Fast Seed Computation for Reseeding Shift Register in Test Pattern Compression

Nahmsuk Oh, Rohit Kapur, and T. W. Williams
Synopsys Inc., 700 E. Middlefield Road, Mountain View, CA 94043
{nahmsuk, rkapur, tww}@synopsys.com

ABSTRACT

Solving a system of linear equations has been widely used to compute seeds for LFSR reseeding to compress test patterns. However, as chip size is growing, solving linear equations requires a large number of computations that is proportional to n^3. This paper proposes a new scan chain architecture and algorithm so that the order of computation is proportional to the number of scan cells in a chip. The new architecture is a methodology change that does not require complex Design-For-Testability (DFT) as proposed in the previous techniques. Instead of solving linear equations, the proposed new seed computation algorithm topologically determines seeds for test vectors. The compression ratio might be slightly lower than the other approaches, but the proposed approach can handle larger designs in a reasonable amount of time. Computation analysis shows that, for 1 million scan cell design, if we assume it takes 1 *msec* for the proposed technique to compute seeds, it would take more than 14 minutes for other techniques that solve linear equations.

1. INTRODUCTION

As more and more transistors are placed on a single chip, the test data storage on the tester and the test data bandwidth (needed to adequately test the chip) between the tester and the chip grows rapidly [1]. With the escalating cost of test, the need for reducing test data is increasing. A Linear Feedback Shift Register (LFSR) with reseeding method has been used to compress test patterns. In this method, a seed is a starting state of the LFSR, and the LFSR runs autonomously for a certain number of cycles after each seed is loaded into the LFSR. The seed can be loaded as frequently as required to obtain a balance between the amount of test data and the test application time, which depends on the number of test patterns generated from the seeds. Thus, the tester stores much smaller seeds instead of storing full test patterns. In other words, the test patterns can be compressed into seeds.

There is a direct relationship between the seed values and the values that are achieved in the scan chains connected to the LFSR. Due to dependencies between the relationships, a seed is computed by solving a system of linear equations [2]. Several techniques have been proposed to improve encoding efficiency of reseeding the LFSR, but they are all based on solving linear equations. The computation complexity of solving linear equations is $O(n^3)$, where n represents the number of variables in the equations. Thus, for larger chip sizes, the number of unknowns in the equations could be quite large, and the computation time grows rapidly. For example, in [3], the largest number of unspecified bits – the same as the number of unknowns in linear equations – is 85, and a few hours of computation time for 340 test patterns were reported. Suppose the chip contains more than 1 million scan cells, with an assumption that 0.1% of scan cells are specified in each test pattern - 1000 bits in each test pattern are specified. Solving 1000 unknowns in 1000 equations will take much longer than a few hours.

In this paper, we propose a new scan chain architecture and Fast Seed Computation (FSC) algorithm to encode a test pattern into a seed vector with the computation complexity $O(n)$. Since the order of computation increases linearly with the number of scan cells in the Circuit Under Test (CUT), our FSC technique can handle large size of chips easily.

This paper is organized as the following: Section 2 discusses previous works briefly. Section 3 illustrates our new scan chain architecture and describes the FSC algorithm. Section 4 and 5 analyzes the computation complexity of the FSC, and compares it with the previous approaches. Finally, Sec. 6 concludes the paper.

2. PREVIOUS WORK

Encoding scan test patterns using LFSR was originally proposed in [2], in which LFSR length needed for successfully finding seeds for tests should be 20 more bits than the specified bits in test cubes. A *test cube* is a test pattern in which the bits, that are not specified by ATPG, are left as don't cares. Thus, each bit of the test cube has one of the three values: logic-0, logic-1, and logic-X (don't care). An efficient encoding approach using multiple-polynomial LFSR was later proposed in [4]. Instead of using a single LFSR that has more bits than specified bits in the test cube, this technique used 16 different polynomials for LFSR. In this approach, the LFSR size can be reduced to the maximum number of specified bits in test cubes.

Since the number of specified bits in a test cube varies among test cubes, variable-length seeds for LFSR may encode test cubes more efficiently than the fixed size of seeds. Variable-length seed approach was proposed in [5][6]. A larger size of LFSR is used for larger number of specified bits in test cubes, and a smaller size of LFSR is used to encode test cubes with the smaller number of specified bits.

One of the disadvantages of the previous approaches is that they are static reseeding methods [3]. After loading a seed into CUT, the tester should stop so that the LFSR runs autonomously to fill in the scan chains. A dynamic reseeding approach using partial reseeding method was proposed in [3] to avoid stopping testers during testing. In this technique, not only the LFSR but also scan-in vectors (which are seeds) generate a test cube. Instead of loading an entire seed onto the LFSR, the partial reseeding technique lets the LFSR run continuously and performs XOR operation with the LFSR output and the scanned-in vector to fill in the scan chain. The dynamic reseeding shows very good compression ratio by preserving the degrees of freedom in the solution space for the test cube, but it may take longer than other techniques to compute seed vectors because it has to solve much larger system of linear equations.

All the previous approaches for reseeding the LFSR requires solving a system of linear equations. Suppose each arithmetic

0-7803-7607-2/02/$17.00 ©2002 IEEE

computation and logical operation is a single operation. Then, the number of operations to solve linear equations with n unknowns is approximately $n^3/3$ [7]. However, the number of operations in the FSC method proposed in this paper increases linearly with the number of scan cells. The next section describes this technique in detail.

3. PROPOSED APPROACH

3.1. Architecture

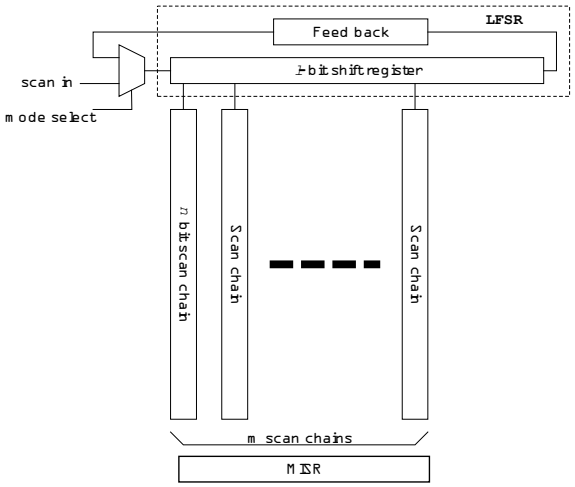

Figure 3.1. The FSC architecture.

Figure 3.1 shows the architecture of the proposed approach. In this example, the CUT has at least mn scan cells and each scan chain of m scan chains has maximum n scan cells. Every scan chain is connected to one bit of l bit shift register (SR). The SR operates in two modes: the linear feed back shift register with feed back loop and shift register with scan-in port. The LFSR is used for autonomous run to fill scan chains with random test patterns to prune out easy faults. The remaining undetected faults are *random resistant faults* and the second mode is used for reseeding the SR to detect random pattern resistant faults. The size of the shift register, l, should be large enough to produce pseudo random values for the longest scan chain in the first mode, but can be m in the second mode. Thus, l is the maximum of the two values.

Note that this architecture does not have a phase shifter between the LFSR and the scan chains as proposed in STUMPS architecture [8].

3.2. Overview of our approach

Our approach has two modes in testing a CUT. In the first phase, the LFSR runs autonomously and fills the scan chains with random patterns. Capture clocks are applied and the scan chains are filled with measured values. These patterns are scanned out to the MISR to detect any faults while the next input patterns are shifted in to the scan chains by running the LFSR again. The first phase of the test detects easy faults in the system.

The second phase of the test targets the random pattern resistant faults remained after the first phase. We assume that a test cube has the number of specified bits less than $m + n - 1$. After removing easy faults by running LFSR autonomously, only a few number of scan cells in a test cube need to be specified to detect one or a few of the remaining faults. The rest of the scan cells are don't cares. Thus, the maximum number of specified bits s_{max} is $m + n - 1$.

The Automatic Test Pattern Generation (ATPG) tool generates patterns for the second phase. It generates a pattern with the constraint of specifying s_{max} bits at maximum. Therefore, $mn - s_{max}$ bits are unspecified in one test pattern and filled with some values. This test cube is encoded into a seed of s_{max} bits. In the second phase, the feed back loop of the SR is disconnected and the multiplexor selects the input from scan-in port so that a seed is shifted into the shift register. As shifting bits into the SR, the seed is expanded and fill in the scan chains. After s_{max} of scan shift clocks, the required bits of the scan chains are specified, and the remaining bits are filled with either 0 or 1. The following section describes how to encode the test cube into a seed, and how the seed is expanded to fill in the scan chains.

3.3. Seed computation

Before describing the detailed method, we explain some of the basic concepts.

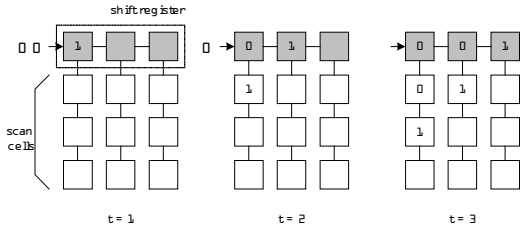

Figure 3.2. The first three clock cycles shifting 1 into SR.

Suppose the CUT has 3 scan chains that have 3 scan cells in each, and the scan chains are connected to a 3 bit LFSR as shown in Figure 3.2. The LFSR is in the second phase; thus, the SR is fed by not the feed back loop of the LFSR but the scan-in input. An input vector 001 is applied to the scan in pin, and Figure 3.2 shows how the scan chains are filled in at each shift cycle. Note that the logic value 1 is always diagonal on the cells at each cycle. In other words, our method fills the scan cells on the same diagonal with the same logic value. We use this characteristic to determine a seed for a given test cube.

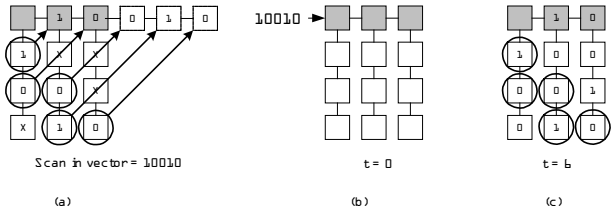

Figure 3.3. Determination of a seed for a given test cube.

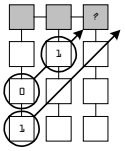

Figure 3.4. Two different logic value 0 and 1 on the same diagonal cells.

For example, suppose ATPG specifies 5 bits of the scan cells in a 9-bit scan cube as shown in Figure 3.3 (a). Circles around the scan cells represent the specified bits in the test cube. The remaining bits are don't cares and represented by X in the scan cells. Then, the specified bits are projected diagonally onto the SR as shown in Figure 3.3 (a). The projected bits in the SR form a seed vector for this test cube (excluding the first bit of the SR). In this example, thus, the seed vector for this particular test cube is 10010. Figure 3.3 (b) shows the state of CUT at t = 0 before the seed 10010 is shifted into the SR. After 6 shift clock cycles, the scan cells are all filled in – all the circled scan cells are successfully filled with the specified bit values as shown in Figure 3.3 (c).

Because the number of diagonal lines in an m by n matrix is $m + n - 1$, the maximum number of specified scan bits in a test pattern is limited to $m + n - 1$, which is also the length of the seed vector. This is the reason why we made the assumption that $s_{max} = m + n - 1$. However, if there are two different logic values 0 and 1 on the same diagonal cells as shown in Figure 3.4, we cannot project these two different values onto one bit of the SR. *Non-uniform diagonal cells* are the diagonal cells that have at least one specified bit with logic value 0 and one specified bit with logic value 1. There are three approaches to resolve this issue. The first one is for ATPG to generate a different pattern to detect the same faults if possible. The second approach is to reconfigure the scan chain so that all the scan chains are connected to form one scan chain, and use the traditional scan method. The third approach is to use Design-for-Test (DFT) technique so that scan chains are independent of one another. In other words, inputs of any logic cones are always in one scan chain. Therefore, a test vector testing a particular logic cone can be always in one of the scan chains. The first two approaches are trivial, thus, the following sections describes the DFT method in detail.

3.4. DFT: Independent scan chains

We can construct scan chains in such a way that the scan chains are independent of one another, i.e., all inputs of any logic cones are always in one scan chain; then, we can divide the test cube into multiple cubes that have no different logic values on the same diagonal cells. For example, consider a scan chain configuration shown in Figure 3.5 (a). To detect the stuck-at 0 fault in one of the inputs of the OR gate as illustrated, we need to specify 0 in the first input of the logic cone and 1 in the last input of the logic cone. Note that the three inputs of the logic cone are placed in two scan chains, sc1 and sc2. Thus, one of the specified bits is in sc1 and the other is in sc2 as shown in Figure 3.5 (b). The scan chain sc1 is not independent of sc2 because we need to use the two scan chains at the same time to detect the fault. However, in Figure 3.5 (c), those two bits are placed in only sc1. Thus, sc1 contains two specified bits as shown in Figure 3.5 (d) and independent of sc2.

Independent scan chains can be implemented in any netlist using a hierarchical methodology as illustrated in Figure 3.6. Hierarchies of the design are IP's possibly created by different teams that are bounded by flip-flops. Scan chains are constructed for these hierarchical entities by putting all the flip-flops that are not at the output of the hierarchical entity into a single scan chain. The output flip-flops are put into a separate scan chain. Thus every hierarchical entity merged into the next level of hierarchy would have two scan chains within it. In the integration of the scan chains at the next level of hierarchy, the output chains of the lower level hierarchies are incorporated into the single scan chain created for the non-output flip-flops of the top levels scan chains. The other chains of the hierarchies being incorporated are kept separate. The top-level hierarchies output flip-flops are put into a separate scan chains. Thus, the number of scan-chains at any level of hierarchy equals two more than the number of non-output scan chains coming from the hierarchical entities being incorporated. This methodology of creating scan chains is repeatable for multiple levels of hierarchies.

In the example shown in Figure 3.6, each IP block has two scan chains. One scan chain consists of the scan cells that are inputs of the outside logic (such as glue logic). Boundary scan cells are examples of the inputs of the outside logic. The other scan chain consists of the rest of the scan cells that are inputs of logic cones inside the IP block. The two scan chains in one IP block are independent of each other because the logic cones that have outputs to outside logic are separated from the logic cones that have outputs inside the IP. Therefore, the three scan chains shown in Figure 3.6 – the internal scan chain in IP block 1, the internal scan chain in IP block 2, and the scan chain connecting output interface logic of IP blocks are independent.

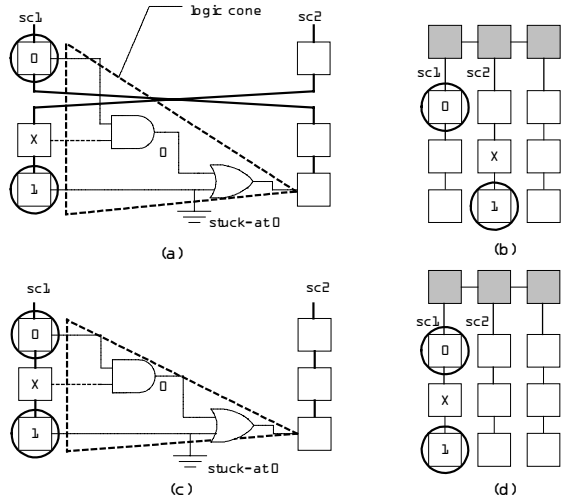

Figure 3.5. (a) Inputs of the logic cone are placed in two scan chains. (b) The two specified bits are in sc1 and sc2; thus, sc1 and sc2 are not independent of each other. (c) Inputs of the logic cone are placed in one scan chain. (d) The scan chain sc1 and sc2 are independent of each other.

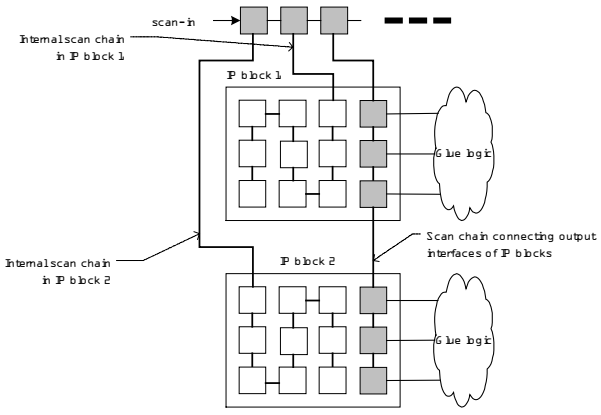

Figure 3.6. Scan chains of IP blocks are independent of one another.

3.5.Resolving non-uniform diagonal scan cells

Since scan chains are independent, one test cube can be partitioned into multiple test cubes as long as the partitioned cubes preserve specified bits in each scan chain. For example, suppose a test pattern P has two different values on the same diagonal cells as shown in Figure 3.7. Since one seed vector is able to specify one full scan chain, this cube can be partitioned into two test cubes: the one with the specified bits in sc1 and the other with the specified bits in sc2. Then, two seed vectors for sc1 and sc2 can be obtained as shown in Figure 3.7.

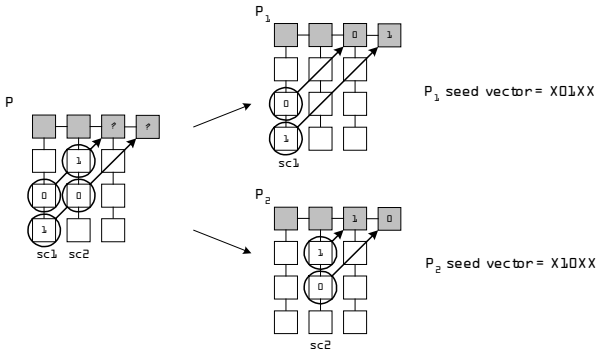

Figure 3.7. Test cube P is partitioned into P_1 and P_2.

More complex cases occur when there are multiple non-uniform diagonal cells. An example is shown in Figure 3.8 (a), in which D_1 and D_2 represent two non-uniform diagonal cells. Partitioning the test cube into minimum number of cubes is a *graph coloring* problem. A graph is constructed such a way that vertices represent scan chain numbers and edge (S_i, S_j) represents two scan chains with different values on the non-uniform diagonal cells. The corresponding graph is shown in Figure 3.8 (b). S_1 has 1 on D_1 while S_2 and S_3 have 0 on D_1, thus two edges, (S_1, S_2) and (S_1, S_3) are drawn in Figure 3.8 (b). Similarly, for D_2, S_2 has 1, but S_4 has 0, thus, (S_2, S_4) is drawn in the graph. After constructing a graph, a graph coloring algorithm is applied. In graph coloring problem, no edge has two end-points with the same color, and the number of colors is minimized. The minimum number of colors is the number of partitioned test cubes, and each partitioned test cube consists of the scan chains with the same color. In the example, the graph can be painted with two colors – white and gray. Thus, it is partitioned into two test cubes, and they are shown in Figure 3.8 (c).

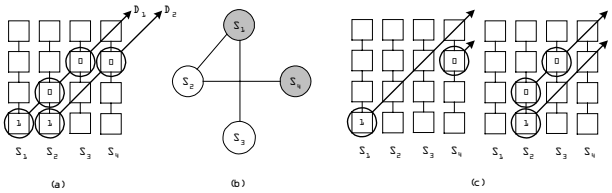

Figure 3.8. (a) Non-uniform diagonal cells D_1 and D_2. (b) A graph representing four scan chains with two colors. (c) Two test cubes based on the colored graph.

Test cube partitioning will increase the number of seed vectors. However, one partitioned test cube might be merged with another partitioned cube. An example is shown in Figure 3.9. In this example, two test cubs P_1 and P_2 are partitioned into P_{11} and P_{12}, and P_{21} and P_{22} respectively. Then, P_{11} and P_{21} are merged into P'_1, and P_{12} and P_{22} are merged into P'_2. There may be some loss of fault detection coverage in the new P'_1, but those faults missed by P'_1 will be detected by P'_2. Therefore, the number of test cubes is not increased while the same number of faults can be detected.

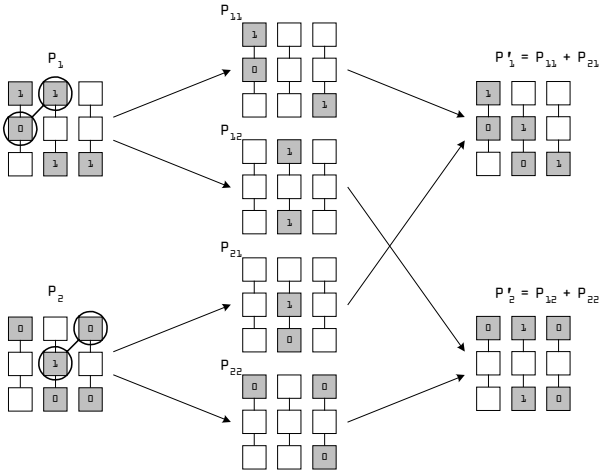

Figure 3.9. Test cube partitioning and re-composition.

4. COMPUTATION ANALYSIS

This section discusses the computation complexity of the proposed FSC algorithm. The algorithm is the following.

1. For each diagonal line of the test pattern that contains any specified bits
   If specified bits have different logic values
      Draw edges between the vertices (scan chains) that contain different logic values
   Else
      Specify the corresponding bit in the seed
2. Color the graph and partition the pattern into the patterns with the same color.

The algorithm has two steps. In the first step, it walks through the diagonal lines of the test cube that contains specified bits and checks whether the specified bits have different logic values. If they have different values, put edges between the vertices (representing scan chains) that have different logic values. This procedure is repeated until all specified bits in the test pattern are visited. The computation complexity of the first step is $O(m + n)$ because the number of diagonal lines is bounded by $m + n - 1$, where m is the number of scan chains and n is the maximum number of scan cells in one scan chain as previously shown in Figure 3.1.

In the second step, if there exist any edges between vertices in the graph constructed in the first step, i.e., if there exist non-uniform scan cells, a graph coloring algorithm is applied to partition the test cube. A graph coloring is an intractable problem, but there are well known heuristic algorithms with polynomial computation time. In our approach, the algorithm described in [9] is used. In this algorithm, coloring can be done in $O(m + |E|)$ time

for *chordal graphs*, where |E| represents the number of edges in the graph. Thus, the computation complexity of the first step in the FSC is:

$$O(m + n) + O(m + |E|) = O(2m + n + |E|) \approx O(m + n).$$

Previous approaches calculate the seed vectors solving a system of linear equations. It is well known that solving n unknowns of linear equations take $n^3/3$ operations [7]. Assuming $m + n - 1$ unknowns in the linear equations ($m + n - 1$ specified bits in the cube), solving these equations takes $(m + n)^3/3$ operations. Thus, the computation complexity of the previous approaches is $O((m+n)^3)$.

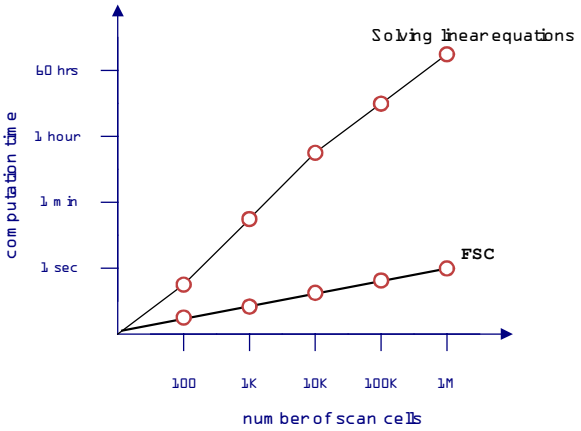

Figure 4.1. Comparison of computation times between the FSC and the previous approaches.

The graph in Figure 4.1 compares the normalized computation time (normalized to 1 sec/1 million cells in the FSC) between the proposed FSC and the previous approaches solving linear equations. The number of scan cells less than 1000 does not make much difference in computation time between the two methods. However, as the number of scan cells increases and reaches to 1 million, the FSC becomes more and more powerful and extremely outperforms the previous approaches. Note that the Y-axis is in log scale. For example, assume a circuit that contains 1 million scan cells resulting in 1000 scan chains of 1000 scan cells. Thus, $m = n = 1000$ and the number of computations are (assuming |E| = 0 for simplicity):

FSC: $2m + n = 3000$ computations

Linear equations: $(m+n)^3/3 = 8/3 \times 10^9$ computations.

If we assume our approach takes 1 *msec* to calculate the seed vector, solving linear equations takes 14.8 minutes.

5. RESULTS COMPARISON

In this section, we compare the proposed FSC technique with previous two techniques. Let us define two metrics for comparison.

$$\text{compression ratio} = \frac{\text{the number of bits in a test cube}}{\text{the number of bits in a seed}}$$

$$\text{compression cost} = \frac{\text{the number of computations to obtain a seed}}{\text{compression ratio}}.$$

We used the largest ISCAS 89 sequential benchmark circuits. In Table 5.1, the first column shows the number of scan cells in the circuit, the second column shows the number of test cubes remaining after detecting easy faults by running LFSR autonomously to produce 10000 patterns, and the third column shows the number of specified bits. They are based on the data shown in [3]. The last column shows the compression ratio of our proposed approach using DFT for independent scan chains.

Table 5.2 compares the FSC with the two previous techniques – Test cube concatenation [4] and Partial reseeding [3] – in terms of compression ratio. The compression ratio was obtained from the data shown in [3] and [4]. The compression ratio of the FSC is higher than the compression ratio of the Test cube concatenation, but slightly lower than the compression ratio of the Partial reseeding. However, if we consider the *compression cost*, our technique outperforms other techniques. The results are shown in Table 5.3. In this table, the number of operations (computations) is calculated by using the equations shown in the previous section, and the compression cost is obtained. For example, the benchmark circuit s38417 requires a compression cost of 62798 in Test cube concatenation and 80018 in Partial reseeding because they have to solve linear equations. However, the FSC requires a compression cost of only 11.83. This result shows that our technique is much powerful in larger circuits in terms of the cost of compression.

Table 5.1. Results for the proposed FSC approach.

Circuit name	scan cells	test cubes	specifi-ed bits	LFSR size	comp. ratio
s5378	214	30	493	29	7.49
s9234	247	138	4674	33	7.48
s13207	700	157	2824	53	13.21
s15850	611	167	5092	49	12.46
s38417	1664	340	23984	81	20.54
s38584	1464	62	2848	77	19.01

Table 5.2. Comparison of compression ratio.

Circuit name	Test cube concatenation		Partial reseeding		Proposed FSC	
	LFSR size	comp. ratio	LFSR size	comp. ratio	LFSR size	comp. ratio
s5378	27	5.55	38	12.8	29	7.49
s9234	61	2.77	81	6.8	33	7.48
s13207	24	20.00	44	36.5	53	13.21
s15850	46	7.69	58	19.6	49	12.46
s38417	91	4.00	105	23.1	81	20.54
s38584	70	16.67	75	30.9	77	19.01

Table 5.3. Comparison of compression cost.

Circuit name	Test cube concatenation		Partial reseeding		Proposed FSC	
	# of oper-ations	comp. cost	# of oper-ations	comp cost	# of oper-ations	comp cost
s5378	6561	1182	52488	4101	87	11.9
s9234	75660	27314	533871	78510	99	13.3
s13207	4608	230	83349	2284	159	12.2
s15850	32445	4219	234990	11989	147	11.8
s38417	251190	62798	1848411	80018	243	11.8
s38584	114333	6846	605283	19588	231	12.2

6. CONCLUSION

Solving linear equations has been a traditional approach to compute seed vectors for LFSR or to compress test cubes using LFSR. It works nicely in reasonable size of the designs. However, as technology improves, the number of transistors and sequential cells is increasing and it is getting difficult to handle larger and larger size of circuit using linear equations solver that has computation complexity of $O(n^3)$.

The FSC requires only $O(n)$ computation time. The compression ratio might be slightly lower than the compression ratio of the previous approaches, but the FSC gives us the great advantage of saving computation time to calculate seed vectors. Thus, our technique is more suitable for a very large circuit exhibiting large number of scan cells than the circuit that requires very high compression ratio on test data volume.

7. REFERENCES

[1] Khoche, A., and J. Rivoir, "I/O Bandwidth Bottleneck for Test: Is it Real?," *Proc. of International Workshop on Test Resource Partitioning,* 2000.

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

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

[4] 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," *Trans. on Computers,* No. 2, Vol. 44, pp. 223-233, Feb. 1995.

[5] Zacharia, N., J. Rajski, J. Tyszer, and J. Waicukauski, "Two Dimensional Test Data Decompressor for Multiple Scan Designs," *Proc. of International Test Conference,* pp. 186-194, 1996.

[6] Rajski, J., J. Tyszer, and N. Zacharia, "Test Data Decompression for Multiple Scan Designs with Boundary Scan," *IEEE Trans. on Computers,* Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.

[7] Gilbert Strang, *Linear Algebra And Its Applications*, Academic Press, Inc., 1980.

[8] Bardell, P.H., and W.H. McAnney, "Self-Testing of Multichip Logic Modules," *Proc. of International Test Conference,* pp. 200-204, 1982.

[9] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs,* Academic Press, San Diego, CA, 1980.