test flow, and its objective is to identify the acceptable circuits from the rejected ones. The phase starts with the generating the test set for the malignant fault set (listed during fault analysis) using any ATPG method. The rest of the steps are similar to the conventional test flow. The test patterns are applied to the simulated model, and the *test responses* generated from this process are captured as *golden signatures* using the *compaction* technique. On the other hand, the same test pattern is applied to the *failed ICs* (rejected during conventional test flow), and the test response is compacted to form the *actual signature*. Finally, the actual signature is compared with the golden signature to decide whether to accept (*pass*) or reject (*fail*) the failed ICs. The ICs that are accepted are known as Acceptable ICs (AcICs).

4.2.3 Case Study

To further explain the procedure and realize our technique's effectiveness, we will describe it using a case study. Figure 4.15 shows the C17 benchmark circuit with the fault locations marked using cross marks. A cross mark represents either a stuck-at-0 or a stuck-at-1 fault. For simplicity, we have not used any fault collapsing method to reduce the number of faults; even collapsing does not hamper the performance of our technique. The circuit has five input lines (a, b, c, d, e) and two output lines (p, q). The remaining lines from f to o represent the intermediate lines. We consider the stuck-at fault model, and any line among them can be affected with a stuck-at-0 or a stuck-at-1 fault. In total, we examine

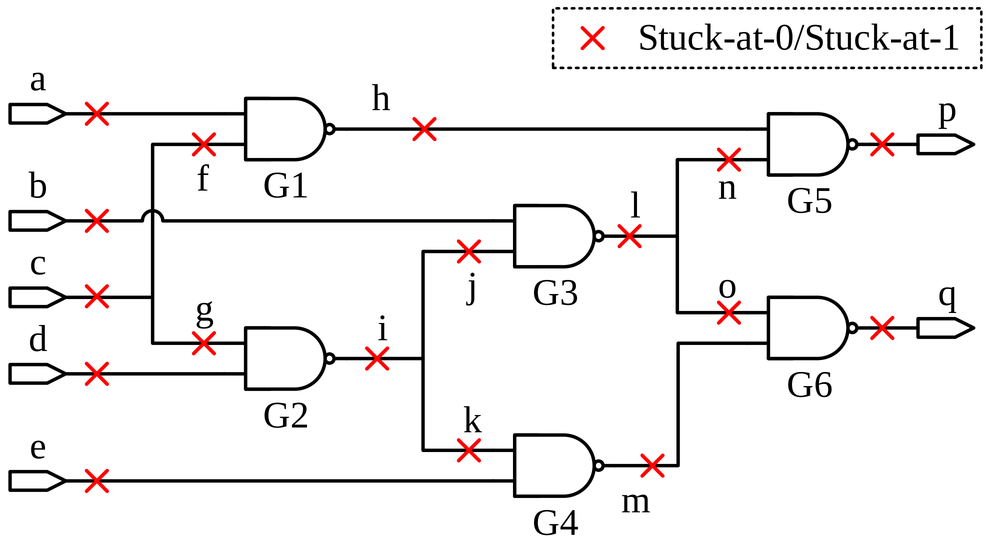


Figure 4.15: *C17 Circuit showing the fault locations (nets) marked with cross symbols. Each cross symbol either represents a stuck-at-0 or a stuck-at-1 fault*

all 34 faults for our case study.

After designing the circuit with Verilog, we injected one fault at a time and tested the impact by supplying a set of exhaustive test patterns. For C17, we used all 32 test patterns and calculated the fault fay-off. Figure 4.16 shows the result of the fault analysis and the fault pay-offs. The figure also shows the value of k: the number of test patterns propagating the fault effect to the output line. From the analysis, we found that the highest fault pay-off is 532 due to the presence of sa0@l fault. On the other hand, the lowest fault pay-off is 16 due to both sa1@f and sa1@k fault. An essential observation we noticed here is that the fault pay-offs of several faults are similar or close to each other. For instance, sa0@h, sa0@m, sa0@n, sa0@o, sa1@p, and sa1@q all have the same fault pay-offs 196. Similarly, sa0@a, sa1@a, sa0@f, sa1@h, sa0@k, sa1@m, and sa1@o have 36 as their fault pay-offs. This observation occurs not only with C17 but also with every other ISCAS 85 benchmark circuit that we studied for our experimental

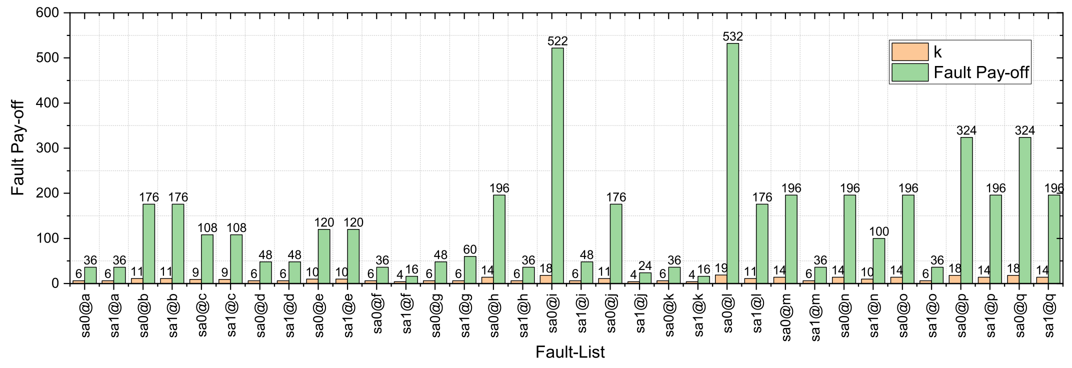


Figure 4.16: *C17 circuit's fault analysis showing the fault list and the fault pay-off*

analysis in Section 4.2.4. This also helps in the effective categorization of faults into benign and malignant.

The next step in the process is to categorize the fault into benign and malignant. Initially, both the set $FA_s^{malignant}$ and FA_s^{benign} is empty. For C17 circuit, the sets look as follows:

$FA_s = \Big\{\big(sa0@a, 36\big), \big(sa1@a, 36\big), \big(sa0@b, 176\big), ..., \big(sa1@q, 196\big)\Big\}$

$FA_s^{malignant} = \Big\{\phi\Big\}$

$FA_s^{benign} = \Big\{\phi\Big\}$

We follow Algorithm 3 discussed in Section 4.2.1 for this classification. The first step in this process is to find the maximum fault pay-off and store it in F_{max}^l, which is 532 for the fault sa0@l. Then we add this fault along with its fault pay-off to the set $FA_s^{malignant}$. After the above assignment, the two sets will look as follows.

$FA_s^{malignant} = \Big\{\big(sa0@l, 532\big), \phi\Big\}$

$FA_s^{benign} = \Big\{\phi\Big\}$

At this point, we need to decide on the value of δ, which helps us identify the fault that needs to be included either in the malignant set or benign set. We did it with a two-step process: (1) Forming the clusters and (2) Finding the distance of each cluster from Fmax. *Forming the Cluster*: It is easier because our analysis says that the fault pay-offs that we calculate for each fault are either the same or

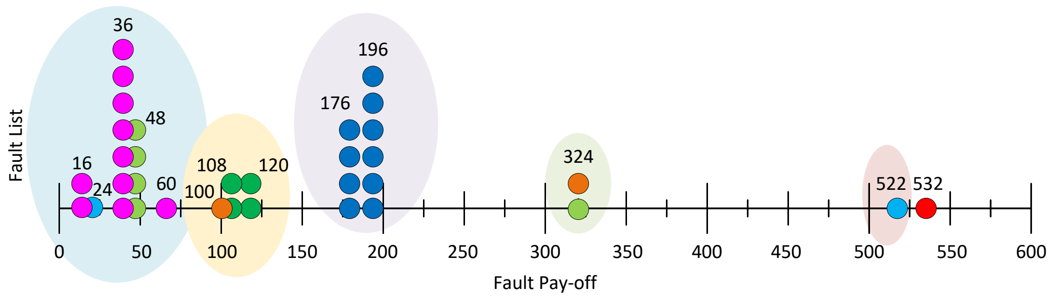


Figure 4.17: *C17 circuit's fault analysis showing fault pay-off clusters in a number line.*

very close to each other. Figure 4.17 shows the fault pay-offs of the C17 circuit when plotted in a line. From the diagram, it is much clear that the fault pay-offs are closer to each other. Using a simple *k-mean clustering*, we found five clusters. *Finding the distance*: In this step, we find the distance of each cluster (mean) from F^i_{max}. Figure 4.18 illustrates the scenario of the C17 benchmark circuit. Clusters identified in Figure 4.17 are plotted in a 2d-plot (Figure 4.18), where the point annotated as F^i_{max} represents the highest fault pay-off. δ_1 through δ_5 shows the distance from F^i_{max} to cluster 1 through 5, respectively. This distance represents the numeric difference between F^i_{max} and the *mean* of each cluster. Table 4.4 lists out the details of these calculations. Column 1 and 2 show the clusters and their mean. Column 4 shows the value of δ for each cluster. To decide on the value of δ (for classification), we need to check the percentage of patterns that alter more than 50% of the output lines. For our experiment, we took 1000 random patterns, applied them on the C17 circuit in the presence of a fault from the clusters, and recorded the variations in the output. The result of this experiment is listed in Column 3. We can read it as, in the presence of sa0@i, 79% of the input patterns alter more than 50% of the output lines. After analyzing the result of Column 3, we choose $\delta = 421$, and accordingly, the faults get classified using Algorithm 3. Column 6 shows this classification for the C17 circuit.

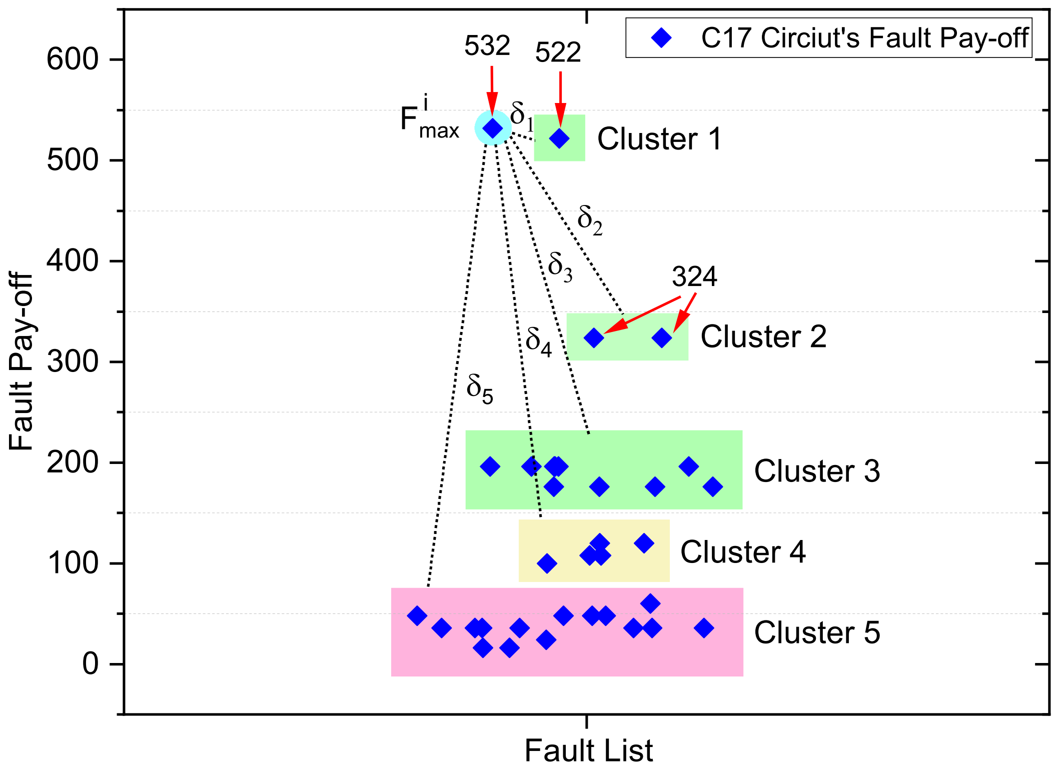


Figure 4.18: *C17 circuit's fault analysis showing fault pay-off clusters. The point annotated with F^i_{max} is the largest fault fay-off with value 532. The remaining fault pay-offs are grouped into 5 clusters. δ_i shows the distance from the point F^i_{max}.*

Table 4.4: *Deciding the value for δ*

Cluster	Mean	Avg.%	δ	Fault List	Class
cluster 1	522	79%	10	sa0@i	Malignant
cluster 2	324	73%	208	sa0@p sa0@q	
cluster 3	188	68%	344	sa0@b sa1@b sa0@j sa1@l sa0@h sa0@m sa0@n sa0@o sa1@p sa1@q	
cluster 4	111	52%	421	sa1@n sa0@c sa1@c sa0@e sa1@e	
cluster 5	37	18%	495	sa1@f sa1@k sa1@j sa0@a sa1@a sa0@f sa1@h sa0@k sa1@m sa1@o sa0@d sa1@d sa0@g sa1@i sa1@g	benign

4.2.4 Evaluation

The first experiment that is conducted is to identify the acceptability and unacceptability of faults for ISCAS-85 benchmark circuits [98, 108]. The test is conducted using the HOPE fault simulator tool [109] for fault simulation. We choose ten benchmarks that include C17, C432, C499, C1355, C880, C1908, C3540, C5315, C6288, and C7552 with primary input lines 5, 36, 41, 41, 60 33, 50, 178, 32, 207, respectively. Applying an exhaustive test pattern to all these circuits is time-consuming; that is why we chose a random test set of size 1000 for analyzing the fault pay-off. In the presence of a fault (injected), 1000 patterns get applied, and the method calculates the pay-off. Finally, we analyze the data to classify the fault. Figure 4.19 shows the fault pay-off result of the selected benchmarks. The X-axis represents the individual faults sorted according to their fault pay-off, and Y-axis represents the fault pay-offs.

After analyzing the fault pay-off, we have summarized the results and classified the faults. Table 4.5 shows this compiled result. Column 1 and 2 show the list of circuits and the number of faults considered for analysis. Columns 3 and 4 show the number of malignant and benign faults. The highest fault pay-off and delta value decided for each circuit are shown in columns 5 and 6, respectively. Finally, we have also tested for the Hamming distance, which is listed in the last column. The result shown in this column represents the average percentage of input patterns that produce more than 50% Hamming distance in the presence of any of the malignant faults in the circuit.

We conducted the following experiment on selected circuits of ITC'99 benchmarks [110]. Seven benchmarks circuits are carefully chosen for this experiment and described in Table 4.6. Because our method is based on the Hamming distance, we choose benchmarks having at least six output lines. We obtained the gate-level netlist of the selected benchmarks from [111]. We chose a random test set of size 10000 for analyzing the fault pay-off. In the presence of a fault (injected), 10000 patterns get applied, and the method calculates the pay-off.

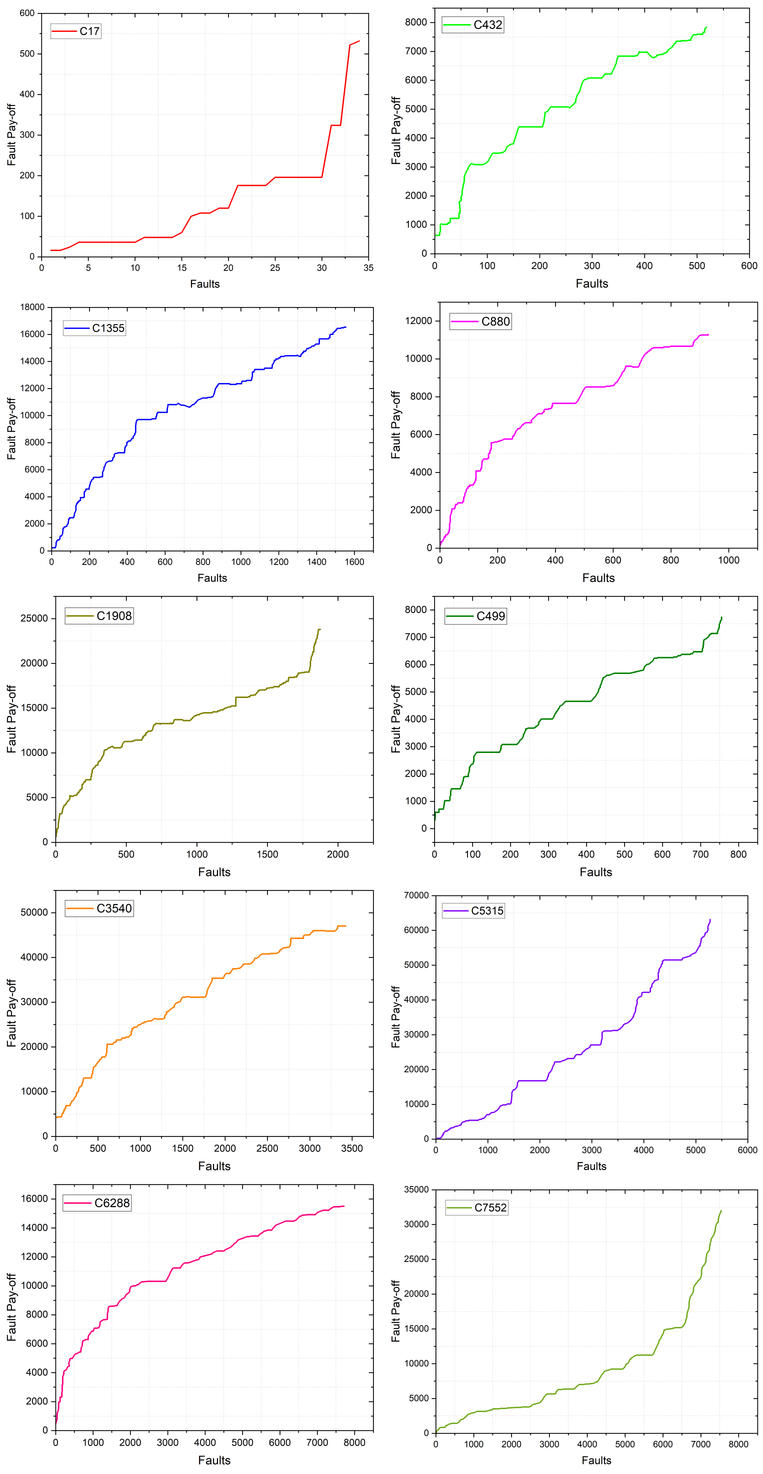


Figure 4.19: *Showing results of fault analysis and their fault pay-off*

Table 4.5: *ISCAS85 Benchmarks Result Analysis and Classification for Faults*

Circuit	# Faults	# Malignant	# Benign	HF_{po}	δ	50% HD
C17	34	19	15	532	421	68%
C432	524	389	135	7856	6238	71%
C499	758	482	276	10893	8456	65%
C1355	1574	818	756	21925	16968	65%
C880	942	638	304	17895	14945	68%
C1908	1879	1023	856	24128	21312	73%
C3540	3428	1952	1476	48436	42234	70%
C5315	5350	3335	2015	63298	57627	73%
C6288	7744	7056	688	15675	12598	68%
C7552	7550	4660	2890	32657	27189	74%

Table 4.6: *ITC'99 Benchmark Circuits*

Name	PI	PO	#Gate	#FF	Fault List
b04	11	8	597	66	3356
b10	11	6	189	17	1054
b11	7	6	481	31	2868
b12	5	6	1036	121	6084
b13	10	10	339	53	1818
b14	32	54	4775	245	28990
b15	36	70	8893	449	55568

Finally, we analyze the data to classify the fault. Table 4.7 shows the compiled result of this experiment. Column 2 shows the number of faults after fault collapsing. Analyzing the result, we found that 30-40% of the faults are benign.

Our system assumes the SOC as testable cores having interconnected modules. Figure 4.20 (a) shows an example of a hierarchical SOC containing several cores having multiple modules in it. The dots show the faults in each module. Without loss of generality, we can assume a simple model to access the faults and carry out our fault analysis step: i.e. the multiplexing architecture [112]. With this architecture, one module can be accessed at a time, and all modules get access to full Test Access Mechanism (TAM) width. Figure 4.20 (c) shows a simplified SOC design. The objective here is to show the fault analysis step and not SOC test optimization. Any other test architecture design for SOCs will also work for

Table 4.7: *ITC'99 Benchmarks Result Analysis and Classification for Faults*

Circuit	# Faults	# Malignant	# Benign	HF_{po}	δ	50% HD
b04	1477	1005	472	24742	19655	68%
b10	468	373	95	8567	7689	64%
b11	1308	904	414	24876	21452	73%
b12	2777	2139	638	48356	37511	71%
b13	835	552	283	17568	12421	69%
b14	12643	7956	4687	93496	84387	73%
b15	23316	15387	7929	94621	85197	70%

the analysis step as well [113–115]. The fault analysis step carried out in test generation phase of the simplified IC fabrication step is shown in Figure 4.20 (b). We designed a simple SOC with selected modules from ISCAS-85 and ITC-99 benchmarks for our implementation [98, 108, 116]. Ten different modules have been chosen, including C1908, C3540, C5315, C6288, C7552 from ISCAS-85, and s298, s526, s641, s820, s1196 from ISCAS-89 benchmark circuits. We use Altera Max Plus II software to design the SOC. We inject the fault at the appropriate points and then start the fault analysis step, which requires a two-step method. In the first step, we chose the core, and in the second step, we performed fault analysis through fault simulation. Like other experiments, we consider a single stuck-at-fault model here. Two different C programs are developed, one that asks the user to choose a module. Once that is done, the program isolates that module and assigns the TAM input to the input line and output line to the TAM output line. Then the second program is invoked that applies the test patterns. We chose a random test set of size 10000 for ISCAS89 benchmark modules. In the presence of a injected fault, 10000 patterns get applied, and the method calculates the pay-off. Finally, we analyze the data to classify the fault. Table 4.8 shows the compiled result of this experiment.

The benefit of our approach mainly contributes to yield improvement like other threshold testing methods. We formulate a separate yield enhancement formula for our technique. A natural yield is defined as the number of good circuits produced from the total number of circuits tested (Equation 4.9).

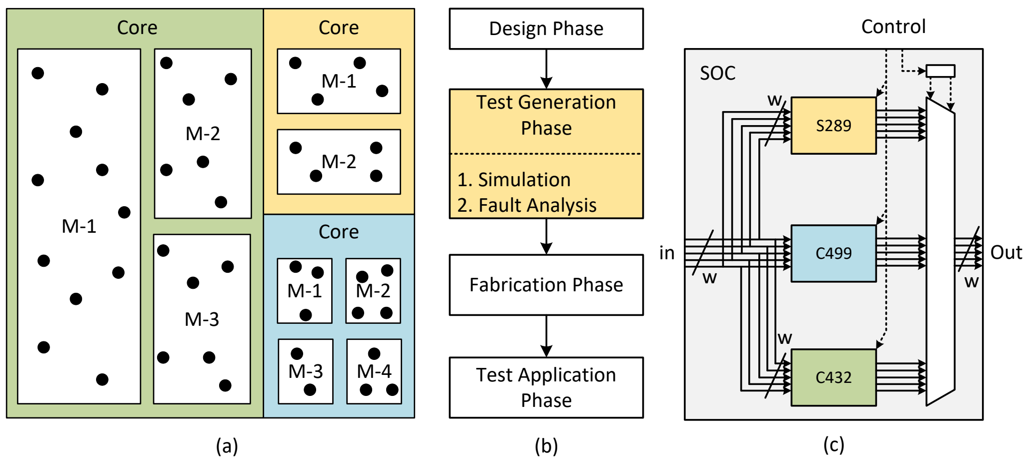


Figure 4.20: *(a) Hierarchical SOC Containing Several Cores Having Multiple Modules (b) Simplified IC Fabrication Steps (c) Simplified SOC Design*

Table 4.8: *SOC Test Result*

Circuits	**Total Fault**	**Malignant**	**Benign**	Hf_{po}	**delta**	**50% HD**
C1908	1879	1023	856	24128	21312	73%
C3540	3428	1952	1476	48436	42234	70%
C5315	5350	3335	2015	63298	57627	73%
C6288	7744	7056	688	15675	12598	68%
C7552	7550	4660	2890	32657	27189	74%
s298	308	235	73	6387	5275	59%
s526	555	346	209	8245	6978	62%
s641	467	283	184	8111	7003	75%
s820	850	572	278	10349	8566	70%
s1196	1242	859	383	14537	11655	71%

$$Natural_{Yield} = \frac{GoodCircuits}{TotalCircuitsTested} \tag{4.9}$$

But the natural yield does not contribute to the effective yield. Here effective yield refers to the number of good circuits plus the acceptable circuits. Therefore it is necessary to find an acceptable yield which is defined in Equation 4.10.

$$Acceptable_{Yield} = \frac{AcceptableCircuits}{FaultyCircuits} \tag{4.10}$$

Here, the faulty circuit refers to the fraction of circuits that fail during the conventional testing process. The acceptable circuit refers to the fraction of faulty circuits identified as acceptable during the approximate testing process. To find the effective yield, it is necessary to find the data about the probability of occurrence of a fault and the natural yield. In the following section, we will describe the proposed yield model.

4.3 Discussion and Summary

This section highlights the process of approximate testing by analyzing the techniques discussed in this chapter. Section 4.1 introduces the idea of approximate testing and proposes a method to identify the sites where, even if a fault exists does not affect the overall output. In section 4.2, we introduce a retesting technique to test the faulty circuits and find the usable circuits among them. The idea in section 4.1 is built around the comparison between the Circuit Under Test (CUT) and its corresponding approximate circuit. This technique can be helpful when the approximate version of the CUT is available. When approximate versions are not available, they can be obtained through the proposed pruning technique. A simple one-to-one mapping is made to identify the approximate site, and then the faults (present in the identified site) are ignored during testing. However, the idea in section 4.2 is based on fault classification using the overall fault effect analysis. To find the overall fault effect, we use the hamming distance and the number of test patterns that alters the output line in the presence of a fault. During the retesting process, some faults are not tested because their overall fault effect is below a decided threshold. Another benefit of this proposed method is yield enhancement. In Appendix A, we have explained how the proposed technique improves the yield compared to the traditional testing method.

4.4 Conclusion

This chapter proposes two different works under the heading "approximate testing." The first work in section proposes a technique to approximate the conventional test procedure with an objective to reduce the test data volume. The basic idea is to design an efficient mechanism to classify the fault into two groups: tolerable fault and intolerable fault. By doing this, we can generate a test pattern only for the intolerable fault that helps in reducing the amount of test data. In this work, we propose a fault-based test technique of fault classification based on the error-significance analysis. We use the conventional ATPG to generate the test pattern for the classified faults (intolerable faults only). We use the image compression experiment to examine our proposed technique's correctness and conclude that the output is quite adequate while considering human perception behavior on image clarity. The immeasurable part of our proposed method is that we can control the visual precision of an image. We have also applied the technique to reduce test pattern for different kinds of arithmetic circuits as well as ISCAS-85 benchmark circuits. Further, we have implemented our technique in designing an adder circuit. For experimental analysis, we have chosen selected benchmark circuits from ISCAS-85 to fit into our application domain. As the proposed technique is based on error-significance classification, we could not include circuits that may produce bit-flip errors.

While the first method proposed section 4.1 uses error significance as a quantification measure, the second technique in section 4.2 uses bit-flip error to classify the faults. Due to the use of bit-flip error, the technique is also applicable to test non-arithmetic circuits. In section 4.2, we presented a retesting technique to identify accepted circuits among the rejected ones. This accepted circuit helps in increasing the yield performance of the manufacturing process. Further, we have extended the idea which introduces the notion of acceptable circuits that are quite different from the approximate circuits. Identifying these acceptable circuits requires a different testing method which is the main contribution of this work. This testing method is known as the approximate testing method. Section 4.2 shows a complete architecture of approximate test flow by incorporating a fault analysis technique into the design phase. The entire process flow follows a two-phase execution; the first phase shows the conventional test flow, and the second phase shows the approximate test flow. Finally, the proposed method is evaluated using the ISCAS 85 and ITC'99 benchmark circuit, and the result is encouraging. We have also constructed an SOC with selected benchmarks to test the efficiency and scalability of our system and found that, on average, 30-40% of the faults are identified as acceptable. Yield gain is a significant benefit of our technique. The proposed yield model (refer Appendix A) shows that, on average, 10-20% of circuits are found acceptable when natural yield gain is between 35-65%.

Chapter 5

Conclusion and Future Works

We examine the concepts and strategies provided throughout this thesis in this chapter and sketch out potential directions.

5.1 Summary of the Contributions

In the world of information technology, the introduction of the approximate computing paradigm opened up a slew of new possibilities. The primary purpose of approximate computing is to increase system efficiency (time, area, and energy) by lowering the accuracy requirements for results. Approximate computing has been used at various levels of computing systems, ranging from hardware to software to architectures. Approximate computing has also been used to construct a new class of integrated circuits, known as approximate integrated circuits or AxICs, in all of the work done over the last two decades. With the advent of a new class of circuits, new challenges and opportunities in chip design, test, and verification arose.

Several research papers in the approximate computing paradigm have been proposed at various levels of abstraction throughout the last decade. Meanwhile, adders have gotten a lot of attention for approximation at the hardware level of abstraction. Adders are believed to be the most basic and extensively used arithmetic operator. The approximation adders are designed using one of three approaches: (1) AxBA, (2) AxSA, or (3) AxPA. The relevance, motivation, design philosophy, design strategy, and background of approximation adders were covered in this thesis. Beginners can use this as a starting point to learn how to use approximate adders in a methodical manner. We also looked at prior adder strategies to see how good they were in producing effective approximate adders. The following is a summary of the proposed adder circuit.

Systematic Design of Approximate Adder using SGLP: We proposed a technique to design an approximate adder using pruning the insignificant gates from the existing adder circuit. Pruning decision is made using the calculation of significance for each gate. The gate that has the lowest significance is pruned first. Significance is a numeric quantity assigned to each gate starting from the gates contributing to the output. Removing gates from the circuit affects the accuracy of the result. However, approximate design philosophy allows a certain amount of error in the output, and the amount is guarded by a terminology called error threshold. The designer decides the error threshold (which is error-significance in our case) before the actual design. During the pruning process, after removing one gate, the error significance is calculated, and if it is found to be below the threshold, the method continues to prune the gates.

The technique discussed above is applicable to redesign simple adder architecture like RCA or complex designs like KSA. The fundamental blocks, i.e., an FA, are passed through the SGLP technique and produce an approximate version of it for RCA. The approximate version of FA is known as AFA. To build an n-bit adder, the most significant bits are replaced with these AFAs. The proposed approximate adder design is tested for its efficiency and effectiveness. We have chosen the image compression application to evaluate its performance by selecting 45 grayscale images. After compression, the image degradation is analyzed using PSNR, and we found that image clarity is not degraded much as compared to the state-of-the-art. We have also checked the visual quality of some images and found that the image clarity is totally fine considering the human perception.

Circuit testing is another issue that has arisen due to the introduction of approximation terminology. To achieve satisfactory results, approximate chip designers meticulously alter the circuit structure to introduce acceptable flaws. Designers use error metrics to accurately define the acceptable notion. Then, to fix the maximum permissible (i.e. acceptable) error, they define error thresholds. As a result, the concept of defective circuits is altered. Indeed, it introduces two new types of faults: benign faults (faults that cause acceptable errors) and malignant faults (i.e., faults causing catastrophic errors). The class of a detectable defect can be identified in the testing context by measuring the caused error at the defective circuit's output. The circuit must be rejected if the measured error is more than the permitted level. However, if the measured error remains below the allowed threshold, the defective circuit does not need to be rejected. This raises to introduce a new type of circuit called Acceptable IC (AcIC). The role of a tester is affected due to the notion of circuit acceptability. Rejected circuits that are identified during the conventional test are still acceptable. Rejected circuits whose observed error is less than the tolerable threshold must be accepted and contributes to the yield. These circuits are termed as AcICs. Identifying AcICs from the rejected one requires approximately modifying the conventional test flow architecture. Instead of generating a test vector for all selected faults, we generate tests using approximate testing for a subset of faults. The following is the summary of approximating test flow architecture.

Approximate Testing of Digital VLSI Circuits using Error Significance based Fault Analysis: We know that a traditional test approach aims for 100 percent fault coverage by creating test patterns for all faults discovered during the fault analysis phase. A large number of test patterns are generated in order to maintain good test quality, which raises the Test Data Volume (TDV). Applying such a large number of test patterns takes longer and consumes more power. As a result, various studies in the literature have focused on using compression techniques to reduce the size of test data. We introduce the term "approximate testing" in this work, which softens the criterion of 100% fault coverage. The primary aim is to find flaws that have a catastrophic effect and create test patterns specifically for them. The remaining defects are not tested, and there aren't any test patterns generated.

We perform an image compression experiment to test the validity of our suggested technique, and we find that the result is sufficiently appropriate when human perception behavior on image clarity is taken into account. We have also used the technique to minimize the number of test patterns for various types of tests ISCAS-85 benchmark circuits and arithmetic circuits. We used our technique to design an adder circuit, and ISCAS-85 benchmark circuits were chosen to meet our application domain. We could not add circuits that could cause bit-flip errors because the proposed technique is based on error-significance classification.

Retesting of Rejected Circuits using Approximation Technique: Circuits that yield acceptable results can be employed in error-resilient applications like Recognition, Mining, and Synthesis (RMS). To put it another way, an error-prone circuit can be used in error-tolerant applications despite the fact that it has a flaw. Acceptable Circuits are the circuits that meet these criteria (AcICs). We found no technique for identifying AcICs through testing when using the usual testing process. We must not forget that a defective circuit detected through traditional testing may produce error-free output for the greater part of test patterns. The basic concept is to divide testing into two sections. Using a standard test flow architecture, we gather the rejected circuits in the first step. The rejected circuits that were found to be defective in the first phase are retested utilising the test patterns in the second stage. At this point, the circuit may produce inaccurate results for certain test patterns, but we should ignore this and keep testing until all of the test patterns have been applied. The test patterns for which the circuit responds incorrectly (error) are measured and compared to the golden output for deviation. If the amount of variation is negligible and has no effect on the circuit's overall performance, the circuit is acceptable.

Retesting Defective Circuits to Allow Acceptable Faults for Yield Enhancement In this contribution, we extend the idea of employing fault classification to distinguish AcICs from rejected ones. The Hamming distance is used to classify the defect. First, we identify the issue, which, if present, will have catastrophic outcomes for the majority of input patterns. To determine this location, we calculate a Fault Pay-off for each fault that could be present in the circuit. According to our observations, the location with the highest fault Pay-off will most likely influence more than half of the output lines if it includes a fault (Hamming Distance is more than 50 percent). As a result, during the retesting phase, we retest the defective circuits by building test patterns for the faults at the place with the lowest Fault Pay-off. We tested the proposed technique on the ISCAS 85, ISCAS 89, and ITC'99 benchmark circuits to evaluate how effective it is. We also used a SOC with predetermined benchmarks to evaluate the efficiency of our method on large circuits, and found that on average, 30-40% of errors are categorised as acceptable. The proposed method has the significant advantage of boosting yield. The effective yield gain, according to the proposed yield model, will be between 10% and 20% on average.

5.2 Fulfilling the Aim and Objective

With the above contributions, we achieve our objectives discussed in section 1.3. The first objective of the thesis is to develop a generalized approach to designing an approximate adder from an existing adder. The SGLP method provides a perfect solution to achieve this objective. One can apply the pruning process on any kind of adder to obtain its approximate version. The work has explained how the approximate adders generated through the SGLP technique produce good enough results in image processing applications. The other contributions described above help achieve the objective of approximating the test flow architecture. The fault analysis technique helps in identifying the critical faults, generates and applies test patterns to test them, and finally concludes that testing a circuit approximately is possible. The remaining contributions propose an approximate technique to retest all faulty circuits and found that 20-30% of them are usable and contribute to enhancing the yield.

5.3 Future Works

While tremendous progress has been made in the field of approximation computing, there is still much to be discovered and many problems to overcome. While considering the progress in approximate adder design and approximate testing, the following design challenges need to be addressed in the near future.

- As discussed, the SGLP technique is used to design an AFA and later used in the LSB part of RCA to generate approximate adders. It also proposed an idea of generating approximate adders from complex designs like KSA using the SGLP method. However, SGLP may not be a good candidate for generating approximate design for non-arithmetic circuits. Because identifying the LSB bits on a non-arithmetic circuit is difficult. Therefore, other quantification measures such as Error-rate may work better. Thus, in the future, we may propose Error Rate based Gate Level Pruning (RGLP) to generate approximate designs for non-arithmetic circuits. Exploring other quantification measures to prune the gates from the original non-arithmetic designs is also possible.

- Considering approximate testing, the technique proposed in this thesis mainly focuses on retesting FaICs to get AcICs. A similar challenge is also posed by approximate circuit design and manufacturing processes. The test engineers must be careful while testing an AxIC because distinguishing actual defects (either caused during design or manufacturing) from what is being approximated becomes more challenging, as design/manufactured defects may result in very similar variations in results. Therefore, a different test procedure is needed to test AxICs. The technique proposed in this thesis for retesting is also appropriate for testing AxICs, but we have not yet explored it. In the future, we may use the proposed approximate testing technique to test the AxICs.

In summary, we can say that the approximate computing techniques are still exploring and require further innovation in broad areas such as designing memories and I/O subsystems, designing approximate processors and accelerators, etc.

References

[1] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual international symposium on computer architecture (ISCA).* IEEE, 2011, pp. 365–376.

[2] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Tsp: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, 2014, pp. 1–10.

[3] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura, "Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping," in *2013 IEEE 31st International Conference on computer design (ICCD).* IEEE, 2013, pp. 349–356.

[4] K. Ma and X. Wang, "Pgcapping: Exploiting power gating for power capping and core lifetime balancing in cmps," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT).* IEEE, 2012, pp. 13–22.

[5] L. Wang and K. Skadron, "Implications of the power wall: Dim cores and reconfigurable logic," *IEEE Micro*, vol. 33, no. 5, pp. 40–48, 2013.

[6] N. Kapadia and S. Pasricha, "Varsha: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *2015*

Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2015, pp. 1060–1065.

[7] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC).* IEEE, 2015, pp. 1–6.

[8] 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.

[9] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[10] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in *2010 International SoC Design Conference.* IEEE, 2010, pp. 323–327.

[11] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2012, pp. 1257–1262.

[12] H. A. Almurib, T. N. Kumar, and F. Lombardi, "Inexact designs for approximate low power addition by cell replacement," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2016, pp. 660–665.

[13] 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.

[14] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power-and area-efficient approximate wallace tree multiplier for error-resilient systems," in *Fifteenth International Symposium on Quality Electronic Design.* IEEE, 2014, pp. 263–269.

[15] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE, 2015, pp. 418–425.

[16] ——, "A low-power dynamic divider for approximate applications," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC).* IEEE, 2016, pp. 1–6.

[17] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram, "Seerad: A high speed yet energy-efficient rounding-based approximate divider," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2016, pp. 1481–1484.

[18] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, "Better-than-voltage scaling energy reduction in approximate srams via bit dropping and bit reuse," in *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS).* IEEE, 2015, pp. 132–139.

[19] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob, "Flexible auto-refresh: Enabling scalable and energy-efficient dram refresh reductions," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 235–246.

[20] C. Kushwah and S. K. Vishvakarma, "A sub-threshold eight transistor (8t) sram cell design for stability improvement," in *2014 IEEE International Conference on IC Design & Technology.* IEEE, 2014, pp. 1–4.

[21] S. Amanollahi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Circuit-level techniques for logic and memory blocks in approximate computing systemsx," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2150–2177, 2020.

[22] Y. Fang, H. Li, and X. Li, "Softpcm: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write," in *2012 IEEE 21st Asian Test Symposium.* IEEE, 2012, pp. 131–136.

[23] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE, 2012, pp. 449–460.

[24] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 554–566.

[25] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* IEEE, 2013, pp. 1–12.

[26] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 124–134.

[27] M. Rodriguez-Cancio, B. Combemale, and B. Baudry, "Approximate loop unrolling," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 94–105.

[28] S. Yesil, I. Akturk, and U. R. Karpuzcu, "Toward dynamic precision scaling," *IEEE Computer Architecture Letters*, vol. 38, no. 04, pp. 30–39, 2018.

[29] M. A. Anam, P. N. Whatmough, and Y. Andreopoulos, "Precision–energy–throughput scaling of generic matrix multiplication and convolution kernels via linear projections," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 11, pp. 1860–1873, 2014.

[30] A. Rahimi, L. Benini, and R. K. Gupta, "Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 847–851, 2013.

[31] Z. Liu, A. Yazdanbakhsh, D. K. Wang, H. Esmaeilzadeh, and N. S. Kim, “Axmemo: hardware-compiler co-design for approximate code memoization,” in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 685–697.

[32] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, “Enerj: Approximate data types for safe and general low-power computation,” *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[33] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Architecture support for disciplined approximate programming,” in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 301–312.

[34] M. Vilim, H. Duwe, and R. Kumar, “Approximate bitcoin mining,” in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[35] Y. Wang, Q. Xu, G. Qu, and J. Dong, “Information hiding behind approximate computation,” in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 405–410.

[36] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, and G. Qu, “Approximate computing for low power and security in the internet of things,” *Computer*, vol. 50, no. 6, pp. 27–34, 2017.

[37] S. Bian, M. Hiromoto, and T. Sato, “Dwe: Decrypting learning with errors with errors,” in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[38] D. E. S. Kundi, S. Bian, A. Khalid, C. Wang, M. O’Neill, and W. Liu, “Axmm: Area and power efficient approximate modular multiplier for r-lwe cryptosystem,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[39] S. Venkataramani, X. Sun, N. Wang, C.-Y. Chen, J. Choi, M. Kang, A. Agarwal, J. Oh, S. Jain, T. Babinsky *et al.*, "Efficient ai system design with cross-layer approximate computing," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2232–2250, 2020.

[40] I. Chakraborty, M. Ali, A. Ankit, S. Jain, S. Roy, S. Sridharan, A. Agrawal, A. Raghunathan, and K. Roy, "Resistive crossbars as approximate hardware building blocks for machine learning: Opportunities and challenges," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2276–2310, 2020.

[41] W. Liu, C. Gu, M. O'Neill, G. Qu, P. Montuschi, and F. Lombardi, "Security in approximate computing and approximate computing for security: Challenges and opportunities," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2214–2231, 2020.

[42] X. Jiao, V. Akhlaghi, Y. Jiang, and R. K. Gupta, "Energy-efficient neural networks using approximate computation reuse," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2018, pp. 1223–1228.

[43] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

[44] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "A survey of testing techniques for approximate integrated circuits," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2178–2194, 2020.

[45] I. Polian, "Test and reliability challenges for approximate circuitry," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 26–29, 2017.

[46] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "A test pattern generation technique for approximate circuits based on an ilp-formulated pattern selection procedure," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 849–857, 2019.

[47] M. Brandalero, A. C. S. Beck, L. Carro, and M. Shafique, "Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[48] M. D. Ercegovac and T. Lang, *Digital arithmetic*. Elsevier, 2004.

[49] 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.

[50] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*. IEEE, 2009, pp. 69–72.

[51] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 130–137.

[52] C. Lin, Y.-M. Yang, and C.-C. Lin, "High-performance low-power carry speculative addition with variable latency," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1591–1603, 2014.

[53] D. Shin and S. K. Gupta, "A re-design technique for datapath modules in error tolerant applications," in *2008 17th Asian test symposium*. IEEE, 2008, pp. 431–437.

[54] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, 2009.

[55] 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.

[56] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, “Design and applications of approximate circuits by gate-level pruning,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.

[57] J. Schlachter, V. Camus, C. Enz, and K. V. Palem, “Automatic generation of inexact digital circuits by gate-level pruning,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 173–176.

[58] B. Garg and G. Sharma, “Acm: An energy-efficient accuracy configurable multiplier for error-resilient applications,” *Journal of Electronic Testing*, vol. 33, no. 4, pp. 479–489, 2017.

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

[60] I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, “Towards approximation during test of integrated circuits,” in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2017, pp. 28–33.

[61] S. K. Jena, “Fault classification based approximate testing of digital vlsi circuit,” in *Electronic Systems and Intelligent Computing*. Springer, 2020, pp. 641–651.

[62] S. Singh, V. Mishra, S. Satapathy, D. Pandey, K. Goswami, D. S. Banerjee, and B. Jajodia, “Efcsa: An efficient carry speculative approximate adder with rectification,” in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*, 2022, pp. 1–7.

[63] A. S. Roy, R. Biswas, and A. S. Dhar, “On fast and exact computation of error metrics in approximate lsb adders,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 876–889, 2020.

[64] W. Ahmad, B. Ayrancioglu, and I. Hamzaoglu, “Low error efficient approximate adders for fpgas,” *IEEE Access*, vol. 9, pp. 117 232–117 243, 2021.

[65] J. Lee, H. Seo, H. Seok, and Y. Kim, "A novel approximate adder design using error reduced carry prediction and constant truncation," *IEEE Access*, vol. 9, pp. 119 939–119 953, 2021.

[66] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, *Digital integrated circuits: a design perspective.* Pearson education Upper Saddle River, NJ, 2003, vol. 7.

[67] H. Li, X. Fan, Q. Li, and H. Liu, "An efficient light-weight configurable approximate adder design," in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2021, pp. 1–6.

[68] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," in *2011 Design, Automation & Test in Europe.* IEEE, 2011, pp. 1–6.

[69] N. K. Jha and S. Gupta, *Testing of digital systems.* Cambridge University Press, 2003.

[70] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI test principles and architectures: design for testability.* Elsevier, 2006.

[71] Z. Jiang and S. K. Gupta, "An atpg for threshold testing: Obtaining acceptable yield in future processes," in *Proceedings. International Test Conference.* IEEE, 2002, pp. 824–833.

[72] M. A. Breuer, "Intelligible test techniques to support error-tolerance," in *13th Asian test symposium.* IEEE, 2004, pp. 386–393.

[73] L. Anghel, M. Benabdenbi, A. Bosio, M. Traiola, and E. I. Vatajelu, "Test and reliability in approximate computing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, 2018.

[74] R. I. Bahar, U. Karpuzcu, and S. Misailovic, "Special session: Does approximation make testing harder (or easier)?" in *2019 IEEE 37th VLSI Test Symposium (VTS).* IEEE, 2019, pp. 1–9.

[75] T.-Y. Hsieh, T.-A. Cheng, and C.-R. Chen, “Approximate functional testing for image applications based on error-tolerance,” in *2017 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*. IEEE, 2017, pp. 203–204.

[76] K.-J. Lee, T.-Y. Hsieh, and M. A. Breuer, “A novel test methodology based on error-rate to support error-tolerance,” in *IEEE International Conference on Test, 2005*. IEEE, 2005, pp. 9–pp.

[77] T.-Y. Hsieh, K.-J. Lee, and M. A. Breuer, “An error-oriented test methodology to improve yield with error-tolerance,” in *24th IEEE VLSI Test Symposium*. IEEE, 2006, pp. 6–pp.

[78] S. Shahidi and S. Gupta, “Multi-vector tests: A path to perfect error-rate testing,” in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1178–1183.

[79] Z. Jiang and S. K. Gupta, “Threshold testing: improving yield for nanoscale vlsi,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1883–1895, 2009.

[80] ——, “Threshold testing: Covering bridging and other realistic faults,” in *14th Asian Test Symposium (ATS’05)*. IEEE, 2005, pp. 390–397.

[81] H. Ichihara, K. Sutoh, Y. Yoshikawa, and T. Inoue, “A practical approach to threshold test generation for error tolerant circuits,” in *2009 Asian Test Symposium*. IEEE, 2009, pp. 171–176.

[82] Z. Pan and M. A. Breuer, “Basing acceptable error-tolerant performance on significance-based error-rate (sber),” in *26th IEEE VLSI Test Symposium (vts 2008)*. IEEE, 2008, pp. 59–66.

[83] 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.

[84] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, “On the comparison of different atpg approaches for approximate integrated circuits,” in *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2018, pp. 85–90.

[85] M. Masadeh, O. Hasan, and S. Tahar, “Approximation-conscious ic testing,” in *2018 30th International Conference on Microelectronics (ICM)*. IEEE, 2018, pp. 56–59.

[86] 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.

[87] G. Karakonstantis, N. Banerjee, and K. Roy, “Process-variation resilient and voltage-scalable dct architecture for robust low-power computing,” *IEEE transactions on very large scale integration (VLSI) systems*, vol. 18, no. 10, pp. 1461–1470, 2009.

[88] R. J. Cintra and F. M. Bayer, “A dct approximation for image compression,” *IEEE Signal Processing Letters*, vol. 18, no. 10, pp. 579–582, 2011.

[89] S. Bouguezel, M. O. Ahmad, and M. Swamy, “A low-complexity parametric transform for image compression,” in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 2145–2148.

[90] T. I. Haweel, “A new square wave transform based on the dct,” *Signal processing*, vol. 81, no. 11, pp. 2309–2319, 2001.

[91] K. Lengwehasatit and A. Ortega, “Scalable variable complexity approximate forward dct,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 11, pp. 1236–1248, 2004.

[92] S. Bouguezel, M. O. Ahmad, and M. Swamy, “Low-complexity $8\times$ 8 transform for image compression,” *Electronics Letters*, vol. 44, no. 21, pp. 1249–1250, 2008.

[93] ——, “A fast 8× 8 transform for image compression,” in *2009 International Conference on Microelectronics-ICM*. IEEE, 2009, pp. 74–77.

[94] A. G. Weber, “The usc-sipi image database version 5,” *USC-SIPI Report*, vol. 315, no. 1, 1997.

[95] G. K. Wallace, “The jpeg still picture compression standard,” *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[96] I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, “Can we approximate the test of integrated circuits?” in *WAPCO: Workshop On Approximate Computing*, 2017.

[97] “Arithmetic module generator,” https://www.ecsis.riec.tohoku.ac.jp/topics/amg/, accessed: 2020-04-21.

[98] M. C. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the iscas-85 benchmarks: A case study in reverse engineering,” *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[99] “The usc-sipi image database, university of southern california, signal and image processing institute,” https://sipi.usc.edu/database/, accessed: 2020-04-21.

[100] T.-Y. Hsieh, P.-X. Wu, and C.-C. Cheng, “On classification of acceptable images for reliable artificial intelligence systems: A case study on pedestrian detection,” in *2020 IEEE 38th VLSI Test Symposium (VTS)*. IEEE, 2020, pp. 1–6.

[101] T.-Y. Hsieh, K.-J. Lee, and M. A. Breuer, “Reduction of detected acceptable faults for yield improvement via error-tolerance,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[102] ——, “An error rate based test methodology to support error-tolerance,” *IEEE Transactions on Reliability*, vol. 57, no. 1, pp. 204–214, 2008.

[103] ——, “An error-tolerance-based test methodology to support product grading for yield enhancement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 6, pp. 930–934, 2011.

[104] T.-Y. Hsieh, Y.-H. Peng, and K.-H. Li, “Efficient error-tolerability testing on image processing circuits based on equivalent error rate transformation,” *Journal of Electronic Testing*, vol. 30, no. 6, pp. 687–699, 2014.

[105] S. Shahidi and S. K. Gupta, “Estimating error rate during self-test via one's counting,” in *2006 IEEE International Test Conference*. IEEE, 2006, pp. 1–9.

[106] S. Sindia and V. D. Agrawal, “Tailoring tests for functional binning of integrated circuits,” in *2012 IEEE 21st Asian Test Symposium*. IEEE, 2012, pp. 95–100.

[107] S. K. Jena, S. Biswas, and J. K. Deka, “Approximate testing of digital vlsi circuits using error significance based fault analysis,” in *2020 24th International Symposium on VLSI Design and Test (VDAT)*. IEEE, 2020, pp. 1–6.

[108] “The description and verilog code for all iscas-85 benchmark circuits,” http://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html, accessed: 2021-06-15.

[109] H. K. Lee and D. S. Ha, “Hope: An efficient parallel fault simulator for synchronous sequential circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.

[110] F. Corno, M. S. Reorda, and G. Squillero, “Rt-level itc'99 benchmarks and first atpg results,” *IEEE Design & Test of computers*, vol. 17, no. 3, pp. 44–53, 2000.

[111] “Politecnico di torino itc'99 benchkmarks,” https://github.com/squillero/itc99-poli, accessed: 2021-06-16.

[112] S. K. Goel and E. J. Marinissen, "Effective and efficient test architecture design for socs," in *Proceedings. International Test Conference.* IEEE, 2002, pp. 529–538.

[113] K. Chakrabarty, V. Iyengar, and M. D. Krasniewski, "Test planning for modular testing of hierarchical socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 435–448, 2005.

[114] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, S. M. Reddy, and K. Chakrabarty, "On concurrent test of core-based soc design," in *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation.* Springer, 2002, pp. 37–50.

[115] K. Chakrabarty, E. Marinissen *et al.*, "Test access mechanism optimization, test scheduling, and tester data volume reduction for system-on-chip," *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1619–1632, 2003.

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

[117] S. K. Tewksbury, *Wafer-level integrated systems: implementation issues.* Springer Science & Business Media, 2012, vol. 70.

[118] B. T. Murphy, "Cost-size optima of monolithic integrated circuits," *Proceedings of the IEEE*, vol. 52, no. 12, pp. 1537–1545, 1964.

[119] R. Seeds, "Yield and cost analysis of bipolar lsi," *IEEE Transactions on Electron Devices*, vol. 15, no. 6, pp. 409–409, 1968.

Appendix: A

Yield Modeling and Analysis

In a traditional test paradigm, all testable defects should be discovered, and any chip having a fault should be isolated. Under the approximate testing paradigm, acceptable faults are found that can only generate errors of magnitudes more diminutive than the set threshold. Consider the faults in the circuit shown in Fig. 4.15. In traditional testing, each of these faults are identified as testable. During approximate testing of C17 circuit, 15 faults are determined as acceptable. This means that via approximate testing, some faults are identified as acceptable and lead to the tested circuit being acceptable. During the discussion of fault model, we will use the word chip and circuit interchangeably.

Several yield models proposed in the literature are based on a common theme: "as the chip area increases, the yield essentially drops." Some of the traditional yield models extracted from [117] are listed in Table 1. These equations work only when perfect chips (i.e., chips with no defects) contribute to the yield. However, in threshold testing, chips having acceptable faults also contribute to the yield. Therefore, a different yield model is necessary. Table 1 shows some threshold testing yield models.

In our proposed method of approximate testing, any chip with a single fault that belongs to the benign set is also considered while computing effective yield. We follow a similar yield model like [79]. Assume that the mean number of faults per chip is λ. According to Poisson distribution, the probability that a die has k faults is discussed in [117] as:

Table 1: *Existing Yield Models*

Traditional Yield Models	
Model Name	**Equation**
Poisson Model	$Y_1 = e^{-D_0 A}$
Binomial Model	$Y_2 = \left[1 - \frac{A}{A_w}\right]^{D_0 A_w}$
Murphy's Model	$Y_3 = \left[\frac{1-e^{-D_0 A}}{D_0 A}\right]^2$
Seeds' Model	$Y_4 = \frac{1}{1+D_0 A}$
Threshold Testing Yield Models	
Model Name	**Equation**
Hsieh et al [77]	$EY = P_0 + \frac{AC_1}{C_1^n} \times P_1 + \frac{AC_2}{C_2^n} \times P_2 + \frac{AC_3}{C_3^n} \times P_3 + ...$
Jiang et al [79]	$Y_N = \frac{1}{1+AD_0} + \frac{P_A D_0 A}{(1+AD_0)^2}$
Sindia et al [106]	$Y_N = \frac{1}{(1-p)^{N-b}}$

$$P_k = \frac{e^{-\lambda}\lambda^k}{k!} \tag{1}$$

To find the perfect chip, k must be 0. The expected chip yield is:

$$P_0 = e^{-\lambda} \tag{2}$$

According to [118], let a be the susceptible area and D be the fault density, then the probability that the device is good is:

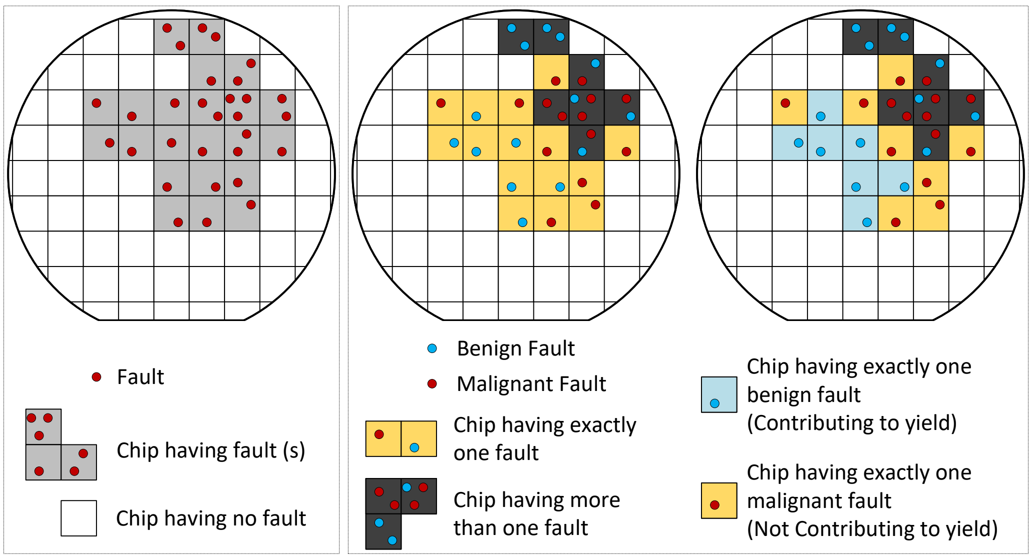
$$P_0 = e^{-Da} \tag{3}$$

As D varies from chip to chip and follows a nonuniform density distribution, the overall device yield is:

$$Y = \int_0^\infty P_0 f(D)dD = \int_0^\infty e^{-Da} f(D)dD \tag{4}$$

where, $f(D)$ denotes a nonuniform distribution function and the exact form is unknown. Substituting the exponential distribution function $f(D) = \frac{e^{-D/D_0}}{D_0}$ in Equation 4, we get the yield as shown in Equation 5 and is known as Seeds' model for die yield [119].

$$Y = \frac{1}{1 + aD_0} \tag{5}$$


Fault
Chip having fault (s)
Chip having no fault
Benign Fault
Malignant Fault
Chip having exactly one fault
Chip having more than one fault
Chip having exactly one benign fault (Contributing to yield)
Chip having exactly one malignant fault (Not Contributing to yield)

(a) Classical Testing Yield Model (b) Approximate Testing Yield Model

Figure 1: *Fault Distribution on a Wafer in Classical Yield Model and Approximate Yield Model*

where D_0 represents the defect density. The above yield model is known as the classical yield model or natural yield model, where a perfect circuit (chip having no fault) contributes to the yield. Figure 1 (a) shows a visual representation of faults on a wafer. The white square boxes and the gray boxes represent perfect chips and faulty chips, respectively. The red dot on a chip refers to a fault. In our proposed method of approximate testing, two types of fault can occur on a chip. The red dot refers to a malignant fault, and the blue dot refers to a benign fault. The situation is shown in Figure 1 (b). According to the proposed technique, a chip with a single blue dot also contributes to the yield. Hence, first, we need to identify the circuits having one fault. Substituting k as 1 in Equation 1 we get:

$$P_1 = e^{-\lambda}\lambda = e^{-Da}Da \tag{6}$$

Finally, the modified yield formula for the proposed approximate testing is:

$$Y_{mod} = \int_0^\infty (P_0 + bP_1) f(D)dD = \int_0^\infty \left(e^{-Da} + be^{-Da}Da\right) f(D)dD \tag{7}$$

where, b refers to the percentage of acceptable faults. Now, substituting the exponential distribution function $f(D) = \frac{e^{-D/D_0}}{D_0}$ in Equation 7, we get the yield as shown in Equation 8.

$$Y_{mod} = \frac{1}{1 + aD_0} + \frac{bD_0a}{(1 + aD_0)^2} \tag{8}$$

The fractional increase in the effective yield can be found using the Equation 9.

$$E_{yield} = \frac{Y_{mod} - Y}{Y} \tag{9}$$

Analysis of the Yield Model

To analyze our proposed yield model, we have considered the entire range of classical yield from 0% to 100%. For the given classical yield, we calculate D_0 using Equation 5, where $D_0 = \frac{1-Y}{Ya}$. As per the proposed technique 30-40% of the fault belongs to the benign set. Considering the value of b to be 30%, we can calculate the effective yield using Equation 9, where $E_{yield} = \frac{bD_0a}{1+aD_0}$. Table 2 shows the E_{yield} for different values of Y when $b = 30\%$. We found some interesting observations from the result. When the natural yield (Y) decreases, the effective yield increases and vice versa. The area of a chip takes part in the result calculation but does not alter the effective yield value. The result of effective yield depends on the value of natural yield and the percentage of benign fault (b). The last row of Table 2 shows that when the natural yield is 100%, no fault exists in the entire wafer, and the effective yield is 0. Finally, we have plotted the result of effective yield for different values of b in Figure 2. The plot shows the result for three values of b, viz 10%, 20%, and 30%. We observe that the effective yield decreases when the value of b goes down and vice versa.

Table 2: *Effective Yield for 30% Benign Fault*

$Area(mm^2)$	Y	D_0	aD_0	b	$Y_{mod} - Y$	E_{yield}
514	0.05	0.036965	19	0.3	1.425	28.5
514	0.1	0.01751	9	0.3	2.7	27
514	0.15	0.011025	5.666667	0.3	3.825	25.5
514	0.2	0.007782	4	0.3	4.8	24
514	0.25	0.005837	3	0.3	5.625	22.5
514	0.3	0.00454	2.333333	0.3	6.3	21
514	0.35	0.003613	1.857143	0.3	6.825	19.5
514	0.4	0.002918	1.5	0.3	7.2	18
514	0.45	0.002378	1.222222	0.3	7.425	16.5
514	0.5	0.001946	1	0.3	7.5	15
514	0.55	0.001592	0.818182	0.3	7.425	13.5
514	0.6	0.001297	0.666667	0.3	7.2	12
514	0.65	0.001048	0.538462	0.3	6.825	10.5
514	0.7	0.000834	0.428571	0.3	6.3	9
514	0.75	0.000649	0.333333	0.3	5.625	7.5
514	0.8	0.000486	0.25	0.3	4.8	6
514	0.85	0.000343	0.176471	0.3	3.825	4.5
514	0.9	0.000216	0.111111	0.3	2.7	3
514	0.95	0.000102	0.052632	0.3	1.425	1.5
514	1	0	0	0.3	0	0

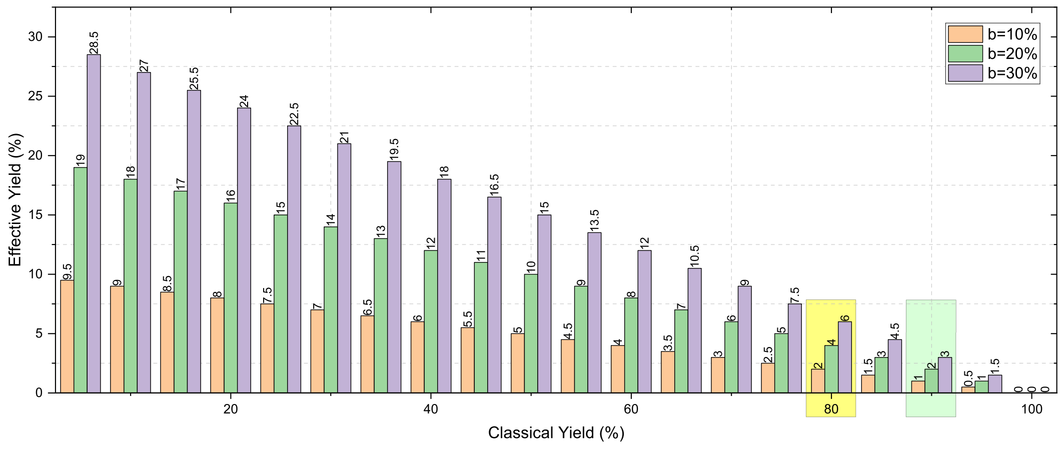


Figure 2: *Result of Effective Yield for different values of b*

Appendix: B

Journal Publications:

- **Sisir Kumar Jena**, Santosh Biswas, and Jatindra Kumar Deka, ***"Retesting Defective Circuits to Allow Acceptable Faults for Yield Enhancement".*** Journal of Electronic Testing, Volume-37, (December 2021), 20 pages. DOI: https://doi.org/10.1007/s10836-021-05980-y

Conference Publications:

- **Sisir Kumar Jena**, Santosh Biswas and Jatindra Kumar Deka **"Maximizing Yield through Retesting of Rejected Circuits using Approximation Technique**," 2020 IEEE REGION 10 CONFERENCE (TENCON), 2020, pp. 182-187, doi: 10.1109/TENCON50793.2020.9293891.

- **Sisir Kumar Jena**, Santosh Biswas and Jatindra Kumar Deka, ***"Approximate Testing of Digital VLSI Circuits using Error Significance based Fault Analysis***," 2020 24th International Symposium on VLSI Design and Test (VDAT), 2020, pp. 1-6, doi: 10.1109/VDAT50263.2020.9190571.

- **Sisir Kumar Jena**, Santosh Biswas, and Jatindra Kumar Deka. ***"Systematic design of approximate adder using significance based gate-level pruning (SGLP) for image processing application***." International Conference on Pattern Recognition and Machine Intelligence. Springer, Cham, 2019.