EEE -160002

# Comparative Study on Space Compaction by Using Minimal Logic Gate in a Graph Theoretical Approach

A Dissertation Submitted to The Department of Electrical and Electronics Engineering of Sonargaon University in Patiala Fulfillment of the Requirements for the Degree of Bachelor of Science in Engineering.

## SONARGAON UNIVERSITY

**Submitted To**


**Md. Fairuz Siddiquee**

Lecturer EEE
Sonargaon University


**Submitted By**


**Mohammad Masum Sarker**
ID No: EEE 1403003117

**Md. Shariful Islam**
ID No: EEE 1403003077

**Md. Ashiqur Rahman**
ID No: EEE 1403003074

Department of Electrical & Electronic Engineering
Sonargaon University
Dhaka-1208, Bangladesh



Date of Submit: 05.02.2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate, in scope and quality as a dissertation for the degree of BSC in Engineering.

Md. Fairuz Siddiquee

Lecturer EEE
Sonargaon University

# *Dedication*

*I dedicate this dissertation to*

***My beloved parents***

# *Preface*

Despite the fact that new design automation tools have allowed designers to work on higher abstraction levels, test-related activities are still mainly performed at the lower levels of abstraction. At the same time, testing is quickly becoming one of the most time and resource consuming tasks of the electronic system development and production cycle. Therefore, traditional gate-level methods are not any more practical nowadays and test activities should be migrated to the higher levels of abstraction as well. It is also very important that all design tasks can be performed with careful consideration of the overall testability of the resulting system. The main objective of this thesis work has been to investigate possibilities to support reasoning about system testability in the early phases of the design cycle (behavioral and system levels) and to provide methods for systematic design modifications from a stability perspective. Our research is carried out in close cooperation with both the industry. We would like to mention here the very fruitful cooperation with the groups at Sonargaon University Research Center. Our work has also been regularly presented and discussed in The Sonargaon University thesis coordinator. This cooperation has opened new horizons and produced several results, some of which are presented in this thesis.

# *Acknowledgments*

I would like to sincerely thank my supervisor Md. Fairuz Siddiquee for all support in the work toward this thesis. Md. Fairuz Siddiquee has always given me excellent guidance and I have learned a lot from him. He has also given me the opportunity and support to work with problems not directly related to the thesis, but very relevant for understanding the research organization and administrative processes. Who has always been an excellent generator of new ideas and enriched our regular meetings with very useful remarks. The colleagues at IDA have provided a nice working environment. They have through the past few years grown to be more than colleagues but good friends. I also thanks to Rajib Baran Roy, Assistant Professor & Department Head of EEE for support and encouragement as well as the wonderful atmosphere. He give me good guideline & mentally support for complete my thesis. Many thanks also to Professor Abdur Razzak, Vice Chancellor and Abdul Kalam, Assistant Professor & Department Head of Business Administration of Sonargaon University, who is responsible for bringing me to the wonderful world of science. The continuing cooperation with him for thesis.

Finally I would like to thank my parents, my brother & sister, my nephew and all my friends. You have always been there, whenever I have needed it.

**Mohammad Masum Sarker**

ID No: EEE1403003117
Department of Electrical & Electronic Engineering
Sonargaon University

**Md. Shariful Islam**

ID No: EEE1403003077
Department of Electrical & Electronic Engineering
Sonargaon University

**Md. Ashiqur Rahman**

ID No: EEE1403003074
Department of Electrical & Electronic Engineering
Sonargaon University

# *Abstract*

We try to improve space compaction one stage or multistage use logic gate implementation through graph Theory. We implement Brute Force & Brone Kerbosch theory at this thesis for improve space compaction. The technological development is enabling production of increasingly complex electronic systems. All those systems must be verified and tested to guarantee correct behavior. The established low-level methods for space compaction are not any more sufficient and more work has to be done at abstraction levels higher than the logic gate. This thesis reports on one such work that space compaction techniques. The contribution of this thesis is twofold. First, we investigate the possibilities of space compaction. We have try to developed space compaction for this purpose Brute Force & Brone Kerbosch algorithm. The second part of the thesis concentrates on efficiency & compaction time at space compaction & compeer at another method. We investigate which method is perfect or sufficient at space compaction. We have also developed methods for space compaction implementation graph theory and efficiency of the proposed technique.

This thesis presents a new technique for merging output test vectors and compares different types of compaction methods. The proposed technique takes advantage of some well- known concepts of conventional switching theory, together with the selection of specific gates for merger of an arbitrary but optimal number of output bit streams from the circuit under test.

This is a new technique as it is implemented without any modification of the original circuit. But the maximum compaction is achieved in almost all cases within a reasonable time span. The proposed technique is illustrated with design details of space compactors for ISCAS (International Symposium on Circuits and Systems) 85 combinational benchmark circuits using simulation programs ATALANTA. The simulation result confirms the usefulness of the approach for its simplicity, resulting low area overhead, and full fault coverage for single stuck-line faults, thereby making it suitable in a VLSI design environment.

# *Contents*

# *List of Figures*

# *List of Tables*

# *Chapter 1*
# *Introduction*

Design for Test (also known as "Design for Testability" or "DFT") is a name for design techniques that add certain testability features to a microelectronic hardware product design. The purpose of manufacturing tests is to validate that the product hardware contains no defects that could, otherwise, adversely affect the product's correct functioning.

Tests are applied at several steps in the hardware manufacturing flow and, for certain products, may also be used for hardware maintenance in the hardware environment. The tests generally are driven by test programs that execute in Automatic Test Equipment (ATE) or, in the case of system maintenance, inside the assembled system itself. In addition to finding and indicating the presence of defects (i.e., the test fails), tests may be able to log diagnostic information about the nature of the encountered test fails. The diagnostic information can be used to locate the source of the failure.

In other words, the response of vectors (patterns) from a good circuit is compared with the response of vectors (using same patterns) from a DUT (device under test). If the response is the same or matches, the circuit is good. Otherwise, the circuit is faulty.

DFT plays an important role in the development of test programs and as an interface for test application and diagnostics. Automatic test pattern generation, or ATPG, is much easier if appropriate DFT rules and suggestions have been implemented.

Automatic or Automated Test Equipment (ATE) is any apparatus that performs tests on a device, known as the Device Under Test (DUT) or Unit Under Test (UUT), using automation to quickly perform measurements and evaluate the test results. An ATE can be a simple computer controlled digital multi-meter, or a complicated system containing dozens of complex test instruments (real or simulated electronic test equipment) capable of automatically testing and diagnosing faults in sophisticated electronic packaged parts or on Wafer testing, including System-On-Chips and Integrated circuits.

Space compaction of test response provides parallel access to functional output and reduces testing time and test Data volume. The realization of space-efficient support hardware for built-in self-testing (BIST) is of great significance in VLSI circuits design. Testing in its broadest sense means to examine a production and to ensure that it functions and exhibits the properties and capabilities that it was designed to possess. Main purpose of testing is to detect malfunctions in the product hardware and to locate their causes so that they may be eliminated.

A built-in self-test (BIST) or built-in test (BIT) is a mechanism that permits a machine to test itself. Engineers design BISTs to meet requirements such as:

1. High reliability
2. Lower repair cycle times

Constraints are:

1. Limited technician accessibility
2. Cost of testing during manufacture

The main purpose of BIST is to reduce the complexity, and thereby decrease the cost and reduce reliant upon external (pattern-programmed) test equipment. BIST reduces cost in two ways:

1. Reduces test-cycle duration
2. Reduces the complexity of the test/probe setup, by reducing the number of I/O signals that must be driven /examined under tester control. Both lead to a reduction in hourly charges for automated test equipment (ATE) service. Space compaction of test response provides parallel access to functional output and reduces testing time and test data volume. The realization of space-efficient support hardware for built-in self-testing (BIST) is of great significance in VLSI circuits design.

A device under test (DUT) is a device that is tested to determine performance and proficiency. A DUT also may be a component of a bigger module or unit known as a unit under test (UUT). A DUT is checked for defects to make sure the device is working. The testing is designed to prevent damaged devices from entering the market, which also may reduce manufacturing costs.

A DUT is usually tested by automatic or automated test equipment (ATE), which may be used to conduct simple or complex testing, depending on the device tested. ATEs may include testing performed on software, hardware, electronics, semiconductors or avionics.

Automatic test equipment (ATE) is a machine that is designed to perform tests on different devices referred to as a devices under test (DUT). An ATE uses control systems and automated information technology to rapidly perform tests that measure and evaluate a DUT.

ATE tests can be both simple and complex depending on the equipment tested. ATE testing is used in wireless communication and radar as well as electronic component manufacturing. There is also specialized semiconductor ATE for testing semiconductor devices.

Automated test equipment is a computer-operated machine used to test devices for performance and capabilities. A device that is being tested is known as device under test (DUT). ATE can include testing for electronics, hardware, software, semiconductors or avionics.

There are uncomplicated ATEs such as volt-ohm meters that measure resistance and voltage in PCs. There are also complex ATE systems that have several test mechanisms that automatically run high-level electronic diagnostics such as wafer testing for semiconductor device fabrication or for integrated circuits. Most high-tech ATE systems use automation to perform the test quickly.

The objective of ATE is to quickly confirm whether a DUT works and to find defects. This testing method saves on manufacturing costs and helps prevent a faulty device from entering the market. Because ATE is used in a wide array of DUTs, each testing has a different procedure. One actuality in all testing is that when the first out-of-tolerance value is detected, the testing stops and the DUT fails the evaluation.

Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

Hybrid BIST for sequential circuits. In this thesis we have proposed a hybrid BIST approach for combinational circuits. A more complex problem is to propose an architecture and optimization mechanisms for sequential circuits. The difficulty of developing such architecture and mechanisms is not only due to the complex nature of sequential circuits, but also related to pseudorandom testability. In case of combinatorial circuits, pseudorandom patterns have relatively high fault detection capabilities. This is not valid for sequential circuits and alternative methods for reducing the test data amount has to be developed. One of the possibilities is to apply pseudorandom patterns only for a combinatorial section of the design while the rest of the design is tested with deterministic patterns.

Self-test methods for other fault models. Most of the existing work in the area of BIST is targeting the classical SSA fault model. At the same time it has been demonstrated that the SSA fault model can only cover some failure modes in CMOS technology. Thus, the importance of other fault models (like transition and path delay) is increasing rapidly. Therefore, it would be very interesting to analyze the quality of hybrid test set in terms of defect detection capabilities and to develop a methodology to support the detection of other failures than the stuck-at ones.

Automatic Test Pattern Generation (ATPG) has several purposes:

01. It can generate test patterns.
02. It can find redundant circuit logic.
03. It can prove one implementation matches another.

# *History*

During the mid-1920s, several inventors attempted devices that were intended to control current in solid-state diodes and convert them into triodes. Success did not come until after WWII, during which the attempt to improve silicon and germanium crystals for use as radar detectors led to improvements in fabrication and in the understanding of quantum mechanical states of carriers in semiconductors. Then scientists who had been diverted to radar development returned to solid-state device development. With the invention of transistors at Bell Labs in 1947, the field of electronics shifted from vacuum tubes to solid-state devices.

With the small transistor at their hands, electrical engineers of the 1950s saw the possibilities of constructing far more advanced circuits. As the complexity of circuits grew, problems arose.

One problem was the size of the circuit. A complex circuit, like a computer, was dependent on speed. If the components of the computer were too large or the wires interconnecting them too long, the electric signals couldn't travel fast enough through the circuit, thus making the computer too slow to be effective.

Jack Kilby at Texas Instruments found a solution to this problem in 1958. Kilby's idea was to make all the components and the chip out of the same block (monolith) of semiconductor material. Kilby presented his idea to his superiors, and was allowed to build a test version of his circuit. In September 1958, he had his first integrated circuit ready. Although the first integrated circuit was crude and had some problems, the idea was groundbreaking. By making all the parts out of the same block of material and adding the metal needed to connect them as a layer on top of it, there was no need for discrete components. No more wires and components had to be assembled manually. The circuits could be made smaller, and the manufacturing process could be automated. From here, the idea of integrating all components on a single silicon wafer came into existence, which led to development in small-scale integration (SSI) in the early 1960s, medium-scale integration (MSI) in the late 1960s, and then large-scale integration (LSI) as well as VLSI in the 1970s and 1980s, with tens of thousands of transistors on a single chip (later hundreds of thousands, then millions, and now billions (109)).

# *Motivation*

Hardware testing is a process to check whether a manufactured integrated circuit is error-free. As the produced circuits may contain different types of errors or defects that are very complex, we have to define a model to represent these defects to ease the test generation and test quality analysis problems. This is usually done at the logic level. Test patterns are then generated based on a defined fault model and applied to the manufactured circuitry. It has been proven mathematically that the generation of test patterns is an NP- complete problem and therefore different heuristics are usually used. Most of the existing hardware testing techniques work at the abstraction levels where information about the final implementation architecture is already available. Due to the growth of systems complexity these established low-level methods are not any more sufficient and more work has to be done at abstraction levels higher than the classical gate and register-transfer level (RT-level) in order to ensure that the final design is testable and the time-to-market schedule is followed. More and more frequently designers also introduce special structures, called design for testability (DFT) structures, during the design phase of a digital system for improving its testability. Several such approaches have been standardized and widely accepted. However, all those approaches entail an overhead in terms of additional silicon area and performance degradation. Therefore it will be highly beneficial to develop DFT solutions that not only are efficient in terms of testability but also require minimal amount of overhead. Most of the DFT techniques require external test equipment for test application. BIST technique, on the other hand, implements all test resources inside the chip. This technique does not suffer from the bandwidth limitations which exist for external testers and allows to apply at-speed tests. The disadvantage of this approach is that it cannot guarantee sufficiently high fault coverage and may lead to very long test sequences. Therefore a hybrid BIST approach that is implemented on-chip and can guarantee high fault coverage can be very profitable when testing modern systems-on-chip (SoC).

# *Chapter 2*
# *TESTING*

## *Hierarchical Test Generation*

The main idea of the hierarchical test generation (HTG) technique is to use information from different abstraction levels while generating tests. One of the main principles is to use a modular design style, which allows to divide a larger problem into several smaller problems and to solve them separately. This approach allows generating test vectors for the lower level modules based on different techniques suitable for the respective entities. In hierarchical testing, two different strategies are known: top- down and bottom-up. In the bottom-up approach, tests generated at the lower level will be assembled at the higher abstraction level. The top-down strategy, introduced in, uses information,

Generated at the higher level, to derive tests for the lower level. Previously mentioned as well as more recent approaches have been successfully used for hardware test generation at the gate, logical and register-transfer (RT) levels. In this thesis, the input to the HTG is a behavioral description of the design and a technology dependent, gate level library of functional units. Figure 2.1 shows an example of such a hierarchical representation of a digital design. It demonstrates a behavioral specification, a fragment of a corresponding behavioral level decision diagram and a gate level net list of one of the functional units.

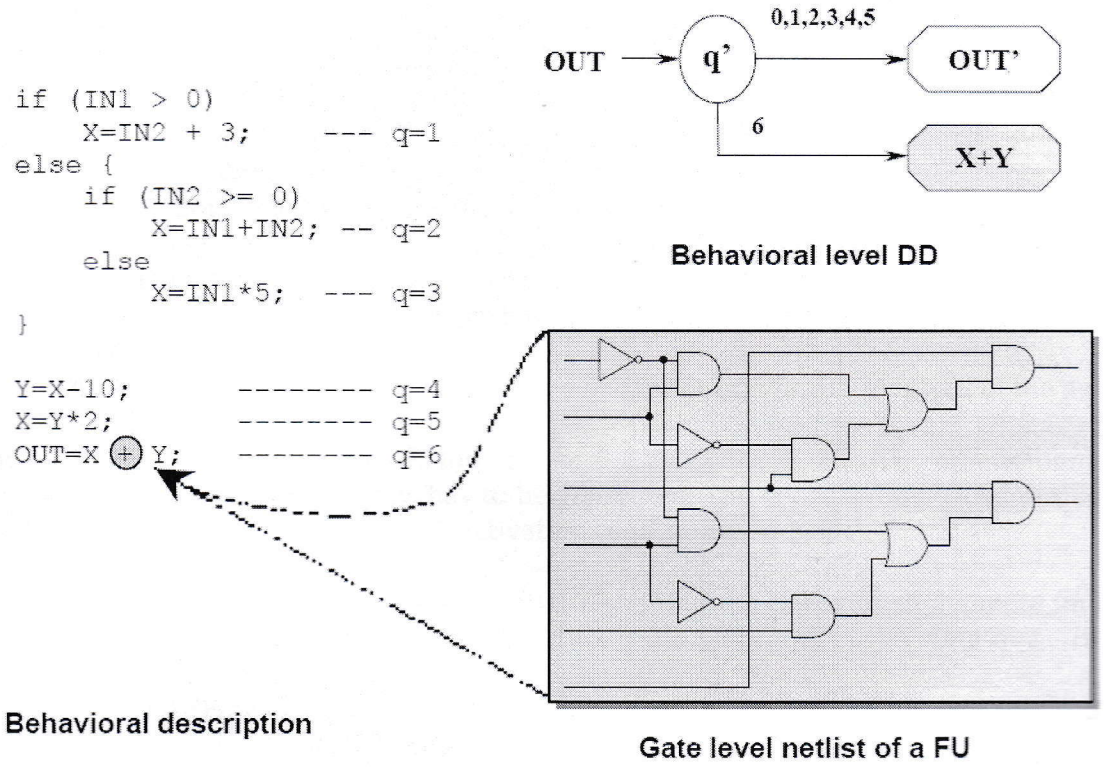


Figure 2.1 Hierarchical representation of a digital design

Our high-level hierarchical test generation approach starts from a behavioral specification, given in VHDL. At this level the design does not include any details about the final implementation, however we assume that a simple finite-state machine (FSM) has already been introduced and therefore the design is conceptually partitioned into the data path and control part. For this transformation we are using the CAMAD high-level synthesis system.

DD synthesis from a high-level description language consists of several steps, where data path and control part of the design will be converted into the DDs separately. In the following, an overview of the DD synthesis process, starting from a VHDL description, will be given.

In the general case, a DD is a directed, acyclic graph where non- terminal nodes represent logical conditions, terminal nodes represent operations, while branches hold the subset of condition values for which the successor node corresponding to the branch will be chosen. The variables in nonterminal nodes can be either Boolean (describing flags, logical conditions etc.) or integer (describing instruction words, control fields, etc.) The terminal nodes are labeled by constants, variables (Boolean or integer) or by expressions for calculating integer values.

At the behavioral level, for every internal variable and primary output of the design a data-flow DD will be generated. Such a data- flow DD has so many branches, as many times the variable appears on the left-hand side of the assignment. Further, an additional DD, which describes the control-flow, has to be generated. The control- flow DD describes the succession of statements and branch activation conditions.

Figure 2.2 depicts an example of DD, describing the behavior of a simple function. For example, variable A will be equal to IN1+2, if the system is in the state q=2 (Figure 2.2c). If this state is to be activated, condition IN1≥ *0*
Should be true (Figure 2.2b). The DDs, extracted from a specification, will be used as a computational model in the HTG environment.

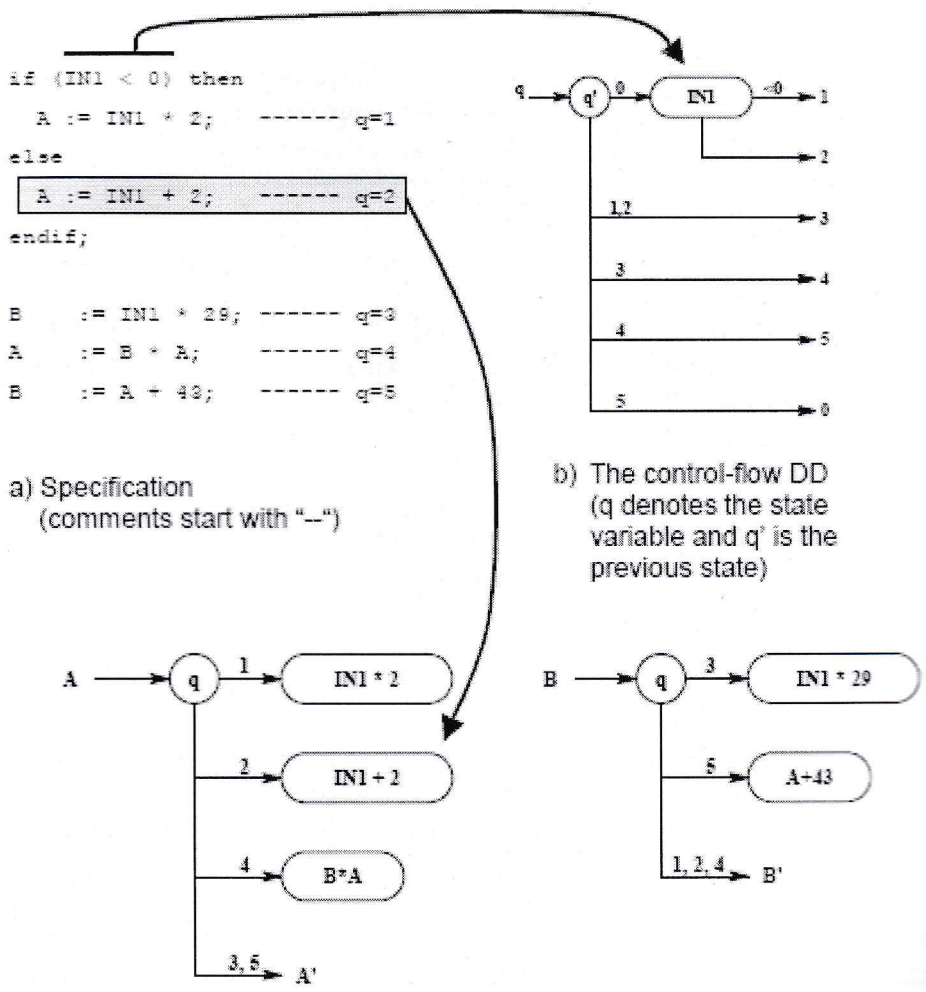


Figure 2.2 A decision diagram example.

SICStus Prolog representation of Decision Diagrams As described earlier, at the behavioral level there exist two types of DDs: control-flow DD and data-flow DDs. The control-flow DD carries two types of information: state transition information and path activation information. The state transition information captures the state transitions that are given in the FSM corresponding to the specified system. The path activation information holds conditions associated to state transitions.

For each internal or primary output variable corresponds one data- flow DD. In a certain system state, the value of a variable is determined by the terminal node in the data graph. In this case, the relationship between the terminal node and the variable can be viewed as a functional constraint on the variable at the state.

To generate a test pattern for a fault we have to excite the fault (justification) and to sensitize the fault effect at the primary outputs (propagation). For example, if we want to test the statement that is highlighted in Figure 2.2a, we have to bring the system to the state $q=2$.

This can be guaranteed only when $q'=0$ and $IN1 \geq 0$

Those requirements can be seen as justification constraints. For observing the fault effect at primary outputs, we have to distinguish between the faulty and the correct behavior of a variable under test (Variable "A" in our example).

This requires, that B $\neq 0$ (from the statement *A: =B*A*) and consequently IN1*29≠ *0* (from the Statement *B: =IN1*29*), otherwise the variable "A" will have always value 0 and the fault cannot be detected. Those conditions can be seen as propagation constraints.

By solving the extracted constraints we will have a test pattern (combination of input values) which can excite the fault and propagate the fault effect to the primary outputs. For solving these constraints we employ a commercial constraint solver SICStus and have developed a framework for representing a DD model in the form of constraints. First, we translate the control-flow DD into a set of state transition predicates and path activation constraints are extracted along the activated path. Then all the data-flow DDs are

Parsed as functional constraints at different states by using predicates. Finally, a DD model is represented as a single Prolog module. See for technical details about the translation process.

# *Hierarchical Test Generation Algorithm*

This section presents our high-level hierarchical test generation algorithm. At first we introduce fault models used in our approach. Thereafter the corresponding tests are discussed and finally the whole test generation environment is presented.

The test generation task is performed in the following way (Figure 2.3). Tests are generated sequentially for each nonterminal node of the control-flow DD. Symbolic path activation is performed and functional constraints are extracted. Solving the constraints gives us the path activation conditions to reach a particular segment of the specification. In order to test the operations, presented in the terminal nodes of the data-flow DD, different approaches can be used. In our approach we employ a gate level test pattern generator. In this way we can incorporate accurate structural information into the high-level test pattern generation environment while keeping the propagation and justification task still on a high abstraction level.

If the constraint solver is not able to find a solution, a new test case should be generated, if possible. This cycle should be continued until a solution is found or a timeout occurs.

In the following, the test pattern generation algorithm is described in more detail.

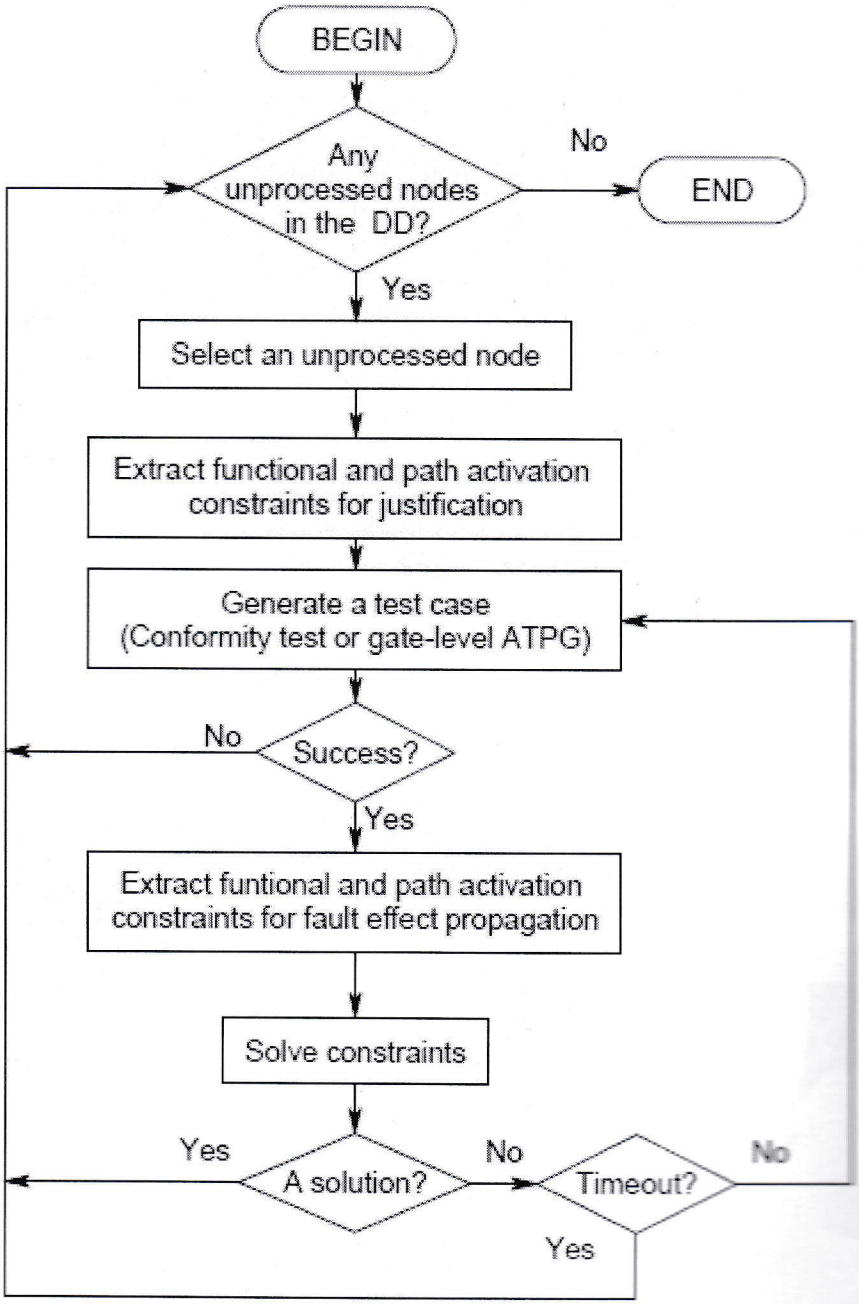


Figure 2.3: The general flow for hierarchical test generation algorithm.

For the nonterminal nodes of the control-flow DD, conformity tests will be applied. The conformity tests target errors in branch activation. For example, in order to test nonterminal node IN1 (Figure 2.4), one of the output branches of this node should be activated. Activation of the output branch means activation of a certain set of program statements. In our example, activation of the branch IN1<0 will activate the branches in the data-flow DD where *q=1 (A: =X)*. For observability the values of the variables calculated in all the other branches of IN1 have to be distinguished from the value of the variables calculated by the activated branch. In our example, Node IN1 is tested, in the case of IN1<0, if $X \neq Y$. The path from the root node of the control-flow DD to the node IN1 has to be activated to ensure the execution of this particular specification segment and the conditions generated here should be justified to the primary inputs of the module. This process will be repeated for each output branch of the node. In the general case there will be n(n-1) tests, for every node, where n is the number of output branches.

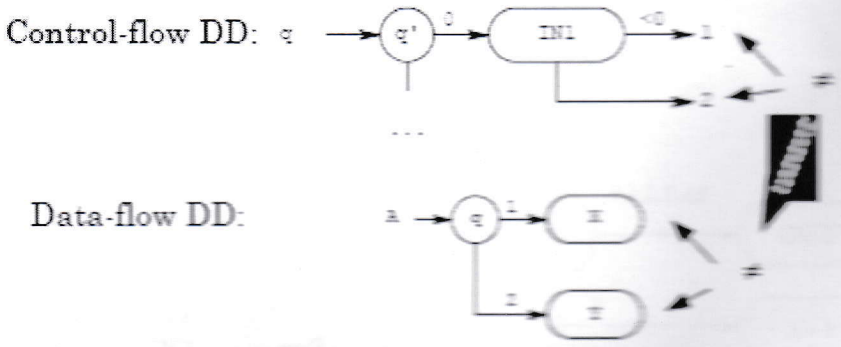


Figure 2.4 Conformity test

Synthesis is the translation of a behavioral representation of a design into a structural one. One of the most important parameters guiding the synthesis process is the technology that will be used in the final implementation. After the technology is defined, the implementation details of the functional units (FUs) that will be used in the final design can be found usually in the technology library. Our hierarchical test generation algorithm employs this structural
Information for generating tests and estimating the testability of the final implementation when using one or another implementation of the FU from the same or even from completely different libraries. This reveals another advantage of our test pattern generation algorithm: we can derive information about the testability of a system, depending on what target technology it will be implemented in. We can generate tests for different possible implementations (different implementations of the same FU) and to select the solution that is the best from the testability point of view. This information can be used later in the synthesis while performing
Allocation and mapping.

Tests are generated in cooperation with low-level test pattern generators. The functional unit test generation is performed one by one for every FU given in the specification as depicted in Figure 2.5, where an example of generating low-level tests for an adder is given.

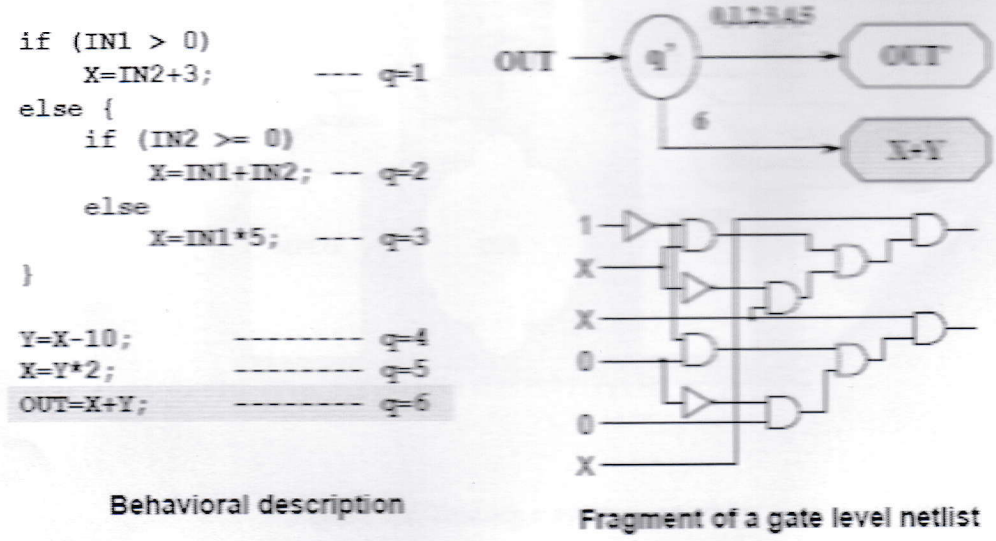


Figure 2.5 Testing functional units.

To test the individual cores on SoC, the test pattern source and sink have to be available together with an appropriate test access mechanism (TAM), as depicted in Figure 2.6. We can implement such a test architecture in several different ways. One widespread approach is to implement both source and sink off-chip and require therefore the use of external Automatic Test Equipment (ATE). But, as discussed earlier, the internal speed of SoC is constantly increasing and, thus, the demands for the ATE speed and memory size are continuously increasing too. However, the technology used in ATE is always one step behind the one used for advanced SoCs and, the ATE solution will soon become unacceptably expensive and inaccurate. Therefore, in order to apply at-speed tests and to keep the test costs under control, on-chip self-test solutions are becoming more and more popular.

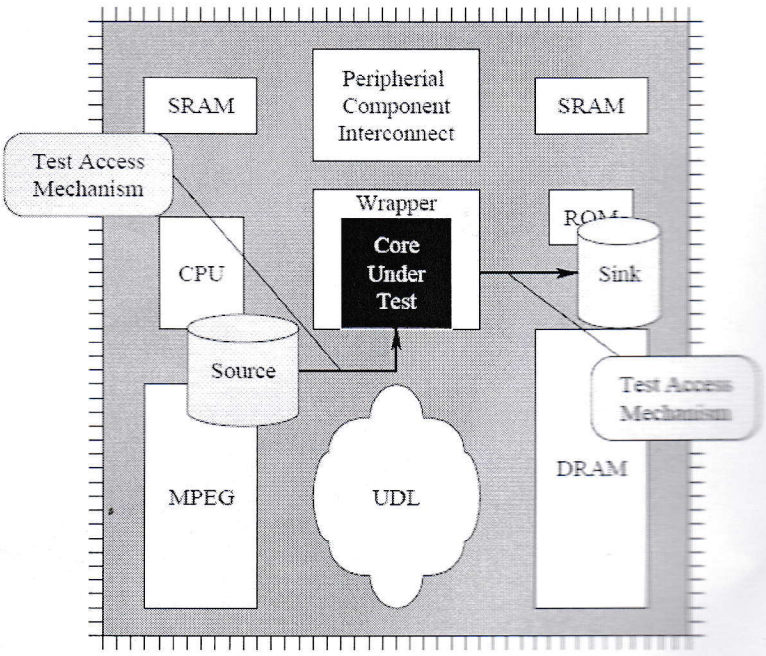


Figure 2.6 Testing a system-on-chip.

A typical BIST architecture consists of a test pattern generator (TPG), a test response analyzer (TRA) and a BIST control unit (BCU), all implemented on the chip. This approach allows applying at-speed tests and eliminates the need for an external tester. Different BIST approaches have been available for a while and have got wide acceptance especially for memory test. For logic BIST (LBIST) there is still no industry-wide acceptance. One of the main reasons is the hardware overhead required to implement a BIST architecture. The BIST approach can also introduce additional delay to the circuitry and requires a relatively long test application time. At the same time, BIST is basically the only practical solution to perform at-speed test and can be used not only for manufacturing test but also for periodic field maintenance tests.

## *Hybrid BIST Architecture*

A hardware-based hybrid BIST architecture is depicted in Figure 2.7a, where the pseudorandom pattern generator (PRPG) and the Multiple Input Signature Analyzer (MISR) are implemented inside the core under test (CUT). The PRPG and MISR can be implemented by using LFSRs or any other structure able to provide pseudorandom test vectors with a required degree of randomness. The deterministic test patterns are recomputed off-line and stored inside the system.

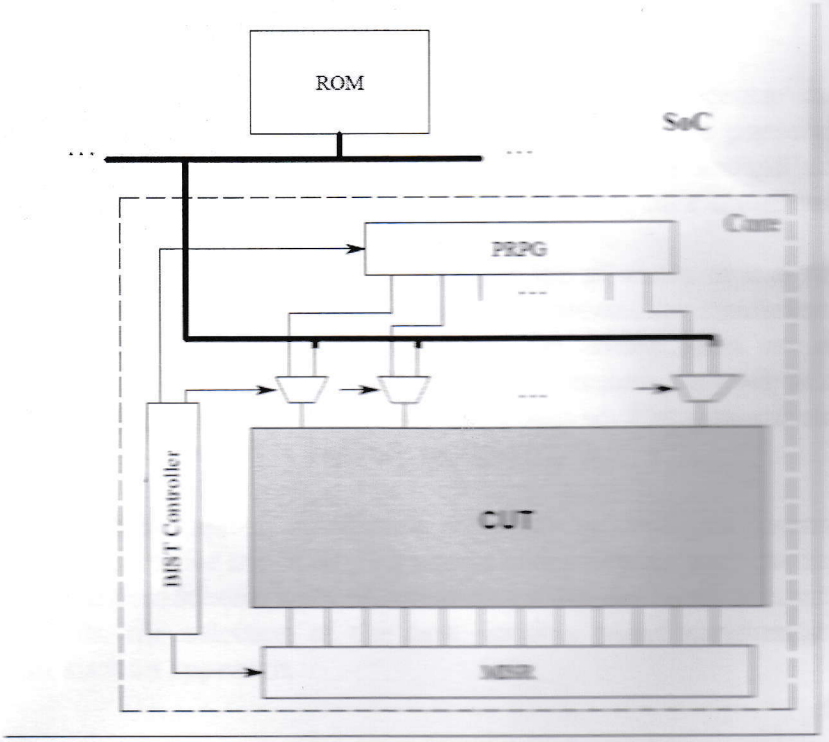


Figure 2.7a Hardware-based hybrid BIST architecture.

Core test is performed in two consecutive stages. During the first stage pseudorandom test patterns are generated and applied. After a predetermined number of test cycles, additional test is performed with deterministic test patterns from the memory. Each primary input of CUT has a MUX at the input that determines whether the test is coming from the PRPG or from the memory (Figure 2.7a).

To avoid the hardware overhead caused by the PRPG and MISR, and the performance degradation due to excessively large LFSRs, a software-based hybrid BIST can be used where pseudorandom test patterns are produced by the test software. However, the cost Calculation and optimization algorithms to be proposed are general, and can be applied to the hardware-based as well as to the software- based hybrid BIST optimization.

In case of a software-based solution, the test program, together with all necessary test data (LFSR polynomials, initial states, pseudorandom test length, and signatures) are kept in a ROM. The deterministic test vectors are generated during the development process and are stored in the same place. For transporting the test patterns, we assume that some form of TAM is available.

In test mode the test program will be executed by the processor core. The test program proceeds in two successive stages. In the first stage the pseudorandom test pattern generator, which emulates the LFSR, is executed. In the second stage the test program will apply recomputed deterministic test vectors to the core under test.

The pseudorandom TPG software is the same for all cores in the system and is stored as one single copy. All characteristics of the LFSR needed for emulation are specific to each core and are stored in the ROM. They will be loaded upon request. Such an approach is very effective in the case of multiple cores, because for each additional core only the BIST characteristics for this core have to be stored. The general concept of the software based pseudorandom TPG is depicted in Figure 2.7b.

As the LFSR is implemented in software, there are no hardware constraints for the actual implementation except for the ROM. This allows to develop for each particular core the most efficient pseudorandom scheme without concerning about the hardware cost. As has been shown by experiments, the selection of the best possible pseudorandom scheme is an important factor for such an approach.

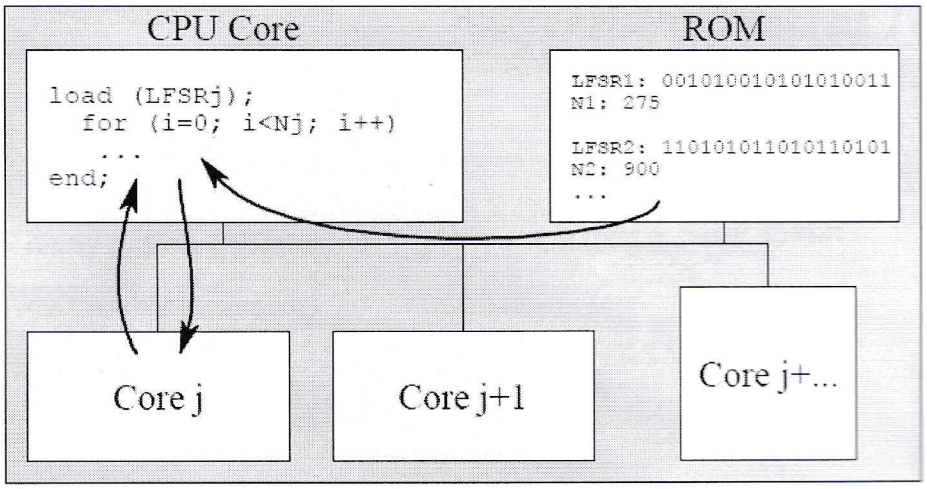


Figure 2.7b LFSR emulation.

The quality of the pseudorandom test is of great importance. It is assumed that for the hybrid BIST the best pseudorandom sequence will be chosen. However, not always all parts of the system are testable by a pure pseudorandom sequence. It takes often a very long test application time to reach a good fault coverage level. In case of hybrid BIST, we can dramatically reduce the length of the initial pseudorandom sequence by complementing it with deterministic stored test patterns, and achieve the 100% fault coverage.

As discussed in, the program to emulate the LFSR can be very simple and therefore the memory requirements for storing the pseudorandom TPG program together with the LFSR parameters are relatively small.

In the ideal case, the LFSR-based test generator can be developed in such a way that a high fault coverage will be reached by a sufficiently short test sequence. In our approach, the task is not to develop a new "ideal" LFSR-based test generator for BIST with the length equal to the minimal test set with 100% fault coverage. In general, for complex circuits such a task is rather difficult and may be even impossible to solve, especially for sequential circuits. In this sense, the hybrid BIST considered here suggests a more general and simple solution, applicable also for sequential cores.

Let, 'N' is the number of transistors in a chip and 'P' is the probability of that a transistor is faulty & 'Pf' is the probability that the chip is faulty. Then,

$Pf = 1 - (1 - p)^N$

If $p = 10^{-6}$

$N = 10^6$

Then, Pf = 63.2%

*A. Fault Coverage*

FC = No faults detected /No faults in fault list

Let a AND gate has two inputs: a, b and one output: c

So, possible 6 stuck-at faults: (a0, a1, b0, b1, c0, c1)

**Table 2.8: Testing Result**

| Test | Faults detected | FC |
|---|---|---|
| {(0,0)} | c1 | 16.67% |
| {(0,1)} | a1,c1 | 33.33% |
| {(1,1)} | a0,b0,c0 | 50.00% |
| {(0,0),(1,1)} | a0,b0,c0,c1 | 66.67% |
| {(1,0),(0,1),(1,1) | all | 100.00% |

*B. Clique*

The 'Clique' terminology comes from Luce and Perry (1949). First Algorithm for solving the Clique problem is that of Harary and Ross (1957). Tarjan and Trojanowski (1977), an early work on the worst-case complexity of the Maximum Clique problem.

Maximum Clique: *A Clique of the largest possible size in a given graph.*

Maximal Clique: *A Clique that cannot be extended by including one more adjacent vertex.*

In the 1990s, a breakthrough series of papers beginning with Feige (1991) and reported at the time in major newspapers, showed that it is not even possible to approximate the problem accurately and efficiently.

# *Chapter 3*
# *Algorithm*

## *Bron-Kerbosch Algorithm*

The Algorithm was designed and Published in 1973 by the Dutch scientists Joep Kerbosch and Coenradd Bron.

Bron-Kerbosch Algorithm is for finding the Maximal Cliques in undirected graph. It is known to be one of the most efficient algorithms which uses recursive backtracking to find Cliques is practically proven. The Bron-Kerbosch Algorithm uses the vertex in graph and its neighbors with few functions to generate some effective results.

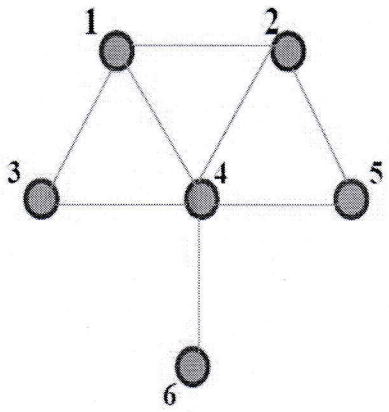


Fig 3.1: Bron-Kerbosch Algorithm.

R=X=∅, P = (1, 2, 3, 4, 5, 6)

Choosing the pivot element as 4.

4 in P \ N (v) = (1, 2, 3, 4, 5, 6) \ (1, 2, 3, 5, 6) = 4 in 4 Find's the values of Pnew, Rnew ,Snew

Pnew = P ∩ N (v), Rnew = R∪ (v), Xnew = X ∩ N (v)

Rnew = 4, Pnew = (1, 2, 3, 5, 6), Xnew = ∅

Bron-kerbosch (4), (1,2,3,5,6),∅)

Bron-kerbosch ((4,1),(2,3),Ø)

Bron-kerbosch ((4,1,2),Ø,Ø)

Report (4, 1, 2) as one of the Maximal Clique

Bron-kerbosch((4),(1,2,3,5,6),Ø) Bron-kerbosch((4,3),(1),Ø) Bron-kerbosch((4,3,1),Ø,Ø)

Report (4, 3, 1) as one of the other Maximal Clique

Bron-kerbosch((4),(1,2,3,5,6),Ø) Bron-kerbosch((4,2),(1,5), Ø) Bron-kerbosch((4,2,5), Ø,Ø)

Report (4, 2, 5) as another Maximal Clique

Bron-kerbosch((4),(1,2,3,5,6), Ø) Bron-kerbosch((4,6), Ø,Ø)

Report (4, 6) as the Maximal Clique

# *Algorithm 1*

Step 1) Define the possible maximum number of stages in the space compaction trees at the circuit under test output.

Step 2) Compute the total number of output lines in the circuit under test. Continue the following steps unless there is only a single output line (possibly).

Step 3) Find the sets of all maximal compatible classes from the circuit under test for logic AND, OR, and XOR by employing Algorithm B (mentioned later).

Step 4) Select a maximal compatible class MC*i based* on largest number of output lines, from the sets of all maximal classes. Select the second largest class during subsequent iteration, if 100% fault coverage is not realized in the preceding iteration from the same circuit under test.

Step 5) Merge the selected output lines of the MC*i using* appropriate logic gates AND, OR, or XOR.

Step 6) Add a new output line corresponding to the selected merged outputs in MC*i*.

Step 7) Discard all the output lines already used in MC*i*.

Step 8) Search for another MC class MC*j* from the remaining output lines.

Step 9) Merge the selected output lines in MC*j using* appropriate logic gates.

Step 10) Add a new output line corresponding to the selected merged outputs in MC*j*.

Step 11) Discard all the output lines already used in MC*j*.

Step 12) Repeat step 8 as long as there are maximal classes in the sets, and enough output lines.

Step 13) Calculate all the remaining output lines that do not belong to any of the selected maximal classes.

Step 14) Merge all the remaining lines with XOR gate.

Step 15) Add a new output line corresponding to the selected merged outputs.

Step 16) Inject stuck-at logic faults into the newly generated circuit under test (original circuit under test + compactor hardware).

Step 17) Compute fault coverage by applying input test patterns.

Step 18) If the fault coverage is 100%, then replace the old circuit under test with the new circuit under test, and repeat step 2 for computing the second stage of the compactor.

Step 19) If the fault coverage is less than 100%, then merge all the remaining lines with two-input XOR two output lines at a time.

Step 20) Add a new output line corresponding to the selected merged outputs.

Step 21) Inject stuck-at logic faults into the newly generated circuit under test (original circuit under test + compactor hardware).

Step 22) Compute fault coverage by applying input test patterns.

Step 23) If the fault coverage is less than 100%, then continue to work on the same circuit under test and repeat step 4 for selecting a new maximal compatible class MC*k*.

Step 24) If the fault coverage is 100%, then replace the old circuit under test with the new circuit under test, and repeat step 2 for computing the second stage and subsequent stages of the compactor.

# *Algorithm 2*

This algorithm implements the well-known Bron-Kerbosch algorithm for maximal clique finding. Steps are mentioned below:

Step 1) Calculate the total number of vertices in the undirected graph.

Step 2) Find the connected diagonal elements of the graph.

Step 3) Select a candidate point.

Step 4) Merge the selected candidate to a set called comp sub, which is to be extended by a new point, or shrunk by a point on traveling along a branch of the backtracking tree.

Step 5) Generate a new set called candidates, which is the set of all points that will in due time serve as an extension to the present configuration of comp sub.

Step 6) Create another set called not, which is the set of all points that, at an earlier stage, already served as an extension of the present configuration of comp sub, and are now explicitly excluded.

Step 7) Remove all points not connected to the selected candidate, keeping the old sets intact.

Step 8) Call the extension operator to perform on the newly generated sets.

Step 9) Remove the selected candidate from the comp sub, and add it to the old set not after returning.

# *Algorithm 3*

Algorithm 3 is the algorithm to compute and generate all the compatible pairs of the circuit under test for logic gates AND, OR, XOR.

Step 1) Calculate the total number of output lines of the circuit under test.

Step 2) Generate all possible combinations (*Ai, Aj*) of output lines, taken two at a time, and store all pairs of the output lines (*Ai, Aj*).

Step 3) Select the first pair from the list of combined output lines (*Ai, Aj*).

Step 4) Merge the selected pair of output lines (*Ai, Aj*) using logic gates AND, OR, and XOR respectively, using only one logic gate at a time.

Step 5) Add a new output line to the original circuit under test corresponding to the outputs (*Ai, Aj*), one at a time.

Step 6) Discard the output lines (*Ai, Aj*) from the original circuit under test, and generate a new modified circuit.

Step 7) Inject stuck-at logic faults into the newly generated circuit and apply test patterns.

Step 8) If the fault coverage is equal to 100%, then store the output pair (*Ai, Aj*) in the compatible pairs database of logic AND, OR, and XOR respectively.

Step 9) Delete the pair just selected, from the list of combined output lines (*Ai, Aj*) and select the next pair.

Step 10) Repeat step 4 and continue until all pairs are selected.

### A. ISCAS Combinational Benchmark Circuit C432

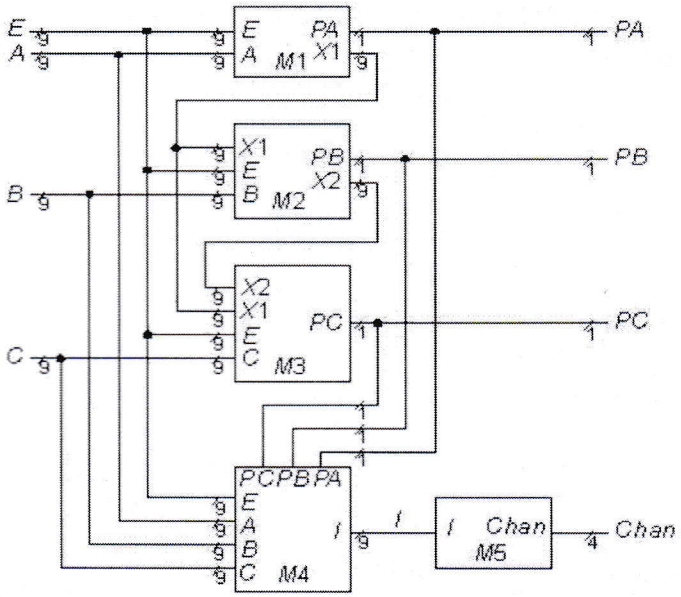


Fig 3.5a: Circuit of C432

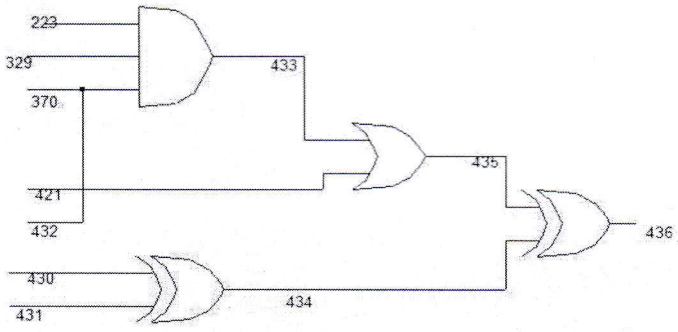


Fig 3.5b: Compactor Circuit 1 for C432

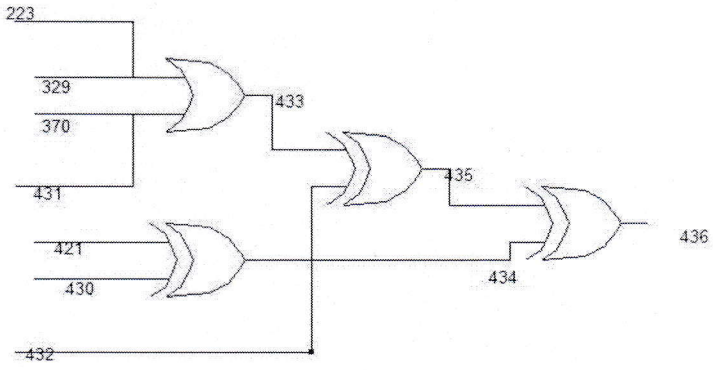


Fig 3.5c: Compactor circuit 2 for c432

### *B. ISCAS Combinational Benchmark Circuit C3540*

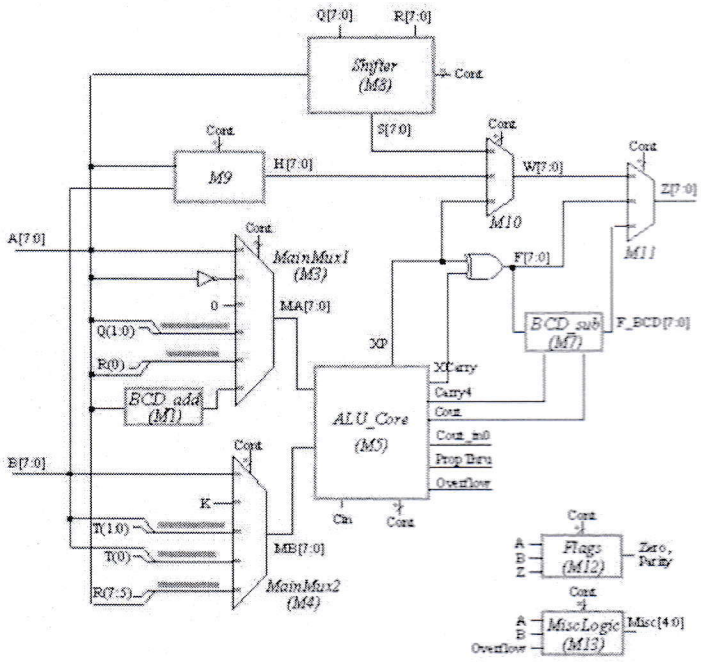


Fig 3.5d: Circuit of C3540

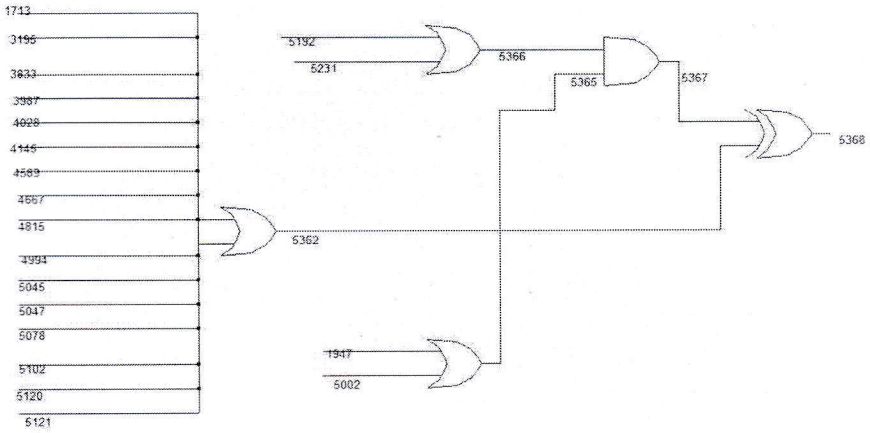


Fig 3.5e: Compactor Circuit 1 for C3540

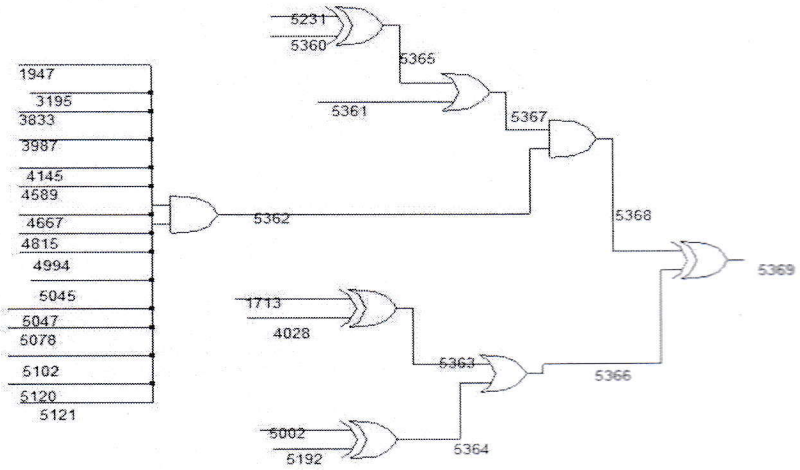


Fig 3.5f: Compactor Circuit 2 for C3540

Tab 3.5a: Fault simulation result in ATALANTA without compaction:

```
        *                                                          *
        *          Welcome to atalanta (version 2.0)               *
        *                                                          *
        *              Dong S. Ha (ha@vt.edu)                      *
        *            Web: http://www.ee.vt.edu/ha                  *
        *  Virginia Polytechnic Institute & State University       *
        *                                                          *
        ************************************************************

******   SUMMARY OF TEST PATTERN GENERATION RESULTS   ******
1. Circuit structure
   Name of the circuit                          : c432
   Number of primary inputs                     : 36
   Number of primary outputs                    : 7
   Number of gates                              : 160
   Level of the circuit                         : 17

2. ATPG parameters
   Test pattern generation mode                 : RPT + DTPG + TC
   Limit of random patterns (packets)           : 16
   Backtrack limit                              : 10
   Initial random number generator seed         : 1399286857
   Test pattern compaction mode                 : REVERSE + SHUFFLE
   Limit of suffling compaction                 : 2
   Number of shuffles                           : 8

3. Test pattern generation results
   Number of test patterns before compaction : 78
   Number of test patterns after compaction  : 48
   Fault coverage                               : 99.237 %
   Number of collapsed faults                   : 524
   Number of identified redundant faults        : 1
   Number of aborted faults                     : 3
   Total number of backtrackings                : 33

4. Memory used                                  : 1024 Kbytes

5. CPU time
   Initialization                               : 0.000 secs
   Fault simulation                             : 0.000 secs
   FAN                                          : 0.000 secs
   Total                                        : 0.000 secs
```

Tab 3.5b: Fault simulation result in ATALANTA with compaction:

```
        *          Welcome to atalanta (version 2.0)               *
        *                                                          *
        *              Dong S. Ha (ha@vt.edu)                      *
        *            Web: http://www.ee.vt.edu/ha                  *
        *  Virginia Polytechnic Institute & State University       *
        *                                                          *
        ************************************************************

******   SUMMARY OF TEST PATTERN GENERATION RESULTS   ******
1. Circuit structure
   Name of the circuit                          : c432
   Number of primary inputs                     : 36
   Number of primary outputs                    : 1
   Number of gates                              : 164
   Level of the circuit                         : 20

2. ATPG parameters
   Test pattern generation mode                 : RPT + DTPG + TC
   Limit of random patterns (packets)           : 16
   Backtrack limit                              : 10
   Initial random number generator seed         : 1399289671
   Test pattern compaction mode                 : REVERSE + SHUFFLE
   Limit of suffling compaction                 : 2
   Number of shuffles                           : 8

3. Test pattern generation results
   Number of test patterns before compaction : 128
   Number of test patterns after compaction  : 82
   Fault coverage                               : 99.240 %
   Number of collapsed faults                   : 526
   Number of identified redundant faults        : 1
   Number of aborted faults                     : 3
   Total number of backtrackings                : 34

4. Memory used                                  : 1024 Kbytes

5. CPU time
   Initialization                               : 0.000 secs
   Fault simulation                             : 0.000 secs
   FAN                                          : 0.000 secs
   Total                                        : 0.000 secs
```

Tab 3.5c: Fault simulation result in ATALANTA without compaction:

```
        *                                                     *
        *          Welcome to atalanta (version 2.0)          *
        *                                                     *
        *                Dong S. Ha (ha@vt.edu)               *
        *              Web: http://www.ee.vt.edu/ha           *
        *  Virginia Polytechnic Institute & State University  *
        *                                                     *
        *******************************************************

******   SUMMARY OF TEST PATTERN GENERATION RESULTS   ******
1. Circuit structure
   Name of the circuit                      : c3540
   Number of primary inputs                 : 50
   Number of primary outputs                : 22
   Number of gates                          : 1669
   Level of the circuit                     : 47

2. ATPG parameters
   Test pattern generation mode             : RPT + DTPG + TC
   Limit of random patterns (packets)       : 16
   Backtrack limit                          : 10
   Initial random number generator seed     : 1399318836
   Test pattern compaction mode             : REVERSE + SHUFFLE
   Limit of suffling compaction             : 2
   Number of shuffles                       : 15

3. Test pattern generation results
   Number of test patterns before compaction : 253
   Number of test patterns after compaction  : 151
   Fault coverage                           : 96.004 %
   Number of collapsed faults               : 3428
   Number of identified redundant faults    : 137
   Number of aborted faults                 : 0
   Total number of backtrackings            : 10

4. Memory used                              : 1024 Kbytes

5. CPU time
   Initialization                           : 0.000 secs
   Fault simulation                         : 0.000 secs
   FAN                                      : 0.000 secs
   Total                                    : 0.000 secs
```

Tab 3.5d: Fault simulation result in ATALANTA with compactor-circuit 1:

```
        *          Welcome to atalanta (version 2.0)          *
        *                                                     *
        *                Dong S. Ha (ha@vt.edu)               *
        *              Web: http://www.ee.vt.edu/ha           *
        *  Virginia Polytechnic Institute & State University  *
        *                                                     *
        *******************************************************

******   SUMMARY OF TEST PATTERN GENERATION RESULTS   ******
1. Circuit structure
   Name of the circuit                      : c3540
   Number of primary inputs                 : 50
   Number of primary outputs                : 1
   Number of gates                          : 1677
   Level of the circuit                     : 51

2. ATPG parameters
   Test pattern generation mode             : RPT + DTPG + TC
   Limit of random patterns (packets)       : 16
   Backtrack limit                          : 10
   Initial random number generator seed     : 1399318923
   Test pattern compaction mode             : REVERSE + SHUFFLE
   Limit of suffling compaction             : 2
   Number of shuffles                       : 15

3. Test pattern generation results
   Number of test patterns before compaction : 333
   Number of test patterns after compaction  : 193
   Fault coverage                           : 91.323 %
   Number of collapsed faults               : 3423
   Number of identified redundant faults    : 283
   Number of aborted faults                 : 14
   Total number of backtrackings            : 190

4. Memory used                              : 1024 Kbytes

5. CPU time
   Initialization                           : 0.000 secs
   Fault simulation                         : 0.000 secs
   FAN                                      : 0.000 secs
   Total                                    : 0.000 secs
```

Tab 3.5e: Fault simulation result in ATALANTA with compactor- circuit 2:

```
        *          Welcome to atalanta (version 2.0)          *
        *                                                     *
        *                Dong S. Ha (ha@vt.edu)               *
        *             Web: http://www.ee.vt.edu/ha            *
        *  Virginia Polytechnic Institute & State University  *
        *                                                     *
        *******************************************************

******   SUMMARY OF TEST PATTERN GENERATION RESULTS   ******
1. Circuit structure
   Name of the circuit                          : c3540
   Number of primary inputs                     : 50
   Number of primary outputs                    : 1
   Number of gates                              : 1676
   Level of the circuit                         : 50

2. ATPG parameters
   Test pattern generation mode                 : RPT + DTPG + TC
   Limit of random patterns (packets)           : 16
   Backtrack limit                              : 10
   Initial random number generator seed         : 1399319009
   Test pattern compaction mode                 : REVERSE + SHUFFLE
   Limit of suffling compaction                 : 2
   Number of shuffles                           : 14

3. Test pattern generation results
   Number of test patterns before compaction : 385
   Number of test patterns after compaction  : 208
   Fault coverage                            : 94.708 %
   Number of collapsed faults                : 3420
   Number of identified redundant faults     : 143
   Number of aborted faults                  : 38
   Total number of backtrackings             : 579

4. Memory used                               : 1024 Kbytes

5. CPU time
   Initialization                            : 0.000 secs
   Fault simulation                          : 0.000 secs
   FAN                                       : 0.000 secs
   Total                                     : 0.000 secs
```

# *Fault simulation result comparison*

## Tab 3.6a: Comparison Table for Circuit: c432

| Name of Test Pattern Generation Result | Without Compaction | With Compaction |
|---|---|---|
| **01. Circuit Structure:** | | |
| Number of Primary Output | 7 | 1 |
| Number of Gate | 160 | 164 |
| Level of the Circuit | 17 | 20 |
| **02. ATPG Parameters:** | | |
| Initial Random Number Generator Speed | 1399286857 | 1399289671 |
| **03. Test Pattern Generation Result:** | | |
| Number of Test Patterns before Compaction | 78 | 128 |
| Number of Test Patterns after Compaction | 48 | 82 |
| Fault Coverage | 99.237 % | 99.240 % |
| Number of Collapsed Faults | 524 | 526 |
| Total Number of Backtracking | 33 | 34 |

## Tab 3.6b: Comparison Table for Circuit: c3540

| Name of Test Pattern Generation Result | Without Compaction | With Compaction | |
|---|---|---|---|
| | | Circuit – 1 | Circuit – 2 |
| **01. Circuit Structure:** | | | |
| Number of Primary Output | 22 | 1 | 1 |
| Number of Gate | 1669 | 1677 | 1676 |
| Level of the Circuit | 47 | 51 | 50 |
| **02. ATPG Parameters:** | | | |
| Initial Random Number Generator Speed | 1399318836 | 1399318923 | 1399319009 |
| **03. Test Pattern Generation Result:** | | | |
| Number of Test Patterns before Compaction | 253 | 333 | 385 |
| Number of Test Patterns after Compaction | 151 | 193 | 208 |
| Fault Coverage | 96.004 % | 91.323 % | 94.708 % |
| Number of Collapsed Faults | 3428 | 3423 | 3420 |
| Number of Identified Redundant Faults | 137 | 283 | 143 |
| Number of Aborted Faults | 0 | 14 | 38 |
| Total Number of Backtracking | 10 | 190 | 579 |

# Chapter 4

# *MATLAB Based Cost Modeling for VLSI Testing*

Focus on cost of the test will result in a better understanding of cost trade-offs between test methodologies as per ITRS 2007 as shown in figure1. Typically, the cost of test boosts exponentially with an improvement in defects per million (DPM). Mathematical modeling in MATLAB GUI (graphic user interface) is very powerful as it reduces the designer's time. Modification of any designed function in MATLAB GUI is very easy. The flexibility of MATLAB is used for rapid deployment of the complex software to the end user.

.

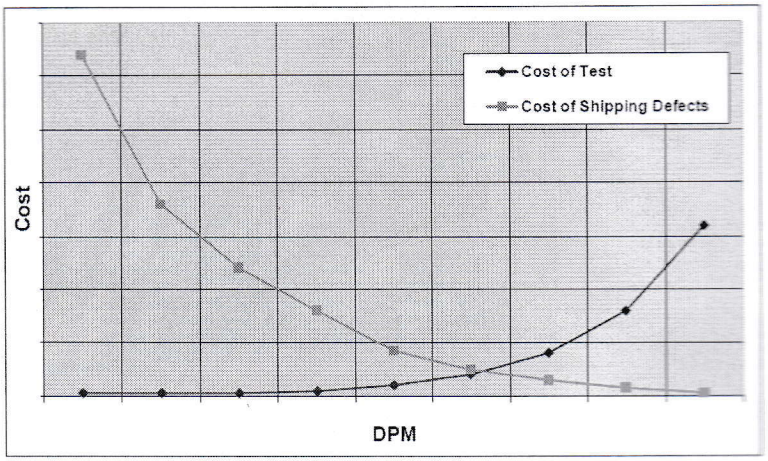


Figure 4.1: Quality and cost Trade-offs [ITRS 2007]

The paper is organized as follows: Section 2 describes the related work of economics of VLSI Testing and cost modeling. Section 3 presents the test cost model that is used for the automatic test equipment for multisite module testing. In Section 4 gives procedure for the cost modeling tool development using Matlab. Section 5 discusses a case study three devices for the verification of the economic analysis tool. The paper ends with the conclusions in Section 6 the DFT and also suggested that testability features should not be added to complex or high volume products. Abadir et al developed Hi-TEA, a MCM testing strategy selection tool, which helps to select the cost effective test strategy for the multi-chip module (MCM). Their tool required cost parameters such as die test cost and wafer yield, which are the parameters difficult to know in the early stage of design. Therefore, their tool may not be practical to predict a chip testing cost early on.

# ECONOMIC COST MODEL FOR ATEBASED VLSI TESTING

The cost of semiconductor test to the organization has many drivers that are labor cost, floor space cost, Maintenance cost, ATE cost per site etc. The significance of these drivers varies substantially from one device to another. Test development costs are more important for the products with lower volume. Cost model is structured with the help of cost parameters figure 2.

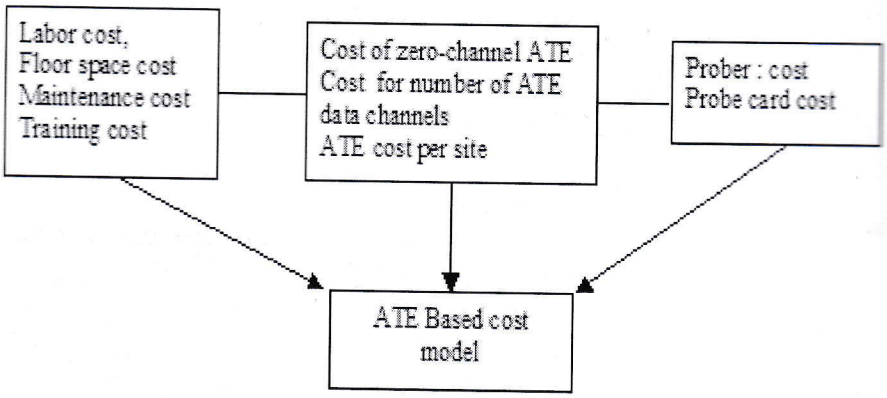


Figure 4.2: Economic cost model for ATE based VLSI Testing.

The cost model is targeted the reduction of the capital equipment cost and the test time. The area overhead due to Design for Testability (DFT) implementation is not considered here so silicon overhead cost for DFT is not modeled. Therefore, this model considers only cost associated with spending time on equipment and test engineering. Cost model is used for wafer sorting during testing. One assumption is made here is that all functional tests are done in package test.

The cost is calculated as:

$$C_t = C_{cap} \quad \frac{C_{testcell} \cdot T_{total}}{N_{site}} \cdot \quad \frac{N_p \cdot C_p}{N\text{-}N_{life\text{-}volume}} \ldots\ldots\ldots 1$$

$C_{test\ cell}$ : Total capital equipment cost of the test cell,
$N_p$ : Number of Probe cards
$T_{total}$ : Total time a die spends on the equipment
$Ct$ : Total time of a die spends on the ATE
$N_{sites}$ : Number of dies tested in parallel.
$C_{cap}$ : Constant consist amortization, utilization, labor cost, floor space, maintenance and training cost.

To calculate $N_p$ equation is given as

$$N_p = \left\lceil \frac{N_{life-volume}}{N_{site} * N_{mtd}} \right\rceil \ldots\ldots\ldots\ldots\ldots\ldots 2$$

Where $N_p$ : No. of probe cards
$N_{mtd}$ : Maximum touchdowns and
$N_{life\text{-}volume}$: Life time volume of dies

floor space, maintenance and training cost.

To calculate $N_p$ equation is given as

$$N_p = \left\lceil \frac{N_{life-volume}}{N_{site} * N_{mtd}} \right\rceil \ldots\ldots\ldots\ldots\ldots\ldots 2$$

W The cost of $N_p$ probe cards, which cost $N_{lifetime\text{-}Volume}$ of the product and maximum touchdowns $N_{mtd}$. The pseudo code the developed cost model is given below

```
{Enter the required data from user like
Enter Probe card cost;
Enter number of devices ;}
First of all for the calculation of Nsites
Matlab code is
{Rsignal = .1;
Nsite_ms = str2double (a)/str2double (b);
Nsite_ms_ATE=
str2double(c)/ str2double (d);
if (Nsite_ms< Nsite_ms_ATE)
    Nsites_MS = Nsite_ms;
Else(Nsites_MS= Nsite_ms_ATE;)
:  end  }}{Similarly  Matab  code  is  developed  to
calculate the total cost Ct}
```

# *COST MODELING TOOL WITH MATLAB GRAPHICAL USER INTERFACE*

Market modeling and Cost prediction/Estimation are new areas in which interest of physical and mathematical researchers is growing due the stochastic nature of the financial processes. Constraining by this interest it becomes necessary to develop comprehensive software environment, which will use the same models for the simplification for quantitative analysis. The main advantage for such approach is that it provides rapid prototyping, high-quality visualization, and enhanced model testing to the end users.

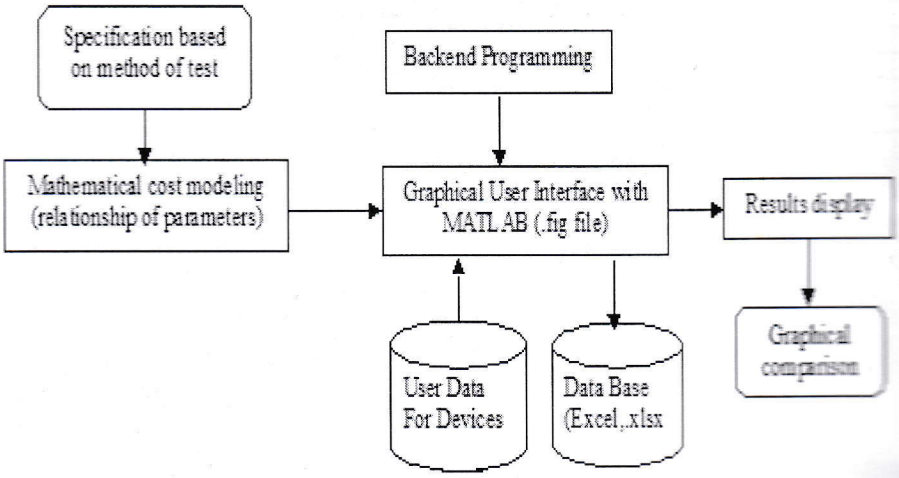


Figure 4.3: Tool development of Cost Modeling

GUI design is based on mathematical equations and user inputs Graphical User Interface is designed in MATLAB (.fig file) and backend callback functions are called from GUI for each calculation. Development of GUI and its link with the database are shown in figure 3.

In this, GUI has been created for mathematical equations. Numbers of input variables are set depending upon the equation to be designed. Property of every single component is set in the Property Inspector. A MATLAB code is written, which is generated by the callbacks of particular push button. An event is created by clicking on push button for final result calculations, which causes the function of the button to be executed. A link is established between database which is created in excel file and GUI.

The model will help us evaluate the direct cost impacts of various values in the balanced scorecard, for the test processes. Using cost models, the relative effectiveness of two different test processes can be evaluated. To calculate the total cost of each test process user is required to enter the input data as per parameters required for estimation are shown in the figure 4.3

Table 4.3 Devices for cost modeling and parameter specifications.

| Parameter | Description | Application | | |
|---|---|---|---|---|
| | | Set-top-box | µP | Device A |
| N_lifetime | Device: Lifetime volume [k] | 1000 | 5000 | 250 |
| G | Device: Number of logic gates [M] | 2 | 20 | 6 |
| f_max,chain | Device: Maximum scan chain frequency [MHz] | 20 | 100 | 30 |
| f_max,I/O | Device: Maximum I/O frequency for scan [MHz] | 100 | 400 | 200 |
| P_total | Device: Total number of device pads for wafer test | 300 | 1000 | 400 |
| t_fix | Time for DC+PLL+ Mixed-signal+ Mem [s] | 6 | 2 | 2 |
| C_ATE0 | ATE: Cost zero channel [k$] | 150 | 150 | 150 |
| C_site | ATE: Cost per site resource [k$] | 25 | 60 | 25 |
| P_max,ATE | ATE: Maximum number of channels | 1000 | 1000 | 1000 |
| f_max,ATE | ATE: Maximum ATE data channel frequency [MHz] | 250 | 400 | 400 |
| C_chan | ATE: Channel Cost High / Low [$] | 1K/400 | 1K/400 | 1K/400 |
| C_prober | Prober: Cost [k$] | 350 | 350 | 350 |
| t_index | Prober: Index time [s] | 1 | 1 | 1 |
| c_capital | Test cost per sec on M$ [$] | 0.02 | 0.02 | 0.02 |
| c_volume | ATPG test data volume per gate [bits] | 400 | 400 | 400 |
| P_ctrl | Number of control signals | 3 | 6 | 3 |
| f_max,probecard | Probe card: Maximum frequency [MHz] | 150 | 150 | 150 |
| P_max,probecard | Probe card: Maximum number of contacts | 1000 | 1000 | 1000 |
| C_probecard | Probe card: Cost [k$] | 20 | 60 | 20(reusable) |
| N_max,touchdown ns | Probe card: Maximum number of touchdowns [k] | 150 | 150 | 150 |
| R_testport | Test port [%] | 50 | 50 | 50 |

# *Conclusions*

This paper presents a new technique for solving space compaction problem of circuit under test (CUT). The technique utilizes Brone Karbosch Algorithm concepts. Brone Karbosch Algorithm is well for space compaction but it's some limitation for full fill all space compaction. We optimized get batter result at space compaction using Brone Karbosch & Brute Force Algorithm. We think space compaction is not little work, it is more deep work, so need to more work at space compaction. This is an efficient technique and it is achieved without any prior modification of the original circuit (MUT), as all the modification for testing is done in software mode, whereas maximal compaction is realized in most cases in reasonable time utilizing some simple heuristics. The technique, illustrated with details of design of space compactors for ISCAS 85 combinational with ATALANTA simulation programs, confirms the usefulness of the suggested approach, its simplicity, resulting low area overhead, and full fault coverage (FC) for single stuck-line faults, making it suitable in a VLSI design environment as BIST support hardware. It might be fair to mention here that the test vectors used for detecting single stuck-line faults in digital BIST these days, irrespective of whether the circuits are combinational or full- scan sequential, are either deterministic, minimal or non- minimal, complete, or simply reduced but not necessarily complete, or pseudorandom. If minimal complete test sets are available for ISCAS 85 circuits, this is obviously would be the best choice in testing any circuit for complete FC using minimal time and resources.

On the other hand, programs such as ATALANTA as used in the paper generate reduced test sets that detect most single stuck-line faults for ISCAS 85 circuits, which give very good fault coverage, not necessarily 100%, for most of the ISCAS benchmark circuits. We found 95% average fault coverage & minimum error at space compaction. But some limitation at propose Algorithm, those are not sufficient at all space compaction. All these types of test vectors are absolutely compatible with the approach developed in this research.

We found at compeer of circuit c432 without & with compaction, number of test patterns before compaction, without compaction is 78 & with compaction is 128. Number of test patterns after compaction, without compaction is 48 & with compaction is 82. Fault coverage with compaction 99.24%. Also we found at compeer of circuit c3540 without & with compaction, number of test patterns before compaction, without compaction is 253 & with compaction is circuit-1: 333 & circuit-2: 385. Number of test patterns after compaction, without compaction is 151 & with compaction is circuit-1: 193 & circuit-2: 208. Fault coverage with compaction 94.7%. Notwithstanding this, the proposed theoretical framework, although was simulated on the assumption of single stuck-line faults, is amenable to modification to take care of multiple fault situations as well, if programs are available to simulate multiple stuck line faults. But, multiple faults are extremely difficult to analyze in view of their sheer number. Besides, modern day circuits are extremely reliable not to exhibit simultaneous multiple stuck-line faults, although there are occasions where these faults need to be considered. However, in this work, we did not, as in the case of most other studies. Besides, we do not consider faults such as stuck-open faults, short faults, bridging faults, intermittent faults or transient faults in our analysis,

the last two types being very difficult to investigate theoretically, requiring discrete or continuous Markov modeling. With advances in computational resources, evidently this heuristic space compaction algorithm might be improved upon for better efficiency in respect of time and storage.

# References


- S.R. Das, [3]Built-in self-testing of VLSI circuits', *IEEE Potentials, 10*,1991, pp. 23-26.
- Satyendra Biswas, Sunil R. Das, Emil M. Petriu; *Space Compactor Design in VLSI Circuits Based on Graph Theoretic Concepts*; IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 55, NO. 4, AUGUST 2006.
- Das, S.R. , Hossain, A. , Biswas, S. , Petriu, E.M.; *Aliasing-free compaction* Circuits, Devices & Systems, IET (Volume:2 , Issue: 1 )Page(s):166 ± 178 February 2008.
- Das, S.R. , Hossain, A. , Biswas, S. , Petriu, E.M.; *Aliasing-free compaction* Circuits, Devices & Systems, IET (Volume:2 , Issue: 1 )Page(s):166 ± 178 February 2008.
- Biswas, S.N.; Das, S.R. ; Petriu, E.M. ; Hossain, A.; *Hybrid test vector compression in system-on-chip test — An overview and methodology;* Computers and Devices for Communication, 2009. CODEC 2009.
- Biswas, S.; Das, S.R.; Hossain, A.*; VLSI Circuit Test Vector Compression Technique;* Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007. IEEE; Page(s):1 ± 6; 1-3 May 2007.
- Siddiquee, M.F. ; Hasan, M.M. ; "Space Compaction and Use of Minimal Logic Gate in VLSI Test Circuit Design by Uniquely Developed Algorithm Based on Graph Theoretical Approach"; ICEEICT, 2014.
- M. Abramovici, M. A. Breuer, A. D. Friedman, "Digital Systems Testing and Testable Design," IEEE Press, 1990.
- V. D. Agrawal, C. R. Kime, K. K. Saluja, "A Tutorial on Built-In Self-Test," IEEE Design and Test of Computers , pp. 73-82, March 1993, pp. 69-77, June 1993.
- S. B. Akers, "Binary Decision Diagrams" IEEE Trans. On Computers, Vol. 27, pp. 509-516, 1978.
- C. Angelbro, "P-Bist Test Method Conquer the Ericsson World," *Ericsson Telecom AB*, 1997.
- P. H. Bardell, W. H. McAnney, J. Savir, "Built-In Test for VLSI Pseudorandom Techniques," John Wiley and Sons, 1987.
- B. Beizer, "Software Testing Techniques," (2nd edition) Van *Nostrand Reinhold*, 1990.

- M. A. Breuer, A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, 1976.

- F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," IEEE *Int. Symp. on Circuits and Systems*, pp. 663-698, June 1985.

- M. Chatterjee, D. K. Pradhan, "A novel pattern generator for Near-perfect fault-coverage," VLSI Test Symposium, pp. 417-425, 1995.

- [R. A. DeMillo, R. J. Lipton, F. G. Sayward, "Hints on Test Data Selection: Help for the Practical Programmer," IEEE Computer, Vol.11, No.4, April 1978.

- E. B. Eichelberg, E. Lidbloom, "Random Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," IBM *Journal of Research and Development*, Vol. 27, No. 3, pp. 265-272, May 1983.

- R. D. Eldred, "Test Routines Based on Symbolic Logic Systems," *Journal of the ACM*, Vol. 6, No. 1, pp. 33-36, 1959.

- P. Eles, K. Kuchcinski, Z. Peng, M. Minea, "Compiling VHDL into a High-Level Synthesis Design Representation," EURO- *DAC*, pp. 604-609, 1992.

- F. Ferrandi, G. Ferrara, D. Scuito, A. Fin, F. Fummi, "Functional Test Generation for Behaviorally Sequential Models," Design, Automation and Test in Europe (DATE 2001), pp. 403-410, 2001.

- F. Glover and M. Laguna. "Modern Heuristic Techniques for Combinatorial Problems", Blackwell Scientic Publishing, pp. 70- 141, 1993.

- A. Grochowski, D. Bhattacharya, T. R. Viswanathan, K. Laker, "Integrated Circuit Testing for Quality Assurance in Manufacturing: History, Current Status, and Future Trends," *IEEE Trans. on Circuits and Systems – II*, Vol. 44, pp. 610-633, August 1997.

- P. Hansen. The steepest ascent mildest descent heuristic for combinational programming. Congress on Numerical Methods in *Combinatorial Optimization*, 1986.

- W. E. Howden, "Weak Mutation Testing and Completeness of Test Sets," IEEE Transactions on Software Engineering," Vol. SE-8, No.4, July 1982.

- O. H. Ibarra, S. Sahni, "Polynomially Complete Fault Detection Problems," IEEE Transactions on Computers, Vol. C-24, No. 3, pp. 242-249, March 1975.

- "IEEE Standard VHDL Language Reference Manual," *ANSI/IEEE Std 1076-1993, (Revision of IEEE Std 1076-1987)*, June 6, 1994

- G. Jervan, Z. Peng, R. Ubar, "Test Cost Minimization for Hybrid BIST," IEEE Int. Symp. On Defect and Fault Tolerance in VLSI *Systems (DFT€)*, pp.283-291, 2000.

- J. Khare, W. Maly, N. Tiday, "Fault characterization of standard cell libraries using inductive contamination analysis (ICA)," 14th *VLSI Test Symposium*, pp. 405-413, 1996.

- S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, pp. 671-680, 1983.

- B. Könemann, J. Mucha, G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuits", IEEE J. Solid-State Circuits, Vol. SC-15, No. 3, pp. 315-319, June 1980.

- E. J. Marinissen, Y. Zorian, "Challenges in Testing Core-Based System ICs," IEEE Communications Magazine, pp. 104-109, June 1999.

- K. Marriott, P. J. Stuckey, Programming with Constraints: Introduction, MIT Press, 1998.

- P. Michel, U. Lauther, P. Duzy,"The Synthesis Approach To Digital System Design," Kluwer Academic Publishers, 1992

- B. T. Murray, J. P. Hayes, "Hierarchical Test Generation Using Precompiled Tests for Modules," International Test Conference, pp. 221-229, 1988.

- J. Raik, R. Ubar. "Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations." Journal of *Electronic Testing: Theory and Applications (JETTA)*, vol. 16, no. 3, pp. 213-226, June, 2000.

- J. P. Shen, W. Maly, F. J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," IEEE Design and Test of Computers, Vol. 2, No. 6, pp. 13-26, December 1985.

- M. Sugihara, H. Date, H. Yasuura, "Analysis and Minimization of Test Time in a Combined BIST and External Test Approach," *Design, Automation & Test in Europe Conference (DATE 2000)*,pp. 134-140, March 2000.

- Y. Sun, "Automatic Behavioral Test Generation By Using a Constraint Solver", Final Thesis, LiTH-IDA-Ex-02/13, Linköping *University*, 2001.

- N. A. Touba, E. J. McCluskey, "Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST," *IEEE Int. Test Conference (ITCÍ)*, pp. 674-682, 1995.

- N. Zacharia, J. Rajski, J. Tyzer, "Decompression of Test Data Using Variable-Length Seed LFSRs," *13th VLSI Test Symposium*, pp. 426-433, 1995.

- Y. Zorian, E. J. Marinissen, S. Dey, "Testing Embedded Core- Based System Chips," IEEE International Test Conference (ITC), pp. 130-143, IEEE Computer Society Press, October 1998.