Copyright

by

Jinkyu Lee

2006

The Dissertation Committee for Jinkyu Lee Certifies that this is the approved version of the following dissertation:

Low Power Scan Testing and Test Data Compression

Committee:

Nur A. Touba

Tony Ambler

Adnan Aziz

David Z. Pan

Changhae Park

Low Power Scan Testing and Test Data Compression

by

Jinkyu Lee, B.A.; M.S.

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 2006

Dedication

My thanks go, as always, to my parents, Byoung-Ik Lee and Suk-Nye Kim, and my sister, Shin-Young Lee, to whom this dissertation is dedicated. They have been strong supporters of me since I first started my Ph.D study. I am a blessed person from God for having parents and a sister like them.

Dedicated to my beloved parents-in-law, Jong-Suk Lee and Kwan-Hee Kim who give me constant love and care. My thanks will be always with them.

To my wife, Hwa-Young Lee who is my best critic and supporter. She is the only person who can accept and love everything about me. Thanks again.

To my sister-in-law, Sookyoung Lee, her husband, Byounghee Jin, and brother-in-law, Soongoo Lee. Thanks for their love and support.

And

To my supervising professor, Nur A. Touba.

Acknowledgements

I wish to thank my advisor, Professor Nur A. Touba. He has given me tremendous support, care and encouragement while working toward my Ph.D. He has truly enabled me to express my research and has encouraged me in all of my research experiences.

Finally, my thanks to the other committee members, Dr. Changhae Park, Prof. Tony Ambler, Prof. Adnan Aziz and Prof. David Pan for their input and support.

Low Power Scan Testing and Test Data Compression

Publication No.______________

Jinkyu Lee, Ph.D.
The University of Texas at Austin, 2006

Supervisor: Nur A. Touba

As the size and complexity of systems-on-a-chips (SOCs) continue to grow, test data volume and test power consumption have increased dramatically. A large amount of test data causes long test time and a large memory requirement on the tester. Large power consumption during test can result in high packaging cost and Vdd drop/ground bounce problems. In this dissertation, five techniques for reducing test data volume, test power consumption, or both, are proposed. The first is a new encoding algorithm that can be used in conjunction with any LFSR reseeding scheme to significantly reduce power consumption during test. The second is a scheme for inserting a linear feedforward network composed of XOR gates in the scan chains to reduce power consumption during test by reducing the number of scan shift cycles. The third is a built-in self-test (BIST) scheme that both reduces overhead for detecting random-pattern-resistant (r.p.r.) faults as well as reduces power consumption during test. The fourth is a technique for improving the compression achieved with any linear decompressor by adding a small non-linear decoder that exploits bit-wise and pattern-wise correlation present in test vectors. Finally, the fifth is a new and more efficient approach for test vector compression utilizing ATE vector repeat.

Table of Contents

List of Tables x

List of Figures xi

Chapter 1 Introduction 1

- 1.1 Test power 2
- 1.2 Test data 3
- 1.3 Test power vs. test data 3
- 1.4 Overview 4

Chapter 2 LFSR reseeding scheme achieving low power dissipation during test 9

- 2.1 Related work 9
- 2.2 Encoding algorithm 12
  - 2.2.1 Basic concept 12
  - 2.2.2 Conversion procedure 14
  - 2.2.3 Partitioning into hold cube compatible sets 15
- 2.3 Hardware implementation 17
- 2.4 Experimental results 19
- 2.5 Conclusion 24

Chapter 3 Reducing shift cycles and test power using linear feedforward network 25

- 3.1 Scan chain modification 25
  - 3.1.1 Linear feedforward network generation 25
  - 3.1.2 Scan chain length modification 29
  - 3.1.3 Test response observation 30
- 3.2 Test stimulus and response generation 32
  - 3.2.1 Test stimulus transformation 32
  - 3.2.2 Test response transformation 35
- 3.3 Applicable architecture 36
  - 3.2.1 Tester direct access 36

3.2.2 EDT architecture 36
3.4 Experimental results.. 37
3.5 Conclusion.. 39

Chapter 4 Low power BIST based on scan partitioning 40
4.1 Proposed scheme 40
4.1.1 3-valued weight set generation... 41
4.1.2 Scan chain partitioning.. 42
4.1.3 3-weight decoder minimization... 47
4.2 Architecture 48
4.2.1 Fixed-length scan architecture 48
4.2.2 Variable-length scan architecture.. 49
4.3 Experimental results... 50
4.4 Conclusion... 53

Chapter 5 Combining linear and non-linear test vector compression using correlation-based rectangular encoding 54
5.1 Rectangular encoding.. 54
5,1.1 Overview 55
5.1.2 Partitioning scan slices into rectangles 57
5.1.3 Reducing size of chain select mask.. 58
5.1.4 Forming test cube clusters.. 58
5.2 Rectangular decoder.. 60
5.3 Experimental results... 62
5.4 Conclusion... 67

Chapter 6 Efficiently utilizing ATE vector repeat for compression by scan vector decomposition 68
6.1 Related work 68
6.2 Decompression hardware 71
6.3 Forming test cube clusters... 73
6.3.1 Clustering algorithm.. 73
6.3.2 Handling constraints on number of clusters 76
6.4 Experimental results... 79

6.5 Conclusion 82
Chapter 7 Conclusion and future work 83
7.1 Conclusion 83
7.2 Future work 83
Bibliography 86
Vita 89

List of Tables

Table 2.1: Results for proposed encoding scheme 20

Table 2.2: Results for partial reseeding and proposed scheme 22

Table 2.3: Results comparing alternating dual-LFSR reseeding, run-length code and proposed scheme 23

Table 3.1: Results for proposed scheme... 38

Table 3.2: Results comparing proposed scheme with previous schemes.......... 39

Table 4.1: Results using LT-RTPG.. 50

Table 4.2: Results for proposed method in fixed-length scan architecture 51

Table 4.3: Results for proposed method in variable-length scan architecture .. 51

Table 4.4: Comparison with serial-fixing WRBIST... 53

Table 5.1: Results for proposed rectangular encoding scheme 64

Table 5.2: Results combined with partial reseeding... 65

Table 5.3: Results comparing with [Ward 05] 65

Table 6.1: Results for proposed scheme on benchmark citcuits 80

Table 6.2: Results for using linear decompressor in [Krishna 01] alone versus using it with proposed scheme.. 80

Table 6.3: Results comparing with [Vranken 03] 80

Table 6.4: Results comparing with [Wang 05]... 81

List of Figures

Figure 2.1: Example of encoding test data 13

Figure 2.2: Example of conversion procedure 15

Figure 2.3: Hold cube before partitioning 16

Figure 2.4: Hold cube after partitioning 16

Figure 2.5: Hardware implementation 18

Figure 2.6: Data format 18

Figure 2.7: Percentage change in specified bits 21

Figure 2.8: Power reduction 21

Figure 3.1: Test set 26

Figure 3.2: Scan chain configuration initialization 26

Figure 3.3: Test cube generated 27

Figure 3.4: New scan chain configuration 27

Figure 3.5: Original scan chains 30

Figure 3.6: Scan chain length modification 30

Figure 3.7: 3 unobservable test response bits 31

Figure 3.8: Adding another scan-out 31

Figure 3.9: Reverse transformation 33

Figure 3.10: After reverse transformation 33

Figure 3.11: After forward transformation 34

Figure 3.12: Final matix operation 34

Figure 3.13: Test response transformation 36

Figure 4.1: 3-valued weight set generation 42

Figure 4.2: Step 1: Initialization 44

Figure 4.3: Step 2, 3: Minimum clique covering 44
Figure 4.4: Step 4: Adjusting scan length and the number of scan chains 45
Figure 4.5: Decoder minimization 47
Figure 4.6: Architecture 49
Figure 4.7: Step 4: Adjustment in variable-length scan architecture 50
Figure 5.1: Example of rectangles 56
Figure 5.2: Data format 56
Figure 5.3: Control data for three rectangles 56
Figure 5.4: Example of clustering 60
Figure 5.5: Block diagram for rectangular decoder 61
Figure 5.6: Specified bits vs. width and *k* value for *s38417* 63
Figure 5.7: Results with test set that has few specified bits 66
Figure 6.1: Example of proposed encoding scheme 70
Figure 6.2: Selecting scan sequence 70
Figure 6.3: Decompression hardware 72
Figure 6.4: Example of sequential linear decompressor 73
Figure 6.5: Global diagram 74
Figure 6.6: Sub-diagram of one cluster generation 74
Figure 6.7: Example of clustering 76
Figure 6.8: Modification of clustering algorithm 77
Figure 6.9: Number of specified bits in encoded data vs. *k* 78
Figure 6.10: Number of clusters vs. *k* 78

Chapter 1: *Introduction*

As the size and complexity of systems-on-a-chips (SOCs) continue to grow, the cost of VLSI test is increasing drastically. Larger chips require a larger amount of test data and dissipate a larger amount of power during test. Moreover, they are typically harder to test because they tend to have more hard-to-detect faults. Test time is a critical part of test cost and increases as the size and complexity of a chip increase. Reducing test cost is becoming an increasingly critical issue. This dissertation focuses on two important sources of the test cost, namely test data volume and test power.

Test time depends on both test data volume and test power. Large test data volume increases test time because it requires more time to transfer the data to and from the chip. Test power can slow down test speed, thereby increasing test time. If the average power consumption during test is higher than the chip package's capability to dissipate heat, the test must be run at a lower frequency. Therefore, both test power and test data should be considered to reduce test time effectively. Conventional test data compression schemes generally dissipate high power. Most conventional compression schemes exploit the fact that a test set has a large number of don't cares and only 1~5% of specified (care) bits. The don't cares are assigned to maximize compression. In this process, a large number of transitions may occur in test patterns. The larger the number of transitions in the test patterns, the larger the power dissipation. This dissertation addresses these two important problems in the VLSI testing area, namely test data volume and test power.

This chapter provides background on the issues related to test power and test data volume. Section 1.1 describes test power and the reason why test power is an important issue. Section 1.2 describes the problems that large test data volume can cause.

Section 1.3 describes why and how those two problems are considered together in this dissertation. Section 1.4 provides an overview of the dissertation.

1.1 TEST POWER

Power consumption during test is proportional to the frequency of charging and discharging of internal parasitic capacitances of a component, which means that the power consumption depends on the number of 0-to-1/1-to-0 transitions in a chip. The reason why power dissipation during test is a big problem is because power dissipation during test is much larger than power dissipation during functional operation. During functional operation, only a small percentage of nodes in a chip have a transition during a clock cycle because only a small percentage of input bits change while the rest of the input bits keep their previous values. On the other hand, test patterns make transitions on a large number of nodes in a chip to provoke and propagate as many potential faults as possible. And, there is typically little to no correlation between two consecutive test patterns while two consecutive functional input patterns are highly correlated to each other. When a scan architecture is used during test, a test pattern is loaded by shifting one bit of the test pattern per clock cycle into the scan chain while at the same time, the test response from the previous test pattern is unloaded from the scan chain. During scan shifting, large power dissipation occurs in the scan chains and the circuit under test. Due to the reasons that are described above, power dissipation during test is much larger than power dissipation during functional operation. Large power dissipation during test can cause problems with heat dissipation thereby requiring that the chip be tested at lower frequency, thereby increasing test time. Minimizing power during test is important for reducing test time and hence test cost.

1.2 TEST DATA

Test data includes both test vectors and test responses. In conventional external testing, all the test data is stored on an external tester and transferred to/from the chip. Testers have limited memory, speed and I/O channels. The chip cannot be tested any faster than the time required to transfer the data to/from the chip which is equal to the amount of test data divided by test data bandwidth between the tester and chip. Reducing the amount of test data stored on the tester directly reduces test cost.

1.3 TEST POWER VS. TEST DATA

Conventional test data compression schemes generally increase test power. Most conventional test data compression techniques are based on the fact that a large percentage of test set, typically 90%~95%, is filled with don't care bits. The don't care bits are assigned in a way that minimizes test data volume and not test power. To reduce the number of transitions in a chip during test, the don't care bits should be set to constant values. Then, test power dissipation will be minimized, but the don't care bits would not be used for test data compression. On the other hand, if the don't care bits are used for test data compression, then the don't care bits cannot be used for test power reduction. For example, in linear feedback shift register (LFSR) reseeding scheme that is used in several commercial tools including TestKompress by Mentor Graphics [Rajski 02] and DBIST by Synopsys, the don't care bits are assigned almost randomly, which results in large power dissipation. This is why test power can be a serious problem in test data compression techniques. A large number of test pattern bits being assigned randomly cause a large number of transitions in the scan chains thereby increasing power dissipation during test drastically. To overcome this, there is a need to find new techniques to achieve test data compression without high test power.

1.4 OVERVIEW

In this dissertation, we propose five techniques that reduce test power, test data, or both. While most previous work has focused on reducing test power for general scan testing, only recently has work been done on considering together the problems of test data compression and low power test. This dissertation makes several contributions in this area.

In Chapter 2, we propose a low-power test data compression technique based on LFSR reseeding. The basic idea in LFSR reseeding is to generate deterministic test cubes by expanding seeds. A seed is an initial state of the LFSR that is expanded by running the LFSR in autonomous mode. Since typically only 1-5% of the bits in a test vector are specified, most bits in a test cube do not need to be considered when a seed is computed because they are don't care bits. Therefore, the size of a seed is much smaller than the size of a test vector. Consequently, reseeding can significantly reduce test data storage and bandwidth. Many test data compression schemes are based on LFSR reseeding. Several commercial tools for test data compression based on LFSR reseeding have been introduced including TestKompress by Mentor Graphics [Rajski 02] and DBIST by Synopsys. A drawback of this scheme is that the unspecified bits are filled with random values resulting in a large number of transitions during scan-in thereby causing high power dissipation. We present a new encoding scheme that can be used in conjunction with any LFSR reseeding scheme to significantly reduce test power and even further reduce test storage.

In Chapter 3, we propose a scheme for inserting a linear feedforward network composed of XOR gates in scan chains. The proposed scheme reduces scan-in, scan-out, and scan clocking power by reducing the number of shift cycles required to load test patterns into scan chains. Scan-in refers to loading a test pattern in scan chains. Scan-

out refers to unloading a test response from scan chains. Scan clocking power means power dissipated due to clocking the scan chains. Reducing shift cycles also contributes to test time and test storage reduction. Using the feedforward XOR gates, the number of shift cycles can be reduced significantly with relatively small hardware overhead.

By reducing the number of the shift cycles, test power during scan-in, scan-out, and scan clocking can be reduced simultaneously. Moreover, it can also be used in conjunction with other test data compression techniques as many of the don't cares in the test set are preserved. By using it in conjunction with other test data compression schemes, the test power can be reduced significantly compared with using those test data compression schemes by themselves. This is especially useful for test data compression schemes based on linear expansion which fill don't cares with random values resulting in very high power dissipation during scan shifting.

In Chapter 4, a built-in self-test (BIST) scheme is presented which both reduces overhead for detecting random-pattern-resistant (r.p.r.) faults as well as reduces power consumption during test. 3-valued weights are employed to detect the r.p.r. faults. The key idea is to use a new scan partitioning technique and decoding methodology that exploits correlations in the weight sets to greatly reduce the hardware overhead for multiple weight sets and reduce the number of transitions during scan shifting.

Weighted pattern testing has been well studied in the literature. Early work in [Shnurnann 75] described a weighted random pattern generator in which pseudo-random patterns are biased by changing the probability of each input bit position being a '0' or a '1'. The "weight" assigned to each input bit position is determined in a way that detects as many r.p.r. faults as possible, thereby increasing the fault coverage. A 3-valued weight set generation algorithm based on a deterministic test set was presented in [Pomeranz 92]. Each input bit position is weighted to one of three values, 0, 1, or random, and this can

reduce hardware overhead for weighted random test and increase fault coverage compared to the conventional weighted random test. We describe a new BIST scheme that achieves low power without the need to re-order the scan chains. It uses a scan partitioning technique that exploits correlation in the weight sets to both simplify the weight logic as well as reduce the number of transitions in the scan chains. A methodology for designing the decoding logic is described which takes advantage of the scan partitioning. The scan cells in each partition can be ordered in any way desired (e.g., to minimize routing overhead). Consequently, the proposed scheme can be easily implemented in the standard design flow used in industry.

In Chapter 5, a technique is presented for improving the compression achieved with any linear decompressor by adding a small non-linear decoder that exploits bit-wise and pattern-wise correlation present in test vectors. The proposed non-linear decoder has a regular and compact structure and allows continuous-flow decompression. It has a very important feature which is that its design does not depend on the test data. This simplifies the design flow and allows the decoder to be reused when testing multiple cores on a chip.

The amount of compression that can be achieved with linear compression schemes depends directly on the number of specified bits in the test cubes. While linear decompressors are very efficient at exploiting don't cares in the test set, they cannot exploit correlations in the test cubes, and hence they cannot compress the test data to less than the total number of specified bits in the test data. Non-linear decompressors on the other hand can exploit correlations in the test cubes, but are not as efficient as linear decompressors in exploiting don't cares. Because test data is typically only 1-5% specified with the rest as don't cares, linear decompressors are generally more effective

overall. This fact coupled with the simple and compact design of linear decompressors is the main reason why they are used in commercial tools.

The approach taken is to combine linear and non-linear compression together to get the advantages of both. A non-linear decompressor is used to exploit correlations in the specified bits to reduce the number of specified bits that the linear decompressor has to produce. Since the amount of compression achieved with a linear decompressor depends on the number of specified bits it needs to produce, this approach results in much greater compression than what the linear decompressor could achieve by itself.

In Chapter 6, a new and more efficient approach is proposed for utilizing ATE vector repeat. The scan vector sequence is partitioned and decomposed into a common sequence which is the same for an entire cluster of test cubes and a unique sequence that is different for each test cube. The common sequence can be generated very efficiently using ATE vector repeat.

Test vector compression involves storing a deterministic test set on the automated test equipment (ATE) in a compressed form. One instruction that is commonly found in most ATEs is *vector repeat* which allows the ATE to repeat a vector *n* times. The amount of compression is limited by the amount of ATE instruction memory available and the length of the runs. Chapter 6 proposes a new and more efficient way of utilizing ATE vector repeat. The idea is to exploit the fact that many test cubes have similar input assignments due to the fact that they are structurally related in the circuit. The set of test cubes in a test set are partitioned into clusters that share many input assignments. The scan sequence is then decomposed into two components: the sequence of specified bits that is common across all the test cubes in a cluster, and the sequence of specified bits that is unique to each test cube. Two separate on-chip decompressors are then used to generate these two sequences. Since the common sequence is the same for all the test

cubes in a cluster, the input stream to its decompressor can be generated using ATE vector repeat. Only one copy of this input stream needs to be stored in the ATE vector memory, and only one ATE vector repeat instruction needs to be stored in the ATE instruction memory for decompressing the entire test cube cluster. For the unique sequence corresponding to each test cube, it is generated with its own decompressor. The unique sequence will only contain very few specified bits because most of the specified bits will be generated by the common sequence decompressor. If a linear decompressor based on dynamic LFSR reseeding such as those described in [Krishna 01], [Konemann 01], and [Rajski 02] is used, the amount of compression depends only on the number of specified bits in the sequence. Thus, the unique sequence for each test cube can be highly compressed. The same type of decompressor can also be used to generate the common sequence for each test cluster to achieve high compression for it as well since 95-99% of the specified bits in a test cube are don't cares. Note that the design of these decompressors is independent of the test set, so they can be reused when testing multiple cores in a system-on-chip (SOC) design.

Chapter 2: *LFSR Reseeding Scheme Achieving Low Power Dissipation During Test*

This chapter presents a new low power test data compression scheme based on LFSR reseeding. A drawback of compression schemes based on LFSR reseeding is that the unspecified bits are filled with random values resulting in a large number of transitions during scan-in thereby causing high power dissipation. A new encoding scheme that can be used in conjunction with any LFSR reseeding scheme to significantly reduce test power and even further reduce test storage is presented. The proposed encoding scheme acts as a second stage of compression after LFSR reseeding. It accomplishes two goals. First, it reduces the number of transitions in the scan chains (by filling the unspecified bits in a different manner), and second it reduces the number of specified bits that need to be generated via LFSR reseeding. Experimental results indicate that the proposed method significantly reduces test power and in most cases provides greater test data compression than LFSR reseeding alone.

2.1 RELATED WORK

The idea of considering together the problems of test data compression and low power test has been previously investigated in a few papers. In [Sankaralingam 00], a procedure for directing the static compaction process in a manner that reduces test power and test data was described. In [Chandra 01], an encoding algorithm that reduces both test storage and test power was presented. The test cubes are encoded using a Golomb code which is a run-length code. All don't care bits are mapped to 0 and the Golomb code is used to encode runs of 0's. The Golomb code efficiently compresses the test data, and the mapping of the don't cares to all 0's reduces the number of transitions during scan-in and thus power. One drawback of a Golomb code is that it is very

inefficient for runs of 1's. In fact, the test storage can even increase for test cubes that have many runs of 1's.

A method based on an alternating run-length code was presented in [Chandra 02] to improve on the encoding efficiency of a Golomb code. While a Golomb code only encodes runs of 0's, an alternating run-length code can encode both runs of 0's and runs of 1's. However, the code becomes inefficient when a pattern is encoded where the runs are short.

While both Golomb codes and alternating run-length codes are good for reducing test power, they are not as efficient as LFSR reseeding for compressing test data. With LFSR reseeding, only the specified bits, which generally account for only 1-5% of the test data, need to be considered.

One previous method has been proposed in [Rosinger 02] for reducing test power for LFSR reseeding. Two LFSRs are used. The main LFSR generates the test cube through conventional reseeding. An extra "masking" LFSR is used to generate a set of mask bits. If the number of 1's in a test cube is less than the number of 0's, then the output of the two LFSRs are ANDed together and the mask cube will have a 1 for each specified 1 in the test cube and an X for each specified 0 or X in the test cube. If the number of 0's in the test cube is less than the number of 1's, then the output of the two LFSRs are ORed together and the mask cube will have a 0 for each specified 0 in the test cube and an X for each specified 1 or X in the test cube. A seed is computed for the extra "masking" LFSR so that it generates the mask cube. Thus the effective number of specified bits that must be generated using this method is equal to the original number of specified bits in the test cube plus the number of specified bits in each mask cube (which is equal to the minimum of the number of 0's or 1's in each test cube). The size of the main LFSR is the same as for conventional reseeding, and the size of the extra "masking"

LFSR depends on the maximum number of specified bits in any mask cube. Test power is reduced because the output of the two LFSRs are ANDed or ORed, thus reducing the transition probability. However, the test data compression for this scheme is greatly reduced compared with conventional LFSR reseeding because it requires storing an extra set of seeds for the extra "masking" LFSR. Results indicate that the test storage was increased by 21% to 54% compared with conventional LFSR reseeding while the transition count was reduced by about 24%.

The proposed method addresses the problem of reducing test power for LFSR reseeding. However, a different approach is taken which does not compromise the amount of test data compression. Moreover, the number of transitions is reduced much more significantly than in [Rosinger 02]. The experimental results for the proposed method are compared with the previous methods in Chapter 2.4.

The proposed scheme has some similarity to the dynamic scan scheme presented in [Samaranayake 02], however there are a number of significant differences. Both schemes use the fact that different test cubes have compatible values for a significant number of scan elements. The dynamic scan scheme identifies scan chain segments that have compatible values across a set of test cubes and bypasses those segments in order to reduce test time and test storage. This approach also serves to reduce power as well, although that is not the focus of the paper. The proposed scheme also takes advantage of compatible scan segments, but in a different way. The proposed scheme exploits the property that the number of transitions in a test cube is smaller than the number of specified bits. By dividing the scan chains into blocks and identifying blocks that do not contain transitions, the proposed approach is able to fill those blocks with constant values. The compatibility of blocks across different test cubes is exploited in reducing control information and not through bypassing. In [Samaranayake 02], scan chain

reconfiguration is necessary which requires inserting design-for-test (DFT) logic in the scan chains themselves to provide the bypass capability, whereas for the proposed scheme, DFT logic is inserted only at the inputs of the scan chains and is thus compatible with conventional scan chains. The proposed scheme can be used for hard cores and firm cores.

2.2 ENCODING ALGORITHM

Let a transition in a test cube be defined as a specified 0 (1) followed by zero or more X's followed by a specified 1 (0). The key idea of the proposed encoding algorithm is to take advantage of the fact that number of transitions in a test cube is always less than the number of specified bits in a test cube. Thus, rather than using LFSR reseeding to directly encode the specified bits as in conventional LFSR reseeding, the proposed encoding algorithm divides the test cube into blocks and only uses LFSR reseeding to produce the blocks that contain transitions. For the blocks that do not contain transitions, the logic value fed into the scan chain is simply held constant. This approach reduces the number of transitions in the scan chains and in most cases also reduces the total number of specified bits that must be generated by the LFSR as compared with conventional LFSR reseeding.

2.2.1 Basic Concept

The proposed encoding scheme encodes each test cube with two kinds of data: *hold flags* and *data bits*. Each test cube is divided into several blocks and each block has a one-bit hold flag. The hold flag indicates whether a transition occurs in a block. There are three types of blocks:

1) Transition block (Hold flag = 0)

One or more transitions exist in the block. Either both 0 and 1 are present in the block (e.g., XX1X0X), or only 0 or 1 is present but the last specified bit from a previous block was opposite.

2) Non-transition block (Hold flag = 1)

No transition occurs in current block. Only 0 or 1 is present in the block, and the last specified bit from a previous block is same (e.g., X0XX0X).

3) Don't care block (Hold flag = X)

No specified bits occur in the block, all are don't cares.

If the hold flag for a block is 1, then the data bits in the block are simply held constant from the last data bit in the previous block. If the hold flag is 0, then the data bits are loaded directly from the LFSR. If the hold flag is X, then it can be either treated as a non-transition block or as a transition block with all X data. Both the hold flags and the data bits are generated from a single LFSR using reseeding.

Block	Block1					Block2					Block3					Block4				
Original		0	X	X	1		X	1	1	1		1	X	1	X		X	X	X	X
Encoded	**0**	0	X	X	1	**1**	-	-	-	-	**1**	-	-	-	-	**X**	X	X	X	X

Figure 2.1. Example of encoding test data

An example of the proposed encoding is shown in Fig. 2.1. The test sequence in the example is composed of 4 blocks and each block has 1 hold flag and 4 data bits. The hold flags are shown in bold in the "Encoded" bit sequence row. In Fig. 2.1, the original test cube contains 7 specified bits. However, using the proposed encoding scheme, the encoded data has only 3 specified hold flags and 2 specified data bits giving a total of only 5 specified bits. Thus, the proposed encoding scheme reduces the number of specified bits that need to be generated using LFSR reseeding. As shown in Fig. 2.1, the 1's in block 2 and block 3 don't need to be generated directly by the LFSR, but are

rather generated as a by-product of the fact that the hold flags keep the input to the scan chain held constant at 1. Thus, test data compression can be achieved in this way. Moreover, no transitions will occur when generating block 2 and block 3 because the hold flags are 1 thus keeping all the bits in the blocks constant. This would not be the case in conventional LFSR reseeding where the X's in blocks 1 and 2 get filled with random data which may results in many more transitions. Thus, a reduction in the number of transitions can be achieved in this way.

2.2.2 Conversion Procedure

It is possible to increase the number of non-transition blocks by converting some transitions blocks into non-transition blocks. There are two requirements that must be satisfied in order to convert a transition block into a non-transition block. The first is that it cannot contain both specified 0's and specified 1's. The second is that the last bit of the previous block must be an X. Two examples of this are shown in Fig. 2.2. Block 2 is initially a transition block even though it only contains specified 0's because the last specified bit in block 1 was a 1. However, the very last bit of block 1 is a don't care, so a *conversion procedure* can be used to specify that don't care as a 0 and thereby convert block 2 into a non-transition block. Even though this conversion required adding an extra specified data bit, the net result is still a reduction in the total number of specified bits because now block 2 is a non-transition block and thus none of its data bits need to be generated by the LFSR. This same conversion procedure can also be used to convert block 4 in Fig. 2.2 into a non-transition block.

By increasing the number of non-transition blocks, the conversion procedure can help to reduce both test storage (since it can reduce the total number of specified bits) as well as test power (since it can reduce the number of transitions by enabling all the X's in the converted non-transition block to be filled with the same logic value).

Block	Block1					Block2					Block3					Block4				
Original		X	0	1	X		X	0	X	0		X	X	X	X		1	1	1	X
Encoded	**0**	X	0	1	0	**1**	-	-	-	-	**0**	X	X	X	1	**1**	-	-	-	-

Figure 2.2. Example of conversion procedure (last bit of blocks 1 and 3 are specified to convert blocks 2 and 4 into non-transitons blocks)

2.2.3 Partitioning into Hold Cube Compatible Sets

The test storage for LFSR reseeding depends on the number of specified bits. For each block that is not a don't care block, the hold flag for that block is specified. If the number of specified hold flags becomes larger than the number of the specified test data bits that are reduced by using the proposed encoding scheme, then the encoding scheme would be reducing test power dissipation at the cost of test storage. The test storage would increase because the total number of specified data bits plus specified hold bits would exceed the total number of specified bits in the original test cubes. However, in this chapter, a method for reducing the number of specified hold flags is introduced.

The key idea is to take advantage of the fact that many test cubes may have compatible assignments in their corresponding hold flags. We will denote the set of hold flags for one test cube as a *hold cube* since each hold flag can be either a 1, 0, or don't care (X). If several consecutive test cubes have the same hold cube, it is not necessary to change any of the hold flags. Thus, the hold flags could be loaded once and then reused when applying subsequent test cubes. The hold cubes for a pair of test cubes are compatible if they do not conflict in any specified bit positions. In other words, for every bit position where one hold cube has a specified value, the other hold cube has either the same specified value or a don't care (and vice versa). Let a *hold cube compatible set* be defined as a set of test cubes with mutually compatible hold cubes. Since typically only around 1-5% of the data bits in a test cube are specified, the

corresponding hold cube will typically have a large number of don't cares. Thus the test cubes can generally be partitioned into a relatively small number of hold cube compatible sets. The test cubes can then be ordered so that the test cubes in each hold cube compatible set will occur in succession. Thus, the hold flags only need to be loaded once for each hold cube compatible set. One extra bit per test cube is required to indicate if the hold flags for the current test cube needs to be updated or not.

Test Cube	Hold cube					
1	1	X	X	1	0	X
2	X	1	X	X	1	1
3	1	0	X	1	X	X
4	X	0	0	1	0	X
5	X	X	X	0	1	1

Figure 2.3. Hold cube before partitioning

Test Cube	Update Flag	Hold cube					
1	1	1	0	0	1	0	X
3	0	X	X	X	X	X	X
4	0	X	X	X	X	X	X
2	1	X	1	X	0	1	1
5	0	X	X	X	X	X	X

Figure 2.4. Hold cube after partitioning

Fig. 2.3 and Fig. 2.4 show an example of partitioning a test set into hold cube compatible sets. The original hold cubes for each test cube are shown in Fig. 2.3. Originally, they require 16 specified bits. They are then grouped into 2 hold cube compatible sets. The first set contains test cubes 1, 3, and 4, and the second contains test cubes 2 and 5. The test cubes are reordered so that the hold cube compatible sets are grouped together. An extra update flag bit is added to each test cube to indicate if the

hold flags need to be updated. This flag is set only for the first test cube in each hold cube compatible set. In this example, the total number of specified bits (including the added update flag bits) is reduced to 14 as shown in Fig. 2.4. In a typical circuit, the number of test cubes and the number of don't care bits are much larger than those in the example shown in Fig. 2.3. Thus, the reduction in specified bits using this approach will be sizable. In fact, in our experiments (shown in Chapter 2.5), we found that in most cases, this encoding scheme was capable of reducing the total number of specified bits (including data bits, hold flags, and update flags) to below that of the original test cubes.

2.3 HARDWARE IMPLEMENTATION

The hardware implementation for the proposed scheme is shown in Fig. 2.5. Each scan chain is divided into one or more blocks. Let B be the number of blocks per scan chain. Each scan chain has a *hold flag shift register (HF-SR)* whose size is equal to B. LFSR reseeding is used to generate all of the data for each test cube which consists of three components: update flag, hold flags, and test data. The format for the data coming out of the LFSR for each test cube is shown in Fig. 2.6.

There is a small finite state machine (FSM) controller that controls where the data coming out from the LFSR is stored. In the first clock cycle, the LFSR generates a single bit which is the update flag. If the update flag is 1, then in the next B clock cycles, the LFSR generates the hold flags for each of the scan chains which are shifted into the HF-SRs. If the update flag is 0, then the HF-SRs are not loaded. Let the length of each scan chain be L. Then for the next L clock cycles, the LFSR generates the test data. For each L/B clock cycles, if the corresponding hold flag for a scan chain is 0, then the scan chain is loaded from the LFSR. If the corresponding hold flag is a 1, then the last value shifted into the scan chain is repeatedly shifted into the scan chain and the data from the LFSR is ignored. After each L/B clock cycles, the hold flag shift

register is shifted so that the next hold flag becomes active for its corresponding block and is used as the control signal to a MUX (as shown in Fig. 2.5). After the scan chains have been filled, then the scan vector is applied to the circuit-under-test and the response is loaded back into the scan chain. The process is then repeated to generate the next scan vector.

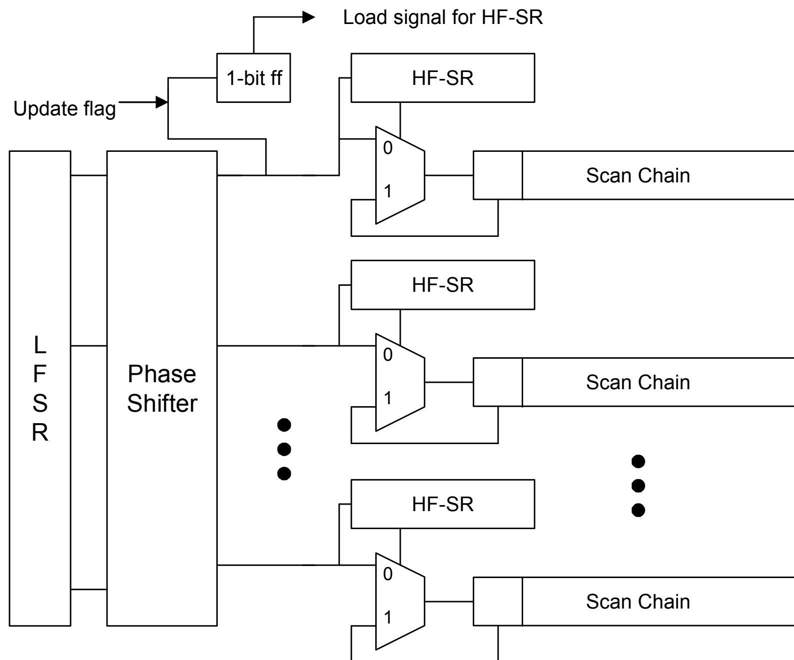


Figure 2.5. Hardware implementation

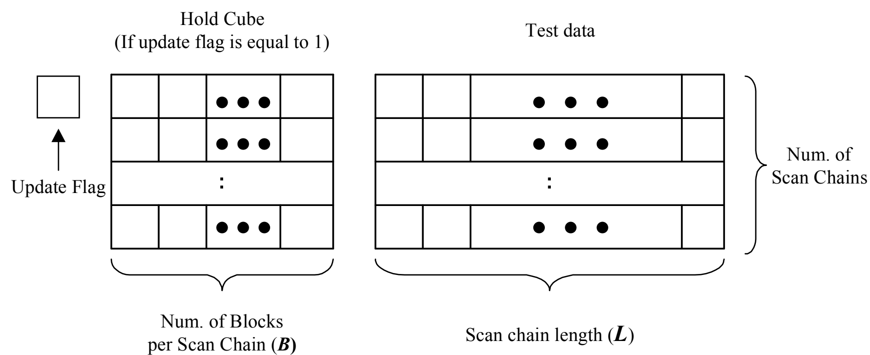


Figure 2.6. Data format

The hardware overhead consists of one 2-to-1 MUX and a HF-SR per scan chain, one 1-bit update flag flip-flop and the small FSM controller. The FSM controller consists of a bit counter (which is present for LFSR reseeding anyway) and some small combinational logic. The size of the HF-SR dominantly determines the hardware overhead in this scheme. It depends on the number of scan chains and the total number of blocks.

2.4 EXPERIMENTAL RESULTS

Experimental results for the proposed scheme for the largest ISCAS 89 benchmark circuits are shown in Table 2.1. Results are shown for dividing each test cube into different numbers of blocks (note that there is one hold flag for each block). The test cubes were partitioned into hold cube compatible sets, and the number of such sets is shown in each case. The total number of specified bits required for the proposed encoding scheme is shown (including update flags, hold flags, and data bits). The total number of specified bits and total number of transitions (computed as described in [Sankaralingam 00]) for the proposed encoding scheme is compared with that for the original test cubes. In most cases, the total number of specified bits is reduced, which will result in less test data bits in the test data and hold flags. The more blocks that are used, the less specified bits in the test data, but the more specified hold flags. Moreover, the reduction in the number of transitions increases as the number of blocks in a test pattern increases. Note that the number of transitions in HF-SR is included in the number of transitions shown in the ninth column. The hardware overhead also increases in this case as the size of the hold flag shift registers becomes larger. The hardware overhead depends on the number of scan chains. In these experiments, the number of scan chains is chosen depending on the circuit size. The eleventh column indicates the number of 2-to-1 MUXs required which is equal to the number of scan chains because

one MUX is located on the entrance of each scan chain. The size of HF-SR is indicated in the last column. It depends on the number of blocks.

Table 2.1: Results for proposed encoding scheme

Circuit Information				Test Storage				Test Power		Overhead	
Circuit Name	Num. Test Patt.	Num. Blocks	Num. Compat. Sets	Num. Spec. Data Bits	Num. Spec. Hold Flag	Total Spec. Bits	Red.	Num. Trans.	Red.	Num. MUXs	HF-SR
s5378	196	5	51	4613	476	5089	-2.8	361552	31.6	5	1
		10	94	4312	961	5273	+0.7	154111	38.7	10	1
		20	126	4151	1352	5503	+5.1	62141	47.9	10	2
		30	143	3972	2050	6022	+15.0	42057	50.8	10	3
		40	147	3902	2370	6272	+19.8	29120	54.4	10	4
s9234	205	5	28	9840	143	9983	-3.3	524899	26.5	5	1
		10	91	8486	1333	9819	-4.9	208477	34.3	10	1
		20	115	7783	2386	10169	-1.5	85634	46.3	10	2
		30	143	6906	3325	10231	-0.9	53283	50.6	10	3
		40	175	6432	4037	10469	+1.4	40184	52.9	10	4
s13207	266	5	26	8958	196	9154	-2.5	5542237	34.0	5	1
		10	63	8536	609	9145	-2.6	2114606	36.3	10	1
		30	81	7910	1310	9220	-1.8	633997	48.7	20	2
		50	92	7697	1899	9596	+2.2	353450	51.5	20	3
		70	120	6735	2419	9154	-2.5	243268	52.1	20	4
s15850	269	10	95	9873	786	10659	-2.6	1731388	34.6	10	1
		20	115	9502	1267	10769	-1.6	777594	40.1	20	1
		40	139	8940	2190	11130	+1.7	348558	47.5	20	2
		60	139	8278	2939	11217	+2.5	231309	49.5	20	3
		80	156	7475	3884	11359	+3.8	154525	52.7	20	4
s38417	376	10	46	28106	661	28767	-6.2	21328669	25.0	10	1
		30	117	27035	2039	29074	-5.2	6247820	33.7	30	1
		60	161	24891	4673	29564	-3.6	2625507	41.4	30	2
		90	181	22883	6375	29258	-4.6	1600121	45.7	30	3
		120	164	20486	8005	28491	-7.1	1107107	52.0	30	4
s38584	296	10	125	25338	1161	26499	+1.2	14001208	20.3	10	1
		30	244	24008	5214	29222	+11.6	3659975	28.4	30	1
		60	276	22770	8024	30794	+17.6	1559152	33.1	30	2
		90	267	21464	10403	31867	+21.7	995869	46.7	30	3
		120	265	20684	11549	32233	+23.1	715933	53.9	30	4

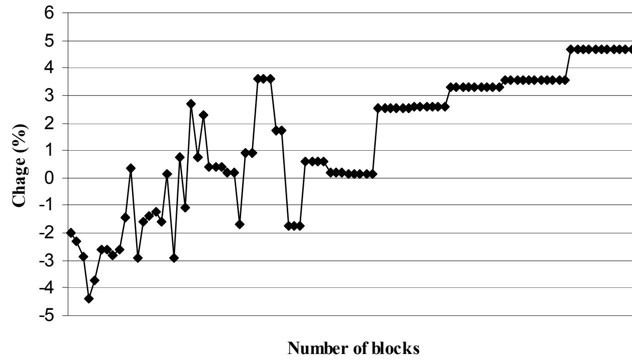


Figure 2.7. Percentage change in specified bits

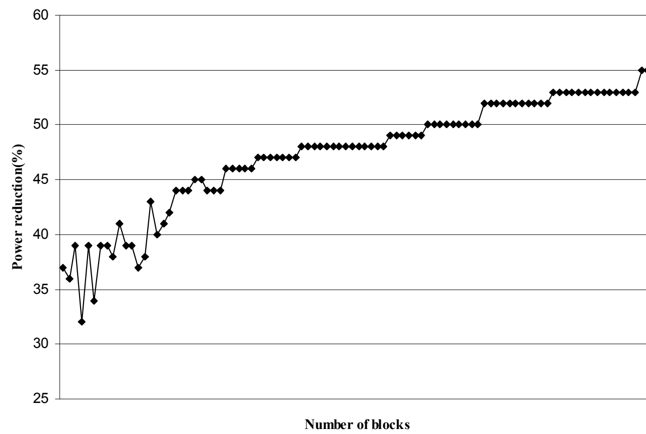


Figure 2.8. Power reduction

Fig. 2.7 and Fig. 2.8 show the percentage change in specified bits and power reduction compared to the original test cubes for one benchmark circuit, *s15850*. It illustrates how to choose the number of blocks for a corresponding circuit and test set. The number of blocks is a user-defined variable. It is chosen by considering test power, the number of specified bits, and hardware overhead simultaneously. The number of blocks simulated varies from 5 to 100, which is represented on x-axis for both graphs in Fig. 2.7 and Fig. 2.8. With a small number of blocks, the number of specified bits is reduced to less than the number of specified bits in the original test cubes. This is

observed in most of the other circuits as well. A small number of blocks is also good with respect to hardware overhead because it means small hardware overhead. However, the amount of power reduction is almost proportional to the number of blocks as shown in Fig. 2.8, which means the power consumption with a small number of blocks is small. With 5 to 10 blocks, the power reduction is about 37% while the average of the power reduction is about 50%. If 37% power reduction is good enough for a user's test methodology, a number from 5 to 10 is chosen as the number of blocks because it causes very small hardware overhead and can achieve high test data compression. If 37% power reduction is not good enough, a larger number of blocks is chosen while still trying to minimize the number of specified bits. In Fig. 2.7 and Fig. 2.8, 38 or 39 blocks can be chosen. This can achieve 48% power reduction and 1.8% reduction of the number of specified bits with relatively small hardware overhead, but not as small as hardware overhead with 5 to 10 blocks.

Table 2.2: Simulation results for partial reseeding and proposed scheme (After Pseudo-random sequence of 10,000 patterns)

Circuit Name	Partial Reseeding [Krishna 01]		Partial Reseeding with Proposed Encoding Scheme				
	Num. Specified Bits	Test Storage	Num. Specified Bits	Test Storage	Reduction of Specified Bits	Change in Test Storage	Reduction in Transitions
s5378	508	533	501	514	1	-3.56	54
s9234	5198	5537	4031	4556	22	-17.7	57
s13207	2824	3008	2779	3021	1	+0.4	56
s15850	5092	5204	4166	4896	18	-5.9	57
s38417	23984	24513	20688	23166	13	-5.5	52
s38584	2848	2942	2263	2827	7	-3.9	45

The proposed encoding scheme can be used in conjunction with any LFSR reseeding scheme. Experiments were performed for using the proposed encoding scheme in conjunction with the partial LFSR reseeding scheme described in [Krishna 01]. The results are shown in Table 2.2. The exact same set of test cubes that were used for

generating the results published in [Krishna 01] were encoded using the proposed encoding scheme in conjunction with the scheme in [Krishna 01]. As can be seen, in most cases both the test storage and test power are reduced using the proposed scheme.

Table 2.3: Simulation results comparing dual-LFST reseeding, alternating run-length code and proposed scheme

Circuit	Dual-LFSR Reseeding [Rosinger 02]			Alternating Run-Length code [Chandra 02]			Proposed Scheme		
	Test Storage	Comp. (%)	% Power Reduc.	Test Storage	Comp. (%)	% Power Reduc.	Test storage	Comp. (%)	% Power Reduc.
s9234	19440	68	24	21612	44	76	10302	79	53
s13207	11803	94	25	32648	80	93	10484	94	53
s15850	14518	90	25	26306	65	85	11411	93	52
s38417	66234	92	25	64976	60	81	32152	95	52
s38584	23835	94	25	77372	61	83	31152	93	40

Table 2.3 shows a comparison of the experimental results in [Chandra 02] and [Rosinger 02] with the proposed encoding scheme (used in conjunction with partial LFSR reseeding as described in [Krishna 01]). As can be seen, the proposed scheme reduces the test storage requirements much more than the other schemes. Note that the test storage for the method in [Rosinger 02] was calculated here by multiplying the size of the primary and secondary LFSR by the number of test cubes. In terms of reducing test power, the proposed scheme is much more effective than the scheme in [Rosinger 02] which is also applicable for LFSR reseeding. Moreover, the compression ratio in the proposed scheme is similar or even higher than that in [Rosinger 02] even though 1,000 pseudo-random patterns are applied first in [Rosinger 02]. Note that the results for both [Chandra 02] and the proposed scheme are for encoding the entire deterministic test set. While the test power for the proposed scheme is not reduced as much as for the scheme in [Chandra 02] which is based on run-length encoding, much more compression is achieved. The key advantage of the proposed scheme compared with [Chandra 02] is

that it is compatible with LFSR reseeding which is used in commercial tools due to its superior encoding efficiency.

2.5 Conclusion

LFSR reseeding is a powerful approach for reducing test storage. The proposed encoding scheme provides a way to reduce test power for LFSR reseeding. It acts as a second stage of compression after LFSR reseeding. By employing hold flags, not only is test power reduced, but also test storage can be reduced.

Chapter 3: *Reducing Shift Cycles and Test Power Using Linear Feedforward Network*

A scheme is proposed for inserting a linear feedforward network composed of XOR gates in scan chains. The proposed scheme reduces scan-in, scan-out, and scan clocking power by reducing the number of shift cycles required to load test patterns into scan chains. Reducing shift cycles also contributes to test time and test storage reduction. After inserting the linear feedforward network, the corresponding test stimulus and test response are obtained by solving linear equations for the specified bits. Using the feedforward XOR gates, the number of shift cycles can be reduced significantly with relatively small hardware overhead. The proposed scheme can be used in standard scan architectures as well as in conjunction with other test data compression techniques. Because it can preserve many of the X's in the test cubes, it is very efficient when used with linear expansion schemes to reduce power significantly.

3.1 Scan Chain Modification

The proposed scheme involves modifying scan chains to reduce the number of shift cycles needed to load each test cube by generating only the specified bits. Chapter 3.1.1 describes a procedure for designing a linear feedforward network. Chapter 3.1.2 describes a method to modify scan chain lengths to improve the efficiency of linear feedforward networks. Chapter 3.1.3 discusses test response observation with a linear feedforward network.

3.1.1 Linear Feedforward Network Generation

Fig. 3.1 shows a test set that needs to be generated. In this example, only one scan chain is used for simplification and the scan chain is 8-bits long. Therefore, 8 shift cycles are needed to load a scan vector. The first step of feedforward network

generation is to pick the test cube that has the largest number of specified bits, *t2* or *t5* in this example. If *t2* is chosen, then *c1*, *c2, c7* and *c8* are required to be specified to generate *t2*. Since *c3-c6* are not specified, the output of the scan cell *c7* can be feedforwarded to the input of the scan cell *c2* as shown in Fig. 3.2 since it does not matter what values end up in *c3-c6*. Now, *t2* is removed from the list of test cubes because it is guaranteed that it can be generated in the current scan chain configuration. We define the path that consists of the scan cells that must be specified as the *specified scan path*. In Fig. 3.2, the specified scan path is {*c8-c7-c2-c1*}.

t1	**1**	**X**	**X**	**X**	**X**	**X**	**1**	**0**
t2	**0**	**0**	**X**	**X**	**X**	**X**	**1**	**1**
t3	**1**	**0**	**X**	**1**	**X**	**X**	**X**	**X**
t4	**1**	**X**	**1**	**X**	**X**	**X**	**X**	**0**
t5	**X**	**0**	**X**	**X**	**1**	**X**	**1**	**0**
t6	**X**	**1**	**X**	**X**	**X**	**0**	**X**	**1**

Figure 3.1. Test set

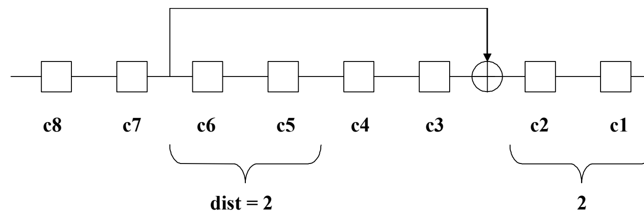


Figure 3.2. Scan chain configuration initialization

The second step is to check if other test cubes can be generated by the initial configuration. This process can be divided into two cases.

Case 1*:* If all of the specified bit positions in a test cube are included in the specified scan path*,* the test cube can be generated with current scan chain configuration without

any extra feedforward network. So, the test cubes in which all of the specified bit positions are included in the specified scan path (e.g., $t1$) are removed from the list of test cubes.

Expression : [c8 c7 c6 c5 c4 c3 c2 c1]

- **Previous test response**
  [r8 r7 r6 r5 r4 r3 r2 r1]
- **Current test pattern (Current test stimulus : s4 s3 s2 s1)**
  C = [s4 s3 s2 s1 r8 r7 (s2+r6) (s1+r5)]

Figure 3.3. Test cube generated

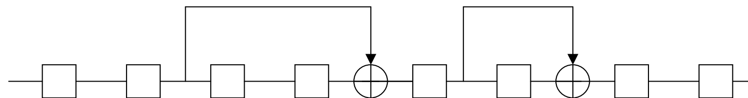


Figure 3.4. New scan chain configuration

Case 2*:* If some of the specified bit positions in a test cube are not included in the specified scan path, it may still be possible to generate the test cube with the current scan configuration. For example, $t3$ and $t4$ fall into this category. Fig. 3.3 shows the test pattern in the scan chain after 4 test stimulus bits are inserted. After 4 test stimulus bits are inserted, $s2$ determines the bit value in $c2$ and $c6$, and $s1$ determines the bit value in $c1$ and $c5$. In the other words, unless $c2$($c1$) and $c6$($c5$) need to be specified at the same time in one test cube, one of $c2$($c1$) and $c6$($c5$) can be generated with the current scan chain configuration. Therefore, $t3$and $t4$ can also be generated without any extra feedforward network. These kinds of test cubes in which the specified bits can be specified by test stimulus on the current scan chain configuration even though some of those bits are not on the specified scan path can also be removed from the list of test cubes.

A systematic method to find out if a test cube is included in case 2 is explained by the example of *t3*. In *t3*, *c5* is not on the specified scan path in Fig. 3.2. This means that *c5* is bypassed by the feedforward network. The distance between the *starting* point of that feedforward network and *c5*, *dist* in Fig. 3.2, is calculated (in this example, *dist* is 2). The distance from the *ending* point to *c1* is 2 which is the same as *dist*. The current scan chain configuration can generate *t3* if not both of *c1* and *c5* must be specified in *t3*. Therefore, *t3* can be generated with the current scan chain configuration because *c1* contains a don't care. If both must be specified in *t3*, the current scan chain is modified so that the specified scan path includes *c5*.

Now, the list of test cubes contains only *t5* and *t6*. *t5* requires *c4* to be specified. As seen in Fig. 3.3, *c4* is not controllable by any test stimulus bit. All the specified bits must be controllable by the test stimulus bits. Therefore, the current scan chain configuration should be modified to make *c4* controllable as shown in Fig. 3.4. So, the specified scan path is changed to {*c8-c7-c4-c2-c1*}. When the specified scan path is modified, a check is made again to see if there is a test cube in which all of the specified bits are included in the modified specified scan path, which means case 1 and those test cubes such as *t5* are removed from the test set. And then, it is also checked if there is any test cube that can be included in case 2. *t6* is included in case 2 with the modified scan chain configuration shown in Fig. 3.4, and it is removed from the test set. Finally, the list of test cubes is empty, and the linear feedforward network generation has been completed. For multiple scan chains, the linear feedforward network generation process for each scan chain is completely independent from each other. So the procedure above can be simply used for each scan chain. The big advantage is that the number of scan shift cycles is reduced to only 5 instead of the original 8. This improves test storage, test time, and test power.

3.1.2 Scan Chain Length Modification

One difficulty that arises for using a feedforward network with multiple scan chains is that the number of specified bits in each scan chain may be very uneven. Because the largest number of shift cycles among the scan chains determines the final shift time. The worst case scan chain may limit the benefit of the linear feedforward network. Fig. 3.5 and Fig. 3.6 show an example of this. As shown in Fig. 3.5, two scan chains are used where one requires only 2 shift cycles while the other requires 7 shift cycles such that the final shift time is 7 shift cycles. To make the proposed scheme efficient for multiple scan chains, a method to overcome this problem is proposed.

The key idea is that the lengths of the scan chains do not need to be uniform. Based on the test set, the length of each scan chain can be adjusted to distribute the specified bits more evenly. Therefore, the lengths of the two scan chains are adjusted so that *scan chain 1* and *scan chain 2* have 5 scan cells and 4 scan cells, respectively, that are included in the specified scan path, as shown in Fig. 3.6. Now, the number of shift cycles for *scan chain 1* and *scan chain 2* are 5 shift cycles and 4 shift cycles, respectively. Therefore, the final shift time is 5 scan clock cycles, and the difference between the original shift time and the reduced shift time is 2 scan clock cycles. Note that after modifying the lengths of the scan chains, the linear feedforward network generation process that is proposed in Chapter 3.1.1 should be performed again based on the modified scan chain lengths, because the linear feedforward network generation partly depends on the scan chain lengths. Experimental results show that shift times can be reduced 0~30% before scan chain length modification, but 23~68% after scan chain length modification.

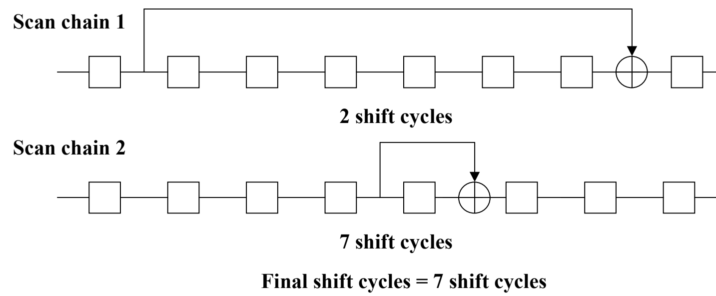


Figure 3.5. Original scan chains

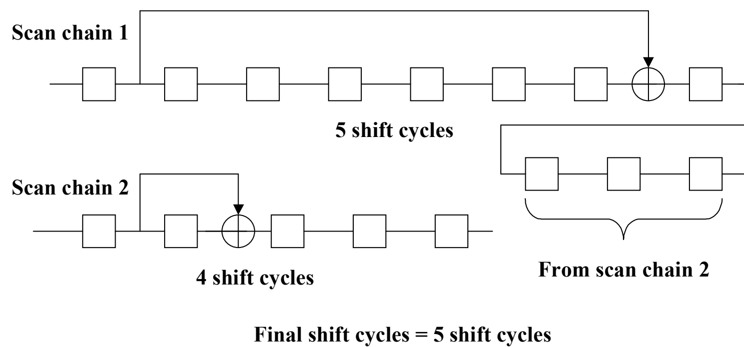


Figure 3.6. Scan chain length modification

3.1.3 Test Response Observation

Because the number of shift cycles is reduced, some test response bits are no longer observable. All of the test response bits on the specified scan path are observable. Among the response bits that are not on the specified scan path, the response bits for which the shortest path to the scan-out is longer than the number of shift cycles are unobservable. In Fig. 3.7, *c1*, *c7* and *c8* are observable because those bits are on the specified scan path. The shortest path from *c2* and *c3* to the scan-out are 2 and 3 respectively, which is less than or equal to the number of shift cycles, 3. So, these response bits are also observable at the scan-out. However, the shortest path from *c4, c5* and *c6* to the scan-out are 4, 5 and 6, respectively, which are larger than the number of

shift cycles thereby causing them to be unobservable. To avoid fault coverage reduction due to this, an additional scan-out is added at the point between the last observable response bit and the first unobservable response bit as shown in Fig 3.8. The added scan-out can be XORed with another scan-out or can go to a multiple input shift register (MISR) by itself. In this way, all of the test response bits can be observable at the scan-out.

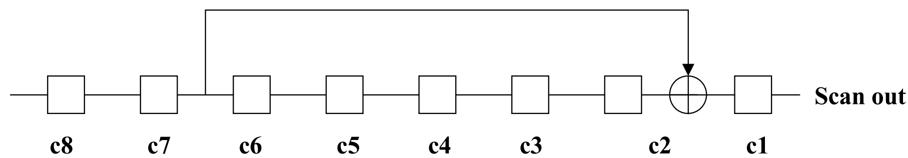


Figure 3.7. 3 unobservable test response bits

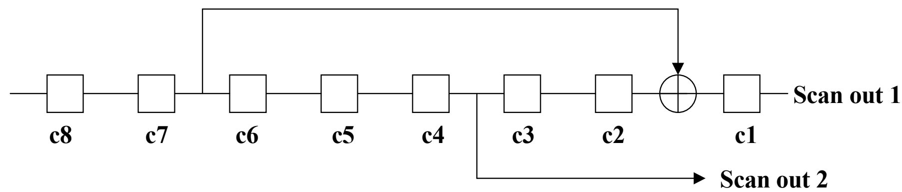


Figure 3.8. Adding another scan-out

Note that the aliasing probability can increase in the proposed scheme because many test response bits are observed in linear combination with other test response bits. However, as seen in Chapter 3.1.1, several previous test response bits are coupled with the current test pattern generation. This means that a previous faulty response may also change the current test pattern thereby causing a different test response from the fault-free response.

3.2 TEST STIMULUS AND RESPONSE GENERATION

In this chapter, "test stimulus" will refer to the actual bits that will be shifted into a scan chain after a feedforward network has been added, and "scan pattern" will refer to the bits that are generated in the scan chain after the test stimulus has been shifted in. Given a scan pattern that needs to be applied to the CUT, this chapter describes how to obtain the corresponding test stimulus that should be shifted into the scan chain having a particular feedforward network. This process is divided into two parts, *reverse transformation* and *forward transformation*.

3.2.1 Test Stimulus Transformation

The original transformation problem that was proposed in [Sinanoglu 04] involved transforming test stimuli into scan patterns within the same number of the shift cycles as the length of the scan chain. The scan patterns were determined solely by the test stimuli and were completely independent of previous test response. In the proposed scheme, a feedforward network is used such that the test stimuli is transformed into scan patterns within a smaller number of shift cycles than the length of the scan chain. Thus, a scan pattern is generated not only by the test stimuli, but also by some of the previous test response bits.

To use the matrix generation solution that was proposed in [Sinanoglu 04], the number of shift cycles must be the same as the length of scan chains. To accomplish this, an additional transformation is required to make the number of shift cycles the same as the length of scan chains by computing some *pseudo test stimulus bits* that actually do not exist. We define this process as *reverse transformation,* and it is illustrated in Fig. 3.9 and Fig. 3.10. The matrix that is used to get the pseudo test stimulus bits (in this example, *R4* and *R5*) is defined as the *reverse transformation matrix* as shown in Fig. 3.9.

In Fig. 3.9 and Fig. 3.10, the scan chain is 5-bits long and the shift time is 3 scan clock cycles because the specified scan path has 3 scan cells. Because the shift cycles are 2-bits shorter than the length of the scan chain, the last two response bits, *r4* and *r5* will still remain in the scan chain after 3 test stimulus bits are scanned in. Therefore, each scan pattern will be determined by *s1, s2, s3, r4* and *r5*. Since the scan pattern depends on *r4* and *r5*, they act as sort of "pseudo test stimulus bits".

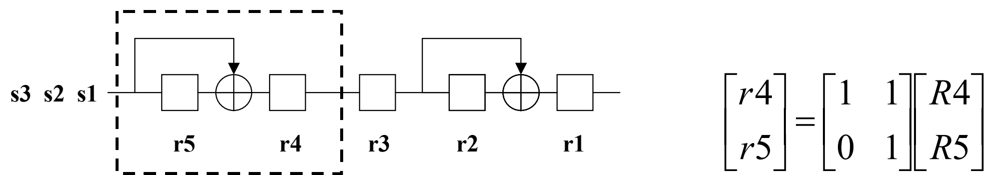


Figure 3.9. Reverse transformation

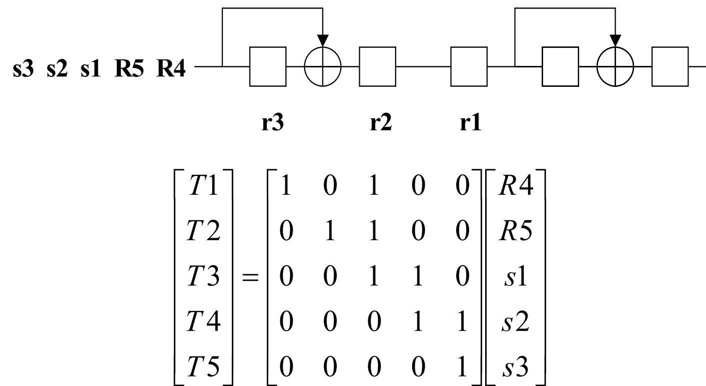


Figure 3.10. After reverse transformation

The reverse transformation matrix can be used to transform them into pseudo test stimulus bits at the input of the scan chain, i.e., *R4* and *R5*. So, we can assume that *R4* and *R5* are inserted during the first two scan clock cycles and the first two scan cells contain *r4* and *r5* respectively after the two scan clock cycles. *R4* and *R5* can be solved

easily through the transformation matrix based on the small segment of the scan chain in the dotted box in Fig. 3.9. Through the reverse transformation, a test stimulus that is 5-bits long can be established as shown in Fig. 3.10. Now the transformation method that has been proposed in [Sinanoglu 04] can be used, and we refer to that transformation as *forward transformation*. Fig. 3.10 also shows the forward transformation matrix of the example. The forward transformation is exactly the same as proposed in [Sinanoglu 04].

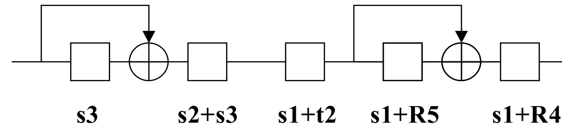


Figure 3.11. After forward transformation

$$\begin{bmatrix} T1 \oplus R4 \\ T4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} s1 \\ s2 \\ s3 \end{bmatrix}$$

Figure 3.12. Final matrix operation

To simplify the matrix equations, the constant values in the forward transformation matrix are extracted and XORed with the left-hand and only the specified bits in a scan pattern are considered in the equation. Assuming *T1* and *T4* in the scan pattern should be specified while the others are don't care bits, the matrix is simplified as shown in Fig. 3.12. The forward transformation matrix after the simplification process is defined as *reduced forward transformation matrix*. The more columns and the fewer rows in the reduced forward transformation matrix, the larger the solution space for the test stimulus that generates a certain scan pattern. Through the experimental results, the number of stimulus bits is 23~68% of the total scan pattern bits, and the number of the specified bits is 5~10% of the total scan pattern bits. This means that on average, the

number of the columns is about 3-to-13 times larger than the number of the rows. In other words, the forward transformation in the proposed scheme has a large solution space. This results in a large number of X's in the test stimulus bits thereby allowing the proposed scheme to be used with other test data compression while still retaining the low power benefits.

3.2.2 Test Response Transformation

The test response that is shifted out from the scan chain is also transformed by the feedforward network. To avoid confusing the matrix in test stimulus transformation and the matrix in test response transformation, we will refer to the matrix in test response transformation as the *response transformation matrix*.

The response transformation matrix generation method presented in [Sinanoglu 04] can be used here. It can be generated from the forward transformation matrix. The only difference is that the shift cycle reduction scheme used here results in the observed test response bits at the scan-out not being affected by any test stimulus bits. This is different from the scheme in [Sinanoglu 04] where the bits are affected by some test stimulus bits. Therefore, after generating the initial response transformation matrix in the way that was proposed in [Sinanoglu 04], the matrix columns that are related to test stimulus bits are simply removed, and the number of the observed test response bits is reduced such that it matches the number of the reduced shift cycles. Fig. 3.13 shows the response transformation of the example in Fig. 3.9. The left matrix in Fig. 3.13 shows the original response transformation matrix and the right one shows the test response transformation matrix in the proposed scheme.

$$\begin{bmatrix} o1 \\ o2 \\ o3 \\ o4 \\ o5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ s1 \\ s2 \end{bmatrix} \Rightarrow \begin{bmatrix} o1 \\ o2 \\ o3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} r1 \\ r2 \\ r3 \\ r4 \\ r5 \end{bmatrix}$$

Figure 3.13. Test response transformation

3.3 APPLICABLE ARCHITECTURE

The proposed scheme can be used anywhere that a standard scan architecture is used. It can be directly connected to a tester, or it can be used in conjunction with other test data compression schemes.

3.3.1 Tester Direct Access

If the proposed scheme is used in place of a conventional scan architecture that is directly connected to a tester, then the proposed scheme can reduce test storage in addition to test power. The number of shift cycles to load each scan vector is reduced. Hence test storage and test time are reduced accordingly. Our experimental results indicate that the number of shift cycles can be reduced by 23~68% for the largest ISCAS89 benchmark circuits. In addition to providing test data compression, the proposed scheme is reducing scan-in, scan-out, and scan clock power in comparison to other test data compression methods that provide similar amount of test data compression.

3.3.2 EDT Architecture

The other nice feature of the proposed scheme is that it can be used in conjunction with other test data compression schemes to achieve a high degree of compression while

reducing power compared with using those test data compression schemes by themselves. This is possible because the proposed scheme retains many of the don't cares in the test set. For example, the proposed scheme could be used in conjunction with the embedded deterministic test (EDT) scheme described in [Rajski 02]. The EDT architecture that has been proposed in [Rajski 02] is composed of a ring generator, a phase shifter, and a response compactor. It dynamically alters the seed of the ring generator every tester cycle to specify the bits that must be specified. Because the test set after the proposed transformation still have many don't cares, the EDT architecture can be used to achieve a high degree of compression while the proposed scheme reduces the number of shift cycles thereby reducing test power including scan-in, scan-out, and scan clock power.

3.4 EXPERIMENTAL RESULTS

Experimental results using the proposed scheme for the largest ISCAS 89 benchmark circuits are shown in Table 3.1. Results are shown for dividing each test cube into different numbers of scan chains. "standard" in the fourth column refers to the case where all the scan chains have same length. As explained in Chapter 3.3.2, the problem of unevenly distributed specified bits exists in the "standard" case. This problem becomes more pronounced the shorter the length of each scan chain is, and in some cases, shift cycle reduction does not occur. "variable" in the fifth column means that the length of each scan chain is modified as described in Chpater 3.3.2 to overcome this problem. Shift cycle reduction was increased significantly as seen in the results in the fifth column. However, note that the shift cycle reduction percentage decreases as the number of scan chains increases. The last column shows the hardware overhead. Note that the experimental results for s9234 show only 23% shift cycles reduction. The reason for this is because the portion of specified bits in the test set is high, more than 20%.

Table 3.1. Results for proposed scheme

Circuit Information			Shift cycles		Hardware Overhead
Circuit	Portion of Specified Bits	Num. of Scan Chains	Reduction		Num. of XOR Gates
			Standard	Variable	
s5378	12.4%	5	13%	50%	35
		10	9%	31%	35
s9234	20.3%	5	4%	24%	23
		10	0%	24%	16
s13207	5%	7	32%	68%	73
		14	12%	54%	67
		28	8%	24%	55
s15850	6.6%	5	30%	51%	61
		10	1%	43%	43
		20	0%	23%	43
s38417	4.9%	10	16%	56%	153
		20	1%	53%	147
		40	0%	40%	153
s38584	6%	10	20%	65%	200
		20	10%	56%	200
		40	8%	38%	227

A comparison with three previous techniques is shown in Table 3.2. The compression ratio in [Chandra 02] is almost similar to that in the proposed scheme. However, the amount of power reduction in [Chandra 02] and the proposed scheme are quite different to each other. Even though the power consumption has been reduced significantly in [Chandra 02] and [Sinanoglu 04], not all of the three power consumption components are reduced. Therefore, reduction of the total power consumption including scan-in, scan-out and scan clocking power will not be as big as the power reduction in the proposed scheme. [Sinanoglu 04] does not compress test data and can reduce only scan-in and scan-out power, while the proposed scheme can compress test data about 50% and can reduce scan clocking power as well as scan-in power and scan-out power. [Sankaralingam 01] proposed a scheme that can reduce scan-in, scan-out and scan clocking power. However, it does not provide any compression and the amount of power reduction is less than the amount of the proposed scheme.

Table 3.2. Results comparing proposed scheme with previous schemes

Circuit	Alternating Run-Length Code [Chandra 02]				XOR Insertion [Sinanoglu 04]			Scan Disabling [Sankaralingam 01]			Proposed Scheme		
	C. (%)	Power Reduc. (%)			C. (%)	Power Reduc. (%)		C. (%)	Power Reduc. (%)		C. (%)	Power Reduc. (%)	
		S.I.	S.O.	S.C.		S.I. S.O.	S.C.		S.I. S.O.	S.C.		S.I. S.O.	S.C.
s5378	-	-	N	N	N	69.3	N	N	-	-	50	50	50
s9234	44	76	N	N	N	68.7	N	N	17	9.8	24	27	24
s13207	80	93	N	N	N	70.4	N	N	41.8	41.2	68	65	68
s15850	65	85	N	N	N	73.4	N	N	25.3	32.2	51	52	51
s38417	60	81	N	N	N	76.3	N	N	24.1	20.9	56	59	56
s38584	61	83	N	N	N	78.8	N	N	-	-	65	62	65

3.5 CONCLUSION

An efficient scheme that can reduce test power while compressing test data is proposed in Chapter 3. The key idea is to reduce the number of shift cycles for each scan chain. In this way, test power, test storage, and test time can be reduced at the same time with relatively small hardware overhead.

CHAPTER 4: *Low Power BIST Based on Scan Partitioning*

A built-in self-test (BIST) scheme is presented which both reduces overhead for detecting random-pattern-resistant (r.p.r.) faults as well as reduces power consumption during test. 3-valued weights are employed to detect the r.p.r. faults. The key idea is to use a new scan partitioning technique and decoding methodology that exploits correlations in the weight sets to greatly reduce the hardware overhead for multiple weight sets and reduce the number of transitions during scan shifting. The proposed scheme is simple to implement and only constrains the partitioning of scan elements into scan chains and not the scan order thereby having minimal impact on routing. Consequently, the proposed scheme can be easily implemented in standard design flows used in industry. Experiments indicate the scheme can achieve 100% fault coverage and 55% to 59% scan power reduction with relatively small hardware overhead.

4.1 PROPOSED SCHEME

The proposed scheme is composed of two parts: random test for the easy-to-detect faults and weighted random test for r.p.r. faults. The random test is performed by LT-RTPG as proposed in [Wang 99] to reduce power consumption. The important aspect of the proposed scheme is the weighted random test since it determines the hardware overhead for the 3-weight decoder and the total power consumption. The key concept in the proposed scheme is to partition the scan cells in a way that entire scan chains can be weighted to a uniform weight instead of individual scan cells each having their own weight. We define two types of scan chains: "*uniform scan*" and "*non-uniform scan*". A *uniform scan* is defined as a scan chain in which all scan cells have the same weight in each weight set which will be referred to as its "*scan weight*". A *non-uniform scan* is defined as a scan chain in which each scan cell has an individual

weight as is the case for conventional weighting. The goal is to maximize the number of the uniform scans so that power consumption and hardware overhead are minimized. Section 4.1.1 describes the 3-valued weight set generation algorithm that was used in the proposed work. The scan chain partitioning algorithm is introduced in Section 4.1.2. Section 4.1.3 explains the 3-weight decoder minimization process.

4.1.1 3-valued weight set generation

In a 3-valued weight scheme, there are only 3 weights, 0, 1 and *r* (random). '*r*' means that corresponding bit position is not weighted, but determined pseudo-randomly. Because there are only three kinds of weights, the hardware overhead can be reduced significantly compared with conventional weighted random test [Pomeranz 92]. The whole process of weight set generation is shown in Fig. 4.1. First of all, a certain number of random patterns are applied to a circuit and the detected faults are dropped. The undetected faults after random pattern test are the r.p.r. faults that are targeted by weighted random test. A 3-valued weight set is then generated based on the deterministic test set as was proposed in [Pomeranz 92]. Consider an example where T = {011x, 00xx, 0x11, x1x0} is a set of deterministic test cubes. A weight set is generated so that the largest number of the undetected faults can be detected. In the other words, the weight set that can cover the largest number of the deterministic test cubes is generated. In this example, '0*r*1*r*' is generated based on the deterministic test set. The weight set, '0*r*1*r*' means that the first and the third input bit position are weighted to 0 and 1 respectively and the second and the fourth input bit position are generated pseudo-randomly. Note that the number of the bit positions that are weighted to '*r*', that is, determined pseudo-randomly, is limited by the number of patterns applied with one weight set. The more patterns that are applied per weight set, the more '*r*' positions there can be since the probability of covering each test cube is increased with

more patterns. In this small example, if the maximum number of '*r*' positions was limited to 1 then any single weight set could not cover all of the deterministic test cubes (two '*r*' positions are required for that). Each weight set '0*r*11', '011*r*' or '0*r*10' can cover only three deterministic test cubes (not all four). After a weight set is generated, fault simulation with the weight set is performed and the detected faults are dropped. The final step is to check if all faults are detected or not. If all of the r.p.r. faults are detected, the 3-valued weight set generation is completed. Otherwise, another weight set is generated until the set of the undetected faults is empty.

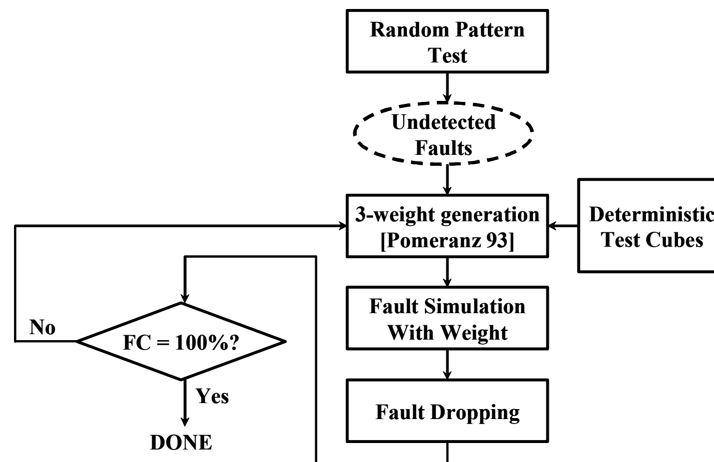


Figure 4.1. 3-valued weight set generation

4.1.2 Scan chain partitioning

In conventional 3-valued weighted random test, each individual scan cell is weighted to 1, 0, or *r*. This can result in a large hardware overhead for the 3-weight decoder. In the proposed work, careful partitioning of the scan chains is done so that all of scan cells in many of the scan chains are uniformly weighted to the same weight. This reduces the hardware overhead for the 3-weight decoder significantly and also reduces the power consumption during scan shifting. The actual order of the scan cells

within each partition doesn't matter and hence they can be ordered in the conventional manner to minimize routing.

The proposed scan partitioning technique is formulated as a minimum clique covering problem. Note that in the remainder of the paper, a "weight cube" is defined as the set of weight assignments for a scan cell where each bit position in the weight cube corresponds to the weight assignment in a particular weight set (0, 1, R, or x). An 'x' is a don't care where the weight assignment can be anything with no impact on fault coverage. This is illustrated in Fig. 4.2 where there are 9 scan cells and 4 weight sets. The weight cube for scan cell *s2* is *x0xx* as the only requirement for *s2* is that it be weighted to 0 in weight set 2. The proposed scan partitioning procedure has four steps which are described below:

<u>Step 1: Prune and group scan cells</u>

The first step involves pruning and grouping the scan cells to simplify the graph constructed in Step 2. All "don't care cells" which are scan cells that have no specified weight in any weight set are removed from the set of scan cells since they need not be considered. Then the scan cells are grouped together where if all the specified bits of any weight cube for a scan cell is a subset of the specified bits of a weight cube for another scan cell, those two scan cells are put into the same group. Step 1 is illustrated in Fig. 4.2 where there are 4 weight sets and the number of scan cells is 9. In this example, *s1* and *s4* are don't care cells which are removed from the set of scan cells. Also, the specified bit in the weight cube for *s7*, xxx1, is a subset the specified bits in the weight cube for *s3*, 1xx1, and thus *s3* and *s7* are placed into the same group, *g1*. The end result of Step 1 is a set of scan cell groups (as shown on the right of Fig. 4.2).

	s1	s2	s3	s4	s5	s6	s7	s8	s9
w1	x	x	1	x	0	1	x	1	0
w2	x	0	x	x	1	x	x	R	x
w3	x	x	x	x	x	1	x	R	1
w4	x	x	1	x	R	x	1	x	x

Weight cube for s2

Group table

G	Weight cube	Scan cells
g1	1xx1	s3 s7
g2	x0xx	s2
g3	1x1x	s6
g4	0x1x	s9
g5	01xR	s5
g6	1RRx	s8

Figure 4.2. Step 1: Initialization

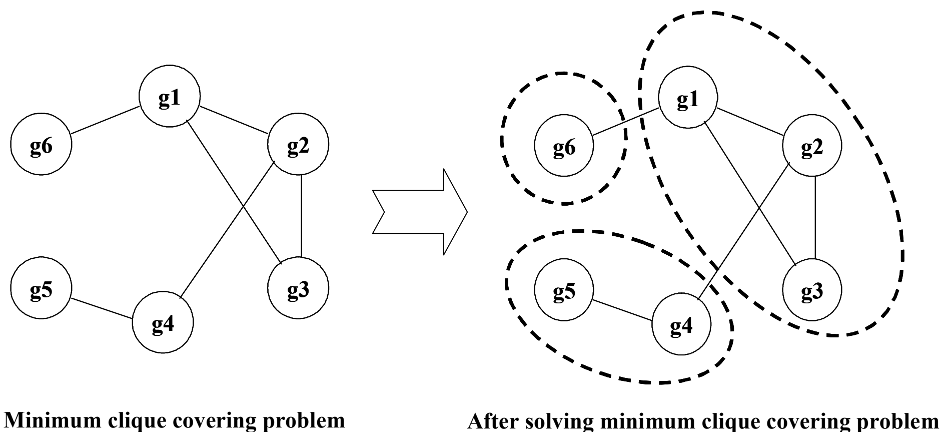


Figure 4.3. Step 2, 3: Minimum clique covering

Step 2: Construct compatibility graph

The second step involves constructing a compatibility graph in which each vertex in the graph corresponds to a group formed in Step 1 and an edge is placed between two groups if they are compatible. Two groups are said to be compatible if they do not conflict in any specified bit position (1,0, and R). In other words, for each bit position in which both groups are specified, they must match. In the example in Fig. 4.2, the

weight cube for *g1*, 1xx1, is compatible with the weight cube for *g6*, 1rrx. Therefore, an edge exists between *g1* and *g6* in Figure 4.3.

Step 3: Find minimum clique cover

A clique is a complete sub-graph and corresponds to a set of scan cells that can be used to construct a uniform scan chain since their weight assignment in all weight sets are compatible. In the example in Fig. 4.3, *g1*, *g2* and *g3* compose of a complete sub-graph. Therefore, *g1*, *g2* and *g3* can be in one clique. The minimum clique cover for the compatibility graph corresponds to a partitioning of the scan cells that allows the construction of a maximal set of uniform scan chains. The right part of Figure 4.3 shows the solution of the minimum clique covering problem. In this example, there are 3 cliques that cover all of the vertices in the graph. Clique 1 includes *g1*, *g2* and *g3*, that is, four scan cells, *s2*, *s3*, *s6* and *s7*, clique 2 includes *g4* and *g5*, that is, two scan cells, *s5* and *s9*, and clique 3 includes *g6*, that is, one scan cell, *s8*. The scan cells in each clique can be weighted by the same weight value in each weight set.

Note that there can be more than one minimum clique cover. Note also that finding a minimum clique cover is an NP-complete problem, however, there are good heuristic algorithms for it (see [Cormen 97] for more details). A greedy algorithm was used in our experiments.

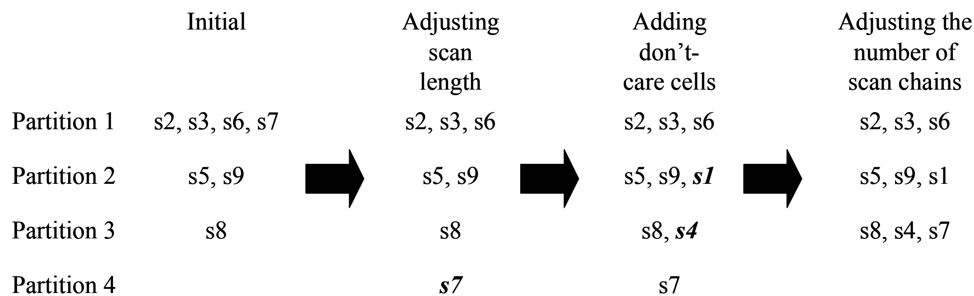


Figure 4.4. Step 4: Adjusting scan length and the number of scan chains

Step 4: Form scan chains

The final step is to form the scan chains from the partitions obtained in the minimum clique cover. Given a certain fixed scan length, the partitions in which the number of scan cells is larger than the scan length are divided. Ideally, it would be best if each scan chain was formed using scan cells from a single partition so that it can be a uniform scan. However, it is unlikely that the number of scan cells in each partition will be a multiple of the scan length. However, there are two degrees of freedom that can be used. The first is that some scan cells may be compatible with more than one partition and thus can be moved between partitions to help balance things out. The second degree of freedom is that the don't care cells which were pruned out in step 1 can now be added back in to any of the partitions (since they are don't cares) to help balance things out. Any scan chains that cannot be made equal to the scan length using these two degrees of freedom must be merged together to form non-uniform scan chains.

Consider the example in Fig. 4.4 where the scan length is 3, the number of scan cells in partition 1 is 4, which is larger than the scan length. Therefore, one of 4 scan cells in the partition 1 should be moved to another partition if possible without destroying the property of a clique. Otherwise, a new partition is added. In Fig. 4.4, partition 4 is added, and *s7* is movedto the partition 4. After this, the number of scan cells that each partition has is less than or equal to the scan length. Any scan chain that has 3 scan cells at this point (which is the same as the scan length) is a uniform scan. Therefore, partition 1 is a uniform scan. Next, the don't care cells are added to make as many additional uniform scans as possible. In the example in Fig. 4.4, *s1* is added to partition 2, and *s4* is added to partition 3, and thus partition 2 is made a uniform scan. Finally, partition 4 is merged into partition 3 to form the last scan chain. Because two different partitions are merged, partition 3 is not a clique anymore and thus forms a non-uniform

scan. The end result is that there are two uniform scans, partition 1 and partition 2, and one non-uniform scan, partition 3. All the scan cells in each of the uniform scans are weighted by the same scan weight in each weight set. This will reduce power consumption and also minimize the hardware overhead for the 3-weight decoder. The non-uniform scan is weighted by conventional weighting where each scan cell has its own weight.

4.1.3 3-weight decoder minimization

	Scan weight 1	Scan weight 2	Scan weight 3	Scan weight 4	Scan weight 5
w1	1	X	X	1	0
w2	1	0	X	X	0
w3	X	1	0	0	X
w4	X	0	1	1	X
w5	X	1	X	0	1

Figure 4.5. Decoder minimization

The decoding logic for the 3-valued weight sets can be minimized further by using compatibility of scan weights. If two scan chains have compatible scan weights across all weight sets, they can share the same decoding logic and weight logic. One way that this commonly happens is when two scan chains are formed from the same original partition. Figure 4.5 shows an example of decoder minimization. In this example, there are 5 weight sets and 5 uniform scans, that is, 5 scan weights per weight set. *Scan weight 1*, *scan weight 2,* and *scan weight 4* are mutually compatible because there is no conflict in any specified bit position. Therefore, *scan weight 1*, *scan weight 2,* and *scan weight 4* can be generated by the same decoding logic. In the same way, *scan weight 2* and *scan weight 5* can be generated by the same decoding logic. Therefore, only two separate decoding logic and weight logic (one AND gate and one OR

gate) are required to handle all 5 scan chains, thereby reducing hardware overhead for the decoding logic.

4.2 ARCHITECTURE

In this section, two different architectures are presented. One is the conventional fixed-length scan architecture, and the other is a variable-length scan architecture that allows better optimization for the proposed scheme as will be described. In the fixed-length scan architecture each scan chain has the same length, whereas in the variable-length scan architecture, the scan chains have variable lengths.

4.2.1 Fixed-length scan architecture

Figure 4.6 shows a fixed-length scan architecture. There are 3 uniform scans and 1 non-uniform scan and each scan chain has the same scan length. The uniform scans are weighted by a scan weight decoder. Note that the input of the scan weight decoder is only the weight counter information since all the scan cells in a uniform scan are weighted by the same weight and thus the scan weight decoder does not require bit counter information. The non-uniform scan is weighted by conventional 3-valued weighted random testing.

Note that hardware overhead is primarily determined by the size of the 3-weight decoder for non-uniform scans. The size of the 3-weight decoder for non-uniform scans is much larger than the size of the scan weight decoder for uniform scans. The conventional 3-weight decoder requires bit counter information as well as the weight counter information, thereby making the decoding logic much more complex. Therefore, the main objective in scan partitioning is to minimize the number of non-uniform scans.

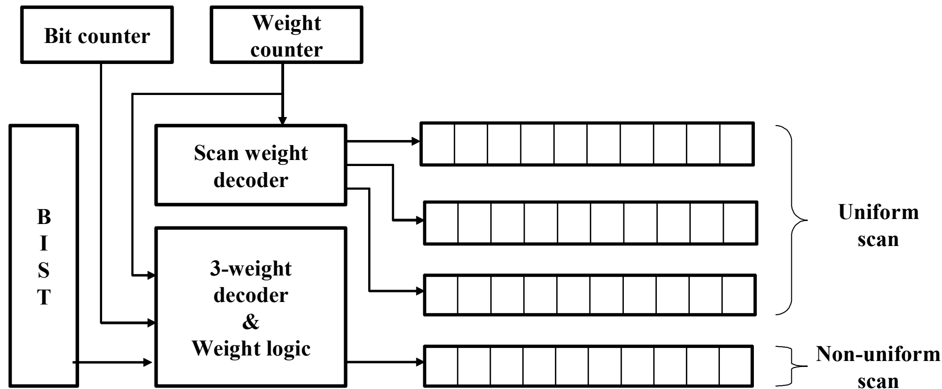


Figure 4.6. Architecture

4.2.2 Variable-length scan architecture

A variable-length scan architecture is proposed to reduce the number and length of non-uniform scans since the size of the 3-weight decoder for non-uniform scans depends on this. The final step, Step 4, in the scan partitioning technique proposed in Sec. 4.1.2 can be modified for variable-length scan chains to reduce the number of non-uniform scans and the length of non-uniform scans. The drawback of a variable-length scan architecture is that the test time increases because the number of clock cycles required to shift in each pattern is determined by the longest scan chain, and the length of the longest scan chain increases in the variable-length scan architecture. However, typically in BIST, test time is not as an important issue as hardware overhead and power consumption.

Step 4 in Sec. 4.1.2 is modified for variable-length scan in the following way. Instead of a constraint on the length of each scan chain, there is a constraint on the number of scan chains. If the number of partitions is less than the number of scan chains, then the largest partitions can be divided to minimize the maximum scan length. If the number of partitions is larger than the number of scan chains, then the smallest

partitions can be merged to create a non-uniform scan. Once the number of partitions is the same as the number of scan chains, then the don't care cells are added to make the lengths of the uniform scans as balanced as possible. In the example in Fig. 4.7, the don't care cells, *s1* and *s4,* are added to *partition 2*.

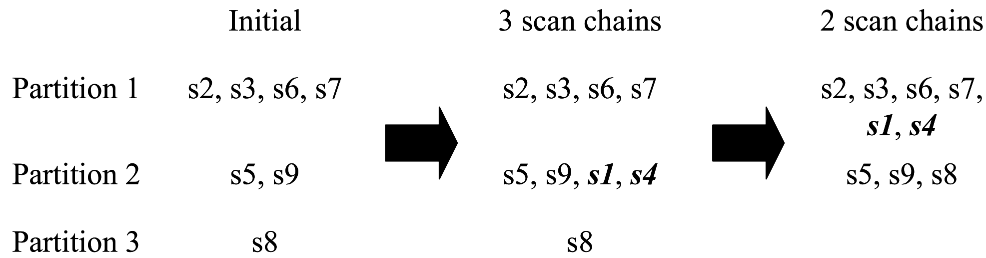


Figure 4.7. Step 4 : Adjustment in variable-length scan architecture

4.3 EXPERIMENTAL RESULTS

The proposed scheme has been applied to the 5 largest ISCAS89 circuits [Brglez 89]. Table 4.1 shows the number of random patterns generated by LT-RTPG and the size of LFSR used in the proposed work. The number of weight sets required to achieve 100% fault coverage for both the proposed method and the method described in [Wang 02] are also shown in Table 4.1. Note that they are similar.

Table 4.1. Results using LT-RTPG

Circuit	Number of Patterns	Serial-fixing WRBIST [Wang 02]	Proposed	
		Num. Weight Sets	LFSR size	Num. Weight Sets
s9234	132072	14	32	15
s13207	65536	2	32	2
s15850	131072	12	32	8
s38417	131072	23	64	19
s38584	65536	12	64	3

Table 4.2. Results for proposed method in fixed-length scan architecture

Circuit	Num W.S.	Num Scan	Fixed-length scan architecture			
			Num U.S.	Num N.S.	Area Overhead	Trans. Reduc.
s9234	15	16	11	5	459	68.7%
		32	22	10	437.5	68.7%
s13207	2	16	14	2	84.5	87.5%
		32	29	3	67.5	90.6%
s15850	8	16	13	3	241.5	81.2%
		32	26	6	202	81.2%
s38417	19	32	23	9	1175.5	71.8%
		50	38	12	981.5	76.0%
s38584	3	32	30	2	197.5	93.7%
		50	47	3	149	94.0%

Table 4.3. Results for proposed method in variable-length scan architecture

Circuit	Num W.S.	Num Scan	Variable-length scan architecture					
			Num. U.S.	Num. N.S.	Area Overhead	Overhead Reduc.	Trans. Reduc.	Test Time
s9234	15	16	13	3	411.5	10.3%	82.2%	53.8%
		32	27	5	377.5	13.6%	84.3%	67.5%
s13207	2	16	15	1	66	21.8%	93.7%	29.5%
		32	31	1	47.5	29.6%	96.8%	36.3%
s15850	8	16	14	2	201.5	16.5%	87.1%	35.8%
		32	28	4	179.5	11.1%	87.1%	45.0%
s38417	19	32	26	6	1020.5	13.1%	80.3%	23.0%
		50	42	8	884	9.9%	79.4%	29.4%
s38584	3	32	30	2	143	27.5%	93.3%	41.3%
		50	48	2	128.5	13.7%	95.0%	40.0%

Table 4.2 and Table 4.3 show the hardware overhead and power consumption resulting from the proposed work. The third column shows the number of scan chains. The fourth and the fifth column, and the eighth and ninth column show the number of uniform scans (U.S.) and the number of non-uniform scans (N.S.) respectively. Note that in all of the circuits, the number of uniform scans is much larger than the number of non-uniform scans, and that means the proposed scan partitioning technique can reduce power consumption and hardware overhead effectively. The hardware overhead shown in the sixth and the tenth column is expressed as gate equivalents which were computed

as 0.5n for an n-input NAND/NOR gate and 0.5 for an inverter. The 3-weight decoding logic was obtained by running SIS [Sentovich 92] for a 2-level circuit implementation. A comparison between the hardware overhead in a fixed-length scan architecture and the hardware overhead in a variable-length scan architecture is shown in the eleventh column. Note that the hardware overhead in the variable-length scan architecture is 9.9% ~ 29.6% smaller than the hardware overhead in fixed-length scan architecture. Reduction in the number of transitions in the scan chains are shown in the seventh and twelfth column. The last column shows the drawback of variable-length scan architecture, test time increase. The test time in variable-length scan architecture increased 23.0% ~ 67.5% compared with the test time in the fixed-length scan architecture.

In Table 4.4, the experimental results in [Wang 02] are compared with the results in the proposed work. In all circuits except for one, *s9234*, both the hardware overhead and the amount of power consumed in the proposed work is smaller than the hardware overhead and the power consumed in [Wang 02]. Note that in the proposed method, no constraints are placed on the scan order, thus the routing complexity for the proposed method is much less. The number of transitions in the scan chains is calculated in the manner described in [Wang 02]. 512 consecutive patterns generated by LT-RTPG and the first 128 patterns generated by each weight set are used to count the number of transitions.

The experimental results described in this section show that the proposed scheme can be an effective solution for two critical problems in BIST, power consumption and overhead to detect r.p.r. faults. The proposed scheme can reduce the hardware overhead to detect r.p.r. faults and the amount of power consumed during test.

Table 4.4. Comparison with serial-fixing WRBIST [Wang 02]

Circuit Name	Serial-fixing WRBIST [Wang 02]			Proposed				
				Fixed-Length			Variable-Length	
	Num W.S.	Overhead (GE)	Trans. Reduc.	Num W.S.	Overhead (GE)	Trans. Reduc.	Overhead (GE)	Trans. Reduc.
s9234	14	213.5	36%	15	437.5	53.7%	377.5	56.8%
s13207	2	63	27%	2	67.5	58.1%	47.5	59.3%
s15850	12	207.5	37%	8	202	56.4%	179.5	57.4%
s38417	23	1031.5	37%	19	981.5	55.2%	914	55.8%
s38594	12	671	45%	3	149	58.8%	128.5	59.0%

4.4 CONCLUSIONS

In this chapter, a technique to reduce power consumption and hardware overhead in BIST is proposed. It uses scan partitioning, but does not place any constraints on scan order thereby allowing the scan order to be selected to minimize routing complexity. The proposed method is simple and can be easily incorporated in the standard design flows used in industry.

Chapter 5: *Combining Linear and Non-Linear Test Vector Compression using Correlation-Based Rectangular Encoding*

Linear decompressors, which are widely used in commercial test data compression tools, are generally not able to exploit correlations in the test data, and thus the amount of compression that can be achieved with a linear decompressor is directly limited by the number of specified bits in the test data. A technique is presented here for improving the compression achieved with any linear decompressor by adding a small non-linear decoder that exploits bit-wise and pattern-wise correlation present in test vectors. The proposed non-linear decoder has a regular and compact structure and allows continuous-flow decompression. It has a very important feature which is that its design does not depend on the test data. This simplifies the design flow and allows the decoder to be reused when testing multiple cores on a chip. Experimental results show that combining a linear decompressor with the small non-linear decoder proposed here significantly improves the overall compression.

5.1 Rectangular Encoding

One well-known characteristic of test data is that certain bit positions in test cubes tend to be correlated across many patterns. This arises from the fact that many faults in the circuit require similar input assignments to detect. This characteristic of test data is exploited by weighted pattern testing methods. Each weight set targets a subset of the test cubes that have highly correlated values in a subset of the bit positions. If one represents the test set as a test matrix in which each row corresponds to a test cube and each column corresponds to a bit position, then the correlations tend to exist in rectangles in this matrix. The idea with rectangular encoding is to encode these rectangles with a small number of specified bits and then have a simple decoder that decodes them. Since

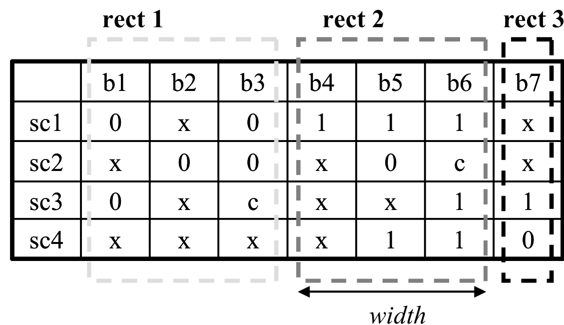
the rectangles have a regular structure, the decoder design is simple and independent of the test data.

5.1.1 Overview

The first step in rectangular encoding is to partition the test cubes into clusters such that pattern-wise correlation within a cluster is maximized. This is done using a clustering algorithm that will be described in Sec. 5.1.4. Each cluster is then encoded as one unit. If there are n scan chains, then each *scan slice* consists of the n bits that are shifted into the n scan chains in a clock cycle. A test cube with m bits consists of m/n scan slices. In rectangular encoding the scan slices for a test cube are partitioned into a sequence of variable-length rectangles. All the test cubes within each test cube cluster are partitioned identically. So in effect, the entire test matrix is partitioned into rectangles where the height of each rectangle is determined by the number of test cubes in the test cube cluster it belongs to, and the width is determined by the scan slice partitioning for the test cube cluster it belongs to. A heuristic procedure will be described in Sec. 5.1.2 for partitioning the scan slices to maximize compression.

Within each rectangle, the largest set of scan chains that has compatible values is identified. This set of scan chains must have either a *1(0)* or *X* for every scan slice across the width of the rectangle and every test cube across the height of the rectangle. A *chain select mask* is then defined for the rectangle which identifies which scan chains should be loaded from the linear decompressor and which scan chains should be filled with a specified fill value (*1* or *0*). So the information that is needed to decode each rectangle in a particular test cube cluster consists of three things, the width of the rectangle, the *chain select mask*, and the fill value. This is illustrated in the small example shown in Figs. 5.1 and 5.2.

In Fig. 5.1, there are 7 scan slices (columns) and 4 scan chains (rows). The bit compatibility for a test cube cluster is shown where a value of *X* indicates all the test cubes in the cluster have *X* for that scan cell, a value of *1(0)* indicates that they all have either *1(0)* or *X* for that scan cell, and a value of *C* indicates a conflict where both *1* and *0* are present. The scan slices are partitioned into 3 rectangles.



	b1	b2	b3	b4	b5	b6	b7
sc1	0	x	0	1	1	1	x
sc2	x	0	0	x	0	c	x
sc3	0	x	c	x	x	1	1
sc4	x	x	x	x	1	1	0

Figure 5.1. Example of rectangles

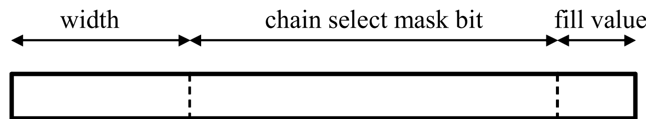


Figure 5.2. Data format

rect 1	1	1	1	1	0	x	0
rect 2	1	1	1	0	1	1	1
rect 3	0	1	x	x	x	x	x

Figure 5.3. Control data for three rectangles

Figure 5.2 shows the format for rectangular control data, and Fig. 5.3 shows the specific values for the three rectangles in Fig. 5.1. The width of the rectangle in terms of scan slices is encoded as a binary number. The maximum width of a rectangle is a user-defined parameter and determines the number of bits allocated for specifying the width (experimental data for different maximum widths is discussed in Sec. 5.3). The

chain select mask contains one bit for each scan chain to indicate if the scan chain should be loaded from the linear decompressor or loaded with the fill value. The last bit indicates the fill value (either 1 or 0). In the second rectangle, *rect2*, all the bits in *sc1*, *sc3*, and *sc4* can be filled with 1 (this is not the case for *sc2* because there is a conflict in scan slice 6). As can be seen in Fig. 5.3, the *chain select mask* in this case would be 1011 which would load all the scan chains except *sc2* with the fill value of *1* (*sc2* would be loaded from the linear decompressor). In *rect1*, all the scan chains except *sc3* can be loaded with the fill value. Moreover, in this case *sc4* has only *X* values and thus it doesn't matter whether it is loaded with the fill value or from the linear decompressor, and therefore the *chain select mask* is *110X* in this case. The last rectangle, *rect3*, is only one scan slice wide. For very narrow rectangles, it is generally more efficient to simply load them from the linear decompressor and not bother specifying a *chain select mask* and fill value. For this reason, if the width of a rectangle is below some user-defined minimum threshold, the *chain select mask* is simply ignored and all the scan chains are filled with from linear decompressor. The advantage of this is that as can be seen in Fig. 5.2 and Fig. 5.3, the *chain select mask* and fill value are simply don't cares for *rect3*.

5.1.2 Partitioning Scan Slices into Rectangles

The reduction in the number of specified bits that the linear decompressor has to produce (and hence the amount of compression achieved) for each rectangle depends on the number of control bits that need to be specified for decoding the rectangle versus the number of specified bits in the test cubes that get covered with the fill value (and hence do not need to be generated by the linear decompressor). The goal in partitioning the scan slices for a test cube cluster into rectangles is to achieve the greatest overall

reduction in the number of specified bits. A greedy heuristic procedure for this is described in this subsection.

The first step is to generate a *compatibility cube* for the test cube cluster. This is illustrated in Fig. 5.1 and has been explained earlier. From the compatibility cube, the rectangle which provides the largest reduction in specified bits is identified. This is done by considering each scan slice as a starting point for a rectangle and considering all possible rectangle widths (up to the user-defined maximum rectangle width) from that starting point. Once the best rectangle is identified, it is marked as selected and the procedure repeats taking into consideration that rectangles cannot overlap. Rectangles continue to be selected in a greedy manner until all scan slices have been included in a rectangle.

5.1.3 Reducing size of chain select mask

The amount of control data that needs to be specified for decoding the rectangles is typically dominated by the bits for the *chain select mask*. One way to reduce this data is that instead of using one bit in the *chain select mask* for each scan chain, one bit can be used per k scan chains. This reduces some of the flexibility since now all k scan chains controlled by the same bit in the *chain select mask* need to be compatible in order to use the fill value. However, it generally provides greater compression since he control data is significantly reduced. Experimental results are shown in Sec. 5.3 for different values of k.

5.1.4 Forming Test Cube Clusters

In rectangular encoding, the test cubes are partitioned into clusters and each cluster is then divided into rectangles. This kind of clustering algorithm has been introduced in previous papers [Alleyne 94]. We use a different benefit function to

maximize correlation in a cluster and also minimize the number of clusters. In order to maximize the compression achieved for each rectangle, it is important that the test cubes in each cluster have many bit positions with compatible values. As more test cubes are added to a cluster, the height of each rectangle increases. This has the benefit of amortizing the control bits required for decoding each rectangle over more test cubes, but there is a tradeoff as more bit positions are likely to have conflicts (thereby increasing the number of *C*'s in the compatibility cube) and thus reducing the effectiveness of each rectangle. A greedy clustering procedure that takes this tradeoff into consideration is described here.

One test cube is used as a seed for the cluster. All other test cubes are then considered as candidates to add to the cluster. The heuristic that is used to measure the optimality of a cluster is the total number of specified bits that are present in each compatible bit position of the cluster. This value forms a benefit function for the cluster. It is computed by considering each compatible bit position and adding up the number of test cubes that have a specified value in that bit position. Consider the test cubes in Fig. 5.4. A cluster consisting of test cubes *t1*, *t2*, and *t3*, is compatible in the first 3 bit positions and the total number of specified bits in those 3 bit positions is 7 (the *X*'s are not counted). The change in this benefit function is computed for adding each candidate test cube. The one that gives the greatest improvement in the benefit function is added to the cluster. This continues until a point is reached where no positive improvement in the benefit function can be obtained by adding another test cube to the cluster. For example, in Fig. 5.4, if test cube *t4* was added to the cluster, then the benefit function would actually decrease because bit position 3 would no longer be compatible. Since the final cluster is very dependent on the initial seed, all test cubes

are used as seeds and the best resulting cluster is selected. This process is repeated iteratively for the remaining test cubes until all test cubes are members of a cluster.

Note that while a greedy clustering procedure is described here, any clustering procedure in the literature can be used to maximize the benefit function defined above.

t1	x	0	0	0
t2	x	0	0	1
t3	0	x	0	x
t4	0	x	1	x

Figure 5.4. Example of clustering

5.2 RECTANGULAR DECODER

Decoding the rectangles is done with a sequential non-linear decoder that is placed between a linear decompressor and the scan chains. A block diagram for the rectangular decoder is shown in Fig. 5.5. It consists of a controller which is a small finite state machine, a RAM that stores the rectangular control data, a RAM address pointer that points to the control data for the next rectangle, a width counter, and a rectangular control register that stores the control data for the current rectangle (rectangle width, *chain select mask*, and fill value). Note that a RAM that is present for functional purposes can be utilized in the rectangular decoder (it is not necessary to add an extra RAM). A MUX is placed in front of each scan chain. The select line to the MUX is the bit in the *chain select mask* that corresponds to that scan chain. Note that if $k>1$, then one bit in the *chain select mask* will fan out to k MUXes. Depending on the corresponding value in the *chain select mask*, each scan chain will either be loaded with the fill value or be loaded from the linear decompressor. Note that if the rectangle width is below the user-defined threshold, then the scan chain is loaded from the linear

decompressor regardless of the value of the chain select mask. This is implemented by adding another MUX whose select line comes from a less-than comparator that checks the value of rectangle width. Note that this is not shown in Fig. 5.5 for sake of readability.

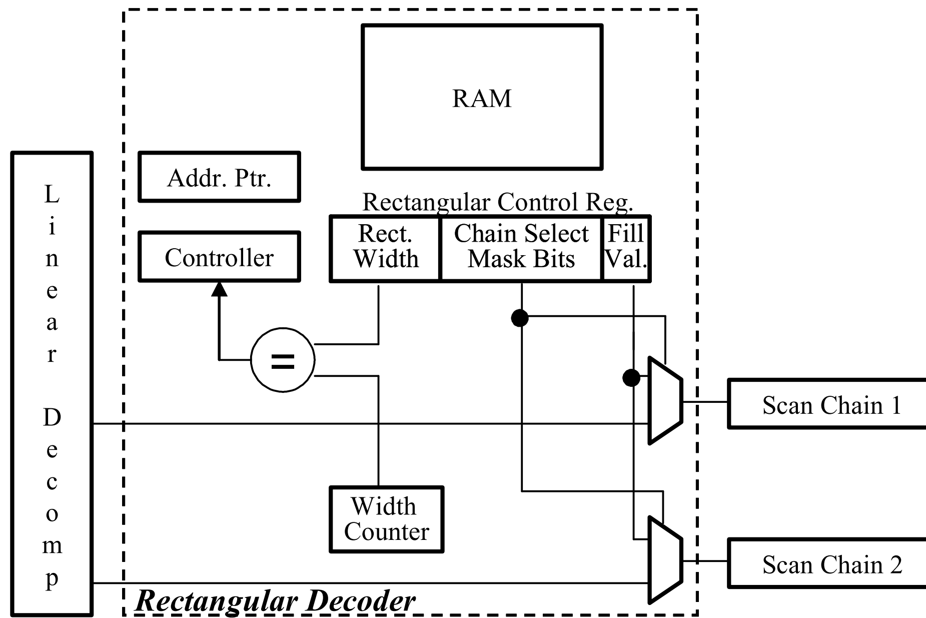


Figure 5.5. Block diagram for rectangular decoder

The RAM holding the rectangular control data can be loaded from the tester either all at the beginning of the test session or incrementally during the test session. If it is all loaded at the beginning, the entire rectangular control data is transferred either directly from the tester or through the linear decompressor to the RAM. In this case, the RAM must be large enough to store all the rectangular control data. The other option is to incrementally load the rectangular control data each time a new test cube cluster is started. In this case, only the rectangular control data for one cluster needs to be stored on-chip at a time. Thus, the required RAM size would only depend on the maximum number of rectangles in any cluster.

The test set is ordered so that all the test cubes in a cluster come in succession. An extra clock cycle is added at the start of each test cube in which the linear decompressor generates one specified bit to tell the controller whether or not this is the start of a new test cube cluster. If it is not the start of a new cluster, then the same rectangular data that was used for the previous test cube is used for this one (the RAM pointer is simply reset back to the first rectangle for this cluster). If it is the start of a new cluster then there are two cases. If the rectangular control data is to be loaded incrementally, it is done at this point (only the data needed for this cluster). If the rectangular control data was all loaded into the RAM at the start, then the RAM address pointer is incremented to point to the start of the rectangular control data for this new cluster.

After this, each rectangle is decoded one at a time as the test cube is shifted into the scan chains. For each rectangle, the controller loads the rectangular control data from the RAM into the rectangular control register, and the width counter is reset to 0. As each scan slice is loaded into the scan chains, the width counter is incremented. When it becomes equal to the rectangle width, then the next rectangle is loaded from the RAM into the rectangular control register and the RAM pointer is incremented. This process repeats until the entire test cube has been shifted in.

As can be seen, the rectangular decoder design is simple, compact, and very regular. A very nice feature is that it does not depend on the actual test data. It can be designed so that it is capable of decoding any set of rectangles. This simplifies the design flow since there is no need to have the test data when implementing the decoder.

5.3 Experimental Results

Experiments were performed on the four largest ISCAS89 benchmark circuits. In Table 5.1, the number of test cubes and the number of specified bits in deterministic

test sets are shown in the second and third column. The fourth column shows the number of clusters obtained with the pattern clustering algorithm described in Sec.5.1.4. Results for the proposed method where generated for three different numbers of scan chains. In the sixth column, the number of total rectangles across all clusters is shown. The next three columns show the size of a rectangle control data required for each rectangle. '*W*' is the number bits used for the rectangle width. '*C*' is the number of *chain select mask* bits, and '*T*' is the number of total bits per rectangle (which is equal to $W+C$ plus 1 for the fill value). Note that in all cases, $k=2$ (i.e., each *chain select mask* bit controlled two scan chains), and thus C is equal to the number of scan chains divided by 2 in all cases.

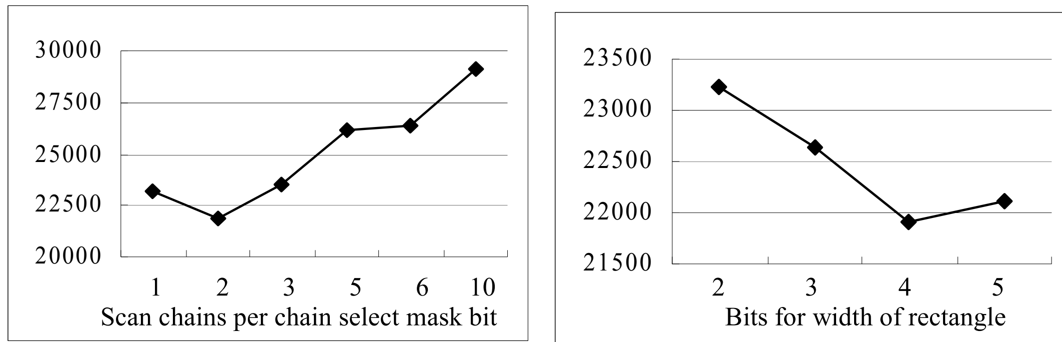


Figure 5.6. Specified bits vs. width and k value for *s38417*

Note that the graph on the left side of Fig. 4.6 shows the total number of specified bits with different k values for *s38417*. As can be seen, the best result occurs for $k=2$. The graph on the right side of Fig. 6 shows the total number of specified bits using different numbers of bits for specifying the rectangle width (i.e., using different maximum rectangle widths) for *s38417*. The best result is observed when using 4 bits

for the width. In all of the circuits except for *s38584*, the best result is observed using 4 bits while for *s38584* it is observed for 3 bits.

Table 5.1. Results for proposed rectangular encoding scheme

Circuit	Test Cube	Orig. Spec. Bits	Cl.	Scan Chain	Num. Rect.	Rect. Control			Data Spec. Bits	Cont. Spec. Bits	Total Spec. Bits	Red. (%)	RAM (bits)
						W	C	T					
s13207	266	9389	8	10	86	4	5	10	5774	1126	6900	26.5	130
				20	75	4	10	15	5665	1391	7056	24.8	150
				30	54	4	15	20	6217	1346	7563	19.5	180
s15850	269	10944	9	10	75	4	5	10	5707	1019	6726	38.5	100
				20	56	4	10	15	6190	1109	7335	33.0	135
				30	49	4	15	20	6602	1249	7851	28.3.	160
s38417	376	30669	7	20	107	4	10	15	19880	1981	21861	28.7	255
				30	71	4	15	20	19306	1796	21102	31.2	280
				40	64	4	20	25	19735	1976	21711	29.2	250
s38584	296	26185	8	20	135	3	10	15	19331	2321	21652	17.3	300
				30	108	3	15	20	19693	2348	21941	16.2	320
				40	101	3	20	25	19508	2720	22228	15.1	375

The total number of specified bits that the linear decompressor has to produce when using the proposed non-linear decoder is shown in the tenth column (this includes all the specified rectangular control data as well as the extra bit for each test cube to indicate if it is the start of a new cluster. The percentage reduction in specified bits is shown in the next column. As can be seen, the number of specified bits that the linear decompressor has to produce is significantly reduced. This reduction in the specified bits is a very powerful result because it means that in most cases, up to an additional 30% or more compression can be achieved *on top of* the best possible compression that is currently available for any linear decompression scheme. If the test data bandwidth is held constant, this translates to an equivalent reduction in test time. As the number of scan chains increases, the number of specified bits required for the proposed scheme increases slightly, but not much. The last column shows the size (in number of bits)

of the RAM required to store the rectangular control data if it is incrementally loaded. Note that it is very small.

Experiments results for combining the proposed scheme with an actual linear decompressor are shown in Table 5.2. The number of test patterns in the test set and the number of specified bits that need to be generated using the linear decompressor in [Krishna 01] alone and using it with the proposed scheme are shown in Table 5.2. As can be seen, the reduction in test storage is very closely related to the reduction in specified bits. Note that the proposed scheme can be used with any linear decompressor.

Table 5.2. Results combined with partial reseeding

Circuit	Test Cube	[Krishna 01]		Proposed		
		Specified	Storage	Specified	Storage	Reduction
s13207	266	9389	9872	7231	7678	22.2%
s15850	269	10944	11322	7115	7393	32.4%
s38417	376	30669	31245	21102	21780	30.3%
s38584	296	26185	28312	21652	22199	21.6%

Table 5.3. Results comparing with [Ward 05]

Circuit	Num. of Specified Bits	
	[Ward 05]	Proposed
s13207	7499	6900
s15850	8333	6726
s38417	22277	21102
s38584	23254	21652

We also compared the number of specified bits in the proposed scheme to the number of specified bits in [Ward 05] in Table 5.3. Note that [Ward 05] reports the best tester storage results among compression schemes that use both linear and non-linear techniques. In all the cases shown in [Ward 05], the proposed scheme reduces the number of specified bits more. Not only does the proposed scheme provide greater compression than previous schemes that combine linear and non-linear compression

techniques (i.e., [Krishna 02], [Sun 04], and [Ward 05]), it also allows continuous flow decompression and the design of the decoder is independent of the test data.

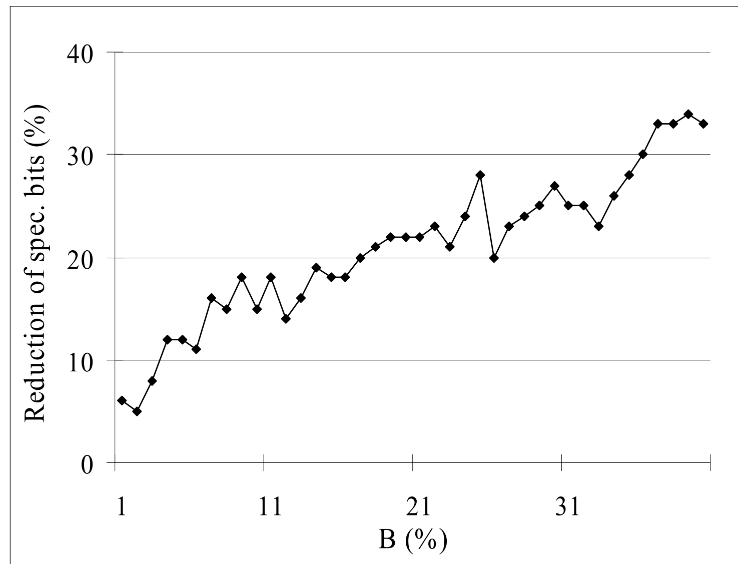


Figure 5.7. Results with test set that has few specified bits

The percentage of specified bits in the test sets for the ISCAS89 circuits (around 5-20%) is typically much higher than what is reported for industrial circuits. Experiments were performed to see how effective the proposed scheme would be with much lower percentages of specified bits. A test set that has 2% specified bits was randomly generated with different degrees of correlation. The amount of correlation was controlled by a variable *B%* which determines both the bit-wise correlation and the pattern-wise correlation. Each bit has a 2% probability of being specified and 98% probability of being a don't-care. If a bit is specified, then it has *B%* chance of having the same specified value as the previous specified bit in the test cube. Pattern-wise correlation is generated in the following way. If the previous test cube was specified in

some bit position, then there is a 50% chance for the current test cube to also be specified in the same bit position and a *B%* chance of having the same specified value as the previous test cube. Figure 5.7 shows how the percentage reduction in the number of specified bits varies with the amount of correlation. Test sets typically have quite a bit of correlation, so this data suggests the proposed method can be quite effective.

5.4. CONCLUSIONS

The proposed scheme harnesses the power of linear and non-linear decompression together using a simple and compact decoder whose design is independent of the test set. Note that the compression could be significantly improved if scan chain reordering was employed along with the proposed scheme to increase bit-wise correlation.

Chapter 6: *Efficiently Utilizing ATE Vector Repeat for Compression by Scan Vector Decomposition*

Previous approaches for utilizing ATE vector repeat are based on identifying runs of repeated scan data and directly generating that data using ATE vector repeat. Each run requires a separate vector repeat instruction, so the amount of compression is limited by the amount of ATE instruction memory available and the length of the runs (which typically will be much shorter than the length of a scan vector). In this chapter a new and more efficient approach is proposed for utilizing ATE vector repeat. The scan vector sequence is partitioned and decomposed into a common sequence which is the same for an entire cluster of test cubes and a unique sequence that is different for each test cube. The common sequence can be generated very efficiently using ATE vector repeat. Experimental results demonstrate that the proposed approach can achieve much greater compression while using many fewer vector repeat instructions compared with previous methods.

6.1 Decomposing Scan Data

In the proposed scheme, the set of test cubes in a test set are partitioned into clusters that share many input assignments. The scan data is then decomposed into two components: the sequence of specified bits that is common across all the test cubes in a cluster, and the sequence of specified bits that is unique to each test cube. An example is shown in Fig. 6.1 to illustrate how the scan data is decomposed. Assume that the eight test cubes shown in Fig. 6.1 are included in one cluster. Each bit position in a test cube cluster can be classified as either being a don’t care if no test cube has a specified value in that bit position, having “common data” if all test cubes have compatible values in that bit position, or having “unique data” if two or more test cubes have conflicting

specified values. In the example in Fig. 6.1, the last bit position is a don't care, The 1[s] and 3rd bit positions have compatible value across all of the eight test cubes and thus are common data. The common data can be generated by the common sequence decompressor that operates based on ATE vector repeat since it is the same for each test cube. The 2nd, 4th, 5th, 6th, and 7th bit positions have conflicting values and thus are unique data. They must be generated by the unique sequence generator. The common data for the test cube cluster is shown in Fig. 6.1 along with the unique data for each test cube.

Because the scan data is decomposed into common data and unique data, a control signal is required to indicate if a bit position should be filled from the common data or the unique data. This is illustrated in Fig. 6.2. The control signal is a don't care for any don't care bit position in a cluster (in the example in Fig. 6.2, only the last bit position is a don't care), and it has a specified value for all other bit positions. A key property is that the same control sequence can be used when decompressing all test cubes in a cluster and thus it is a "common control". This means that the control signal can be generated by the common sequence generator using ATE vector repeat and thus the storage required for the common control is amortized across all the test cubes in the cluster. The common control sequence for the example test cube cluster is shown in Fig. 6.1.

Typically a large number of specified bits can be included in the common data because many test cubes have similar input assignments due to the fact that they are structurally related in the circuit. The clustering procedure described in Sec. 6.3 selects the clusters to maximize the amount of compression for the cluster. During decompression, the common sequence generator produces both the common data and common control. Only one copy of the input stream for the common sequence generator needs to be stored in the ATE vector memory since it can be applied using the ATE

vector repeat instruction for all the test cubes in the cluster. Therefore, a large reduction in storage requirements can be achieved.

Original								
Test cube 1	0	1	1	1	1	0	1	x
Test cube 2	0	0	1	1	1	0	1	x
Test cube 3	0	1	1	1	1	0	0	x
Test cube 4	0	1	1	0	0	1	1	x
Test cube 5	0	0	x	1	1	0	0	x
Test cube 6	0	1	1	0	1	1	1	x
Test cube 7	x	1	1	1	0	1	1	x
Test cube 8	x	1	1	1	1	1	1	x
Encoded								
Common control	1	0	1	0	0	0	0	x
Common data	0	x	1	x	x	x	x	x
Unique data 1	x	1	x	1	1	0	1	x
Unique data 2	x	0	x	1	1	0	1	x
Unique data 3	x	1	x	1	1	0	0	x
Unique data 4	x	1	x	0	0	1	1	x
Unique data 5	x	0	x	1	1	0	0	x
Unique data 6	x	1	x	0	1	1	1	x
Unique data 7	x	1	x	1	0	1	1	x
Unique data 8	x	1	x	1	1	1	1	x

Figure 6.1. Example of proposed encoding scheme

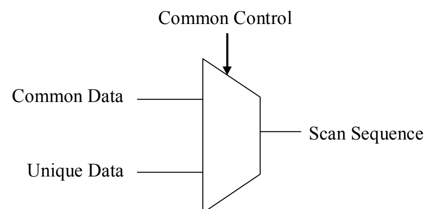


Figure 6.2. Selecting scan sequence

In the example in Fig. 6.1, the number of specified bits in the original test cubes is 53, and the number of specified bits in the encoded test cubes is 49 (2 specified bits in the common data, 40 specified bits in the unique data, and 7 specified bits in the common

control). The reduction in the example shown in Fig. 6.1 is small, but in real cases, the number of test cubes in a cluster is much greater than the number of test cubes in this example, so the number of specified bits generated from the common data is much higher, thereby making the reduction in the total number of specified bits larger.

6.2 DECOMPRESSION HARDWARE

The proposed scheme requires relatively simple decompression hardware (two sequential linear decompressors and one MUX per scan chain). The proposed decompression hardware is shown in Fig. 6.3. There are two types of memory in the ATE, instruction memory and vector memory. The instruction memory stores the ATE instructions including the vector repeat instructions, and the vector memory stores the data. The data is transferred to the decompressors based on the ATE instructions. There are two sequential linear decomp-ressors in Fig. 6.3. One decompressor (the upper one in Fig. 6.3) operates with ATE vector repeat and the other decompressor operates without the ATE vector repeat. The decompressor that operates with vector repeat generates the common data and common control signals for each scan chain. Only one copy of the input stream for generating the common data and common control for all the test cubes in a cluster is stored in the ATE and applied repeatedly for each test cube in the cluster using an ATE vector repeat instruction. The other decompressor (the lower one in Fig. 6.3) generates the unique data and operates without vector repeat. This decompressor operates in the same way as the decompressor in conventional linear compression schemes. A 2-to-1 multiplexer is placed between the decompressors and each scan chain.

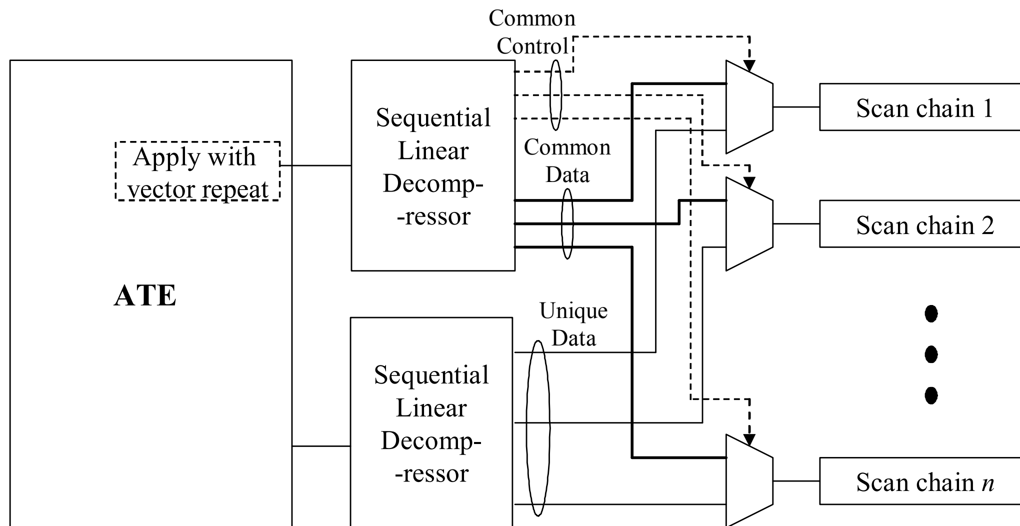


Figure 6.3. Decompression hardware

One very nice property of sequential linear decompressors is that regardless of how many outputs signals they generate or how long of a sequence they generate, the number of input bits that are required for the decompressor depends only on the total number of specified bits that it needs to generate (the rest of the bits are essentially filled with random data). Thus the architecture can be easily scaled to any number of scan chains limited only by the rate at which data from the ATE can be transferred to the sequential linear decompressors relative to the number of specified bits that the decompressor needs to generate. Note that the decompression hardware does not depend on the circuit or test set, which makes it possible to reuse it when testing multiple cores in a system-on-chip (SOC) design.

The sequential linear decompressors that are used for this scheme could be any of the ones described in [Krishna 01], [Konemann 01], or [Rajski 02]. An example of the sequential linear decompressor is shown in Fig. 6.4.

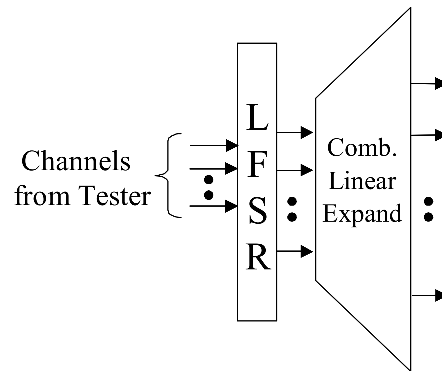


Figure 6.4. Example of sequential linear decompressor

6.3 FORMING TEST CUBE CLUSTERS

In the proposed scheme, test cubes are grouped into clusters. Each cluster requires the use of an ATE vector repeat instruction for generating the common sequence for the cluster. Because the ATE instruction memory is limited, the number of clusters cannot exceed the amount of ATE instruction memory available. For that reason, the proposed clustering algorithm tries to maximize the correlation in each cluster (to reduce the number of specified bits, thereby minimizing the tester storage), while at the same time generating a small number of clusters (to minimize the number of ATE repeat instructions required).

6.3.1 Clustering algorithm

Test cube clustering has been previously studied and some nice algorithms can be found in [Alleyne 94]. For the proposed scheme, a special benefit function is needed to account for the both the control and data bits required to encode each cluster.

In order to maximize the compression achieved for each cluster, it is important that the test cubes in each cluster have many bit positions with compatible values. As more test cubes are added to a cluster, the number of clusters is reduced. This has the

benefit of minimizing the number of the ATE instructions required for the vector repeat, but there is a tradeoff as more bit positions are likely to have conflicts and thus reducing the effectiveness of each repeat instruction. A greedy clustering procedure that takes this tradeoff into consideration is described here.

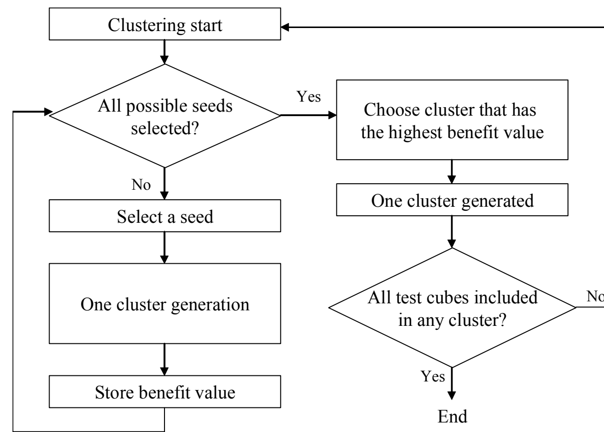


Figure 6.5. Global diagram

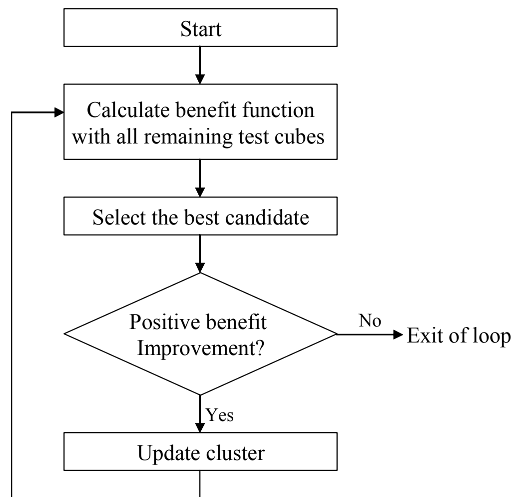


Figure 6.6. Sub-diagram of one cluster generation

The flow diagram of the proposed clustering algorithm is shown in Fig. 6.5 and Fig. 6.6. One test cube is used as a seed for each cluster. All other test cubes are then considered as candidates to add to the cluster. The heuristic that is used to measure the optimality of a cluster depends on the number of specified bits that are present in each compatible bit position of the cluster as well as the number of specified bits in the common data and common control for the cluster. A large number of specified bits in the common data and the common control may degrade the efficiency of the proposed scheme. The benefit function that is used to assign a value to a cluster is shown in the below:

$$Benefit = \frac{total_spec}{compatible_pos + spec_pos + total_unique_spec}$$

where *total_spec* is the total number of specified bits in the cluster, *compatible_pos* is the number of bit positions that are compatible in the cluster, *spec_pos* is the number of bit positions that have specified bits, and *total_unique_spec* is the total number of specified bits not in compatible bit positions. Essentially the numerator corresponds to the uncompressed storage requirements, and the denominator corresponds to the compressed storage requirements (the common data has a specified bit for each compatible bit position in the cluster, the common control has a specified bit for each bit position that has one or more specified bits in the cluster, and the unique data has one specified bit for each specified bit not in a compatible bit position in the cluster). The larger the benefit value, the larger the amount of compression that is achieved when encoding the cluster.

Consider the example test cubes in Fig. 6.7. Assume that *test cube 1* is chosen as a seed. *test cube 1* is included in the cluster. A test cube that generates the largest benefit value when added to the cluster is *test cube 2*, so it is added to the cluster, and the

benefit value (0.818) is stored. The next test cube that generates the largest benefit value is *test cube 3*, which gives 1.077. Because the current benefit value (1.077) is larger than the previous benefit value (0.818), *test cube 3* is added to the cluster. Then, *test cube 4* is chosen as a candidate. The benefit value is 1.058, which is smaller than the previous benefit value (1.077). So *test cube 4* is not added, and the final cluster consists of *test cube 1*, *test cube 2,* and *test cube 3*. This process is repeated iteratively for the remaining test cubes until all test cubes are members of a cluster.

Note that while a greedy clustering procedure is described here, any clustering procedure in the literature can be used to maximize the benefit function defined here.

Original								
test cube 1	x	1	1	1	0	x	0	x
test cube 2	x	1	1	x	0	x	1	x
test cube 3	x	1	1	1	1	x	x	x
test cube 4	x	1	0	1	1	x	x	x

Figure 6.7. Example of clustering

6.3.2 Handling Constraints on Number of Clusters

The clustering algorithm described in Sec. 6.3.1 maximizes the reduction in the number of specified bits. However, it may generate too many clusters in some cases. Note that one ATE repeat instruction is required in the proposed scheme for each cluster. Thus, there is a limit on how many clusters can be used based on the amount of ATE instruction memory that is available. To provide a mechanism for reducing the number of clusters generated by the clustering procedure, a tuning variable, *k*, is added to the algorithm shown in Fig. 6.6. The modified algorithm is shown in Fig. 6.8.

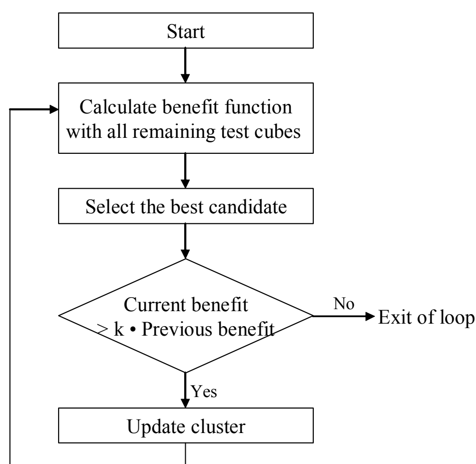


Figure 6.8. Modification of clustering algorithm

In the original clustering algorithm, the value of *k* is 1. To increase the number of test cubes in each cluster, the condition for adding a test cube into a cluster is loosened by making the value of *k* lower than 1. By lowering the value of *k*, the number of clusters reduces with a reasonable sacrifice in the number of specified bits. In the example in Fig. 6.7, all of 4 test cubes are included in the cluster with *k*=0.9 while only 3 test cubes are included in the cluster with *k*=1. The benefit value for merging test cube 4 is 1.058. The previous benefit value (1.077) is multiplied by *k* (0.9) and thus reduced to 0.969. This allows the benefit value of including *test cube 4* be larger than the adjusted previous benefit value. Therefore, *test cube 4* is also included in the cluster. In this manner, the number of test cubes in each cluster increases depending on the value of *k*. Experimental results with different values of *k* are shown in Fig. 6.9 and Fig. 6.10. When *k*=1, the number of clusters in *s38584* is 98. If that is too large for the available ATE instruction memory, then *k* can be adjusted lower than 1 so that the clustering algorithm generates a smaller number of clusters. Note that if the value of *k* is reduced a lot, at some point the number of specified bits in the encoded test cubes approaches the

number of specified bits in original test cubes and hence no compression is achieved. As can be seen for *s38584* in Fig. 6.9 and Fig. 6.10, if the value for *k* is chosen between 0.88 and 0.91 a significant reduction in the number of specified bits is achieved using only 15~30 clusters.

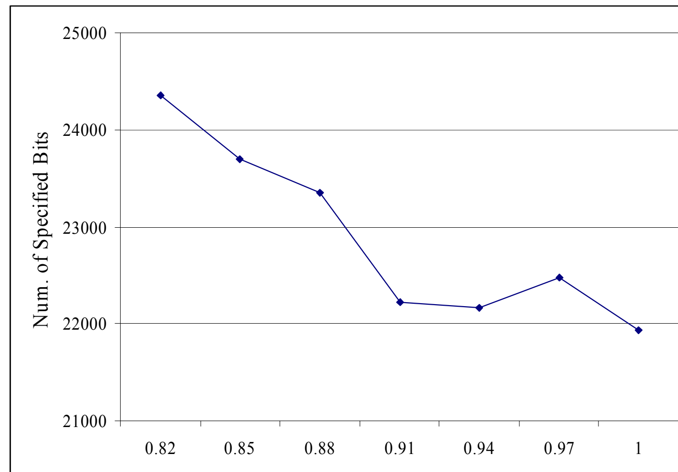


Figure 6.9. Number of specified bits in encoded data vs. *k*

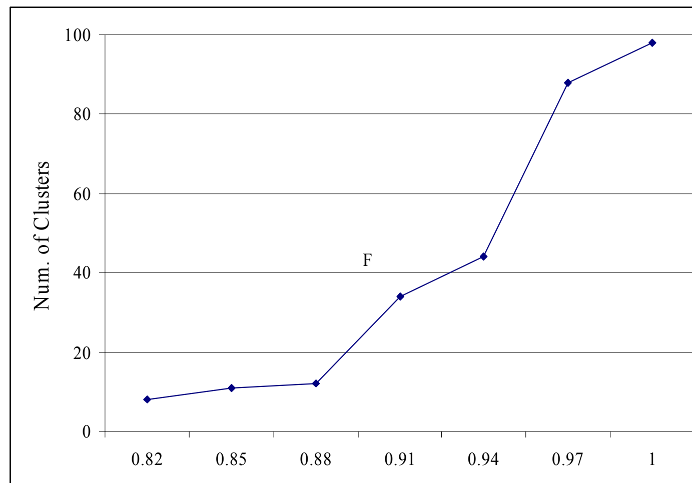


Figure 6.10. Number of clusters vs. *k*

6.4 EXPERIMENTAL RESULTS

Experiments were performed on the six largest ISCAS-89 benchmark circuits. In Table 6.1, the number of test cubes and the original number of specified bits in the deterministic test sets are shown in the second and third columns. The fourth column shows the number of clusters generated by the clustering algorithm described in Sec. 6.3. For each circuit, results are shown for two different numbers of clusters. One was obtained with a value of $k=1$, and the other was obtained by adjusting the value of k. The fifth and sixth columns show the number of specified bits in the unique sequence and the number of specified bits in common sequence. The seventh column shows the total number of specified bits in the encoded test cubes. The last column shows the reduction in the number of specified bits. Note that the maximum number of clusters shown in Table 6.1 is 98, which means that the maximum number of ATE vector repeat instructions that have to be stored in the ATE instruction memory is only 98 or less for these circuits. In most circuits, the number of clusters is below 40. Of course, the number of clusters can also be reduced if necessary by lowering k. Since the number of specified bits with the proposed scheme is reduced, the number of free-variables that are needed to encode the data using the linear decompressors will also reduce correspondingly.

To get results for the actual tester storage requirements using the proposed approach, we did experiments using the linear decompressor described in [Krishna 01] (other linear decompressors could also be used). There results are shown in Table 6.2. The same test sets are compressed using the linear decompressor in [Krishna 01] by itself, and using it in conjunction with the proposed scheme to utilize ATE vector repeat. The results show that the amount of data that needs to be stored in the ATE vector memory is significantly reduced with the proposed scheme.

Table 6.1. Results for proposed scheme on benchmark circuits

Circuit	Num. Cubes	Original Spec. Bits	Num. Clust.	Num. Spec. in Unique Seq.	Num. Spec. in Common Seq.	Encoded Specified Bits	Reduc.
s5378	196	5236	10	3366	893	4259	18.7%
			31	2462	1633	4095	21.8%
s9234	205	10324	13	5667	1709	7376	28.6%
			26	4045	3067	7112	31.1%
s13207	266	9389	18	5612	1881	7493	20.2%
			40	4551	3049	7600	19.1%
s15850	269	10944	15	6335	2103	8438	22.9%
			34	4650	3579	8229	24.8%
s38417	376	30669	19	12427	7207	19634	35.9%
			31	11044	8131	19175	37.5%
s38584	296	26185	18	17505	5849	23354	10.8%
			98	11293	10642	21935	16.2%

Table 6.2. Results for using linear decompressor in [Krishna 01] alone versus using it with the proposed scheme

Circuit	[Krishna 01]		Proposed		
	Num. Specified	Vector Memory	Num. Specified	Vector Memory	Reduction
s5378	5236	5512	4095	4358	20.9%
s9234	10324	10692	7112	7503	29.8%
s13207	9389	9872	7493	7750	21.5%
s15850	10944	11322	8229	8694	23.2%
s38417	30669	31245	19175	20769	33.5%
s38584	26185	28312	21935	24268	14.3%

Table 6.3. Results comparing with [Vranken 03]

Circuit	[Vranken 03]		Proposed			
	Num. Repeat inst.	Vector Memory	Num. Repeat inst.	Vector Memory	Reduction	
					Inst. Mem.	Vector Mem.
S5378	498	8286	31	4358	93.8%	47.4%
S9234	540	14214	26	7503	95.2%	47.2%
s13207	976	8464	18	7750	98.1%	8.4%
s15850	894	15974	34	8694	96.2%	45.6%
s38417	2518	41506	31	20769	98.8%	50.0%
s38584	3264	53952	98	24268	96.9%	55.1%

In Table 6.3, the proposed scheme is compared with the scheme described in [Vranken 03] for utilizing ATE vector repeat. In [Vranken 03], the best results were obtained when a sequencer controls two pins, so we also assumed that a sequencer

controls two pins. And as suggested in [Vranken 03], only the vectors that can be repeated at least 16 times are encoded by the ATE vector repeat to reduce the number of the repeat instructions. Using this criteria, we generated experimental results on our test sets in the manner described in [Vranken 03] (note that results published in [Vranken 03] were for test sets that are not publicly available). The number of ATE repeat instructions is shown in the second and the fourth columns, and the amount of data stored in the vector memory is shown in the third and fifth columns. As can be seen, much larger reductions in the vector memory can be obtained with the proposed approach using an order of magnitude fewer ATE repeat instructions compared with [Vranken 03]. Of course, it should be pointed out that the method in [Vranken 03] does not require any on-chip hardware, whereas the proposed method requires two on-chip linear decompressors. However, note that the linear decompressors used in the proposed scheme will require a very small amount of area with current chip densities, and they can be reused when testing multiple cores.

Table 6.4. Results comparing with [Wang 05]

Circuit			[Wang 05]	Proposed		
Name	Num. Test Cubes	Num. Scan Cells	Vector Memory	Num. Repeat Inst.	Vector Memory	Reduction
ckt-4	1529	43414	3264850	121	1518409	53.5%
				404	1341943	58.9%
ckt-5	4900	26970	6079410	283	2579402	57.6%
				841	2268261	62.7%

In Table 6.4, a comparison is made with the scheme described in [Wang 05] that also utilizes ATE vector repeat. Here results are shown for the exact same test cube files for two of the industrial circuits that were used in [Wang 05] (the others were not publicly available). The circuit information is shown in the first, second and third columns. The fourth column shows the best results in terms of vector memory

requirements reported in [Wang 05]. For the proposed method, results are shown for two different numbers of ATE vector repeat instructions. Note that the number of repeat instructions used in [Wang 05] is not reported in the paper and thus is not shown in Table 6.4. The vector memory required is shown in the sixth column. The last column shows the reduction in the vector memory required. There is a substantial reduction. A lot of the reduction comes from the fact that the proposed scheme is based on linear decompression. A major advantage of the proposed scheme is that it is compatible with linear decompression which is known to be highly efficient.

6.5 CONCLUSIONS

The proposed scheme provides a way to utilize ATE vector repeat to achieve additional compression on top of the compression achieved using a linear decompressor. The design of the decompressor for the proposed scheme is independent of the test set or CUT and thus can be reused when testing multiple cores.

Chapter 7: *Conclusion and Future Work*

7.1 Conclusion

As mentioned in Chapter 1, large test data volume and large power consumption during test can cause unacceptably high test cost. The five techniques described in this dissertation are able to reduce test power consumption, test data volume, or both of them, thereby reducing the test cost significantly. Two schemes that reduce both test data volume and test power consumption were described in Chapters 2 and 3. One fixes don't cares in LFSR reseeding schemes thereby reducing the number of transitions in scan chains as well as the number of specified bits. The other scheme reduces the number of shift cycles based on linear feedforward network in scan chains. A low power BIST scheme that requires small hardware overhead was presented in Chapter 4, which is based on scan chain partitioning. Two test vector compression techniques have been presented in Chapters 5 and 6. In Chapter 5, a compression technique that combines linear and non-linear test vector compression using correlation-based rectangular encoding was introduced. Another test vector compression scheme that uses ATE vector repeat instructions was proposed in Chapter 6. The techniques proposed in this dissertation contribute to reducing test cost.

7.2 Future Work

All of the schemes described in this dissertation are based on the stuck-at fault model, which is the most-widely used fault model because of its efficiency and simplicity. Areas for future research include considering delay testing and considering an SOC test methodology. Delay testing is becoming increasingly important in the test flow. Delay testing is based on a delay fault model. The test methodology for delay

faults is different from the test methodology for stuck-at faults. For delay testing, two test patterns are required to detect a fault, one test pattern for initialization and the other test pattern for propagation. Furthermore, delay test should be performed at-speed while stuck-at test can be performed at much slower speed than the chips rated speed. As described above, delay testing has several restrictions that the stuck-at test does not have. Therefore, an efficient delay test compression methodology for reducing test cost is an area for future work.

SOC (System-on-a-Chip) test methodology is another active research area in VLSI testing. Recent developments in semiconductor technology and design techniques make SOC designs possible. However, problems related to test still remain. Three major problems related to test are test time, test data and test power. The power problem in a conventional chip is directly forwarded to a power problem in an SOC. Moreover, the test time problem and the test power problem are highly related to each other. In SOC testing, each core is tested independently and the interconnect wires between the cores are tested, which is very similar to SOB (System-On-a-Board) testing. If several cores are tested serially, the test time will increase too much. The cores can be tested concurrently so that the amount of test time increase is not too large. However, in some cases, cores cannot be tested concurrently. The number of input pins of a chip is limited and different cores require different test patterns. Even if these limitations are overcome, concurrent test for several cores may be impossible because of a large amount of power consumption. Because several cores are tested at the same time, the amount of power consumption during test will be much greater than the amount of power consumption when testing one core. Therefore, test power consumption restricts the number of cores that can be tested concurrently. If power consumption in an SOC can be reduced, a larger number of cores can be tested concurrently thereby

reducing test time. Test data compression is also a major issue in SOC testing. The amount of test data for an SOC is much larger than the amount of test data for a conventional chip. Developing effective test compression methodologies for SOCs is an area for future research.

Bibliography

[Alleyne 94] Alleyne, Ronald, "Clustering of Test Cubes: a Procedure for the Efficient Encoding of Complete Test Sets Based on the Intelligent Reseeding of LFSRs", Master thesis at McGill University, 1994.

[Brglez 89] F.D. Brglez, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," Proc. Of International Symposium on Circuits and Systems, pp. 1929-1934, 1989

[Chandra 01] Chandra, A., and K. Chakrabarty, "Combining low-power scan testing and test data compr---ession for system-on-a-chip," Proc. of Design Automation Conf., pp. 166-169, 2001.

[Chandra 02] Chandra, A., and K. Chakrabarty, "Reduction of SOC Test Data Volume, Scan Power and Testing Time Using Alternating Run-length Codes," Proc. of Design Automation Conference, pp. 673-678, 2002.

[Chandramouli 03] Chandramouli, M., "How to Implement Deterministic Logic Built-In Self-Test (BIST)," Complier: A Monthly Magazine for Technologists Worldwide, Synopsys, Jan. 2003.

[Cormen 97] T. Cormen, C. Leiserson, and R. Rivet, "Introduction to Algorithms,", The MIT Press, 1997

[Eichelberger 83] E.B. Eichelberger, and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," IBM Journal of Research & Development, Vol. 27, No. 3, pp. 265-272, 1983

[Girard 02] Girard, P., "Survey of Low-Power Testing of VLSI Circuits," IEEE Design & Test of Computers, pp. 82-92, 2002.

[Hellebrand 92] Hellebrand, S., S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Register," Proc. of International Test Conference, pp. 120-129, 1992.

[Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," IEEE Trans. on Computers, Vol. 44, No. 2, pp. 223-233, Feb. 1995

[Krishna 01] Krishna, C.V., A. Jas, and N.A. Touba, "Test Vector Encoding Using Partial LFSR Reseeding", Proc. of IEEE International Test Conference, pp. 885-893, 2001.

[Krishna 02] Krishna, C.V., and N.A. Touba, "Reducing Test Data Volume Using LFSR Reseeding with Seed Compression ", Proc. of IEEE International Test Conference, pp. 321-330, 2001.

[Krishna 04] Krishna, C.V., and N.A. Touba, “3-Stage Variable Length Continuous-Flow Scan Vector Decompression Scheme”, Proc. of IEEE VLSI Test Symposium, pp. 79-86, 2004.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," Proc. of European Test Conference, pp. 237-242, 1991.

[Koenemann 01] Koenemann, B., C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding," Proc. of VLSI Test Symposium, pp. 325-330, 2001

[Lee 04] Lee, J., and N.A. Touba, "Low Power Test Data Compression Based on LFSR Reseeding," Proc. of Int. Conf. on Comp. Design, pp. 180-185, 2004.

[Liang 01] Liang, H.-G., S. Hellebrand, and H.-J. Wunderlich, “Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST,” Proc. of International Test Conf., pp. 894-902, 2001.

[Pomeranz 92] I. Pomeranz, and S.M. Reddy, “3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set”, Proc. of International Conf. On VLSI Design, pp. 148-153, 1992

[Rajski 02] Rajski, J., J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, T. Kun-Han, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eider, and Q. Jun, "Embedded deterministic test for low cost manufacturing test," Proc. of International Test Conference, pp. 301-310, 2002.

[Rosinger 02] Rosinger, P. M., B.M. Al-Hashimi, and N. Nicolici, "Low Power Mixed-Mode BIST Based on Mask Pattern Generation Using Dual LFSR Re-seeding," Proc. of Int. Conference on Computer Design, pp. 474-479, 2002.

[Samaranayake 02] Samaranayake, S., N. Sitchinava, R. Kapur, M.B. Amin, and T.W. Williams, "Dynamic Scan: Driving Down the Cost of Test," Computer, Vol. 35, Issue 10, pp. 63 - 68, 2002

[Sankaralingam 00] Sankaralingam, R., R.R. Oruganti, and N.A. Touba, "Static compaction techniques to control scan vector power dissipation," Proc. of VLSI Test Symp., pp. 35-40, 2000.

[Sankaralingam 01] Sankaralingam, R., B. Pouya, and N.A. Touba, "Reducing Power Dissipation During Test Using Scan Chain Disable", Proc. of IEEE VLSI Test Symp., pp. 319-324, 2001.

[Saxena 01] Saxena, J., K.M. Butler, and L. Whetsel, "An Analysis of Power Reduction Techniques in Scan Testing," Proc. of International Test Conference, pp. 670-677, 2001

[Sentovich 92] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saklanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli, “SIS: A System for Sequential Circuit Synthesis”, University of California-Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, 1992

[Shnurnann 75] H.D. Shnurnann, E. Lindbloom, and R.G. Carpenter, “The Weighted Random Test-Pattern Generator,” IEEE Tran. On Computers, pp. 695-700, 1975

[Sinanoglu 04] Sinanoglu, O., and A. Orailoglu, "Scan Power Minimization Through Stimulus and Response Transformations" Proc. of IEEE Desgin, Automation and Test in Europe Conf. and Exhibition, Vol. 1, pp. 404-409, 2004.

[Sun 04] Sun, X., L. Kinney, and B. Vinnakota, “Combining Dictionary Coding and LFSR Reseding for Test Data Compression,” Proc. of Design Automation Conference, pp. 944-947, 2004.

[Whetsel 01] Whetsel, L., "Adapting scan architectures for low power operation" Proc. of International Test Conference, pp. 863-872, 2000.

[Zacharia 95] Zacharia, N., J. Rasjski, and J. Tyszer, "Decompression of Test Data Using Variable-Length Seed LFSRs," Proc. of VLSI Test Symposium, pp. 426-433, 1995.

[Vranken 03] Vranken, H., Hapke, F., Rogge, S., Chindamo, D., and Volkerink, E., “Atpg Padding and ATE Vector Repeat per Port for Reducing Test Data Volume,” *Proc. of IEEE International Test Conference*, pp. 1069-1078, 2003

[Wang 99] S. Wang and S.K. Gupta, “LT-RTPG: A New Test-Per-Scan BIST TPG for Low Heat Dissipation”, Proc. of International Test Conference, pp. 85-94, 1999.

[Wang 01] S. Wang, “Low Hardware Overhead Scan Based 3-Weight Weighted Random BIST”, Proc, of International Test Conference, pp. 868-877, 2001

[Wang 02] S. Wang, “Generation of Low Power Dissipation and High Fault Coverage Patterns For Scan-Based BIST”, Proc. of International Test Conference, pp. 834-843, 2002

[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 Compression Scan Technology for Test Cost Reduction,” Proc. of International Test Conference, pp. 916-925, 2004.

[Wang 05] Wang, Z., and Chakrabarty, K., "Test Data Compression for IP Embedded Cores Using Selective Encoding of Scan Slices," *Proc. of IEEE International Test Conference,* pp. 581-590, 2005

[Ward 05] Ward. I. S., Schattauer, C., and N. A. Touba, "Using Statistical Transformations to Improve Compression for Linear Decompressors," *Proc. Of Defect and Fault Tol. in VLSI Sys*., pp. 42-50, 2005.

Vita

Jinkyu Lee was born in Seoul, Korea on February, 26 in 1976, the son of Byoung-ik Lee and Suk-nye Kim. He graduated O-gum High School in 1994 and earned his Bachelors in 2001 from YONSEI University, Seoul, Korea, where he majored in Electrical Engineering with Honors. He joined the University of Texas at Austin in 2002, where he is currently doing his PhD in Computer Engineering. He did his MS in Computer Engineering at the University of Texas at Austin with a thesis on VLSI Testing in 2004. His area of research interest is VLSI Testing, with focus on test power reduction and test data compression.

Permanent Address : 118-1101 Samsung Le-mi-an APT
Munjeong-Dong, Songpa-Gu
Seoul, Korea (138-764)

This dissertation was typed by Jinkyu Lee.