Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Journal of Systems Architecture 50 (2004) 247–256

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Matrix-based software test data decompression for systems-on-a-chip

Kedarnath Balakrishnan, Nur A. Touba *

Computer Engineering Research Center, Department of Electrical and Computer Engineering, Engineering Science Building, University of Texas at Austin, Austin, TX 78712-1084, USA

Abstract

This paper describes a new compression/decompression methodology for using an embedded processor to test the other components of a system-on-a-chip (SoC). The deterministic test vectors for each core are compressed using matrix-based operations that significantly reduce the amount of test data that needs to be stored on the tester. The compressed data is transferred from the tester to the processor's on-chip memory. The processor executes a program which decompresses the data and applies it to the scan chains of each core-under-test. The matrix-based operations that are used to decompress the test vectors can be performed very efficiently by the embedded processor thereby allowing the decompression program to be very fast and provide high throughput of the test data to minimize test time. Experimental results demonstrate that the proposed approach provides greater compression than previous methods.

1. Introduction

Systems-on-a-chip (SoCs) have become ubiquitous nowadays because of the advances in design technology that make it possible to build complete systems containing different types of components (called cores) on the same chip. Typically, these cores are pre-designed and pre-verified by their vendors and the SoC designer has to just integrate them in the system. Each core typically comes with its own set of test patterns and hence the SoC on the whole ends up with a large set of test vectors. This translates into two major issues that affect the testing of such SoCs. First, there is an increase in tester memory requirements since the entire set of test patterns for all the cores need to be stored on the tester and transferred to the chip during testing. Next, it results in longer test application times since the time required to test a chip depends on the amount of test data that needs to be transferred and the bandwidth of the channel connecting the tester to the chip.

These problems result in higher test costs since the cost of automated test equipment (ATE) scale rapidly with memory requirements. Built-in-self-test (BIST) has been employed elsewhere to reduce dependency on expensive ATEs. But it may not be applicable in such scenarios since it requires the core-under-test to be BIST-able. This would also require a lot more information about the cores, which may not be possible if the cores are

* Corresponding author. Tel.: +1-512-232-1456; fax: +1-512-471-5532.

E-mail addresses: kjbala@ece.utexas.edu (K. Balakrishnan), touba@ece.utexas.edu (N.A. Touba).

1383-7621/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2003.08.007

intellectual property (IP) cores. For cores that do not come with their own BIST structure, it will be difficult to generate patterns on-chip in a cost-effective manner. The test patterns of such cores that come with functional tests or automatic test pattern generator (ATPG) generated tests are often irregular in structure and cannot be generated on-chip at acceptable area costs.

Test data reduction by compressing the test vectors is another possible solution to this problem. The compressed data is stored on the tester and transferred to the SoC where an on-chip decompressor decompresses the test data and applies it to the appropriate core. The decompression can be done in hardware using a dedicated decompressing circuit or in software by an embedded processor. In software decompression, a decompression program is transferred to an on-chip memory along with the compressed test data. The embedded processor then executes this program, which decompresses the test vectors and applies it to the core-under-test.

A number of test data reduction techniques have been proposed in the literature. Most use special decompression hardware on the chip [2,3,9,10,12,16]. In this paper, we focus on software-based test data compression techniques that are applicable to deterministic test vectors since the test vectors for a core in a SoC are usually precomputed by the vendor.

Techniques for using embedded processors to perform memory tests have been proposed in [14,17], while using the processor for generating patterns and compacting test responses in pseudorandom built-in-self-test (BIST) have been proposed in [4,5,8,15,18–20]. Software techniques based on a combination of the Burrows–Wheeler transformation and modified run length encoding have been proposed in [7] and [21] to reduce the time needed to download the test patterns across a network to the tester. However, these techniques are not suited for on-chip decompression since the decompression process is complex and time consuming. Decompression schemes that can be implemented using an embedded processor have been proposed in [11,13]. In [13], encoding is done on optimally reordered test vectors based on geometric shapes. The software decompression algorithm required to get back the original test vectors by this method is very complex and is a limitation of the approach. In [11], each test vector is divided into blocks and only those blocks that are different from the preceding vector are stored. Test vectors are then constructed on the fly by a program running on the embedded processor and sent to the appropriate core.

In this paper, we present an efficient compression/decompression scheme for deterministic test vectors with unspecified bits based on matrix operations. The decompression algorithm is much simpler than earlier methods and an embedded processor present in the SoC can efficiently implement it with relatively fewer processor instructions. Note that we assume that the processor itself has been tested prior to its use by self-test or other means and the proposed technique is applicable to test all the other cores in the SoC.

The organization of this paper is as follows. The proposed compression scheme is described in Section 2. The decompression architecture and algorithm is discussed in Section 3. An analysis of the decompression time and buffer size is given in Section 4. Experimental results on benchmark circuits are presented in Section 5. Conclusions are given in Section 6.

2. Proposed scheme

The proposed scheme is based on the decomposition of a matrix into two vectors based on a relation that we define below. The operation $\boldsymbol{A}\widetilde{\oplus}\boldsymbol{B}$ between two boolean vectors $\boldsymbol{A} = [a_1, a_2, a_3, \ldots, a_n]$ and $\boldsymbol{B} = [b_1, b_2, b_3, \ldots, b_n]$ where $a_i, b_i \in \{0, 1\}$ is defined as shown in Fig. 1. Note that this is very similar to matrix multiplication except that the elements in the product matrix are defined differently ($a_i \oplus b_i$ instead of $a_i \bullet b_i$). This helps increase the chances of decomposition since the *XOR* operation puts less constraints on the inputs than *AND* by making the equations linear.

In this way, an $n \times n$ matrix can be represented with the two vectors $\boldsymbol{A}$ and $\boldsymbol{B}$ and the operation $\boldsymbol{A}\widetilde{\oplus}\boldsymbol{B}$. This decomposition can be realized by solving a simultaneous set of equations in the variables a_i, b_i. The set of equations is represented

$$A \widetilde{\oplus} B = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \widetilde{\oplus} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1 \oplus b_1 & a_1 \oplus b_2 & a_1 \oplus b_3 \\ a_2 \oplus b_1 & a_2 \oplus b_2 & a_2 \oplus b_3 \\ a_3 \oplus b_1 & a_3 \oplus b_2 & a_3 \oplus b_3 \end{bmatrix}$$

Fig. 1. Matrix operation $A \widetilde{\oplus} B$.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{bmatrix}$$

Fig. 2. Set of linear equations for the decomposition.

in the matrix format ($Ax = b$) as shown in Fig. 2. If and only if a solution of this set of equation exists, the given matrix can be decomposed.

We propose a compression scheme for test vectors based on the above decomposition. This involves writing n^2 bits of the test vector as an $n \times n$ matrix M. The matrix M is then decomposed by solving the set of linear equations. Although the decomposition in not always possible, the unspecified bits in the test vectors increase the chances of decomposition. This is because only equations for the specified bits of the test vector need to be satisfied. The more unspecified bits there are, the fewer the number of equations and hence less constraints on the variables. If the matrix is not decomposable, then the next few bits of the test vector are stored as is (uncompressed) and the algorithm proceeds with the next set of bits. Hence at least one bit is required at the beginning of each set of compressed bits to indicate whether the bits are compressed or not compressed. Several different heuristics can be applied to form the matrix M that needs to be decomposed from the given set of test vectors. They vary according to the complexity of the decompressing process. Three are discussed below.

2.1. Single size decomposition

This is the simplest method to form the matrix M and also the easiest to decode. The first n^2 bits of the test vector are written as an $n \times n$ matrix with the first n bits being the first row of the matrix and the next n bits the next row and so on and so forth as illustrated in Fig. 3.

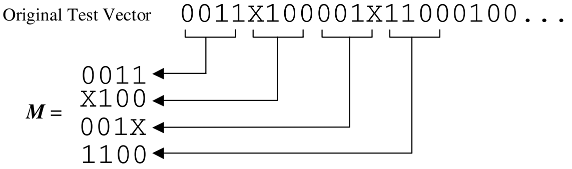

Fig. 3. Formation of matrix M with $n = 4$.

The choice of the size of the matrix (i.e., n) would depend on the word size of the processor. If the matrix M thus formed cannot be decomposed for the next n^2 bits of the test vector, we store the first n bits as they are (i.e. uncompressed) and then proceed with the algorithm for the next n^2 bits after that. Hence we need one bit at the start of every set of bits to indicate whether the bits are compressed or not. In this method each test vector is compressed separately and hence the ordering of the test vectors will not make a difference in the compression obtained.

2.2. Multiple size decomposition

A simple optimization of the method in Section 2.1 would be to try to form the matrix M with multiple sizes. The largest size is tried first since that gives the maximum compression. If the matrix for the largest size is not decomposable, then the next size is tried and so on. If none of the different sizes of the matrix are decomposable, then the next set of m bits are left uncompressed and the algorithm is tried on the successive bits of the test vector. If the number of different sizes is k, then $\lceil \log_2 (k+1) \rceil$ additional bits are needed to encode the $k+1$ cases that can occur indicating whether the succeeding bits have been compressed by any of the different sizes or they are left uncompressed.

2.3. Multiple vector decomposition

The matrix M can also be formed from multiple test vectors as illustrated in Fig. 4. In this case, each row of the matrix would correspond to a

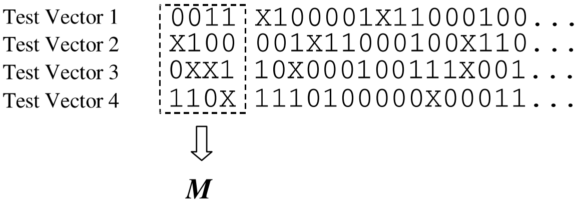

Fig. 4. Formation of matrix ***M*** using MVD.

different test vector. This is a better alternative than the earlier ones since the test vectors can be ordered in an optimal way such that the chances of decomposition of the matrix are increased. A limiting factor of this method is that the decoding is more complex and the partial test vectors constructed after decoding each matrix cannot be directly applied to the core-under-test until a sufficient number of matrices have been decoded to get the complete test vector.

The amount of compression obtained in this scheme depends on the ordering of the test vectors. Since it is impractical to examine all possible orderings (a set of n test vectors has $n!$ different ways of ordering), we have used a greedy reordering heuristic referred to as the hill climbing approach in our experiments. In the hill climbing approach, initially the given order of test vectors is used to calculate the compression. The positions of two vectors are then randomly exchanged and the new compression calculated. If the compression is better, the new order is saved; else, the old order is maintained. This process is continued until no better compression is obtained for a specific number of exchanges. Since this procedure is done during the compression process, it does not affect the decompression procedure in any way.

3. Decompression using an embedded processor

An embedded processor present in the SoC can be used to efficiently decompress the compressed data and send it to the core-under-test (CUT). This is illustrated in Fig. 5. An external tester supplies the compressed data while a simple software program running on the embedded processor decodes the test data. The tester loads the test data into a specific set of addresses of the system memory through the memory I/O controller. The tester also writes to a given location to indicate the end of the current test vector. The processor reads the data from the corresponding locations in memory and decompresses it accordingly. Depending on the number of scan chains in the core, the processor either sends the data directly to the core or stores it back to memory so that it can apply the data to the core when it has a sufficient amount of decompressed test data. If the end of

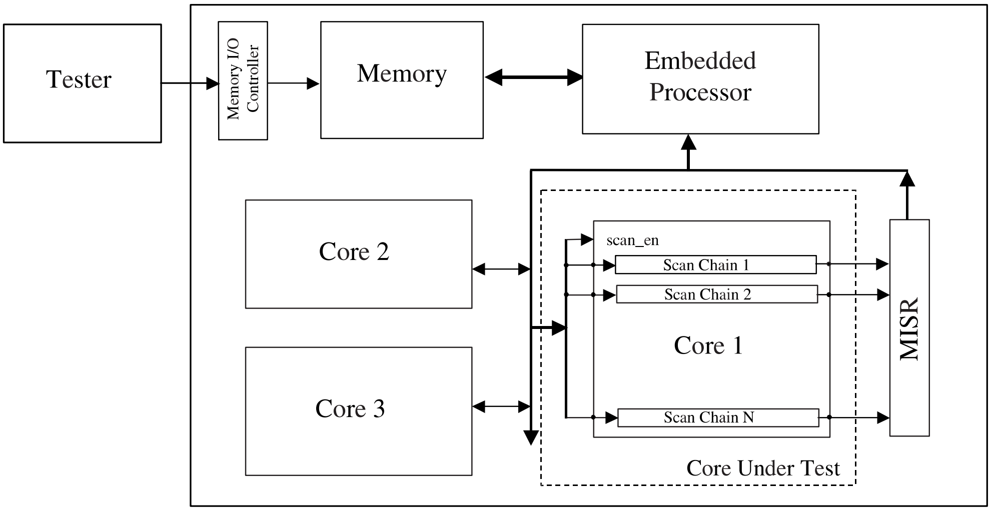

Fig. 5. Example of test architecture.

the test vector is reached, it sends an instruction to apply a capture cycle to capture the response into the scan chain. The response is shifted out into a multi-input shift register (MISR) for compaction as the next test vector is shifted into the core.

Fig. 6 shows the *pseudo-code* of the test procedure for the multiple size decomposition (MSD) heuristic when three different sizes (n_1, n_2, n_3) are used. In this case, the first two bits denote which of the four cases apply to the next set of bits. Hence, we first check them and proceed accordingly. The macro *Apply* writes the test vector to the appropriate core. It will depend on the number of scan chains in the core and the width of the bus connecting the processor to the core. The macro will exit after receiving a signal that the decompressed vectors have been scanned into the core and the program proceeds to decompress the next set of bits. After each test vector is decompressed and sent to the scan chains, a capture signal is given to the core-under-test by the macro *Capture*.

```
Data:  numVectors := number of testvectors
       ScanSize := size of the scan chain
       n1, n2, n3 := the three matrix sizes
       m := number of uncompressed bits

Test_procedure() {
  while(vectorsProcessed < numVectors)
      while(bitsProcessed < scanSize)
        readNextMemoryLocation();
        /*  check the first two bits */
        case 00:
          A[1.. n1] = next n1 bits
          B[1.. n1] = next n1 bits
          for (i = 1, n1){
             M[i][0.. n1] = A[i]⊕B[0.. n1];
             Apply(M[i][0.. n1]);
        end
        case 01:
          A[1.. n2] = next n2 bits
          B[1.. n2] = next n2 bits
          for(i = 1, n2){
             M[i][0.. n2] = A[i]⊕B[0.. n2];
             Apply(M[i][0.. n2]);
        end
        case 10:
          A[1.. n3] = next n3 bits
          B[1.. n3] = next n3  bits
          for(i = 1, n3){
             M[i][0.. n3] = A[i]⊕B[0.. n3];
             Apply(M[i][0.. n3]);
        end
        case 11:
          M[0..m] = next m bits
          Apply(M[0..m]);
        end
      end while
      Capture();
      vectorsProcessed++;
    end while
 }
```

Fig. 6. *Pseudo-code* of decompression algorithm for MSD scheme.

The implementation of decompression algorithm can be made more efficient by appropriately choosing the matrix sizes according to the word size of the embedded processor. For single size decomposition (SSD), the matrix size, s, should be such that $s + 1$ is equal to the word size of the processor. This is because we store either $2s + 1$ bits (if matrix can be decomposed) or $s + 1$ bits (if matrix cannot be decomposed). In the case of MSD, the three different sizes, s_1, s_2, s_3 and the number of bits left uncompressed, m, should be such that $2s_1 + 2$, $2s_2 + 2$, $2s_3 + 2$, and $m + 2$ are multiples of the word size of the processor. This would minimize the number of reads from the system memory (since each set of bits will be at word boundaries) and hence reduce the decompression time.

4. Analysis

The number of cycles that the decompression program runs, n, can be determined by studying the time complexity of the different schemes. For example, in the SSD heuristic, this time will depend on the number of solvable and unsolvable matrix decompositions during compression. If the number of solvable and unsolvable decompositions are n_s and n_u respectively, then the number of cycles will be

$$n = n_s c_1 + n_s s c_2 + n_u c_3 + c_4$$

where s is the size of the matrix and c_1, c_2, c_3, and c_4 are constants depending on the instruction set. These constants are essentially the number of cycles it takes for the processor to read from the memory, perform XOR operations and to write to the core-under-test. We performed simulations on the ISCAS 89 [1] benchmark circuits to measure the number of cycles required for decompression. The decompression program for the SSD with size $s = 7$ was written in 'C' language and implemented on the ARM7 architecture. The SimpleScalar toolset was then used to run the program and estimate the number of processor clock cycles. Table 1 shows the number of solvable and unsolvable decompositions and the number of clock cycles obtained by simulation for the ISCAS 89 benchmark circuits. The values for the constants c_1, c_2, c_3, and c_4 can be calculated for the ARM7 from the simulated results. A similar analysis can be done for the other two heuristics to get the time complexity of the proposed schemes.

In general, the rate at which the tester transfers test data to the SoC will be different from the rate at which the embedded processor processes the compressed test data. The former depends on the tester clock cycle and the number of channels from the tester to SoC, while the latter depends on the operating frequency of the processor and its instruction set architecture. Two potential problems could arise because of this discrepancy. If the processor is able to process the written data before the tester loads new data into the memory location, it has to wait until the tester writes new data into the location. The second problem arises if the tester overwrites new data to a memory location before the processor is able to process the old data in it. These problems can be taken care of by inserting *NOPs* at appropriate places into the decompression program or the tester program to slow it, but this solution will result in an increase in the test time. Alternatively, by choosing the size of the buffer (set of memory locations where the tester writes the data in a cyclic fashion) and the start time for the processor appropriately, it may be possible to circumvent this problem. In this case, there is no need for synchronization between the tester and the processor—the tester keeps on writing to the buffer at its own speed while the processor keeps on decompressing data at its own rate. The analysis for calculating the buffer size for the SSD is given below.

Table 1
Simulated values for the number of cycles

Circuit	No. of solvable	No. of unsolvable	Number of cycles
s5378	437	769	24013
s9234	583	1204	29373
s13207	3232	1580	99430
s15850	1264	1475	47870
s38417	2350	4523	84569
s38584	3140	4185	104433

The number of cycles the processor takes to process each compressed and each uncompressed word can be determined through simulation. For the ARM7 processor, the number of cycles were 4 for each uncompressed word and 26 for each compressed set (two words) when the matrix size was equal to 7. This means if the current word is an uncompressed word, the processor will access the next word after 4 cycles or if the current word is a compressed word, the processor will access the word subsequent to the next word after 26 cycles (since it uses two words together if they are compressed). Using this and the trace of compressed/uncompressed words, the cycles in which the processor will access the buffer can be determined. If each word in buffer is guaranteed to be valid when the processor accesses it, then there is no need for explicit synchronization. Each word will be valid during the time interval between end of the previous loading of the tester and start of the next loading of the tester. This time interval will depend on the buffer size and the data transfer rate of tester. The minimum buffer size for which each accessed word is valid (with an appropriate start time for the processor) can be calculated for different tester data transfer rates.

Table 2 shows the buffer sizes required for each of the ISCAS 89 benchmark circuits for different tester speeds with 7 as the size of the matrix for decomposition. The column labeled “total data” is the total number of bytes in the compressed set. This is shown to compare the required buffer sizes with the total amount of memory required to store the entire compressed test set. The other columns show the size of the buffer (in bytes) required so that no synchronization is needed between tester and processor. The tester data transfer rate of 0.5 means the tester transfers 0.5 bits per processor clock cycle (i.e., 1 bit every 2 processor clock cycles). The tester transfer rate depends on the tester clock speed and the number of channels. Note from the results in Table 2 that the buffer size first decreases with an increase in tester rate and then increases again. If the tester is too slow, a larger buffer is required and the tester needs to start writing much earlier than the processor can start

Table 2
Buffer sizes for different data transfer rates of the tester

Circuit	Total data	Tester data transfer rate (bits per processor clock cycle)				
		0.5	0.75	0.875	1.0	1.25
s5378	1643	743	298	237	340	834
s9234	2370	1124	485	367	390	1019
s13207	8044	2399	1440	2347	3505	7122
s15850	4033	1583	579	724	1112	2503
s38417	9223	4276	1718	1266	1489	4164
s38584	10465	4319	1766	1969	3069	6546

processing. If the tester is too fast, again a larger buffer is required since the tester must not overwrite memory before it is processed.

5. Experimental results

The proposed compression schemes were implemented and experiments were performed on the larger ISCAS 89 [1] and ITC99 [6] benchmark circuits. The following sections discuss the results in detail.

5.1. Program size

In software based testing, the decompression program also needs to be transferred to the chip. Hence, the program size is required to calculate the total amount of data that needs to be transferred. In these experiments, the decompression program was implemented in ‘C’ language and cross-compiled for the ARM7 architecture. Table 3 shows the program size for the three heuristics. As expected, the SSD requires the least number of instructions to decode. The multiple vector decomposition (MVD) heuristic requires the maximum number of instructions among the three heuristics.

Table 3
Program size for different schemes

Program size	SSD	MSD	MVD
# instr.	100	294	475
# bytes	400	1176	1900

5.2. Data compression

Test cubes that provide 100% coverage of detectable faults were generated using the SynTest™ commercial ATPG tool for each circuit. The unspecified input assignments were left as Xs for better compression. The three heuristics described in Section 2 were used to compress the test set. The percentage compression is computed as

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

Table 4 shows the compression obtained using the SSD for three different matrix sizes. The first four columns have the details of the benchmark circuit including circuit name, number of scan elements, number of test vectors in the test set, and total number of bits in the test set. The remaining columns show the compressed test set size and the corresponding percentage compression for the three different sizes of the matrix. The matrix size under which maximum compression is obtained depends on the size of the circuit. As can be seen from the table, for the smaller circuits, a matrix size of $n = 7$ gives maximum compression, while for some of the larger circuits, $n = 10$ gives better compression. But as n increases further, the compression decreases since very few of the matrices are decomposable.

Table 5 compares the compression obtained using the MSD heuristic with three sizes for three different configurations. The first configuration has the three matrix sizes $n_1 = 15$, $n_2 = 7$, $n_3 = 3$ and the number of bits left uncompressed, $m = 14$. The next configuration has sizes $n_1 = 31$, $n_2 = 15$,

Table 4
Compression obtained using SSD

Circuit	Scan size	Test vectors	Original bits	$n = 7$		$n = 10$		$n = 13$	
				Comp. bits	Percent comp.	Comp. bits	Percent comp.	Comp. bits	Percent comp.
s5378	214	119	25466	12707	**50.1**	16026	37.1	22869	10.2
s9234	247	147	36309	18377	**49.4**	24386	32.8	31925	12.1
s13207	700	239	167300	61120	63.5	48766	**70.9**	52596	68.6
s15850	611	120	73320	30760	**58.1**	35842	51.1	45997	37.3
s38417	1664	95	158080	71434	**54.8**	98904	37.4	141437	10.5
s38584	1464	131	191784	80580	**58.0**	94300	50.9	127191	33.7
b14s	281	110	30910	14000	**54.7**	17865	42.2	20977	32.1
b15s	489	240	117360	41041	65.0	36606	**68.8**	41986	64.2
b17s	1456	233	339248	120124	64.6	112454	**66.9**	135822	60.0
b20s	526	325	170950	70980	**58.5**	86720	49.3	108800	36.4
b21s	526	325	170950	80170	**53.1**	109764	35.8	153395	10.3
b22s	771	324	249804	99774	**60.1**	111829	55.2	138174	44.7

Table 5
Compression obtained using MSD

Circuit	Original bits	$n_1 = 15$, $n_2 = 7$, $n_3 = 3$ and $m = 14$		$n_1 = 31$, $n_2 = 15$, $n_3 = 7$ and $m = 14$		$n_1 = 23$, $n_2 = 15$, $n_3 = 7$ and $m = 14$	
		Comp. bits	Percent comp.	Comp. bits	Percent comp.	Comp. bits	Percent comp.
s5378	25466	12504	**50.9**	13216	48.1	13264	47.9
s9234	36309	19480	**46.3**	20880	42.5	20976	42.2
s13207	167300	40512	75.8	29088	**82.6**	35504	78.8
s15850	73320	26680	**63.6**	27584	62.4	29200	60.2
s38417	158080	72864	**53.9**	82240	48.0	82640	47.7
s38584	191784	72048	**62.4**	74800	61.0	75360	60.7
b14s	30910	13560	**56.1**	13856	55.2	14352	53.6
b15s	117360	31592	73.1	24432	**79.2**	24976	78.7
b17s	339248	84880	75.0	75360	77.8	75152	**77.9**
b20s	170950	65744	61.5	63040	**63.1**	65488	61.7
b21s	170950	86440	**49.4**	90928	46.8	92016	46.2
b22s	249804	84312	66.2	82992	66.8	82576	**67.0**

$n_3 = 7$ and $m = 14$, while the last configuration has sizes $n_1 = 23$, $n_2 = 15$, $n_3 = 7$ and $m = 14$. These sizes have been chosen such that the decompression program can fetch either one byte (8 bits) or multiples of bytes from system memory and operate on it.

In Table 6, the compression obtained using MVD heuristic is presented. Three different values for the number of vectors compressed together, N, were tried. The hill climbing approach discussed in Section 2.3 was implemented. In general, the compression obtained increases with the number of vectors compressed together. Compressing eight vectors at a time ($N = 8$) gives better compression for most of the circuits. However, the complexity of the decompression program will increase with N. The amount of on-chip memory required for decompression will also increase since N test vectors need to be stored at a time.

A comparison of the best compression obtained using the three different heuristics is given in Table 7. From the results, it can be seen that the MVD scheme has the best compression ratio for most of the circuits but it is also the most complex to de-

Table 6
Compression obtained using MVD

Circuit	Original bits	$N = 4$		$N = 6$		$N = 8$	
		Comp. bits	Percent comp.	Comp. bits	Percent comp.	Comp. bits	Percent comp.
s5378	25466	11628	54.3	10374	59.3	9630	**62.2**
s9234	36309	17206	52.6	16858	53.6	15986	**56.0**
s13207	167300	59970	64.2	44790	73.2	37256	**77.7**
s15850	73320	30018	59.1	26292	64.1	24408	**66.7**
s38417	158080	73006	53.8	68496	**56.7**	68576	56.6
s38584	191784	76100	60.3	66092	**65.5**	66534	65.3
b14s	30910	13982	54.8	12680	59.0	12096	**60.9**
b15s	117360	44052	62.5	34673	70.5	29796	**74.6**
b17s	339248	132820	60.8	106988	68.5	97502	**71.3**
b20s	170950	72100	57.8	66274	61.2	64284	**62.4**
b21s	170950	83198	51.3	80722	**52.8**	81980	52.0
b22s	249804	100508	59.8	88818	64.4	88690	**64.5**

code among the three methods discussed. The test application time will be the longest in this case since the test vectors need to be constructed completely and stored in the system memory before they can be applied to the core. The MSD heuristic obtains a good amount of compression and is relatively simpler to decode. The choice of which method to apply will depend on the operating conditions of the test architecture. If test application time is critical, MSD is a better alternative.

Table 7
Comparison of the three methods

Circuit	Original bits	Percent. comp.		
		SSD	MSD	MVD
s5378	25466	50.1	50.9	62.2
s9234	36309	49.4	46.3	56.0
s13207	167300	70.9	82.6	77.7
s15850	73320	58.1	63.6	66.7
s38417	158080	54.8	53.9	56.7
s38584	191784	58.0	62.4	65.5
b14s	30910	54.7	56.1	60.9
b15s	117360	68.8	79.2	74.6
b17s	339248	66.9	77.9	71.3
b20s	170950	58.5	63.1	62.4
b21s	170950	53.1	49.4	52.8
b22s	249804	60.1	67.0	64.5

Table 8
Comparison with other techniques

Circuit	[13]	[11]	Prop. scheme
s5378	51.6	49.1	62.2
s9234	43.5	49.3	56.0
s13207	85.0	85.5	82.6
s15850	60.9	66.8	66.7
s38417	46.6	38.6	56.7
s38584	–	53.9	65.5

In Table 8, the compression results for the proposed scheme are compared with the geometric primitives based compression technique described in [13] and the difference vectors based compression described in [11]. As seen from the table, the compression obtained by the proposed scheme is higher than both of the earlier methods for almost all of the circuits.

6. Conclusions

In this paper, an efficient test vector compression method was proposed that utilizes the power of an embedded processor present on the chip to help in testing the cores in the SoC. The decompression is performed in software by a program running on the embedded processor. Hence both test storage and test time is reduced. The compression scheme described in this paper is very simple compared with methods proposed earlier, and hence requires less software code and allows faster decompression.

References

[1] F. Brglez, D. Bryan, K. Kozminski, Combinational profiles of sequential benchmark circuits, in: Proc. International Symposium on Circuits and Systems, 1989, pp. 1929–1934.

[2] A. Chandra, K. Chakrabarty, Test data compression for system-on-a-chip using Golomb codes, in: Proc. of VLSI Test Symposium, 2000, pp. 113–120.

[3] A. Chandra, K. Chakrabarty, Frequency-directed run length (FDR) codes with application to system-on-a-chip test data compression, in: Proc. of VLSI Test Symposium, 2001, pp. 42–47.

[4] R. Dorsch, H.-J. Wunderlich, Accumulator based deterministic BIST, in: Proc. of International Test Conference, 1998, pp. 412–421.

[5] S. Gupta, J. Rajski, J. Tyszer, Test pattern generation based on arithmetic operations, in: Proc. of International Conference on Computer-Aided Design (ICCAD), 1994, pp. 117–124.

[6] ITC 99 Benchmarks, http://www.cerc.utexas.edu/itc99-benchmarks/bench.html.

[7] M. Ishida, D.S. Ha, T. Yamaguchi, COMPACT: a hybrid method for compressing test data, in: Proc. VLSI Test Symposium, 1998, pp. 418–423.

[8] M.K. Iyer, K.-T. Cheng, Software-based weighted random testing for IP cores in bus-based programmable SoCs, in: Proc. IEEE VLSI Test Symposium, 2002, pp. 139–144.

[9] A. Jas, N.A. Touba, Test vector decompression via cyclical scan chains and its application to testing core-based designs, in: Proc. of IEEE International Test Conference, 1998, pp. 458–464.

[10] A. Jas, J. Ghosh-Dastidar, N.A. Touba, Scan vector compression/decompression using statistical coding, in: Proc. of IEEE VLSI Test Symposium, 1999, pp. 114–120.

[11] A. Jas, N.A. Touba, Deterministic test vector compression/decompression for systems-on-a-chip using an embedded processor, Journal of Electronic Testing: Theory and Applications (JETTA) 18 (4/5) (2002) 503–514.

[12] B. Könemann, A SmartBIST variant with guaranteed encoding, in: Proc. of Asian Test Symposium, 2001, pp. 325–330.

[13] A. El-Maleh, S. Al-Zahir, E. Khan, A geometric primitives based compression scheme for testing systems-on-a-chip, in: Proc. IEEE VLSI Test Symposium, 2001, pp. 540–559.

[14] R. Rajsuman, Testing a system on a chip with embedded microprocessor, in: Proc. Int. Test Conference, 1999, pp. 499–508.

[15] J. Rajski, J. Tyszer, Accumulator-based compaction of test responses, IEEE Transactions on Computers 42 (6) (1993) 643–650.

[16] J. Rajski et al., Embedded deterministic test for low cost manufacturing test, in: Proc. of International Test Conference, 2002, pp. 301–310.

[17] J. Saxena, P. Ploicke, K. Cyr, A. Benavides, M. Malpass, Test strategy for TI's TMS320AV7100 Device, in: IEEE Int. Workshop on Testing Embedded Core Based Systems, 1998.

[18] A.P. Stroele, A self-test approach using accumulators as test pattern generators, in: Proc. of International Symposium on Circuits and Systems, 1995, pp. 2010–2013.

[19] A.P. Stroele, Test response compaction using arithmetic functions, in: Proc. of VLSI Test Symposium, 1996, pp. 380–386.

[20] A.P. Stroele, Bit serial pattern generation and response compaction using arithmetic functions, in: Proc. of VLSI Test Symposium, 1998, pp. 78–84.

[21] T. Yamaguchi, M. Tilgner, M. Ishida, D.S. Ha, An efficient method for compressing test data, in: Proc. International Test Conference, 1997, pp. 191–197.

Kedarnath Balakrishnan received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, India in 2000, the M.S. degree in electrical and computer engineering from the University of Texas at Austin in 2002, and is working towards the Ph.D. degree at the same university. His research interests are in the fields of VLSI design and testing and computer architecture. He is currently working in the areas of test data reduction and embedded processor based system-on-a-chip testing.

Nur A. Touba is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin. He received his B.S. degree from the University of Minnesota, and his M.S. and Ph.D. degrees from Stanford University all in electrical engineering. He received a National Science Foundation (NSF) Early Faculty CAREER Award in 1997, and the Best Paper Award at the 2001 VLSI Test Symposium. He is on the program committee for the International Test Conference, International Conference on Computer Design, Design Automation and Test in Europe Conference, International On-Line Test Symposium, International Test Synthesis Workshop, and European Test Workshop.