Hayawic Logic Application to Test Pattern Generation Using the Interest Square Unity (ISU) Technique

Marguerite “Peggy” J. Palmer

INFT 858 Class Project
Professor Alnakari
Spring 1998
George Mason University
Fairfax, VA

Table of Contents

Abstract

This paper examines the various issues associated with a behavioral level methodology for achieving 100% fault coverage in Complementary Metal Oxide Semiconductor (CMOS) Very Large Scale Integrated (VLSI) circuits. First, research on fault models and their applicability to CMOS is explored leading to a basis for determining fault coverage. Second, a number of behavioral test generation techniques are evaluated for their ability to detect faults and to achieve 100% fault coverage. The conclusion indicates the need for further research at the behavioral level to achieve 100% fault coverage for CMOS VLSI circuits.

The Interest Square Unity (ISU) technique, a tool of Unilogic, was developed by the Damascus School of Hayawic Logic and offers a systematic method of selecting, analyzing, and solving any problem. The ISU can be shown graphically to contain four global states, with each state representing the possible locations of the entity in its existence. It represents a logic in that the state transitions can be made from one state to another. It can be used as an analysis tool because each state can be used to represent the factors or perspective of the problem.

ISU could be used through analysis and design to help improve fault models, fault coverage, and address any test pattern generation areas which are not well supported by current Design for Test (DFT) tool suites. Therefore, this paper will limit its scope to analyzing only the four behavioral level test generation approaches since they represent a range of techniques which support design for testability. Also, since there is so much research in this field, the four approaches will be used as representative cases to show the utility of ISU for analysis of them.

As ISU is applied to the test generation problem, several solutions should become apparent for solving the behavioral level issues for achieving 100% fault coverage in CMOS VLSI circuits. ISU and Unilogic can then be applied to each of the four different test generation techniques presented. This will evaluate their ability to detect faults and to achieve 100% fault coverage to see where they exist in the universe of test generation. Finally, ISU can be applied to help address the problem and develop a direction for solving it. ISU will not solve the problem but will provide an analysis tool which will provide insight on the dimensions and the dynamics of the problem and the result will be a new direction from which to solve the problem.

Our objective is to determine a Unilogic form which can help steer further research efforts in the direction which can be one alternative to achieving the holistic testing approach. The differences in this approach compared to the other four will be described and it will be shown how this new holistic approach direction may be used to analyze and discover the different types of faults in CMOS VLSI circuits. Lastly, we will discuss what the expected results would be if the new direction were to be pursued and whether the new holistic approach can be implemented and what would be required.

Also, we will describe what criteria should be used to test the efficiency and effectiveness of this new holistic testing approach based on ISU.

Preface

This paper was prepared in partial fulfillment of the requirements of ECE858, Object Oriented Modeling and Artificial Intelligence, Spring Semester 1998, George Mason University, under the professorship of Dr. Raiek Alnakari. This work began at the midway through the semester. Many sources for information were contacted during this project and a list of those contact and the status is available in Appendix D. Much of the detailed information to accomplish more in-depth analysis was either not available or arrived close to the end of the project and therefore was not as fully factored into the analysis, however it can be used for any potential follow-on work pursued. The objective of this project is to apply the Hayawic Unilogic approach to the problem of providing a high level of fault coverage in digital integrated circuits. Preliminary results were produced from this work but additional research is required to fully benefit form the Unilogic approach and to derive more specific solutions. I would like to thank Dr. Alnakari for providing valuable guidance and direction for this project.

ABOUT THE AUTHOR: Marguerite J. Palmer is currently a Major in the U.S. Air Force and is assigned as the Radio Frequency Spectrum Manager for the Military Communications-Electronics Board for the Command, Control, Communications, and Computers (C4) Directorate of the Joint Staff in the Pentagon, Washington, DC. She has served as a Deputy Division Chief for an Intelligence Dissemination System Program Office; Legislative Liaison and Program Manager, On-board Processing and Microelectronics Programs for the Ballistic Missile Defense Organization (BMDO); Lead Engineer for an Early Warning Satellite System Program Office; and a Program Control Officer for an Advanced Communications Program Office. Maj Palmer is currently working on a PhD and holds a MEEE from the University of Virginia, an MBA from Chapman University, and a BS from Rensselaer Polytechnic Institute.

Introduction

This paper will address an area which is extremely important for information technology, computer engineering, and telecommunications fields. Large integrated circuits (ICs) are the key elements which have made possible the high performance of personal computers and the ability to use multi-media and real-time interactive applications. The challenge for the IC designers is to ensure that their large, complex chips function as they are intended to. That is, they can operate correctly and without faults. Or, if there are faults, they do not corrupt the proper operation of the circuit and produce erroneous results. Recall several years ago when a researcher discovered that an Intel Pentium microprocessor produced errors during certain operations. Intel at first did not take this news seriously and said that most users would never notice this erroneous behavior. However, to their surprise, users of PCs did not want to have the "faulty" microprocessor in their system. Therefore, there was a very large public outcry for replacing the bad ICs. Intel finally responded by announcing they would replace all the microprocessors free of charge.

In order to avoid this negative experience from recurring, the IC community wants to ensure that the chips that finally make it to the commercial world are fault-free. In order to do this, they must have a way to test these complex chips for the presence of faults. Challenging chip test is the large number of test patterns that have to be generated to completely (or exhaustively) test the circuit. For example, if you have a simple digital circuit with two inputs and one output, you can test all 9 input combinations by applying the following inputs: (00), (10), (01), and (11). This turns out to be 2^N test patterns. This is very manageable for small circuits because you can easily generate the input combinations and verify that the actual output matches the expected output, thus verifying correct circuit operation. However, for a circuit with 64 inputs, 2^{64} input patterns equal 1.84×10^{19} different input patterns which becomes an impossible number to exhaustively test. Therefore, automatic test pattern generation has been a booming research field. In addition, the ability to measure an IC's design quality by the fault coverage is an extremely important measure of merit. For ICs, most faults are produced during the manufacturing IC process, by such things as defects, extensions or omissions in the various layers such as the gate oxide, the metalization, the polysilicon, which can create shorts, opens, or extraneous circuit behavior creating faults. Many different types of faults exist and a very large number of algorithms and models have been developed to help detect these faults.

Fault coverage is defined as the number of detected faults divided by the total number of faults, and is given in a percentage. Fault coverage figures as close to 100% are desired. Usually simulation is used to measure the fault coverage of a circuit with fewer than 200,000 logic gates, which are typically NAND or NOR gates. But with ICs with over 4 million transistors, this means they have approximately 1 million logic gates. For ICs with more than 200,000 gates, simulation is very time consuming and

computationally intensive. For those tools which can handle such a design, the fault simulation can take can take several weeks or even months to complete.[1] Therefore, Design for Test (DFT) techniques are required.

DFT is a booming field. The world leader in DFT tools is Mentor Graphics. Mentor Graphics is an important supplier of both automatic test pattern generation (ATPG) and memory and logic built-in-self-test (BIST) software, critical for deep submicron design, while also offering the industry's most complete line of DFT solutions. Mentor Graphics Corporation (NASDAQ: MENT) is a leading supplier of electronic hardware and software design solutions, providing products and consulting services for the world's largest electronics and semiconductor companies. Established in 1981, the company reported revenues over the last 12 months of $455 million and employs approximately 2,570 people worldwide. Company headquarters are located at 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.[2]

A recent query on the Internet under the search phrase of "faults and test pattern generation" using the Alta Vista search engine resulted in over 1.67 million web page hits. Obviously, there is considerable research ongoing in this field. In addition, University of California at Santa Barbara is teaching a sophomore level Electrical and Computer Engineering class in VLSI Testing Technique by Dr. K. T. Tim Cheng located at timcheng@ece.ucsb.edu.

Therefore, any improvements in the ability to improve the abilty to test ICs or to improve fault coverage is highly desired. This paper will address exactly that. This paper examines the various issues associated with a behavioral level methodology for achieving 100% fault coverage in Complementary Metal Oxide Semiconductor (CMOS) Very Large Scale Integrated (VLSI) circuits. First, research on fault models and their applicability to CMOS is explored leading to a basis for determining fault coverage. Second, a number of behavioral test generation techniques are evaluated for their ability to detect faults and to achieve 100% fault coverage. The conclusion indicates the need for further research at the behavioral level to achieve 100% fault coverage for CMOS VLSI circuits.

The Interest Square Unity (ISU) technique, a tool of Unilogic, was developed by the Damascus School of Hayawic Logic and offers a systematic method of selecting, analyzing, and solving any problem. The ISU can be shown graphically to contain four global states, with each state representing the possible locations of the entity in its existence. It represents a logic in that the state transitions can be made from one state to another. It can be used as an analysis tool because each state can be used to represent the factors or perspective of the problem.

ISU could be used through analysis and design to help improve fault models, fault coverage, and address any test pattern generation areas which are not well supported by current Design for Test (DFT) tool suites. Therefore, this paper will limit its scope to analyzing only the four behavioral level test generation approaches since they represent a

range of techniques which support design for testability. Also, since there is so much research in this field, the four approaches will be used as representative cases to show the utility of ISU for analysis of them.

Our objective is to determine a Unilogic form which can help steer further research efforts in the direction which can be one alternative to achieving the holistic testing approach. The differences in this approach compared to the other four will be described and it will be shown how this new holistic approach direction may be used to analyze and discover the different types of faults in CMOS VLSI circuits. Lastly, we will discuss what the expected results would be if the new direction were to be pursued and whether the new holistic approach can be implemented and what would be required. Also, we will describe what criteria should be used to test the efficiency and effectiveness of this new holistic testing approach based on ISU.

Overview

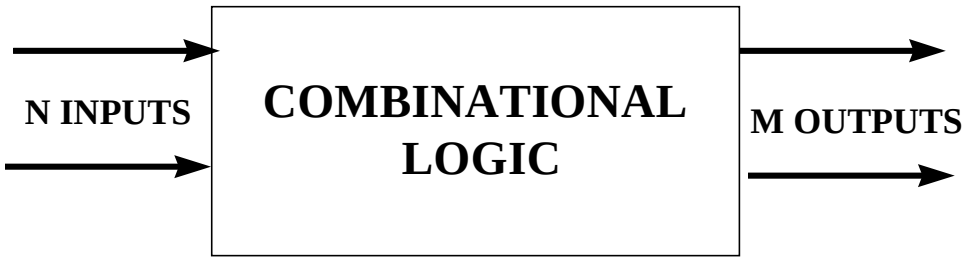
Problem Statement

Integrated circuit complexity is increasing at a tremendous pace based upon advances in high level design tools, microprocessor performance improvements, and smaller feature size process technology. Unfortunately, conventional test generation techniques are challenged by this increased complexity, the need for test vector generation and fault simulation in a timely manner, providing adequate fault coverage, and the newer process technologies requiring additional fault models. Much Very Large Scale Integration (VLSI) design is conducted using behavioral level design tools such as Very High Speed Integrated Circuit (VHSIC) Hardware Design Language (VHDL). Numerous advantages can be gained by test generation conducted at this same level. Most gate level test generation techniques rely on fault models, such as the single stuck-at (SSA) fault, which do not completely characterize the fault modes for Complementary Metal Oxide Semiconductor (CMOS) circuits. Lastly, high fault coverage, preferably 100%, is desired for long term Integrated Circuit (IC) applications where devices must operate as expected in the presence of faults without repair. Therefore, much benefit can be gained through 100% fault coverage using behavioral test generation techniques for CMOS ICs.

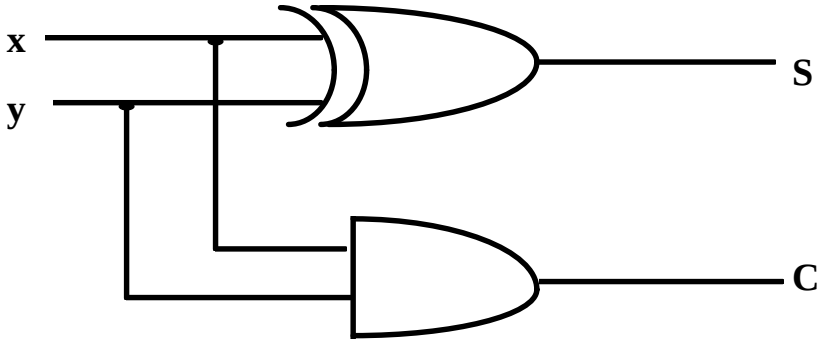
Fault modeling can occur at many levels such as the electrical, logical, functional, register-transfer level, or behavioral. Regardless of the level, the objective of fault modeling is to accurately represent the physical faults so that the circuit can be simulated and test vectors generated to detect as many of the faults as possible. Physical faults, such as a short circuit, can be modeled at the electrical level, providing higher accuracy but at the expense of increased computational complexity whereas the same fault can be modeled at the logic gate level with greater simplicity. As one attempts to model faults at higher levels of abstraction, such as the functional or behavioral level, these models must cover a wide range of physical faults. These models, by virtue of their simplicity, may not detect all physical failures for which they are designed to cover. However, the tremendous circuit complexity of VLSI makes it paramount to develop behavioral level fault models which can be executed in a reasonable time and yet provide a high degree of fault coverage.

To better understand fault detection and test generation, a simple example will be provided. We will use a half-adder circuit and show how a fault can impact its correct operation. A combinational logic circuit can be viewed as a box with n inputs and m outputs. For n inputs, there are 2^n possible combinations of binary input values. For each input combination, there is only one output combination. Shown below is the one Boolean logic truth table for the half adder with x and y as the two input variables and C and S as the Carry and Sum output variables. A half adder performs the addition operation of two binary digits, which are 0+0=0, 0+1=1, 1+0=1, and 1+1=10. When both

inputs are 1, the binary sum produces a two digit binary number, where the right bit equals 0 x 2^0 = 0 and the left bit equals 1 x 2^1 = 2, which is the expected answer when adding 1 to 1.



One implementation of the half adder is shown below using one AND gate and one exclusion or (XOR) logic gate. The truth table for correct operation is also shown.



HALF ADDER IMPLEMENTATION

x	y	/	C	S
0	0	/	0	0
0	1	/	0	1
1	0	/	0	1
1	1	/	1	0

HALF ADDER TRUTH TABLE DIAGRAM

If one of the inputs, for instance x, is stuck at 1, from perhaps an electrical short shown at location X between the input wire x and the power supply line, then regardless of the value provided for the x input, the combinational logic will always interpret its value as being 1. This is called a single stuck at one (SSA1) fault. The truth table will now be shown with the faulty input. By comparing its output with the correct truth table output above, one can see where the fault will produce erroneous output data. These incorrect values produced by the SSA1 at x fault are shown in bold and underlined below.

x	y	/	C	S
1	0	/	0	**1**
1	1	/	**1**	**0**
1	0	/	0	1

FAULTY HALF ADDER TRUTH TABLE DIAGRAM FOR x SSA1

1 1 / 1 0

The process of test generation involves determining what type of fault one wants to test for and at what fault site. If we wanted to test for the SSA1 fault described above at the x input line, then we would activate the fault. This means setting the x input value to 1. In this case x is both the primary input, which is an input line whose value can be controlled or set, and the fault site. Then we would propagate the fault to a primary output. Primary means that you can observe the value on the output. In this simple example, these steps are trivial because the fault can be set at a primary input and the results can be seen at a primary output. However, in larger circuits, one must propagate the values at the fault site to a primary output. This involves selecting all the subsequent values of signals that would interact with the value at the fault site to ensure that they pass the value you are testing for to a primary output unchanged. One also has to select the input values which would activate and test for the fault.

Further challenging fault model methodology are process technology dependencies producing different types of physical defects, thereby creating different fault types. However, one fault model, the single stuck fault, ssf, model is almost always used as a basis for fault grading.[3] McCluskey explains that although different fault models exist, such as multiple stuck fault, bridging faults, delay faults, stuck-open faults, etc., the ssf is the most prevalently used. His reasons are the big investment in the ssf, it creates many test patterns applied to a single gate resulting in propagation to a primary circuit output, and fault simulation can determine the coverage of ssf test patterns and a high fault coverage indicates that the test set is close to a gate exhaustive test.

Other research indicates that the ssf model was based and applied to early digital circuit and that high ssf coverage can no longer guarantee high quality for CMOS ICs.[4] Soden and Hawkins explain different fault simulators give different results and often calculations are modified to produce high ssf coverages. They also state the ssf model doesn't accurately reflect CMOS IC defects and failure mechanisms which produce effects other than a single stuck-at node. They recommend replacing abstract models with realistic defect models, such as the transistor bridging model, and using test methods such as I_{DDQ} and at-speed testing to minimize the difference between actual and predicted defect coverage.

Research by John P. Shen, et al[5] developed Inductive Fault Analysis (IFA) to predict all faults expected to occur in a Metal Oxide Semiconductor (MOS) IC based upon the IC layout. The result is a fault model and a ranked fault list which factors in the technology, layout, and process characteristics. They applied their method to an NMOS modified full-adder circuit having 4 inputs, 3 outputs, and 29 transistors. The results indicated that out of the original 4800 defects, only 476 would produce faulty behavior at the circuit level. These are further broken out by fault type: 28% line stuck-at faults, 15% transistor stuck at faults, 21% floating line faults, 30% bridging faults, and 6% miscellaneous faults. Therefore, the classical stuck-at fault model, which could explicitly

model the first type and implicitly model the second and third type faults, could only account for 64% of all the faults. This leaves 36% unaccounted for, of which bridging faults were the predominant fault type. They felt that although this was an NMOS circuit, the results apply to CMOS circuits. Their IFA procedure can be used to develop automatic test generation programs using the ranked fault list.

Abramovici states[6] that high fault coverage for ssfs is a necessary but not sufficient condition to achieve high defect coverage. He defines the correct way of computing (absolute) fault coverage as FC=D/T where D is the number of detected faults by an automatic test generation program and T is the total number of target ssfs. He further shows how another measure, called detectable fault coverage, defined as DFC=D/(T-U), where U is the number of undetectable faults, is not a reliable measure of fault coverage for either sequential or combinational circuits because the undetected faults can impair circuit performance or mask other faults. Therefore, test generation must not only target all relevant fault classes, beyond the ssf, but needs to detect all faults in order to achieve 100% fault coverage in CMOS ICs.

The environment is test generation of very large scale integration (VLSI) Complementary Metal Oxide Semiconductor (CMOS) integrated circuits. The problem can be broken down into the following parts:

1) Need high fault coverage (approaching 100%) by detecting the multiple faults types which exist
2) IC complexity increasing at a tremendous rate but test generation methods not keeping up
3) Existing test generation and fault detection methods too computationally intensive and too slow

Test generation techniques which depend on an efficient fault detection methodology and accurate fault models are either not applied well or are non-existent at the behavioral level.

Objective

The objective is to improve fault models, fault detection coverage, and improve the performance of automatic test generation techniques. In addition, the desire is to move the process of test technique insertion up in the development cycle to early in the design phase. In addition, developing test generation techniques and fault models at higher levels of abstraction, such as the functional or behavioral level, have the promise to improve the test generation and fault detection execution times.

Methodology

This paper will look at four recent concepts proposed in the recent research literature which propose a solution to different aspects of the problem. Only four representative papers will be analyzed to limit the scope of this work since there are such a large number of papers in the literature. The Unilogic ISU technique, which will be described below, will be employed as an analysis tool to determine how well each the four research concepts address the different aspects of the problem and where further research is required. ISU will then be used to model the problem and a proposal for a new direction in research will be developed.

Recent work in the commercial sector has produced an implementation of numerous research methodologies which address different aspects of the problem. The world leader in DFT techniques is Mentor Graphics. An overview of their DFT tools will also be provided. These represent the state of the art in automatic test pattern generation techniques and fault grading. Although the scope of this paper is limited to analyzing only the four research proposals, ISU can be applied as a follow-on project to some of the Mentor Graphics tools to see if there are ways of improving their performance and achieving 100% fault coverage. Unfortunately, insufficient information on the Mentor Graphics tools, implementation algorithms, and underlying fault models were available to support this paper.

The methodology that will be employed in this paper will attempt to answer the following questions:

- Is there any new (holistic) testing approach that can include different functions which are currently applied in the existing methods?
- Can the new (holistic) testing approach solve the problem of high level fault testing (existence separated approach)?
- What is the general organization and specific functions which are needed to solve that problem?
- Can Unilogic form be one alternative to achieve the holistic testing approach? If yes, what differences in approach can be achieved based on this logical approach and how can it be used to analyze and discover the different types of faults in CMOS VLSI circuits?
- How can it be implemented and what is required? Also, what criteria should be used to test the efficiency and effectiveness of this new holistic testing approach based on ISU?

To reiterate, this paper will not develop a new testing approach but will use ISU and Unilogic to determine a new direction to pursue which will lead to a holistic testing approach which can later be implemented.

Justification (using Unilogic Form ISU as useful for solving that problem):

ISU can be applied to this problem because the environment of test generation has certain global states. Each of the four ISU states can be used to represent the two

variables, fault coverage and level of abstraction, belonging to the universe of test generation. The Unilogic form can provide a logical way to examine the various test generation state units and better understand the rhythmic sequence of the entity. Therefore, a better understanding of what the underlying factors are which influence the ability to detect faults in relation to the level of abstraction can assist in identifying a future direction to pursue in further research of the problem.

Expected Results

As ISU, an analysis technique of Unilogic form, is applied to the test generation problem, several solutions should become apparent for solving the behavioral level issues for achieving 100% fault coverage in Complementary Metal Oxide Semiconductor (CMOS) Very Large Scale Integrated (VLSI) circuits. ISU can then be applied to each of the four different test generation techniques presented. This will evaluate their ability to detect faults and to achieve 100% fault coverage to see where they exist in the universe of test generation. Finally, ISU can be applied to help address the problem and develop a direction for solving it. ISU will not solve the problem but will provide an analysis tool which will provide insight on the dimensions and the dynamics of the problem and the result will be a new direction from which to solve the problem.

Application of ISU

Overall Problem

Test generation for high complexity VLSI ICs is not only difficult but extremely time consuming. Gate level fault and circuit models are cumbersome and not computationally practical for complex VLSI circuit test generation use because of the tremendous numbers of test vectors and large search for solutions. Therefore, there is a need to model faults at a higher level of abstraction such as the behavioral level, and to generate tests based upon these fault models. However, faults unique to CMOS VLSI must be captured and correlated as a basis for behavioral level fault models to achieve high fault coverage. A selected set of four research efforts will be presented to show how close each approaches behavioral level fault modeling and high coverage for CMOS.

Four existing high level test methodologies

- Approach 1 - B-algorithm: A Behavioral Test Generation Algorithm

Cho and Armstrong[7] have proposed the B-algorithm which generates tests directly from behavioral VHDL circuit descriptions using three types of behavioral faults: stuck-at, stuck-open, and micro-operation—all defined by altering VHDL constructs. The behavioral stuck-at (BSA) fault defines two stuck-at faults (SA0 and SA1) for each bit where a bit is defined as a signal, a virtual signal, a fanout stem, or a fanout branch. The behavioral stuck-open (BSO) fault occurs when the value of an assignment (the right

hand side) is not transferred correctly to its target. A BSO fault represents an open circuit in a model and can represent additional functional faults when a control expression handles the assignment statement. The micro-operation (MOP) fault is when an operator is perturbed to another one or selected instead of the correct one such as ADD instead of SUB. Test generation is performed at the behavioral level using a good/bad value pair G/B in place of D or D' for the D-algorithm. The B-algorithm can generate tests with a higher equivalent gate level fault coverage than the Behavioral Test Generator however this was not quantified. The B-algorithm can generate tests for BSO faults which can detect gate-level transition as well as gate-level stuck-at faults. Lastly, it employs two-phase testing to activate BSO faults and to propagate fault effects through control constructs. Undetermined with this approach is the degree to which gate level and process level faults can be accurately correlated at this behavioral level. In addition, many actual physical faults may occur which are not taken into account at such a high level of abstraction with an ADD operation changing to a SUB. In contrast to this approach, the next one uses specific fault modules to modify the VHDL code.

- Approach 2 - System Level Fault Simulation

Sanchez and Hidalgo[8] have proposed a fault methodology in VHDL, independent of the fault model, based on segments. They employ specialized fault injectors which generate a VHDL description for each fault to model the module's faulty behavior and how the module test structure reflect the presence of faults. Each fault injector is a program which modifies the VHDL code in the presence of any particular fault. The result is an efficient simulation of the fault code produced by the fault injectors allowing a system-level test strategy which provides information on the fault coverage obtained by it. The concept allows any fault to be injected as long as a VHDL model for it exists and there are three ways to inject faults in subprograms. *Instancing* allows the same number of faults to be injected as there are call statements. *Sharing* allows all calls to execute the same code. *Elaboration* occurs every time the subprogram is called with different faults injected. The first two mechanisms produce static fault injection and the third, dynamic fault injection. This method appears to be very robust and as long as adequate fault model injectors exist, a VHDL program can be modified to reflect fault injection and only the parts of the VHDL code affected by faults are simulated, hence the efficiency of the technique. A very simple fault simulation is shown which results in 100% fault coverage but it appears too simplistic to determine the actual coverage of this method. In contrast to the first two approaches is the following which uses information at a lower level of abstraction, the control data flow graph (CDFG) and the register transfer level (RTL) to generate test vectors.

- Approach 3 - Behavioral Test Generation using Mixed Integer Non-linear Programming

Ramchandani and Thomas[9] have devised a single stuck-at fault test vector generation technique using the circuit function behavioral description and its mapping into the implementing hardware. The test vectors are obtained by solving a series of

Mixed Integer Non-Linear Programs (MINLPs) and results indicate an order of magnitude speed increase compared with existing gate-level sequential test generation tools. In addition, test generation time remained roughly constant for increasing bitwidths while fault coverage actually increased. However, several faults were not detected by Behavioral Test Generation tool (BTGen) because the faults could not be propagated through the controller to a primary output. If they were, the test generation times would be longer. The circuit used for comparison was the diffeq high-level synthesis benchmark for which they generated test vectors for all single stuck-at faults for each bit of the 11 registers. This circuit employed a relatively simple controller resulting in short test vector sequences. A more complex controller would have skewed the results due to the weakness in BTGen for propagating faults through the controller. BTGen would also be inefficient for test generation of a CISC microprocessor due to the large number of search paths after the instruction decode state is reached. High fault coverage (83-85%) results were achieved in the second experiment using a greatest common divisor (GCD) circuit when test vectors were generated at the boundaries of modules to detect internal faults, however, the types of circuits need to be characterized for which these results can be generalized. Overall, the limitation of this technique is that only faults requiring a single test pattern, such as single stuck-at faults and bridging faults can be supported. This technique cannot generate tests for delay faults or any other faults requiring two test vectors to excite a fault. The next approach uses the behavioral and RTL level for test generation.

- Approach 4 - Test Synthesis in the Behavioral Domain

Papachristou and Carletta[10] have developed an approach for test synthesis in the behavioral domain by inserting Built In Self Test (BIST) structures into the design description in VHDL and synthesizing this to produce a design-and-test behavior which is a circuit which can be run in normal mode and one which can be run in test mode. They use two criteria--controllability and observability--to determine where BIST structures should be inserted into the design. Behavioral analysis and insertion is used to enhance the testibility of the datapath and RTL level analysis and insertion is used to enhance the testability of the datapath and the controller. Their approach tests both the datapath and the controller but employs a unique approach which allows the datapath to be operated according to its design behavior even during test by essentially a pseudorandom number generator at the datapath input and shift registers at its output. All signals, for both data and control, are analyzed for testability. If any are below a certain threshold, insertion is done, using further analysis to determine the optimum insertion point. Results on three synthesized circuits, with transistor counts in the 2000-3500 range, showed increases in fault coverage—as high as 30% more in one controller—but with an increased transistor count and increased critical delay penalty—76% transistor count overhead and 43% slowdown in that same controller. They claim that the controller overhead is minimal since it comprises a small area of the circuit, and the critical delay of the datapath determines the maximum clock speed of circuit. Therefore, their technique provides enhanced fault coverage but may be come with too much of an overhead penalty for circuits with large controllers. The process may also be somewhat cumbersome for

systems containing many signal lines. Lastly, faults are detected using the BIST patterns, therefore no specific fault model was employed, other than deviations from correct operation to detect faults.

- Conclusion

The preceding examination showed that there are many types of faults which can occur in CMOS VLSI circuits, beyond the classical ssf. A sample of current research activities were shown which use fault models at higher levels of abstraction but the degree to which these adequately capture the process defects which create the different fault types is not well correlated. Furthermore, four higher level test generation techniques were discussed which can detect a range of fault types, from the ssf to as many fault types as there are behavioral models for them. The fault coverage data for each approach was not consistently stated so it made it difficult for a comparison based on this figure of merit—especially in determining a method to achieve 100% fault coverage. Overall, no one approach provides a viable mechanism for adequately modeling faults and generating tests at the behavioral level for CMOS VLSI circuits. Therefore, further research is required in this critical area.

State of the Art in Commercial Design for Test Tools

Much recent research in design for test has been implemented in commercial test generation and fault detection tools. Mentor Graphics was named by Dataquest in October 1996 is one industry leader in supplying design for test technology. They have a wide range of tools which perform full and partial scan automatic test pattern generation (ATPG), memory build-in self test (BIST), logic BIST, and boundary scan. Some of these tools are listed in Appendix B.

The Mentor Graphics suite of DFT tools are comprehensive and have incorporated a number of different methodologies to solve the complex problem of automatic test pattern generation and fault coverage during the design phase and at a higher level of abstraction and with good execution times. Unfortunately, all the details of their algorithms, fault models and implementation approach were not available in time to apply ISU and Unilogic to them to determine what areas require improvements. However, that would be an appropriate goal for the next phase of this research project. A list of research papers written by the Mentor Graphics designers are listed in Appendix C for insight into some of the theory and techniques that are used in the Mentor Graphics DFT tool suite. ISU and Unilogic could be used to help improve their fault models, fault coverage, and address any test pattern generation areas which are not well supported by the tool suite. Therefore, this paper will limit its scope to analyzing only the four approaches described above since they represent a range of techniques which support design for testability, several aspects of which are incorporated into the Mentor Graphics tools. Also, since there is so much research in this field, the four approaches will be used to show the utility of ISU for analysis of them.

Use of ISU to Develop a New Testing Approach to Solve Problem

The approach that will be used to help determine a new approach for solving the problem is based upon work done by Rine and Alnakari. The following definitions[11] will be used to explain the concept:

Unilogic is a logic which affords a way to reason about Rhythmic Sequences of an entity. A Rhythmic Sequence is the universal vocabulary of Form Unity that can be represented by binary symbols. Form Unity is the simplest and most abstract form of any entity, representing the Hayawic Interest, ie. The Hayawic form. The Hayawic Interest is the dynamic containing form of an entity, ie. That interest motivated by need in time to define or to recycle its Interest Square Root. The Interest Square Root is the universal vocabulary used to define any Life Interest Cycle of an entity, including the binary operations of negative/positive, inner/outer, mono/poly, open/closed, extreme/moderate, partial/total, etc. The Life Interest Cycle is the system that an entity uses in order to achieve an ultimate goal in a given time period in the framework of the Interest Square Unity. Interest Square Unity is a dynamic logic square that can classify any permutation of the four categories of a life cycle whose binary operations are opposite, contradicted, completed, included, excluded, etc. Aristotle's opposites logic square represent only one variation of the interest square unity. The Paradox of Russell or Cantor also represents just one variation of the interest square unity. Hayawic Logic is based upon the "Unilogic Form" of Alnakari.

These concepts will be applied to the behavioral level test generation problem to help arrive at a solution. First, we will apply the above definitions to the current problem. The entity will be test generation. The Rhythmic Sequences will be the level of abstraction of test generation and the amount of fault coverage. Level of abstraction varies from the lowest level which is the transistor level to the highest which is the behavioral level. Fault coverage includes the ability to detect the fault given the total number of faults that exist in a circuit. These two dimensions will represent the entity as a Form Unity which is the most abstract view. The approach, however, will allow for a analysis of the different components of the entity at lower levels of abstraction in order to zoom into the aspects of the problem. This is based upon the Form Unity of test generation being motivated by the Hayawic Interest to redefine or cycle through its Interest Square Root. This is the universal language which describes the Life Cycle Interest of the entity which is to achieve its ultimate goal of 100% fault coverage in a given time period in the framework of the Interest Square Unity.

The Interest Square Unity will serve as the basis for our analysis of the entity given the four approaches already discussed and will provide us the ability to determine a new approach to solve the problem. This new direction will be obtained through several iterations through the permutations of the four categories of a life cycle whose binary operations are opposite, contradicted, completed, included, excluded, etc. Using the ISU,

we can determine what aspects of the existing four approaches are in the exclusive, inclusive, contradicted, and complimentary categories. We can then isolate those properties which exist in the inclusive state of each of the four existing approaches and attempt to incorporate them into the new holistic testing approach. We will also use the ISU to determine if the new holistic approach satisfies all the requirements of the inclusive state for the ultimate goal of 100% fault coverage in CMOS VLSI. If not, we continue to iterate the ISU using Unilogic to determine what additional capabilities are required. We will use this approach to determine if the new (holistic) testing approach direction can solve the problem of high level fault testing (existence separated approach). We will then define what the general organization and specific functions are which are needed to solve the 100% fault coverage problem. Our objective is to determine a Unilogic form which can help steer further research efforts in the direction which can be one alternative to achieving the holistic testing approach. The differences in this approach compared to the other four will be described and it will be shown how this new holistic approach direction may be used to analyze and discover the different types of faults in CMOS VLSI circuits. Lastly, we will discuss what the expected results would be if the new direction were to be pursued and whether the new holistic approach can be implemented and what would be required. Also, we will describe what criteria should be used to test the efficiency and effectiveness of this new holistic testing approach based on ISU.

Since the ISU will be the primary analysis tool used to help solve the 100% fault coverage problem for CMOS VLSI, it will be described in more detail. The ISU was developed by the Damascus School of Hayawic Logic and offers a systematic method of selecting, analyzing, and solving any problem. The ISU can be shown graphically to contain four global states, with each state representing the possible locations of the entity in its existence. It represents a logic in that the state transitions can be made from one state to another. It can be used as an analysis tool because each state can be used to represent the factors or perspective of the problem. The four states are shown below:

2. CONFLICTING (CONTRADICTORY)	**4. UNIFYING (INCLUSIVE)**
1. ISOLATING (EXCLUSIVE)	**3. CO-EXISTING (COMPLIMENTARY)**

ISU has two coordinate systems. The vertical axis represents inner-outer or low-high. This is used to represent the degree of intensity or the degree of resonance. Inner is low intensity and is passive movement of interest intensity. The horizontal axis represents closed-open or negative-positive. This is used to represent the degree of inclusion of others in the intuitive key interest or degree of circulation (sizing). Open is extended to having multiple centers of interest and closed is limited to only self centered interest.

The combination of any of the two terms from the above define the four states in the ISU. They are:

- UNIFYING is the combination of both outer/high and open/positive
- COEXISTING is the combination of both inner/low and open/positive
- CONFLICTING is the combination of both outer/high and closed/negative
- ISOLATED is the combination of both inner/low and closed/negative

ISU is a logical presentation of the different cycles of an entity. It is a Unilogic square containing four cycles of behavior or analysis related and sequenced to each other. It can be used for problem solving analysis in the following manner: The first step is to isolate the problem by defining it. The second step is to find out what is conflicting with what you want to attain. The third step is to find the co-existing problem which is how or what are the options that can be used to solve the problem. The last step is to unify the outcome which is where the components of the problem will be brought together and solved. The dimensions of the ISU can be changed during the analysis so different aspects of the problem can be evaluated. The two ISU dimensions are resonance and circulatory of interests.

Before we apply ISU to the test generation case, ISU will be used to describe how it can be used to detect a fault. The isolating state, square one, can be used as a filter to detect behavior. In the abstract sense, it receives an input, whether it is a word as in speech processing, an electrical signal as in neural analysis, or a color for code recognition. Initially, the behavior or input is unrecognized, but is processed and if recognized, can move on to another state. If it is unrecognized, the input remains in the isolated state until it can be addressed and resolved. For example, the colors red, yellow, and green, are universally known input codes for the traffic management application, and most people can automatically process these inputs and rapidly move them out of the isolated state. However, this is not always the case. A small child who has not yet learned the meaning of these colors, may receive them at a traffic light and not know what they represent. Until the child learns how to process them, these inputs can remain in the isolated state. For most people, the process is so automatic that the time the input spends in the isolating state is virtually zero before it is processed and resolved so it can move into a different state. However, there may be times when a traffic light is unexpected, or someone is in a day dream, and all of a sudden the traffic light input is seen by the individual. If an unexpected red light is seen, this input will remain in the isolating state for a much longer time, until someone considers the options and takes appropriate action, thus moving the input to another state. However, if a traffic light of the color blue were seen, people would not know the meaning of this and not recognize the pattern to determine what course of action should be taken, causing this input to stay in the isolating state. Depending on their response, it can remain in the isolating state or move into another state.

- <u>Example of Fault Detection Process Using ISU</u>

From this simple traffic light analogy, the ISU can be used to analyze the processing of faults. Faults can be recognized by whether or not they conform to a certain set of rules. If they do, they can be detected automatically and moved into a different state. If not, they can remain in the isolated state awaiting further processing for their detection or they may move into another state incorrectly. Since there are many different fault types, each fault type can be characterized by a set of filters tailored to detect that particular fault type. However, new fault types can occur which do not yet have filters or certain faults may not be detected by the existing filters. These faults require a new mechanism for detection so the fault coverage rates can increase. The following analysis will show how faults are processed by the ISU.

Most fault detection algorithms will activate a fault site with a specific type of fault and generate test patterns which will detect that type of fault. The automatic test pattern generation techniques are usually tailored for specific types of circuits like combinational logic, sequential logic, non-scan latches, circuitry around embedded random access memories, etc. but will result in fault detection if the circuit produces an output which recognizes the presence of a fault. If the ISU is envisioned as analyzing a test pattern generation process, the following is the process undertaken.

The output of test patterns are the input to the ISU. Faults will be detected if they conform to a specific output pattern which detects them. The isolating state, square 1, can be viewed as this filter. If a test pattern recognizes a known fault, then that fault moves from square 1 to square 4, the unifying state because the fault has been detected. Square 1 is a (-,-) state in that it is closed and inner. It is closed (-) in that it is a filter and minimizes the degree of inclusion of others—it is a filter and therefore represents a stopping gate that one must pass through before moving to another state. It is also inner/low (-) in that its interest is in the basic unit or fault site, not in the overall circuit. In contrast, square 4 is a (+,+) state in that it is open and outer/high. It is open (+) because it will now include the known faulty behavior with the correct operation of the other logic in the circuit. It is outer/high because its interest is in the overall circuit. Faults successfully detected in square 1 will move here because the known faulty behavior can now be integrated (included) into the overall circuit.

Square 2 is conflicting (-,+) and is both closed (-) and outer (+). Test pattern results which indicate faults that require a second phase test will move to this square. The second phase test will involve more of the circuit and the potential fault has been identified for further testing. If the second phase test successful in either identifying the fault or determining there is no fault, then the fault then moves to square 4. If the second phase test doesn't detect a fault but a fault actually exists at the site, then the fault remains in state 2 because it represents conflicting behavior. The fault site was tagged as a possible fault but could not be proven as such. The potential fault site is now included in the higher level circuit but can continue to be monitored for faulty behavior.

Square 3 is coexisting (+,-) and is both open (+) and inner (-). Test pattern results which do not indicate a fault but when a fault actually does exist in the circuit, will end up here because the fault site is integrated into the rest of the circuit behavior however, there is an underlying inner fault. This results in an undetected fault coexisting with the rest of the circuit which may result in faulty behavior.

The process above analyzed how faults initially are detected and handled by test pattern generation within the existence of the circuit. During the use of the circuit, the fault behavior may change. This can be represented in the ISU. Each ISU state can be further broken down into four quadrants, each representing the aspects of the inner-outer and open-closed behavior but within the given state. For example, the undetected fault which resides in state 3, coexisting, can be in the upper right hand quadrant if it does not affect the proper circuit behavior. In this way, it is still coexisting because it is an undetected fault, but within that state it is exhibiting unifying properties. If, on the other hand, the undetected fault began to negatively impair circuit operation, then that undetected fault would move to the upper left hand quadrant because it was exhibiting conflicting behavior.

Also, if a new fault was created such as a single event upset which is a temporary change in the value of a logic value, this would move the fault site from the upper right hand quadrant of state 4 to the upper left hand quadrant of state 4 and then back to the upper right hand quadrant. So, within state 4, this type of fault would shift from the unifying sub-state to the conflicting sub-state and back again to the unifying sub-state of the unifying state because the fault was temporary in nature.

This above discussion was meant to give the reader an understanding of how ISU can be used to categorize the behavior of individual faults and help determine the behavior of faults in the context of the environment of a circuit.

ISU is very flexible, however. It can be used to analyze different aspects of a problem. The ISU will now be applied to the abstract test generation case, each of the four existing approaches, and to achieve a new holistic test generation approach.

- Application of Interest Charge Elements (ICE) to Fault Detection

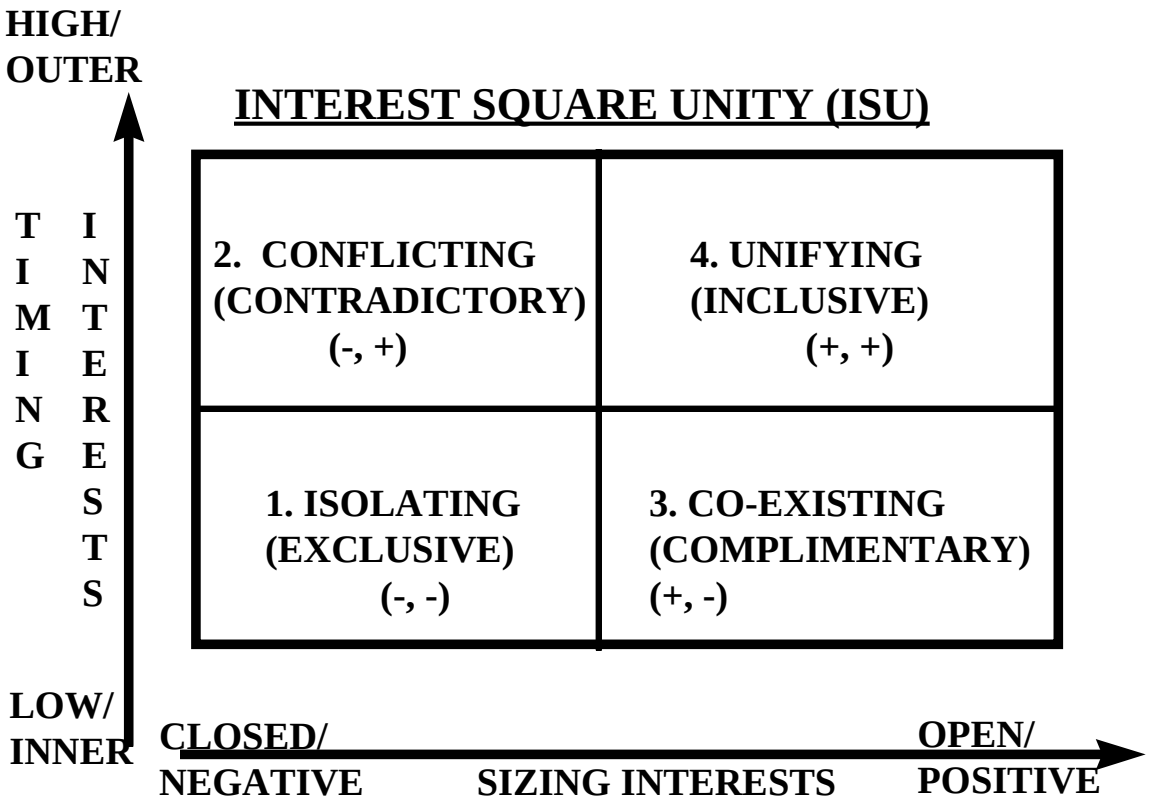
The concept of ICE can be applied to the ability of a test generation algorithm to detect faults. According to Alnakari and Rine[12], the dimensions of space and time can be used to analyze the communication process and the way a software specification can change from initially satisfying the end user, to no longer being applicable once developed because of the aspects of space and time altering the end user requirements. This same concept can be applied to fault detection. Test generation algorithms once developed to detect specific types of faults may no longer be effective in detecting new types of faults or permutations of faults based upon changes in process technology or design techniques. Therefore, test generation algorithms can be analyzed by the ICE to determine what problems may be expected to occur over space and time. The entity

described by a specific test generation algorithm using certain fault models include Hayawic basic interests involving the two interrelated dimensions of space and time. Each dimension represents a logical basis for systematic behavior. Time-oriented logic is a rhythmic systematic frequency and space-oriented logic is a size systematic circulation. These dimensions can used used to determine what causes the test generation algorithm to be unifying or coexisting in its ability to detect faults and what causes a divergence in the space and time dimensions to result in isolated or conflicting behavior.

Hayawic Form Unity provides a basic UniLogic Form by which all things in the universe can be represented. Hayawic Form Unity uses two Cartesian coordinates with the first being the Timing Resonance dimension which is the vertical factor represented as need or demand rhythm. This can be used to express the desire or goal of the test generation algorithm to detect as many faults as possible in order to achieve 100% fault coverage. This is measured in a framework of binary interest roots indicating a high or low speed of interest resonance. This dimension can be used to determine the percentage of actual faults detected and represents the fault coverage factor. The second coordinate is the horizontal factor which is the Sizing Circulation dimension represented as spacing of life interest. This is measured in a framework of binary numbers indicated as negative or positive location of interest circulation. This dimension can be used to measure the range of fault types tested by the behavioral level test generation algorithm. Positive implies a large range of fault types can be supported and negative implies a low level, such as being able to test for only one or two types, such a SSA fault.

These binary symbols can be represented as signs where one is plus and the other is minus on each dimension, since Form Unity has a midpoint as zero or neutral, and the entity can be represented as a point on the continuum at a certain place displaced from the midpoint.

The pictorial representation of the ISU is shown below and each quadrant will be explained in more detail.



The Resonance and Circulation of Interest produce a Cartesian product forming Interest Square Unity (ISU). The different portions of the ISU represent different cycles of interest. In a clockwise direction, square one is the isolated cycle, square two is the conflicting cycle, square three is the coexisting cycle, and square four is the unifying cycle. The ISU symbolic notation of universal logic will be shown how it represents the connective symbolic logic. The isolated cycle of square one has the symbol not-/ for negation because Resonance is at a low level and has closed (negative directed) Circulation. This is because it represents the process of excluding incoming ICEs or breaking them down by differentiating or diminishing them. This isolating operation represents the inability of a behavioral level test generation algorithm to detect the few faults that it can support. This implies a very low fault coverage because only a small number of the different fault types can be tested for and of those, very few can be detected. Therefore, this square is minus-minus.

The second square representing the conflicting operation expresses the union of alternative interests, either + or -, and this is where the Resonance is high but it is still negative or closed circulation. This is described as a contradiction and it is an opposing operation in the form of resistance of the interest charge element input. This extends the initial negative operation within a paradoxical positive union to partially favor or support one of two options in a non-stable final stage of interest. We can represent this state with high levels of fault coverage demonstrated on only a few fault types by a high level of abstraction test generation algorithm. Although the fault coverage is very high for a small

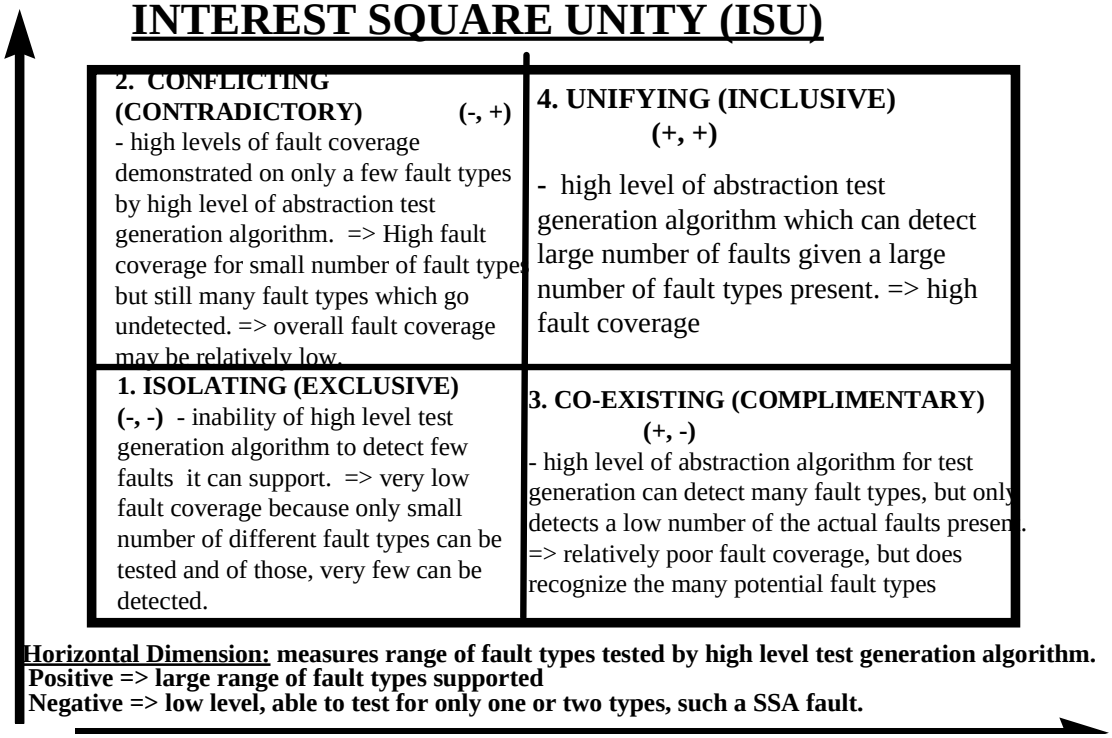
number of fault types, there are still many fault types which may go undetected. Therefore, the overall fault coverage may be relatively low. Therefore, the resistive nature of certain fault types to detection by this algorithm can be symbolized by the minus expression and the OR operation. The Cartesian product of square two is minus-plus.

The third square representing the coexisting operation expresses an intersection because in this cycle the Resonance is low but there is open (positive directed) Circulation. This is a complimentary operation interest of the entity because it symbolizes a positive approach which is in harmony with the ultimate goal. This square represents a high level of abstraction algorithm for test generation can detect many fault types, it only detects a low number of the actual faults present. Despite the relatively poor fault coverage, this case represents the supportive and compatible aspects associated with a technique which does recognize the many potential fault types and is on the correct path towards supporting the ICE input but requires finer tuning or improvements in fault detection and thus fault coverage. The coexisting cycle operation may be symbolized by the plus expression and the AND operation. The Cartesian product of square three is plus-minus.

The fourth square representing the unifying operation is the implication expression because the Resonance is higher and the Circulation is open (Positive directed). This is the if->then operation related to different cycles in which the ICEs should be inclusive, amplifying, and common space interests. The Unifying cycle operation can be symbolized by * because it is amplifying and multiplying. This square represents the high level of abstraction test generation algorithm which can detect a large number of faults given a large number of fault types present. So, if there is a fault, then it probably will be detected for all ICE inputs. The unifying cycle operation may be symbolized by the times expression and the If-then operation. The Cartesian product of square four is plus-plus.

As described above, the ISU tailored for analyzing the high level test generation entity is shown pictorially below:

Vertical Diemsion: **- goal of high level test generation is to detect as many faults as possible => determines the percentage of actual faults detected and represents the fault coverage factor.**



The ISU can be viewed as a balanced charge of interest. Two interest charge elements in the form of positive and/or negative as a Cartesian product but in different amount of charge notation as (-,-), (+,+), (+,-), or (-,+) are included in every state.

The ISU can further be divided into two modules, with the Positive module including the (+,-) and (+,+) which consist of a set of three positive ICEs and one negative ICE in one distinguished module. In comparison, the Negative module includes (-,+) and (-,-) which consists of a set of three negative ICEs and one positive ICE in one opposite distinguished module related to the Positive making this balancing.

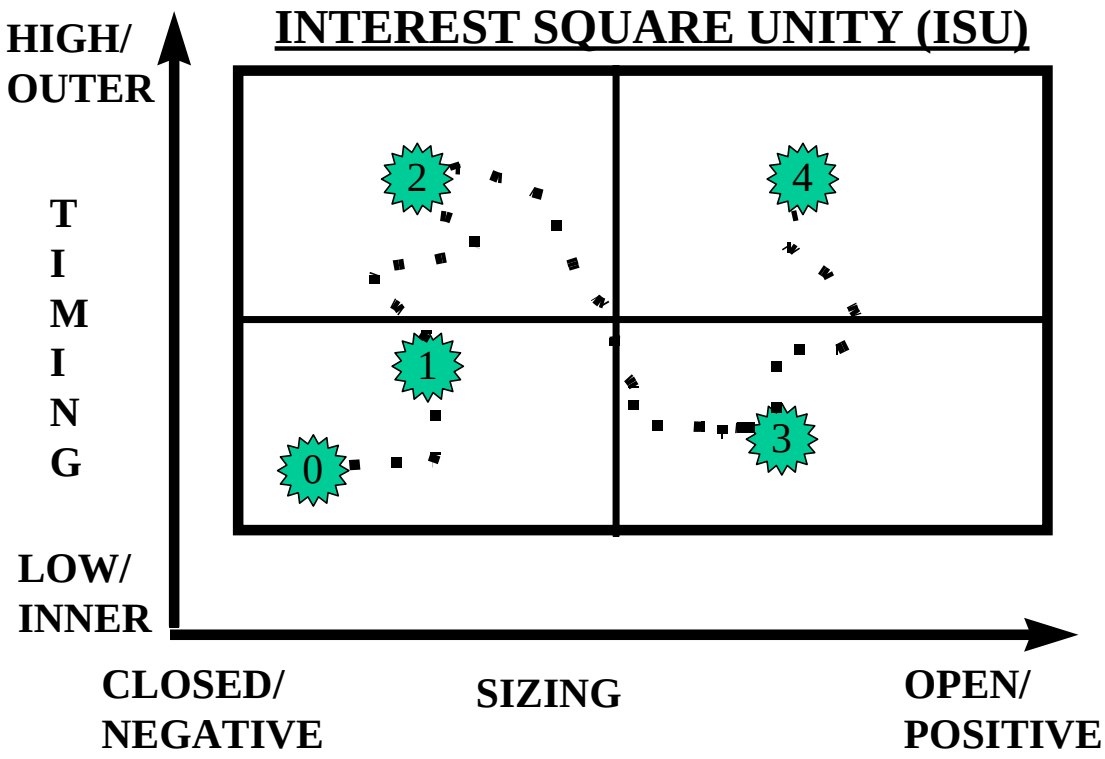
Each module includes two states of transition where a transition is the transfer from one state to the other. A transition is actualized when a state is stimulated from an inside and/or outside event. An event may or may not be effective depending on its ICE quantity and quality. If the level of the charge is small or causes a bias in filtering the meaning, purpose, or processing required to make threshold of reception to cause access into any of the two modules of the ISU, then this is called the neutralization process. This Neutral interest element is halted in the filtering process because they are non-transmittable but provoke a neutral transition which causes no positive or negative response. This can be considered as either a non-focused or confusing event.

Given a high level of abstraction test generation algorithm, it can transition from one ISU state to another depending on certain factors. If the complex IC is of a type which only contains one fault type, such as the SSA fault, then the algorithm would be in

square 2. If however, the same algorithm were applied to another complex IC, such as CMOS, which contains other fault types, such as the bridging fault, the algorithm would move to square 3, since it could only detect one fault type and there are many others contained in the IC. Therefore, the overall fault coverage would be relatively low. As another example, if a high level of abstraction test generation algorithm was designed to detect all known faults types and successful in determining them, it would be in square 4. However, if the IC process technology were to change in such a manner as to introduce new faults types that the algorithm could not detect, then the fault coverage would decrease and the algorithm would transition to square 3. The Neutral interest element can be perceived as deviations in the circuit, perhaps caused by subtle processing nuances or design changes, which may cause an effect on the test generation algorithm, either causing it to be slightly more or less effective but not cause any significant change in its performance, thereby keeping it within its existing ISU module.

The execution of a high level of abstraction test generation algorithm will begin in the isolated state when no fault types are detected and no fault coverage occurs. As it begins to perform the tests, it will either remain in square 1 or it will move up to square 2 if it starts detecting a large number of only one or two different types of faults. It can also transition instead from square 1 to square 3 directly if the algorithm can detect many different fault types in parallel. If the algorithm is successful in continuing to detect faults from different fault classes and reaches a high level of fault coverage across the board, then it will transition to square 4. If the test generation algorithm uses a sequential approach in testing for different fault types, it can begin in square 1, move up to square 2 after covering a large number of a certain fault class, then if it begins to detect additional fault types, it will begin to shift towards square 3 and eventually move into it. Once in square 3, if the algorithm performs well in covering the additional fault classes, then it will transition into square 4. The actual path transitioned through the ISU by the execution of the high level of abstraction test generation algorithm will depend on how it is designed to detect faults, whether it focuses on one type first and once completed with that type begins testing for another type of fault (sequential fault type execution), or if all faults types are tested at once (parallel fault type execution). The path transitioned through the ISU described in the previous example by a sequential fault detection algorithm is shown below.

ONE EXAMPLE



Now each of the 4 approaches will be analyzed using the ISU, as follows:

- **Approach 1 - B-Algorithm: A Behavioral Test Generation Algorithm**

Critical Analysis

The degree to which gate level and process level faults are captured is not clear with this approach. First, no quantitative fault coverage numbers were given—just that the fault coverage numbers were better than the previous Behavioral Test Generator (BTG) approaches. Secondly, whereas the BTG uses goal trees this approach leverages off the D-algorithm which finds test patterns for any single stuck at fault in any non-redundant combinational circuit. The extensibility of some of the underlying assumptions when applied to VLSI may be questionable. Thirdly, not all VHDL constructs like loop statements were considered because of the degree of complexity in generating tests for them. Lastly, the micro-operation fault is based upon two premises; the decoder error can change an add to subtract or a defect on the carry out line. A decoder error can change an add to a number of operations, not just a subtract and depending on how the adder is implemented, you would need to either complement one operand and add a carry

one in 2's complement to change an add to subtract—not a single fault on a carry out line to change an add to a subtract. For a sum of products implementation, you would have to complement one of the operands to each of the carry outs so there is not a good correlation between defects and some of the behavioral level faults in this approach.

Since the above critical analysis was performed, the B-X TPG Algorithm, based upon the B-Algorithm, was implemented[13] on several simple circuits and on a few benchmark circuits. The fault coverage results ranged from 76% to 100% on the benchmark circuits. These results were obtained by generating faults for the circuit based upon its VHDL description and determining which of the faults could be detected. Since the B-X Algorithm implementation code was not available in time for this project, only the original B-Algorithm will be analyzed with ISU.

ISU Analysis

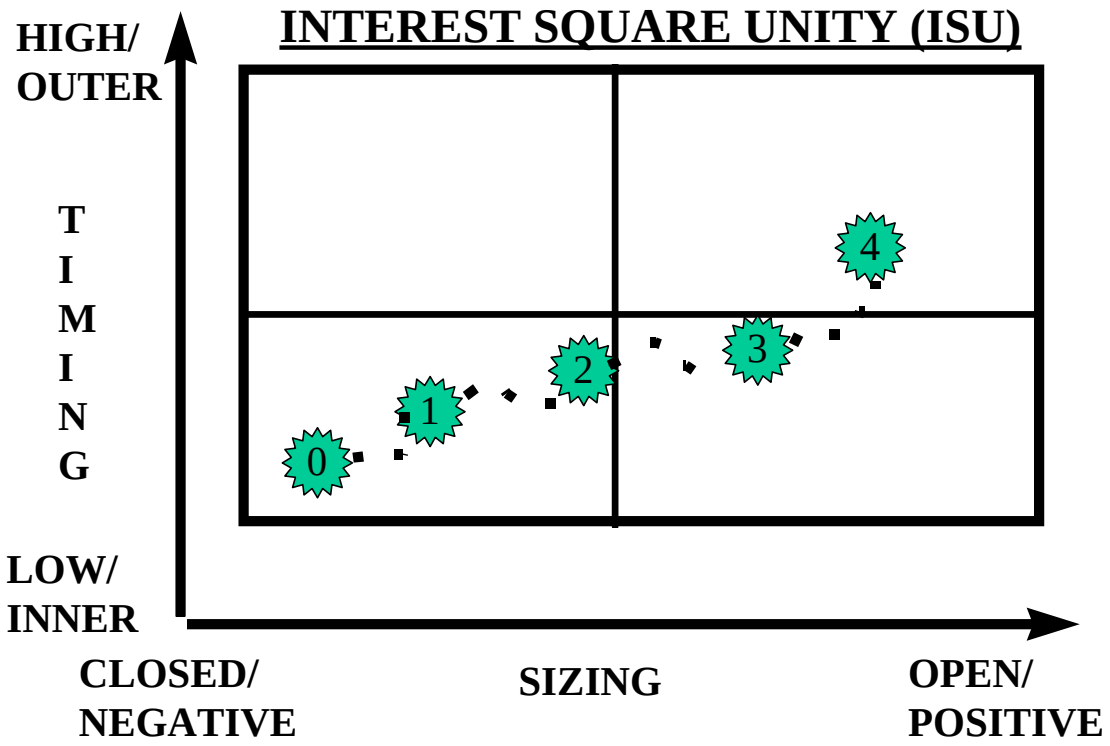
The B-algorithm assumes the behavioral fault model is a single fault model which means that only one behavioral fault occurs at any one time. The B-algorithm generates a test for a given fault through fault activation. This consists of creating a good/bad value pair at the fault site, propagating the good/bad value pair to a primary output and justifying the conditions necessary for the activation and propagation to primary inputs. The B-algorithm consists of three phases: initialization, first test, and second test.

During initialization, all signals are set to x, the two-phase flag is reset, and can perform circuit initialization. The two-phase flag indicates that two-phase testing is required. During the first phase, good/bad value pairs are created by activating the specific fault type under consideration and the algorithm performs the completion procedure appropriate for that particular fault type. If no flag for two-phase testing is set and the first test phase is successful, the algorithm reports success and stops. If this first phase is not successful, failure is reported and the B-algorithm stops. Two phase testing, a testing strategy where a fault is detected using two consecutive test sequences, is performed only if the two phase flag has been set during initialization. If so, a new good/bad value pair is created If this second phase is successful, the B-algorithm reports success, otherwise it reports failure.[14]

The execution of the B-algorithm will begin in the isolated state when no fault types are detected and no fault coverage occurs. The B-algorithm generates fault test sequentially because only one behavioral fault can exist at any one time. It probably can generate tests for any one of the 3 behavioral fault types in any order. Assuming a random distribution of the order of the fault types tested, the cumulative results of the algorithm will move from square 1 toward square 3 with a positive slope. If it can detect a large number of faults, it will go to the upper region of square 3 and start to enter square 4, still with a positive slope. When the second phase tests are performed, the algorithm can continue increasing it's fault coverage and end up in square 4, still with a positive slope. Since the algorithm may not be successful in all phases, or in any phase, it may

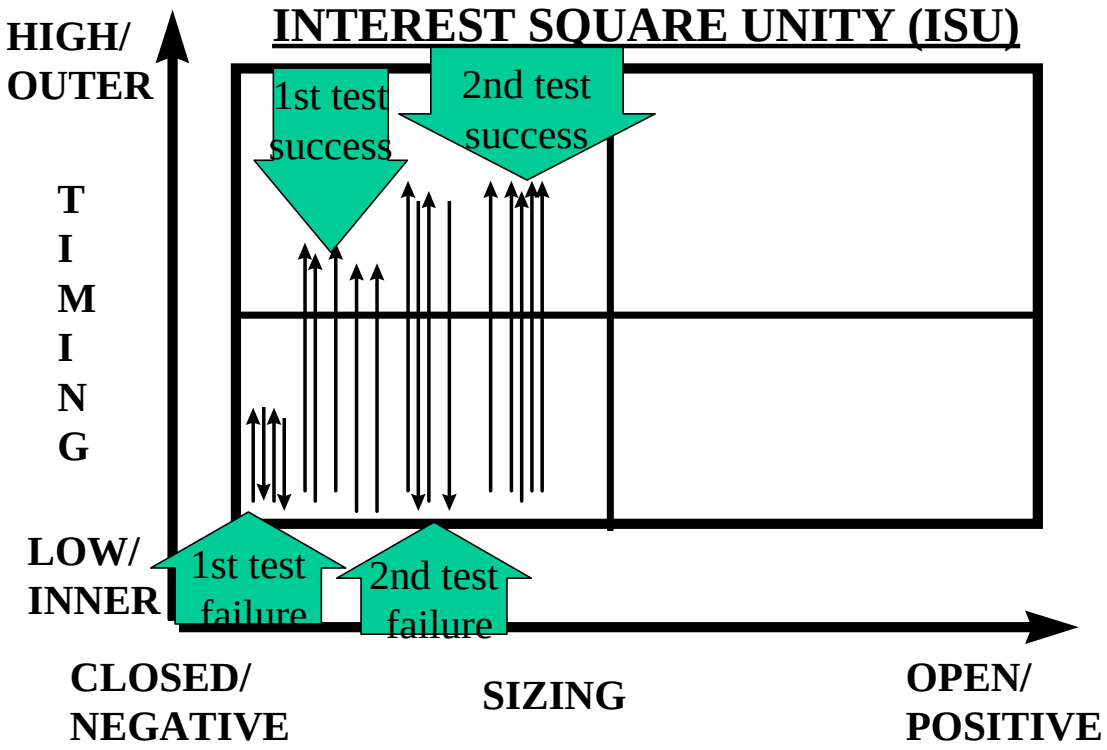
stop execution of that particular fault type at the specific location under test. The diagram below shows the B-Algorithm aggregate example.

B-ALGORITHM AGGREGATE EXAMPLE



The algorithm during its execution actually is comprised of segments, with one for each fault type tested at each location. So, if 100 fault sites were tested for the same type of fault, ie. a behavioral stuck at (BSA) fault, 30 sites may fail the first phase test, each segment created by the algorithm test for a specific fault at a specific site with the execution aborting after a failure is reported. This means that the segment never got out of square 1. In that case, 30 short segments remain within square 1. If the remaining 70 sites report success after the first phase test and only 20 sites set the two-phase flag, then those 50 complete successes can be represented by longer individual segments starting in square 1 and ending in square 2. They end up in square 2 and not square 3 because we are still only testing for one type of fault. If only 10 of the 20 sites report success after phase two testing, then those segments end up in square 2, otherwise, the failures end up in square 1. But as the algorithm starts testing for the two other fault types, although each fault type ISU performance would be represented similarly to the case just described, the aggregate algorithm performance will shift right into squares 3 and 4 due to the increasing numbers of fault types detected and the improved fault coverage. Since no specific fault coverage figures were provided, it is not clear whether the algorithm will end up in square 3 or in square 4. The diagram below illustrates the execution of a single fault type at multiple fault sites as described above.

B-ALGORITHM TEST OF ONE FAULT TYPE AT MULTIPLE SITES EXAMPLE



- **Approach 2 - System Level Fault Simulation**

Critical Analysis

This approach injects faults by modifying the VHDL code in the presence of a fault. It can inject any fault as long as it is available in VHDL. The process alters the normal kernal process by breaking up the process into segments with only one segment per process is queued or active, the rest are waiting or non-queued. This segment approach allows input and output signals to be perfectly defined whereas in a process they depend upon a wait statement in which their execution is resumed. Segment outputs depend solely on inputs. This approach is similar to a logic gate level fault simulation. When a segment can model both a fault and a fault-free case, it remains a single segment. If not, it is broken into two segments. An example using four segments with a fault injected into segment two and another fault injected into segment four resulted in 100% fault coverage by identifying both faults by the execution of the fourth segment. Fault coverage numbers of 100% appear to be unrealistic for larger numbers of segments and were obtained using a very simple test and a longer run was needed to develop more realistic fault coverage numbers.. The fault coverage may degrade if more complex

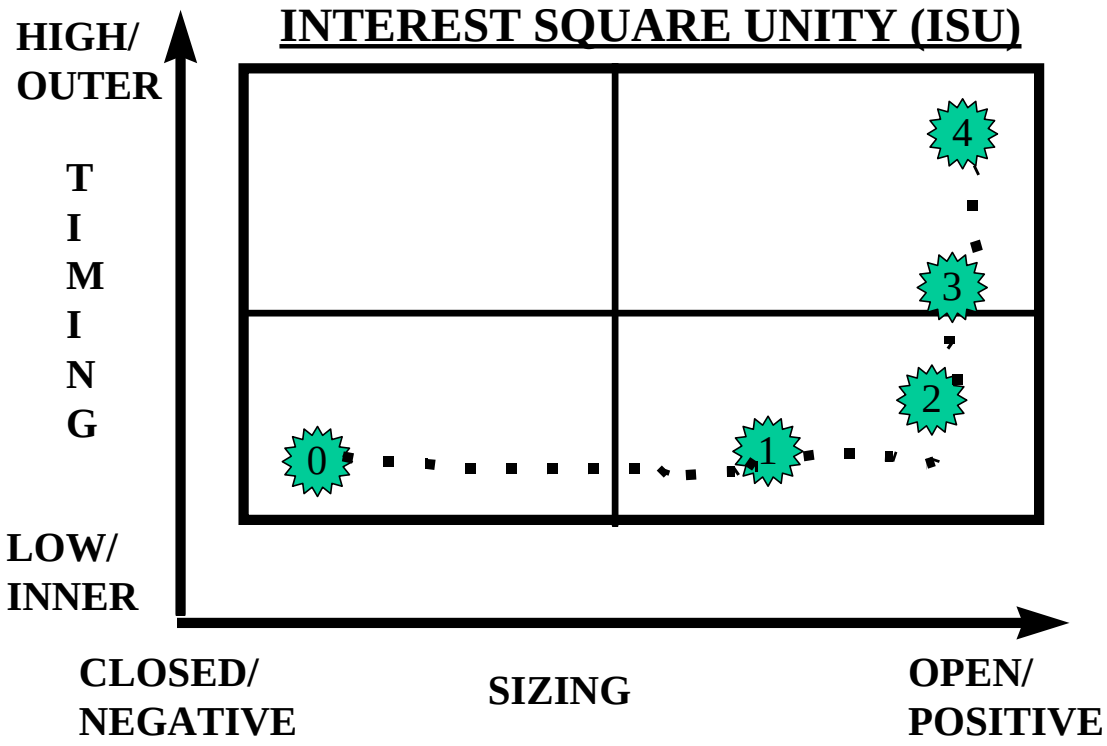
devices were tested. Overall, this approach has merit because it occurs at the VHDL level but the fault injection success depends on the quality of the fault models used. This appears to be an efficient process but more analysis is needed to determine the effect of breaking the process into segments and what effect loops and changes in the flow of control have on the execution and fault simulation.

ISU Analysis:

This simulation uses a VHDL description of the circuit plus VHDL descriptions of the faults to be injected into specific points in the circuit. Each fault injector can represent multiple faults so concurrent fault simulation is supported. For example, one stuck-type fault and one architecture fault can be inserted into a VHDL description of a NOR logic gate. This combination would be considered as a single fault by the fault simulator but in actuality would represent several different fault types. The concurrent fault simulation simulates the system using the segments from the fault-free model. When a segment having a different code for specific faults is encountered, both codes are executed resulting in signals generated from both the faulty and the correct model. The first step in the fault simulation process is to extract the segment. The segment description is then modified by fault injectors. Then, the simulation begins. The simulation begins with the initialization phase during which time the objects take on their initial values and those segments which are associated to the begin statement are activated. After the first segment completes its execution, the next segment to be run is enqueued. If the injected fault affects the second segment, then both the faulty and fault-free versions of the second segment are enqueued. They can each produce different results and can enqueue different next segments. Therefore, one path will have a faulty output and the other will have a fault free output. This process continues until either the objects of the faulty and fault-free models take on the same value or when the fault is detected.[15]

Applying the ISU technique to analyze this system level fault simulation approach, upon execution start, this approach will begin in the isolated state when no fault types are detected and no fault coverage occurs. Assuming that multiple fault types have been injected into the VHDL code since this approach can support concurrent fault simulation, this process will move from square 1 quickly into square 3, with a very small positive slope. As it continues to perform the fault tests, it will move up within square 3 as it starts detecting a large number of different types of faults. If the algorithm is successful in continuing to detect faults from different fault classes and reaches a high level of fault coverage across the board, then it will transition to a high level in square 4. This path is shown in the figure below. If high fault coverage is not achieved, then the algorithm will end up either in square 3 or on a lower level of square 4.

SYSTEM LEVEL FAULT SIMULATION EXAMPLE



- **Approach 3 - Behavioral Test Generation Using Mixed Integer Non Linear Programming**

Critical Analysis:

This technique starts with a behavioral level description and translates it to a control data flow graph where nodes represent sequences of executable instructions and arcs represent the flow of control between nodes which is then mapped to an RTL structure and schedule. This includes hardware modules like registers and functional units and their interconnections and the operations performed by each hardware module in each clock step. This approach leverages off the path tracing and symbolic execution techniques used in the software domain for over 20 years and applies them to hardware testing by linking it with hardware information from high level synthesis. Test Vectors are generated from the control data flow graph and RTL descriptions by modifying the conditions under which a module is to be tested to reflect a fault and those conditions are transformed into a mixed integer non-linear program to be solved by a commercial solver like GAMS. The shortest path algorithm was used to both sensitize the fault and module and to propagate it to primary outputs. Two circuits were tested: differential equation high level benchmark and a greatest common divisor. Results showed roughly an order of magnitude improvement in execution time compared with gate level test generation

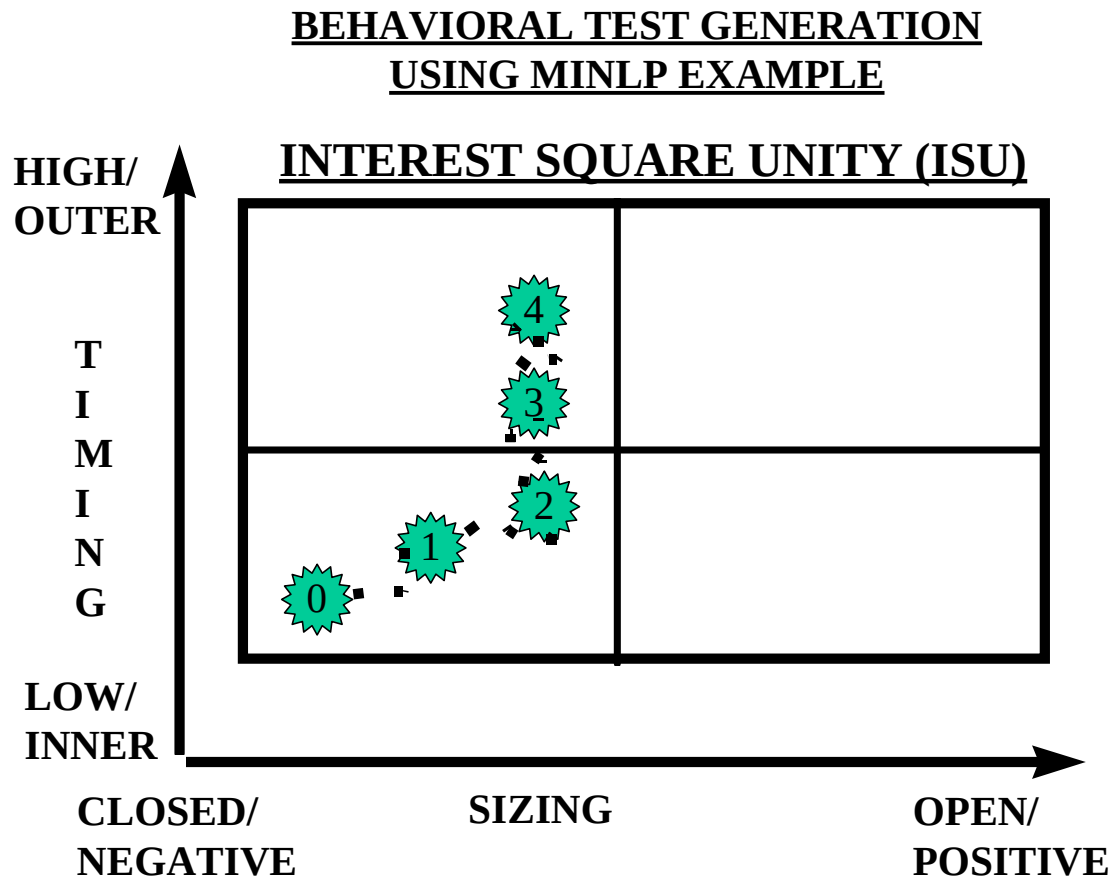
results. And as bit widths increase, much better fault coverage occurred compared with gate level test generation methods. The only limitation was that only faults with single test patterns, such as single stuck at faults and bridging faults were supported. Delay faults were not supported because they require two test vectors. Also, only faults in the data path were tested, not faults in the controller. In addition, fault coverage in the gcd was 83-85%. Lastly, this technique is inefficient for CISC microprocessor circuits because after instruction decode, there are too many search paths to arrive at a timely solution. However, several faults were not detected by Behavioral Test Generation tool (BTGen) because the faults could not be propagated through the controller to a primary output. If they were, the test generation times would be longer. The circuit used for comparison was the diffeq high-level synthesis benchmark for which they generated test vectors for all single stuck-at faults for each bit of the 11 registers. This circuit employed a relatively simple controller resulting in short test vector sequences. A more complex controller would have skewed the results due to the weakness in BTGen for propagating faults through the controller.

ISU Analysis:

This behavioral test generation techniques has two major steps. These include: one, generating conditions, and two, generating test vectors. The first step obtains the conditions under which a module can be tested. The second step modifies the conditions to reflect a fault in the circuit depending on the selected fault model and the conditions are transformed into a mixed integer non-linear program (MINLP) and are solved using a commercial tool. The work as published was limited to only testing for faults within the datapath and not the controller so the total number of faults will not be detected. Fault sensitization identifies the conditions required on the primary inputs to sensitize a fault in the module under test. A shortest path algorithm is used to minimize this path. The next step is to find a path which can be used to observe the fault at some primary output and must propagate the faulty value based on the other signals it will encounter. Again, a shortest path algorithm is used to propagate this sensitized value. Both of these shortest paths result in a complete test path from the primary inputs to the primary outputs. This process of fault propagation and sensitization does not assume any fault model and the conditions generated are valid for a fault-free circuit. These conditions can be modified to reflect any fault requiring a single test pattern. To excite the fault. Therefore, single stuck-at faults and bridging faults can be supported. Faults requiring a sequence of patterns, such as delay faults, cannot be supported. The conditions are modified to reflect a fault by introducing two symbolic variables to represent the good and faulty value of the fault site in the selected module and all subsequent conditions are updated to reflect the fault. A new GAMS (commercial MINLP solver) model has to be generated for each fault in each module in the RTL description. Solving all the GAMS models produces all the test vectors for the circuit.[16]

The execution of this behavioral test generation algorithm will begin in the isolated state when no fault types are detected and no fault coverage occurs. As it begins

to perform the tests, it will remain in square 1 and not move into square 3 because it can only detect a limited number of faults—those requiring a single test pattern. As it starts detecting a large number of only one or two different types of faults, it can transition from square 1 to square 2. If the algorithm is successful in continuing to detect faults from the few fault classes it can support and reaches a high level of fault coverage it will transition to a higher level in square 3. Because the algorithm cannot detect faults in the controller—it can only detect faults in the datapath—it will never reach the highest levels of square 2 because it cannot detect all the faults that exist in the circuit. The path transitioned through the ISU by this algorithm is shown below.



- **Approach 4 - Test Synthesis in the Behavioral Domain**

Critical Analysis:

This approach which inserts BIST into the VHDL design and after high level synthesis, using RTL and behavioral information, produces a hardware description which can be run in normal or test mode, selected by a switch. Instead of segmenting the structure in the datapath, they maintain it as a whole by placing a test pattern generation register at the input and shift registers at the output, to test for faulty behavior. This

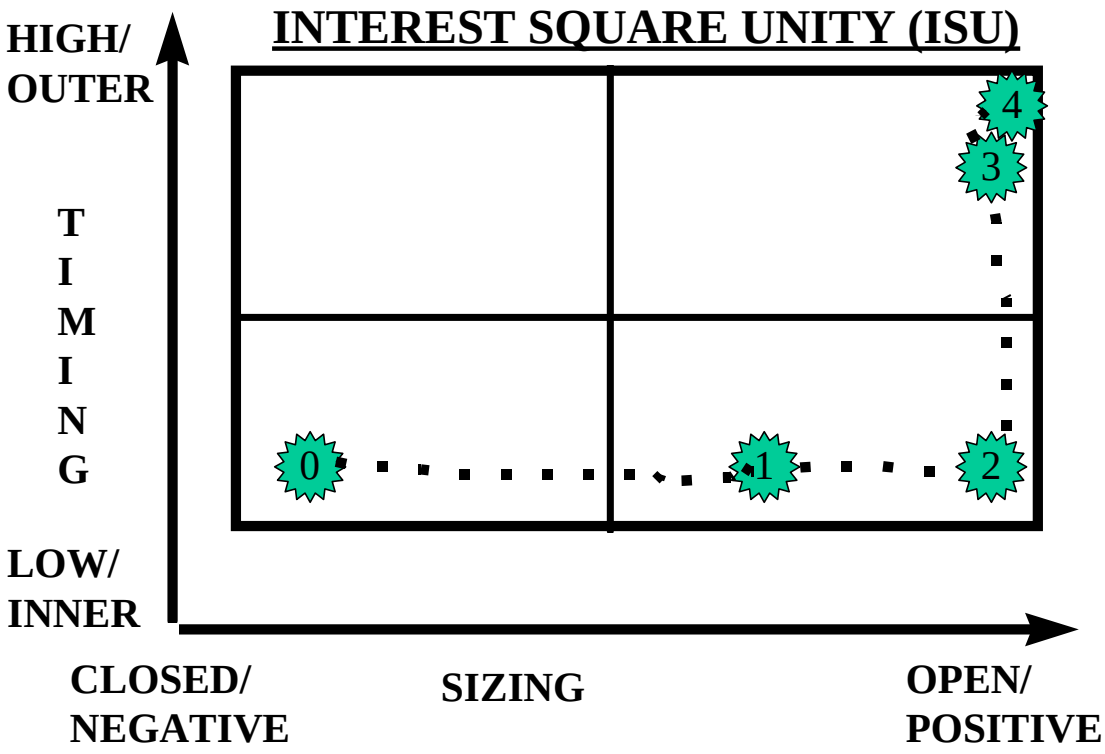
concept, which modifies the VHDL code, certainly employs DFT at the behavioral level. Overall, this approach results in a more easily testable design and allows the testability of he complete physical implementation of the behavioral description including the data path, controller, and interface between them. Drawbacks include increased area and decreased performance overheads which may not be acceptable to large, complex ICs. No specific fault models were specified so faults were detected from deviations from correct operations. The fault coverage was dependent on how well the BIST test vectors were selected in determining the faults.

ISU Analysis:

This technique inserts BIST into the VHDL description and generates a test behavior supporting both the datapath and the controller from a given design behavior. This is accomplished using three test sessions. The first test session detects any faults on the reset lines of flip-flops to determine if any faults would prevent proper initialization of the state. The second test session uses a behavioral test scheme to detect faults within the datapath. This is done in both design and in the test mode. The design mode is used to ensure the multiplexors that switch between the two modes are fault free. The third test session tests faults in the combinational logic of the controller. This includes the state and the control signal logic. This technique allows for a high fault coverage in both the datapath and the controller. No mention was made of the types of faults detected but very high fault coverage figures, approaching 100%, were achieved.[17]

The execution of this test generation algorithm will begin in the isolated state when no fault types are detected and no fault coverage occurs. As it begins to perform the tests, it will move quickly to the right from square 1 to square 3 as it executes the first test session. However, it will stay very close to the bottom of these states since it is only detecting initialization faults but appears to test for many different types of faults. Once it enters the second test session, it will quickly rise through square 3 and into square 4 beacuse it is testing the datapath. Since the datapath represents the majority of the circuit, it should contain the majority of the faults. Since the algorithm achieves high fault coverage, the algorithm quaickly changes state.. When it begins to execute test session three, it will rise slightly more within square 4 and approach the upper right hand corner. This slow rise occurs because although there is high fault coverage, the controller represents only a small portion of the logic of the circuit and therefore would contain only a small number of faults. This algorithm progressing through the ISU is shown in the diagram below.

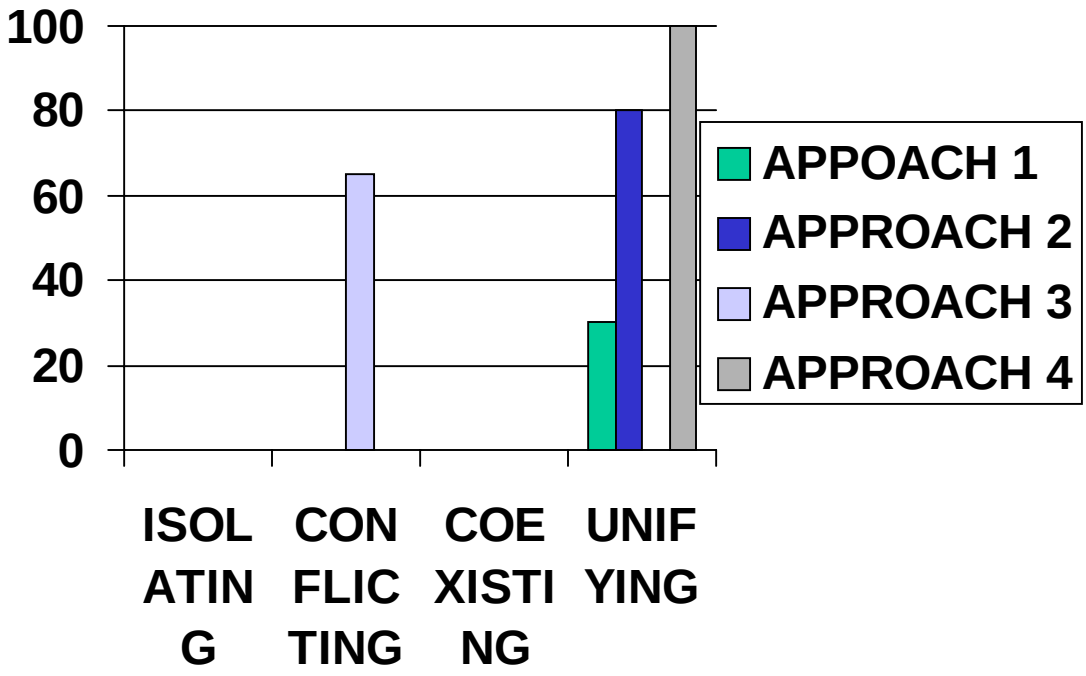
TEST SYNTHESIS IN THE BEHAVIORAL DOMAIN EXAMPLE



- **Results**

Based upon the ISU analysis of the four previous approaches, the following graph pictorially displays the ending ISU state of each algorithm execution, and the vertical height it was able to achieve within each state shown as a percentage. All but the third approach ended in state 4, the unifying state but the height within the state was different for all. Not shown on the graph is the horizontal displacement within each state. This would indicate which sub-state within the ending state each technique arrived at. The third approach ended up in the conflicting state because it could only detect a small number of fault types, which was the sizing dimension analysis criterion.

Overall, the results are promising with the best ISU performance, given the criteria, achieved by the fourth technique. However, this fourth technique, despite high fault coverage data, did result in circuit performance degradation and an increased area penalty, which may not be desirable factors.



The results from this ISU analysis have shown the different paths each technique employs in the process of determining how many faults can be detected in a circuit. By better understanding a fault and fault detection techniques through an ISU analysis, better fault coverage can be achieved. Although it is out of the scope of this paper to develop a new holistic fault detection technique, a few starting points for further ISU analysis will be discussed as a way to achieve 100% fault coverage for CMOS VLSI.

- **New Holistic Approach Direction**

The ISU uses the following operators: +, -, *, and '/'. It also uses the logical operators AND, OR, IF THEN, and NOT. Two of the approaches analyzed above, Approach 1: the B-Algorithm (implemented by the B-X TPG Algorithm) and Approach 4: Test Synthesis in the Behavioral Domain can be modified and combined to include the ISU techniques in order to test for faults. The entire ISU can be applied to test for faults. It can perform four tests on a fault site, AND, OR, IF THEN, and NOT. If the result for each test passes, then the result is 100% fault coverage and enters the unifying state. If only three of the four tests pass, then the result is still 100% because of the majority vote and the result also enters the unifying state. If only two of the tests are passed, then the result enters the coexisting state. Lastly, if only one test is passed, the result enters the conflicting state. If no test is passed, then the result enters the isolating state. All entries in the isolated state are regarded as a fault. The entries in both the conflicting and the coexisting states are regarded as potential faults and may need a second phase test to confirm this.

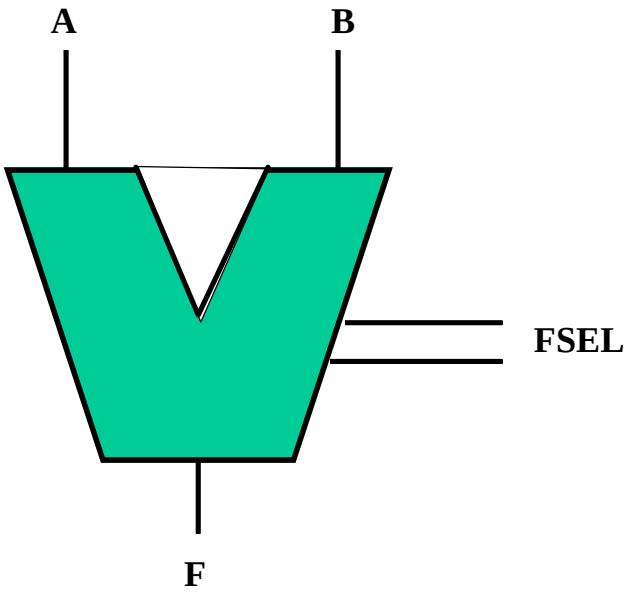
The hardware to implement the ISU is an arithmetic logic unit (ALU) and can be described in VHDL by the following form for a 1-bit input where we have implemented the IF THEN case as an exclusive NOR (XNOR):

```
ENTITY alu IS
PORT (A,B: IN bit_vector (0);
        FSEL:  IN bit_vector (0);
        F:  OUT bit vector (0));
END alu;

ARCHITECTURE data_flow of alu IS
BEGIN
process (A,B,FSEL)
BEGIN
CASE FSEL IS
        when “00”=> F<=NOT A;
        when “01”=> F<=A+B;
        when “10”=>F<=A AND B;
        when “11”=>F<=A XNOR B;
END CASE;
END process;;
END data_flow;
```

This can be implemented in the circuit as part of the test mode, similar to Approach 4 above so that a switch can determine when the circuit will operate in the test mode and perform the ISU testing. Otherwise, the circuit can operate in the normal mode. There will be a hardware penalty for inserting the ISU test structures, but the intent is to minimize the area penalty. The locations for inserting this structure will have to be chosen carefully and ensure enough signal lines can be connected to it without impairing the performance of the circuit.

A simple example of how this can be implemented is shown below. A follow-on research project will need to perform the appropriate analysis and implementation considerations to determine how best to optimally insert the ISU technique into ICs for test pattern generation and fault detection and which fault types can be supported.



ALU IMPLEMENTATION

To test for x stuck at 1 (SSA1), set x=0, which means set A=0 in the ALU for the Good value and B=1 for the Bad value (as in the G/B pair of the B-Algorithm). The following are the function select signals:

FSEL	F
00	NOT A
01	A + B
10	A o B
11	A XOR B

We get the following output the ALU operation:

A	B	NOT A	A+B	AoB	A XNOR B
0	0	1	0	0	1
0	1	1	1	0	0
1	0	0	1	0	0
1	1	0	1	1	1

If we let the ALU be a built-in test structure, for an a signal to be tested for a single stuck at 1 fault (SSA1) we apply an input on x=0. The x signal line is tied to the ALU A input. We also apply an input of B=0, which is opposite to the fault we are testing for on x. The result will be an output which differs in three values from the four values of the correct output if the x signal line is stuck at 1. This means that the SSA1 fault was 100% coverage and the result goes into state 4, unifying, of the ISU.

We can also test for a signal line which is stuck at 0. For example, if A is stuck at 0, we apply the opposite value on x which would be a 1. We also apply the 1 to B which

is the opposite of the fault we are testing for. If x is stuck at 0, then the output will differ by three out of four values from the correct output. This means that the SSA1 fault was 100% coverage and the result goes into state 4, unifying, of the ISU.

To determine the testing result automatically, we can implement another ALU which receives the correct input values. The output from the correct ALU can be exclusively OR'd (XOR'd) together to the test ALU. Performing 4 tests will result in four output values from each ALU. If we XOR the outputs from each ALU, we will get a "1" when one of the tests produced an incorrect result. In the above cases, we will receive 3 out of 4 "1's" meaning that 3 of the 4 tests did not pass. This means that the fault was detected and the result can enter the ISU unifying state.

Obviously this is one application of ISU to test pattern generation and fault detection. Much more analysis is required to see if this is an optimal approach and how it would best be implemented. Consideration of hardware area penalty, signal line routing, fault types supported, fault coverage, etc. all have to be taken into account. After performing this analysis, a better way may be found to apply ISU to fault detection. But this required analysis is beyond the scope of this paper and could be performed as a follow-on study.

Conclusion

As the feature sizes of CMOS technology are further reduced to achieve faster performance, to pack more transistors on an integrated circuit, and to reduce the power dissipation, the physics of the transistor can change and hence the possible fault manifestations. Faults may occur due to changes in device operation caused by the higher electric fields encountered in smaller channel length devices. New ways to determine faults and to improve fault detection techniques and fault coverage are required.

This paper applied the ISU analysis technique to several different test pattern generation approaches. It was also used to analyze the fault detection process. It provided insight as to how the different fault detection algorithms processed faults and their comparative performance based upon two evaluation criteria.

This paper determined that there may be a new (holistic) testing approach that can incorporate different functions that are currently applied in the existing methods. It is too early to tell without further rigorous analysis as to whether the new ISU (holistic) testing approach can solve the problem of high level fault testing. There may be a way to implement aspects of ISU in hardware as a built in self test techniques as part of a high level design of a circuit and implement it in a test mode. A review was provided for the general organization and specific functions which are needed to solve that problem.

Unilogic form may be one alternative to achieve the holistic testing approach. A comparative analysis is needed once the details of this new holistic approach are developed to determine what differences in approach can be achieved based on this new

logical approach and how can it be used to analyze and discover the different types of faults in CMOS VLSI circuits. A very simple example was provided as to how it may be implemented and but more study is needed to determine what is required. Further analysis is necessary to identify what criteria should be used to test the efficiency and effectiveness of this new holistic testing approach based on ISU.

Appendix A - Key Concepts[18] and Terms[19]

VHDL - is an acronym for Very High Speed Integrated Circuit (VHSIC) Hardware Description Language. VHDL is a high level of abstraction language for describing general digital hardware device. It is independent of the technology or methodology used to build the device. The general VHDL model contains three interdependent submodels: the behavioral model; the timing model; and the structural model, which are integrated into the single VHDL device description. Each sub-model will be described as follows:

Behavioral Model - Each operation is a process and the pathways in which values are passed are signals. The discrete system used to model a digital device is contained within a design entity, which is comprised of 2 parts: an entity declaration and an architectural body. The interface defines the inputs and outputs of the device and the body defines the discrete system used to model the device.

Timing Model - This is based on a stimulus-response paradigm. When there is a stimulus, the model responds and then waits for more stimulus. When a process generates a value on the data pathway, it may also designate the amount of time before the value is sent over the pathway. This is scheduling a transaction after a given time. The collections of transactions fro a signal is called the driver to the signal. The driver is a set of time/value pairs which hold the value of each transaction and the time at which the transaction should occur.

Structural model - The decomposition of the device into functionally related sections. Many digital devices are designed by tying (wiring) together many subdevices. Each subdevice is a discrete system onto itself. The outermost data pathways of a discrete system are defined by the interface to the digital device which is defined by the entity. When a discrete system ties together two subsystems, it is really connecting a data pathway from one subsystem to the data pathway to the other subsystem. These connections are called ports. The definition of a port represents a declaration of a signal, and therefore, a data pathway.

Hardware structure - includes the component, the port, and the signal.

Component - is the basic building block of the hardware description, and can describe a gate, chip, or board, or any level of the device being described.

Port - is the component's point of connection to the outside world, the point through which data flows into and out of the component.

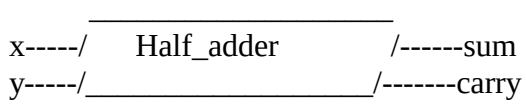
Signal - ties the behavioral view to the structural view. It is an abstraction of a hardware wire. The signal holds changing data values and also connects the subcomponents at their

ports. The signal is the path from one component to another component, along which data flows.

Design entity - is the VHDL representation of a component. It is comprised of an entity declaration and an architecture body.

Entity declaration - provides the external view of the component and describes what can be seen from the outside, including the entity's ports. The following is an example of a half adder entity declaration which declares 2 in ports and 2 out ports:

```
--The entity declaration
entity half_adder is
        port (
                x:  in Bit;
                y:  in Bit;
                sum:  out bit;
                carry:  out bit);
end half_adder;
```



Architecture Body - provides the internal view. It describes the behavior or the structure of the component. The half_adder behavior is described below:

```
--the architecture body
architecture behavioral_description of half_adder is
begin
        process
                sum <= x or y after 5 ns;
                carry <= x and y after 5 ns;
                wait on x, y;
        end process;
end behavioral_description
```

The architecture body has a process statement containing 2 signal assignment statements and a wait statement. The wait statement suspends the process until there is an event on either of the signals x or y, at which time execution resumes at the top of the process statement.

Structure - is described in VHDL by declaring the signals and connections to the ports of the subcomponents. The connections are specified in component instantiation statements.

The component instantiation statements name a component that has been declared in a local component declaration.

Built-in Test Techniques - test techniques incorporated into the design of a system

Component defects - physical imperfections or flaws in an electronic component

Concurrent test techniques - test techniques that can be performed while the system is operating normally

Controllability - the ability to set a line to a specific logic value

D-algorithm - an algorithmic approach to implement the process of path sensitization and guarantees to find a test pattern for any single, stuck-type fault in any non-redundant, combinational circuit, provided the algorithm is allowed to execute for a sufficient length of time. It is a method of implementing the forward drive and backward trace processes of path sensitization. These are called the D-drive and the consistency operation, respectively. It uses the pdcf and pdc for the D-drive.

Design for testability - the process of including special features to make a device easily testable

Dynamic test - a test of the timing characteristics of a device

Elaboration - Every time the subprogram is called, the VHDL code is elaborated, with different faults being injected. This produces dynamic fault injection.

Error - occurrence of an incorrect value in some unit of information within a system

Essential test pattern - a test pattern that detects at least one fault that no other test pattern detects

Fault - a physical defect, imperfection, or flaw that occurs in hardware or software

Fault detection - the process of recognizing that a fault has occurred

Fault Detection (or Test) Experiment - a collection of test patterns used to determine if a device is faulty or fault free

External Test Techniques - test techniques that normally require test equipment outside of the device under test

Fault detection table - a table containing one row for each input combination and one column for each fault. The entries in the table specify ahich input combinations detect which faults.

Fault dictionary - a table that shows the response of a circuit to each test pattern under fault-free conditions and when each possible fault is present

Fault location experiment - a set of test vectors used for detecting and locating faults in a circuit

Fault location table - a table containing one row for each input combination, one column for each fault, and one column for each pair of faults. The entries in the table identify which faults are detected by which input combinations and which pairs of faults are distinguished by the input combinations.

Fault testing - the process of checking for physical hardware faults

Functional testing - verification that a device performs its design-specified operations

Indeterminant fault - a fault whose status changes from time to time

Instancing - The number of faults injected will be the same as the number of call statements in the subprogram. For each call statement, the same fault is always generated. This produces static fault injection.

Intermittent fault - a fault that appears, disappears, and then reappears within a system

Logical stuck-fault model, single stuck at (SSA), single stuck fault (SSF) - a representation that assumes all faults will appear as lines in the logical diagram being physically stuck at a logic 1 or logic 0 value

Observability - the ability to propagate the value on a line to an output where the value can be viewed

Parametric testing - verification of characteristics such as timing parameters

Path sensitizing - process of propagating a signal through a logic circuit to a primary output

Primitive D-cube of fault (pdcf) - a test pattern for a logic module that brings the effect of a fault within the module to the module's output

Propagation D-cube (pdc) - an input combination for a module that forces the module's output to be dependent upon one of the module's inputs

Scan design - a design for testability method where all flip flops are connected to form one or more shift registers to allow easy controllability and observability of internal points within a circuit

Sharing - All calls execute the same code, in which only one fault is injected. This produces static fault injection.

Signature Analysis - a test procedure where the response of a device over a period of time is compressed into characteristic values called signatures. The signatures of a device under test are compared to those of a known fault-free device.

Static test - a test of the steady-state characteristics of a device

Stuck-open fault - a physical fault resulting in the output of a gate depending on the present input and the previous output

Transistor stuck-fault model - a representation that assumes all faults will appear as transistors in the circuit diagram being physically stuck on or stuck off

Test, Test Pattern, Test Vector, or fault Detection Test - a primary input combination that causes a device to produce an erroneous output when a fault is present

Testability - the ability to find test vectors for faults on a specific line

Test pattern generation - the process of determining a set of test patterns

Unilogic - is a logic which affords a way to reason about Rhythmic Sequences of an entity.

Rhythmic Sequence - is the universal vocabulary of Form Unity that can be represented by binary symbols.

Form Unity - is the simplest and most abstract form of any entity, representing the Hayawic Interest, ie. The Hayawic form.

Hayawic Interest - is the dynamic containing form of an entity, ie. That interest motivated by need in time to define or to recycle its Interest Square Root.

Interest Square Root - is the universal vocabulary used to define any Life Interest Cycle of an entity, including the binary operations of negative/positive, inner/outer, mono/poly, open/closed, extreme/moderate, partial/total, etc.

Life Interest Cycle - is the system that an entity uses in order to achieve an ultimate goal in a given time period in the framework of the Interest Square Unity.

Interest Square Unity - is a dynamic logic square that can classify any permutation of the four categories of a life cycle whose binary operations are opposite, contradicted, completed, included, excluded, etc. Aristotle's opposites logic square represent only one

variation of the interest square unity. The Paradox of Russell or Cantor also represents just one variation of the interest square unity. Hayawic Logic is based upon the "Unilogic Form" of Alnakari.

Appendix B - Description of Recent Design for Test Tool Implementation[20]

Some of Mentor Graphics DFT tools are described briefly below.

- FastScanTM

FastScan, the industry's premier automatic test pattern generation tool, creates high-quality tests for ASICs and ICs using full or structured partial scan. FastScan extends its already proven production design capabilities with new support for both Verilog and VHDL, along with a new graphical interface. FastScan utilizes its unique Scan Sequential technology and supports all of the predominant fault classes found in manufacturing defects. FastScan generates test patterns for critical paths and IDDQ, as well as devices designed with boundary scan. In addition, FastScan simulates and fault-grades devices with built-in self-test structures.

- QuickFault II:

Supercharge your fault simulation speed with QuickFault II, our high-performance, timing-based fault simulator for ASIC and board designs. Based on the QuickSim II engine, QuickFault II's deterministic concurrent fault simulation runs 10 times faster than competitive fault simulators. QuickFault II helps ASIC and board designers fault simulate in the timing domain, a common requirement for asynchronous designs. QuickFault II supports the full range of board model types, including gate-level, behavorial, and hardware models. It models both pin-level and net-level fault types and generates a fault dictionary. And you can use the same ASIC model libraries with QuickFault II that you use with QuickSimTM II.

- DFTAdvisorTM

Greatly improve your IC and ASIC design testability with DFTAdvisor. DFTAdvisor speeds your test efforts by automatically inserting full or partial internal scan and test logic structures into your design. Its exhaustive testability analysis identifies and corrects test problems before test generation and automates scan synthesis. The results are maximum ATPG effectiveness while addressing testability analysis early in the design process to avoid costly downstream fixes. DFTAdvisor adds Verilog and VHDL to its suite of design databases. Its new graphical user interface guides you through testability analysis and optimized scan insertion. Finally, DFTAdvisor executes a comprehensive rule checker to ensure that any remaining testability problems are identified before test generation.

- QuickGradeTM II

With QuickGrade II's rapid and responsive fault grading, you can measure test vector coverage any time during the design cycle. QuickGrade II applies probabilistic methods to the results of logic simulation to generate an accurate list of undetected faults and an estimate of total fault coverage. Because runtime is faster than deterministic fault simulation, you get timely feedback for test pattern and testpoint development. Its patented high-level modeling technology supports built in primitives, as well as behavioral and hardware models. The SimView graphical user interface displays QuickGrade II results on the schematic for fast comprehension.

Price and Availability - CTIntegrator will be available in May of 1998 starting at $30,000 per seat. Support for HP-PA, Sun SPARC, and Windows NT industry-standard workstations will be provided upon initial shipment. Mentor Graphics DFT will be working with teacher customers throughout Q1 1998.

- LBISTArchitect

Automatically generate built-in self test (BIST) circuitry for your ASIC, IC and core designs with LBISTArchitect. LBISTArchitect automatically adds BIST structures and generates a complete RTL description of the BIST controller, pattern generator and data compressor. LBISTArchitect allows you to analyze and add BIST testability to a core design, synthesize the BIST logic, fault simulate the design and generate the good-circuit signature. LBISTArchitect contains unique patent-pending technology to provide very high fault coverage with very low-impact on-the-core design. LBISTArchitect generates RTL code synthesizable with Synopsys' Design Compiler and Mentor Graphics' AutoLogic II. And it is integrated with BSDArchitect and DFTAdvisor to provide acomplete 1149.1 BIST solution.

Appendix C - Bibliography of Design for Test Research Papers[21]

Automatic Test Pattern Generation

S. Chandra, N. Jacobson, G. Srinath, "Practical Considerations in ATPG using Crosscheck Technology," Proc. of First Asian Test Symposium, November 1992, pp. 88-93.

S. Chandra, T. Ferry, T. R. Gheewala, K. Pierce, "ATPG Based on a Novel Grid Addressable Latch Element," Proc. of the 28th ACM/IEEE Design Automation Conference, 1991, pp. 282-286.

S. Chandra, J. H. Patel, "Experimental Evaluation of Testability Measures for Test Generation," IEEE Trans. on Computer-Aided Design of Integrated Circiuts and Systems, Vol. 8, No. 1, January 1989, pp. 93-97.

S. Chandra, J. H. Patel, "Test Generation in a Parallel Processing Environment," Proc. of 1988 International Conference on Computer Design, October 1988, pp. 11-14.

S. Chandra and J. H. Patel, "A Hierarchical Approach to Test Vector Generation," Proc. of 24th ACM/IEEE Design Automation Conference, June 1987, pp. 495-501.

W.-T. Cheng, S. Davidson, "Sequential Circuit Test Generator (STG) Benchmark Results," Proc. of IEEE International Symposium on Circuits and Systems, Part 3, 1989, pp. 1939-1941.

W.-T. Cheng and T. J. Chakraborty, "GENTEST: An Automatic Test-Generation System for Sequential Circuits," Computer, Vol. 22, No. 4, April 1989, pp. 43-49.

W.-T. Cheng and J. Y. Jou, "A Hierarchical Sequential Test Generator," Proc. of Allerton Conference on Communication, Control, and Computing, 1988, pp. 750-757.

W.-T. Cheng, "The BACK Algorithm for Sequential Test Generation," Proc. of International Conference on Computer Design, October 1988, pp. 66-69.

W.-T. Cheng, "SPLIT Circuit Model for Test Generation," Proc. of 25th ACM/IEEE Design Automation Conference, June 1988, pp. 96-101.

S. Patel and J. H. Patel, "Effectiveness of Heuristics Measures for Automatic Test Pattern Generation," Proc. of 23rd ACM/IEEE Design Automation Conference, June 1986, pp. 547-552.

H. Sucar, S. Chandra, D. Wharton, "High Performance Test Generation for Accurate Defect Models in CMOS Gate Array Technology," Proc. of International Conference on Computer-Aided Design, November 1989, pp. 166-169.

J. A. Waicukauski, P. A. Shupe, D. J. Giramma, A. Matin, "ATPG for Ultra-Large Structured Designs," Proc. of International Test Conference, September 1990, pp. 44-51.

Fault Simulation

W.-T. Cheng and M.-L. Yu, "Differential Fault Simulation - A Fast Method Using Minimal Memory," Proc. of 26th ACM/IEEE Design Automation Conference, June 1989, pp. 424-428.

M. Kassab, N. Mukherjee, J. Rajski, J. Tyszer, "Software Accelerated Functional Fault Simulation for Data-Path Architectures," Proc. of 32nd ACM/IEEE Design Automation Conference, June 1995, pp. 333-338.

M. Kassab, J. Rajski, J. Tyszer, "Hierarchical Functional Fault Simulation for High-Level Synthesis," Proc. of 1995 26th International Test Conference, October 1995, pp. 596-605.

F. Maamari and J. Rajski, "The Dynamic Reduction of Fault Simulation," Proc. of International Test Conference 1990: Digest of Papers, Vol. 12, No. 1, September 1990, pp. 801-808.

F. Maamari and J. Rajski, "A Method of Fault Simulation Based on Stem Regions," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 9, No. 2, February 1990, pp. 212-220.

T. Niermann, W.-T. Cheng, J. H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, No. 2, February 1992, pp. 198-207.

T. Niermann, W.-T. Cheng, J. H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator," Proc. of 27th ACM/IEEE Design Automation Conference, June 1990, pp. 535-540.

J. A. Waicukauski, E. Lindbloom, B. Rosen, V. Iyengar, "Transition Fault Simulation by Parallel Pattern Single Fault Propagation," Proc. of IEEE International Test Conference: Testing's Impact on Design and Technology, 1986, pp. 542-549.

J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, T. McCarthy, "A Statistical Calculation of Fault Detection Probabilities by Fast Fault Simulation," Proc. of IEEE International Test Conference: The Future of Test, 1985, pp. 779-784.

J. A. Waicukauski, et.al., "Fault simulation for Structured VLSI," VLSI Design, Vol. 6, No. 12, December 1985, pp. 20-32.

Diagnostics

W.-T. Cheng, J. L. Lewandowski, E. Wu, "Optimal Diagnostic Methods for Wiring Interconnects," IEEE Transactions on Computer-Aided Design, Vol. 11, No. 9, September 1992, pp. 1161-1166.

W.-T. Cheng, J. L. Lewandowski, E. Wu, "Diagnosis for Wiring Interconnects," Proc. of International Test Conference, September 1990, pp. 565-571.

H. Cox and J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 7, No. 7, July 1988, pp. 813-833.

J. J. Curtin and J. A. Waicukauski, "Multi-Chip Module Test and Diagnostic Methodology," IBM Journal of Research and Development, Vol. 27, No. 1, January 1983, pp. 27-34.

J. A. Waicukauski and E. Lindbloom, "Failure Diagnosis of Structured VLSI," IEEE Design and Test of Computers, Vol. 6, No. 4, August 1989, pp. 49-60.

J. A. Waicukauski, V. P. Gupta, S. T. Patel, "Diagnosis of BIST Failures by PPSFP Simulation," Proc. of IEEE International Test Conference: Integration of Test with Design and Manufacturing, 1987, pp. 480-484.

Design for Testability

S. Chandra and M. Chang, "Scan-Based Testability Analysis and Test Insertion," Proc. of Electronic Design Automation & Test Conference, October 1994, pp. 250-258.

S. Chandra, K. Pierce, G. Srinath, H. Sucar, V. Kulkarni, "CrossCheck: An Innovative Testability Solution," IEEE Design and Test of Computers, Vol. 10, Issue 2, June 1993, pp. 56-67.

S. Chandra, T. Gheewala, H. Sucar, G. Swan, "Use of CrossCheck Technology in Practical Applications," Proc. of VLSI Test Symposium: Digest of Papers, April 1991, pp. 16-21.

A. Jain, B. Mandava, J. Rajski, N. C. Rumin, "A Fault-Tolerant Array Processor Designed for Testability and Self-Reconfiguration," IEEE Journal of Solid State Circuits, Vol. 26, No. 5, May 1991, pp. 778-788.

V.R. Kulkarni and S. Chandra, "CrossCheck: A New Paradigm for Test Transparency," Proc. of ATE & Instrumentation Conference: Work Smarter Not Harder, January 1992, pp. 75-89.

S. Pateras and J. Rajski, "Design of a Self-Reconfiguring Interconnection Network for Fault-Tolerant VLSI Processor Arrays," IEEE Transactions on Reliability, Vol. 38, No. 1, April 1989, pp. 40-50.

S. Pateras and J. Rajski, "A Self-Reconfiguring Interconnection Network for a Fault-Tolerant Mesh-Connected Array of Processors," Electronics Letters, Vol. 24, No. 10, May 1988, pp. 600-602.

R. Sanchez, "Concurrent DFT at all Levels of the Digital System," Proc. of International Test Conference, 1994, pp. 879.

Built-In Self-Test

S. Adham, M. Kassab, N. Mukherjee, K. Radecka, J. Rajski, J. Tyszer, "Arithmetic Built-In Self Test for Digital Signal Processing Architectures," Proc. of 1995 17th IEEE Custom Integrated Circuits Conference, May 1995, pp. 659-662.

S. Gupta, J. Rajski, J. Tyszer, "Arithmetic Additive Generators of Pseudo-Exhaustive Test Patterns," IEEE Transactions on Computers, Vol. 44, No. 2, February 1995, pp. 223-233.

A. S. M. Hassan, V. K. Agarwal, B. Nadeau-Dostie, J. Rajski, "BIST of PCB Interconnects Using Boundary-Scan Architecture," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, No. 10, October 1992, pp. 1278-1288.

S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," IEEE Transactions on Computers, Vol. 44, No. 2, February 1995, pp. 223-233.

B. Nadeau-Dostie, D. Burek, A.S.M. Hassan, "ScanBist: A Multifrequency Scan-based BIST Method," IEEE Design & Test of Computers," Vol. 11, No. 1, Spring 1994, pp. 7-17.

M. Nicolaidis, O. Kebichi, V. C. Alves, "Trade-Offs in Scan Path and BIST Implementations for RAMs," Journal of Electronic Testing: Theory and Applications, Vol. 5, No. 2-3, May-August 1994, pp. 273-283.

O. Kebichi, M. Nicolaidis, V.N. Yarmolik, "Exact Aliasing Computation for RAM BIST," Proc. of IEEE International Test Conference, November 1995, pp.13-22.

O. Kebichi, Y. Zorian, M. Nicolaidis, "Area Versus Detection Latency Trade-Offs in Self-Checking Memory Design," Proc. of European Design and Test Conference, March 1995, pp. 358-362.

O. Kebichi, V.N. Yarmolik, M. Nicolaidis, "Zero Aliasing ROM BIST," Journal of Electronic Testing: Theory and Applications," Vol. 5, No. 4, November 1994, pp. 377-388.

O. Kebichi and M. Nicolaidis, "A Tool for Automatic Generation of BISTed and Transparent BISTed RAMs," Proc. of IEEE 1992 International Conference on Computer Design: VLSI in Computers and Processors, October 1992, pp. 570-575.

N. Mukherjee, M. Kassab, J. Rajski, J. Tyszer, "Arithmetic Built-In Self Test for High-Level Synthesis," Proc. of 13th IEEE VLSI Test Symposium, June 1995, pp. 132-139.

M. Nicolaidis, O. Kebichi, V. C. Alves, "Trade-Offs in Scan Path and BIST Implementation for RAMs," Journal of Electronic Testing: Theory and Applications, Vol. 5, 1994, pp. 273-283.

J. Rajski and J. Tyszer, "Recursive Pseudoexhaustive Test Pattern Generation," IEEE Transactions on Computers, Vol. 42, No. 12, December 1993, pp. 1517-1521.

J. Rajski and J. Tyszer, "Accumulator-Based Compaction of Test Responses," IEEE Transactions on Computers, Vol. 42, No. 6, June 1993, pp. 643-650.

J. Rajski and J. Tyszer, "Test Responses Compaction in Accumulators with Rotate Carry Adders," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 12, No. 4, April 1993, pp. 531-539.

J. Rajski and J. Tyszer, "The Analysis of Digital Integrators for Test Response Compaction," IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 39, No. 5, May 1992, pp. 293-301.

J. Rajski and J. Tyszer, "On the Diagnostic Properties of Linear Feedback Shift Registers," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 10, No. 10, October 1991, pp. 1316-1322.

J. A. Waicukauski, et.al., "A Method for Generating Weighted Random Test Patterns," IBM Journal of Research and Development, Vol. 33, No. 2, March 1989, pp. 149-161.

J. A. Waicukauski and E. Lindbloom, "Fault Detection Effectiveness of Weighted Random Patterns," Proc. of IEEE International Test Conference, 1988, pp. 245-255.

V.N. Yarmolik, M. Nicolaidis, O. Kebichi, "Aliasing-Free Signature Analysis for RAM BIST," Proc. of 1994 IEEE International Test Conference, November 1994, pp. 368-377.

Testability Theory

S. Chandra and J. H. Patel, "Accurate Logic Simulation in the Presence of Unknowns," Proc. of IEEE International Conference on Computer-Aided Design: Digest of Technical Papers, November 1989, pp. 34-37.

W.-T. Cheng and J. H. Patel, "Testing in Two-Dimensional Iterative Logic Arrays," Computers and Mathematics with Applications, Vol. 13, No. 5/6, 1987, pp. 443-454.

W.-T. Cheng and J. H. Patel, "Minimum Test Set for Multiple Fault Detection in Ripple Carry Adders," IEEE Transactions on Computers, Vol. C-36, No. 7, July 1987, pp. 891-895.

W.-T. Cheng and J. H. Patel, "Testing in Two-Dimensional Iterative Logic Arrays," 16th Annual International Symposium on Fault-Tolerant Computing Systems: Digest of Papers, July 1986, pp. 76-81.

W.-T. Cheng and J. H. Patel, "Minimum Test Set for Multiple-Fault Detection in Ripple Carry Adders," IEEE International Conference on Computer Design: VLSI in Computers, 1985, pp. 435-438.

W.-T. Cheng and J. H. Patel, "Multiple-Fault Detection in Iterative Logic Arrays," Proc. of International Test Conference, 1985, pp. 493-499.

W.-T. Cheng and J. H. Patel, "A Shortest Length Test Sequence for Sequential-Fault Detection in Ripple Carry Adders," IEEE International Conference on Computer Aided Design: Digest of Papers, 1985, pp. 71-73.

W.-T. Cheng and J. H. Patel, "Concurrent Error Detection in Iterative Logic Arrays," 14th International Conference on Fault-Tolerant Computing: Digest of Papers, 1984, pp. 10-15.

H. Cox and J. Rajski, "On Necessary and Nonconflicting Assignments in Algorithmic Test Pattern Generation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 4, April 1994, pp. 515-530.

A. El-Maleh, T. Marchok, J. Rajski, W. Maly, "On Test Set Preservation of Retimed Circuits," Proc. of 32nd Design Automation Conference, June 1995, pp. 176-182. (Best paper award nomination)

A. El-Maleh and J. Rajski, "Delay-Fault Testability Preservation of the Concurrent Decomposition and Factorization Transformations," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 5, May 1995, pp. 582-590.

A. El-Maleh and J. Rajski, "Delay-Fault Testability Preservation of the Concurrent Decomposition and Factorization Transformations," Proc. of IEEE VLSI Test Symposium, April 1994, pp. 15-21.

T. Marchok, A. El-Maleh, J. Rajski, W. Maly, "Testability Implications of Performance-Driven Logic Synthesis," IEEE Design & Test of Computers, Vol. 12, No. 2, Summer 1995, pp. 32-39.

T. Marchok, A. El-Maleh, W. Maly, J. Rajski, "Complexity of Sequential ATPG," Proc. of European Design and Test Conference, 1995, pp. 252-261. (Winner of the best paper award for the most outstanding contribution in the field of test for 1995)

A. Pancholy, J. Rajski, L. McNaughton, "Empirical Failure Analysis and Validation of Fault Models in CMOS VLSI," Proc. of International Test Conference 1990: Digest of Papers, September 1990, pp. 938-947.

*J. Rajski and J. Vasudevamurthy, "The Testability-Preserving Concurrent Decomposition and Factorization of Boolean Expressions," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, No. 6, June 1992, pp. 778-793. (Recipient the IEEE Circuits and Systems Society IEEE Transactions on Computer-Aided Design Best Paper Award in 1993)

C.-C. Tsai and M. Marek-Sadowska, "Generalized Reed-Muller Forms as a Tool to Detect Symmetries," IEEE Transactons on Computers, Vol. 45, Issue 1, January 1996, pp. 33-40.

C.-C. Tsai and M. Marek-Sadowska, "Detecting Symmetric Variables in Boolean Functions Using Generalized Reed-Muller Forms," Proc. of 1994 IEEE International Symposium on Circuits and Systems, Vol. 1, 1994, pp. 287-290.

C.-C. Tsai and M. Marek-Sadowska, "Minimization of Fixed-Polarity AND/XOR Canonical Networks," IEE Proceedings: Computers and Digital Techniques, Vol. 141, Pt. E, No. 6, November 1994, pp. 369-374.

C.-C. Tsai and M. Marek-Sadowska, "Boolean Matching Using Generalized Reed-Muller Forms," Proc. of 31st Annual ACM/IEEE Design Automation Conference, June 1994, pp. 339-344.

J. Waicukauski, et.al., "On Computing the Sizes of Detected Delay Faults," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 9, No. 3, March 1990, pp. 299-312.

Test Synthesis

J. Rajski, J. Vasudevamurthy, A. El-Maleh, "Recent Advances in Logic Synthesis with Testability," Proc. of 1992 IEEE VLSI Test Symposium: Digest of Papers, April 1992, pp. 254-256.

C.-C. Tsai and M. Marek-Sadowska, "Logic Synthesis for Testability," Proc. of 6th Great Lake Symposium on VLSI, 1996, pp. 118-121.

C.-C. Tsai and M. Marek-Sadowska, "Multilevel Logic Synthesis for Arithmetic Functions," Proc. of the 33rd ACM/IEEE Design Automation Conference, 1996, pp. 242-247.

Appendix D - List of Sources Contacted and Information Provided for this Project

Since starting this project, the following people were contacted and why are listed below:

Apr 98: Contacted Dr. Jim Armstrong, Virginia Tech, to see if the B-Algorithm is being used commercially (it was not) and if it was ever implemented. He said that it had been implemented in Verilog and Elena Gramatova has also implemented it. He said there may not be as much of a need for test generation at the behavioral level because of some of the good fault coverage results current commercial tools were achieving.

20 Apr 98: Contacted James Hanna, Air Force Research Laboratory, 315-330-3473, to find out what work was being supported in the area of fault injection/simulation in VHDL simulators and in developing and evaluating fault models for RTL level/behavioral VHDL. This work is being done at UVA. The work will continue through Sep 98 but after that the laboratory will stop funding it because they eliminated the reliability section in the laboratory during a restructuring and this is where the fault work was being done.

23 Apr 98: Contacted Michelle Kuyl, Technical Marketing Engineer, Mentor Graphics, 503-685-1768, and found out that enhancements were being made to FastScan to include bridging fault ATPG and this functionality will be available by 4Q98. Also, FastScan uses the parallel Pattern Single Fault Propagation (PPSFP) algorithm which first performs random pattern generation by fault simulating blocks of generated patterns until it no longer finds a value in that process (if the random patterns don't detect at least 0.5% of the remaining faults). Then it switches over to a more intelligent approach to fault detection (deterministic ATPG) using the same algorithm. FastScan has several different ATPG engines within it, each targeted at different types of circuits. It handles non-scan latches, sequential circuitry of limited depth, circuitry around embedded RAMs, tri-state logic, etc. FastScan has been used in production designs of over 4 million gates with coverage of 99%+ fault coverage.

May 98: Received a copy of Elena Gramatova's paper on the B-X algorithm and her fault coverage results. She is from the Institute of Computer Systems, Slovak Academy of Sciences, Bratislava, Slovakia. I also requested a copy of the code she used for a small circuit to implement the B-X Algorithm but I have not yet received any response from her on this.

---

[1] Cheng, Kwang-Ting, Shi-Yu Huang, and Wei-Jin Dai, "Fault Emulation: A New Approach to Fault Grading", p. 1

[2] According the Mentor Graphics DFT web page, located at http://www.mentorg.com/dft

[3] McCluskey, E. J. "Quality and Single-Stuck Faults", *International Test Conference*, 1993, Paper 29.4, p. 597

[4] Soden, Jerry M. and Charles F. Hawkins "Quality Testing Requires Quality Thinking", *International Test Conference*, 1993, Paper 29.3, p. 596

[5] Shen, John P., W. Maly and F. Joel Ferguson "Inductive Fault Analysis of MOS Integrated Circuits", *IEEE Design & Test*, Dec 1985, pp. 13-26
[6] Abramovici, Miron "DOs and DON'Ts in Computing Fault Coverage, *International Test Conference*, 1993, paper 29.1, p. 594
[7] Cho, Chang Hyun and James R. Armstrong, "B-algorithm: A Behavioral Test Generation Algorithm, *International Test Conference*, 1994, paper 39.2, pp. 968-979
[8] Sanchez, Pablo and Isabel Hidalgo, "System Level Fault Simulation", International Test Conference, 1996, paper 27.3, pp. 732-740
[9] Ramchandani, R. S. and D. E. Thomas, "Behavioral Test Generation using Mixed Integer Non-linear Programming", *International Test Conference*, 1994, paper 39.1, pp. 958-967
[10] Papchristou, Christos and Joan Carletta, "Test Synthesis in the Behavioral Domain", *International Test Conference*, 1995, paper 30.3, pp. 693-702
[11] Rine, David. C. And Raiek Alnakari, Hayawic Form Unit Mathematical Systems for a Human Communication Theory, Unilogic Analyses, 1997, pp. 42-43
[12] Alnakari, Raiek A. And David C. Rine, A Hayawic UniLogic Form Mathematical Systems Approach to Knowledge Representation for Human Communication Theory, George Mason University
[13] Gramatova, E., J. Bezakova, and M. Fischerova, *BX-TPG Algorithm at the Behavioral Level Implemented under the LEDA VHDL System*, Institute of Computer Systems, Slovak Academy of Sciences, Batislava, Slovakia
[14] Cho, Chang Hyun and James R. Armstrong, "B-algorithm: A Behavioral Test Generation Algorithm, *International Test Conference*, 1994, paper 39.2, pp. 968-979
[15] Sanchez, Pablo and Isabel Hidalgo, "System Level Fault Simulation", International Test Conference, 1996, paper 27.3, pp. 732-740
[16] Ramchandani, R. S. and D. E. Thomas, "Behavioral Test Generation using Mixed Integer Non-linear Programming", *International Test Conference*, 1994, paper 39.1, pp. 958-967
[17] Papchristou, Christos and Joan Carletta, "Test Synthesis in the Behavioral Domain", *International Test Conference*, 1995, paper 30.3, pp. 693-702
[18] Lipsett, Roger, Carl Schaeffer, and Cary Ussery, *VHDL: Hardware Description and Design*, Kluwer Academic Publishers, 1989
[19] Johnson, Barry W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989
[20] From Mentor Graphics Design For Test Group Web Page. For more information about Mentor Graphics, contact: Amy Malagamba at 503/ 685-1301and at amy_malagamba@mentorg.com
[21] From Mentor Graphics Design For Test Group Web Page