Deterministic Test Vector Decompression in Software Using Linear Operations

Kedarnath J. Balakrishnan and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712
E-mail: {kjbala,touba}@ece.utexas.edu

Abstract

A new software-based test vector compression technique is proposed for using an embedded processor to test the other components of a system-on-a-chip (SOC). The tester transfers compressed test data to the processor's on-chip memory, and the processor executes a small program which decompresses the data and applies it to the scan chains of each core-under-test. The proposed decompression procedure uses word-based linear operations to expand the compressed test data into the corresponding deterministic test vectors. It has a number of nice features that overcome the drawbacks of software-based linear feedback shift register (LFSR) reseeding. The storage requirements for the proposed approach depend only on the total number of specified bits in the test set. There are no restrictions on static compaction or the test generation procedure as a whole. The decompression program can be easily reused for applying different test sets. Experimental results demonstrate that the proposed approach compares very favorably with all previously published results for software-based test vector decompression.

1. Introduction

One major challenge of testing complex system-on-a-chip (SOC) designs is dealing with the enormous amount of test data volume [Zorian 98]. Large test data volume results in long test application time because the data needs to be transferred across the low test data bandwidth link between the tester and chip. Moreover, it requires a large amount of tester memory. Consequently, the costs of conventional testing of an SOC using ATE (automated test equipment) are scaling up rapidly.

One approach for dealing with this problem is to use test data compression techniques to reduce the amount of data that is stored on the tester. Both the test vectors and the output response can be compressed. The output response is much easier to compress since lossy compression techniques can be used. Output response compression has been extensively studied in the past and well established techniques have been developed and are used widely. Test vector compression is much more difficult because lossless compression techniques must be used to preserve the fault coverage. This paper will focus only on the problem of lossless test vector compression. In test vector compression, the test vectors are stored on the tester in a compressed form and then transferred to the chip where it is decompressed using on-chip circuitry. A number of test data compression techniques have been proposed. They differ in terms of the following properties:

- Amount of compression of the test vectors
- Reduction in test time
- Hardware required for decompression
- Requirements on the ATPG (automatic test pattern generation) software

If an SOC contains an embedded processor, then it is possible to perform the test vector decompression in software rather then having special decompression circuitry. A number of techniques have been proposed for software-based decompression. The basic idea is to transfer a decompression program along with compressed test data from a tester to an on-chip memory, and then have the processor execute the program which decompresses the test vectors and applies them to the circuit-under-test (CUT). This paper presents a new technique for software-based test vector decompression that has a number of desirable features. Before describing the proposed technique, a survey of some of the previous work in this area is given.

Previous research in using embedded processors to aid in testing has included using processors to perform memory tests [Saxena 98], [Rajsuman 99], as well as pseudo-random built-in self-test (BIST) [Rajski 93], [Gupta 94], [Stroele 95, 96, 98], [Dorsch 98], [Iyer 02]. However, the previous research most related to this paper is the work in using embedded processors for lossless test vector decompression. In [Yamaguchi 97] and [Ishida 98], techniques based on the Burrows-Wheeler transformation and run length encoding are proposed for compressing the test data transferred between a

Proceedings of the 21st IEEE VLSI Test Symposium (VTS'03)
1093-0167/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

workstation and a tester. While these techniques could also be implemented on an embedded processor for on-chip decompression, they are not well suited for that as the decompression process is complex and time consuming. A deterministic test vector compression technique based on geometric shapes is proposed in [Maleh 01]. This technique provides very good compression, but the software decompression process is complex and cannot be efficiently implemented for fast on-chip decompression. In [Jas 02], a very simple compression scheme is described which is suitable for fast on-chip decompression. Each test vector is divided into blocks, and only the blocks that are different from the preceding test vector are stored. The test vectors are then constructed on the fly by a program running on the embedded processor. In [Balakrishnan 02], matrix-based operations are used to compress deterministic test cubes in a way that takes advantage of the unspecified bits. This approach provides better compression than [Jas 02] while still retaining the advantages of a small and fast decompression program. In [Hwang 02b], a BIST technique composed of both random pattern testing and deterministic pattern testing using the embedded processor is proposed. The processor generates random patterns and selectively applies only those patterns that contribute to the fault coverage. The deterministic patterns are compressed by encoding the differences between them and similar random patterns. Though the number of random vectors that need to be applied is reduced, the total number of vectors is still much greater than the number of vectors in a fully deterministic test set.

Two methods that are closely related to the one proposed here were described in [Hellebrand 96] and [Zacharia 96] where LFSR reseeding schemes are implemented in software for decompressing deterministic test cubes. LFSR reseeding was originally proposed in [Koenemann 91]. It involves finding the starting state of an LFSR such that when the LFSR is run in autonomous mode, it generates a test vector contained in the test cube. Linear equations are formed based on the polynomial of the LFSR and then solved to find the seed. If the maximum number of specified bits in any test cube is s_{max}, then if the number of stages in the LFSR is s_{max}+20, a solution to the system of linear equations can be found with a probability of 0.999999. With LFSR reseeding, only the LFSR seeds need to be stored rather than the entire test vectors thereby providing a compression ratio equal to the length of the test vector divided by s_{max}+20. One variation of LFSR reseeding which improves the compression is to use multiple polynomials as described in [Hellebrand 92]. This reduces the storage requirements for each test vector to s_{max}+1. A software implementation of this scheme is described in [Hellebrand 96] for use on an embedded processor. While this approach is effective, it has a number of drawbacks:

- The storage requirement for each test cube is always s_{max}+1 regardless of the size of the test cube. This limits the encoding efficiency and hence compression.
- Static compaction during ATPG has to be constrained so that s_{max} doesn't become too large. This generally results in a significant increase in the size of the test set.
- The decompression program cannot be easily reused for different test sets (e.g., if different components of the SOC are to be tested), since it requires multiple primitive polynomials of degree equal to s_{max}, and s_{max} will likely be different for each test set.

The method in [Zacharia 96] performs two-dimensional LFSR reseeding to generate a group of test cubes at a time. By grouping the vectors, the s_{max} for the groups can be made more balanced thereby improving the encoding efficiency. It uses multiple LFSRs with the same feedback polynomial so it can be processed in parallel using word-based operations.

In this paper, we present a new software-based compression scheme. Like LFSR reseeding, it is also based on expanding compressed data using linear operations. However, it avoids many of the drawbacks of LFSR reseeding and can be implemented much more efficiently in software. The proposed approach performs pseudo-random word-based linear operations on compressed test data to decompress the test vectors. The compressed test data is obtained by solving linear equations for the specified bits in the test set. The advantages of the proposed technique include the following:

- The storage requirements are independent of s_{max}. They depend only on the total number of specified bits in the test set thereby providing a greater encoding efficiency. Moreover, no restrictions are placed on static compaction or the ATPG process as a whole. Both of these factors result in better compression.
- The proposed approach uses fewer operations for decompression and thus can be executed more efficiently on a processor with fewer instructions. This results in less test time.
- There is no need for primitive polynomials. This allows easy reuse of the same decompression program for multiple test sets.

Experimental results are presented to quantitatively show the advantages of the proposed approach.

IEEE COMPUTER SOCIETY

2. Overview of Proposed Method

The basis for the proposed compression/decompression method is similar to that used in LFSR reseeding [Koenemann 91]. In LFSR reseeding, an LFSR seed is expanded using linear operations to form a test vector. Each bit in the test vector can be represented by a linear equation in terms of the bits of the LFSR seed. Note that many of the bits in the test vector are don't cares, and thus the equations only need to be solved for specified bits. If s is the number of specified bits and m is the size of the LFSR, then a system of linear equations, $\boldsymbol{Ax=b}$, can be formed where $\boldsymbol{A}$ is the coefficient matrix with s rows and m column, $\boldsymbol{x}$ is a vector corresponding to the bits of the seed, and $\boldsymbol{b}$ is a vector corresponding to the specified bits in the test vector. The coefficient matrix $\boldsymbol{A}$ depends on the polynomial of the LFSR.

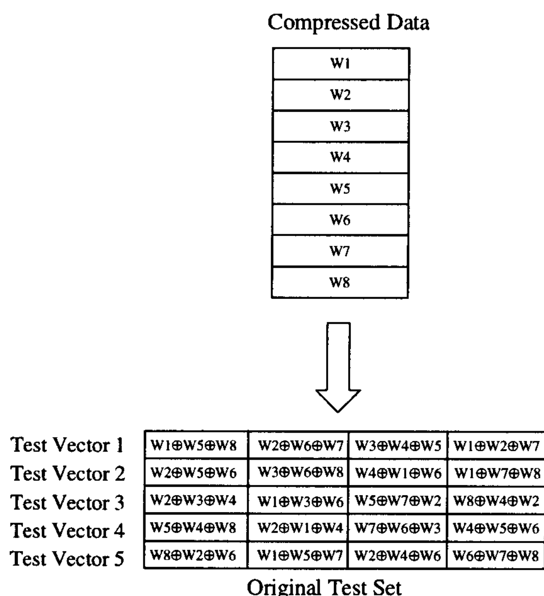

Figure 1. Forming test set from compressed bits

In the proposed method, the compressed data is expanded using word-based linear operations. A pseudo-random generator (e.g., the Mitchell and Moore additive generator [Knuth 97] which can very efficiently be implemented in software) is used select words from the compressed data that are XORed together to form each word of the test set (see example in Fig. 1). Each bit of the test set can be represented by a linear equation in terms of the bits in the compressed data. If s_{tot} is the total number of specified bits in the test set and m is the number of bits in the compressed data, then a system of linear equations, $\boldsymbol{Ax=b}$, can be formed where $\boldsymbol{A}$ is the coefficient matrix with s_{tot} rows and m column, $\boldsymbol{x}$ is a vector corresponding to the bits of the compressed data, and $\boldsymbol{b}$ is a vector corresponding to the specified bits in the test set. In this case, the coefficient matrix $\boldsymbol{A}$ depends on the sequence of numbers generated by the pseudo-random number generator.

The nice features of the proposed method compared with LFSR reseeding are the following: there is no need for primitive polynomials, so it can be easily scaled and adapted for any test set, the word-based linear operations can be efficiently executed on an embedded processor with a single XOR instruction, and the equations are solved across the entire test set so there is no dependence on the maximum number of specified bits, s_{max}, in any single test vector thereby allowing better compression.

In order to make the proposed compression method work, we need to be able to solve the system of linear equations for the care bits in the test set. If the rows of the coefficient matrix $\boldsymbol{A}$ are linearly independent, then the set of equations will definitely have a solution. We will now discuss what needs to be done to ensure the rows are linearly independent.

Suppose the coefficient matrix $\boldsymbol{A}$ is generated randomly, then Theorem 1 below applies.

Theorem 1: The probability that an $n \times m$ matrix ($m > n$) with entries chosen randomly from $\{0,1\}$ has all n rows independent is given by

$$\textbf{Pr [rows are independent]} > 1 - 2^{-(m-n)}$$

Proof: Define the event A_i to be the bad event that the first i-1 rows are linearly independent but the ith row is linearly dependent on the first i-1 rows.

Claim: Pr $[A_{i+1}] = 2^i / 2^m$.

Assume i linearly independent rows. There are 2^i possible linear combinations of these rows. Moreover, each of these 2^i linear combinations produces distinct vectors. If not, then there exists a distinct set of coefficients $\{a_j\}$ & $\{b_j\}$ which when applied to the row vectors $\{x_j\}$ yields:

$$\sum_{j=1}^{i} a_j x_j = \sum_{j=1}^{i} b_j x_j \Rightarrow \sum_{j=1}^{i} c_j x_j = 0, \quad c_j = a_j \oplus b_j$$

Since $\{c_j\}$ is not all zero as $\{a_j\}$ & $\{b_j\}$ are distinct, this contradicts our assumption that these rows are linearly independent. Hence there are 2^i distinct linear combinations. The $(i+1)^{\text{th}}$ row will be dependent on the first i rows if it is equal to any of the linear combinations. Therefore, Pr $[A_{i+1}] = 2^i / 2^m$ when the rows are chosen uniformly at random from a set of 2^m possibilities. Now,

$$\Pr\left[\bigcup_{i=1}^{n} A_i\right] \le \sum_{i=1}^{n} \Pr[A_i] = \sum_{i=0}^{n-1} 2^i / 2^m = (2^n - 1)/2^m < 2^{n-m}$$

Therefore,

$$\Pr[\text{ Rows are independent}] \ge 1 - \Pr\left[\bigcup_{i=1}^{n} A_i\right] > 1 - 2^{-(m-n)} \ \square$$

IEEE COMPUTER SOCIETY

For a randomly generated coefficient matrix A, if we want to make the probability of a solution not existing negligible, say 1 in a million, then based on Theorem 1, the number of columns should be 20 more than the number of rows. So if the number of rows is s_{tot}, then the number of columns should be s_{tot}+20.

In the proposed method, the coefficient matrix, A, is not truly random because a limited number of words are XORed together to form each word in the test set. However, the proposed method approximates a random coefficient matrix, and thus the number of columns (which is the number of words of compressed data times the word size) will be *roughly* s_{tot}+20 though somewhat more depending on how many XOR operations are performed when forming each word. Our experimental results (presented in Sec. 5) validate this.

3. Details of Proposed Method

The previous section described the main ideas of the proposed method. This section discusses some of the details.

If there are M words in the compressed data and N words in the original test set, the decompression procedure is as follows:

1. Generate a pseudo-random number, *rotateby*, between 0 and the word size.
2. Generate k random numbers from 1 to M.
3. For each random number generated in Step 2, select the word of the compressed set that is indexed by it and rotate the selected word by *rotateby*.
4. XOR all the words in Step 3 together to get the next word of the original test set.
5. Continue doing steps 1 to 4 until all the words in the original test set have been generated.

Note that the procedure rotates each selected word by a random number and then XORs them. This is done to mix up the bits and remove the dependency on the bit position.

To calculate the M words in the compressed set, a system of linear equations is formed and solved for the specified bits in the test set. The linear equations are formed by applying the decompression process using variables to represent the bits in the compressed words. If no solution can be found for the system of linear equations, then a larger value for M can be used, or a different value of k can be tried. If M is made sufficiently large, a solution can always be found.

Note that it may not be possible to process the entire test set altogether. There are two factors which may limit the number of test vectors that can be encoded with one set of compressed data. The first is that the amount of on-chip memory may be too small to hold the amount of compressed data required for the entire test set. The second is that the system of linear equations may become too large to solve in a reasonable amount of time. If these factors become an issue, then the test set can be simply partitioned and each partition processed one at a time. Partitioning the test set will reduce the overall compression slightly (the larger the partitions, the better the overall compression).

Using the proposed method to decompress the test vectors, as each word of the test vectors is decompressed; it can be directly applied to the scan chains of the CUT. One easy way for the processor to transfer the data to the scan chains is to make the scan chains a memory-mapped I/O port.

4. Implementation of Proposed Method

A software program for implementing the proposed method is shown in Fig. 2. The data required for the test program is the size of the scan vectors in the core-under-test (*scanSize*), the number of test vectors (*numVectors*), the number of words that need to be XORed to get one word (*numXors*), the seed of the random number generator (*randSeed*), and the number of words in the compressed set (*numWords*) and the compressed set itself (*compBits*).

```
Data: numXors    = Number of words that are XORed together
      numWords   = Total number of compressed words
      randSeed   = Seed of the random number generator
      scanSize   = Number of words in scan vector
      numVectors = Total number of test vectors
      compBits   = The compressed test vectors

Test_Procedure(){
/* Read numXors, numWords, randSeed, scanSize, numVectors */
    InitiateRandomGenerator(randSeed,numWords);
    for i = 1 to numVectors do
       for j = 1 to scanSize do
             word = 0;
             rotateby = (GenerateRandomNumber())mod 32;
             for k = 1 to numXors do
               rand = GenerateRandomNumber();
               tempword = compBits[rand];
               tempword = tempword ROT rotateby;
               word = word XOR tempword;
             end
             Apply(word);
       end
    Capture();
    end
}
```

Figure 2. Software program for proposed method

The random number generator used is the Mitchell and Moore additive generator [Knuth 97]. The generator produces a random number in $Z_m=\{0,1,2\ldots,m-1\}$ given by the equation:

$$X_n= (X_{n-24}+ X_{n-55}) \bmod m, \quad n \geq 55.$$

IEEE COMPUTER SOCIETY

This was shown to have good random characteristics [Knuth 97] and can be implemented very fast on processors [Hwang 02a]. The random number generator is initialized with values given by the random seed in *InitiateRandomGenerator* as shown in Fig. 3. The test program generates one word of a test vector at a time. For every word, (*1+numXors*) random numbers need to be generated. The first random number is *rotateby*, the amount of rotation. The rest of the random numbers denote the index of the selected word in the compressed set. Each selected word is first rotated according to *rotateby* and then XORed to get one word of the test vector. This word is then sent to the appropriate core and shifted into the scan chains. After one full test vector is generated and shifted, it is applied to the core.

```
InitiateRandomGenerator(rand_seed,num_words){
  for i = 1 to 55 do
     randset[i] = ( rand_seed + i ) % num_words;
  end
}
GenerateRandomNumber(){
 randset[i] = (randset[i-55] + randset[i-24]) % num_words;
 return(randset[i]);
}
```

Figure 3. Random Number Generation

The total test time for the test program shown in Fig. 2 will depend on the circuit parameters, e.g., number of test vectors, number of scan bits, etc. If we optimize the innermost loop that generates one word, there will be a corresponding decrease in the total test time. Fig. 4 below shows a piece of assembly code for the loop that is optimized for performance.

```
       LDI  R0, #0
       LDI  R1, #num  Xors   R1=numXors
       LDR  R6, [R3], #4     R6=mem[R3], R3+=4
       LDR  R7, [R4], #4     R7=mem[R4], R4+=4
       ADD  R6, R6, R7       R6+=R7
       ANDI R6, R6, #mod     R6&=mod
       STR  R6, [R5], #4     mem[R5]=R6
       AND  R3, R3, R2
       AND  R4, R4, R2
       AND  R5, R5, R2
       AND  R8, R6, #31      R8=R6&31

Loop:  LDR  R6, [R3], #4
       LDR  R7, [R4], #4
       ADD  R6, R6, R7
       ANDI R6, R6, #mod
       STR  R6, [R5], #4
       AND  R3, R3, R2
       AND  R4, R4, R2
       AND  R5, R5, R2
       LDR  R9, [R10 + R6]   R9= mem[R10+R6]
       ROR  R9, R8
       XOR  R0, R0, R9       R0=R0^R9
       SUBI R1, #1           R1=R1-1
       BNEZ R1, Loop
```

Figure 4. Optimized code for the inner loop

The generated word is stored in R0. R10 points to the starting address of the compressed bits while R6 contains the value of the random number generated. R3, R4, and R5 point to the *n-24*, *n-55* and *n*-th elements respectively in a cyclic list. Even though the size of the cyclic list required is 55 for the Mitchell and Moore generator, it can be implemented on 64 to optimize performance [Hwang 02a]. The list is assumed to start at an address that has 0s in the 8 least significant bits. The three AND instructions with R2 are used to keep the addresses from going over the boundary of the cyclic list.

5. Experimental Results

The proposed compression/decompression method was used to compress test vectors for the larger ISCAS 89 benchmark circuits. The results of these experiments are detailed in the following sections.

5.1 Program Size

The test program was implemented in 'C' language on a Pentium II processor. The program was compiled and then the executable disassembled to get the size of the program that needs to be transferred to the embedded processor. Even though a compact version of the program can be implemented in assembly language, the high-level compiled implementation should be a good indicator of the relative sizes. Table 1 compares the program size obtained with the program size for the methods in [Hellebrand 96], [Zacharia 96] and [Hwang 02b]. Programs implementing the algorithm given in [Hellebrand 96] and [Zacharia 96] were written in 'C' and the program sizes calculated similar to our method. For [Hellebrand 96], the size of the LFSR was assumed to be 64 bits since most of the circuits in that paper required a 64-bit LFSR. The length of the two-dimensional decompressor and the number of parallel LFSR's in [Zacharia 96] were taken as 52 and 32 respectively. The code size for the proposed scheme is much smaller than that of the other methods.

Table 1. Comparison of code size

Code Size	[Hellebrand 96]	[Zacharia 96]	[Hwang 02b]	Proposed Scheme
# instr.	151	152	253	87
# bytes	564	608	701	344

5.2 Data Compression

A commercial ATPG tool was used to generate test cubes with 100% fault coverage for each circuit. The unspecified input assignments were left as X's to enable better compression. We assumed a single scan chain for each circuit in our experiments. Table 2 shows the

1093-0167/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

compression results obtained using the proposed scheme with the number of XORs equal to 3. The number of test vectors and the total number of bits in the test data are shown for each circuit. The column "*Spec. Bits*" shows the number of specified bits in the test set. The column "*Comp. Bits*" has the number of bits in the compressed set. The percentage data compression was computed as:

$$\text{Percentage Data Compression} = \frac{\text{Original Bits - Compressed Bits}}{\text{Original Bits}} \times 100$$

Note that some additional bits are need for the code as shown in Table 1. These are not included in the equation above since the decompression program can be reused for multiple cores. The percentage compression would be slightly lower if the code size is included.

The encoding efficiency is the ratio of the specified bits to the compressed bits. The set of equations for the compressed set were solvable for all the circuits when the number of compressed words was taken to be 18 more than the number of specified bits.

Table 2. Compression results for proposed method

Circuit	Scan Size	Num. Test Vect.	Orig. Bits	Spec. Bits	Comp. Bits	% Comp.	Enc. Eff.
s5378	214	143	30602	5102	5696	81.4	0.896
s9234	247	146	36062	8677	9280	74.3	0.935
s13207	700	255	178500	9335	9920	94.4	0.941
s15850	611	148	90428	10567	11168	87.7	0.946
s38417	1664	105	174720	29847	30432	82.6	0.981
s38584	1464	131	191784	29610	30208	84.3	0.980

5.3 Comparison with Previous Methods

Table 3 shows a comparison of the compressed test data for the proposed method with other software-based compression methods for fully deterministic test sets - geometric primitives method [Maleh 01], incremental test vector construction [Jas 02], and matrix based decompression [Balakrishnan 02]. The size of the compressed test data for the proposed method is much smaller than the other methods.

Table 3. Comparison of compressed test data

Circuit	[Maleh 01]	[Jas 02]	[Balakrishnan 02]	Proposed Scheme
s5378	10057	12950	10390	5696
s9234	14666	18423	16888	9280
s13207	24446	24284	33470	9920
s15850	22458	24335	23552	11168
s38417	60478	97024	69556	30432
s38584	-	88376	66838	30208

Since both of the previous methods most related to this paper, [Hellebrand 96] and [Zacharia 96], are mixed mode techniques, we also ran experiments on mixed mode test sets for easy comparison. Like [Hellebrand 96] and [Zacharia 96], we first applied 10K random patterns to detect easy faults. Deterministic patterns were then generated for the remaining faults. In Table 4, we compare the compression obtained in the mixed mode version of our scheme with the results published in [Hellebrand 96] and [Zacharia 96].

The test data that needs to be transferred from the tester to the embedded processor is composed of both the test program and the compressed data. Hence for any compression scheme to be implemented using an embedded processor, the test data storage requirement should take into account both the code and data sizes. The numbers in brackets in Table 4 indicate the total test data storage requirements. The test data storage requirements for all the three methods were obtained by adding the code sizes from Table 1 to the data sizes. The proposed scheme requires less test data storage requirement than both the other methods.

Table 4. Comparison of test data storage requirements

Circuit	[Hellebrand 96]	[Zacharia 96]	Proposed Scheme
s5378	759 (5271)	-	544 (3296)
s9234	11766 (16278)	4576 (9440)	4448 (7200)
s13207	10796 (15308)	7488 (12352)	2880 (5632)
s15850	11826 (16338)	7392 (12256)	5152 (7904)
s38417	71491 (76003)	16640 (21504)	23136 (25888)
s38584	11529 (16041)	3584 (8448)	2912 (5664)

In Table 5, we compare the results of our fully deterministic scheme with [Hwang 02b]. The first two columns show the test data requirements. Even though our method is fully deterministic, the test data storage requirements of our method are similar to [Hwang 02b] for most of the circuits. The next two columns compare the test application time. The results for each method assume that the scan chains can be loaded as fast as the test data is decompressed, and thus the test application time is just equal to the running time of the decompression program. The test application time was calculated in the same way as described in [Hwang 02b]. The read timestamp register RDTSC instruction of x86 architecture was used to calculate the clock cycle count. The number of processor clock cycles is the difference between the values in the timestamp register before the program starts and after it ends. The test application time for the proposed method is smaller than the other method for most circuits.

IEEE COMPUTER SOCIETY

Table 5. Comparison with [Hwang 02b]

Circuit	Test Data Storage		Test Application Time	
	[Hwang 02b]	Prop. Sch.	[Hwang 02b]	Prop. Sch.
s5378	7885	8448	590,759	150,517
s9234	12058	12032	680,991	174,646
s13207	10412	12672	621,424	766,326
s15850	12514	13920	656,316	432,839
s38417	32715	33184	1,097,264	791,587
s38584	12563	32960	842,572	873,799

6. Conclusions

The proposed method efficiently performs linear expansion using word-based XOR instructions on an embedded processor. It provides a number of advantages compared with software-based LFSR reseeding including better compression, faster decompression (since it is word-based), and easy reusability (since no primitive polynomials are needed). The amount of compressed data required for the proposed method is very close to the total number of specified bits in the test set. The code size for the proposed method is very small, and the amount of compressed data processed at a time can be easily scaled to fit into whatever amount of on-chip memory is available by partitioning the test set appropriately.

Acknowledgement

This material is based on work supported in part by the Intel Corporation, in part by the National Science Foundation under Grant No. MIP-9702236, and in part by the Texas Advanced Technology Program under Grant No. 003658-0644-1999.

References

[Balakrishnan 02] Balakrishnan, K. J., and Touba, N. A., "Matrix Based Deterministic Test Vector Decompression Using an Embedded Processor", *Proc. Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, pp. 159-165, 2002.

[Dorsch 98] Dorsch, R., and H.-J. Wunderlich, "Accumulator Based Deterministic BIST", *Proc. of Int. Test Conference*, pp. 412-421, 1998.

[Gupta 94] Gupta, S., J. Rajski, and J. Tyszer, "Test Pattern Generation Based on Arithmetic Operations," *Proc. Int. Conf. on Computer-Aided Design*, pp. 117-124, 1994.

[Hellebrand 92] Hellebrand, S., Tarnick, S., Rajski J., and Courtois B., "Generation of Vector Patterns through Reseeding of Multiple Polynomial Linear Feedback Shift Registers", *Proc. of Int. Test Conf.*, pp. 120-129, 1992.

[Hellebrand 96] Hellebrand, S., Wunderlich, H.-J., and Hertwig, A., "Mixed mode BIST using an embedded processor", *Proc. Int. Test Conference*, pp. 195-204, 1996.

[Hwang 02a] Hwang, S, and Abraham, J. A., "Selective-run built-in self-test using an embedded processor", *Proc. Great Lakes Symp. on VLSI*, pp. 124-129, 2002.

[Hwang 02b] Hwang, S., and Abraham, J. A., "Optimal BIST Using an Embedded Microprocessor", *Proc. Int. Test Conference*, pp. 736-745, 2002.

[Ishida 98] Ishida, M., Ha, D.S.,Yamaguchi, T.,"COMPACT: A Hybrid Method for Compressing Test Data", *Proc. VLSI Test Symposium*, pp. 418-423, 1998.

[Iyer 02] Iyer, M.K., and K.-T. Cheng, "Software-Based Weighted Random Testing for IP Cores in Bus-Based Programmable SOCs," *Proc. VLSI Test Symposium*, pp. 139-144, 2002.

[Jas 02] Jas, A., and Touba, N. A., "Deterministic Test Vector Compression/Decompression for Systems-on-a-Chip Using an Embedded Processor", *Journal of Electronic Testing: Theory and Applications (JETTA)*, Vol. 18, Issue 4/5, pp. 503-514, Aug. 2002.

[Knuth 97] Knuth, D. E. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, MA, 3rd edition, 1997. ISBN: 0201896842.

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

[Maleh 01] El-Maleh, A., Al-Zahir,S., and Khan, E., "A Geometric Primitives Based Compression Scheme for Testing Systems-on-a-Chip," *Proc. VLSI Test Symposium*, pp. 540-59,2001.

[Rajski 93] Rajski, J., and J. Tyszer, "Accumulator-Based Compaction of Test Responses," *IEEE Transactions on Computers*, Vol. 42, No. 6, pp. 643-650, Jun. 1993.

[Rajsuman 99] Rajsuman, "Testing a System on a Chip with Embedded Microprocessor," *Proc. Int. Test Conference*, pp. 499-508, 1999.

[Saxena 98] Saxena, J., P. Ploicke, K. Cyr, A. Benavides, and M. Malpass, "Test Strategy for TI's TMS320AV7100 Device," *IEEE Int Workshop on Testing Embedded Core Based Systems*, 1998.

[Stroele 95] Stroele, A.P., "A Self-Test Approach Using Accumulators as Test Pattern Generators," *Proc. Int. Symp. on Circuits and Systems*, pp. 2010-2013, 1995.

[Stroele 96] Stroele, A.P., "Test Response Compaction Using Arithmetic Functions," *Proc. VLSI Test Symposium*, pp. 380-386, 1996.

[Stroele 98] Stroele, A.P., "Bit Serial Pattern Generation and Response Compaction Using Arithmetic Functions," *Proc. VLSI Test Symposium*, pp. 78-84, 1998.

[Yamaguchi 97] Yamaguchi, T., Tilgner, M., Ishida, M., and Ha, D. S., "An Efficient Method for Compressing Test Data," *Proc. Int. Test Conference*, pp. 191-197, 1997.

[Zacharia 96] Zacharia, N., Rajski, J., Tyszer, J., and Waicukauski J.A., "Two-Dimensional Test Data Decompression for Multiple Scan Designs" *Proc. of Int. Test Conference*, pp. 186-194, 1996.

[Zorian 98] Zorian, Y., Marinissen, E. J., Dey S., "Testing embedded core based system chips", *Proc. of Int. Test Conference*, pp. 130-143, 1998.

1093-0167/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY