Copyright

by

Richard Dean Putman

2008

The Dissertation Committee for Richard Dean Putman Certifies that this is the approved version of the following dissertation:

TESTING FOR DELAY DEFECTS UTILIZING TEST DATA COMPRESSION TECHNIQUES

Committee:

Nur Touba, Supervisor

Tony Ambler

Earl Swartzlander

Lizy John

William Gallagher

TESTING FOR DELAY DEFECTS UTILIZING TEST DATA COMPRESSION TECHNIQUES

by

Richard Dean Putman, B.S.E.E.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2008

Dedication

In memory of my beloved mother, Cuba.

Acknowledgements

I would like to express my sincere appreciation to Dr. Touba for all of his encouragement and assistance in helping me get started in the Ph.D. program. I would also like to thank him for all of his guidance and support throughout the entire process. I am also grateful to Prof. Tony Ambler, Prof. Earl Swartzlander, Prof. Lizy John, and Dr. Lynn Gallagher for graciously serving on my dissertation committee and reviewing this dissertation. I would also like to acknowledge Prof. Margarida Jacome for serving on my original dissertation committee.

I am also extremely thankful to have such a wonderful, loving, and supportive wife. Her continuous support, encouragement, and proofreading were essential to my completion of this dissertation.

I would also like to thank my parents, Joe and Cuba Putman, for starting me on this path and giving me the foundation to be successful in life.

TESTING FOR DELAY DEFECTS UTILIZING TEST DATA COMPRESSION TECHNIQUES

Publication No._____

Richard Dean Putman, Ph.D.

The University of Texas at Austin, 2008

Supervisor: Nur A. Touba

As technology shrinks new types of defects are being discovered and new fault models are being created for those defects. Transition delay and path delay fault models are two such models that have been created, but they still fall short in that they are unable to obtain a high test coverage of smaller delay defects; these defects can cause functional behavior to fail and also indicate potential reliability issues. The first part of this dissertation addresses these problems by presenting an enhanced timing-based delay fault testing technique that incorporates the use of standard delay ATPG, along with timing information gathered from standard static timing analysis. Utilizing delay fault patterns typically increases the test data volume by 3-5X when compared to stuck-at patterns. Combined with the increase in test data volume associated with the increase in gate count that typically accompanies the miniaturization of technology, this adds up to a very large increase in test data volume that directly affect test time and thus the manufacturing cost. The second part of this dissertation presents a technique for improving test compression and reducing test data volume by using multiple expansion ratios while determining the

configuration of the scan chains for each of the expansion ratios using a dependency analysis procedure that accounts for structural dependencies as well as free variable dependencies to improve the probability of detecting faults. Finally, this dissertation addresses the problem of unknown values (*X*'s) in the output response data corrupting the data and degrading the performance of the output response compactor and thus the overall amount of test compression. Four techniques are presented that focus on handling response data with large percentages of *X*'s. The first uses *X*-canceling MISR architecture that is based on deterministically observing scan cells, and the second is a hybrid approach that combines a simple *X*-masking scheme with the *X*-canceling MISR for further gains in test compression. The third and fourth techniques revolve around reiterative LFSR *X*-masking, which take advantage of LFSR-encoded masks that can be reused for multiple scan slices in novel ways.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In the process of manufacturing semiconductor products, some percentage of the product is going to have physical defects present. Therefore, in the business of selling these semiconductor products, it is necessary to screen or test the product for these physical defects to ensure some measure of quality and reliability for the customer. A test for a semiconductor product consists of a stimulus that is applied to the product (or chip) and an expected response that is measured from the product. In order to measure the quality of a given test, types of possible defects have to first be mapped to fault models. For example, wires (or nets) in the product that are shorted to power or ground can be represented by the most common fault model, the Stuck-At fault model. A stuck-at-one represents a short to power, and a stuck-at-zero represents a short to ground. Once the total number of faults has been determined, a given test can be rated to see how many faults it detects. The measure of fault coverage of a test or group of tests for a given product can be given by a simple expression consisting of the total number of faults detected divided by the total number of faults.

$$Fault\ Coverage = Faults_{det} / Faults_{tot}$$

To test the products, Automatic Test Equipment (ATE) is used to apply the test stimulus, measure the response, and compare it to the expected results. The test data, including the stimulus and the response, is stored on the ATE. The ATE, however, has a number of limitations, including test data memory and test data bandwidth. Test data bandwidth is defined as the number of ATE channels used for driving the test stimulus, or measuring the test response, multiplied by the clock rate of the channels. Test data memory is defined as the amount of memory available for storing test data per ATE test

channel. So, a 32MB tester has 32 million bits of data storage available for each tester channel. The combination of test data volume and test data bandwidth used for a given test or suite of tests determines the test time, or time it takes to apply the stimulus and measure the response. Since each second on the ATE costs in the neighborhood of 2-5 cents, test time plays a big role in the cost of manufacturing for a given product. Because of this cost, it is desirable to utilize the full bandwidth of the ATE and, at the same time, minimize the test data volume, while still having the ability to detect and discard all of the defective products.

1.1 SCAN TESTING

As technology shrinks and semiconductor devices become more and more complex, it is becoming increasingly difficult, if not impossible, to test these devices with pure functional test data (data that exercises all of the functionality of a part in enough combinations to cover all faults). Because of this, some Design-for-Test (DFT) techniques have been developed, and are being developed, that make testing easier. One of the most basic techniques, which is in turn the foundation for many other techniques, is simply referred to as scan testing. In the most common methodology for scan testing, most, if not all, of the flip-flops in the circuit are converted to scan flip-flops, which are simply regular flip-flops with a MUX in front. The MUX has two inputs; the first is the original functional data input that maintains its initial connection, and the second, labeled scan data input, enables each of the scan flip-flops to be connected back-to-back to form one or more shift registers, also called scan chains. Figure 1.1 shows how scan flip-flops are connected together to form scan chains. The select signal of the MUX is connected to a global signal, typically labeled SCAN_ENABLE, which determines whether the scan flip-flops are configured in normal functional mode or in shift mode (i.e., as one or more shift registers).

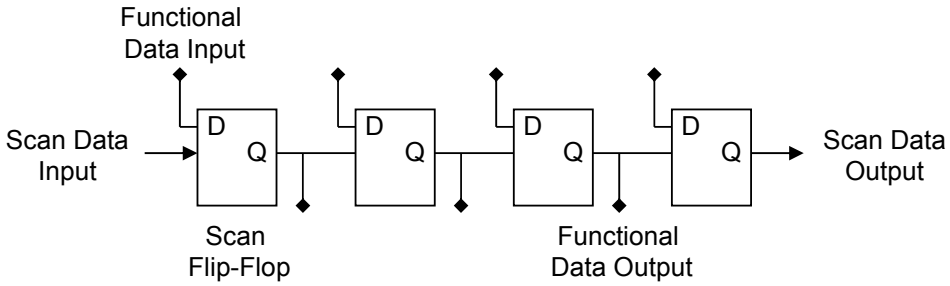

Figure 1.1: Scan Chain Example

The purpose of creating scan chains and performing scan testing is to drastically improve the controllability and observability of the circuit-under-test (CUT). In this type of setup, not only can the primary inputs of the CUT be deterministically set, but all of the internal scan flip-flops can be set as well by loading (i.e., shifting) values into the scan chains. Once all of the internal flip-flops have been loaded, the values can propagate through the surrounding combinational logic; then, one or more functional capture clocks can be triggered, and the resulting values captured into the scan flip-flops and then shifted out of the scan chains to be observed and measured. This technique works particularly well with Automatic Test Pattern Generation (ATPG) tools, which can use this type of setup, shown in Figure 1.2, to achieve very high fault coverage numbers in even the most complex circuits.

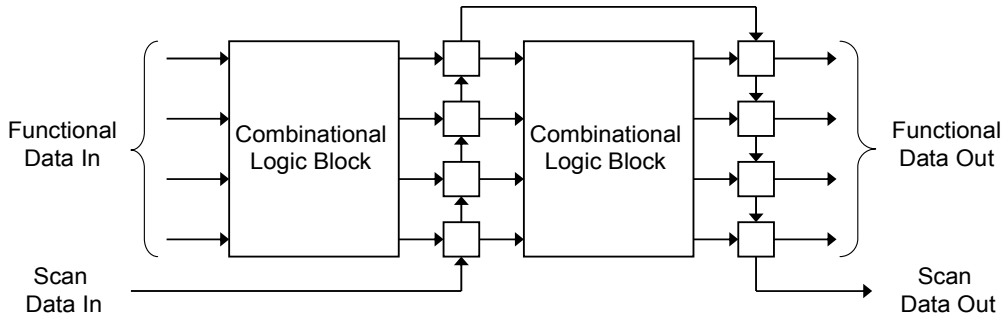

Figure 1.2: Example CUT with Scan Chain

In addition to semiconductor devices becoming increasingly more complex to test as the technology shrinks into the deep submicron range, the types of defects and the fault models they map to are changing as well. For the last couple of decades, the prevailing fault model has been the Stuck-At fault model, as mentioned above. But as technology shrinks, the Stuck-At fault model is no longer adequate to cover all of the different types of physical defects. This is partially due to the static nature of the Stuck-At fault model, which does not explicitly take into account any timing.

Current semiconductor products are producing more and more physical defects that are altering the intended timing of the device but not necessarily the functionality of the device. This is why these types of defects can be present, but the chip can still pass a stuck-at fault test. A few examples of these defects are resistive vias, resistive gate-oxide shorts, insufficient transistor doping, and extra metal capacitive loading on wire traces [Crouch 00]. Because of this, new fault models have been, and are being, developed. One category of such fault models is the delay fault models.

1.2 DELAY FAULT TESTING

Delay faults are timing-based faults categorized as slow-to-rise or slow-to-fall faults or transitions. The object of a delay fault test is to propagate a transition down a circuit path from a start point to an endpoint and see if it makes it there in a predetermined amount of time. As shown in Figure 1.3, this is accomplished by using two clock pulses: a launch clock pulse and a capture clock pulse. After initialization, the first clock pulse launches the transition, and the second clock pulse captures it. The time from the launch to the capture is the time allowed for the defect-free signal to propagate. Ideally, the propagation time for a given transition down every tested path would be known and time between the launch and the capture would be set such that there would

be zero slack on the path. This would enable the detection of very fine or small delay defects on all tested paths.

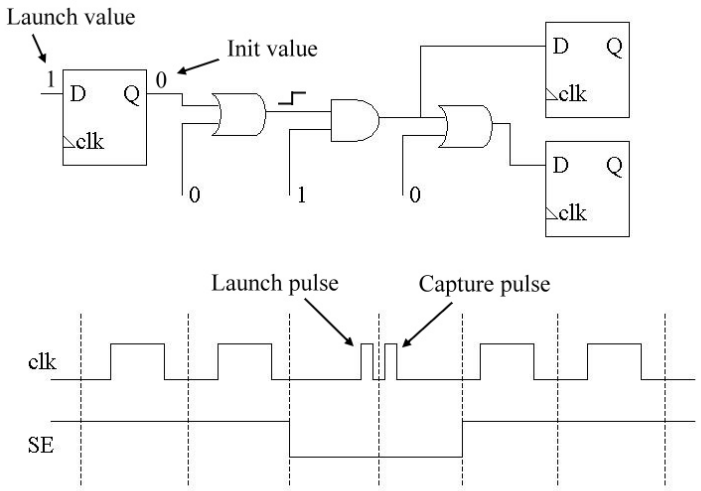

Figure 1.3: Delay Fault Circuit and Waveforms

Currently, the two main types of delay fault models are the Transistion Delay fault model [Waicukauski 87] and the Path Delay fault model [Smith 85][Lin 87].

1.2.1 Transition Delay Fault Model

The transition delay fault model is a node-based fault model, similar to the stuck-at fault model, but instead of stuck-at-one and stuck-at-zero faults, it uses slow-to-rise and slow-to-fall faults placed on gate inputs and outputs. Since many paths with different path timings are tested simultaneously, this fault model is typically only used for testing gross or large delay defects, because the set time between the launch pulse and the capture pulse for a group of tested paths is limited, at a minimum, by the slowest tested path. Because of this limitation, and the fact that this is a node-based fault model, it is primarily used for targeting point delay defects like resistive vias, poly cracks, and

metal/poly bridges. The disadvantage, of course, is that this fault model is not very good at detecting very small delay defects, some of which can have a cumulative effect down long paths and become larger problems. This situation is exacerbated by the tendency of the ATPG tools that implement this fault model to always choose the shortest, easiest paths to propagate and observe transitions.

However, this model does have two advantages: the creation of a finite fault list that is easy to build and the use of simpler stuck-at-like ATPG algorithms compared, to the more complex sequential algorithms needed for the path delay fault model. Also, a higher fault coverage [Kruseman 04][Madge 03][Yan 03] can generally be obtained for the faults targeted with this fault model than with the second delay fault model, the path delay model.

1.2.2 Path Delay Fault Model

The path delay fault model, where each fault represents one entire path, has been the subject of much research. A few examples of this research are [Heragu 96][Liou 00][Crouch 03]. One of the main motivations for this research has been that the path delay model is one of the best ways to detect small delay defects, including the point delay defects mentioned above, and also distributed or cumulative defects that result from things like mask misalignment, oxide too thick or too thin, or ion implant off from specification. It has this ability because the timing for each individual path fault (path tested) can be adjusted such that there is near zero slack, thus allowing the detection of very small delay defects.

In fact, if 100% path delay coverage could be achieved, then all of the small delay defects as well as all of the transition and stuck-at faults would be covered. However, the goal of 100% path delay coverage has yet to be achieved due to the exponential number of path faults that would need to be tested for a given design, the vast number of patterns

that would be required to test all of those paths, and, in general, the inability to create robust patterns for path delay faults [Heragu 96][Pomeranz 98][Crouch 03][Datta 04].

So, with standard transition delay fault testing able to achieve relatively high delay fault coverage but only for gross or large delay defects and with the difficulties associated with path delay testing, there is still a need for something else to obtain high coverage of small or smaller delay defects.

Additionally, transition and path delay ATPG generally creates rather large amounts of test data. The test data volume for each of these models is typically three to five times larger than for the stuck-at model. This increase is amplified by the fact that the design complexity has also increased, causing the number of faults, and thus, the test data volume, to further increase. Because of this, there is also a need to continue to create new techniques that will reduce the test data volume and test time and still be effective for the testing of delay defects.

1.3 Reducing Test Data Volume and Test Time

As test data volume grows, test engineers are running out of memory to contain the increased number of test patterns, and since the device under test (DUT) can typically only be tested as fast as the data can be transferred from the tester, the more test data volume there is the longer the test time and thus, the higher the manufacturing cost. Two undesirable solutions to the memory depth limitation are to truncate the pattern set, which sacrifices test quality, or to load a second set of test patterns into memory, which can be very expensive time-wise.

A more viable approach to reducing test data volume in order to avoid hitting the tester memory depth limitation, and, more importantly, to reduce test time is to use test data compression. Figure 1.4 shows the basic test compression architecture. To use this type of structure for testing, the compressed test data stimulus is stored on ATE and then

fed into a decompressor, which in turn feeds a large number of scan chains where the number of scan chains is *m* and is much larger than the number of tester channels, *b*. The output response data from the scan chains is then fed into an output response compactor, which is then read back into the ATE to be compared to the expected response. A number of test compression schemes following this architecture have been developed and are being used in industrial designs. Most of them were initially targeted at reducing test data volume for the static stuck-at fault model, but they can also be applied to delay fault testing with varying degrees of success. A survey of the different techniques can be found in [Touba 06].

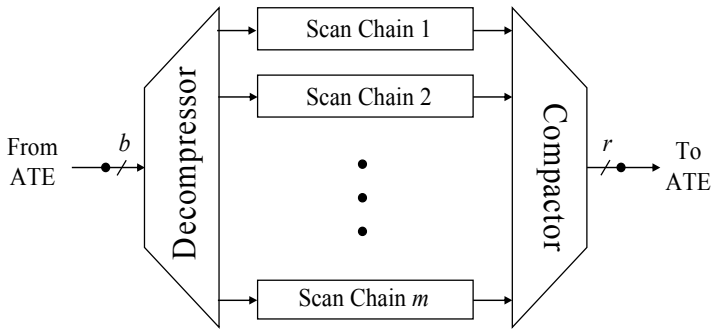

Figure 1.4: Basic Test Compression Scheme

One of the first widely used techniques created to reduce test data volume was the Logic Built-In Self Test (LBIST) technique, based on the STUMPS architecture [Wang 06], as shown in Figure 1.5. Here the ATE provides no stimulus, but instead an LFSR (Linear Feedback Shift Register) generates pseudo-random data which drives the scan chains, and the output response data from the scan chains is compacted in a MISR (Multiple Input Shift Register) and checked at the end of the test. LBIST is one example of a technique that was initially developed for stuck-at testing in order to drastically reduce test data volume, but, when run at full functional speed, can detect some delay

faults. One of problems with LBIST is that it has difficulty with random resistant faults, such as in cases when testing the faults on AND or OR gates with large numbers of inputs, where all inputs, or all but one input, are required to be the same value. These types of patterns are difficult for LBIST to generate because of the pseudo-random patterns it generates. In general, delay faults also tend to be difficult for LBIST to detect because of the extra constraints put on the ATPG tool to set up and maintain the initialization values as well as the launch values used for delay fault testing.

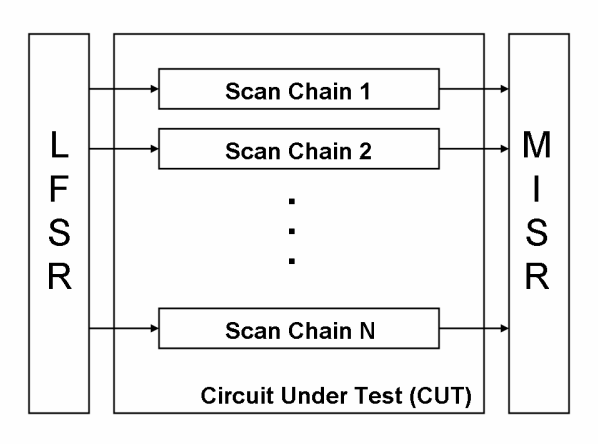

Figure 1.5: LBIST Architecture

Many improvements to the original concept of LBIST have been made in the area of LFSR reseeding. This test compression technique, first proposed in [Könemann 91], takes advantage of the fact that generally only about 1%-5% of the total possible bits (free-variables) that could be assigned for a given set of test patterns are specified (required to be set to a '1' or a '0' value) in order to detect a given set of faults. Based on the polynomial used to construct the LFSR, linear equations can be created and solved to determine the seed needed to be loaded into the LFSR to create or fill in the specified bits. Then the rest of the bits are filled in with essentially random data from the LFSR.

An example of this is shown by the test vector 1011010101110011, where the underlined bits are the bits required to be specified to detect the targeted faults, and the other bits are the ones randomly filled in by the LFSR. Once the full pattern has been filled out, then it is fault graded to determine the total number of faults detected for that pattern.

Most current test compression schemes, like LFSR Reseeding, use a decompressor that, in general, uses (e.g. expands) data from a smaller number of tester (ATE) channels to fill a larger number of internal scan chains per clock cycle. The ratio of internal scan chains to tester channels is called the expansion ratio. The higher the expansion ratio, the greater the potential for test compression (i.e., reduction in test data stored on the ATE) and test time reduction. Test time is reduced because with a progressively higher expansion ratio and a fixed number of scan cells (i.e., scan flip-flops), all of the scan chains become shorter, resulting in fewer clock cycles being needed to load and unload them. The amount of test data, measured in bits, stored on the tester is calculated by multiplying the number of tester channels used by the number of clock cycles of test data. Since the number of tester channels remains constant, but the number of clock cycles to load and unload the scan chains is reduced, then the amount of test data (i.e., test data volume) is also reduced.

Higher expansion ratios increase the potential for higher amounts of test compression, but they also make it more difficult to create test vectors, which are stored on the ATE as compressed test data, that detect as many faults as test vectors created for designs implementing test compression with smaller expansion ratios. This is due to the fact that with smaller expansion ratios, the scan flip-flops are sharing fewer bits (i.e., free-variables) from the ATE test data. With the lowest expansion ratio, a ratio of one, no scan flip-flops are sharing any free-variables from the tester, and the maximum amount of controllability can be achieved. However, as the expansion ratio increases, the values

loaded into the scan flip-flops are dependent on more and more of the same free-variables from the tester, which increases the probability of creating a situation where certain faults cannot be detected. For example, if a 2-input AND gate is driven by two scan flip-flops dependent on the same free-variable from the ATE, then it would be impossible to detect a stuck-at one fault on either input since the test for a stuck-at one in this situation requires opposite values on the two inputs, which is not possible in this case. For this reason, conventional decompression techniques typically pick an expansion ratio which is selected at a level where test vectors can be generated that will detect all or almost all of the faults detected without decompression. Additionally, a mechanism for bypassing the test compression system is typically implemented; this allows the application of uncompressed test vectors to provide a way to apply additional, or top-off, vectors to detect any faults that cannot be detected by the compressed test vectors. Uncompressed vectors applied in bypass mode require much more tester storage and test application time compared to compressed test vectors. Consequently, to maximize the overall compression, conventional approaches must select a low enough expansion ratio to ensure that very few, if any, vectors need to be applied in serial mode.

1.4 HANDLING UNKNOWN VALUES

The types of test compression systems mentioned above have been proven to provide high levels of test data compression as long as all logic values in the output response data contain known values. However, if unknown values (i.e., non-deterministic values) are present in the output response data and enter the output response compactor, then the output of the compactor might become wholly, or partially, undetermined, which could prevent the detection of some faults.

There are many common sources which may introduce unknown logic values (*X*'s). These include unmodeled logic (black-boxes), floating tri-state outputs, bus

contention, and uninitialized memory, as shown in Figure 1.6. Additionally, many improvements to the delay fault testing algorithms have been implemented, including making the algorithms timing-aware and using information on false and multi-cycle paths to enable the creation of more robust test vectors. Using these new algorithms, any time a false or multi-cycle path is activated, an *X* is added on that path. Also, the algorithms will, as a side effect of creating a test vector for targeted paths, activate other extraneous paths. These paths might be related to the targeted paths, or they might be in other parts of the circuitry activated by the bits of the test vector that were not specified but were randomly chosen as mentioned in Section 1.3. If any of these paths do not meet timing, then *X*'s are added there as well.

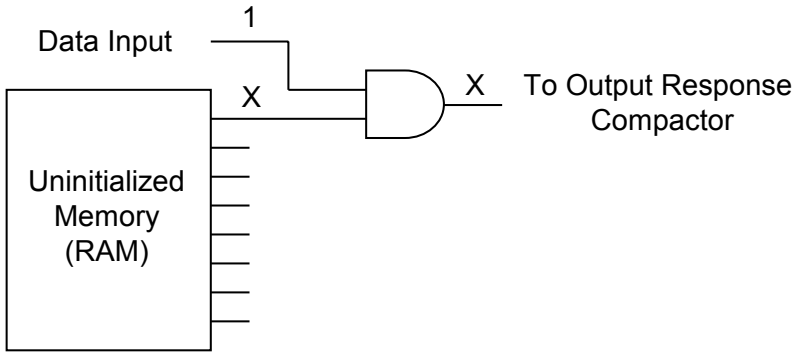

Figure 1.6: Uninitialized Memory (RAM) as an *X* Source

Once these unknown values are introduced into the output response data, the performance of the test compression schemes can be significantly degraded. For the class of output compactors called *time compactors*, MISRs for example, once an unknown is introduced, the signature quickly becomes corrupted and must be reset. For the class of output compactors called *space compactors*, for example XOR trees, once an unknown is introduced, detected faults in the current clock cycle entering into the compactor may become invalidated by the unknown. In both cases, faults that should have been detected

may no longer be detected. This causes the need for new patterns to be created to detect the invalidated and undetected faults and/or new techniques to be employed to deal with the *X*'s.

There are three basic approaches for handling *X*'s: *X-bounding*, *X-masking*, and *X-tolerating*. The first approach, *X*-bounding, as described in [Wang 06], involves analyzing the circuit under test (CUT), identifying all of the sources of unknowns, and adding logic to block the unknowns from propagating to the outputs. The second approach is *X*-masking, which involves blocking, or masking, the *X*'s in the output response data at the input of the compactor each clock cycle. Several techniques to accomplish this have been proposed that include different designs for the masking hardware as well as different methods for compressing the mask and control data such as shown in [Barnhart 01][Wohl 01 03a 04 07][Pomeranz 02][Chickermane 04][Naruse 03][Volkerink 05][Chao 05][Tang 06][Rajski 06a]. In several of these techniques, the resolution of the mask is set low to keep the volume of mask data at a minimum; the trade-off, however, is that some specified bits needed to observe faults are also masked. Some other techniques like [Naruse 03] have a very high mask resolution at the cost of a larger amount of mask data. The third approach is using *X*-tolerant schemes such as those described in [Mitra 04a 04b][Rajski 05][Touba 07] that can still compact the output response data even in the presence of unknowns, but they have limitations, including possible restrictions on the number of *X's* that can be tolerated before the required bits for fault detection get corrupted.

1.5 SUMMARY

This dissertation describes a new approach for testing for delay defects, including smaller delay defects, and a novel technique for compressing the large test data volume that goes along with that. Additionally, new methods for increasing output compaction in

the presence of unknowns (*X's*) are presented. This dissertation is organized as follows: Chapter 2 describes an new enhanced timing-based technique for testing small delay defects which takes advantage of the fact that transition delay ATPG can easily create patterns for node-based delay defects, and it then uses the timing information gathered from STA (Static Timing Analysis) to ensure that the patterns that test given paths are grouped with like patterns and are tested at near maximum frequency. Chapter 3 presents a novel methodology for improving the amount of compression achieved by continuous-flow decompressors by using multiple ratios of scan chains to tester channels (i.e., expansion ratios) and dependency analysis to determine the optimal way to construct the scan chains for multiple expansion ratios for maximum fault detection. Chapter 4 describes a new methodology for using the *X*-canceling MISR architecture, based on deterministically observing scan cells, which provides a higher amount of compaction and is effective for larger percentages of *X's* in the output response. This chapter also describes a novel hybrid approach that combines *X*-masking with an *X*-canceling MISR for further improvements. Chapter 5 presents a new technique for increasing output compression in the presence of unknowns by using reiterative LFSR based *X*-masking. This approach takes advantage of the data correlation in the output response data to enable LFSR encoded masks to be reused for multiple scan slices, while guaranteeing that all unknowns are masked and all bits required for fault detection are allowed to propagate to the compactor. This chapter also presents a novel hybrid approach that combines conventional LFSR based *X*-masking with fixed-interval reiterative LFSR *X*-masking. Experimental results are presented at the end of each chapter to demonstrate the effectiveness of each new technique on industrial designs. Finally, Chapter 6 presents the conclusions, summarizes the work in this dissertation, and proposes future research topics.

Chapter 2: Enhanced Timing-Based Transition Delay Testing for Small Delay Defects

With standard transition delay fault testing targeting only gross delay defects and with the problems associated with path delay testing for small delay defects, there is still a need to obtain high coverage of small or smaller delay defects using some other method. This chapter addresses that need by proposing a new technique called Enhanced Transition Delay Test (ETDT) which was published in [Putman 06]. Using this technique, standard commercially available static timing analysis and transition delay ATPG tools along with the ETDT wrapper can be used to obtain a high test coverage of the small delay defects that lie along the critical paths of a given design. With this technique, the user has the ability to create transition patterns and adjust their timing such that all of the paths tested in a given pattern are tested with almost no slack, similar to path delay testing. Some of the other advantages of this technique include a test coverage and pattern count similar to regular transition delay ATPG on the same faults, greater visibility in the actual coverage of the targeted faults, and reduction of some of the design constraints that regular transition delay ATPG requires. One such requirement of transition delay ATPG is that clock domains running at frequencies that are significantly different should not be tied to the same scan clock in order to maximize the effectiveness of the transition delay test, since the clock can then only run as fast as the slowest clock domain. ETDT removes this requirement by having the ability to group patterns together that test paths with similar path timings or slack. A basic depiction of the new flow is shown in Figure 2.1.

Papers that address the need for testing small delay defects using the transition delay model or a transition-like delay model include [Kruseman 04] and [Gupta 04]. In

[Gupta 04], the authors introduced a new transition fault model called As Late As Possible Transition Fault (ALAPTF) model. Their fault model is implemented by launching one or more transitions towards the fault site as late as possible, launching the transitions down the longest robust segment ending at the fault site. The drawbacks to this method are the use of multiple transitions, which means that faults can only be tested at the speed of the slowest transition; the lack of actual path timing information, so depending on the slack of their longest segments, they still may not be detecting small delay defects; and the difficulties, similar to path delay, finding long segments that can be robustly tested. In [Kruseman 04], the authors describe a technique for testing small delay defects by running a traditional transition delay ATPG, creating a full set of patterns to test those faults. Then, the entire pattern set is copied as many times as the number of frequencies that the patterns are going to be tested at. Each frequency is a multiple of the nominal frequency (i.e., 1X, 1.5X, $2XF_{nom}$). So, if the user wants to test at 10 frequencies, then he would have to have 10 copies of the pattern set. Once the pattern sets are created, each pattern is simulated at its assigned frequency, starting with the highest. After each simulation, the failing flip-flops and primary outputs are masked out. Keep in mind that the more frequencies that can be tested, the better resolution there will be to catch the smallest delay defects. Obviously, the major drawback to this technique is the pattern count. Given the finite amount of memory on a manufacturing tester and the fact that transition delay patterns are typically 3-5X the size of the stuck-at (DC scan) patterns for a given design, pattern count is a real concern.

The remaining portion of this chapter is organized as follows. Section 2.1 discusses the use of transition delay ATPG to test for small delay defects. Section 2.2 covers the overview of the technique. Section 2.3 reports the experimental results.

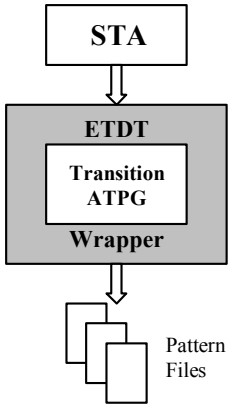

Figure 2.1: Basic ETDT Flow

2.1 USING TRANSITION DELAY ATPG TO TEST FOR SMALL DELAY DEFECTS

To test for path delay faults, the user specifies the complete path, including where the fault effects are observed (i.e., the path endpoint). These specific constraints on the ATPG tool are part of the reason it is difficult to create robust patterns for all specified paths. With transition delay ATPG, the user specifies the fault, but the tool chooses the path of the transition through the fault site along with the path endpoint. The common complaint about transition delay testing is that it is always taking the easiest, and usually the shortest, path [Li 89]. This works fine for gross delay defects but makes it nearly impossible to detect small delay defects. One common modification that test engineers make to their transition delay patterns is to increase the frequency (i.e., over-clock) of each scan clock just to the point of failure. This allows those patterns to detect defects smaller than gross delay defects in general; however, the scan clock can only be increased up to the point that the slowest activated path fails. Thus, many other paths tested in those patterns still have a large amount of slack in them and are therefore only good for large or gross delay defects. An example of this is found later in the chapter in Figure 2.2 on page 28.

In order to test for small delay defects, mostly of the point type variety, the technique proposed in this chapter allows the user to generate a fault list based on path information obtained from static timing analysis of the design's critical paths and then allows the ATPG tool to create a transition test for that fault down a path through the fault site to an observation point (i.e., path endpoint) of its choosing. The path information from the ATPG tool is then compared to the path information obtained from the static timing analysis run, and the slack for that path is reported and doubled if necessary. Once the slack for a given path (or pattern) is known, then the period of the launch and capture clocks can be adjusted such that the pattern is run just to the point of failure so that even very small delay defects can be detected. In order to adjust the launch and capture clocks properly, the slack is doubled if, and only if, the clocks at the launch and capture points are of opposite polarity, indicating a rising edge launch to a falling edge capture or vise-versa. The calculation for the new period looks like:

$$\text{Period}_{\text{new}} = \text{Period}_{\text{orig}} - \text{slack}$$

The result of using transition delay fault ATPG in conjunction with path timing information gathered from static timing analysis is that a higher number of faults can be tested than with path delay methods, and the faults down each transition path can be tested at their maximum, or near maximum, speed, thus enabling the detection of small delay defects.

2.2 ENHANCED DELAY TECHNIQUE DESCRIPTION

The general flow of this technique is to run static timing analysis to determine path and slack information, generate a fault list from that path information for the ETDT wrapper, and run the ETDT scripts, which create the patterns, bin the patterns based on the slack of the path down which the transition is propagated, merge the patterns, and

write the patterns in standard IEEE STIL format which can then be read back into a standard ATPG tool or converted directly to a manufacturing tester format.

2.2.1 Gathering Timing Information based on STA

The first step of this technique involves running static timing analysis to determine the critical paths of the design and their slack, reporting the slack and doubling it if necessary, as mentioned above, and then storing that information. This step is similar to how one would begin determining critical paths for path delay testing, and no more time is spent on it than would be spent for the path delay method. In fact, one can use the same run time to gather information for path delay as for ETDT. One possible flow would then be to do path delay testing on the critical paths, getting the easy paths. Then, fault grade the patterns against the transition delay fault model and remove those detected faults from the fault list that ETDT is using so that those faults are not redundantly tested. Finally, proceed with the proposed ETDT technique. For this chapter, the path delay step was skipped, but it would be an obvious extension of the proposed technique.

Based on experience, a typical transition delay testing ATPG run would be constrained such that primary outputs are masked and primary inputs are constrained to not toggle between the launch and capture transition clocks. Additionally, both the launch and capture clocks are the same clock. This is primarily done to ensure that solid patterns can be created with known timing guaranteed by design that will pass on the manufacturing tester. These constraints guarantee that transitions are not launched through the input/output pads on a given chip that are typically slow at the same time as transitions are launched through the internals of the chip that can run much faster. These types of constraints also help avoid testing false paths that usually lie between two different clock domains. While this is great for generating patterns that pass and work on the manufacturing tester, it leaves two problems: 1) The test coverage for the designs are

low, due to the constraints put on the ATPG mentioned above. 2) There really is not any visibility into how well the ATPG tool is covering the parts of the design that are not constrained (i.e., the faults inside each clock domain). Using static timing analysis to gather path information and generate the fault lists for this technique gives the user much greater control of the fault lists and better visibility into how well the ATPG tool is covering the targeted faults. For example, the user can set up the static timing analysis tool and ETDT scripts to gather path information and create fault lists for paths between all primary inputs and each clock domain, between clock domains and all primary outputs, between each clock domain, and for each clock domain. Then ETDT can be run on each group separately, for maximum visibility of test coverage, or all together, for minimum pattern count.

There are a number of ways that the static timing analysis tool can be set up to gather critical path information and a number of ways the critical path cutoff limit can be set. For example, the criteria for cutoff can be based on percentage of cycle time, a fixed slack value, or the size of a suspected delay fault [Crouch 03]. For this technique, the number of critical paths based on slack was the limiting factor.

2.2.2 Creating the Patterns

For the "create patterns" portion of the flow, faults are deterministically targeted essentially one fault at a time. First, a pattern is created for one fault from the fault list. Next, the pattern is altered such that all observation points are masked out except one path endpoint where the transition through the fault site is observed. Then, the pattern is fault simulated to see what other faults along the transition path can be detected. Lastly, the pattern is saved with just the pertinent information being stored including path information and pattern data.

One of the important functions of this technique is the altering of patterns created by the standard transition delay ATPG tool mentioned above. It was observed that when creating a transition delay pattern for a single fault, in some cases, the transition could be observed or captured at several locations, all with different path timings and slack values. In order to limit the transition to one observation point, so that the exact path timing to that endpoint could be known, all of the other observation points had to be masked out. Additionally, it was observed that due to the nature of how the transition delay patterns were created, there were often other paths with different slack values and path timings that had transitions on them that were not directly related to the targeted fault. These paths were also taken care of by the masking of all observation points except the path endpoint where the transition through the fault site is observed.

2.2.3 Binning the Patterns

Binning the patterns refers to grouping patterns together based on their slack. In path delay fault testing, each pattern typically represents a test for one path; thus, each pattern has its own unique timing. If a large number of paths are being tested, this can lead to a large pattern count and a large number of unique timing sets. However, using the ETDT technique along with the transition delay ATPG algorithms, these problems can be avoided by selecting a finite number of bins or pattern groups that represent an equal number of unique timing sets. For example, if all the detected faults had path timings of less than 5ns, and the user wanted to group patterns with similar path timings and slack, then five bins could be created (0-1ns, 1-2ns, 2-3ns, 3-4ns, and 4-5ns) and the appropriate patterns placed therein.

For the technique proposed in this chapter, the bins are completely user defined and can be set up with any number of bins with any resolution or bin size. In addition to reading in the user defined bin information, the STA database that was created in the first

step is read in and accessed. With this information, the created patterns are binned appropriately. Any patterns that have a path timing that doesn't fall into one of the specified bins are put into a separate bin for patterns with unknown path timings. The timing for those patterns can be determined separately or the bins and/or constraints on the initial STA can be adjusted. At the very least, the patterns in the "unknown" bin, which are usually just patterns testing very short paths with lots of slack, can be run at a frequency corresponding to the upper limit of the last bin.

Also, note that the narrower the bins are, the smaller the size of delay defect that can be detected. However, making the bins narrower usually implies increasing the number of bins. Increasing the number of bins will have some impact on pattern count, since increasing the number of bins reduces the effectiveness of merging the patterns.

2.2.4 Merging the Patterns

Even though each created pattern tests a given path and can detect a number of faults, pattern merging is required in order to end up with a compact set of patterns that can be used in production testing on a manufacturing tester. It is the goal of the proposed technique to create a set of patterns that would be similar in size to a set of patterns created by the same commercially available transition delay ATPG tool testing for gross delay defects, given the same fault set.

The merging technique used in this chapter is just a very basic technique such that for each bin it selects one pattern and tries to merge it with as many other patterns as possible in that bin. For the proposed technique, all pattern data is considered for merging, including the scan input bits, primary input bits, primary output bits, the scan output bits that are the endpoints to tested paths, and the scan out bits that ultimately get masked out. Patterns are merged only if all bits can be merged. If any bit of one pattern conflicts with another pattern, then the merging of those two patterns is aborted and

another pattern is tried. Successful merging of individual bit positions happens when the bit of one pattern is the same as the bit of the other pattern or the bit of one pattern is specified, while the bit of the other pattern is not.

As mentioned above, the scan out bits, and their values, that are to be masked out are maintained even during merging. This is done to avoid contention and faulty patterns, since those masked scan out bits will have real values in them once the pattern is executed.

2.2.5 Writing the Patterns

Once the patterns are binned and merged, the patterns in each bin are combined and written out in standard IEEE STIL format. Then, for each STIL file, the launch and capture waveform tables are modified such that the time between the leading edges of the launch and capture clocks equals the $Period_{new}$ as calculated above. The slack in that calculation should correspond to the lower limit of the bin range (i.e., 2 ns if the bin range is 2-3 ns), and the time between the leading and trailing edges of the launch and capture clocks should be set to $Period_{new}$ divided by two so that a 50% duty cycle is achieved. Note that typically the actual period for a launch and capture waveform is usually much larger that $Period_{new}$. After the waveform tables are set up correctly, the STIL files can then be imported directly into the tester, if it is a STIL compatible tester, or read back in to the ATPG tool. If the files are read in to the ATPG tool, then they can be fault simulated, written out as a simulation test bench for verification, or written out in one of the other available manufacturing tester formats.

2.3 EXPERIMENTAL RESULTS

To demonstrate the viability of the technique proposed in this chapter, experiments were performed on two industrial mixed-signal designs. Some of the key

characteristics of each design and standard scan tests performed are presented in Table 2.1. While neither design is on the cutting edge of the latest fabrication geometries, both designs may provide future data about small delay defects at those geometry levels.

Table 2.1 shows the DC (stuck-at) and transition delay scan's test coverage and pattern count. For Design 1, the transition delay test coverage for all faults in the design is 70.6%. This low number is primarily due to a restrictive setup where the I/O's are constrained due to their slow nature when compared to the rest of the circuitry, and the launch and capture clocks are always restricted to be the same clock to avoid false paths between clock domains. The result is a loss in coverage between clock domains as well as between the I/O's and the clock domains and between the I/O's themselves. Additionally, since all of the faults are lumped together, it is unknown how well each clock domain is being covered. However, using the flow of the new technique, it is now possible to partition the faults based on clock domain (as seen in Table 2.2), between different clock domains, between clock domains and I/O's, and between the I/O's for increased visibility of their test coverage.

Table 2.1**:** Key Scan Design Characteristics

Parameter	Design 1	Design 2
Technology	250 nm	180 nm
Num. of Flip-flops	6988	13,559
Longest Scan Chain	3494	849
DC Scan Coverage	97.3%	98.8%
DC Scan Faults	294,808	898,796
DC Pattern Count	601	1826
Transition Delay Cov.	70.6%	86.9%
Transition Delay Faults	294,808	898,796
Transition Pattern Cnt.	5237	4278

2.3.1 STA Results

The static timing analysis portion of this technique was performed on the fastest clock domains with the least amounts of slack. For purposes of these experiments, two clock domains from Design 1 and one clock domain from Design 2 were chosen. For production use of this technique, all areas of a given design (i.e., all clock domains, I/O's, etc.) should be checked for critical paths to be considered for small delay defect testing.

As a starting point, after the clock domains for this experiment were identified, the clocks were adjusted until the worst case slack was less than 1 nanosecond (ns). Next, the cutoff for the slack was set such that between 2K-6K faults were being identified per domain as shown in Table 2.2. For production use of this technique, a cutoff for slack can be chosen to represent a percentage of the total faults in the design, a percentage of clock cycle, a time relative to a typical delay defect for a given fabrication process, or any number of other factors.

All scripts and tools were run on a 64-bit dual processor Linux machine running at 2 GHz with 8 GB of RAM. For STA, each job was completed within 2 hours. If necessary, for larger designs with more faults, STA jobs can be run in parallel.

2.3.2 Test Coverage Results

For each clock domain's fault list, Table 2.2 shows the regular transition delay ATPG results along with results from the ETDT technique. In the "creating patterns" portion of the flow, all observation bits are masked out except the one endpoint where the propagated transition is being observed. The ETDT "no mask" results in the third column represent the results for the technique if the above masking of observe bits is not performed. These results should more closely match those of regular transition delay ATPG with the differences resulting from running all the faults at once in the tools

"automatic" mode versus running ATPG on each fault one at a time and then separately fault simulating the pattern to see what additional faults were detected. As seen in Table 2.2, two domains showed a difference of less than .5%, while Design 1, Domain 1 showed a difference of 3%.

The fifth column of Table 2.2 shows the test coverage results of the ETDT technique as it was meant to be run. The loss in coverage when compared to the ETDT "no mask" results are slight, being less than 0.3%.

Run time for the smallest number of faults of the three domains, 2294, was completed in about 3 hours, where the largest number of faults, 5631, completed in about 5.5 hours. For larger designs with more faults, the different fault lists from the different fault partitions can be run in parallel to minimize overall run time.

Table 2.2**:** Test Coverage Results

	Number of Faults	Regular Transition Test Coverage	ETDT "No Mask"		Standard ETDT	
			Test Coverage	CPU Time	Test Coverage	CPU Time
Design 1, Domain 1	5631	95.7%	92.7%	3 hrs. 40 mins.	92.5%	5 hrs. 34 mins.
Design 1, Domain 2	4253	82.9%	82.6%	3 hrs. 02 mins.	82.4%	4 hrs. 50 mins.
Design 2, Domain 1	2294	82.7%	82.7%	1 hr. 25 mins.	82.5%	3 hrs. 02 mins.

2.3.3 Pattern Count Results

Table 2.3 shows the pattern count results for regular transition delay ATPG as well as that for ETDT. For regular transition delay ATPG, the default merging values were used, and, as expected, the pattern count was small (< 300 patterns) considering the number of faults tested. Column 2 of Table 3 shows the number of unmerge patterns that the ETDT technique created. The next column shows results of merging all the patterns together regardless of timing information. Although this action is similar to regular

transition ATPG, these patterns only have one observe bit not masked per path tested. Even so, the pattern count is fairly close, with the major difference being the ATPG tool's superior merging capability done during ATPG versus this technique's greedy merging technique that is done post-ATPG.

The last three columns show what effect increasing the number bins has on pattern count. In all three cases, going from 1 timing bin (similar to regular transition delay ATPG) to 6 timing bins only increases the pattern count by less than 1.5X. Certainly, though, as the number of bins grows, the less efficient the merging will be since the merging algorithm will have fewer patterns to work with.

Table 2.3: Pattern Count Results

	Regular Transition Pattern Count	ETDT Pattern Count				
		Unmerged	1 Bin	2 Bins	4 Bins	6 Bins
Design 1, Domain 1	107	2311	261	271	347	368
Design 1, Domain 2	274	1570	367	471	491	531
Design 2, Domain 1	172	390	232	252	260	263

2.3.4 Overall Results

In the end, what really matters is how well the small delay defects were able to be detected and to improve the quality of test for the targeted faults. Figure 2.2 shows all of the paths tested for Design 1 Domain 1 that have a slack of less than 10 ns, grouped into one bin just like for regular transition delay testing. Path number 1 has a slack of 0.8 ns, while the last path, 2283, has a slack of 9.9 ns. What this means is that for path number 1 to fail, at this clock frequency, delay defects totaling more than 0.8 ns must be present. Likewise, for the last path, the total size of the delay defects would have to be greater than 9.9 ns to be caught. Looking at the graph of all of the paths between the first and the

last path, it is easy to see that this type of testing (i.e., regular transition delay testing) does not do much for detecting small delay defects.

Figure 2.3 shows what happens when the proposed technique is applied, and the patterns are sorted into five bins each 2 ns wide (0-2, 2-4,4-6, etc.). It is easy to see now that in this configuration no path has a slack of more than 2 ns. Also, note that the timing for the first 248 paths remains unchanged. However, for the other 89% of the paths, the quality of test has been greatly improved. Additionally, with smaller bin sizes, even smaller delay defects can be detected.

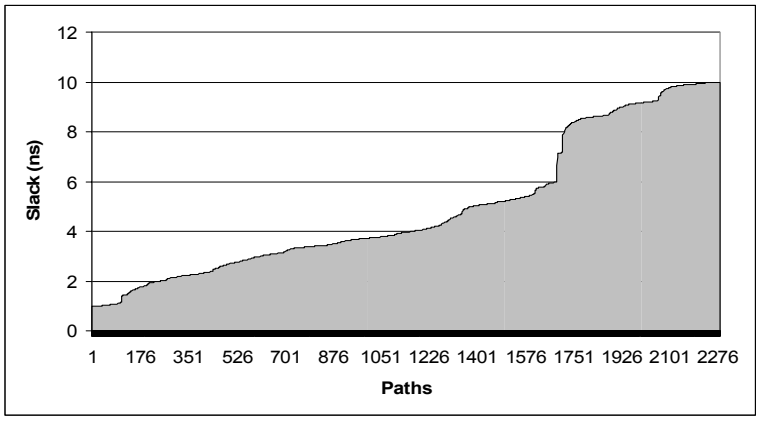

Figure 2.2: Path Slack for Transition Delay Testing (1Bin)

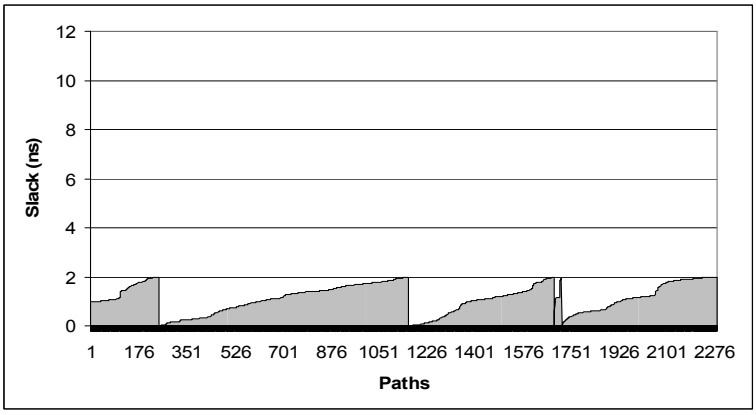

Figure 2.3: Path Slack for 5 Bins

Chapter 3: Using Multiple Expansion Ratios and Dependency Analysis to Improve Test Compression

The importance of delay fault testing, and more specifically smaller delay defects, is only increasing as technology continues to shrink. As shown in the previous chapter, there is a better way to perform delay fault testing that not only improves the detection of smaller delay defects, but also gives a greater visibility of the fault coverage of the individual clock domains. However, to take advantage of this new technique, or even standard non-enhanced delay fault testing techniques, some amount of test compression is needed to deal with all of the test data/vectors that delay fault testing generates, especially for large industrial designs.

This chapter proposes a methodology, published in [Putman 07], for using multiple expansion ratios to improve the overall amount of compression. The idea is to start with a higher expansion ratio than normal and then progressively reduce the expansion ratio to detect any faults that could not be detected at higher expansion ratios. By detecting each fault at the highest expansion ratio that it can be detected, the amount of compression can be significantly improved compared with conventional approaches that use a single expansion ratio. The expansion ratio is progressively reduced by concatenating scan chains using MUXes to make fewer and longer scan chains. Another key idea in the chapter is to exploit the flexibility in choosing which scan chains to concatenate. The scan chains that are concatenated are selected in a way that maximizes the potential for detecting additional faults. This is done by using a dependency analysis procedure that takes into account structural dependencies among the scan chains as well as free-variable dependencies in the logic driving the scan chains. Each bit on the tester can be considered as a "free-variable" which can be assigned either a 0 or 1. Free-

variable dependencies occur when the contents of two scan cells depend on the same free-variable on the tester. By analyzing all of the dependencies, the proposed procedure determines which scan cells to concatenate in order to minimize potential conflicts during ATPG (automatic test pattern generation). This increases the probability of detecting more faults at higher expansion ratios.

The proposed methodology does not have any impact on scan cell ordering/stitching. A conventional design flow can be used for obtaining the initial design with the maximum expansion ratio. The proposed methodology only involves inserting MUXes in the design to provide the ability to concatenate scan chains for different expansion ratios. Note that this is already conventionally done in most commercial test compression schemes to provide for a serial mode. In fact, regardless of the number of expansion ratios used, the proposed method does not use any additional MUXes over what is already used for serial mode. The only difference is that for serial mode, only one control signal is used, but for this method, c control signals are required, where c is the number of expansion ratios. Hence, implementing the proposed methodology is very simple and adds very little overhead. It can be used in conjunction with a number of existing test compression schemes including Illinois scan [Hamzaoglu 99], reconfigurable fanout networks [Pandey 02], [Samaranayake 03], [Sitchinava 04], [Tang 03], Virtualscan [Wang 04], combinational linear decompressors [Bayraktaroglu 03], [Mitra 06], and sequential linear decompressors [Könemann 01], [Rajski 04], [Dutta 06].

There has been some earlier work that also tries to minimize dependencies to reduce conflicts. In [Samaranayake 03], structural analysis is used during scan cell ordering to minimize potential conflicts for a fanout network based decompressor. The structural analysis used here is similar in nature, but is used for scan chain concatenation

and not scan cell ordering. The scan cell order is assumed to be optimized for other criteria (e.g., layout area) and is not modified. Moreover, the proposed method also takes into account more complex free-variable dependence and can be used with decompressors containing XOR gates. In [Shah 04], scan chains are grouped together for Illinois scan with multiple inputs based on compatibility analysis on a set of test patterns. The proposed approach is also based on grouping together scan chains, but the dependency analysis is based on structural analysis and takes into accounts more complex free-variable dependence.

The remaining portion of this chapter is organized as follows. Section 3.1 discusses multiple expansion ratios and how they are implemented. Section 3.2 covers dependency analysis and the cost function. Section 3.3 describes how the cost function is used. Section 3.4 reports the experimental results.

3.1 MULTIPLE EXPANSION RATIOS

To implement this multiple expansion methodology, the first step is to create the first expansion ratio by inserting a large number of scan chains into a given design, with lockup-latches at the end of each chain to avoid clock-skew problems. The number of scan chains inserted depends on the amount of compression desired. Here, the more scan chains that are inserted, the higher the overall compression ratio; however, the trade-off is the extra amount of area for the decompressor logic in front of each additional scan chain, along with the few MUXes needed to reconfigure the scan chains into different expansion ratios and the associated routing.

Then, after the scan chains have been created, some logic, depending on the decompressor that is used, is added in front of every scan chain. For example, if combinational linear decompression was being used with 2-input XOR gates, then one 2-input XOR gate would be placed in front of every scan chain. On the other hand, if

Illinois scan was being used, then each scan chain would be driven by just a fanout network.

The second expansion ratio, and all subsequent follow-on expansion ratios, is created by concatenating pairs of scan chains using MUXes. In this configuration, each follow-on expansion ratio is one-half of the previous expansion ratio. Figure 3.1 shows how this can be implemented by using MUXes. *X1-X4* are tester channels driving a large set of scan chains. Only the first four scan chains are shown in the figure. The control lines to the MUXes come from two registers (alternatively they could come from primary inputs).

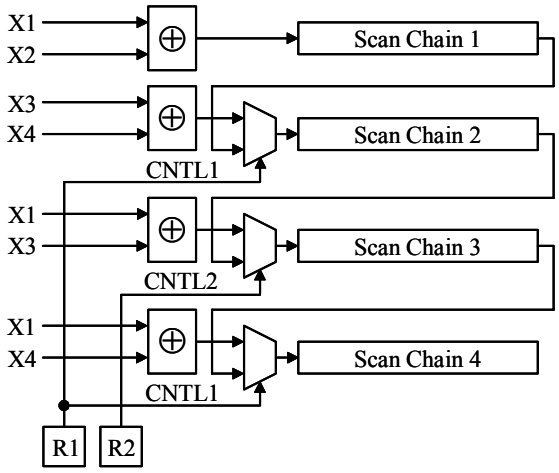

Figure 3.1: Multiple Expansion Ratio Implementation

When both control signals are low, each scan chain is driven by its own XOR gate. When the first control signal goes high, then the XOR gates at the beginning of scan chains 2 and 4 are disconnected, and the beginning of scan chain 2 is connected (via a MUX) to the end of chain 1. Likewise, the beginning of chain 4 is connected to the end of chain 3. This reconfiguration reduces the expansion ratio by a factor of 2. Next, if the second control signal is also brought high, then in this example the XOR gate in front of

chain 3 is disconnected, and the beginning of chain 3 is connected, to the end of chain 2. With this reconfiguration, the expansion ratio is reduced by another factor of 2.

Note that there are many different possible combinations of scan chain pairs to join together to form a new expansion ratio. Additionally, there are two possible orders in which a given pair of scan chains (*A* and *B*) can be concatenated (*A* could come first, or *B* could come first). Which scan chains are joined together and the order in which any pair is concatenated can have a significant impact on the overall test compression. In Figure 3.1, if a random concatenation methodology was used, then scan chains 1 and 2 might get joined together as well as 3 and 4. In this case, the two scan chains created for the new expansion ratio are driven by $X1 \oplus X2$ for the first combined chain and $X1 \oplus X3$ for the second combined chain. Here, for each scan slice, both scan chains will be depending on the same free-variables coming from tester channel *X1*. To avoid this free-variable dependency, it would be better to concatenate scan chains 1 and 3 and 2 and 4 so that there is no free-variable dependency: in this case, the first combined chain will be driven by $X1 \oplus X2$, and the other will be driven by $X3 \oplus X4$. This illustrates how the degree of freedom in choosing which scan chains are concatenated can be used to maximize the overall compression. A systematic approach for exploiting this is described in the following section.

3.2 DEPENDENCY ANALYSIS

Conflicts that can prevent a fault from being detected at a particular expansion ratio arise when a fault requires a set of scan cells to have certain values, but the scan cells cannot be loaded with those values because some of the scan cells depend on the same free-variable. This type of conflict does not arise in serial (uncompressed) mode because in that case each scan cell depends on a unique free-variable.

There are two types of dependencies that both must be present in order to have a conflict. One is that detecting a fault must structurally depend on two scan cells, and the other is that the two scan cells must depend on the same free-variable. This is a necessary but not sufficient condition for a conflict. The dependency analysis proposed here is based on minimizing the number of potential conflicts by trying to minimize the number of scan cell pairs that have both structural and free-variable dependency present. A novel cost function is used which takes into account not only the existence of dependency, but also the degree of dependency. The cost function is then used as a greedy heuristic to guide the selection of which scan chains to concatenate.

Linear Equations for Scan Cells

$SC_1 = X_1 \oplus X_2$ $SC_3 = X_3$

$SC_2 = X_2 \oplus X_3$ $SC_4 = X_3$

Degree of Free-Variable Overlap

$F_{1,2} = 1/3$ $F_{1,3} = F_{1,4} = 0$

$F_{2,3} = F_{2,4} = 1/2$ $F_{3,4} = 1$

Figure 3.2: Example of Calculating Degree of Free-variable Overlap

For a continuous-flow linear decompressor, each scan cell depends on a linear combination of the free-variables on the tester. Illinois scan is a degenerate case where each scan cell depends on a single free-variable. In the proposed cost function, the degree of free-variable dependency is measured by the number of free-variables the linear equations for two scan cells on two different scan chains have in common divided by the total number of distinct free-variables present across both of their linear equations. Consider the example shown in Figure 3.2. Scan cells *SC_1* and *SC_2* have one free-variable in common *(X_2)* and they have a total of three distinct free-variables present across both of their linear equations *(X_1, X_2, X_3)*, hence the degree of free-variable overlap is computed as 1/3. The maximum amount of free-variable overlap possible is 1

which occurs when two scan cells will always have the same value. For Illinois scan, free-variable overlap will always be either 0 or 1 since every scan cells depends on only a single free-variable. However, for linear decompressors that contain XOR gates, the free-variable overlap can be a fraction. The higher the free-variable overlap is, the less flexibility there is to independently control the value of two scan cells.

The second part of the cost function is structural dependency. The degree of structural dependency is measured by looking at logic cone overlap. A one-time backwards trace from endpoints consisting of all primary outputs and the functional data input of all scan cells is performed, and then all of the scan cells driving the data cones of these endpoints are identified. Then a one-time record can be created to use with the cost function algorithm that keeps track of the number of times a given scan cell is in a logic cone with another scan cell. This one-time cone analysis is relatively quick to perform.

The dependency cost function for a pair of scan chains is computed as follows:

$$\text{Cost for Pair of Scan Chains} = \sum_{i,j} (F_{ij}^{e} * S_{ij}^{k})$$

where Fij is the degree of free-variable overlap and Sij is the amount of logic cone overlap between scan cell i in one scan chain and scan cell j in the other scan chain. These numbers are multiplied together so that if either value is zero, then the cost is zero since there is no chance for a conflict. The variables e and k are added as tuning parameters to adjust the relative weighting of the two parts of the cost function to optimize the results for different circuits and different test compression schemes. These values can be calibrated for a particular test compression scheme. In Figure 3.3, the compression ratio that was obtained for different values of e and k were plotted for Design A (which is one of the circuits used in the experimental results reported in Sec. 3.4). Here it can be seen that the compression ratio varied some with different values of e

and *k* but the results tended to be better where *e* was greater than or equal to 1 and *k* was less than or equal to 1. This held true for all three test cases over the three compression schemes that were used. The peak in Design A was 12.4X compression for the 2-input XOR combinational only decompressor.

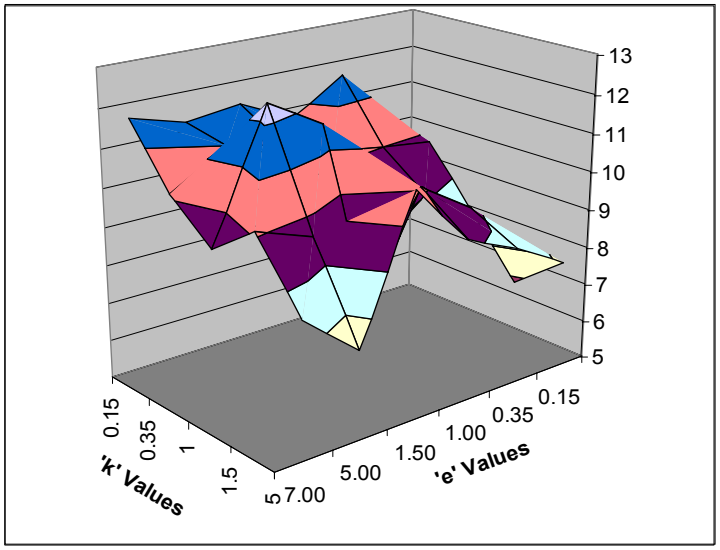

Figure 3.3: Plot of Compression for *Design A* using Different Values of *e* and *k*

3.3 USING DEPENDENCY COST FUNCTION

The cost function can be used to build a new expansion ratio by selecting which pairs of scan chains to concatenate using a greedy procedure. The first task is to find the two scan chains that have the highest dependency cost when compared with each other. To eliminate their effect on each other, they are concatenated to form one scan chain. For the initial highest cost scan chain pair, the second scan chain is just added to the end of the first scan chain. Then, the next highest cost scan chain pair is found. For this pair, and all subsequent pairs, both orders (one scan chain before the other and vise-versa) are considered. The cost function is applied for both orderings against each of the previously

concatenated scan chains pairs for the new expansion ratio. The costs for both orders are summed up and compared. The order resulting in the lowest cost is selected. This continues until all scan chains of the previous expansion ratio have been concatenated. This procedure is repeated for each new expansion ratio.

3.4 EXPERIMENTAL RESULTS

To demonstrate the viability of the proposed methodology, experiments were performed on three industrial designs. Table 3.1 shows the design details, including the number of scan cells and stuck-at faults in each design. The table also includes the number of full serial ATPG patterns required for complete coverage of the detectable faults. The three designs chosen range from a small mixed-signal chip with 4,019 scan cells up to a larger mostly digital chip with 30,776 scan cells. All of the pattern count results presented in this chapter, including the ones in Table 3.1, were generated using a commercially available ATPG tool configured to use a high merge value to maximize dynamic pattern compaction and a limit for the minimum number of faults detected per pattern ranging from one to three.

Table 3.1: Design Details

Design	Scan Cells	Faults	Full Serial Pattern Count
Design A	4019	147,118	347
Design B	13557	696,514	633
Design C	30776	1,417,145	2743

3.4.1 Compression Ratio Results

Table 3.2 shows the compression results for the designs listed in Table 3.1. The designs are indicated in the first column, and the initial number of scan chains inserted into the design are listed in the third column. Three types of decompression schemes

were tested, and they are noted in column two. The first type is the Illinois, or broadcast, scan decompressor [Hamzaoglu 99], and the second type is purely combinational decompression, using a single 2-input XOR gate at the beginning of every scan chain. The third type is the limited dependence sequential linear decompressor proposed in [Dutta 06] that uses two "tester slice registers" and a 2-input XOR gate at the beginning of every scan chain. For each decompressor type, eight external tester channels were expanded to fill the scan chains.

Column four shows the results for a single expansion ratio and a single configuration of the logic driving the scan chains. Any faults not detected with this configuration are detected with top-off vectors in serial mode. Column five shows the results for a single expansion ratio and 4 configurations, where the logic driving the scan chains in each configuration is different. In this case, the ATPG is run on each configuration until it can't detect any more faults. The leftover faults are passed on to the next configuration until all configurations have been used; then, top-off serial patterns are generated for any faults that still have not been detected. Columns six and seven both use a single fixed configuration, but use multiple expansion ratios. The difference between the two columns is that the technique represented by column six generates its second, and subsequent, expansion ratios, and the associated scan chains, by simply concatenating adjacent scan chains, whereas the method reported in column seven uses the dependency analysis procedure described in Sections 3.2 and 3.3 to determine which scan chain pairs are joined together to form the new expansion ratios.

As shown by the results in Table 3.2, in every case, for all three designs, going from a decompressor that uses a single expansion ratio with a single configuration to one that uses multiple expansion ratios (even one that uses a simple method for forming the follow-on expansion ratios) with a single configuration shows a significant improvement

in test data volume compression. Also, using the simply created multiple expansion ratios was at least equal to, but most often significantly better than, using a single expansion ratio with multiple configurations. Additionally, as shown in the comparison of columns six and seven, the compression ratio for multiple expansion ratios created by dependency analysis was always greater, and usually, significantly greater, than the compression ratio resulting from the use of multiple expansion ratios, where the follow-on expansion ratios were created in a simple greedy fashion.

Table 3.2: Test Compression Results

Design	Decomp Type	Num of Scan Chains	Compression Ratio			
			1 Expan Ratio, 1 Config	1 Expan Ratio, 4 Config	4 Expan Ratios, 1 Config	4 Expan Ratios, Depend Analysis, 1 Config
A	Illinois	768	1.1	2.0	2.7	4.6
	xor2 comb	768	1.7	1.9	4.5	12.4
	xor2, 2reg	768	2.4	2.5	7.8	13.1
	Illinois	1024	1.2	1.6	5.7	6.4
	xor2 comb	1024	1.6	1.7	2.9	10.3
	xor2, 2reg	1024	1.9	2.9	2.9	14.8
B	Illinois	768	1.6	3.7	8.5	15.0
	xor2 comb	768	7.4	7.6	14.9	21.9
	xor2, 2reg	768	11.8	16.5	23.5	26.6
	Illinois	1024	1.5	2.2	9.8	16.1
	xor2 comb	1024	5.5	7.0	8.8	17.3
	xor2, 2reg	1024	13.3	20.3	26.9	28.2
C	Illinois	768	12.4	27.5	36.2	36.4
	xor2 comb	768	25.2	24.9	33.0	46.0
	xor2, 2reg	768	25.0	45.8	46.3	49.8
	Illinois	1024	17.0	31.6	29.0	33.7
	xor2 comb	1024	23.5	31.6	34.8	48.0
	xor2, 2reg	1024	25.9	27.9	30.3	54.5

3.4.2 Pattern Count Results

Tables 3.3 and 3.4 show some of the pattern count data results for Designs A and C. The first column shows the number of expansion ratios and configurations used for each experiment. The second column enumerates the configuration or expansion ratio for each reported pattern count. Columns 3, 4, and 5 show the number of ATPG patterns generated for three decompressor schemes. In all cases, the first expansion ratio (or configuration) has the most number of patterns, and each subsequent ATPG run with a new expansion ratio (or configuration) typically generates fewer and fewer patterns. The last run is always the full serial top-off run, and, depending on how many faults are left to detect, the pattern count will vary.

As can be seen in the pattern count results, the ATPG engine has a hard time benefiting from the cheaper first and second expansion ratios (or configuration) when using the Illinois scan decompression scheme. This is primarily due to the constraints it imposes on the ATPG engine, namely the increased free-variable dependencies, when compared to the two other compression schemes that use XOR gates. Likewise, the pattern count results for the 2-input XOR combinational decompressor scheme indicate more constraints on the ATPG tool than the 2-input XOR limited dependence sequential linear decompression scheme, which detects most of its faults with the first two expansion ratios (or configurations) and needs fewer full serial top-off vectors than the other two decompressor types.

Table 3.3: Design A Pattern Count Results

		Pattern Count		
Decompressor Type		Illinois	xor2 comb	xor2, 2 reg
Num. of Scan chains		768	768	768
1 Ratio, 1 Config	1st Expan. Ratio	450	973	1220
	Serial Top-off	301	194	124
1 Ratio, 4 Configs	1st Config.	450	973	1220
	2nd Config.	457	205	75
	3rd Config.	215	78	34
	4th Config.	290	25	3
	Serial Top-off	160	166	120
4 Expan. Ratios, 1 Config	1st Expan. Ratio	450	973	1220
	2nd Expan. Ratio	300	218	205
	3rd Expan. Ratio	307	93	8
	4th Expan. Ratio	309	179	153
	Serial Top-off	74	39	4
4 Expan. Ratios, w/Depend. Ana., 1 Config	1st Expan. Ratio	450	973	1220
	2nd Expan. Ratio	586	485	244
	3rd Expan. Ratio	274	104	5
	4th Expan. Ratio	241	10	1
	Serial Top-off	22	1	0

Table 3.4: Design C Pattern Count Results

		Pattern Count		
Decompressor Type		Illinois	xor2 comb	xor2, 2 reg
Num. of Scan chains		1024	1024	1024
1 Ratio, 1 Config	1st Expan. Ratio	5029	5312	5139
	Serial Top-off	121	74	62
1 Ratio, 4 Configs	1st Config.	5029	5312	5139
	2nd Config.	450	94	136
	3rd Config.	157	9	18
	4th Config.	39	3	4
	Serial Top-off	41	43	53
4 Expan. Ratios, 1 Config	1st Expan. Ratio	5029	5312	5139
	2nd Expan. Ratio	324	93	111
	3rd Expan. Ratio	132	18	19
	4th Expan. Ratio	24	124	13
	Serial Top-off	43	26	43
4 Expan. Ratios, w/Depend. Ana., 1 Config	1st Expan. Ratio	5029	5312	5139
	2nd Expan. Ratio	248	182	123
	3rd Expan. Ratio	136	73	57
	4th Expan. Ratio	137	17	6
	Serial Top-off	24	8	2

Chapter 4: Increasing Output Compaction in Presence of Unknowns using an X-Canceling MISR with Deterministic Observation

The use of a test compression technique like the one presented in the previous chapter is a great way to reduce the test data volume necessary to test large industrial circuits, especially those requiring patterns to test for delay faults. Most test compression schemes consist of three major parts: an input decompressor, the scan chains themselves, and an output compactor used to compact the output response data coming from the outputs of the scan chains. All test compression schemes have to be able to deal with the many sources of unknowns (or *X* values) that commonly arise during ATPG or else the output of the response compactor could become corrupt or the performance severely degraded. As mentioned in a previous chapter, there are a number of ways to deal with unknown values. One such approach is the *X*-canceling MISR that was proposed in [Touba 07] which is based on providing very high probabilistic error coverage by canceling out *X's* in MISR signatures. The error coverage can be made arbitrarily high and match that of using a conventional MISR to compact output responses without *X's*. This approach is highly efficient when the percentage of *X's* is low (e.g., 1% or less). It becomes less efficient for larger percentages of *X's*. This chapter investigates two approaches for handling larger percentages of *X's* using an *X*-canceling MISR, which were published in [Putman 08a]. The first is based on deterministically observing scan cells, and the second is based on using a hybrid approach that combines *X*-masking with an *X*-canceling MISR.

The first approach is a new methodology for using the *X*-canceling MISR architecture which is based on deterministically observing scan cells. It is effective for larger percentages of *X's* in the output response and can provide greater amounts of

compression than probabilistic error detection. It can cancel out all *X's* and deterministically provide observation of any subset of non-*X* values. By having the automatic test pattern generator (ATPG) procedure record the subset of scan cells that must be observed to detect the necessary faults for a particular test pattern, the proposed method can then be used to deterministically observe those scan cells. By so doing, it can preserve the fault coverage of a test set in the presence of any distribution of *X's*.

The second approach described in this chapter combines *X*-masking with an *X*-canceling MISR. The benefit of this hybrid approach is that the *X*-masking hardware can target only the easy to mask *X's* and can let the rest of the *X's* go through to the *X*-canceling MISR to be canceled there. This added flexibility allows for much greater compression of the masking control data without loss of fault coverage. One particular masking technique is proposed which can exploit the added flexibility.

4.1. OVERVIEW OF X-CANCELING MISR

This section gives an overview of an *X*-canceling MISR and describes the new idea of how to use it for deterministic observation. Consider the output response that has been captured in the scan chains after applying a test vector. Let the value in each scan cell be represented by a symbol (as illustrated in Figure 4.1). Symbolic simulation can be performed to obtain the final state of the MISR in terms of the symbols after the output response has been shifted in to the MISR. Each bit of the MISR will be equal to a linear combination of the scan cells. This is shown in Figure 4.1 where, for example, the final value of the top bit of the MISR will be equal to $X_1 \oplus O_3 \oplus D_8 \oplus O_{13}$.

Symbolic simulation is accomplished by shifting bits from the scan chains into the MISR, which is simultaneously clocked, after all of the MISR registers (i.e., flip-flops) have been reset to the value zero. If the top register of the MISR in Figure 4.1 is M_1, and the bottom register is M_6, then after the first clock cycle, $M_1 = X_1 \oplus M_2 = X_1 \oplus 0 =$

X_1, $M_2 = O_2 \oplus M_3 \oplus M_1 = O_2$, and $M_3 = O_3 \oplus M_1 \oplus M_4 = X_2$. After the second clock cycle, $M_3 = O_9 \oplus M_1 \oplus M_4 = O_9 \oplus X_1 \oplus X_2$, $M_2 = D_8 \oplus M_1 \oplus M_3 = D_8 \oplus X_1 \oplus O_3$, and $M_1 = X_3 \oplus M_2 = X_3 \oplus O_2$. After the third, and final, clock cycle, which shifts the last bit of the scan chains into the MISR, $M_2 = O_{14} \oplus M_1 \oplus M_3 = O_{14} \oplus X_3 \oplus O_2 \oplus O_9 \oplus X_1 \oplus X_2$, and $M_1 = O_{13} \oplus M_2 = O_{13} \oplus D_8 \oplus X_1 \oplus O_3$. The rest of the MISR bits can be calculated in the same manner.

In Figure 4.1, assume each symbol X_i has an X value and each symbol D_i and O_i has a non-X value. Moreover, assume each symbol D_i corresponds to a scan cell that needs to be observed to ensure detection of the necessary faults for this particular test vector. In [Touba 07], only the X dependence was considered and all non-X values were observed probabilistically. In this work, the D dependence is also taken into consideration to ensure that the all D's are deterministically observed.

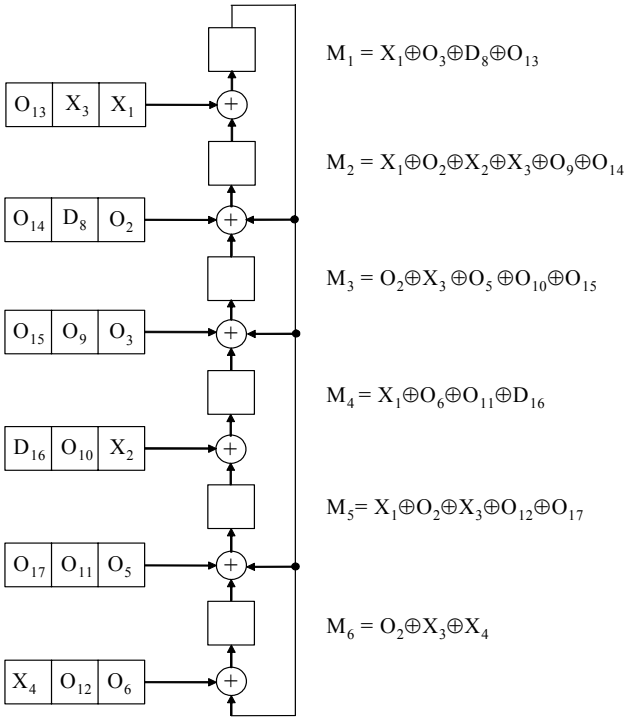

Figure 4.1: Example of Symbolic Simulation of MISR

Without loss of generality, assume all the O_i values in the output response are *0* so that each MISR bit is now simply equal to the linear combination of the *X* and *D* values. The *X* and *D* dependence of the MISR bits in this case are as shown in Figure 4.2. The linear equations for each MISR bit can be represented as a matrix where each row corresponds to a MISR bit and each column corresponds to an *X* or *D*. Each entry in the matrix is a 1 if the MISR bit corresponding to the row depends on the *X* or *D* corresponding to the column. This is illustrated in Figure 4.2. For example, in Figure 4.2, the second row of the matrix corresponds to M_2, and the 1's in the first three columns indicate dependence on X_1, X_2, and X_3, respectively.

$$
\begin{array}{l}
M_1 = X_1 \oplus D_8 \\
M_2 = X_1 \oplus X_2 \oplus X_3 \\
M_3 = X_3 \\
M_4 = X_1 \oplus D_{16} \\
M_5 = X_1 \oplus X_3 \\
M_6 = X_3 \oplus X_4
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\begin{array}{cccc:cc} X_1 & X_2 & X_3 & X_4 & D_8 & D_{16} \end{array} \\
\left[\begin{array}{cccc:cc}
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0
\end{array}\right]
\begin{array}{l} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{array}
\end{array}
$$

Figure 4.2: Linear Equations for MISR in Figure 4.1

$$
\begin{array}{c}
\begin{array}{cccc:cc} X_1 & X_2 & X_3 & X_4 & D_8 & D_{16} \end{array} \\
\left[\begin{array}{cccc:cc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]
\begin{array}{l} M_3 \oplus M_5 \\ M_2 \oplus M_5 \\ M_3 \\ M_3 \oplus M_6 \\ M_1 \oplus M_3 \oplus M_5 \\ M_3 \oplus M_4 \oplus M_5 \end{array}
\end{array}
$$

Figure 4.3: Gauss-Jordan Reduction of MISR Equations

Gauss-Jordan elimination [Cullen 97] can be performed on the matrix in Figure 4.2. Gauss-Jordan elimination involves performing rows operations that transform a set of columns into an identity matrix. Figure 4.3 shows the matrix in Figure 4.2 after Gauss-Jordan elimination has been performed. As was shown in [Touba 07], as long as the number of bits in the MISR is larger than the number of *X*'s compacted in the MISR, it is always possible to obtain rows after Gauss-Jordan elimination that have no dependence on the *X*'s. In Figure 4.3, it can be seen that the last two rows have no dependence on the *X*'s. Looking at the last row, for example, if MISR bits M3, M4, and M5 are XORed together, all the *X*'s cancel out and the resulting value will have no dependence on the *X*'s. This value can be compared with its fault-free value to detect errors in the non-*X* values that it depends on. Any combination of MISR bits can be XORed together using a programmable XOR as shown in Figure 4.4. Since each XOR combination of MISR bits will depend on roughly half of the non-*X* values, it was shown in [Touba 07] that checking q such combinations of MISR bits would give an error coverage approximately equal to $1-2^{-q}$. By simply checking a sufficient number of *X*-canceled combinations of MISR bits, a high probabilistic error coverage could be obtained. For example, by checking 7 *X*-canceled combinations, over 99% error coverage can be obtained. This works great when the percentage of *X*'s is around 1% or less. However, it becomes less efficient for larger percentages of *X*'s.

The idea in this chapter is that rather than checking a larger number of combinations to ensure a high probabilistic error coverage of all non-*X* values, a deterministic procedure could be used to ensure that the necessary to observe values (i.e., the *D*'s) are checked in a small number of combinations. Since checking each combination using the architecture in Figure 4.4 requires m bits be supplied by the tester where m is the number of bits in the MISR, if the number of combinations that need to be

checked can be reduced, then the data stored on the tester can be reduced resulting in greater test data compression.

To observe the *D*'s deterministically, it is necessary to include them in the dependency matrix as is done in Figure 4.2. Then when Gauss-Jordan elimination is performed, it not only processes the *X* columns, but also the *D* columns so that there is a single 1 in every row and column as shown in Figure 4.3. The number of *X*'s and *D*'s compacted should be limited so that it is always possible to obtain such a matrix after Gauss-Jordan elimination. Generally that means that each MISR signature can compact up to a total number of *X*'s plus *D*'s equal to *m*, the size of the MISR, so that the number of columns does not exceed the number of rows in the dependency matrix. After Gauss-Jordan elimination, the rows that depend only on the *D*'s (i.e., not on the *X*'s) are all XORed together to form one MISR bit combination that depends on all the *D*'s. In the example in Figure 4.3, this means that the last two rows are XORed together resulting in the MISR bit combination equal to:

$$(M_1 \oplus M_3 \oplus M_5) \oplus (M_3 \oplus M_4 \oplus M_5) = M_1 \oplus M_4$$

The MISR bit combination $M1 \oplus M4$ will not depend on any *X*'s, but will depend on all the *D*'s. This can be seen by looking back at Figure 4.1, and computing:

$$\begin{aligned} M1 \oplus M4 &= (X1 \oplus O3 \oplus D8 \oplus O13) \oplus (X1 \oplus O6 \oplus O11 \oplus D16) \\ &= O3 \oplus O6 \oplus D8 \oplus O11 \oplus O13 \oplus D16 \end{aligned}$$

As long as the number of *X*'s plus the number of *D*'s compacted by the MISR is less than or equal to *m*, and the Gauss-Jordan elimination produces a single 1 in every row and column, it is always possible to find a MISR bit combination that will depend on all the *D*'s.

Checking a MISR bit combination that depends on all the *D*'s still does not ensure that all errors in the *D*'s will be detected. For example, if an even number of *D*'s have errors, the errors will cancel out in the MISR bit combination and not be detected. So it is still necessary to check more than one combination. So then the question becomes what is the advantage of deterministically considering the *D*'s versus just using probabilistic error coverage as in [Touba 07]. When using just a single large MISR, there may not be much advantage to doing this. However, as will be shown in the next section, if the large MISR is replaced by multiple smaller MISRs, then deterministically considering the *D*'s can provide a significant advantage over probabilistic error coverage for equivalent aggregate MISR sizes. One specific scheme is described in Section 4.3 that uses a total of 16 MISRs each of size 16 bits with deterministic consideration of the *D*'s, and it is shown to provide almost a factor of 3 better compression in comparison to using a single MISR of size 256 with probabilistic error coverage.

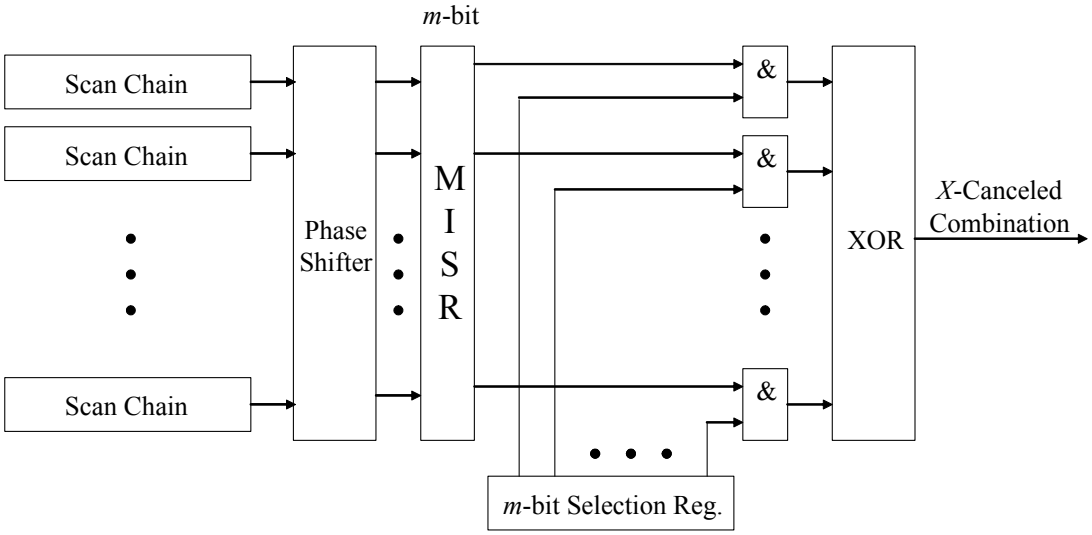

Figure 4.4: *X*-Canceling MISR

4.2 USING MULTIPLE MISRS

As was shown in the previous section, even with deterministic consideration of the *D*'s, it is not sufficient to observe only one MISR bit combination because an even number of errors may cancel out. Assuming all error combinations are equally likely, then the error coverage for the *D*'s would only be 50%. To reduce the probability of error canceling, it is necessary to observe a larger number of MISR bit combinations. However, if only a single large *m*-bit MISR is used, then checking each combination requires *m* bits to be stored on the tester. If two combinations are checked where each combination depends on 50% of the *D*'s, then the error coverage for the *D*'s would be 75% (the probability that there is an even number of errors in each combination is 0.5, so the probability that both have even errors is 0.25). So with a single *m*-bit MISR, obtaining 75% coverage of the *D*'s requires *2m* bits be stored on the tester. Now consider replacing the single *m*-bit MISR with two *m/2*-bit MISRs each of which compact half of the scan chains. For each of the two *m/2*-bit MISRs, a combination of MISR bits can be found using the procedure described in Section 4.1 such that it depends on all the *D*'s captured by that MISR. If half the *D*'s propagate to each MISR, then by checking one MISR bit combination for each of the *m/2*-bit MISRs, the same error coverage for the *D*'s (i.e., 75%) can be obtained as checking two MISR bit combinations of the *m*-bit MISR. However, checking each of the combinations for the *m/2*-bit MISRs requires only *m/2* bits, so in this case 75% coverage of the *D*'s is obtained with storing only *m* bits on the tester which is a factor of 2 improvement.

There are two drawbacks to using the two *m/2*-bit versus using one *m*-bit MISR:

1. The compaction must stop when either of the two MISRs captures a total number of *X*'s plus *D*'s equal to *m/2*. If the *X*'s and *D*'s are exactly evenly distributed between the two *m/2*-bit MISRs, then this would still happen at the same point as

when a single *m*-bit MISR would capture a total number of *X*'s plus *D*'s equal to *m*. However, in reality, it is unlikely that the *X*'s and *D*'s would be exactly evenly distributed, so in general one of the *m/2*-bit MISRs would fill up sooner than the other and thus the total number of *X*'s plus *D*'s compacted for each signature with two *m/2*-bit MISRs would be less than it would for a single *m*-bit MISR.

2. If the *D*'s are not evenly distributed between the two *m/2*-bit MISRs, then the error coverage will be lower than for a single *m*-bit MISR.

One approach to mitigate both of these drawbacks would be to use two pairs of *m/2*-bit MISRs (4 MISRs altogether) in the following way. Divide the scan chains into four evenly sized groups G_1, G_2, G_3, and G_4. Have one MISR capture from G_1 and G_2 and another MISR capture from G_3 and G_4. Then for the other pair of MISRs, have one capture from G_1 and G_3 and the other from G_2 and G_4. For each signature, there is now a choice of using one or the other pair of MISRs. If one pair of MISR is highly skewed in the distribution of *X*'s or *D*'s, then the other pair can be used. This helps to smooth out the distributions so that the performance of the *m*/2-bit MISRs in terms of error coverage and number of scan slices compacted per signature will not significantly lag that of an *m*-bit MISR while still enjoying a factor of 2 reduction in storage requirements on the tester. The cost of using two pairs of *m/2*-bit MISRs versus only a single pair is of course the additional overhead of adding more MISRs plus one extra bit needs to be stored on the tester per signature to dynamically select which pair of MISRs to use for each signature.

This approach of replacing an *m*-bit MISR with two pairs of *m/2*-bit MISRs can be used recursively. Each of the four *m/2*-bit MISR could be replaced by two pairs of *m/4*-bit MISRs. The total number of MISRs would now be 16. A single signature in the *m*-bit MISR is now replaced with signatures from 4 of the *m/4*-bit MISRs. This would give an error coverage for the *D*'s close to that of checking 4 combinations in the *m*-bit

MISR which is ($1-2^{-4} = 93.75\%$), and a reduction in tester storage of almost a factor of 4. Note that now 3 bits must be stored on the tester per signature to select which pairs of *m/4*-bit MISRs are used.

4.3 MULTIPLE MISR DESIGN EXAMPLE

Using this approach of splitting a larger MISR into multiple smaller MISRs as described in Section 4.2, one particular design example is described here.

Consider the design example shown in Figure 4.5. A single 64-bit MISR is replaced with 16 MISRs each of size 16. The 16-bit MISRs compact output response data scan slice by scan slice until a point is reached where compacting an additional slice would make it no longer possible to solve for all the *D*'s using signatures from 4 of the MISRs. At this point, 4 of the MISR signatures are processed. Gauss-Jordan reduction is performed on the linear equations for those MISRs as described in Section 4.1. In order to ensure high error coverage of the *D*'s, two linear combinations are checked for each MISR signature. Each of the two linear combinations per signature are formed by XORing together half of the rows in the Gauss-Jordan reduced matrix that depend only on *D*'s. This evenly divides the *D*'s between the two combinations. The end result is that 8 linear combinations are checked (two combinations for each of the 4 MISRs). Every *D* is included in exactly one of the linearly dependent combinations. The best case is if the *D*'s are evenly distributed among all 4 MISRs. In that case, the error coverage for the *D*'s where all error combinations are equally likely would be approximately $1-2^{-8} = 99.6\%$. All odd errors in the *D*'s would be guaranteed to be detected since at least one of the 8 combinations must have an odd number of errors. The error coverage for all other non-*X* bits would be $1-2^{-2}=75\%$. In general, the error coverage would be slightly lower due to variance in the distribution of *D*'s. Experiments indicate the average error coverage for the *D*'s is greater than 98% and as high as 99.5%.

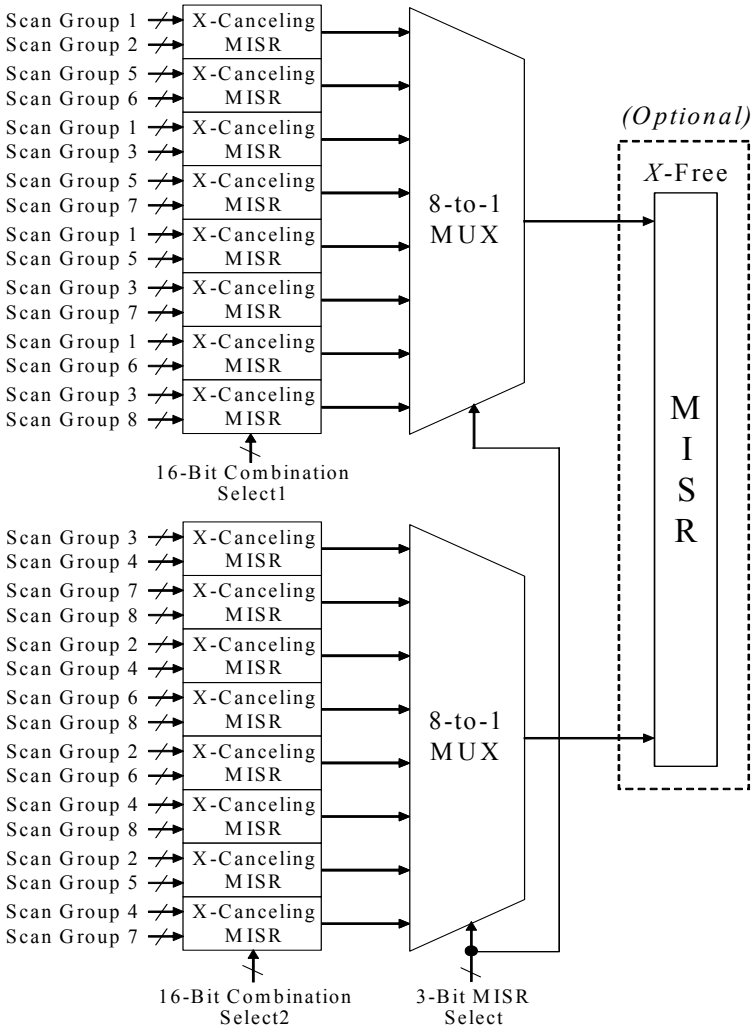

Figure 4.5: 16 *X*-Canceling MISRs Each of Size 16-Bits Compacting Scan Chains Divided Evenly into 8 Groups

The storage requirements for using a single 64-bit MISR checking 8 combinations would be (8*64) = 512 bits per signature, and each signature could compact a total of 64 *X*'s plus *D*'s. So the cost is 512/64=8 bits of storage per *X* and per *D*. Going to a 256-bit MISR would also still require 8 bits per *X* and per *D*. The probabilistic approach in [Touba 07] would require 8 bits of storage per *X* (it doesn't depend on *D*'s) for 99.6% error coverage, so it would actually be better. Now compare this to using the multiple

MISR design in Figure 4.5. For the multiple MISR design, each signature can compact 64 *X*'s plus *D*'s, and requires generating 2 linear combinations of four 16-bit MISRs where each linear combination requires 3-bits to select which MISR and 16-bits to select the combination. So the storage requirement per signature as shown in the figure is 4*(16+16+3) = 140 bits. The cost is 140/64 =2.19 bits of storage per X and per D. Consider a test set with 1% X's and 2% D's, in this case the method in [Touba 07] would require around 8*1%=8% whereas the proposed method would require around (2.19)(1%+2%) =6.57%. However, now consider a design with 3% X's and 2% D's, in this case the method in [Touba 07] would require around 8*3%=24% whereas the proposed method would require around (2.19)(3%+2%)=10.95%. So the bottom line is that for low percentages of *X*'s, the method in [Touba 07] is very efficient. However, for designs with larger percentages of *X*'s, the proposed multiple MISR method is much more efficient.

4.4 COMBINING X-MASKING AND X-CANCELING

To further improve the output compression achieved by an *X*-canceling MISR on designs with larger numbers of *X*'s, a hybrid approach that combines *X*-masking with an *X*-canceling MISR can be used. *X*-masking circuitry is added between the outputs of the scan chains and the inputs to the *X*-canceling MISR. The purpose of the *X*-masking circuitry, as well as the ensuing methodology, is to mask as many *X*'s as possible with the smallest amount of mask and control data, such that the overall amount of data stored on the ATE used to mask all of the *X*'s is less than it would have been if the *X*-canceling MISR was used by itself.

The key advantage of using *X*-masking plus *X*-canceling versus conventional *X*-masking only approaches that is exploited here is that the same mask can be reused for

many scan slices since it is not necessary to mask all *X*'s. This is taken advantage of by using the *X*-masking architecture proposed in Figure 4.6.

For *m* scan chains, the architecture in Figure 4.6 consists of a control signal, an *m*-bit masking register, an interval counter, and two logic gates per internal scan chain for a total of *2m* logic gates. The control signal is used to determine whether or not to apply the mask on a per scan slice basis. If the control signal is a '1', then the mask is applied. If the control signal is a '0', then the mask data itself is blocked and cannot affect the scan output response data. The mask register holds the mask data, which can be applied to the current scan slice. If the control signal is a '1', and a given mask data bit is also a '1', then the corresponding output response data bit is masked and forced to be a '1'. A '0' in the mask data means no masking will occur on the corresponding output response data bit even if the mask control signal is '1'.

The interval counter counts down the number of shift cycles (i.e., scan slices) the current mask can be applied to before a new mask is loaded. A constant value representing that number of scan slices is loaded into the interval counter's control logic one time at the beginning of the scan test. Every time the interval counter hits zero, including when it is reset at the beginning of the scan test, a new mask is loaded into the mask register, and the interval counter is loaded with its preprogrammed value. If there are *m* internal scan chains requiring *m* bits of mask data and *b* tester channels, then it will take *m/b* clock cycles to fully load the mask data at the beginning of each interval.

The goal of creating each mask is to make it applicable to as many adjacent scan slices as possible and mask as many *X*'s in those scan slices as possible, without masking out any of the *D*'s for which observation must be ensured. Also, it is desirable to minimize the number of non-*X* values that are masked as they may be useful for detecting unmodeled faults.

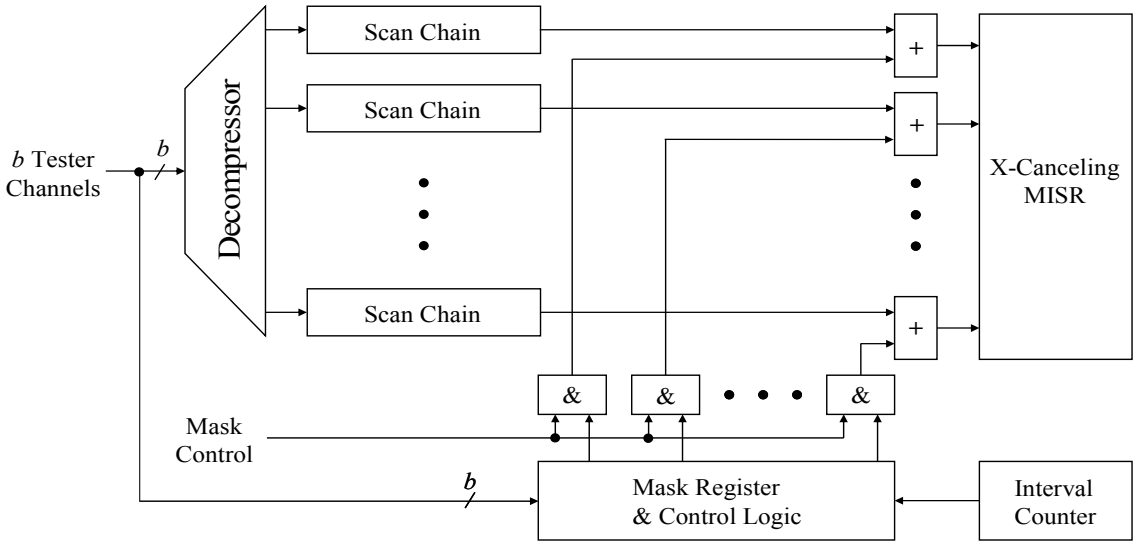

Figure 4.6: Proposed *X*-Masking Architecture for Use with *X*-Canceling MISR

The mask is created by first determining the locations of the *D* and *X* bits for each scan slice. Next, an interval of *n* number of scan slices is chosen to be processed for each new mask. Then an optimal mask is created for each interval by determining which scan slices should not be masked, as well as which scan slice bit position should be masked.

The algorithm for creating all of the masks for the entire pattern set, along with an example illustrated in Figure 4.7, is as follows:

1. Select the next interval of *n* scan slices to process. In the example in Figure 4.7, the interval contains five scan slices (shown as rows numbered 1 to 5).
2. All mask bits corresponding to scan chains in the current interval that contain *X*'s and no *D*'s are set to '1' indicating that those scan chains will be masked. In Figure 4.7, scan chain 6 is the only scan chain that this applies to, so it's mask bit is set to a '1'.
3. For each scan chain containing both *X*'s and *D*'s, the only way to mask the scan chain is if the mask control bit for the scan slices containing the *D*'s is set to 0. Setting the mask control bit for a scan slice to 0 prevents masking any

X's in that scan slice. A benefit function can be computed as the number of *X*'s that can be masked in the scan chain minus the number of *X*'s that cannot be masked for each of the scan slices containing *D*'s for which the mask control bit would need to be set to 0. For example, for scan chain 4, it has three *X*'s and one *D*. The one *D* resides in scan slice 2 which contains one *X*. So the benefit function for scan chain 4 would be 3-1=2. The scan chain with largest positive benefit function is selected first. It's mask bit is set to '1' and the mask control bit for each scan slice containing a *D* in that scan chain is set to '0'. This is repeated in a greedy fashion until no more scan chains exist whose benefit function is positive. In the example in Figure 4.7, there are two scan chains that have both *X*'s and *D*'s, namely scan chains 4 and 5. Their initial benefit functions are 2 and 0 respectively. So only scan chain 4 has its mask bit set to '1' and the mask control bit for scan slice 2 is set to '0' to prevent masking the *D* in scan chain 4.

4. When the greedy mask selection procedure in step 3 completes, then all the beneficial scan chain mask bits have been set to '1' and all the necessary mask control bits have been set to '0'. The procedure can now move to the next interval of *n* scan slices.

For the example output response in Figure 4.7, the algorithm only masks scan chains 4 and 6 and disables masking of scan slice 2. All but two of the *X*'s are masked. The resulting masked output response data (which is shown in Figure 4.7b) is fed into the *X*-canceling MISR.

Scan Slice	Mask Cntrl.	Scan Chains 0	1	2	3	4	5	6	7	8	9
1	1	s	d	s	s	**x**	d	**s**	s	s	s
2	0	s	s	s	s	d	s	x	s	s	s
3	1	s	s	s	s	**x**	d	**x**	s	s	s
4	1	s	d	s	s	**s**	x	**s**	s	s	s
5	1	s	d	s	s	**x**	s	**x**	s	s	s
	Mask	0	0	0	0	1	0	1	0	0	0

(a)

s	d	s	s	s	d	s	s	s	s
s	s	s	s	d	s	**x**	s	s	s
s	s	s	s	s	d	s	s	s	s
s	d	s	s	s	**x**	s	s	s	s
s	d	s	s	s	s	s	s	s	s

(b)

Figure 4.7: (a) Example of Mask Data for Output Response Interval; (b) Output Response after Masking

By being able to reuse a carefully created mask for multiple scan slices, without masking any D's and masking out a large percentage of X's from the output response data, the proposed X-masking approach, when combined with a downstream X-canceling MISR, can provide a significant improvement in output response data compaction, especially in the presence of a large number of unknowns.

4.5 EXPERIMENTAL RESULTS

Experiments were performed using the design in Figure 4.5 with 16 *X*-canceling MISRs each of size 16 for output streams with different percentages of *X*'s and *D*'s. The results are shown in Table 4.1. The first column shows the percentage of *X*'s in the output stream. The remaining columns show the various compressions achieved for different percentages of *D*'s corresponding to the different *X* percentages. Experimental results substantiate that the number of bits stored on the tester is approximately 2.19 bits

for each *X* or *D*, as described in Section 4.3. Thus, the amount of compression reduces as the number of *X*'s plus *D*'s increases.

To obtain higher compression, *X*-masking can be combined with *X*-canceling as described in Section 4.4. Experiments were performed on 3 industrial designs from Cirrus Logic. SynTest's ATPG tool was used to generate the tests and report a scan cell that each necessary fault propagated an error to. These scan cells were marked as *D*'s in the output response. Table 4.2 reports the results. The second and third column shows the percentage of *X*'s and *D*'s present in the corresponding designs. Experiments were performed for 3 different numbers of scan chains which are shown in the fourth column. The output response of all the three designs was compacted with and without using the *X*-masking before *X*-canceling, and the results are tabulated. As shown in the fifth column, a large percentage of *X*'s are masked from all the designs using the *X*-masking technique discussed in Section 4.4. The last three columns show the compression ratio achieved and the percentage improvement when using *X*-masking prior to *X*-canceling as compared to *X*-canceling alone. All the control and masking data needed to support the *X*-masking (as described in Section 4.4) is factored into the compression numbers.

Design *B* has the smallest number of *X*'s. As the number of *X*'s get smaller, the control data for *X*-masking starts to dominate the cost and this is seen from the minimum percentage improvement for Design *B* as compared to the other two designs. Furthermore, this dominance is exacerbated by the fact that although more *X*'s are able to be masked out for smaller numbers of scan chains, much more control data is required and hence there is an even smaller percentage improvement.

Design C has the largest number of *X*'s. Here, as the number of *X*'s increases, the ratio of the amount of control data to the number of *X*'s masked gets reduced since a larger number of *X*'s results in a larger number of easy to mask *X*'s. The overall result is

a larger percentage improvement when comparing the compression of the *X*-canceling alone versus using *X*-masking combined with *X*-canceling.

Table 4.1: Compression for Different Percentages of *X*'s and *D*'s

% of *X*'s	% of *D*'s			
	1%	2%	4%	6%
1%	21.3x	14.5x	9.0x	6.6x
3%	11.0x	9.0x	6.6x	5.3x
5%	7.6x	6.6x	5.3x	4.5x
8%	5.3x	4.8x	4.2x	3.7x
10%	4.5x	4.2x	3.7x	3.3x

Table 4.2: Results for Combining *X*-Masking and *X*-Canceling

Design	% *X*'s	% *D*'s	Scan Chains	Percent X's Masked using *X*-Masking	Compression *X*-Canceling Alone	Compression *X*-Masking and *X*-Canceling	Percentage Improvement
A	5.37%	0.89%	64 128 256	83% 81% 78%	5.9x 6.5x 6.6x	14.3x 15.7x 15.9x	142.4% 141.5% 140.9%
B	2.58%	0.63%	64 128 256	83% 75% 71%	9.1x 10.7x 11.2x	21.0x 22.3x 23.6x	130.8% 108.4% 110.7%
C	8.33%	0.75%	64 128 256	83% 78% 77%	4.4x 4.8x 5.0x	11.6x 12.2x 12.7x	163.6% 154.2% 154.0%

Chapter 5: Using Reiterative LFSR-Based X-Masking to Increase Output Compression in Presence of Unknowns

The previous chapter presented two approaches (an *X*-canceling MISR by itself and an *X*-canceling MISR combined with a very basic *X*-masking technique) for handling unknowns in the output response data to avoid corrupting any data needed for fault detection, and to avoid severely downgrading the performance of the output response compactor, and thus the overall test compression scheme. This chapter is focused more on improving *X*-masking techniques used to deal with *X*'s in the output response data, and presents two new approaches, published in [Putman 08b], for *X*-masking based on reiterative LFSR encoded masks, where both techniques take advantage of the data correlation in the output response data by reusing masks for multiple scan slices. The goals of both approaches are: 1) Mask all *X*'s at the input to the compactor. 2) Allow all specified bits, i.e., response bits that are required by the ATPG tool to be observed, to propagate unmasked through the masking logic so that all faults have the opportunity to be detected. 3) Remain ATPG, pattern, and compactor independent. 4) Maximize output compression.

The first approach, called Variable Reiterative *X*-Masking, identifies masks that can be used for varying numbers of scan slices, and then creates those masks with an LFSR to further reduce the amount of mask data needed to be stored on the ATE. The second approach described in this chapter combines fixed-interval reiterative LFSR *X*-masking with conventional LFSR *X*-masking. The fixed-interval technique is used to create masks that are then used for a fixed number of scan slices. This masks the majority of the *X*'s, but none of the specified bits required for fault detection. The idea behind this hybrid approach is for the reiterative *X*-masking logic to cheaply mask all of

the easy to mask *X*'s, and the conventional LFSR *X*-masking logic to handle the rest of the *X*'s that didn't get masked. The approach works particularly well as the numbers of *X*'s becomes large.

5.1 RELATED WORK

As previously mentioned, a high level of test data compression, specifically output data compression, can be achieved in the presence of all known values. Once unknown values (i.e., *X*'s) are introduced, then the amount of output data compression is reduced, since the output compactors have to deal with these *X*'s in order to enable the detection of all detectable faults. The amount of output compression is given by the following formula:

$$Comp.\ Ratio = \frac{Total\ Scan\ Output\ Bits\ from\ the\ Scan\ Chains}{ATE\ Response\ Data + Mask\ Data + Control\ Data}$$

In general, the output response data is highly compressible, but the amount of compression depends on the percentage of different types of bits that the data contains. The output response data from a typical scan test pattern contains three types of bits: *D*, *S*, and *X*. The first type of bits, D bits, are also sometimes referred to as specified bits. Each one of these bits contains a value that is required to be observed for the detection of one or more faults. In a typical scan test pattern set, the average number of *D* bits, over all of the patterns, is less than 5%. The second type of bits, *S* bits, contains known values, either a logic 1 or a logic 0; however, they are not required for the detection of modeled faults. The third type, *X* bits, or unknowns, can vary in percentage based on the given design of the CUT.

Classic *X*-masking, such as described in [Chickermane 04], handles *X*'s by reusing masks for a large number of scan slices, which when combined with one or more bits of control data per scan slice can selectively mask chosen scan chains on a per scan

slice basis. The advantage to this type of technique is that the amount of mask data is keep to a minimum; however the downside is that a number of non-*X* values get masked as well. This means that new patterns have to be created to compensate for the undetected faults, thus increasing pattern count and the amount of stimulus and response data stored on the ATE, which in turn reduces the amount of output data compression as indicated by the formula above.

Conventional state-of-the-art LFSR *X*-masking, similar to that described in [Naruse 03], creates a mask for every scan slice to ensure a very high mask resolution, such that all *X*'s in each scan slice get masked, and all *D*'s remain unmasked. In order for this technique to work, the mask bit corresponding to each *X* needs to be set to '1,' indicating that this bit is to be masked, and the mask bit corresponding to each *D* needs to be set to '0,' indicating that this bit should not be masked. An example hardware implementation of this technique is shown in Figure 5.1. In this implementation, a MISR is used as the compactor, and an LFSR is used as the decompressor, which not only drives the scan chains, but also creates the mask data via extra outputs from the phase shifter. Since the typical output scan slice does not contain a high percentage of *D*'s or *X*'s, then the amount of compressed mask data stored on the ATE, which is fed to the LFSR in order to create the entire mask, may be small. The amount of compressed mask data stored on the ATE for each scan slice is on the order of $D + X$ bits. While using this technique does help to lower the amount of mask data, the ATE still has to store enough mask data to create a mask for every slice, and, thus, the amount of output compression is reduced.

5.2 VARIABLE REITERATIVE X-MASKING

The first approach presented in this chapter, Variable Reiterative LFSR *X*-Masking, uses the idea of applying a mask that is applicable for a number of scan slices,

taken from classic *X*-masking, with the added constraint that none of the *D*'s can be masked. Then, in order to keep the cost of storing the mask data for a given mask small, a reseedable LFSR is used to encode just the *X* and *D* bits (similar to what is done for conventional LFSR *X*-masking). This technique works by finding a mask that is usable for a varying number of adjacent scan slices such that all of the *X*'s are masked, and all of the *D*'s are left unblocked and allowed to propagate to the inputs of the compactor.

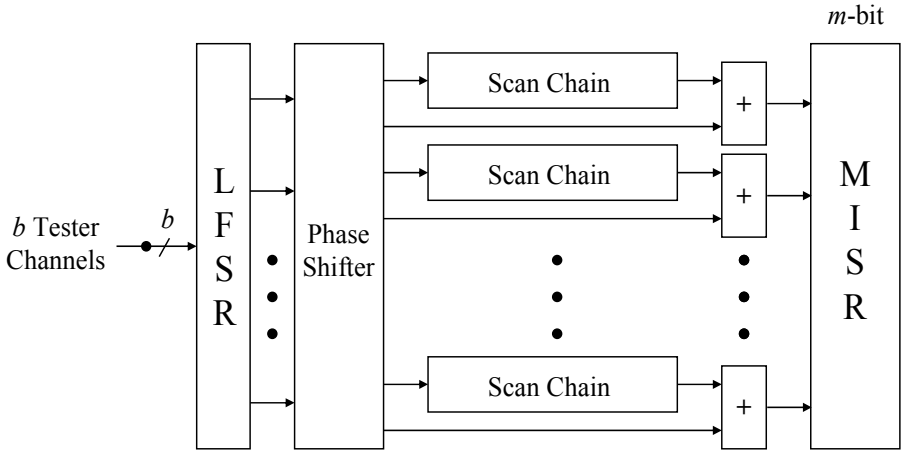

Figure 5.1: Conventional LFSR X-Masking Architecture

One hardware implementation for this approach is shown in Figure 5.2. Here, for *m* scan chains, the reiterative *X*-masking logic consists of an *m*-bit masking register, an *i*-bit interval counter, and *m* OR gates. The existing decompressor, which is used to generate specified bits for the scan chains, also generates the mask and interval bits to be loaded into the mask register every *i* clock cycles. In order to maintain the correct data in the scan chains, the scan chains are not clocked while the mask register is being loaded.

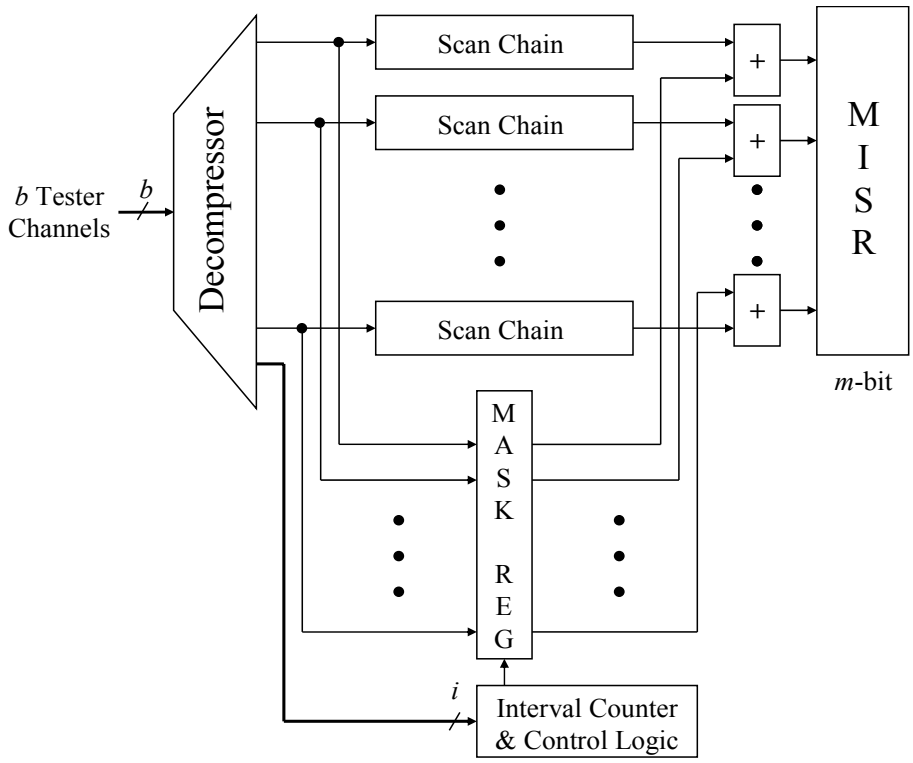

Figure 5.2: Variable Reiterative LFSR X-Masking Architecture

Once the mask register is loaded, it then contains the information about which bits are masked and which bits are not masked. The interval counter contains the number of scan slices that the current mask is applied to. While the interval counter is counting down, the mask register holds its state. Once the interval counter reaches zero, then a new mask loaded into the mask register, and a new interval is loaded into the interval counter.

The overall goal of creating each mask is to mask all of the *X*'s in as many adjacent scan slices as possible without masking out any of the *D*'s for which observation must be ensured. The algorithm for creating all of the masks for a given pattern set is as follows:

1. First determine the location of the *D* and *X* bits for each scan slice.

2. Start with the first scan slice at the beginning of the current interval, and create a mask for it by first setting each mask bit to '1' for all bit positions of the scan slice that are *X*'s, indicating that these positions will be masked. Next, set each mask bit to '0' for all bit positions of the scan slice with *D*'s (i.e., specified bits), indicating that these positions will not be masked. In the example shown in Figure 5.3a, for scan slice 1, bit position 6 has an *X*, so the mask bit for that position would be '1'. Bit positions 1 and 2 are *D*'s, so the mask bits for those positions would be '0's.
3. Increase the interval by one and add to the current mask by setting the mask bits as described in step 1, if necessary. In Figure 5.3a, for scan slice 2, bit position 5 has a *D* in it, so the mask bit for that position is set to '0'. Bit position 4 is set to '1'.
4. Repeat step 3 until at least one mask bit position has a conflict. Once that happens, set the end of the current interval to the previous scan slice, and make the current scan slice the beginning of a new interval. Then proceed to step 2. For the example in Figure 5.3a, when adding scan slice 5 to the interval containing scan slices 1-4, an *X* shows up in bit positions 2 and 5, indicating that those positions need to be masked; however, in the previous scan slices, those bit positions were set to '0' (i.e., "not masked"), so there is a conflict. For that reason, scan slice 4 is set to be the end of the current interval (the resulting mask is shown as M1), and scan slice 5 is set to be the first scan slice in the next interval. The resulting mask representing the second interval (scan slices 5-7) is shown as M2. The '?' characters in masks M1 and M2 represent bits whose value will be determined by the LFSR once the final masks are

encoded. Figure 5.3b shows the resulting output response data after all masking has been performed.

As demonstrated in this example, by being able to reuse masks for multiple adjacent scan slices and encoding the mask data, the amount of mask data needed to be stored on the ATE can be reduced significantly, thus improving the amount of output data compression.

Scan Slice	Scan Chains									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
7	s	d	s	s	s	**x**	s	s	s	s
6	s	s	s	d	d	s	d	s	**x**	s
5	s	d	**x**	s	s	**x**	s	s	s	s
4	s	s	s	d	s	d	s	s	s	s
3	s	s	d	s	s	s	**x**	s	s	s
2	s	s	s	s	**x**	d	s	s	s	s
1	s	d	d	s	s	s	**x**	s	s	s
M 2	?	0	1	0	0	1	0	?	1	?
M 1	?	0	0	0	1	0	1	?	0	?

(a)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
7	s	d	s	s	s	s	s	s	s	s
6	s	s	s	d	d	s	d	s	s	s
5	s	d	s	s	s	s	s	s	s	s
4	s	s	s	d	s	d	s	s	s	s
3	s	s	d	s	s	s	s	s	s	s
2	s	s	s	s	s	d	s	s	s	s
1	s	d	d	s	s	s	s	s	s	s

(b)

Figure 5.3: Variable Reiterative LFSR X-Masking Example

5.3 HYBRID X-MASKING APPROACH

Another approach that improves upon the output compression achieved by using conventional LFSR X-masking alone, especially for designs with larger numbers of *X*'s, is a hybrid approach that combines fixed-interval reiterative *X*-masking with conventional

LFSR *X*-masking. For this approach, the fixed-interval reiterative *X*-masking logic is added between the output of the scan chains and the input of the conventional LFSR *X*-masking logic. The purpose of the reiterative *X*-masking hardware, as well as the ensuing methodology, is to mask as many *X*'s as possible with the smallest amount of mask and control data, such that the total amount of data stored on the tester (ATE) is less than the amount that would have been stored if conventional LFSR *X*-masking were used alone.

The primary advantage of this type of hybrid approach is that the reiterative *X*-masking logic can be used to mask all of the easy to mask *X*'s by reusing masks for many scan slices, since not all *X*'s have to be masked. The one implementation of this approach is shown in Figure 5.4.

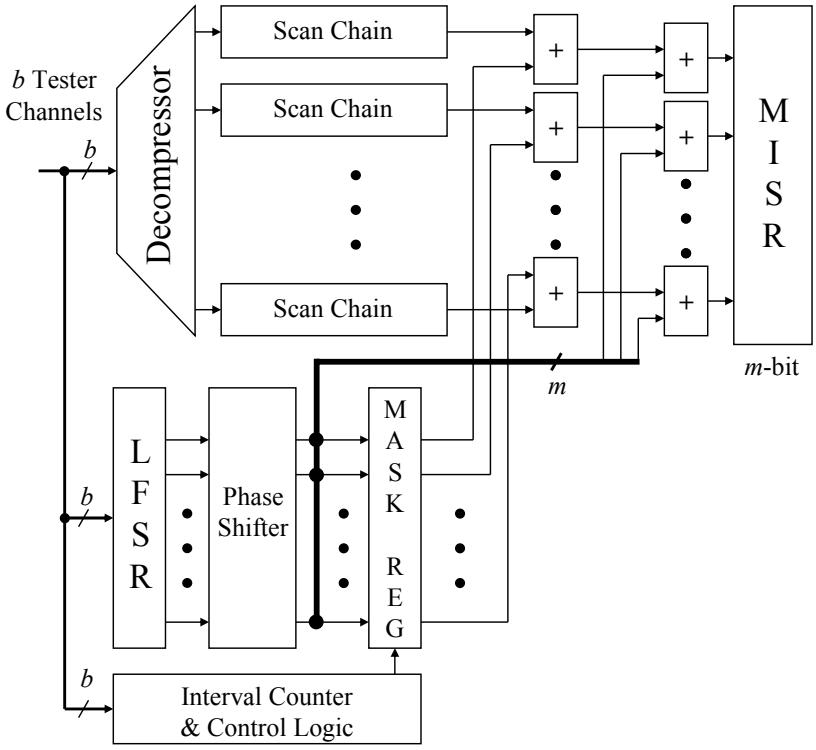

Figure 5.4: Hybrid X-Masking Architecture

For the purposes of this chapter, the LFSR and phase shifter used to generate the mask bits are separate from the decompressor used to drive the scan chains. Here, for m scan chains, the hardware implementation of this approach consists of an m-bit masking register, an i-bit interval counter, $2m$ OR gates, and an LFSR sized to be s_{max}+20, where $s_{max} \leq m$. This LFSR is used primarily to generate the mask for each slice necessary for the conventional LFSR X-masking part of this approach, but is also used to generate the mask bits to be loaded into the mask register for reiterative X-masking part of the approach. As demonstrated in [Könemann 91], in order to generate the specified bits (or in this case the required 'do' and 'do not' mask bits) with a high degree of probability, the length of the LFSR should be at least s_{max}+20. Since the reiterative X-masking part of this hybrid approach uses a fixed interval, the interval counter is loaded one time at the beginning of the test straight from the b tester channels and should take no more than i/b clock cycles. Every i clock cycles, the LFSR generates a reiterative mask that gets loaded into the mask register. The loading of the mask register is controlled by the interval counter. Once the interval counter reaches zero, then a new mask is loaded into the mask register, and the interval counter is returned to its preprogrammed value.

Each mask for the reiterative X-Masking part of this approach is created by first determining the locations of the D and X bits for each scan slice. Then, for each interval of i scan slices, an optimal mask is created determining which bit positions of the scan slices should and should not be masked. The algorithm for creating all of the reiterative masks for a given pattern set is as follows:

1. Select the first interval of i scan slices to process. The example in Figure 5.5a shows five scan slices (numbered 1 to 5).
2. All mask bits corresponding to scan chains in the current interval that contain one or more D's (regardless of X's present in those scan chains) are set to '0,'

indicating that those bit positions (i.e., scan chains) will not be masked for the current interval to help ensure that all faults are detected. In the example shown in Figure 5.5a, the mask bits for scan bit positions 1, 4, and 5 are set to '0.'

3. All mask bits corresponding to scan chains in the current interval that contain only *X*'s are set to '1,' indicating that those bit positions will be masked. In the example shown in Figure 5.5a, the mask bits for scan bit positions 2, 3, and 6 are set to '1.'
4. After all of the beneficial scan chain mask bits have been set to '1' and all the necessary mask bits for fault detection have been set to '0,' then the procedure can move to the next interval of *i* scan chains. Figure 5.5a shows the resulting mask for scan slices 1 to 5. The '?' characters in the mask represent bits whose value will be determined by the LFSR once the final masks are encoded. Figure 5.5b shows the resulting output response data for the example in Figure 5.5a after fixed-interval reiterative *X*-masking has been applied.

The size of the interval determines the overall effectiveness of this hybrid approach. As the interval gets smaller, the number of *X*'s masked increases, but the amount of mask data also increases as well. As the interval gets larger, the opposite occurs. The optimal interval is somewhere in the middle and is somewhat dependent on the CUT and the pattern set generated for it. Since the interval is completely programmable, and only loaded one-time at the beginning of the pattern set, the interval can be re-determined anytime the pattern set changes. For the purpose of determining the optimal interval for a given pattern set, a cost function was employed that calculates the cost of using a given interval on a given pattern set and adds it to a solid estimate of the additional cost of the mask data needed for the downstream conventional LFSR *X*-

masking logic to mask all of the leftover *X*'s that were not masked by the reiterative *X*-masking logic. The equations for calculating the total cost for a given interval are:

Reiterative X-Masking Cost = Total Mask Data +Interval Counter Size

Total Cost = Reiterative X-Masking Cost + Conventional X-masking on Leftover X's

Then, once the lowest cost interval (i.e., the optimal interval) is determined through a number of simulations, the final simulation can be performed that determines the improvement of the hybrid approach over the conventional LFSR *X*-masking approach by itself.

Scan Slice	Scan Chains									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
5	s	d	s	**x**	s	s	**x**	s	s	s
4	s	**x**	s	s	s	d	s	s	s	s
3	s	s	x	s	d	d	s	s	s	s
2	s	d	s	s	s	s	**x**	s	s	s
1	s	d	s	s	s	**x**	s	s	s	s
Mask	?	0	1	1	0	0	1	?	?	?

(a)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
5	s	d	s	s	s	s	s	s	s	s
4	s	**x**	s	s	s	d	s	s	s	s
3	s	s	s	s	d	d	s	s	s	s
2	s	d	s	s	s	s	s	s	s	s
1	s	d	s	s	s	**x**	s	s	s	s

(b)

Figure 5.5: Fixed-Interval Reiterative X-Masking Example

5.4 EXPERIMENTAL RESULTS

Experiments were performed on three industrial designs using the *X*-masking schemes shown in Figures 5.2 and 5.4. SynTest's ATPG tool was used to generate the patterns and report one scan cell for each observable fault. These scan cells were marked as *D*'s in the output response, which was then processed by each approach's algorithm. Table 5.1 reports the results.

The first column in Table 1 designates the design for each row of results. NOTE: Designs A2 and B2 are modified versions of designs A and B with X-generators added randomly. These were added to see what the effect of large numbers of *X*'s would be on the two proposed approaches. The second column shows the percentage of *X*'s present in each design. Results were collected for three different numbers of scan chains, shown in the third column, for each design. The fourth column shows the compression ratio for the output response data for conventional LFSR *X*-masking by itself. The fifth column shows the compression ratio results for the Variable Reiterative LFSR *X*-masking approach by itself. The seventh column reports the results for the hybrid *X*-masking approach, which combines the fixed-interval reiterative LFSR *X*-masking approach with conventional LFSR *X*-masking. The sixth column reports the amount of *X*'s masked for the fixed-interval reiterative part of the hybrid *X*-masking approach, which, in general, remained in the 80% range for the optimal interval. In the few cases where the number of *X*'s masked by the reiterative part of the hybrid approach dropped below 80%, it was necessary to increase the size of the interval in order to reduce the amount of mask data so that the overall amount of output compression for the combined approach was maximized.

Table 5.1: Results for the Different X-Masking Techniques

Design	Percent X's	Scan Chains	Compression Conventional X-Masking Alone	Compression Variable Reiterative Alone	Fixed Interval Reiterative	
					Percent X's Masked	Compression w/ Conventional
A	0.7%	64	66.3x	185.9x	83.9%	86.1x
		128	66.3x	183.4x	84.6%	85.4x
		256	66.5x	185.5x	77.2%	88.1x
B	1.2%	64	49.0x	98.3x	86.4%	69.5x
		128	49.0x	101.3x	85.5%	69.1x
		256	48.9x	98.7x	69.3%	68.8x
C	2.5%	64	31.2x	52.6x	79.9%	55.0x
		128	31.4x	52.2	75.4%	54.9x
		256	31.7x	51.5x	75.1%	58.4x
B2	5.4%	64	16.0x	29.1x	85.6%	37.9x
		128	16.0x	28.3x	85.8%	39.0x
		256	16.1x	26.6x	83.9%	38.0x
A2	8.3%	64	11.0x	19.7x	84.6%	32.9x
		128	11.1x	18.7x	85.1%	33.0x
		256	11.1x	17.1x	84.5%	32.4x

For all three techniques, including the conventional LFSR *X*-masking technique by itself, the amount of compression increased as the number of *X*'s decreased. The results for the Variable Reiterative *X*-masking technique show an improvement over the conventional technique across all numbers of *X*'s ranging from a 79% improvement for 8% *X*'s to 180% improvement for .7% *X*'s. The hybrid approach also shows an improvement over all ranges of *X*'s ranging from a 200% improvement for 8% *X*'s to a 30% improvement for .7% *X*'s. As evidenced by the results, the Variable Reiterative LFSR *X*-masking approach works especially well for smaller numbers of *X*'s, and the hybrid approach works best with larger numbers of *X*'s.

Chapter 6: Conclusions and Future Research

6.1 CONCLUSIONS

The continual miniaturization and advancement of technology has led to the need for new fault models, such as the delay fault models, and ways to efficiently test for those fault models to ensure that all of the defects in the product can be targeted and tested efficiently and cost effectively. In Chapter 2 a new technique was presented for testing for small delay defects that uses standard transition delay ATPG along with timing information gathered from standard static timing analysis of a given design. It was shown that by grouping patterns that test paths with similar slack values and adjusting the timing of those patterns, all the small delay defects could be caught that are relative to the size of the user-specified bin width. Additionally, it was shown on industrial designs that the test coverage and pattern count for the proposed technique were similar to that of regular transition delay ATPG on the same fault set and that, to a point, the pattern count doesn't increase unreasonably as the number of bins is increased. It was also shown that by using this flow and technique other paths besides those within a given clock domain can be easily targeted and tested, and that if the faults are grouped (e.g., by clock domain), then a greater visibility into the test coverage of that group can be achieved.

Along with the new fault models for the smaller technologies comes an increase in design complexity and integration (i.e., higher gate count). This increased gate count along with the need for additional patterns to test the new fault models has led to a very large increase in test data volume. In order to reduce manufacturing costs, new test compression techniques are needed to cope with this increase in test data. In Chapter 3, a test compression technique is presented that demonstrates that by detecting faults with a higher than normal expansion ratio, and then using progressively smaller ones, the

amount of compression can be significantly improved when compared with conventional approaches that use a single expansion ratio with one or more configurations. Furthermore, it also shows that by exploiting the flexibility of choosing how the scan chains are concatenated by using a dependency analysis procedure that takes into account structural dependencies among the scan chains as well as free-variable dependencies in the logic driving the scan chains, the detection of faults is significantly improved, as is the overall test compression.

While improving the amount of test compression from the input (decompressor) side is extremely important, that is only half of the equation when it comes to increasing the overall amount of test compression that can be achieved by a complete test compression scheme. The other half of the equation is the test compression performance of the output response compactor in the presence of *X*'s, which can come from a number of sources including delay fault testing patterns which incorporate timing information as part of their ATPG algorithms. Chapter 4 showed how an *X*-canceling MISR with deterministic observation can be used to achieve greater compression for higher percentages of *X*'s than using an *X*-canceling MISR with probabilistic observation. It was shown that by dividing a large *X*-canceling MISR into multiple smaller *X*-canceling MISRs, the number of bits required to process each signature could be reduced while still observing the necessary to observe bits (i.e., the *D*'s). It was also shown that a hybrid approach combining *X*-masking with an *X*-canceling MISR can significantly increase the amount of compression even further. Large numbers of *X*'s can be masked at low cost by focusing only on the easy to mask *X*'s (where the mask data can be reused across many scan slices) and leaving the rest of the *X*'s to be handled by the *X*-canceling MISR.

X-masking can be a powerful tool used to handle *X*'s (unknowns) in the output response data. A given *X*-masking technique can be used by itself to mask or block all of

the *X*'s in the output response data before that data is processed by an output response compactor such as a MISR, or it can be combined with other techniques. Chapter 5 improved upon the *X*-masking first explored in Chapter 4 and demonstrated that using reiterative LFSR *X*-masking, where masks are reused for multiple scan slices, can provide significant increases of output compression over using conventional LFSR *X*-masking alone. It was also shown that using Variable Reiterative LFSR *X*-masking to mask all of the *X*'s and none of the *D*'s can significantly increase the amount of compression for smaller numbers of *X*'s. Likewise, it was shown that the hybrid approach, combining the fixed-interval reiterative LFSR *X*-masking approach with conventional LFSR *X*-masking, significantly increased the amount of compression for larger numbers of *X*'s. This resulted from the fact that larger numbers of *X*'s can be masked at a minimum cost, since the technique is focused on reusing a mask for many slices, masking the easy to mask *X*'s, while allowing the remaining *X*'s to be masked by the downstream conventional LFSR *X*-masking circuitry.

6.2 FUTURE RESEARCH

Current delay fault testing methodologies, while being thoroughly researched, still have many short comings. Even with new timing-aware algorithms such as the one presented here, small delay defects are still difficult to test due to timing hazards, clock speed resolution issues, and the general inability to sensitize the critical paths. All of these problem areas would be good topics for future research. Delay fault testing may even require a complete rethinking of the delay fault models themselves, the testing methodology, and the testability structures to enable the proper testing for timing related defects.

For test compression there can still be research and improvements done in the ATPG algorithm area. The use of test compression methodologies is very commonplace

now, and as such, the ATPG algorithm should be designed in such a way that the test compression and its controls are an integral part of the ATPG algorithms, as opposed to the opposite extreme of just post processing and compressing the patterns after they have been generated. While some work has already been done in this area, there is still a need for even more sophisticated algorithms, especially ones that are timing-aware that can be used for delay fault testing.

Along with the need for more research in the area of test compression comes the need for more research on how to handle unknowns in the output response data since *X*'s can degrade not only the performance of the output response compactor but also the overall performance of the complete test compression scheme. Specific topics of future research, especially for designs with larger percentages of *X*'s, could include analyzing scan patterns and implementing hardware to take advantage of data correlation between patterns (i.e., scan loads) and not just between adjacent scan slices. Another area of research could involve further investigating the use of multiple MISRs for *X*-canceling to minimize both the control data and the probability of error canceling while also minimizing area overhead.

Bibliography

[Barnhart 01] Barnhart, C., V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann**,** "OPMISR: the Foundation for Compressed ATPG Vectors," *Proc. of International Test Conference* , pp. 748-757, 2001.

[Bayraktaroglu 03] Bayraktaroglu, I. and A. Orailoglu, "Concurrent Application of Compaction and Compression for Test Time and Data Volume Reduction in Scan Designs," IEEE Trans. on Computers, Vol. 52, No. 11, pp. 1480-1489, Nov. 2003.

[Chao 05] Chao, M. C.-T., S. Wang, S.T. Chakradhar, and K.-T. Cheng, "Response Shaper: A Novel Technique to Enhance Unknown Tolerance for Output Response Compaction," *Proc. of International Conference on Computer-Aided Design*, pp. 80-87, 2005.

[Chickermane 04] Chickermane, V., B. Foutz, and B. Keller, "Channel Masking Synthesis for Efficient On-Chip Test Compression," *Proc. of International Test Conference*, pp. 452-461, 2004.

[Crouch 00] Crouch, A. Design-For-Test For Digital IC's and Embedded Core Systems. New Jersey: Prentice Hall PTR, 2000.

[Crouch 03] Crouch, A., J. Potter and J. Doege, "AC Scan Path Selection for Physical Debugging," IEEE Design & Test of Computers, pp.34-40, Sept-Oct 2003.

[Cullen 97] Cullen, C.G., Linear Algebra with Applications, Addison-Wesley, ISBN 0-673-99386-8, 1997.

[Datta 04] Datta, R., R. Gupta, A. Sebastine and J.A. Abraham and M. d'Abreu, "Tri-Scan: A Novel DFT Technique for CMOS Path Delay Fault Testing," *Proc. Int. Test Conf* , pp. 1118-1127, Oct. 2004.

[Dutta 06] Dutta, A. and N.A. Touba, "Using Limited Dependence Sequential Expansion for Decompressing Test Vectors," *Proc. Int. Test Conf,* 2006.

[Gupta 04] Gupta, P., and M. Hsiao, "ALAPTF: A New Transition Fault Model and the ATPG Algorithm," *Proc. Int. Test Conf*, pp. 1053-1060, Oct. 2004.

[Hamzaoglu 99] Hamzaoglu, I. and J.H.Patel, "Reducing Test Application Time for Full Scan Embedded Cores," *Proc. Int. Symp. on Fault-Tolerant Computing*, pp. 260-267, 1999.

[Heragu 96] Heragu, K., J. Patel, and V. Agrawal, "Segment Delay Faults: A New Fault Model," *IEEE VLSI Test Symposium*, 1996.

[Keller 01] Keller, B., et al, "OPMISR: The Foundation for Compressed ATPG Vectors," *Proc. of IEEE International Test Conference,* pp. 748-757, 2001.

[Könemann 91] Könemann, B., "LFSR-coded Test Patterns for Scan Designs," *Proc. European Test Conf.*, pp.237-242, 1991.

[Könemann 01] Könemann, B., C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," *Proc. Asian Test Symp.*, pp. 325-330, 2001.

[Krishna 02] Krishna, C.V., and N.A. Touba, "Reducing Test Data Volume Using LRSR Reseeding with Seed Compression," *Proc. of IEEE International Test Conference,* pp. 321-330, 2002.

[Kruseman 04] Kruseman, B., A. Mahji, G. Gronthoud and S. Eichenberger, "On Hazard-Free Patterns for Fine-Delay Fault Testing," *Proc. of IEEE International Test Conference*, pp. 213-222, Oct. 2004.

[Li 89] Li, W., et al., "On Path Selection in Combinational Logic Circuits," IEEE Trans. on CAD, vol. 8, pp. 56-63, Jan. 1989.

[Lin 87] Lin, C., and S. Reddy, "On Delay Fault Testing in Logic Circuits," IEEE Trans, on CAD, vol. 6,pp. 694-703, Sept. 1987.

[Liou 00] Liou, J., K. Cheng and D. Mukherjee, "Path Selection for Delay Testing of Deep Sub-micron Devices Using Statistical Performance Sensitivity Analysis," *Proc. of IEEE VLSI Test Symposium*, 2000.

[Madge 03] Madge, R., B. Benware and W.R. Daasch, "Obtaining High Defect Coverage for Frequency Dependent Defects in Complex ASICs," IEEE Design & Test of Computers, pp. 46-53, Sept.-Oct. 2003.

[Mitra 02] Mitra, S., and K.S. Kim, "X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction," *Proc. of IEEE International Test Conference,* pp. 311-320, 2002.

[Mitra 04a] Mitra, S., and K.S. Kim, "X-Compact: An Efficient Response Compaction Scheme," IEEE Trans. on Computer-Aided Design, Vol. 23, No. 3, pp. 421-432, Mar. 2004.

[Mitra 04b] Mitra, S., S.S. Lumetta, and M. Mitzenmacher, “X-Tolerant Signature Analysis,” *Proc. of International Test Conference*, pp. 432-441, 2004.

[Mitra 06] Mitra, S. and K.S. Kim, “XPAND: An Efficient Test Stimulus Compression Technique,” IEEE Trans. on Computers, Vol. 55, No. 2, pp. 163-173, Feb. 2006.

[Naruse 03] Naruse, M., I. Pomeranz, S.M. Reddy, and S. Kundu, “On-Chip Compression of Output Responses with Unknown Values Using LFSR Reseeding,” *Proc. of International Test Conference*, pp. 1060-1068, 2003.

[Pandey 02] Pandey, A.R. and J.H. Patel, "Reconfiguration Technique for Reducing Test Time and Test Volume in Illinois Scan Architecture Based Designs," *Proc. VLSI Test Symposium*, pp. 9-15, 2002.

[Patel 03] Patel, J.H., S.S. Lumetta, and S.M. Reddy, “Application of Saluja-Karpovsky Compactors to Test Responses with Many Unknowns,” *Proc. of VLSI Test Symposium*, pp. 107-112, 2003.

[Pomeranz 98] Pomeranz, I., and S. Reddy, “Design-for-Testability for Path Delay Fauilts in Large Combinational Circuits Using Test Points,” IEEE Trans. on CAD, vol. 17, pp. 333-343, Apr. 1998.

[Pomeranz 02] Pomeranz, I., S. Kundu, and S.M. Reddy, “On Output Response Compression in the Presense of Unknown Output Values,” *Proc. of Design Automation Conference*, pp. 255-258, 2002.

[Putman 06] Putman, R. and R. Gawde, "Enhanced Timing-Based Transition Delay Testing for Small Delay Defects," *Proc. VLSI Test Symposium*, pp. 336-342, 2006.

[Putman 07] Putman, R. and N.A. Touba, "Using Multiple Expansion Ratios and Dependency Analysis to Improve Test Compression," *Proc. VLSI Test Symposium*, pp. 211-216, 2007.

[Putman 08a] Putman, R., R. Garg, and N.A. Touba, "Increasing Output Compaction in Presence of Unknowns using an X-Canceling MISR with Deterministic Observation," *Proc. VLSI Test Symposium*, 2008.

[Putman 08b] Putman, R, "Using Reiterative LFSR Based X-Masking to Increase Output Compression in Presence of Unknowns," *Proc. Great Lakes Symposium on VLSI*, 2008.

[Rajski 00] Rajski, J., N. Tamarapalli, and J. Tyszer, “Automated Synthesis of Phase Shifters for Built-In Self-Test Applications,” IEEE Trans. on Computer-Aided Design, Vol. 19, No. 10, pp. 1175-1188, Oct. 2000.

[Rajski 03] Rajski, J., C. Wang, J. Tyszer, and S.M. Reddy, "Convolutional Compaction of Test Responses," *Proc. of IEEE International Test Conference,* pp. 745-754, 2003.

[Rajski 04] Rajski, J., J. Tyszer, M. Kassab, and N. Mukherje, "Embedded Deterministic Test," IEEE Trans. on Computer-Aided Design, Vol. 23, No. 5, pp. 776-792, May 2004.

[Rajski 05] Rajski, J., J. Tyszer, C. Wang, and S.M. Reddy, "Finite Memory Test Response Compactors for Embedded Test Applications," IEEE Trans. on Computer-Aided Design, Vol. 24, No. 4, pp. 622-634, Apr. 2005.

[Rajski 06a] Rajski, J., J. Tyszer, G. Mrugalski, W.-T. Cheng, N. Mukherjee, and M. Kassab, "X-Press Compactor for 1000x Reduction of Test Data," *Proc. of International Test Conference*, Paper 18.1, 2006.

[Rajski 06b] Rajski, W., and J. Rajski, "Modular Compactor of Test Responses," *Proc. of VLSI Test Symposium*, pp. 242-251, 2006.

[Saluja 83] Saluja, K.K., and M. Karpovsky, "Testing Computer Hardware Through Test Data Compression in Space and Time," *Proc. of International Test Conference*, pp. 83-89, 1983.

[Samaranayake 03] Samaranayake, S., E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. W. Williams, "A Reconfigurable Shared Scan-In Architecture," *Proc. VLSI Test Symposium*, pp. 9-14, 2003.

[Shah 04] Shah, M.A. and J.H. Patel, "Enhancement of the Illinois Scan Architecture for Use with Multiple Scan Inputs," *Proc. of Annual Symp. on VLSI*, pp. 167-172, 2004.

[Sharma 05] Sharma M. and W.-T. Cheng, "X-Filter: Filtering Unknowns from Compacted Test Responses," *Proc. of International Test Conference*, Paper 42.1, 2005.

[Sitchinava 04] Sitchinava, N., S. Samaranayake, R. Kapur, E. Gizdarski, F. Neuveux, and T.W. Williams, "Changing the Scan Enable During Shift," *Proc. VLSI Test Symposium*, pp. 73-78, 2004.

[Smith 85] Smith, G., "Model for Delay Faults Based Upon Paths," *Proc. of International Test Conference*, pp. 342-349, Nov. 1985.

[Tang 03] Tang, H., S.M. Reddy, and I. Pomeranz, "On Reducing Test Data Volume and Test Application Time for Multiple Scan Designs," *Proc. Int. Test Conf*., pp. 1079-1088, 2003.

[Tang 06] Tang, Y., H.-J. Wunderlich, P. Engelke, I. Polian, B. Becker, J. Scholöffel, F. Hapke, and M. Wittke, "X-Masking During Logic BIST and Its Impact on Defect Coverage," IEEE Trans. on VLSI, Vol. 14, No. 2, Feb. 2006.

[Touba 06] Touba, N.A., "Survey of Test Vector Compression Techniques," IEEE Design & Test, Vol. 23, Issue 4, pp. 294-303, Jul. 2006.

[Touba 07] Touba, N.A., "X-Canceling MISR – An X-Tolerant Methodology for Compacting Output Responses with Unknowns Using a MISR," *Proc. of International Test Conference*, Paper 6.2, 2007.

[Volkerink 05] Volkerink, E.H., and S. Mitra, "Response Compaction with Any Number of Unknowns Using a New LFSR Architecture," *Proc. of Design Automation Conference*, pp. 117-122, 2005.

[Waicukauski 87] Waicukauski, J., E. Lindbloom, B. Rosen and V. Iyengar, "Transition Fault Simulation," IEEE Design & Test of Computers, pp. 32-38, April 1987.

[Wang 04] Wang, L.-T., X. Wen, H. Furukawa, F.-S. Hsu, S.-H. Lin, S.-W. Tsai, K. S. Abdel-Hafez, and S. Wu, "VirtualScan: A New Compressed Scan Technology for Test Cost Reduction," *Proc. Int. Test Conf.*, pp. 916-925, 2004.

[Wang 06] Wang, L.T., C.-W. Wu, X. Wen, VLSI Test Principles and Architectures, Morgan Kaufmann, 2006.

[Wohl 01] Wohl, P., J.A. Waicukauski, and T.W. Williams, "Design of Compactors for Signature-Analyzers in Built-In Self-Test," *Proc. of International Test Conference*, pp. 54-63, 2001.

[Wohl 03a] Wohl, P., J.A. Waicukauski, S. Patel, and M.B. Amin, "X-Tolerant Compression and Application of Scan-ATPG Patterns in a BIST Architecture," *Proc. of International Test Conference*, pp. 727-736, 2003.

[Wohl 04] Wohl, P., J.A. Waicukauski, and S. Patel, "Scalable Selector Architecture for X-Tolerant Deterministic BIST," *Proc. of Design Automation Conference*, pp. 934-939, 2004.

[Wohl 07] Wohl, P., J.A. Waicukauski, and S. Ramnath, "Fully X-Tolerant Combinational Scan Compression," *Proc. of International Test Conference*, Paper 6.1, 2007.

[Yan 03] Yan, H., and A. Singh, "Experiments in Detecting Delay Faults Using Multiple Higher Frequency Clocks and Results from Neighboring Die," *Proc. of International Test Conference*, pp. 105-111, 2003.

Vita

Richard was born in Fort Worth, Texas on September 6, 1970 to Joe and Cuba Putman. He received his Bachelor of Science in Electrical Engineering (B.S.E.E.) degree from the University of Texas at Austin in December of 1993. He joined Crystal Semiconductor (a subsidiary of Cirrus Logic) in 1994. While working at Cirrus Logic as a Product/Test engineer, he pursued the degree of Master of Science in Engineering (M.S.E), which he obtained from the University of Texas at Austin in December of 1999. Upon graduating, he switched positions to Design-for-Test Engineer, where he has filed for three patents on improved scan testing circuits and techniques. While continuing to work full-time, he has been pursuing his Ph.D. since the fall of 2004. He has worked at Cirrus Logic for 14 years and been an active member of Woodlawn Baptist Church for the past 17 years, serving as a deacon and Bible Study teacher. He has been married to his wife Alicia since 1994 and has two lovely children, Jadon and Brenna, ages 5 and 2 respectively.

Permanent address: 8624 Piney Creek Bend
Austin, Texas, 78745

This dissertation was typed by Richard Putman.