

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Jerome Angel John Rozario 184634IASM

HYBRID BIST BASED ON COMBINING OF PSEUDO-RANDOM AND DETERMINISTIC PATTERNS

Master's thesis

Supervisor: Prof. Raimund-Johannes Ubar

D.Sc. Institute of Computer Engineering, Tallinn University
of Technology

Professor, Chair of Computer systems Test and Verification

Tallinn 2018

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature, and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Jerome Angel John Rozario

13.05.2020

Abstract

A new method to design for testability and evaluation of digital circuits using hybrid Built-In-Self-Test (BIST) is proposed. It focuses on developing, optimizing, and evaluating the hybrid BIST solutions for the ISCAS'85 benchmark circuits. Although many approaches were proposed for the optimization of the hybrid BIST, however, they resulted in a reduced BIST cost but not with the reduced BIST design time cost. The proposed new method focuses on both the minimum BIST cost and the reduced BIST design time, targeting also the high scalability of the optimization method. The experimental results demonstrated the advantage of the proposed novel algorithm based approach compared with the known labor-intensive method. The results of this research are submitted as a research paper to the 30th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIA): Elmet Orasson, Jerome Angel John Rozario, Margus Kruus, Raimund Ubar in the topic Interdisciplinary Research Lab for Project-Based Learning of Hardware and Software Design for Computer Engineering Students.

Annotatsioon

Optimeeritud hübriidtestsüsteem digitaalskeemide isetestimiseks

Käesolevas uurimistöös on välja töötatud, implementeeritud ja katsetatud uus meetod digitaalskeemide isetestimiseks hübriidsetel testide genereerimise põhimõttel, kasutades nii juhuslikke kui ka deterministlikke stiimuleid testimiseks. Töö fookuseks on hübriidtest-süsteemi disainiprotsessi ajaline minimeerimine. Kui senised analoogsed uuringud on pühendatud üksnes testsüsteemi optimeeritud lahenduse leidmisele, siis käesoleva töö eesmärgiks on ka optimaalse lahendi leidmiseks vajaliku aja vähendamine. Eksperimentaalsed uuringud näitasid, et uudne optimeerimisalgoritm on hästi skaleeruv ja tagab ligilähedaselt samad optimeerimistulemused, mis senised meetodid, mis aga on palju töömahukamad. Käesoleva töö tulemused on vormistatud ka publikatsioonina, mis on esitatud konverentsile „30th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE)“ artiklina Elmet Orasson, Jerome Angel John Rozario, Margus Kruus, Raimund Ubar. “Interdisciplinary Research Lab for Project-Based Learning of Hardware and Software Design for Computer Engineering Students”.

Acknowledgments

Firstly, I would like to thank my almighty God for His unconditional love, grace, and guidance, which encouraged me to complete this thesis.

Secondly, I would like to thank my supervisor, Prof. Raimund-Johanned Ubar, for his full guidance and support, even in this difficult situation throughout this research. It was a great pleasure working with you. I would also like to thank Prof. Elmet Orasson for his guidance in practical work.

To my lovable mother, who had always stood by me, encouraged me and supported me throughout my studies. Everything I am now is because of her prayers. To my dad, who has been my support system always. Thanks, both of you.

I'm eternally grateful to my uncle Sekar, who always believed in me when no one else does. Thank you for your valuable advice and guidance that you are giving to me always.

A special thanks to my lovely sisters' Mercy, Marilyn, and Mira, who always had there for me and encouraged me whenever I feel down. I must say a very big thank you to Shanthi chitha, Rani chitha, Sheela chitha, and to my entire family and friends, who prayed for me and checked on me and encouraged me throughout this journey.

Finally, none of this would have been possible without my brother Vinod. My deep and sincere thanks to him.

God bless you all.

List of abbreviations and terms

BIST	Built-In-Self-Test
CUT	Circuit Under Test
ATPG	Automated Test Pattern Generation
FC	Fault Coverage
VLSI	Very Large Scale Integration Circuit
ATG	Automated Test Generation
IC	Integrated Circuit
TPG	Test Pattern Generation
PRPG	Pseudorandom Pattern Generator
MISR	Multiple Input Signature Analyzer

Table of Contents

Author's declaration of originality	2
Abstract	3
Annotatsioon.....	4
Acknowledgments	5
List of abbreviations and terms	6
List of figures.....	9
List of tables	10
1 Introduction	11
1.1 Background and Problem	11
1.2 Description of Task Solved.....	12
1.3 Thesis Structure.....	13
2 Overview of digital testing	14
2.1 Significance of Testing.....	14
2.2 Test Generation	15
2.2.1 Exhaustive testing.....	15
2.2.2 Functional testing	15
2.2.3 Structural testing.....	16
2.3 Fault Models	16
2.3.1 Stuck-At Faults.....	17
2.4 Levels of Abstractions in VLSI Testing.....	18
2.4.1 Register-Transfer Level and Behavioral Level	18
2.4.2 Gate Level	18
2.4.3 Switch Level.....	18
2.4.4 Physical Level	19
2.5 Automatic Test Pattern Generation (ATPG)	19
2.5.1 Backtracking	19
2.6 Fault Simulation.....	21
2.6.1 Serial Fault simulation	21
2.6.2 Parallel fault simulation	21
2.6.3 Deductive fault simulation	23
2.6.4 Concurrent fault simulation	24

2.6.5 Differential fault simulation.....	24
2.7 Digital Circuit Testing	25
2.8 Analog and Mixed-signal circuit testing	25
2.9 Conclusion	26
3 Built-In-Self-Test.....	27
3.1 BIST Architecture	28
3.2 Built-In-Self-Test techniques.....	29
3.2.1 On-line BIST.....	29
3.2.2 Off-line BIST	29
3.3 BIST: Test Generation methods.....	30
3.3.1 Exhaustive testing.....	30
3.3.2 Pseudo-exhaustive testing.....	30
3.4 Pseudorandom Pattern Generation.....	30
3.4.1 Standard LFSR	31
3.4.2 Modular LFSR.....	31
3.5 BIST fault coverage.....	32
3.6 Conclusion	32
4 Development of a new algorithm.....	33
4.1 The architecture of the hybrid BIST	33
4.2 Cost factors of Hybrid BIST.....	35
4.3 Optimization algorithm	38
4.4 Description of the idea of the new algorithm	41
4.5 Description of the algorithm	43
4.6 Conclusion	46
5 Experimental research of the developed method.....	47
5.1 Experimental environment.....	47
5.2 Experimental results	49
6 Conclusion.....	52
References	53
Appendix 1 – Program Description and Manual	55
Appendix 2 – Source.....	56

List of figures

Figure 1 Testing process	15
Figure 2 Single stuck-at-fault	17
Figure 3 backtracking.....	20
Figure 4 Parallel Fault Simulation.....	22
Figure 5 Deductive fault simulation	23
Figure 6 Basic BIST architecture	28
Figure 7 Standard LFSR.....	31
Figure 8 Modular LFSR.....	31
Figure 9 Hardware architecture of hybrid BIST.....	34
Figure 10 Total cost calculation of hybrid BIST	36
Figure 11 ATPG and fault-table based approach	39
Figure 12 Cost functions for Hybrid BIST.....	41
Figure 13 Deterministic test length estimation.....	42
Figure 14 Random pattern generation in Turbo Tester.....	48

List of tables

Table 1 Pseudorandom test results.....	37
Table 2 ATPG results.....	38
Table 3 Results of the exact method and new method for c880.....	45
Table 4 Comparison of the new Hybrid BIST cost minimization method vs. exact method.....	49
Table 5 Comparison of the new Hybrid BIST cost minimization method vs. exact method.....	50
Table 6 Starting point to the Tabu search	51

1 Introduction

The thesis focuses on the new method for the optimization of hybrid BIST by combining the pseudorandom test patterns and the deterministic test patterns to perform the test with a minimum cost of both time, memory, and without losing in test quality. A novel algorithm was developed and experimented with the ISCAS'85 benchmark circuits.

This chapter discusses the background and problem, followed by the goal of the thesis and, finally, the overview of the thesis structure.

1.1 Background and Problem

The exponential growth in recent years has brought many new prospects in VLSI, mainly in the area of integrated circuits design and manufacturing. Nowadays, predesigned complex functional blocks are used in the system for designing. This is called System-on-Chip (SoC) approach, and this leads to a shorter time to market and with a reduced cost. Hence this SoC approach is very inviting from the designer's viewpoint. However, testing of SoC has several problems due to the presence of submicron chips because the complexity is increased as well as another big challenge is due to the protection of intellectual property [1].

Several test approaches were proposed, but those approaches are slow, expensive, and inaccurate. Therefore, Built-In-Self-Test (BIST) is another testing approach where the system is allowed to test itself, and this test approach is highly reliable and reduces time and cost. This BIST needs linear feedback shift registers (LFSR); however, the LFSR does not provide 100% fault coverage, and it takes a longer time to achieve 100% fault coverage. Therefore, a hybrid BIST approach is used to improve the fault coverage. In this approach, pre-computed test patterns are stored in the memory, which is called deterministic test patterns, and it is used to achieve maximum fault coverage.

The main concern of this hybrid BIST approach is to achieve maximum fault coverage by combining the pseudorandom test vectors (generated by LFSR) with the deterministic test patterns. The main objective is to find an optimal balance between the pseudorandom and stored test patterns along with the minimum cost of time and memory, and this is the main contribution to this thesis.

1.2 Description of Task Solved

Two algorithms had been proposed already they are ATPG-based approach and Fault-table based approach [2]. These approaches were resulted in reducing test costs, but it is time-consuming for complex circuits. In order to reduce cost as well as time, this thesis presents the following goals:

- Develop a new algorithm for the optimization of hybrid BIST for the ISCAS'85 benchmark circuits, which will result in reduced cost and a huge gain in time for even complex circuits.
- The second result is to propose a starting point to the Tabu search as close as possible to the exact optimum

1.3 Thesis Structure

The thesis is organized as follows.

In chapter 2, an overview of digital testing is given, and the importance of digital testing, types involved in testing a circuit, and several other things related to digital testing are discussed.

Thereafter, the concepts of Built-In-Self-Test (BIST) and how the testing problems can be overcome by using a BIST strategy are discussed in chapter 3.

In chapter 4, the development of a new method for the optimization of hybrid BIST has explained as well as the cost factors of hybrid BIST and already proposed optimization algorithms, are discussed. The implementation of the new method is discussed in the next chapter, and thereafter the experimental results are discussed in chapter 6.

Finally, the summary and conclusion are discussed in chapter 7.

2 Overview of digital testing

This chapter gives an overview of digital testing. Testing is one of the vital processes for producing a fault-free device. In this chapter, we will discuss why testing is essential for electronic devices and how testing is performed and stages involved during the process of testing. In the test generation section, types of testing are explained, and why structural testing is more practical compared with other types of testing is also discussed. The stuck-at-fault model is explained with an example, and various levels of abstraction in VLSI testing, types of fault simulation, and finally, digital circuit testing, analog, and mixed-signal testing are discussed in this chapter.

2.1 Significance of Testing

In today's electronic devices, many millions of transistors used, and this shows the steady decrease in dimensions, which referred to as the feature size of the transistors. When the feature size reduces, simultaneously operating frequency, clock speed increases. Also, there is a high chance that a manufacturing defect in the Integrated Chip may result in a faulty chip. Even a tiny defect in a chip may ruin the whole device; that is why testing the components at each stage of the manufacturing process is very important. During the testing process, when the cause of defects is found, and faults are met, there will be an improvement in the production at every stage. There is a general agreement with the rule of ten, which says that the cost of detecting a faulty IC increases by order of magnitude as we move through each stage of manufacturing, from the device level to the board level, and finally to system operation in the field [3].

Testing is done by applying the input test stimuli to the inputs of the circuit under test (CUT) while analyzing the output responses [3], as shown in figure 1. Those circuits which give the correct output responses for all the inputs applies then it is a fault-free circuit. Testing carried out at the various stages in the lifecycle of a VLSI device. The lifecycle of the VLSI device consists of various stages, such as the VLSI development process, the electronic system manufacturing process, and system-level operation.

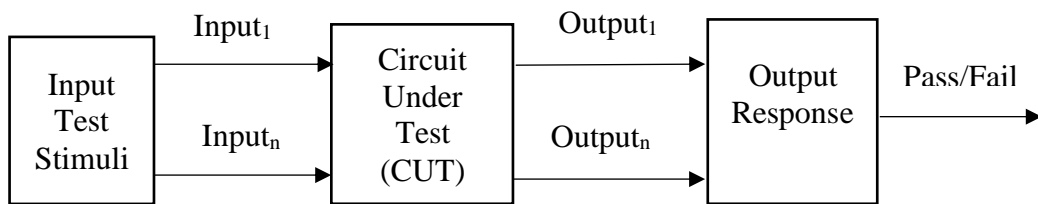


Figure 1 Testing process

The VLSI development process involves testing at each stage of the process. The process involves design, fabrication, packaging, and quality assurance. At the design stage, designers are responsible for analyzing the circuit which satisfies the design specification and design verification. The design verification is the form of testing, and once verified, the VLSI design goes for fabrication. The very first testing is done during the manufacturing process to test the IC is fabricated on the wafer to find which device is defective. The IC's, which pass the wafer-level testing, are extracted and packaged.

2.2 Test Generation

To test a circuit, input patterns are applied to the circuit under test (CUT), and the output is compared with the fault-free circuit [3]. Each input pattern in the circuit is called a test vector. The main goal of test generation is to find an excellent structured set of test vectors that can detect all the faults in the circuit. In order to test one full circuit, more test vectors are needed, but the number of test vectors is not precisely known. For that, the following testing is performed.

2.2.1 Exhaustive testing

If the Circuit Under Test (CUT) is an n -input combinational logic circuit [3], then all 2^n possible input test patterns are applied for testing stuck-at-faults. This is called Exhaustive testing. If the circuit passes the exhaustive testing, then it is assumed that the circuit has no functional faults. This testing is not practical when n is significant [3].

2.2.2 Functional testing

Like Exhaustive testing, all the 2^n possible input test patterns are applied to the n -input combinational logic circuit in functional testing where each value in the truth table for

the combinational logic circuit is tested to check whether it produces the correct [3] output response.

2.2.3 Structural testing

Structural testing is more practical when compared with other types of testing because specific test patterns are selected based on the circuit structural information and a set of fault models. The main advantage of this structural testing is it saves time, and the test efficiency is improved because of the decrease in the number of test patterns as the test vectors targets only specific faults that would result from defects in the manufacturing circuit. The use of fault models provides a quantitative measure of the fault-detection capabilities of a given set of test vectors for the desired fault model. This type of measure is called fault coverage and is defined as [3]

$$\textit{Fault coverage} = \frac{\text{Number of detected faults}}{\text{Total number of faults}}$$

Because of undetectable faults, it is impossible to get 100 percent fault coverage. So, the fault coverage can be modified further as fault detection efficiency which is defined as

$$\textit{Fault detection efficiency} = \frac{\text{Number of detected faults}}{\text{Total number of faults} - \text{number of undetectable faults}}$$

2.3 Fault Models

Fault models are a must for generating and evaluating a set of test vectors. A fault model should accurately give back the behavior of defects, and it should be systematic in terms of fault simulation and test generation. However, unfortunately, not even a single fault model could accurately reflect the behavior of defects. So the combination of different models is used in the generation and evaluation of test vectors for VLSI devices.

A single fault assumption is given by

$$\text{Number of single faults in the circuit} = k \times n$$

Where k is the different types of faults, and n is the possible fault sites in a given circuit.

The multiple fault model is defined as

$$\text{Number of multiple faults in the circuit} = (k + 1)n - 1$$

In the multiple fault model, sometimes the circuit could be fault-free, or it may have a fault, so the “-1” represents the fault-free circuit.

2.3.1 Stuck-At Faults

A stuck-at-fault affects the state of logic gates on lines in the logical circuit, where it includes all the primary inputs, primary outputs, internal gate inputs, and outputs as well as fanout branches. When the logic line is stuck at either a constant logic value 0 or 1, then it is called stuck-at-0 and stuck-at-1, respectively.

Let us consider a pure single stuck-at-fault example circuit in figure 2

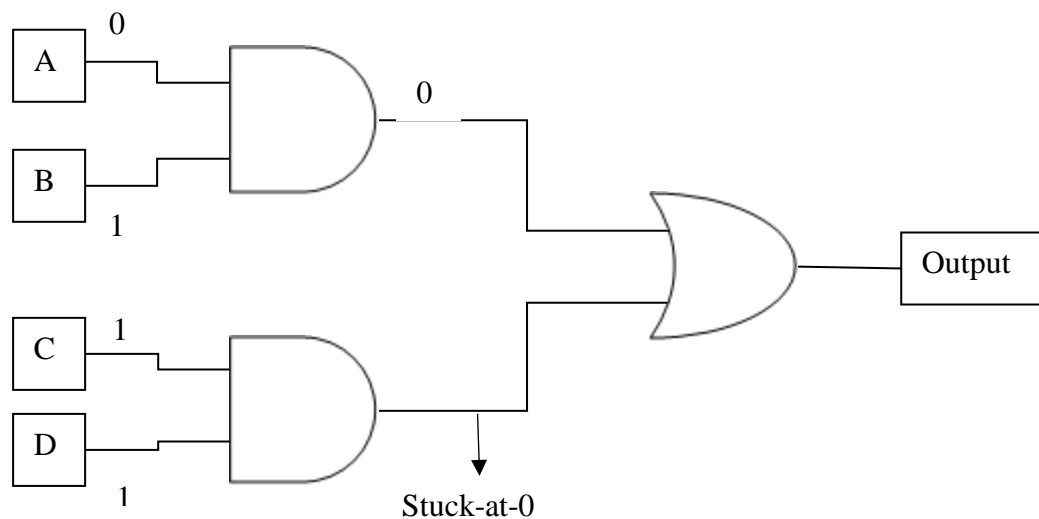


Figure 2 Single stuck-at-fault

From the above logical circuit, we can able to see that there is only one line that is a faulty response. So, if the output is ‘0’, then it is a faulty response when the output is ‘1’, then it is a fault-free response.

2.4 Levels of Abstractions in VLSI Testing

The levels of abstraction include behavioral, register-transfer, logical, and physical levels. In this section, we will discuss the test generation and the use of fault models at each level of abstraction and testing methods.

2.4.1 Register-Transfer Level and Behavioral Level

The register-transfer level contains a netlist of gates, and the stuck-at-faults at this level are the most famous fault models in digital testing [4]. The ATPG tools cannot handle designs exerting blocks because the implementation details are not known. Several approaches are proposed for the test pattern generation at the RTL, but most of the approaches lack general connectivity. Some experimental results had shown that the fault coverage achieved at the RTL could be close to the fault coverage achieved at the gate level, and also stuck-at-fault coverage at the RTL will not be as high as the gate level.

2.4.2 Gate Level

At the gate-level, the stuck-at-fault model can be applied efficiently because many ATPG and fault simulation tools are available. Not only a stuck-at-fault model but also delay fault models, and delay testing is also based on the gate-level description. Since the gate-level model lies between the RTL and physical level, it has the advantage of functionality and tractability. For deep submicron designs, test development at the gate-level is not widely used.

2.4.3 Switch Level

In the switch-level simulation, the whole circuit is cast as an interlink of MOS transistors, which are referred to as ideal switches. The logic behavior of MOS circuits that have no fundamental logic gate structure can be simulated by the switch-level simulation. For both ATPG and fault simulation, the switch-level description is more complicated than the gate-level description. The switch-level is used to simulate stuck-at-short and stuck-open faults, but fault-effect propagation steps are too complicated, and therefore, usage of this level has been limited in the industry [4].

2.4.4 Physical Level

The physical level of abstraction is crucial for VLSI testing [3]. At this level, it provides actual layout and routing information for the fabricated device, and therefore the most precise information for delay faults, crosstalk effects, and bridging faults is obtained. A distributed resistance-inductance-capacitance (RLC) model is used as a basis to characterize electrical properties of interconnections in the deep submicron IC chips. This is used to analyze and test for resolving crosstalk problems. The solution for bridging fault is to take out the capacitance between the wires, and this gives the accurate determination of those wires which are adjacent and, therefore, likely to undergo bridging faults. Hence the fault sites with the highest capacitance can be selected for test generation and evaluation.

2.5 Automatic Test Pattern Generation (ATPG)

ATPG is used to find an input or test sequence when applied to a digital circuit, which automatic test equipment to distinguish between the fault-free circuit behavior and faulty circuit behavior that is caused by defects. Many ATPG algorithms and tools have been proposed, and one of the approaches (a common approach) is to begin from a random set of test patterns. Then fault simulation will determine the detected faults, and to this, additional test vectors are generated to obtain the desired fault coverage.

One of the significant problems in the sizeable sequential circuit, when compared with the combinational circuit, is to identify the undetectable faults because of thousands of gates in the circuit, and also it is difficult to reach 100% fault coverage.

In a naive ATPG algorithm [3], ATPG has to make some decision where it has to select some input vector with primary input #i, which is to set to a specific logic value, which will lead to a solution. In order to make a decision, a decision tree method is followed.

2.5.1 Backtracking

In the decision tree method, when the path leads to no solution, then it should not continue with the same path. Instead, it should go back to the earlier point or node, and it should re-decide the previous decision. If the other branch in the decision tree has not been explored before and if there are only two choices for a decision variable, then

some of the previous decision should be reversed. This type of reversal in the decision is called backtracks.

Consider a decision tree in Figure 2.3 as an example [3].

In this example, the decision made so far is $a = 0$, $c = 1$, $d = 0$, and suppose if this causes the conflict in detecting the target fault then, the most recently made decision should be reversed. As per this example, the most recently made decision is $d = 0$, and this should be reversed as $d = 1$, and all the resulted values from $d = 0$ should be undone. Then the search continues as $a = 0$, $c = 1$, $d = 1$. If this decision also causes conflict, then this path is actually could not lead to any solution that could the target vector. The decision on d is finally assigned as a 'don't care' value. The backtracking mechanism should be continued by changing the previous decision or searching the portion of the search $a = 0$, $c = 0$, and if there is no previous decision, then the ATPG concludes that the target fault is undetectable.

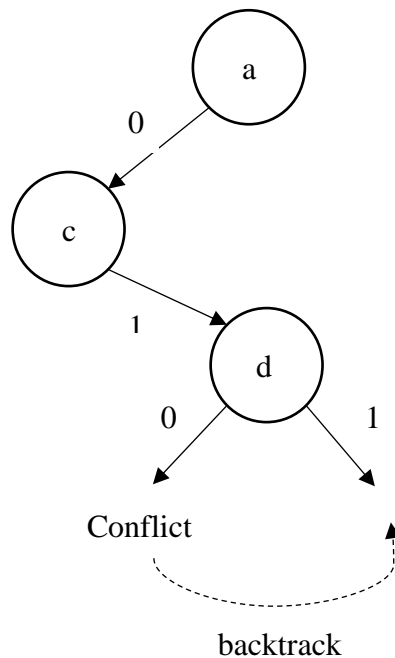


Figure 3 backtracking

2.6 Fault Simulation

Fault simulation is the demanding task because of the complexity that is the circuit with all the faults that must be simulated. While simulating, the amount of computation time is approximately proportional to the number of faults, circuit size, number of test patterns, or test vectors. This becomes infeasible for larger circuits because the computation time will be more. In the following section, we discuss the techniques of fault simulation.

2.6.1 Serial Fault simulation

This is the most straightforward algorithm, among other simulations [4]. Fault-free circuits are simulated first, and the primary output values (true responses) are saved, and the next faulty circuits are simulated. AS the simulation continues, the output responses of the faulty circuit are compared with the results of the saved true responses. When the comparison indicates the detection of target faults, then the simulation of the faulty circuit is stopped soon.

A serial fault simulator can simulate any kind of fault that is introduces in the circuit description. It also simulates bridging faults, dealy faults, and analog faults apart from stuck-at-short and stuck-at-open types [4].

2.6.2 Parallel fault simulation

The idea of this type of fault simulation is to use the bit-parallelism of logical operations ina digital computer [4]. The circuit consists of only logic gates, and so the signals are assumed to have only binary 0 and 1 values, and all the gates have the same delay. Under these conditions, the parallel fault simulation is most effective. On considering a 32-bit machine word, an integer consists of a 32-bit binary vector. The AND or OR logical operation implying two words performs simultaneous AND or OR operation on all pairs of bits. This makes the simultaneous simulation of 32 circuits with the same connectivity but with different signal values.

Let us consider an example [4] of parallel fault simulation in Figure 2.4

Figure 2.4 shows that the circuit is being simulated for two faults, which are c stuck-at-0 and f stuck-at-1. As you can see from the circuit that the computer has a three-bit word. The left-most bit represents the signal value in the fault-free circuit, bit 1, which is the

middle bit represents c stuck-at-0 fault and the right-most bit, which bit 2 represents f stuck-at-1. When we apply a vector a =1 and b = 1, the line a and the stem b are not affected by any fault, and these will have the same value for all the three circuits.

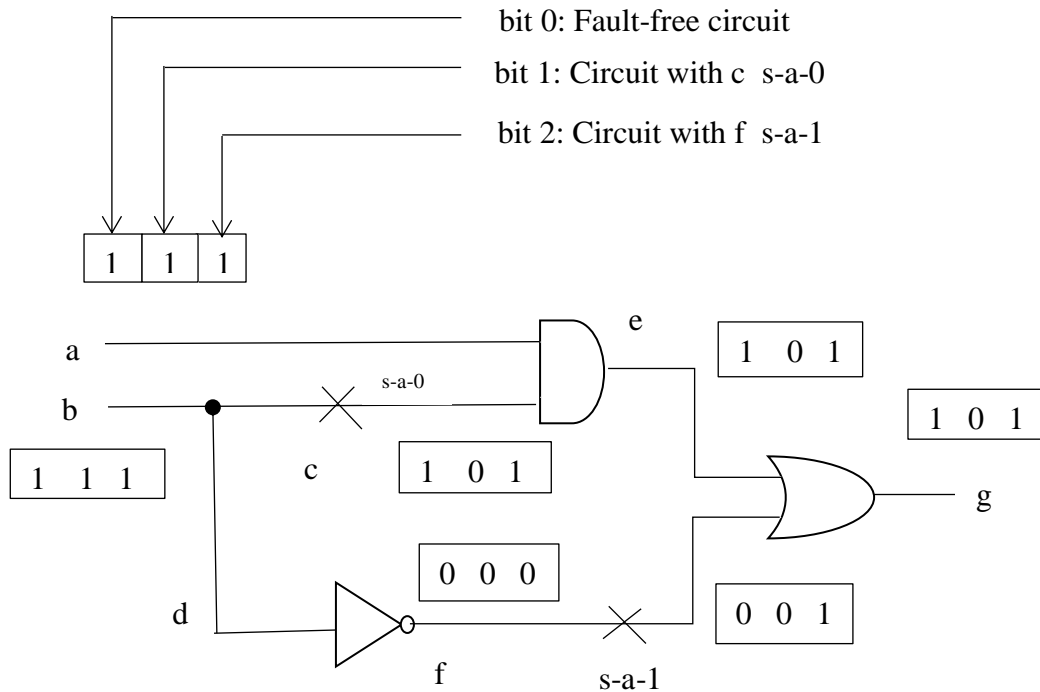


Figure 4 Parallel Fault Simulation

The fault c is the first faulty circuit, and therefore it affects the middle bit. In the first circuit when a and c performs AND operation, we get the 3-bit word for the line e. The second circuit with the NOT gate which has an input 111 and the line f is the inversion of d where the right-most bit is affected by s-a-1 then the word is given by 001. Finally, the output g is obtained by bit-by-bit OR of e and f. Therefore, the output g is 101.

From the result, we can able to see that the output of the circuit c s-a-0 is 101, and the output of the fault-free circuit is 111, it clearly differs. Hence the fault is detected.

2.6.3 Deductive fault simulation

Deductive fault simulation is a very different approach from other simulation techniques because it is based on logical reasoning. This type of simulation can be very fast because only fault-free simulations have to be performed. Let us consider an example [3] figure 2.5

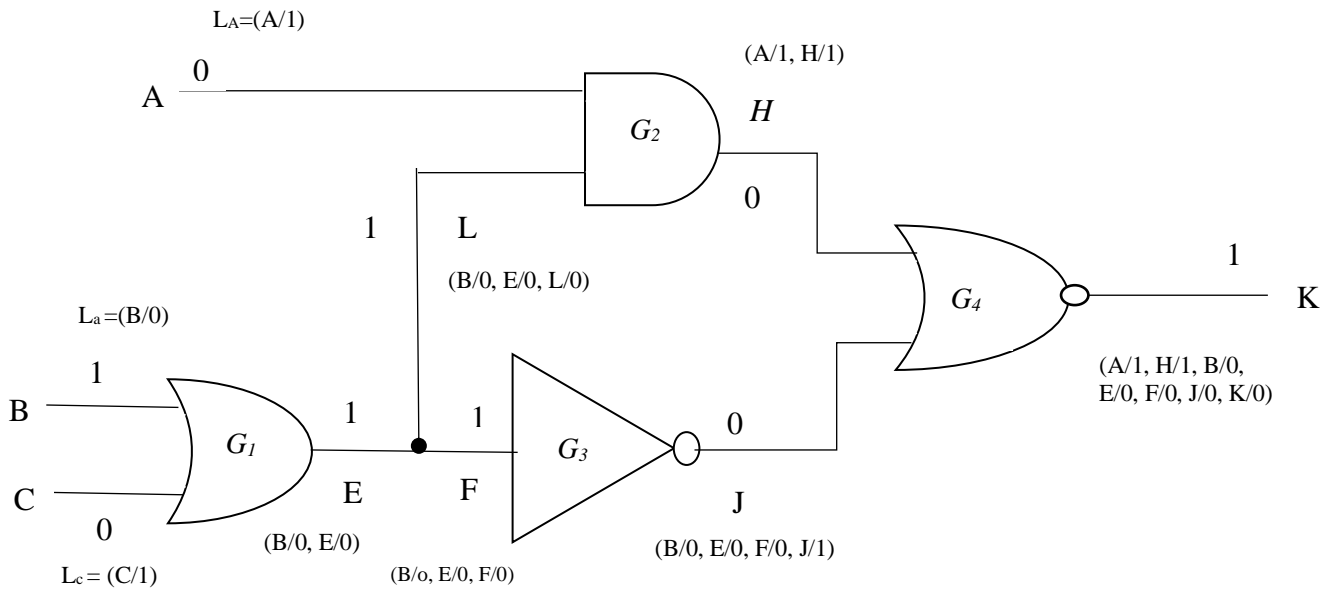


Figure 5 Deductive fault simulation

From figure 5, L_x is the fault list, and it is related to a signal x , and also, it is shown that the fault list of each signal concerning the test pattern P_1 . For the primary input A , the fault list is L_A , and the fault is $A/1$, not $A/0$ because the value of A remains correct when the fault $A/0$ is present. In the same way, the fault lists are derived for the inputs B and C . The method of obtaining the fault list of a gate output from the gate inputs is called **fault list propagation**. Thus, all the faults at the gate inputs are propagated to the gate output. From figure 2.5, we can conclude that the test pattern P_1 directed seven faults in L_K . Therefore, we can see that the advantage of deductive fault simulation is that all the faults detected by the test pattern are acquired in one fault list propagation. In practice, faults are collapsed before the simulation, but only the collapsed faults are considered for the fault list propagation.

2.6.4 Concurrent fault simulation

The fault-free circuit and faulty circuit differ in a very small part, so the concurrent fault simulation only makes use of this fact and simulates only the differential parts of the whole circuit. Hence the fault-free circuits and faulty circuits are simulated altogether.

In this type of simulation, every gate has a concurrent fault list which comprises of a set of bad gates. Each bad gate contains a fault index and the related gate I/O values in the presence of the corresponding fault [3]. The concurrent fault list of gate 1 contains local faults of gate 1. The local faults of gate 1 are faults on the inputs or outputs of gate 1. While the simulation proceeds, the concurrent fault simulation contains local faults, as well as the faults propagated from the previous stages. Local faults in gate 1 remain in the fault list until they are detected.

2.6.5 Differential fault simulation

The concurrent fault simulation has a memory problem because the size of the fault list changes at run time while the single fault propagation technique builds the state of the faulty circuit from the fault-free circuit. However, both of the techniques are not good for sequential fault simulation. Differential fault simulation combines the advantages of both the techniques in such a way to simulate only the difference between the faulty circuit and previously simulated one.

In the differential fault simulation, for every test pattern, a fault-free simulation is accomplished first, and then the faulty circuits simulations are performed. As mentioned before, the states of every circuit are reinstated from the last simulation. If there is a difference in the outputs of faulty circuits and the good circuits, then the faults are detected and dropped. Every time the state difference is stored for every circuit. Also, the state difference of dropped faults should be collected in the state differences of the next undetected fault. This is repeated until there are no test patterns or no undetected faults are left.

2.7 Digital Circuit Testing

The digital circuit testing began with the stuck-at-fault model and then followed by the bridging fault model, transistor fault model, and delay fault model. Digital testing uses the combination of tests developed for different fault models because the tests for fault model cannot guarantee the detection of all defects.

Digital testing is improved by quiescent power supply current (I_{DDQ}) and transistor power supply current (I_{DDT}). The leakage of current in the CMOS circuit under a quiescent state is very small. When a fault occurs, it causes the conducting path from power to the ground, and so it might draw an excessive supply current. However, I_{DDQ} testing is ineffective for larger devices because it consists of millions of transistors, and there will be more leakage current in the circuit. It is similar to the I_{DDT} testing approach. As the number of transistors in the VLSI devices continues to grow, both I_{DDQ} and I_{DDT} testing suffer.

2.8 Analog and Mixed-signal circuit testing

Analog circuits are used in various applications, and mixed-signal circuits include analog circuitry and digital circuitry. Analog circuit testing involves two categories: Specification-oriented testing and waveform-oriented testing. The specification-oriented testing approach tests each and every specification in the datasheet to regulate the pass/failure of the circuit. It undergoes two types of tests which are bench test and the final test. The bench test is done in the laboratory for the characterization purpose while the final test is done in the industry before delivering it to the customers.

The waveform-oriented testing measures selected or particular parameters of response waveforms to check the pass/failure of the circuit under test (CUT). The test points must be chosen carefully from the waveform-oriented testing. The correlation of the test points along with the specifications is essential for the improvement of test yield and in reducing the defect level. Mixed-signal circuit testing begins by converting Analog to digital and digital to analog. After that testing method is applied, it involves testing like time-domain testing, code-transition level testing, gains, and offset testing.

2.9 Conclusion

This chapter provides an overview of VLSI testing. The significance and challenges of VLSI testing at various levels were discussed. Why do we need VLSI testing? How difficult is VLSI testing? Although many of these points were briefly reviewed in this chapter. However, the new and continuing testing challenges and advancements in technology need testing like Built-In-Self-Test (BIST), where the system tests on itself, which will result in reduced testing time, test cost, power consumption and many more. More about BIST is discussed in the next chapter.

3 Built-In-Self-Test

The complex functional blocks, usually referred to as cores, are very different and can be represented in several different ways. This type of design style allows designers to reuse the previously designed cores, and therefore it will take a shorter time to the market and reduces cost. However, testing of SoC (System-on-Chip) has some problems due to the protection of intellectual property, increased complexity, and higher density [1]. In order to test the individual cores of the system, we need test pattern source and sink available together with test access mechanism [2] [5] , but the sink off-chip requires the use of external Automatic Test Equipment (ATE), but the technology of ATE is always one step behind when compared the internal speed of SoC which constantly increases and therefore the ATE solution becomes unacceptably expensive, inaccurate and unacceptable yield loss [2] [6]. Hence, to keep the test costs under control and to apply at-speed tests, on-chip solutions are needed. Such a kind of solution is referred to as a built-in-self-test (BIST) [2]. The SoC BIST will result in reduced testing time, reduced memory cost, reduced power consumption and hardware cost, and increased test quality.

The first definition of BIST was given by Richard M. Sedmak which is “... *the ability of logic to verify a failure-free status automatically, without the need for externally applied test stimuli (other than power and the clock), and without the need for the logic to be part of a running system.*” [7].

Built-In-Self-Test (BIST) is nothing but to design a circuit where the circuit can test itself and determine whether it is good or bad, i.e., fault-free or faulty circuit [8]. This needs an added circuitry and functionality to be incorporated in the circuit design to ease the self-testing feature. This added feature should be capable of generating test patterns, and also it should provide a mechanism to check the output responses of CUT to the test patterns correlate to that of the fault-free circuit.

In this chapter, we will discuss the BIST architecture, where we will examine how the BIST works, BIST techniques where we will discuss the on-line BIST and off-line BIST, test generation methods, pseudorandom test pattern generation, fault coverage and finally the logic BIST.

3.1 BIST Architecture

The BIST architecture consists of two essential functions and two additional functions that are necessary to facilitate the execution self-testing feature in the system. The architecture of the BIST is shown in figure 3.1. The two essential functions are the Test Pattern Generator (TPG) and Output Response Analyzer (ORA).

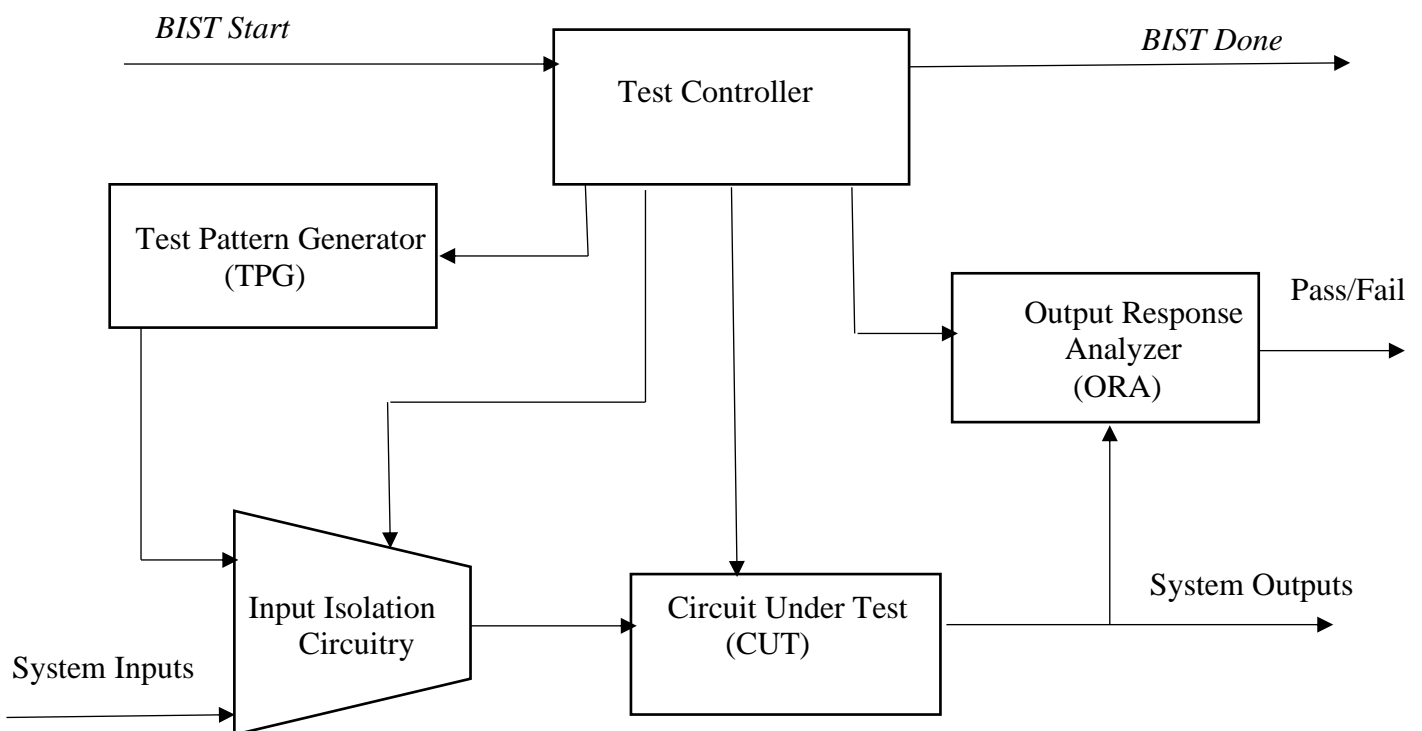


Figure 6 Basic BIST architecture

The test pattern generator produces a sequence of patterns for testing the circuit under test, the output response analyzer (ORA) compress inputs from CUT and produces the outputs as pass/fail indication. Two other functions needed in the BIST design for the system-level use include the test controller and the input isolation circuitry. It requires additional I/O pins apart from normal pins, for activating the BIST sequence, reports the

results of the BIST and an optional pass/fail indication that the BIST sequence is done and the BIST results are valid, and it can determine the fault-free or faulty status of the CUT.

3.2 Built-In-Self-Test techniques

BIST allows complete testing at a reasonable cost [9]. The most commonly BIST techniques are classified as on-line BIST and off-line BIST, where on-line BIST comprises of concurrent and nonconcurrent techniques, where off-line BIST comprises of functional and structural approaches.

3.2.1 On-line BIST

This test occurs during normal functional operation. Below are the types of on-line BIST

Concurrent online BIST

This type of testing occurs simultaneously with the normal operation mode; coding-techniques or duplication are usually used. It detects faults during this testing process.

Nonconcurrent on-line BIST

To test for faults, the testing process requires the suspension of normal system operation, frequently by executing diagnostic software or firmware routines.

3.2.2 Off-line BIST

During off-line BIST, the system will not work on its normal working mode, but it will be in a test mode where the test patterns are given to the circuit, and the output responses are analyzed. Usually, on-chip generators and test response analyzers work during this process. There are two types of off-line BIST, and they are explained below.

Functional off-line BIST

It is based on testing the system functionality that is on the functional description of the CUT, which is Component Under Test here, and it uses functional high-level fault models.

Structural off-line BIST

The test is performed based on the structure of the functional circuitry. There are two general categories of Structural off-line BIST: 1. External BIST, where the output response analysis and test pattern generation are done by circuitry, which is separate from the functional circuitry, which is being tested. 2. Internal BIST, where the functional storage elements are converted into output response analyzers and test pattern generators [10] .

3.3 BIST: Test Generation methods

3.3.1 Exhaustive testing

A BIST approach in which all the possible 2^n patterns are applied to n circuit inputs [4]. The main advantage of exhaustive tests is that no need for test pattern generation, fault simulation, and fault model is not needed; redundancy problem is eliminated, 100% fault coverage for single and multiple fault-models. However, long test length is needed, and also there will be a problem in CMOS stuck-open faults.

3.3.2 Pseudo-exhaustive testing

A BIST approach in which a particular circuit is having n number of primary inputs are broken into smaller, each with $< n$ inputs. Each smaller blocks is tested exhaustively [4]. The pseudo-exhaustive test sets consist of types of function verifications. They are output function verification and module function verification.

3.4 Pseudorandom Pattern Generation

The linear feedback shift register (LFSR) pattern generator is commonly used in pseudorandom pattern generation. These patterns have all the advantageous properties of random numbers but algorithmically generated by the hardware pattern generator, and hence it is repeatable, which is more needed for BIST. Basically, pseudorandom pattern generation needs more test patterns than deterministic ATPG [4]. There are two types of LFSR, which are standard LFSR and Modular LFSR.

3.4.1 Standard LFSR

A linear feedback shift register (LFSR) is a shift register where the input bit is a linear function of its previous state. Figure 7 shows the 4-stage standard LFSR. It consists of 4 flipflops and one exclusive-OR gate. Normally, XOR gates are placed on the external feedback path, and henceforth, the standard LFSR is also called an **external-XOR LFSR** [3].

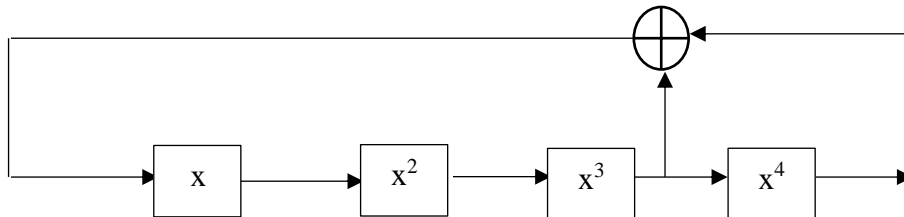


Figure 7 Standard LFSR

3.4.2 Modular LFSR

Figure 8 shows the 4-stage modular LFSR, where the XOR gate is placed between two adjacent D flipflops, and it is also known as an **internal-XOR LFSR**. The modular LFSR can run faster than the standard LFSR because it has an XOR-gate delay between adjacent flipflops.

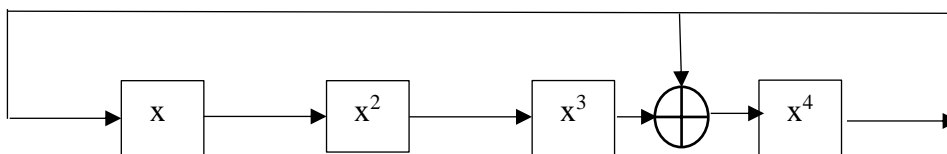


Figure 8 Modular LFSR

3.5 BIST fault coverage

The fault coverage can be regulated for the set of test vectors. The fault coverage is a perceptible measure of the effectiveness of the set of test vectors in detecting the faults and the fault coverage, FC, is given by [8]

$$FC = \frac{D}{T}$$

Where D is the number of detected faults, and T is the total number of faults in the fault list. Achieving 100% fault coverage is difficult, but in the ISCAS'85 benchmark circuits, a circuit like c880 had achieved 100%.

3.6 Conclusion

The chapter discussed the need for BIST, the architecture of BIST, the techniques of BIST, test generation methods, and BIST fault coverage. From this chapter, it is proved that the BIST is for cost-efficient testing; it reduces the difficulties with TPG (Test Pattern Generation), it reduces the test application time.

However, few drawbacks like decreased reliability due to increased silicon area in the circuit and performance impact due to additional circuitry. Also, LFSR does not always guarantee the 100 % fault coverage. In order to overcome these drawbacks, hybrid BIST is introduced to achieve maximum fault coverage. In the following chapter, hybrid BIST is discussed in detail.

4 Development of a new algorithm

In this chapter, the previous methods and the newly developed method for the optimization of hybrid BIST are discussed. The main concern of the hybrid BIST is to improve the fault coverage by mixing pseudorandom test patterns with the deterministic test patterns at the same time with a minimum cost of both time and memory. For selecting the optimal brake point from PRG mode to deterministic pattern mode, two algorithms are implemented in [2]. Hence, the previous methods resulted in cost optimization, but it is time-consuming at the same time it is not applicable for complex circuits. A new algorithm is developed to find the optimal balance between the pseudorandom and deterministic test patterns with a minimum cost of both time and memory, which results in a huge gain in time.

The first section presents the general architecture of the hybrid BIST which combines the sources of pseudorandom and deterministic test patterns.

The second and third section presents the main cost factors of hybrid BIST and analyzes two typical schemes for BIST design cost optimization, respectively.

The fourth and fifth section presents the idea of a new algorithm and explains the new algorithm with an example.

4.1 The architecture of the hybrid BIST

The hardware architecture of hybrid BIST is depicted in figure 9, where the Multiple Input Signature Analyzer (MISR) and pseudorandom pattern generator (PRPG) are enforced inside the Core Under Test (CUT). The deterministic test pattern is pre-computed and stored inside the system.

Normally the cores are divided into two classes. In the first class, the core contains its own pseudorandom test pattern generator, and deterministic test patterns alone have to be transported to the core. The second class has cores with no prior BIST structures; therefore, pseudorandom test vectors and stored test patterns have to be shifted to the CUT from external sources.

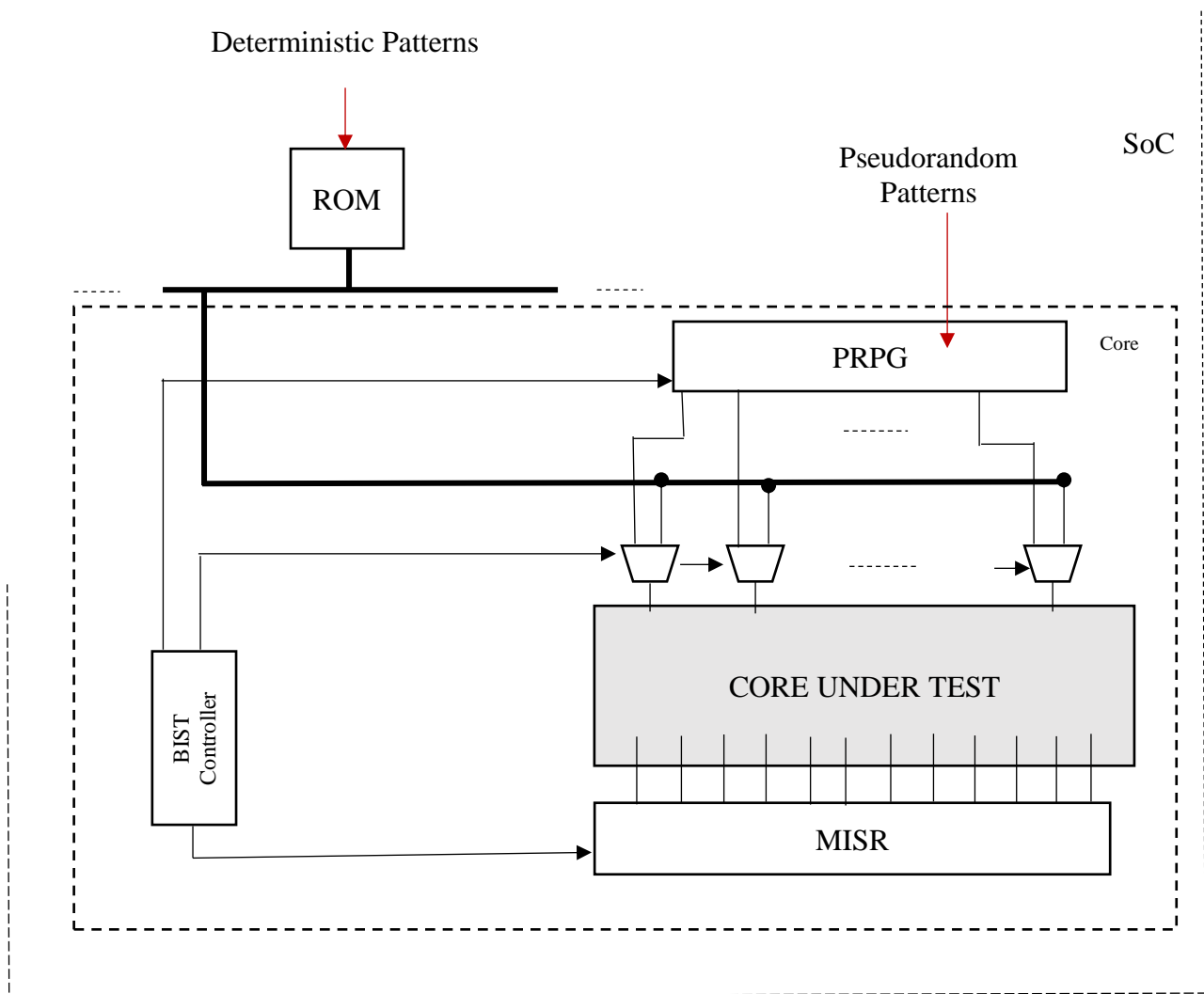


Figure 9 Hardware architecture of hybrid BIST

In figure 9, we considered a core-level hybrid BIST where PRPG and MISR are implemented inside the CUT using LFSR. The deterministic test patterns are pre-computed and stored outside the core in a ROM.

The pseudorandom patterns generated by LFSR do not always guarantee 100 % fault coverage, and sometimes for reaching maximum fault coverage, it needs longer pseudorandom test, which will lead to longer test application time. As mentioned earlier in the previous section, if the LFSR had fixed to run the patterns for a specific period of interval (window), it will result in a lot of hard-to-test faults that will remain behind the window and not detected.

So, the purpose of hybrid BIST is that the pseudorandom test is improved by a stored test set (deterministic test patterns), which is especially generated to target the **random resistant faults**. The stored test patterns will reduce the overall testing time, but it needs additional memory space. This hybrid BIST approach begins with the on-line generation of pseudorandom test sequence where the length is determined as L , and then the stored test approach takes place, where the pre-computed test patterns are stored in the memory, are applied to the CUT to achieve the 100% fault coverage. Based on deterministic, random, or genetic algorithms, arbitrary software generators may be used for off-line generation of deterministic test patterns S (number of stored test patterns) [11].

4.2 Cost factors of Hybrid BIST

The parameter L , which is the pseudorandom test patterns, is considered important because it determines the performance of the whole test. According to [11] and [2], the shorter pseudorandom test set entails a larger deterministic test set. Therefore it shortens the overall testing time at the same time it requires additional memory space. On the other hand, a longer pseudorandom test will lead to longer test application time with reduced memory space. Thus it is crucial to determine the optimal length of the pseudorandom test set in order to minimize the total test cost. Let us discuss the cost factor of hybrid BIST in detail.

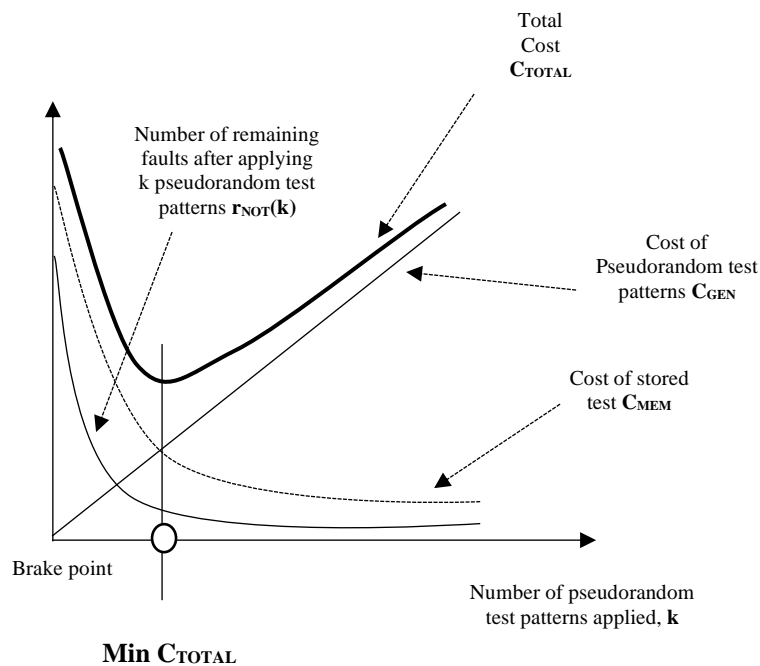


Figure 10 Total cost calculation of hybrid BIST

The above figure 10 demonstrates graphically the calculation of the total cost of hybrid BIST composed of pseudorandom test patterns and deterministic test patterns, which are generated off-line. The total test cost is defined as

$$C_{TOTAL} = C_{GEN} + C_{MEM} = \alpha L + \beta S$$

Where C_{TOTAL} is the total test cost of hybrid BIST, C_{GEN} is the cost associated to the time for generating pseudorandom test patterns L (number of clock cycles), C_{MEM} is the memory cost for storing pre-computed test patterns S (number of deterministic test patterns) and the constants α and β are to map the pseudorandom test length and memory space to the costs of two parts of the test solutions to be mixed [2] [11].

Each value for the total cost C_{TOTAL} is found by adding the pseudorandom test patterns L with the constant α and the deterministic test patterns S with the constant β . Each value for the cost of pseudorandom test patterns C_{GEN} is found by multiplying the pseudorandom test patterns L (number of clock cycles) with the constant α . Each value for the memory cost C_{MEM} is found by multiplying the deterministic test patterns S with the constant β . Likewise, all the cost values are found and plotted in the graph. From figure 10, we can able to see the minimum value of C_{TOTAL} , which is considered to be the brake point.

Also, Figure 10 shows how the cost of testing for pseudorandom test αL is increasing when striving to achieve higher fault coverage. Generally, the testing cost can be very expensive to achieve higher fault coverage with pseudorandom test patterns only. The curve βS illustrates the cost that we have to pay for storing the pre-computed test patterns at the given fault coverage achieved by pseudorandom testing. The sum of the two costs αL and βS is the total cost C_{TOTAL} . The weights of α and β show interrelation between the cost and the memory size needed for storing the pre-computed test sets or the cost and the pseudorandom test time, i.e., the number of clock cycles. For example, if we take $\alpha=1$, and $\beta = B$ is the number of bytes of the input test vector that are given to the core under test (CUT).

A BIST simulation for the ISCAS'85 circuit c880 is carried out using the Turbo Tester software [12] , and the results are given below in table 1, where it shows the pseudo-random test results. Here

k is the number of the clock cycle,

$r_{DET}(k)$ is the number of new faults detected at the clock signal k ,

$r_{NOT}(k)$ is the number of faults which are not yet detected by the sequence by k clock signals,

$FC(K)$ is the fault coverage achieved by the sequence of patterns generated by k clock signals.

Table 1 Pseudorandom test results

k	$r_{DET}(k)$	$r_{NOT}(k)$	FC(k) (%)
1	298	1252	19.22
2	494	1056	31.87
3	619	931	39.93
4	697	853	44.96
50	1131	239	84.58
100	1401	149	90.38
250	1221	329	89.25
500	1131	239	84.5
1000	1498	52	96.64
2000	1511	39	97.48
5000	1547	3	99.8
8000	1544	6	99.61
10,000	1543	7	99.54
12,000	1550	0	100

The parameter k is found by generating a BIST emulator for each number of clock cycle along with that the number of detected faults $r_{DET}(k)$, and the fault coverage $FC(k)$ is also generated. From the table, we can able to see that from the result of simulation for each clock cycle, the set of faults that were covered at this clock cycle. In the table, not all clock cycles are represented only the clock cycles that are interesting that cover at least one new fault is mentioned. At the same time, the total fault coverage for the

pseudorandom test sequence increases up to this clock number increases. Hence such clock numbers and the related pseudorandom test patterns are referred to as resultative clocks and resultative patterns [11].

Creating the curve aL is not difficult. In contrast, it is more difficult to find the values for βS . Let $t(k)$ be the number of stored test patterns (these patterns are pre-computed and use as a stored test pattern in the hybrid BIST) required to cover the not yet detected faults $R_{NOT}(k)$. Meanwhile, calculating $t(k)$ is the most expensive procedure. Two algorithms were proposed [2] for calculating $t(k)$. Let us see in detail the optimization algorithm in the next section.

4.3 Optimization algorithm

As mentioned before, calculating $t(k)$ is the most expensive procedure. The data for the circuit c880 is given below in table 2.

Table 2 ATPG results

k	t(k)
1	88
2	78
3	74
4	77
50	52
100	47
250	45
500	52
1000	9
2000	6
5000	3
8000	2
10000	1
12000	0

Two algorithms were proposed [2] to find $t(k)$: They are ATPG based and fault table-based approach. They have the following representations.

- i is the number of the resultative clock cycle.

- $k(i)$ is the number of the clock cycle of the resultative clock i .
- $R_{DET}(i)$ is the new faults detected by the pseudorandom test pattern that is generated at the resultative clock number i .
- $R_{NOT}(i)$ is the set of not detected faults even after applying for the pseudorandom test pattern number i .
- $T(i)$ is the set of test patterns required and found by the ATPG to cover the faults in $R_{NOT}(i)$.
- N is the number of resultative patterns created by the pseudorandom test.
- FT is the fault table for a given set of tests T and for the given set of faults R .

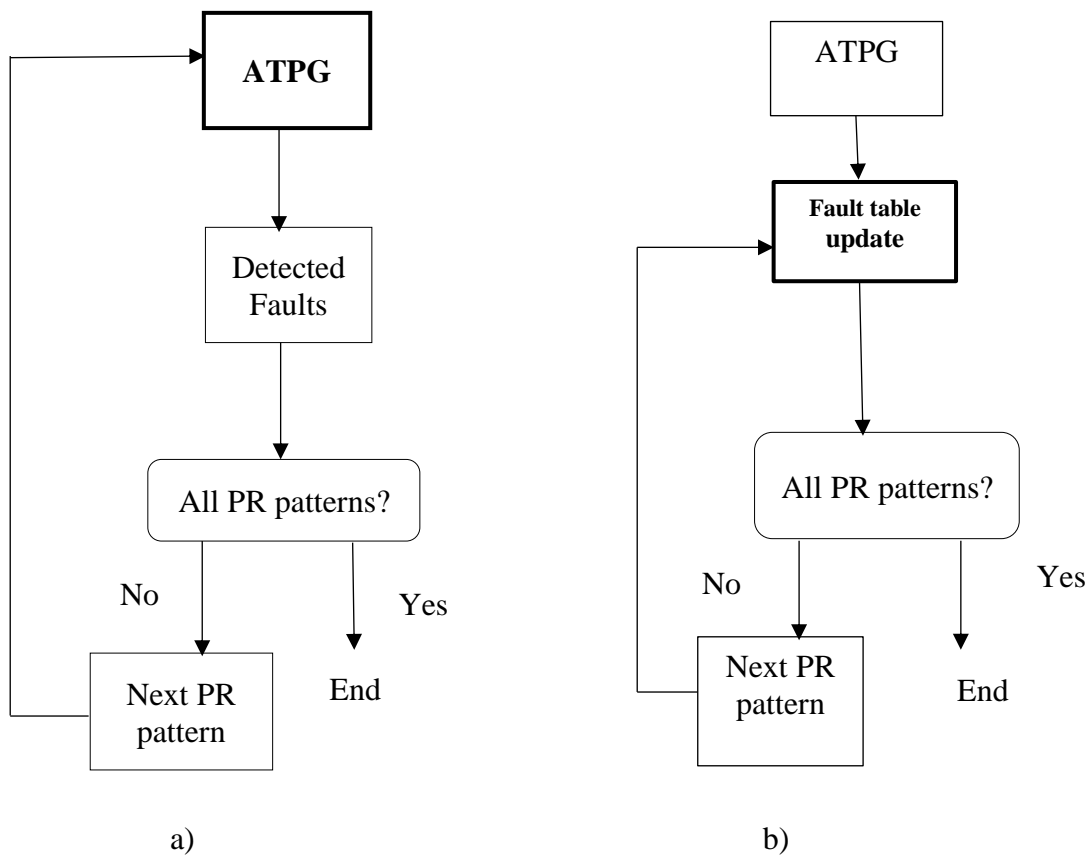


Figure 11 ATPG and fault-table based approach

In ATPG based approach, for each breakpoint of p-sequence, ATPG is used. Figure 11 shows how the algorithm is carried out. The steps that are carried out in the algorithm are given below.

1. Take $i=N$;

2. Generate a test set $T(i)$ for $R_{NOT}(i)$, $T = T(i)$, $t(i) = |T(i)|$;
 3. For all the values of $i = N-1, N-2, \dots, 1$;
 Generate a test set $T(i)$, $T := T + T(i)$, $t(i) := |T|$ for all the faults $R_{NOT}(i)$ not covered by the test T .
- End.

While in the fault table-based approach, the first deterministic test set with the fault table is calculated, and for each breakpoint of p-sequence, the fault table is updated every time, and the remaining deterministic patterns are determined. The steps that are carried out in the algorithm are given below.

1. Take $i=1$; calculate the whole test T for the whole set of faults R , create the fault table $FT(i)$;
 Rename $T(i) = T$, $R(i) = R$;
 2. For all $i = 2, 3, \dots, N$;
 Create a new fault table $FT(i)$ by removing it from the faults $R_{DET}(i-1)$,
 And optimize the test $T(i-1)$ in relation to $FT(i)$. The optimized test set is $T(i)$.
- End.

For implementing the first approach for the whole set of ISCAS'85 benchmark, circuits are carried out within about 8 hours. However, in the case of very large circuits, both of these algorithms will be very expensive and time-consuming experiments. The cost calculation for the algorithm one is carried out as $C_{TOTAL} = C_{GEN} + C_{MEM} = \alpha L + \beta S$ where the weight α was taken as 1 for the number of clocks in the PRG mode, and the weight β was taken as the number of bytes in the memory needed for storing the pre-computed test patterns. For the second approach, based on the number of remaining faults for the resultative clocks, a cheap cost estimation function was used, and the coefficient was chosen as one remaining fault will be equal to 0.45 test patterns and the cost is given as $C_{TOTAL_EST} = C_{GEN} + C_{MEM_EST} = \alpha L + 0.45\beta F$ [2] where F is the number of remaining faults which are not covered by the PRG mode.

The experimental results for these two algorithms from [2] show that the first approach is time-consuming, and it cannot be suggested for larger circuits while the second approach, which is a fast cost estimation method can give the results with acceptable

accuracy, but it doesn't work for complex circuits. Therefore from these two algorithms, the first approach is costly and time-consuming. In this research, a new method is developed, which is also a fast estimation method to find the value of $t(k)$, which works for even complex circuits with less time.

4.4 Description of the idea of the new algorithm

The goal of the thesis is to develop, evaluate, and optimize hybrid BIST solution for the ISCAS' 85 benchmark circuits by combining pseudorandom and deterministic test sequences. For optimization of the test cost, the total cost C_{TOTAL} of hybrid BIST was selected in the form

$$C_{TOTAL} = C_{PR} + C_D = \alpha L + \beta S$$

It is similar to the total cost that we have discussed earlier in the previous chapter. In this total cost, where C_{PR} is the cost related to the time for generating pseudorandom patterns L , C_D is the memory cost for storing pre-computed deterministic test patterns S . The constants α and β are the weights to unify the cost for C_{PR} and C_D . For simplification, the assumption is taken as $\alpha = \beta = 1$, and the graphical representation of the functions $C_{PR}(L)$, $C_D(L)$, and $C_{TOTAL}(L)$ is shown in figure 12.

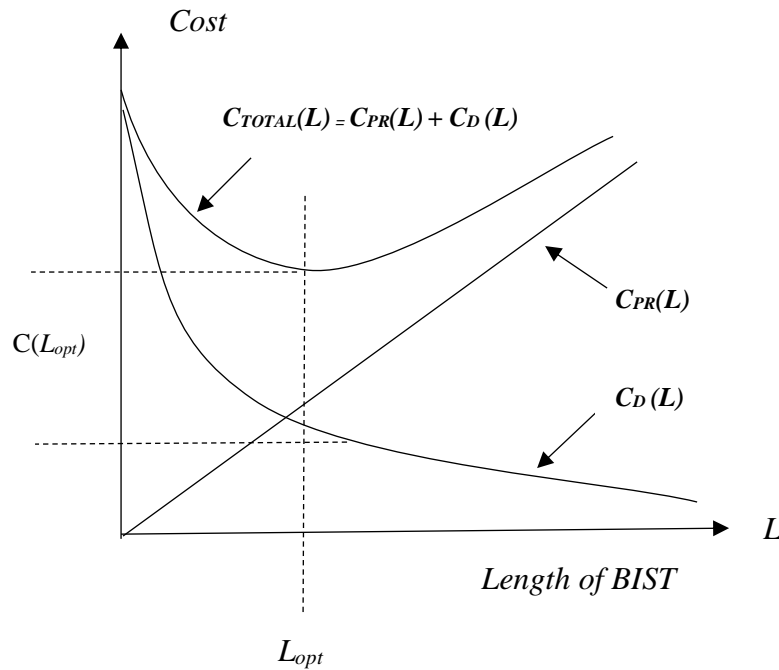


Figure 12 Cost functions for Hybrid BIST

In figure 12, we can able to see that from the curve $C_{TOTAL}(L)$, we can easily find the minimum value and determine the brake point of the optimized test process with minimum BIST cost from the pseudorandom test sequence PR to deterministic test sequence D.

The problem of implementing this BIST cost optimization measure is the high-cost calculation of the curve $C_D(L)$ because, at each point, a deterministic test has to be generated, which is very expensive. To avoid this drawback, a new algorithm is developed for the fast estimation of the value $C_D(L)$ for each point of L . The main idea of the algorithm is given in figure 13.

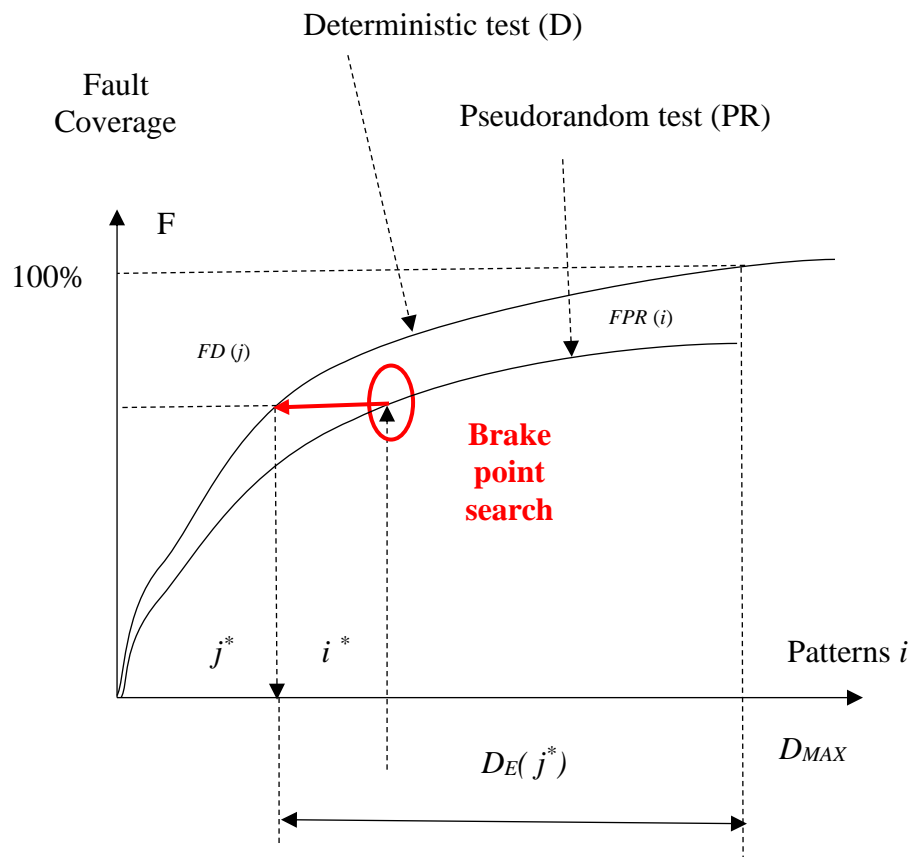


Figure 13 Deterministic test length estimation

Figure 13 represents the deterministic test length estimation, and the graph is plotted between the number of pseudorandom test patterns i (x-axis) and the fault coverage. As

you can see from the graph (figure 13), the pseudorandom fault coverage $FPR(i)$ does not reach 100 % fault coverage, but the deterministic fault coverage $FD(j)$ achieved 100% fault coverage. From the graph (figure 13), for each value of i^* , $FPR(i^*)$ is calculated, and its corresponding deterministic test pattern j^* is found. By subtracting the j^* from the maximum value of deterministic test patterns, the estimated number of deterministic test patterns $DE(j^*)$ can be calculated. Using this estimated deterministic test pattern, $C_{TOTAL}(L)$ is calculated, and we can find the optimal brake point. Using this optimal point, we have to switch from PRG mode to deterministic test pattern mode, which allows achieving maximum fault coverage with the huge gain in time because of the brake point, the longer test application time of pseudorandom test pattern is stopped, and the stored test patterns are used. Here the Turbo Tester software is used to generate all the data we required. The Turbo Tester system consists of tools for Automatic Test Pattern Generation, fault simulation, Built-In-Self-Test(BIST), test set optimization, and other interesting tools [12]. The turbo tester is discussed in detail in the next chapter.

4.5 Description of the algorithm

The input of the algorithm is the full deterministic test set and the pseudorandom test sequence generated by the BIST emulator. The output of the algorithm is the estimated deterministic test patterns, which helps to reduce testing time.

The following defined notations are used in the algorithm.

Notations:

- $FD(j)$ - fault coverage of deterministic test set
- D_{MAX} - number of test patterns
- $FPR(i)$ - fault coverage of pseudorandom test patterns
- $C_D(L)$ - deterministic test cost
- $DE(j^*)$ - estimated number of deterministic test patterns

The new algorithm for the optimization of BIST cost :

1. Generate a full deterministic test set by a single run and calculate $FD(j)$ and D_{MAX} .
 2. Generate PR sequence with BIST simulator and calculate $FPR(i)$.
 3. For selected values of i^* , calculate $FPR(i^*)$ and find the number of deterministic patterns j^* , so that $FD(j^*) = FPR(i^*)$.
 4. Estimated number of patterns is obtained by $DE(j^*) = D_{MAX} - j^*$
- End

Therefore from the algorithm, the full curve for estimated $C_D(L)$ can be calculated, and also estimated minimum value of $C_D(L_{opt})$ can also be calculated, which you can see from figure 12. The parameter L is cost related to time for generating pseudorandom test patterns here. It is given in the form of a number of clock cycles. The constants α and β are taken as one and the number of bytes of the input test vector. Let us consider an example for the circuit c880; the number of inputs is 60; hence the β weightage is 60/8. The experiments are carried out for this weightage for the ISCAS'85 benchmark circuits, and this simplification could make the designers decide according to their requirements. This experiment was carried out for the benchmark ISCAS'85 circuits [13] , and it is investigated, and optimized test sequences are synthesized for two approaches: the exact method and the new method.

The exact method is the labor-intensive method where a bash Unix shell script is written (see Appendix 1) for each circuit which gives the results of a number of clock cycles for generating pseudorandom vectors k , the number of deterministic test patterns $t(k)$ from this we can find the total cost for the exact method C_{exact} (see Table 3). This experiment is performed again for the comparison purpose, and the results are given below in table 3 for the circuit c880.

Table 3 Results of the exact method and new method for c880

k	t(k)	Det.time	Rnot	DT		Cexact	Cnew	Csimple	
				estim					
1	94	0.031	1315	83		706	624	9864	
2	94	0.016	1260	83		707	625	9452	
4	85	0.016	1098	83		642	627	8239	
10	74	0.016	722	79		565	603	5425	
21	62	0.016	419	69		486	539	3164	
36	52	0.000	192	45		426	374	1476	
51	44	0.016	134	33		381	299	1056	Error
101	28	0.016	54	17		311	229	506	0%
209	16	0.000	52	13		329	307	599	
259	15	0.000	28	11		372	342	469	
268	12	0.000	19	10		358	343	411	
286	10	0.000	12	8		361	346	376	
300	8	0.000	10	7		360	353	375	
367	6	0.000	6	5		412	405	412	
700	5	0.000	5	5		738	738	738	
820	3	0.000	3	3		843	843	843	
1144	2	0.016	2	2		1159	1159	1159	
1517	1	0.000	1	1		1525	1525	1525	
1821	0	0.000	0	0		1821	1821	1821	

In table 3, the parameter k is the number of clock cycles for generating pseudorandom test vectors, $t(k)$ is a number of deterministic test patterns to cover remained faults generated using the exact method, $Det.time$ is the time for generating deterministic test patterns, $Rnot$ is the number of remaining faults which are not detected, $DT estim$ is the number of estimated deterministic test patterns resulted from the newly developed method, C_{exact} is the total cost of exact method which is calculated as $C_{exact} = I*k + (60/8)*t(k)$, C_{new} is the total cost of a new method which is calculated using the estimated number of deterministic test patterns $DT estim$, and it is given as $C_{new} = I*k + (60/8)*DT estim$, C_{simple} is the total cost of the not detected faults, and it is calculated as $C_{simple} = I*k + (60/8)*Rnot$.

From table 3, we can able to see the cost difference between the exact method and the new method. By using the estimated deterministic patterns, the cost of the new method is reduced, notably with zero % error. The calculation time for the new method is also

reduced, which is the huge gain for the newly developed method. We will discuss the experimental results more in detail for other circuits in the next chapter.

4.6 Conclusion

The chapter discussed the development of a new algorithm. Firstly, the architecture of hybrid BIST and the advantage of introducing the stored test patterns and the increase in fault coverage are described.

The calculation of cost factors of hybrid BIST such as pseudorandom test cost, cost of stored test patterns, the total cost of hybrid BIST is explained with the example of the c880 circuit.

The two previously developed methods for the optimization of hybrid BIST are discussed. These methods resulted in the reduced test cost, but it is time-consuming for complex circuits. In order to reduce the time, a new algorithm was developed. The development of a new algorithm was explained with an example in this chapter.

5 Experimental research of the developed method

The chapter presents the experimental results of the newly developed method. It also presents the results of the exact method for comparison. The experimental results show a huge gain in the calculation time from the developed method for the complex circuits.

In the first section, ISCAS'85 benchmark circuits and how the Turbo Tester software used in this thesis are explained in detail with the example.

The second section presents the experimental results with the table representing the comparison of new hybrid BIST cost optimization with the exact method for the ISCAS'85 benchmark family, and it discusses the starting point to the Tabu search in detail.

5.1 Experimental environment

The experiments are performed for the ISCAS'85 benchmark circuits. These circuits were published at the International Symposium on Circuits & Systems 1985 to help in comparing the Automated Test Pattern Generation (ATPG) tools. It consists of 10 sets of combinational circuits, and these benchmark circuits have proven to be useful tools in different areas of digital design, including timing analysis, test generation, and technology mapping.

The experiments are carried out using Turbo Tester software. It can be operated in both windows and Linux operating systems. The Turbo tester system consists of tools for BIST emulation, fault simulation, test set optimization, Automated Test Pattern Generation (ATPG), and multi-valued simulation. Fault simulators and Test pattern generators are available for both sequential and combinational circuits. In addition to this, the package includes tools for design error localization and diagnosis [12]. For different tools, different syntaxes are used to get the result.

In this thesis, we used a deterministic Test pattern Generator, Random Test Generator, BIST emulator from the Turbo Tester system. For example, a random pattern generator for combinational circuits generates random patterns in packages of 32 vectors [12]. The following command and syntax should be used for a random pattern generation.

Command : random

Input : SSBDD model file (.agm)

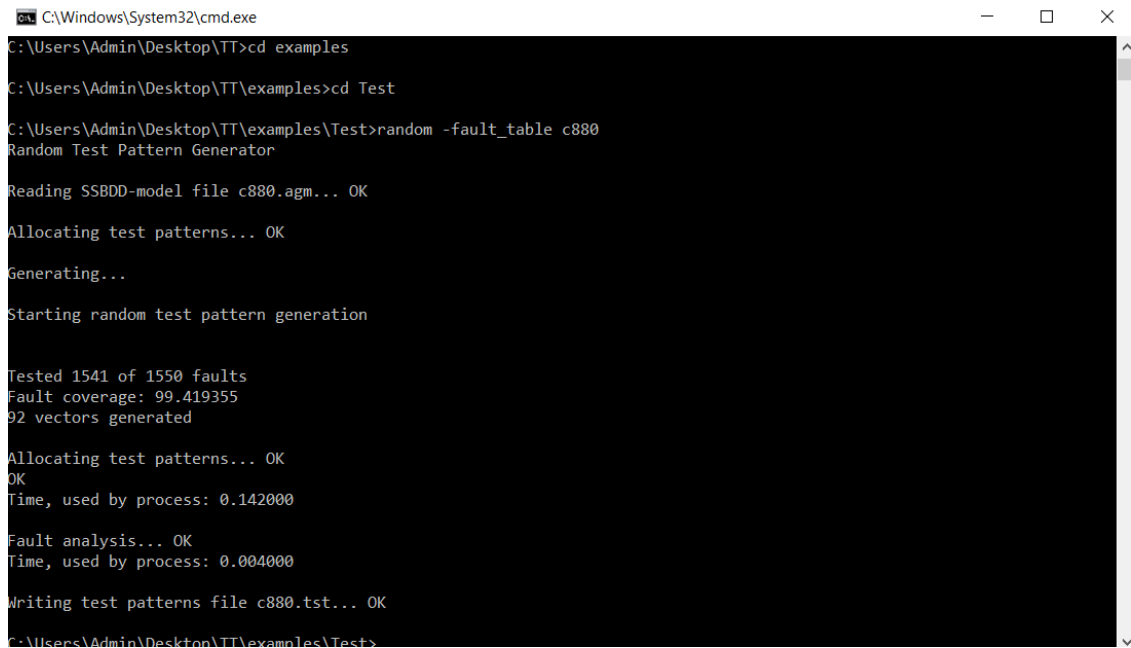
Output : test pattern file (.tst)

Syntax: random [*options*] <*design*>

Design : Name of the design file without .agm extension.

Options : -failure_limit, -pack_size, -criterion, -packages, -select_max, -fault_table, -infile.

The fault table for random pattern generation for the circuit c880 in the command line is given in figure 14 below. The syntax is given as *random -fault_table c880*.



```
C:\Windows\System32\cmd.exe
C:\Users\Admin\Desktop\TT>cd examples
C:\Users\Admin\Desktop\TT\examples>cd Test
C:\Users\Admin\Desktop\TT\examples\Test>random -fault_table c880
Random Test Pattern Generator
Reading SSBDD-model file c880.agm... OK
Allocating test patterns... OK
Generating...
Starting random test pattern generation

Tested 1541 of 1550 faults
Fault coverage: 99.419355
92 vectors generated

Allocating test patterns... OK
OK
Time, used by process: 0.142000

Fault analysis... OK
Time, used by process: 0.004000

Writing test patterns file c880.tst... OK
C:\Users\Admin\Desktop\TT\examples\Test>
```

Figure 14 Random pattern generation in Turbo Tester.

Figure 14, as a result, it shows the number of tested faults, fault coverage, number of vectors generated, and the time used by the process.

In a similar way, deterministic test pattern generation is carried out, and the syntax for deterministic test pattern generation is given as *generate -fault_table c432*. In order to increase the fault coverage backtrack option is used. For example, the syntax is given as *generate -fault_table -backtracks 1000 c432*. By increasing the backtracks, aborted faults, which are the faults that are not detected, are decreased. Simultaneously, it will increase the fault efficiency.

5.2 Experimental results

The goal of this experimental results is to show that the new method is scalable, and the results are depicted in Table 4, where it can be seen that the errors of the developed fast estimation method compared with the exact method is negligible but with the huge gain in the calculation time. This experiment is carried out for the ISCAS'85 benchmark circuits.

Table 4 Comparison of the new Hybrid BIST cost minimization method vs. exact method
for the benchmark family ISCAS'85 circuits

Circuit	Comparison of the methods: exact vs. prediction (new method)											
	BIST Cost C_{TOTAL}			Calculation time		Fault Coverage (%)	Test Efficiency (%)	Aborted faults	Pseudorandom test length		Deterministic test patterns	
	Exact	New	Error %	Exact method	New method				Exact	New	Exact	New
c3540	903	834	7.6	4.5h	25m50s	95.68	98.99	54	209	203	111	101
c2670	3044	2752	1.20	8.15h	29m11s	95.68	99.42	23	277	305	95	84
c7552	4186	3322	0	30.8h	3h	97.79	98.65	154	563	502	140	109
c880	311	229	0	0.558s	0.152s	100	100	0	101	101	28	17
c499	414	429	2.10	21m57s	2m15s	99.63	99.63	8	51	106	60	63
c1355	412	422	0	22m09s	2m21s	99.63	99.63	8	94	94	62	64
c6288	35	39	2.9	43m6s	4m3s	99.30	99.95	4	30	31	1	2
c5315	557	507	-3	10m82s	1m08s	99.29	100	0	201	304	48	49
c1908	588	596	-2.70	19m29s	1m93s	99.60	99.82	5	2	10	148	142
c432	197	161	14.87	12m22s	1m2s	95.27	100	0	102	102	21	13

Table 4 presents the experimental results for 10 ISCAS'85 benchmark circuits. The column BIST Cost C_{TOTAL} presents the cost of the exact and new method along with error accuracy, the column calculation time presents the calculation time of exact method and the new method, and it is clear that the new method resulted in a huge gain in time, fault coverage, test efficiency, aborted faults are also presented in the table for

each circuit. The pseudorandom test length and deterministic test patterns for both exact and new methods are also presented in table 4.

From the table, it is clear that the currently developed fast estimation method is more effective for complex circuits such as c3540, c2670, c7552. The time difference between the exact method and the new method is huge, and hence new method saves more time for all the circuits.

For the complex circuits, the exact method took more time nearly 30.8 hours with the bigger number of backtracks for c7552 circuit to achieve maximum fault coverage 98.65%, but with the new fast estimation method, it took nearly 30 minutes to achieve the same fault coverage with the negligible error compared to the exact method. To have a clear look, let us consider the circuit c3540 with three different backtracks and see how the error will increase if we save time.

Table 5 Comparison of the new Hybrid BIST cost minimization method vs. exact method for the circuit c3540 with different backtracks.

Number of backtracks	Comparison of the methods: exact vs. prediction (new method)							
	BIST Cost C_{TOTAL}			Calculation time		Fault Coverage (%)	Test Efficiency (%)	Aborted faults
	Exact	New	Error %	Exact method	New method			
2264	834	722	3.3	2m50s	0.5s	95.68	96.15	213
116703	878	722	4.7	22m18s	5.3s	95.68	98.19	98
52098184	903	834	7.6	4.5h	25m50s	95.68	98.99	54

In table 5, in each row, different backtracks are applied, and its corresponding results are tabulated. By increasing the backtracks, we can see that the test efficiency or fault efficiency increases as well as it removes the not detected faults (aborted faults). The BIST cost span between the fast calculated cost and the exact cost is 3.3 and 7.6, which is very small and negligible. Moreover, the time difference is the huge gain here because it took only 25 minutes in the newly developed method, whereas the exact method took nearly 4.5 hours. Hence, the new method is scalable for complex circuits with minimal time.

A second result of the thesis is to propose a starting point to the Tabu search as close as possible to the exact optimum. Tabu search was first introduced by Fred Glover [14], [15], [16] as a common iterative heuristic for solving optimization problems.

In [11], the Tabu search was applied to the similar task of hybrid BIST optimization using a starting point of search – the pseudorandom test length L calculated with the methodology developed in [2].

By experimental research with results in Table 6, it was shown that the method proposed in the thesis could be used for improving the location point of starting the Tabu search, by moving it closer to the exact optimum point as the target of the search.

Table 6 Starting point to the Tabu search

Circuits	Locations of the exact solution and the starting point candidates for tabu search			Distance from the exact solution		Gain in the distance of the proposed method
	Exact solution	Proposed method	Method [2]	Proposed method	Method [2]	
1	2	3	4	5	6	7
c3540	209	203	297	-6	88	82
c2670	277	305	444	28	167	139
c7552	563	502	583	-61	20	-41
c880	101	101	121	0	20	20
c499	51	106	78	55	27	-28
c1355	94	94	121	0	27	27
c6288	30	31	31	1	1	0
c5315	201	304	711	103	510	407
c1908	2	10	105	8	103	95
c432	102	102	91	0	-11	11

In Table 6, in column 2, the switching point from pseudorandom test session to deterministic test session is notated as the length of the pseudorandom test (measured in clock cycles). Columns 3 and 4 represent the estimated pseudorandom test lengths calculated by the proposed method and the method published in [2]. The entries in columns 2 and 3 are also shown in Table 4. In columns 5 and 6, the distances of the locations 3 and 4 from the exact solution to be found by Tabu search are presented, respectively. Column 7 shows the gain of the method proposed in the thesis, compared with the method in [2].

6 Conclusion

This thesis aimed to propose a new fast estimation method for the optimization of hybrid BIST by combining the pseudorandom test patterns and the deterministic test patterns to perform the test with a minimum cost of both the time and memory, without losing test quality.

Chapter two emphasized the importance of digital testing and the levels of abstraction in VLSI testing. In chapter three, an overview of BIST and its techniques was given. Chapter four covers in detail the hybrid BIST and the need for the new algorithm. Chapters five and six covers in detail the proposed approach, implementation, and experimental results, respectively.

1. A new method is proposed for hybrid BIST optimization, which outperforms the straightforward exact method about ten times in speed with average deviation from the exact optimum in average 0 – 3% and with the most important property of high scalability and hence well usable also for hybrid BIST of complex self-testing systems:
2. as the added value of the proposed method is providing a better starting point for the alternative optimization method of Tabu search for finding the exact optimum of the BIST solution, compared with the reference earlier method.

The experimental results have shown that the proposed approach is scalable and efficient for finding optimized solutions for hybrid BSIT architectures in SoC.

The results of this thesis are submitted as a research paper to the 30th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIA): Elmet Orasson, Jerome Angel John Rozario, Margus Kruus, Raimund Ubar in the topic “Interdisciplinary Research Lab for Project-Based Learning of Hardware and Software Design for Computer Engineering Students.”

References

- [1] E. J. Marinissen and Y. Zorian, "Challenges in Testing Core-Based System ICs," in *IEEE Communications Magazine*, June 1999.
- [2] G. Jervan, Z. Peng and R. Ubar, "Test cost minimization for hybrid BIST," in *Proceedings IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Yamanashi, Japan, Japan, 25-27 Oct. 2000.
- [3] L.-T. Wang, C.-W. Wu and X. Wen, *VLSI Test Principles And Architectures: Design For Testability*, Morgan Kaufmann Publishers, 2006.
- [4] M. L. Bushnell and V. D. Agarwal, *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*, USA: Kluwer Academic Publishers, 2002.
- [5] Y. Zorian, E. J. Marinissen and S. Dey, "Testing Embedded Core-Based System Chips," in *IEEE International Test Conference (ITC)*, Washington, DC, October 1998.
- [6] "The National Technology Roadmap for Semiconductors," San Jose, California: Semiconductor Industry Association, 1997, 1998.
- [7] R. Sedmak, "Design for Self-Verification: An Approach for Dealing with Testability Problems in VLSI-Based Designs," in *IEEE International Test Conference*, 1979.
- [8] C. E. Stroud, *A Designer's Guide to Built-In-Self-Test*, USA: KLUWER ACADEMIC PUBLISHERS, 2002.
- [9] McCluskey and E. J., "Built-In-Self-Test Techniques," in *IEEE*, 1985.
- [10] J. Kim and H. Shin, "Algorithm & SoC Design for Automotive Vision Systems: For Smart Safe Driving System," Springer Publishing Company, Incorporated, 2014.
- [11] H. Kruus, R. Ubar, G. Jervan, and Z. Peng, "Using Tabu Search Method for Optimizing the Cost of Hybrid BIST," in *Conference on Design of Circuits and Integrated Systems*, Porto, Portugal, Nov. 20-23, 2001.
- [12] "Turbo Tester Reference Manual. Version 3.99.03.," Tallinn Technical University, 1999.

- [13] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," in *ITC*, 1985, pp. 785-794.
- [14] E. Orasson, R. Raidma, R. Ubar, G. Jervan, and Z. Peng, "Fast Test Cost Calculation for Hybrid BIST in Digital Systems," Tallinn, Estonia.
- [15] G. Jervan, R. Ubar, and Z. Peng, "Test Cost Minimization for Hybrid BIST," in *IEEE International Symp. on Defect and Fault Tolerance in VLSI Systems.*, Tokyo, October 25-28, 2000, pp.283-291.
- [16] R. Ubar, G. Jervan, Z. Peng, E. Orasson, and R. Raidma, "Fast Test Cost Calculation for Hybrid BIST in Digital Systems," in *EUROMICRO Symposium on Digital Systems Design*, Warsaw, September 4-6, 2001, pp.318-325.
- [17] H. Kruus, G. Jervan, and R. Ubar, "Using Tabu search for optimization of memory-constrained Hybrid BIST," in *11th IEEE Biennial Baltic Electronics Conference*, Tallinn, Estonia, October 6-8, 2008, pp. 155 - 158.
- [18] R. Ubar and H. D. Wuttke, "Research and Training Scenarios for Design and Test of SoC," in *Proc. of the World Congress on Engineering and Technology Education*, Guaruja/Santos, Brasil, pp.320-324, March 14-17, 2004.
- [19] H. Kruus, R. Ubar, and J. Raik, "Defect-Oriented BIST Quality Analysis," in *Baltic Electronics Conference*, Oct 4-6, 2010, pp. 153-156.
- [20] S.Kostin, R.Ubar, G.Mägi, and M.Gorev, "Comparison of two approaches to improve functional BIST fault coverage," in *Proceedings of Baltic Electronics Conference*, Tallinn, October 6-8, 2014, pp.1-4.
- [21] M. Sugihara, H. Date, and H. Yasuura, "Analysis and Minimization of Test Time in a Combined BIST and External Test Approach," in *Design, Automation & Test in Europe Conference*, Paris, France, March 2000.
- [22] F. Glover and M. Laguna, "TABU search," kluwer, MA, 1997.
- [23] F. Glover, "Tabu Search - Part 1," 1(3): 190-206, 1989, ORSA Journal on Computing.
- [24] F. Glover, E. Thailand, and D. d. Werra., "A user's guide to tabu search," in *Annals of Operations Research*, 41: 3-28, 1993.

Appendix 1 – Program Description and Manual

This section describes how the experiments were performed.

1. A bash Unix shell script was written for the exact method for the ISCAS'85 benchmark circuits.
2. The folders and files for the experiments can be downloaded through this link: <https://drive.google.com/open?id=1qqdAYSgyEnABzkEIfvDPhOF0k7oEpj1v>
3. Linux terminal is recommended to perform the experiment. Each of the folders from the *exact_method_experiments* folder contains a test pattern file, deterministic test pattern report, pseudorandom test pattern report of the respective circuit.
4. The folder name with *larger_backtracks* contains the report file for the complex circuits c3540, c432, c2670, c7552. The experiments were performed on these circuits to get the maximum fault coverage.
5. *getting_max_fault_coverage.py* is the python code used to experiment with the complex circuits with larger backtracks because circuits like c7552 took nearly 31 hours to get the maximum fault coverage.
6. *time_minimization.py* is the python code for a new method, and in.txt file takes all the pseudorandom and deterministic report files for the circuits.
7. The folder *results* contain the output results of the new method, which are in the Excel file format for each circuit. The excel file consists of three columns: pseudorandom test vectors, deterministic test vectors, and the third column is the estimated deterministic number of patterns $C_D(L)$.
8. The file *Experiments.xlsx* contains the experimental results of the ISCAS'85 benchmark circuits. It contains nine columns where the column k is the pseudorandom test vectors, the column $t(k)$ is the deterministic test vectors; the column Det.time is the time taken to generate deterministic test patterns, the column *Rnot* is the number of not detected faults, the column *DT estim* is the estimated deterministic number of patterns, the column C_{exact} is the total test cost

for the exact method, the column C_{new} is the total test cost for the new method, the column C_{simple} is the total cost for not yet detected faults, $C_{total}(k)$ is the cost calculation for tabu search.

9. The file *Experiments_larger_backtracks.xlsx* contains the experimental results for the complex circuits with larger backtracks.
10. The file *c3540_comparison.xlsx* shows all three cases with different backtracks.

Appendix 2 – Source

```
import sys, re, time
files = open("C:\\Users\\Admin\\PycharmProjects\\untitled\\in.txt", "r")
for file in files:
    start_time = time.time()
    splitted_file_list = file.split()
    dt_file = open(splitted_file_list[0], "r")
    pt_file = open(splitted_file_list[1], "r")
    output_file = open(splitted_file_list[0].split("\\")[-1]+".csv", "w")
    # pt_file = open(sys.argv[2], "r")
    dt_list = []
    r = r"-?d+\\.?.?d*"
    for dt_line in dt_file:
        number_list = re.findall("\\d+\\.?.?d*", dt_line)
        if len(number_list) != 0:
            dt_list.append([number_list[0], number_list[1]])

    # print("=====")
    pt_list = []
    for pt_line in pt_file:
        number_list = re.findall("\\d+\\.?.?d*", pt_line)
        if len(number_list) != 0:
            pt_list.append([number_list[0], number_list[1]])
x = 0
```



```

last_dt = dt_list[len(dt_list) - 1][0]
for pt in pt_list:
    min_number = 100000000
    dt_row = []
    for dt in dt_list:
        if abs(float(dt[1]) - float(pt[1])) < min_number:
            min_number = abs(float(dt[1]) - float(pt[1]))
            dt_row = dt
    # print(pt[1] + " " + dt_row[1] + " " + str(float(last_dt) -
float(dt_row[0])))
    output_file.write(pt[0] + "," + dt_row[0] + "," + str(float(last_dt)
- float(dt_row[0])) + ",\n")
    end_time = time.time()
    print((end_time-start_time))

```