Nine-Coded Compression Technique for Testing Embedded Cores in SoCs

Mohammad Tehranipoor, *Member, IEEE*, Mehrdad Nourani, *Senior Member, IEEE*, and Krishnendu Chakrabarty, *Senior Member, IEEE*

***Abstract*—This paper presents a new test-data compression technique that uses exactly nine codewords. Our technique aims at precomputed data of intellectual property cores in system-on-chips and does not require any structural information of cores. The technique is flexible in utilizing both fixed- and variable-length blocks. In spite of its simplicity, it provides significant reduction in test-data volume and test-application time. The decompression logic is very small and can be implemented fully independent of the precomputed test-data set. Our technique is flexible and can be efficiently adopted for single- or multiple-scan chain designs. Experimental results for ISCAS'89 benchmarks illustrate the flexibility and efficiency of the proposed technique.**

***Index Terms*—Capacitor switching, data compression, digital system testing, Huffman codes, integrated circuit testing, power demand, run length codes.**

I. Introduction

TESTING today's system-on-chip (SoC) circuits is a challenge due to several limiting factors. These factors include large volume of test data, long test application time, high power consumption during test, and limited bandwidth of automatic test equipment (ATE). New techniques are required to test embedded cores in a SoC without exceeding limits of power, memory, and bandwidth. These techniques are often based on test resource partitioning to tackle ATE memory, test power, and ATE bandwidth limitations. Built-in self-test (BIST) and test-data compression are two widely used techniques to reduce SoC's test-data volume and test application time.

A. BIST

BIST methodology reduces the need for an expensive ATE [1] as on-chip pseudorandom pattern generators and signature compaction are used. In practice, BIST may not always replace other test methods, especially for large chips, due to the long time needed to detect random pattern resistant faults. To overcome these difficulties, deterministic test patterns need to be transferred from ATE to SoC under test to shorten the overall test time and improve fault coverage. The fault coverage in BIST can be improved using techniques such as reseeding [2], bit flipping [3], and bit-fixing [4]. These techniques need structural information for fault simulation and test pattern generation.

B. Test-Data Compression

Test-data compression techniques are used to speed up the ATE-SoC interaction during test. These techniques are used to compress the precomputed test-data set (T_D), often provided by core vendor, to a smaller test set $T_E(|T_E| < |T_D|)$ which is then stored in the ATE's memory. An on-chip decoder decompresses T_E to T_D to be applied to the system under test.

Most compression techniques compress T_D without requiring any structural information about the embedded cores. Compression methods such as statistical coding [5], [6], selective Huffman coding [7], mixed run-length and Huffman coding [8], Golomb coding [9], frequency-directed run-length (FDR) coding [10], alternating run-length coding using FDR [11], extended FDR coding [12], MTC coding [13], and variable-input Huffman coding (VIHC) coding [14] are among techniques that use this strategy.

There are some methods that reduce test volume and time using design modification. The proposed Illinois scan architecture (ILS) [15] needs fault simulation and test generation as postprocessing steps to get high fault coverage. An embedded deterministic test technology for a low-cost test to reduce the scan test-data volume and scan test time is presented in [23].

There are some other techniques which are based on on-chip pattern decompression, such as scan-chain concealment [16], geometric-primitive-based compression [17], mutation encoding [18], deterministic embedded test (reusing the scan chain of one core in a SoC to compress the patterns for another core) [19], packet-based compression [20], and linear feedback shift register (LSFR) reseeding [2], [21], [22].

Several dictionary-based compression methods have been proposed to reduce test-data volume. In [6], frequently occurring blocks are encoded into variable-length indices using Huffman coding. A dictionary with fixed-length indices is used to generate all distinct output vectors in [24]. Test-data compression techniques based on LZ77 and LZW and test-data realignment methods are proposed in [25]–[27], respectively. The method proposed in [28] is a compression technique using dictionary with fixed-length indices for multiple-scan chain designs. A hybrid coding strategy, combining alternating

Manuscript received May 25, 2004; revised November 30, 2004. The work of M. Tehranipoor was supported in part by the University of Maryland Baltimore County's start-up fund. The work of M. Nourani was supported in part by the National Science Foundation CAREER Award CCR-0130513. The work of K. Chakrabarty was supported in part by the National Science Foundation under Grants CCR-9875324 and CCR-0204077.

M. Tehranipoor is with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD 21250 USA (e-mail: tehrani@umbc.edu).

M. Nourani is with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75083 USA (e-mail: nourani@utdallas.edu).

K. Chakrabarty is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: krish@ee.duke.edu).

Digital Object Identifier 10.1109/TVLSI.2005.844311

run-length and dictionary-based encoding, is also proposed [29] to improve the compression. Unfortunately, dictionary-based compression techniques have difficulties in terms of size and organization to increase compression, complexity of algorithms, and required large on-chip memory.

The test generation process generally provides a test cube which has only a few specified bits for each fault. Test compaction schemes merge the compatible cubes and result in compacted deterministic test patterns [32]. The compaction scheme reduces the number of don't cares but often the test patterns still include a large number of don't-cares. The existing don't-cares in a test-data set often help compression techniques achieve higher compression. Generally, an automatic test pattern generator (ATPG) fills the don't-cares randomly to have more chance of detecting nonmodeled faults to increase defect coverage. Some of the proposed techniques, such as [8], [11], [14], and [20], replace existing don't-cares in T_D with zeros or ones to achieve higher compression ratio. The on-chip decoder generates T_D according to don't-care replacement strategy, performed during compression. Such compression may adversely affect the fault coverage of nonmodeled faults. Therefore, techniques such as [30], that leave at least a portion of don't-cares unchanged may be preferred. These leftover don't-cares can be replaced randomly to help detect nonmodeled faults.

C. Main Contribution

In this paper, we present a new compression technique based on a compression code with exactly nine codewords, called 9C. Our method achieves a high compression ratio for a precomputed test set, reduces test application time, and requires a very small decoder. The proposed method can use a decoder independent of test data although a dictionary-based style is also possible. In either case our technique offers ease of synchronization between the decoder and ATE. Our technique is quite flexible in reducing both test time and ATE channel requirement for single- and multiple-scan chain designs.

9C is a fixed-length fixed-code technique with a very small decoder. We will later improve the basic (fixed-length) 9C compression technique to increase the compression ratio by using variable-length blocks. The variable-length block 9C technique, called V9C, achieves compression ratio higher than 9C. In the 9C technique, a fixed-length block size (i.e., K bits) is used for the *entire test-data set* to achieve the best compression while V9C finds the best K for *each test pattern* in the test-data set. This significantly improves the compression ratio with a small increase of decoder cost.

Note that the 9C technique can be implemented as a dictionary-based technique. Our technique, in spite of its simplicity in using a small dictionary, provides high compression and does not need a complicated algorithm to find the best indices and patterns like other dictionary-based techniques [26]–[29]. The decompression logic is quite simple and small because it decodes only nine codewords.

Note carefully that in this work we do not perform any test generation or test set modification. The assumption is that the test patterns, generated and compacted using test generation tools, consist of a very large number of don't-cares. Separation of test generation and compression steps is a common practice as a core vendor does not know what kind of compression technique will be used for compression at the SoC level. Therefore, quite often the conventional test generation is performed without considering the type of compression techniques. Our proposed technique compresses the precomputed test-data set without requiring any structural information of embedded cores. In other words, our technique aims at intellectual property (IP) cores in SoCs. The precomputed test data will be regenerated with no change after decompression.

TABLE I
COMPRESSION RATIO FOR DIFFERENT FIXED-CODING AND BLOCK SIZE $K = 8$ FOR s5378

Circuit	Compression Ratio [%]			
	3-Symbol	5-Symbol	9-Symbol	17-Symbol
s5378	49.22	49.97	51.64	52.11
s13207	71.63	78.08	82.31	83.12

The rest of this paper is organized as follows. Section II reviews the basic fixed-length 9C compression technique. Variable-length block compression (V9C) technique is discussed in Section III. Section IV describes different decoding architectures. Section V analyzes test application time. The experimental results are discussed in Section VI. Finally, the concluding remarks are in Section VII.

II. FIXED-LENGTH BLOCK 9C COMPRESSION

A. Motivation

Our method was inspired by multivalued logic (usually 3, 5, or 9) used in logic simulation and fault testing. An identical or nonidentical block of data is represented as $\mathbf{0}$ (all zeros), $\mathbf{1}$ (all ones) or $\mathbf{U}$ (mixed 0, 1 and don't-care bits). For example, for an eight-bit block, three cases "00 000 000", "11 111 111", and "0xx0xxx1" can be represented by three-symbol coding $\mathbf{0}, \mathbf{1}$, and $\mathbf{U}$. If we divide each block into two halves, five symbols $\mathbf{00}, \mathbf{11}, \mathbf{01}, \mathbf{10}$, and $\mathbf{UU}$ can be employed. An actual example of eight-bit blocks corresponding to these five symbols is "00 000 000", "11 111 111", "00 001 111", "11 110 000", and "x10x01xx". Such division and usage of more symbols can be continued for providing more information and finer block compression.

Table I shows the effectiveness of such fixed-codeword compression using a fixed-length block for s5378 ($K = 12$) and s13207 ($K = 16$) circuits, i.e., two of ISCAS'89 benchmarks. Note that the best compression ratio for these benchmarks reported in the literature, to the best of our knowledge, is 55% [7] and 83% [14], respectively. Table I reports very close compression ratio using a simple fixed-code compression. The size of decompressor for 9- and 17-symbol are 338 and 591 equivalent NAND gates, respectively. Our empirical data shows that beyond nine symbols the overhead (e.g., in terms of decoder cost and/or total test time) added for codewords often offsets the gain. Thus, beyond nine codewords the improvement is negligible and does not justify these overhead increase. In this paper, we fix the number of symbols to be nine and continue our work on other aspects of fixed/variable-length block compression. In Section VI-B, without optimality of 9, we will analytically argue why 9C produces the test results for majority of applications.

TABLE II
9C CODING FOR $K = 8$

Case (i)	Input Block	Symbol (S_i)	Description	Codeword (C_i)	Decoder Input (I_i)	Size $\|I_i\|$ [bits]
1	0000 0000	**00**	All 0's	0	0	1
2	1111 1111	**11**	All 1's	10	10	2
3	0000 1111	**01**	Left half 0, right half 1	11000	11000	5
4	1111 0000	**10**	Left half 1, right half 0	11001	11001	5
5	1111 uuuu	**1U**	Left half 1, right half mismatch	11010	11010uuuu	5+K/2=9
6	uuuu 1111	**U1**	Left half mismatch, right half 1	11011	11011uuuu	5+K/2=9
7	0000 uuuu	**0U**	Left half 0, right half mismatch	11100	11100uuuu	5+K/2=9
8	uuuu 0000	**U0**	Left half mismatch, right half 0	11101	11101uuuu	5+K/2=9
9	uuuu uuuu	**UU**	All mismatch	1111	1111uuuuuuuu	4+K=12

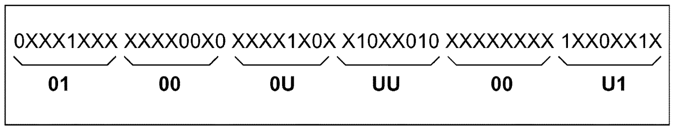


Fig. 1. Small example of formatting the symbols when block size $K = 8$.

B. 9C Compression Technique

The 9C technique is based on fixed-length blocks of input test data. Assume that K is the size of the input blocks. The input test vectors are partitioned into groups of K bits, and encoding is performed on each K-bit block. Table II shows the proposed coding for $K = 8$. As shown in the second column of the table, each K-bit block is divided into two $K/2$ halves. There are overall nine fixed codewords. For cases 1–4, each half K-bit matches all zeros or ones. Cases 5–8 show the mismatched bits, which have to be sent along with the codeword. uuuu denotes a mismatch meaning this half has mix of 0 and 1 and perhaps don't-care x. Case 9 shows that each half is a mismatch and the entire K-bit block has to be sent along with the codeword. The third column shows a symbol (S_i) for each case.

In this technique, regardless of K, we use only nine unique codewords (C_i) shown in the fifth column in Table II. Hence, we called this method nine-coded compression, or 9C for short [30]. The sixth column shows the decoder input (I_i) which can be only a codeword (in the first four cases) or codeword plus the mismatch portion (in the last five cases). Finally, the last column shows the final code size for each case $(|I_i|)$.

As mentioned earlier, in the 9C technique, the input test vectors are divided into K-bit blocks and each K-bit block is divided into two halves. Various cases that can arise match one of the above symbols. For example, blocks of 0000 0000, 0000 xxxx, xxxx 0000, and xxxx xxxx all match symbol $\mathbf{00}$. Blocks of 1111 1111, 1111 xxxx, and xxxx 1111 match symbol $\mathbf{11}$. Blocks of xxxx uuuu, 0000 uuuu and uuuu xxxx, uuuu 0000 match symbols $\mathbf{0U}$ and $\mathbf{U0}$, respectively. We form all test patterns of a circuit as a long test sequence and assume that all test patterns have the same length. Fig. 1 shows a small example of formatting symbols in a test sequence for $K = 8$. These data bits are matched with the nine symbols shown in Table II, and the final set of symbols of this test sequence is $\{\mathbf{01}, \mathbf{00}, \mathbf{0U}, \mathbf{UU}, \mathbf{00}, \mathbf{U1}\}$.

The following two points about our coding scheme are worth mentioning.

1) According to Table II the correspondence between codewords and input blocks (or symbols) is fixed. Clearly, to gain maximum compression, a Huffman coding could be used to assign shorter codewords to the most frequent blocks. However, it makes the coding/decoding scheme dependent on the circuit and precomputed test-data set. Our experimental results show that Huffman coding increases the compression ratio up to only 1.7% for the ISCAS'89 benchmarks. This negligible improvement is achieved at a cost of losing test-data independency which is an important feature in SoC testing. In this paper we pursue data-independent methodology and thus we fixed the codewords-symbols assignment as shown in Table II. Please see some empirical results in Section VI-A.
2) More number of uniform K-bit blocks (e.g., 00 110 011 or 11 001 100) can be added to Table II. However, a systematic coding in such cases requires more (e.g., $2^4+1 = 17$) codewords. This may slightly (less than 1%—see Table I) improve the compression ratio compared to 9 codewords but results in a more complicated and expensive (more than 75% cost increase) decoder and larger overall test time in some cores. Hence, we focused only on having nine codewords [30]. We present an analytical justification for this choice in Section VI-B.

III. Variable-Length Block 9C (V9C) Compression

Comparing fixed-length [6], [7] and variable-length [9], [10], [13], [14] compression techniques based on the results reported in literature indicates that variable-length techniques generally produce more compression with the price of more costly decompression. In the proposed 9C technique only one fixed-length block (K) was used for the entire test-data set. This has its own advantages and disadvantages in terms of decoder size and compression ratio. The main advantage is that the decoder size is very small because of using a fixed-length block for the entire test-data set. It is also flexible to be implemented for single- or multiple-scan chain designs. Moreover, due to its fixed-length nature, when desired, multiple decoders can work in parallel to decrease the test application time further.

Not achieving the highest possible compression ratio is the disadvantage of this technique. Using a fixed-length block may produce good compression for circuits with low don't-care percentage because it finds the best combinations of ones and zeros to match one of the nine symbols. However, this may not be true for a test-data set with high percentage of don't-cares. In other words, using one fixed K may not be the best for all test patterns in a test set.

```
T1:  X0X1XXX0X1XXX1XX0XX01XXXX0XXX0X0
T2:  00XX1001XX11111XXXXX0XXXXX00XX0X
T3:  XXXXXX0XXX0XXXXX0X0XXX0XXXX00XXX

n=3, m=32, T_D= n.m = 96
```

(a)

```
      ┌T1:  X0X1XXX0  X1XXX1XX  0XX01XXX  X0XXX0X0
K=8  <T2:  00XX1001  XX11111X  XXXX0XXX  XX00XX0X
      └T3:  XXXXXX0X  XX0XXXXX  0X0XXX0X  XXX00XXX

T1 Symbols : U0 11 01 00   T1 Codeword Size = 9+2+5+1=17
T2 Symbols : 0U 11 00 00   T2 Codeword Size = 9+2+1+1=13
T3 Symbols : 00 00 00 00   T3 Codeword Size = 1+1+1+1=4

Final Compressed Data Size = 34
```

(b)

Fig. 2. Compression using a fixed K for entire test-data set (9C technique).

```
K1=4   T1:  X0X1 XXX0 X1XX X1XX 0XX0 1XXX X0XX X0X0
K2=8   T2:  00XX1001 XX11111X XXXX0XXX XX00XX0X
K3=32  T3:  XXXXXX0XXX0XXXXX0X0XXX0XXXX00XXX

T1 Symbols : 01 00 11 11 00 11 00 00   T1 Codeword Size = 15
T2 Symbols : 0U 11 00 00               T2 Codeword Size = 13
T3 Symbols : 00                        T3 Codeword Size = 1

Final Compressed Data Size = 29
```

Fig. 3. Compression using different K's for different patterns.

A. V9C Technique

The basic idea behind V9C is to improve the compression ratio by finding the best K for each test pattern in a test-data set. In that case, different K may be obtained for different patterns. Intuitively, for those patterns that contain small number of don't-cares a smaller K may be obtained while a larger K is expected for patterns with higher number of don't-cares.

Fig. 2 shows an example of finding K using the 9C technique. Fig. 2(a) shows $n = 3$ patterns (T_1, T_2, and T_3) each of which contains $m = 32$ bits. These three test patterns form a test sequence containing 96 bits. The 9C technique with block size $K = 8$ provides the best compression for the entire test-data set. Fig. 2(b) shows the associated symbols for each eight-bit block in each test pattern (T_1 to T_3) and the codeword size for each pattern according to Table II. The final compressed data size in case of using $K = 8$ for all three test patterns is $|T_E| = 34$. In the next step, we find the best compression for each test pattern separately. As shown in Fig. 3, $K_1 = 4, K_2 = 8$, and $K_3 = 32$ result in best compression for each test pattern T_1, T_2, and T_3, respectively. The final compressed data size is 29 in this case. This example shows that having a fixed-length block for a test-data set does not guarantee achieving the best results. In other words, if we use different K for different patterns higher compression will most likely be achieved.

Note that a test-data set can be viewed as a long sequence of $|T_D| = n{\cdot}m$ bits, where n is the number of test patterns and m is the length of each test pattern. However, for many applications such as scan we can regroup test data to sizes other than m. In compression, for having more flexibility, we can regroup test data into L-bit vectors (instead of the original m-bit patterns). In general, L can be smaller or larger than m.

```
T'1:  X0X1XXX0X1XXX1XX0XX01XXX
T'2:  X0XXX0X000XX1001XX11111X
T'3:  XXXX0XXXXX00XX0XXXXXXXX0X
T'4:  XX0XXXXX0X0XXX0XXXX00XXX
```

(a)

```
K1=4   T'1:  X0X1  XXX0  X1XX  X1XX  0XX0  1XXX
K2=12  T'2:  X0XXX0X000XX  1001XX11111X
K3=24  T'3:  XXXX0XXXXX00XX0XXXXXXXX0X
K4=24  T'4:  XX0XXXXX0X0XXX0XXXX00XXX

T'1 Symbols : 01 00 11 11 00 11   T'1 Codeword Size = 13
T'2 Symbols : 00 U1               T'2 Codeword Size = 11
T'3 Symbols : 00                  T'3 Codeword Size = 1
T'4 Symbols : 00                  T'4 Codeword Size = 1

Final Compressed Data Size = 27
```

(b)

Fig. 4. Example of creating new patterns using the V9C compression technique.

Fig. 4 clarifies this discussion. Based on the three test patterns shown in Fig. 2(a), the test sequence length is $|T_D| = 96$ bits. We create four new test patterns, T_1', T_2', T_3', and T_4', each with $L = 24$ bits as shown in Fig. 4(a). If we again use the 9C technique for these new test patterns, still a fixed $K = 8$ achieves the best compression for entire test-data set as shown in Fig. 2. But, Fig. 4(b) shows that using different K for these patterns provides higher compression ratio by reducing the final compressed data size. As shown, $K_1 = 4, K_2 = 12, K_3 = 24$, and $K_4 = 24$ result in best compression for each of T_1', T_2', T_3', and T_4', respectively. This eventually provides the best compression for the entire test-data set.

Note that for finding the best compression we need to find two factors: 1) the length of new groups (also called *patterns* hereafter) (L) and 2) block length (K_j) for each L-bit pattern. To reduce the complexity of finding K_j's and decoding structure, for any L's only those K_j's which are divisible by L's (starting from $K = 4$) are considered in our technique. For example, when $L = 32$, only $K_j = 4, 8, 16$, and 32 are tried for each test pattern. Mathematically speaking, K_j is even, $4 \leq K_j \leq L$ and L MOD $K_j = 0$. When a test-data set with $|T_D|$-bit is divided into L-bit patterns, overall ($\lceil |T_D|/L \rceil$ new L-bit patterns are created which is the same as required number of K_j's, therefore, $|T_D| = \sum_{j=1}^{\lceil |T_D|/L \rceil} L_j$.

B. Comparing 9C and V9C

Only one fixed K is required in the 9C technique for the entire test-data set ($|T_D|$ bits). Value K is sent into the decoder only once at the beginning of the decoding process so that every code coming into decoder is decoded based on that K. This process requires an inexpensive decoder.

The disadvantage of 9C is that using only one K for the entire test session may not guarantee a good compression for that test set. In contrast, the V9C technique needs one K_j for each L-bit

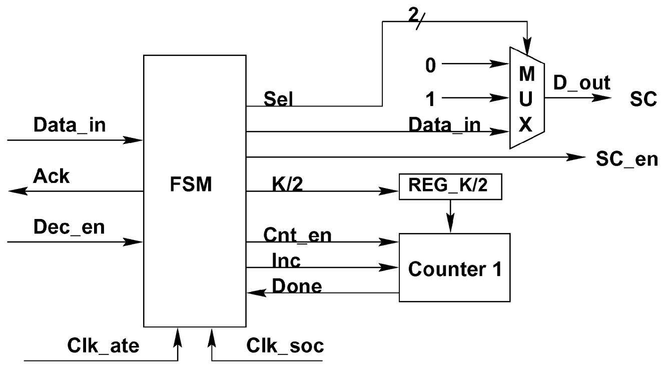


Fig. 5. Single-scan chain decoder for 9C.

pattern. Briefly, K_j is fixed for each pattern but it is variable across the test-data set. So, the decoder needs to know K_j for each L-bit pattern. This leads us to propose the following two different techniques to construct K_j's in V9C.

- *Data-Independent V9C*: In this technique, since K_j is fixed for each L-bit pattern, K_j is sent to the on-chip decoder just before sending the codewords of that L-bit pattern. Therefore, one K_j is sent to the on-chip decoder for each L-bit pattern and overall $\lceil |T_D|/L \rceil$ number of K_j's are sent for a test-data set. Of course, to reduce the overhead of sending K_j's, the related code of each K_j's will be sent. Assume that G denotes the total number of distinct divisors of L starting from 4. For example, if $L = 32$, divisors are $K_j = 4, 8, 16$, and 32. Thus, $G = 4$ and only two bits are required to detect each K_j's. Therefore, the total number of bits sent into decoder, $|T_E|$, is equal to final compressed data size plus $(\lceil |T_D|/L \rceil) \cdot \lceil \log_2 G \rceil$. The advantages of this technique is that: 1) it is test-data independent and thus can be reused even if the test-data set changes and 2) it achieves higher compression ratio compared to the 9C technique.
- *Dictionary-Based V9C*: In this technique, we assume that K_j's can be stored in an on-chip dictionary (memory). In that case, there is no need to send K_j's with test patterns and $|T_E|$ is equal to the final compressed data size. This significantly reduces the final compressed data size and increases the compression ratio. It, however, increases the hardware overhead and the decoder is dependent on the test-data set. The decoding architecture for each of these techniques will be discussed in the next section.

IV. Decoding Architecture

9C and V9C techniques are flexible to be used for compressing input test data for single- or multiple-scan chain designs. Here, we propose a small and flexible decompression architecture for each case.

A. Single-Scan Chain Decoder

1) 9C Technique: Fig. 5 shows the decoder architecture for a single-scan chain. Decoder decodes only nine prefix-free codewords independent of K value and precomputed test-data set. The decoder consists of a finite-state machine (FSM), Counter 1, and REG_$K/2$ register. The first few bits transferred into FSM through *Data_in* indicate $K/2$ and are sent into REG_$K/2$ register. Thus, decoder uses this K for the entire test-data set. We

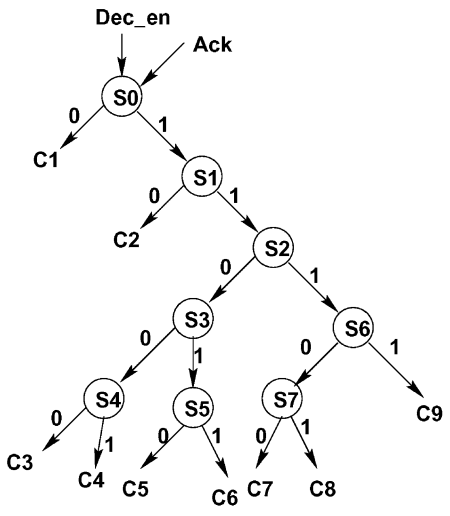


Fig. 6. FSM diagram of 9C decoder.

send $K/2$ because the 9C technique detects each half of a K-bit block separately for each codeword. Next, FSM takes data from Data_in to find out which codeword has been entered. If the input codeword is C_1, C_2, C_3, or C_4, all K bits are generated only with zeros or ones accordingly. When the FSM receives the codewords C_5, C_6, C_7, C_8, or C_9, it expects to receive $K/2$-bit or K-bit exact data from *Data_in*.

As shown in Fig. 5, MUX has three inputs 0, 1, and *Data_in* which is shifted through FSM. Two bits select (*Sel*) come from FSM to MUX. Counter 1 is used to control sending $K/2$-bit into scan chain. The FSM sends signals Cnt_en and *Inc* to activate and increment the counter, respectively. At the same time, it activates signal SC_en to enable the scan chain, hence D_out is shifted into the scan chain (SC). When the counter finishes counting, it sends signal *Done* to the FSM to send the next *Sel* and Cnt_en signals. After sending the second signal *Done* by the counter, FSM sends *Ack* signal to the ATE to send the next codeword and deactivates signal SC_en.

Fig. 6 shows the state diagram of FSM used in 9C decoder. Note carefully that it is totally independent of K. It starts by checking Dec_en. When Dec_en is active, it receives *Data_in* which is the codeword from ATE. Maximum of five cycles are required for the longest codeword. Since a K-bit block is divided into two halves, the FSM sends one *Sel* for each half. After detecting a codeword the required signals are sent to the counter, MUX, and scan chain. When the job is done for one received codeword, the state controller starts again from state 0. This is done when *Ack* signal becomes active, then ATE sends the next codeword.

2) Data-Independent V9C: In data-independent V9C, after extracting the best K_j's they can be sent into the on-chip decoder along with the codewords. Fig. 7 shows the decoder architecture for data-independent V9C. The decoding architecture here is slightly more expensive (shaded components) than the decoder proposed for the 9C technique. ATE sends L's into FSM to be sent into REG_L that will be used for the entire test-data set. To transfer codewords, corresponding to L-bit test pattern, to the decoder, first the related codeword's K_j is sent. FSM detects $K_j/2$ and sends it into (REG_$K_j/2$) register. Counter 1

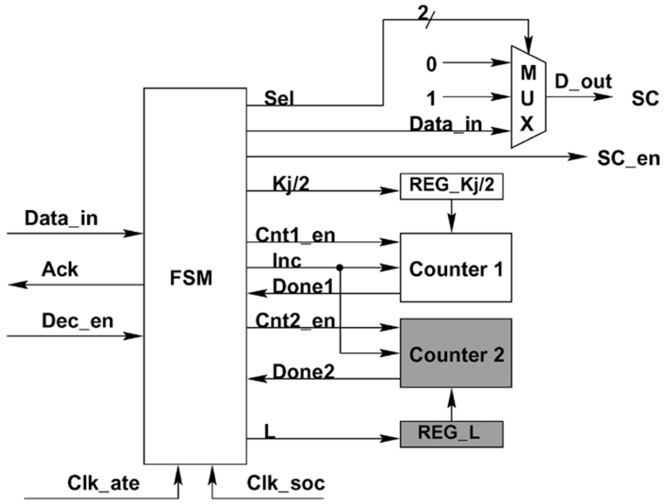


Fig. 7. Single-scan chain decoder for data-independent V9C.

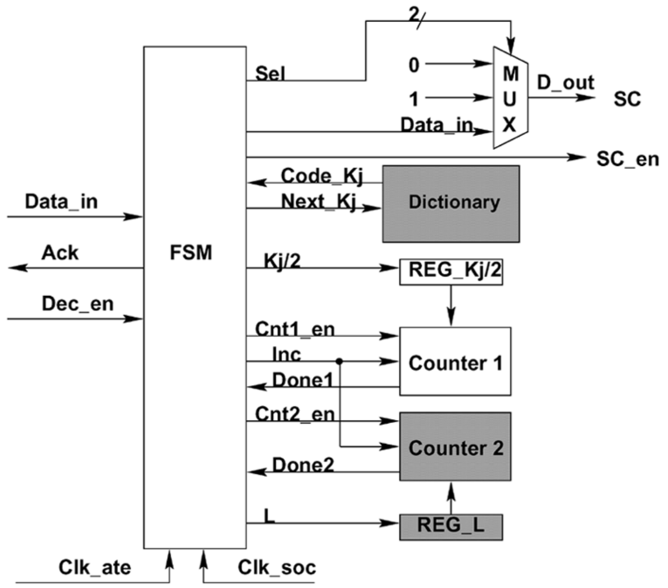


Fig. 8. Single-scan chain decoder for dictionary-based V9C.

is used to control sending $K/2$-bit into scan chain. Counter 2 is used to send L bits into the scan chain for each K_j. When Counter 2 finishes counting, L bits have been shifted into the scan chain, it sends signal *Done2* to FSM. In this case, FSM reads the next $\lceil \log_2 G \rceil$ bits of encoded K_j's for the next L-bit pattern through *Data_in*.

3) Dictionary-Based V9C: In this technique, K_j's are stored in an on-chip dictionary (memory). Fig. 8 shows the decoder architecture for case of having a dictionary or a memory to store the required K_j's for each L-bit pattern. To reduce the size of dictionary, the encoded K_j's ($\lceil \log_2 G \rceil$ bits each) is stored in the memory. In this case, any time that decoder starts sending L bits into the scan chain, FSM sends the Next_K_j signal to receive the codeword of the next K_j, and FSM decodes $K_j/2$ and sends it into REG_$K_j/2$ register. Then, $K_j/2$ value is sent to the inputs of Counter 1. When Counter 1 finishes counting, it sends signal *Done1* to FSM to generate signal Next_K_j and sends it to the dictionary. The size of memory to store all K_j's is calculated as $(\lceil |T_D|/L \rceil) \lceil \log_2 G \rceil$ where G is the distinct number of even divisors of L's starting from 4.

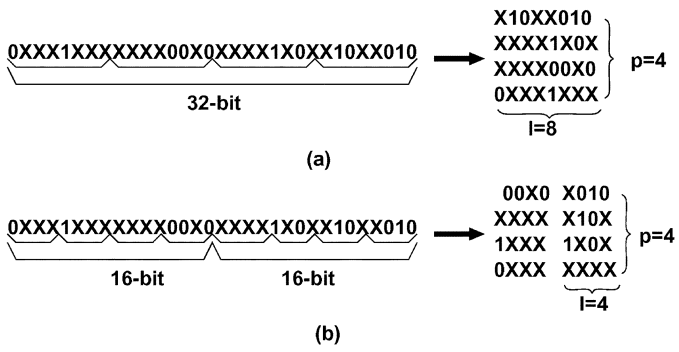


Fig. 9. Example of formatting the test data for multiple-scan chain (a) $p = 4$ and $l = 8$ and (b) $p = 4$, and $l = 4$.

The decoding process is almost the same as data-independent decoding architecture. FSM used in this architecture is slightly different from the one used in Fig. 7 because it has to receive codewords corresponding to K_j's from the dictionary. This decoder is dependent on the precomputed test-data set but achieves very high compression compared to 9C.

B. Multiple-Scan Chain Decoder

In this section we only discuss about 9C decoder architecture to handle multiple-scan chain as it would be the same for the V9C technique.

Assume that the scan chain with length m of a circuit under test is divided into p equal length scan subchains. Each test vector can therefore be viewed as p subvectors. If one or more subvectors are shorter than others, don't-cares are added to the end of these subvectors so that all subvectors have the same length, which is denoted as l. The same p-bit data at the same position of each subvector constitute a p-bit word. Then, a total of $n \cdot lp$-bit words are formed and encoded during the compression procedure where n is the number of test vectors. Fig. 9 illustrates the formatting of the given test sequence for multiple-scan chains for two different l's. In Fig. 9(a), $p = 4$ and $l = 8$, corresponding to a test vector with length $m \cdot l =$ 32-bit. Fig. 9(b) shows a test vector with $p = 4$ and $l = 4$ corresponding to two 16-bit consecutive test vectors. During test application, after a codeword is shifted into the decoder, a p-bit word is generated by the decoder and fed into the scan chains (one bit for each scan chain).

Fig. 10 shows the decoder architecture for a multiple-scan chain. Using the 9C technique, the p-bit word is divided into groups of K-bit blocks. The decoder consists of an additional counter, Counter 2, compared to 9C decoder for single-scan chain (Fig. 5), FSM is the same and so is the decoding process. The FSM sends Cnt_en and *Inc* signals to enable and increment Counters 1 and 2 and Shift_en to let D_out to be shifted into p-bit shifter. Counter 2 is used to control shifting p bits into the p-bit shifter. Any time that Counter 1 sends $K/2$ bits into the p-bit shifter, signal *Done* is sent to the FSM to send the next *Sel* and Cnt_en signals. When Counter 2 reaches p it sends the signal *Load* to the p-bit shifter to load its content into the scan chains SC_1 to SC_p. This architecture reduces the input test pins to only one, i.e., *Data_in*. In this case, only one decoder is used for p scan chains. Shaded units show the additional components compared to decoder used for 9C in single-scan chain.

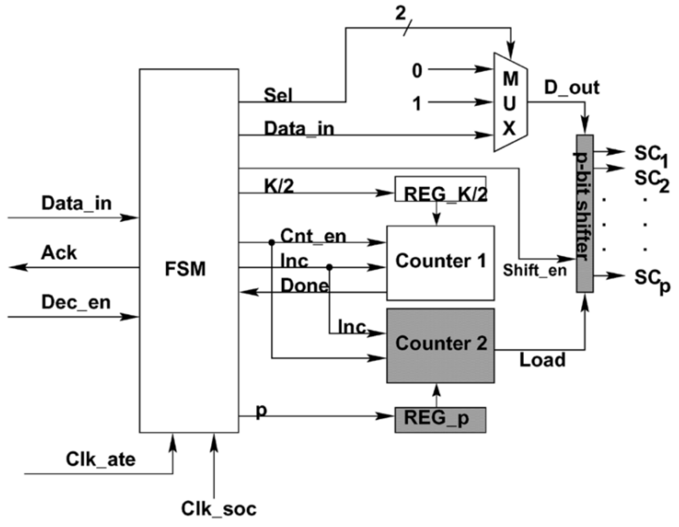


Fig. 10. Multiple-scan chain decoder for 9C.

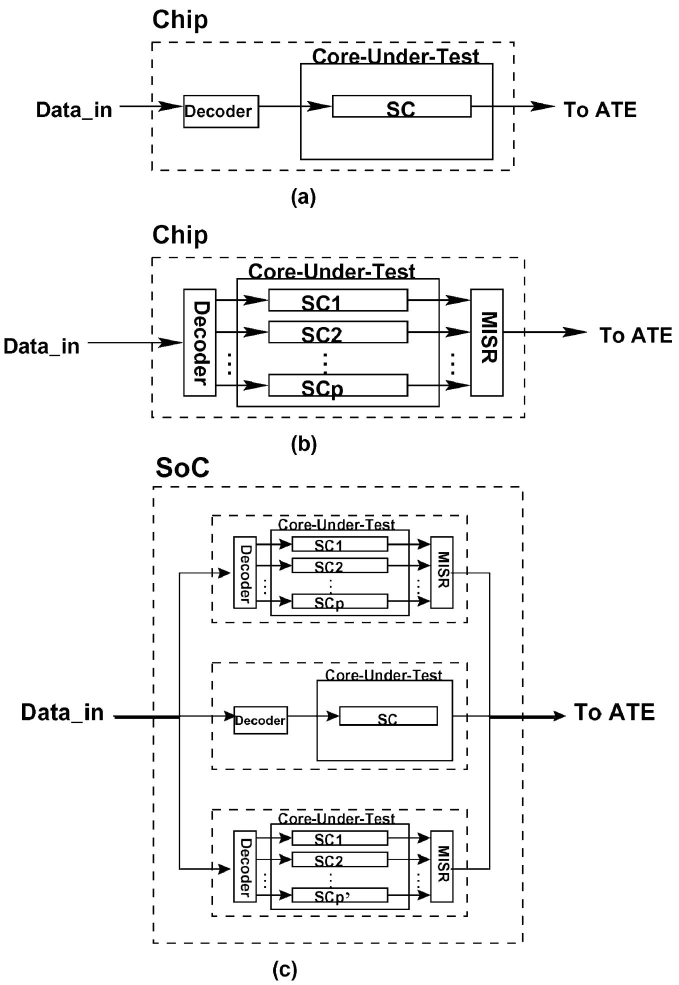


Fig. 11. (a) Single-scan chain decoder, (b) multiple-scan chain decoder, and (c) a SoC including three different cores.

C. Distributed Decoder

Assume that p and l are the number of scan chains and the length of scan chain, respectively. Fig. 11 shows the flexibility of our method in using ATE pins and distributed style of decompresser units. Fig. 11(a) shows a single decoder, used to decode and send the test patterns into a single-scan chain core. Fig. 11(b) shows a multiple-scan chain decoder with one input-data pin *Data_in* for a multiple-scan chain core assuming the length of scan chains is the same. The ATE sends codewords through *Data_in* and the decoder drives the detected bits into the p-bit shifter (as shown in Fig. 10). In this case each detected K-bit is shifted into the p-bit shifter and after completing p-bit, the content of p-bit shifter is sent into p scan chains. Using a multiple input shift register (MISR) is optional to optimize signature analysis. This architecture reduces the number of required decoders and test pins. In practice, most cores include multiple-scan chains. Hence, the decoders should be reusable for different cores in a SoC and flexible to be utilized for both single- and multiple-scan chains. Fig. 11(c) shows a SoC containing three different cores with single- or multiple-scan chains assuming the length of scan chains in a core is the same but varies in different cores. In today's test synthesis tools like the design-for-test (DFT) compiler from SYNOPSYS, we can define the length of scan chains and it actually forms all subchains with the same length. This figure shows the basic architecture and one decoder is used per core in SoC. Further optimization (e.g., sharing decoders or MISR, test control/access mechanism, etc.) can be performed depending on the application. All the decoders are initialized at the beginning of test application process.

V. Test Application Time

Reducing the overall test application time is one of the important goals of any test-data compression method. In general, the amount of time reduction depends on the compression ratio and decompression method. We analyze the test application time reduction for both the 9C and V9C techniques. Since 9C is a fixed-block compression technique, the test time analysis is simpler compared to the V9C technique. Suppose ATE and SoC scan frequencies are f_{ate} and $f_{\text{scan}} (f_{\text{ate}} \le f_{\text{scan}})$, respectively. Test application time reduction (TR) is given by

$$\text{TR}\% = \frac{t_{\text{no\_comp}} - t_{\text{comp}}}{t_{\text{no\_comp}}} \times 100 \qquad (1)$$

where $t_{\text{no\_comp}}$ and t_{comp} are the test application times for applying uncompressed and compressed test data, respectively. Assume that the scan clock frequency of the system under test is q times that of the ATE's clock frequency, i.e., $(f_{\text{ate}})/(f_{\text{scan}}) = (1/q)$. The test application time for the case of no compression $(t_{\text{no\_comp}})$ depends on the total number of input bits $(|T_D|)$ and ATE's working frequency

$$t_{\text{no\_comp}} = \frac{|T_D|}{f_{\text{ate}}} = \frac{q \cdot |T_D|}{f_{\text{scan}}}.$$

A. 9C Technique

After compressing the test data, the test application time t_{comp} will depend on the occurrence frequency (repetitions) of each symbol (N_i). Therefore, in the 9C technique, t_{comp} is given by

$$t_{\text{comp}} = \sum_{i=1}^{9} t_i$$

TABLE III
COMPRESSION RATIO FOR DIFFERENT K'S FOR 9C

Circuit	$\lvert T_D\rvert$	*CR%*							
		K=4	K=8	K=12	K=16	K=20	K=24	K=28	K=32
s5378	23754	43.98	50.69	**51.64**	49.41	46.61	44.77	42.18	39.37
s9234	39273	48.87	**50.91**	46.53	41.92	37.38	33.27	28.97	26.44
s13207	165200	69.63	79.81	81.86	**82.31**	80.65	80.93	79.80	78.96
s15850	76986	60.14	**66.38**	65.11	62.95	60.73	58.51	56.55	55.13
s38417	164736	52.63	**60.63**	59.37	57.54	54.40	51.78	49.31	47.44
s38584	199104	58.82	**65.53**	64.43	62.39	59.79	56.98	54.66	52.13

where t_i is the application time of symbol S_i. When decoder input I_i is entered into FSM, $|I_i|$ ATEs and K system clocks are needed for applying K bits into the scan chain. Therefore, t_i is computed by

$$t_i = \left(\frac{K}{f_{\text{scan}}} + \frac{|I_i|}{f_{\text{ate}}}\right) N_i$$

where $f_{\text{scan}} = q \cdot f_{\text{ate}}$. Thus, t_i is

$$\begin{aligned} t_i &= \left(\frac{K + q|I_i|}{f_{\text{scan}}}\right) N_i \\ t_{\text{comp}} &= \sum_{i=1}^{9} \left(\frac{K + q|I_i|}{f_{\text{scan}}}\right) N_i \\ &= \frac{K}{f_{\text{scan}}} \sum_{i=1}^{9} N_i + \frac{q}{f_{\text{scan}}} \sum_{i=1}^{9} |I_i| N_i \\ &= \frac{KN + q|T_E|}{f_{\text{scan}}} \end{aligned}$$

where $N = \sum_{i=1}^{9} N_i$ and $|T_E| = \sum_{i=1}^{9} |I_i| N_i$. Finally, TR is computed by

$$\text{TR} = 1 - \frac{KN + q|T_E|}{q|T_D|} = \text{CR} - \frac{KN}{q|T_D|}.$$

B. V9C Technique

In the V9C technique, the test sequence is divided into L-bit patterns and K is obtained for each L-bit pattern. We define S_{ji} to be the ith symbol of the jth L-bit pattern in the test sequence. In this case, $\sum_{i=1}^{9} t_i$ is defined to be the test application time of sending L bits into the scan chain. Here, we analyze the test application time for both data-independent and dictionary-based V9C techniques.

1) Data-Independent V9C: In this case, K_j's are sent along with the codewords. So, t_{comp} is computed by

$$t_{\text{comp}} = \sum_{j=1}^{\lceil |T_D|/L \rceil} \sum_{i=1}^{9} t_{ji} + \left(\frac{\lceil |T_D|/L \rceil}{f_{\text{scan}}} \lceil \log_2 \max_j K_j \rceil\right)$$

where t_{ji} is the application time of symbol S_{ji} and is obtained by

$$t_{ji} = \left(\frac{K_j}{f_{\text{scan}}} + \frac{|I_{ji}|}{f_{\text{ate}}}\right) N_{ji}$$

where N_{ji} is the occurrence frequency of the ith symbol in the jth L-bit pattern. Having $t_{\text{no\_comp}}$ and t_{comp}, TR is computed using (1).

2) Dictionary-Based V9C: In this case, K_j's are stored in an on-chip memory. Therefore, t_{comp} is computed by

$$t_{\text{comp}} = \sum_{j=1}^{\lceil |T_D|/L \rceil} \sum_{i=1}^{9} t_{ji} \quad t_{ji} = \left(\frac{K_j}{f_{\text{scan}}} + \frac{|I_{ji}|}{f_{\text{ate}}}\right) N_{ji}.$$

VI. EXPERIMENTAL RESULTS AND ANALYSIS

A. Results

Test generation tools generate test cubes for each fault in a circuit and compaction process generates the final deterministic test patterns that include a large number of don't-cares. Specifically, in today's large circuits, we expect to have only 1%–5% specified bits in test-data set [31]. In our experiments test-data set is assumed to be the only data provided and no structural information of the core is required. In our technique we do not combine ATPG and compression like some other techniques [15], [23].

The compression ratio (CR) for the 9C and V9C techniques is computed by

$$\text{CR}\% = \frac{|T_D| - |T_E|}{|T_D|} \times 100.$$

$|T_E|$ for different cases is computed by the following:

1) *9C*:

$$|T_E| = \sum_{i=1}^{9} |I_i| N_i.$$

2) *Data-Independent V9C*:

$$|T_E| = \sum_{j=1}^{\lceil |T_D|/L \rceil} \sum_{i=1}^{9} |I_{ji}| N_{ji} + \lceil |T_D|/L \rceil \lceil \log_2(\max K_j) \rceil.$$

3) *Dictionary-Based V9C*:

$$|T_E| = \sum_{j=1}^{\lceil |T_D|/L \rceil} \sum_{i=1}^{9} |I_{ji}| N_{ji}.$$

Table III shows the compression results of ISCAS'89 benchmarks for different K using the 9C technique for a single-scan chain. As seen in Table III, the maximum compression ratio for these benchmarks (i.e., boldface numbers) happens in $K = 8, 12$, or 16. When K increases the compression ratio increases to reach a peak at $K = 8, 12$, or 16 and then it starts to decrease in most cases. As shown, $K = 32$ generates less compression ratio compared to other K.

TABLE IV
COMPRESSION RATIO FOR DIFFERENT L's USING DATA-INDEPENDENT V9C

Circuit	$\|T_D\|$	*CR%*								
		L=20	L=32	L=40	L=48	L=60	L=80	L=100	L=200	L=400
s5378	23754	50.00	54.11	**55.75**	54.27	55.06	54.13	52.83	53.81	53.50
s9234	39273	49.94	52.15	49.53	**54.77**	53.74	53.99	53.29	53.80	53.65
s13207	165200	77.21	81.66	81.09	82.72	84.78	84.21	84.68	**85.20**	84.86
s15850	76986	65.00	68.70	68.05	69.69	69.99	71.12	70.34	**71.26**	69.69
s38417	164736	57.94	60.47	59.96	61.53	61.75	**62.89**	60.51	61.58	61.88
s38584	199104	62.91	66.29	68.10	67.51	66.17	68.41	68.12	**69.11**	68.72

TABLE V
COMPRESSION RATIO FOR DIFFERENT L's USING DICTIONARY-BASED V9C

Circuit	$\|T_D\|$	*CR%*								
		L=20	L=32	L=40	L=48	L=60	L=80	L=100	L=200	L=400
s5378	23754	59.99	60.36	**63.24**	60.51	60.05	57.87	55.83	55.30	54.49
s9234	39273	59.93	58.40	57.03	**61.02**	58.74	57.74	56.29	55.30	54.65
s13207	165200	87.21	87.91	88.59	88.97	88.53	**89.22**	87.68	86.69	85.86
s15850	76986	74.74	74.95	75.54	**75.94**	74.99	74.86	73.34	72.75	70.68
s38417	164736	66.72	66.72	68.46	**71.38**	69.75	65.47	63.51	63.08	62.88
s38584	199104	72.91	72.54	73.10	73.67	**73.77**	72.17	71.13	70.61	69.72

TABLE VI
COMPRESSION RESULTS FOR TWO LARGE CIRCUITS FROM IBM IN THE CASE OF SINGLE-SCAN CHAIN USING DICTIONARY-BASED V9C

Circuit	*X%*	$\|T_D\|$	*CR%*				
			L=80	L=100	L=200	L=400	L=1000
CKT1	99.36	11613472	98.00	**98.12**	97.98	97.82	97.45
CKT2	97.90	4124288	96.38	96.31	96.06	**96.65**	96.31

Table IV shows the compression ratios of ISCAS'89 benchmarks for different L's using the data-independent V9C technique. In this case, K are sent to the on-chip decoder along with the codewords. The CRs of different L's are very close because V9C finds the best compression by tuning K for each test pattern. For very small and very large L's such tuning is harder and CR values are overall lower. The maximum CR for each benchmark is written in boldface font.

Table V shows the compression results for different L's using the dictionary-based V9C technique. In this case K are stored in an on-chip memory or dictionary and only codewords are sent to the on-chip decoder. As shown, this technique shows much higher compression ratios.

We also evaluated the compression efficiency of the dictionary-based V9C for very large test sets from IBM previously reported in [28]. Circuit CKT1 contains 3.6 million gates, 726 000 flip flops and requires about 11.6 Mbit test data. Circuit CKT2 has 1.2 million gates, 32 200 flip flops and needs 4.1 Mbit test data. Table VI shows the compression ratio for different L's. As shown, $L = 100$ and $L = 400$ show the maximum compression for CKT1 and CKT2, respectively.

In the proposed 9C coding the codewords size are fixed. As we explained in Section II, a Huffman coding also could be used to increase compression ratio. Using fixed codewords not only simplifies decoding process but also decoder will be independent to precomputed test-data set which is not the case for Huffman coding. Table VII compares these two cases. As seen Huffman coding only increases the compression ratio up to 1.7% which is negligible. The cost of decoder is almost the same for these cases.

TABLE VII
COMPARISON BETWEEN USING FIXED CODE AND HUFFMAN CODE IN 9C TECHNIQUE

Circuit	9C (fixed Codewords)		9C (Huffman Codewords)	
	CR%	*Cost*	*CR%*	*Cost*
s5378	51.64	338	52.68	324
s9234	50.91	338	51.12	360
s13207	82.31	338	84.04	330
s15850	66.38	338	66.88	346
s38417	60.63	338	61.45	346
s38584	65.53	338	65.99	330

Comparing data-independent V9C to the other data-independent techniques such as 9C [30], FDR [11], VIHC [14], and MTC [13] is shown in Table VIII. As seen, data-independent V9C shows better compression than 9C and other techniques.

Table IX shows comparison between dictionary-based V9C and some other dictionary-based techniques (fixed-length indices [28], LZ77 [26], and LZW [27] and selective Huffman [7]). Overall, our technique shows close results to the others. Note that dictionary-based V9C may not achieve the highest possible CR but it is flexible and less costly compared to others. The second column of the table shows the size of required dictionary. The size of dictionary in V9C is comparable to [28] and [7] and relatively small compared to other techniques especially LZ77 [26] and LZW [27]. Direct comparison of dictionary cost is not possible due to differences among implementation style, tools, and statistics reported.

Table X shows the test application time reduction (TR) for ISCAS'89 benchmarks. Assume that $f_{\rm scan}$ and $f_{\rm ate}$ are the frequency of shifting test patterns into the scan chain and ATE clock frequency, respectively. Our analysis shows that TR is bounded by CR, and as $f_{\rm scan}/f_{\rm ate}$ increases, the test application time approaches the compression ratio. For example, assume that $f_{\rm ate} = 20$ MHz (a very slow ATE) and the SoC frequency to shift test data into scan chain $f_{\rm scan} = 100$ MHz. Reduction of up to 67% for data-independent V9C and up to 71% for dictionary-based V9C, indicates good reduction in test application time for such a low speed ATE. It is clear that dictionary-based

TABLE VIII
COMPARING 9C AND DATA-INDEPENDENT V9C WITH OTHER TEST DATA-INDEPENDENT TECHNIQUES

Circuit	*CR%*				
	Data-Independent V9C	*9C*	*FDR* [11]	*VIHC* [14]	*MTC* [13]
s5378	55.75	51.64	50.77	51.52	38.49
s9234	54.77	50.91	44.96	54.84	39.06
s13207	85.20	82.31	80.23	83.21	77.00
s15850	71.26	66.38	65.83	60.68	59.32
s38417	62.89	60.63	60.55	54.51	55.42
s38584	69.11	65.53	61.13	56.97	56.63
Avg.	66.49	62.90	60.58	60.28	54.32

TABLE IX
COMPARING DICTIONARY-BASED V9C WITH OTHER DICTIONARY-BASED TECHNIQUES

Circuit	V9C Dictionary [Bits]	*CR%*				
		Dictionary-Based V9C	Fixed-Length Indices [28]	*LZW* [27]	*LZ77* [26]	*Selective Huffman [7]*
s5378	1187	63.24	73.28	N/A	61.00	55.10
s9234	2154	61.02	70.72	70.26	55.00	54.20
s13207	5595	89.22	94.84	81.69	81.45	77.00
s15850	3527	75.94	81.97	76.26	79.00	66.00
s38417	8804	71.39	61.79	70.60	61.56	59.00
s38584	9152	73.77	73.23	75.14	59.97	64.10
Avg.	–	72.43	75.97	74.79	66.33	62.56

TABLE X
TEST APPLICATION TIME REDUCTION (TR%) USING V9C TECHNIQUE

Circuit	9C		Data-Independent V9C		Dictionary-Based V9C	
	CR%	*TR%*	*CR%*	*TR%*	*CR%*	*TR%*
s5378	51.64	34.96	55.75	38.16	63.24	43.62
s9234	50.91	34.91	54.77	37.23	61.02	43.08
s13207	82.31	65.21	85.20	66.91	89.22	71.18
s15850	66.38	43.78	71.26	48.12	75.94	51.36
s38417	60.63	40.21	62.89	43.85	71.39	48.90
s38584	65.53	45.23	69.11	48.68	73.77	55.72

V9C must give better TR% because it reads K from an internal dictionary instead of a low speed external ATE.

Table XI shows the cost of the on-chip decompression logic for different techniques. The results for our techniques (9C and V9C) are obtained using the synopsys design compiler [32]. The size of the 9C technique is comparable to other techniques as the size of 9C is fixed while the cost of other techniques like FDR and VIHC increases as the longest run (in FDR) or group size (in VIHC) increase.

The table also compares decoder cost of our dictionary-based V9C with other proposed dictionary-based techniques. The comparison with fixed-length indices technique [28] is not possible as the authors in [28] reported the size of decoder based on required number of gates not bits to implement the dictionary. The size of V9C dictionary for different benchmarks is also listed in the second column of Table IX.

Another important feature of the V9C technique is its flexibility to provide decompressor *reuse* in SoCs. To achieve the maximum CR using dictionary-based V9C, cores in a SoC should have a dedicated decompressor. This could be too costly. Tradeoffs between CR% and cost can be explored by designer to satisfy time/cost requirement. Table XII summarizes this concept for a small example by showing cost and CR. We assumed our SoC has three cores, i.e., s5378, s13207, and s38584. Using a dedicated decompressor provides the highest CR% as shown using boldface numbers. The average compression ratio will be $\mathrm{CR}_{\mathrm{avg}} = (\mathrm{CR}_a + \mathrm{CR}_b + \mathrm{CR}_c)/(3) = 75.41\%$ with cost of $3 * 401 = 1203$ NAND gates plus $1187 + 5595 + 9152 = 15\,934$ bits for dictionary memory. If we use only one of these decompressors to cover all three cores, this average drops to 51.81%, 53.84%, and 54.63%. This example shows that using our technique cost saving through the decompressor reuse is possible. In the last column of table, we designed decompressor using data-independent V9C that is able to work with all cores. In this case, the average CR for these three cores is 70.02% in a cost of $3 * 416 = 1248$ NAND gates.

The style, cost, and flexibility of on-chip decompresser have become important factors in practicality of compression techniques. Specifically, some decoders such as those proposed in [5]–[7], [14], [24] are dependent on the precomputed test set and thus are customized for the circuit under test. The decompression logics in [10], [11], and [30] are independent of the precomputed test set. The 9C and data-independent V9C decoders are totally independent of the circuit under test and precomputed test set. In other words, it guarantees the best compression for each circuit no matter what the precomputed test-data set is. This feature makes our V9C technique superior in terms of cost, flexibility, and design reuse. However, dictionary-based V9C is test-data-dependent. It achieves a higher compression ratio with a price of higher decoder cost.

B. Optimum Number of Codewords

As shown throughout the paper so far, the use of nine codewords (9C encoding) is our preferred choice. While we acknowledge that 9 may not be a global optimum in an analytical sense, it empirically provides the best compromise among three important factors of compression ratio (CR), time reduction (TR), and decoder cost (DC). In this subsection we briefly justify this preferred choice. Fig. 12 demonstrates the general trend for CR, TR, and DC metrics that we observed for various codeword sizes (3C, 5C, 9C, 17C, and 33C) when applied to benchmarks. Note that the number of codewords (x)

TABLE XI
COST OF DIFFERENT DECODERS

Cost [# NAND]						
9C	Data-Independent V9C	FDR [11]	VIHC [14]	Dictionary-Based V9C	LZW [27]	Fixed-Length Indices [28]
338	416	320	296	401+dictionary bit	500+1024×64 bit	n/a

TABLE XII
REUSING V9C DECOMPRESSORS FOR DIFFERENT CORES IN A SoC

Circuit	Dictionary-Based V9C			Data-Independent V9C
	s5378 [401 NAND + 1187 bits]	s13207 [401 NAND + 5595 bits]	s38584 [401 NAND + 9152]	[416 NAND]
s5378	**63.24**	35.44	34.90	55.75
s13207	59.61	**89.22**	55.23	85.20
s38584	32.56	36.87	**73.77**	69.11
Avg.	51.81	53.84	54.63	70.02

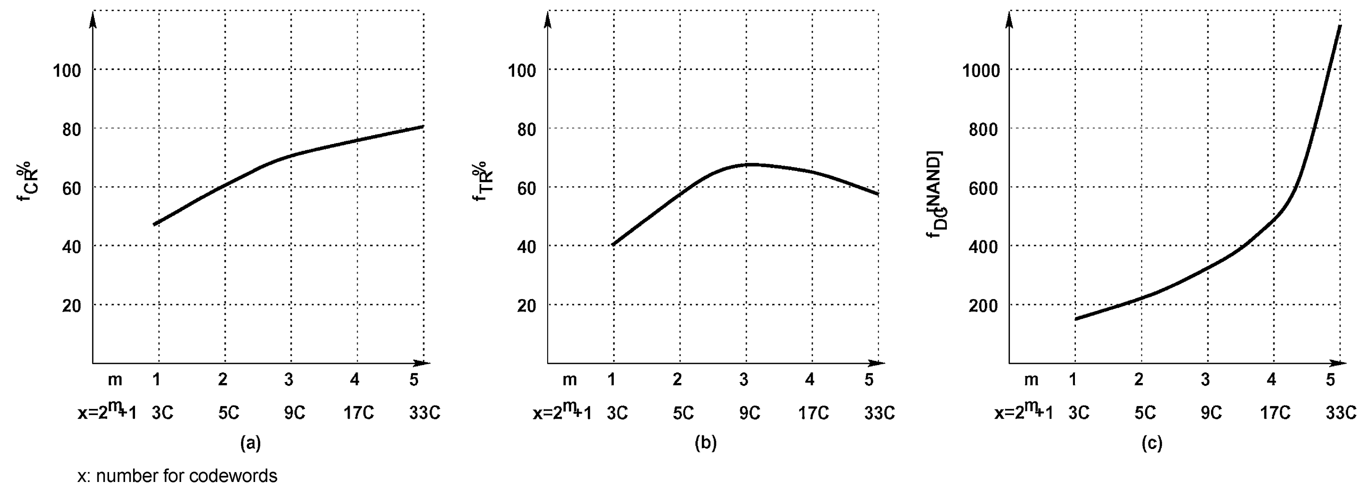


Fig. 12. (a) CR, (b) TR, and (c) DC using 3C, 5C, 9C, 17C, and 33C techniques.

is an integer that satisfies the basic relation $x = 2^m + 1$ for some integer m. Thus, functions $f_{\rm CR}(x)$, $f_{\rm TR}(x)$, and $f_{\rm DC}(x)$) shown in Fig. 12 are only approximations reflecting the basic trend. Yet, these approximations stay valid when we analyze the fidelity of a weighted mix of these functions. Our empirical evidences (see also Fig. 12) indicate that for $3 \le x \le 33$ these are good approximation functions

$$\begin{cases} f_{\rm CR}(x) = Ax + B \\ f_{\rm TR}(x) = Cx^2 + Dx + E \\ f_{\rm DC}(x) = Fx^2 + G \end{cases}$$

where A, B, C, D, E, F, and G are constants that depend on the circuit under test and its precomputed test set. Intuitively, increasing the number of codewords almost linearly increases CR. TR increases when we use larger number of codewords until it reaches the peak at 9C, and it then slightly decreases. The main reason is that the codewords become longer and FSM uses more number of ATE clocks to detect and decode the received codewords. Finally, the decoder cost increases quadratically as the number of codewords increases due to its need for a larger FSM and internal memory. For example, in case of s13207 benchmark we have found $\{A, B, C, D, E, F, G\} = \{3.3, 61.0, 1.8, -5.0, -6.0, 2.2, 70.0\}$ to specify the above functions. Details of numerical methods used to find these coefficients are beyond the scope of this paper and reader is referred to [33]. So far we have individually approximated the main three evaluation factors for our compression methodology. To be more flexible in our analysis, we have combined them in a weighted function using $W_{\rm CR}, W_{\rm TR}$, and $W_{\rm DC}$ coefficients $(W_{\rm CR} + W_{\rm TR} + W_{\rm DC} = 1)$ predefined by a user showing their relative importance. Therefore, the main evaluation (quality) function $f(x)$ is defined as

$$f(x) = W_{\rm CR} \cdot f_{\rm CR}(x) + W_{\rm TR} \cdot f_{\rm TR}(x) + W_{\rm DC} \cdot f_{\rm DC}(x).$$

In our experimental results shown in Section VI-A we assumed that $W_{\rm CR} = W_{\rm TR} \ge W_{\rm DC}$ as the aim of our compression technique is to improve CR and TR as primary concern. Cost is a secondary issue for us as in practice it is negligible compared to overall design cost. We have approximated these functions for all of benchmarks and computed $df(x)/dx = 0$ to find the optimum point (x^*). The results are tabulated in Table XIII for various choices of weights. This table also shows clearly that optimality depends on relative importance (weight coefficients) of these three metrics. When DC has the largest weight, the optimal choice will be using the smallest possible codeword size (i.e., 3; see the last column). For a very conservative design, when three factors are almost equally weighted (fourth column) choice of 5 will be an optimal choice. On the other hand, assigning larger weights on the CR and TR results in optimal choice of 9 in almost all cases (see second and third columns).

TABLE XIII
ANALYSIS OF THE REQUIRED NUMBER OF CODEWORDS FOR DIFFERENT BENCHMARKS ($3 \leq x \leq 33$)

Circuit	*Optimum Number of Codewords (x^*)*			
	$W_{CR} = W_{TR} > .4$ $W_{DC} < .2$	$.35 < W_{CR} = W_{TR} \leq .4$ $.2 \leq W_{DC} < .3$	$.2 < W_{CR} = W_{TR} \leq .35$ $.3 \leq W_{DC} < .6$	$W_{CR} = W_{TR} \leq .2$ $W_{DC} \geq .6$
s5378	9	5	3	3
s9234	9	9	5	3
s13207	9	9	5	3
s15850	9	9	5	3
s38417	9	9	5	3
s38584	9	9	5	3

VII. CONCLUSION

This paper presents a new compression technique using only nine fixed codewords and fixed or variable-length blocks called 9C and V9C, respectively. Our technique aims at precomputed data of IP cores in Soc's and does not require any structural information of cores. 9C is very simple but flexible coding technique that uses a small decoder. V9C uses variable-length block for each pattern to achieve higher compression ratio. We also proposed two implementations which are based on dependency to the precomputed test-data set. Data-independent V9C, which sends each pattern's block length along with codewords, is test-data independent and increases compression compared to original 9C. Dictionary-based V9C achieves very high compression compared to 9C and data-independent V9C. Applying V9C to ISCAS'89 benchmarks has shown up to 90% compression ratio while the decompression logic remains quite small.

REFERENCES

[1] Y. Zorian, E. Marinissen, and S. Dey, "Testing embedded-core-based system chips," *Comput. Mag.*, vol. 32, no. 6, pp. 52–60, 1999.

[2] S. Hellebrand, H. Linag, and H.-J. Wunderlich, "A mixed-mode BIST scheme based on reseeding of folding counters," in *Proc. Int. Test Conf. (ITC'00)*, 2000, pp. 778–784.

[3] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *Proc. Int. Test Conf. (ITC'00)*, 1996, pp. 337–343.

[4] N. Touba and E. McCluskey, "Altering a pseudo-random bit sequence for scan based BIST," in *Proc. Int. Test Conf. (ITC'00)*, 1996, pp. 167–175.

[5] V. Iyengar, K. Chakrabarty, and B. Murray, "Built-in self testing of sequential circuits using precomputed test sets," in *Proc. VLSI Test Symp. (VTS'98)*, 1998, pp. 418–423.

[6] A. Jas, J. Ghosh-Dastidar, and N. Touba, "Scan vector compression/decompression using statistical coding," in *Proc. VLSI Test Symp. (VTS'99)*, 1999, pp. 114–120.

[7] A. Jas, J. Gosh-Dastidar, M. Ng, and N. Touba, "An efficient test vector compression scheme using selective Huffman coding," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 22, no. 6, pp. 797–806, Jun. 2003.

[8] M. Nourani and M. Tehranipour, "RL-Huffman encoding for test compression and power reduction in scan application," *ACM Trans. Design Automat. Electron. Syst.*, vol. 10, no. 1, pp. 91–115, Jan. 2005.

[9] A. Chandra and K. Chakrabarty, "System-on-a-chip data compression and decompression architecture based on Golomb codes," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 20, no. 3, pp. 355–368, Mar. 2001.

[10] ——, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes," *IEEE Trans. Comput.*, vol. 52, no. 8, pp. 1076–1088, Aug. 2003.

[11] ——, "A unified approach to reduce SoC test data volume, scan power, and testing time," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 22, no. 3, pp. 352–363, Mar. 2003.

[12] A. El. Maleh and R. Al-Abaji, "Extended frequency-directed run-length codes with improved application to system-on-a-chip test data compression," in *Proc. Int. Conf. Electronic Circuits Systems (ICECS'02)*, 2002, pp. 449–452.

[13] P. Rosinger, P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Simultaneous reduction in volume of test data and power dissipation for system-on-a-chip," *Electron. Lett.*, vol. 37, no. 24, pp. 1434–1436, 2001.

[14] P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," in *Proc. Design Automation Test in Europe (DATE'02)*, 2002, pp. 604–611.

[15] F. Hsu, K. Butler, and J. Patel, "A case study on the implementation of the Illinois scan architecture," in *Proc. Int. Test Conf. (ITC'01)*, 2001, pp. 538–547.

[16] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. Design Automation Conf. (DAC'01)*, 2001, pp. 151–155.

[17] A. El-Maleh, S. Al Zahir, and E. Khan, "A geometric-primitives-based compression scheme for testing system-on-chip," in *Proc. VLSI Test Symp. (VTS'01)*, 2001, pp. 54–59.

[18] S. Reda and A. Orailoglu, "Reducing test application time through test data mutation encoding," in *Proc. Design Automation Test in Europe (DATE'02)*, 2002, pp. 387–393.

[19] L. Schafer, R. Dorsch, and H.-J. Wunderlich, "RESPIN++-deterministic embedded test," in *Proc. Eur. Test Workshop (ETW'02)*, 2002, pp. 37–44.

[20] E. Volkerink, A. Khoche, and S. Mitra, "Packet-based input test data compression technique," in *Proc. Int. Test Conf. (ITC'02)*, 2002, pp. 167–175.

[21] C. Krishna, A. Jas, and N. Touba, "Test vector encoding using partial LFSR reseeding," in *Proc. Int. Test Conf. (ITC'01)*, 2001, pp. 885–893.

[22] E. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *Proc. VLSI Test Symp. (VTS'03)*, 2003, pp. 232–237.

[23] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 776–792, May 2004.

[24] S. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain designs," in *Proc. VLSI Test Symp. (VTS'02)*, 2002, pp. 103–108.

[25] I. Pomeranz and S. Reddy, "Test data volume reduction by test data realignment," in *Proc. Asian Test Symp. (ATS'03)*, 2003, pp. 434–439.

[26] F. Wolff and C. Papachristou, "Multiscan-based test compression and hardware decompression using LZ77," in *Proc. Int. Test Conf. (ITC'02)*, 2002, pp. 331–339.

[27] M. Knieser, F. Wolff, C. Papachristou, D. Weyer, and D. McIntyre, "A technique for high ratio LZW compression," in *Proc. Design Automation Test in Europe (DATE'03)*, 2003, pp. 116–121.

[28] L. Li and K. Chakrabarty, "Test data compression using dictionaries with fixed length indices," in *Proc. VLSI Test Symp. (VTS'03)*, 2003, pp. 219–224.

[29] A. Wurtenberger, C. Tautermann, and S. Hellebrand, "A hybrid coding strategy for optimized test data compression," in *Proc. Int. Test Conf. (ITC'03)*, 2003, pp. 451–459.

[30] M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique with application to reduced pin-count testing and flexible on-chip decompression," in *Proc. Design Automation Test in Europe (DATE'04)*, 2004, pp. 1284–1289.

[31] T. Hiraide, K. Boateng, H. Konishi, K. Itaya, M. Emori, and H. Yamanaka, "BIST-aided scan test—A new method for test cost reduction," in *Proc. VLSI Test Symp. (VTS'03)*, 2003, pp. 359–364.

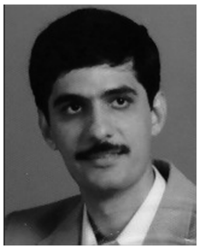
[32] *User Manuals for SYNOPSYS Toolset Version 2003.12*, Synopsys, Inc., Mountain View, CA, 2003.

[33] M. Tehranipoor, "Enhanced scan architectures for improving application time and power," Ph.D. disseration, Univ. Texas, Dallas, Aug. 2004.

Mohammad Tehranipoor (S'02–M'04) received the B.Sc. degree in electrical engineering from Amirkabir University of Technology (Tehran Polytechnic University), Tehran, Iran, the M.Sc. degree in electrical engineering from the University of Tehran, Tehran, and the Ph.D. degree in electrical engineering from University of Texas at Dallas, Richardson, in 1997, 2000, and 2004, respectively.

He has worked in the VLSI Circuits and Systems Laboratory, ECE Department, University of Tehran from 1998 to 2002 as a Research Assistant, Research Associate, and Laboratory Administrator. He also worked as a Research Assistant in the Center for Integrated Circuits and Systems, Department of Electrical Engineering, University of Texas. He is currently an Assistant Professor of electrical and computer engineering at the University of Maryland Baltimore County, Baltimore. His current research interests include computer-aided design and test, design-for-testability, signal integrity modeling and test, test resource partitioning, and DSP architectures. He has published over 25 journal articles and refereed conference papers in the area of VLSI design and test. He is a reviewer for many journals, magazines, and conferences.

Dr. Tehranipoor is a member of the program committee of the IEEE North Atlantic Test Workshop (NATW). He is a member of ACM and ACM SIGDA.



Mehrdad Nourani (S'91–M'94–SM'05) received the B.Sc. and M.Sc. degrees in electrical engineering from the University of Tehran, Tehran, Iran, in 1986, and the Ph.D. degree in computer engineering from Case Western Reserve University, Cleveland, OH, in 1993.

During the academic year 1994, he was a Postdoctoral Fellow at the Department of Computer Engineering, Case Western Reserve University. He was with the Department of Electrical and Computer Engineering, University of Tehran from 1995 to 1998 and the Department of Electrical Engineering and Computer Science, Case Western Reserve University from 1998 to 1999. Since August 1999, he has been on the faculty of the University of Texas at Dallas, Richardson, where he is currently an Associate Professor of electrical engineering and a member of Center for Integrated Circuits and Systems. His current research interests include design for testability, system-on-chip testing, signal integrity modeling and test, application specific processor architectures, packet processing devices, high-level synthesis, and low-power design methodologies. He has published over 120 papers in journals and refereed conference proceedings.

Dr. Nourani received the Texas Telecommunications Consortium Award (1999), The Clark Foundation Research Initiation Grant (2001), the National Science Foundation Career Award (2002), and Cisco Systems Inc. URP Award (2004). He is a member of the IEEE Computer Society and the ACM SIGDA.

Krishnendu Chakrabarty (S'92–M'96–SM'00) received the B.Tech. degree from the Indian Institute of Technology, Kharagpur, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively, all in computer science and engineering.

He is currently an Associate Professor of electrical and computer engineering at Duke University, Durham, NC. During 2000–2002, he was also a Mercator Visiting Professor at University of Potsdam in Germany. His current research projects include design and testing of system-on-chip integrated circuits, embedded real-time systems, distributed sensor networks, design automation tools for microfluidics-based biochips, and microfluidics-based chip cooling. He is a coauthor of two books: *Microelectrofluidic Systems: Modeling and Simulation* (Boca Raton, FL: CRC Press, 2002) and *Test Resource Partitioning for System-on-a-Chip* (Amsterdam, The Netherlands: Kluwer, 2002), and the editor of *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation* (Amsterdam, The Netherlands: Kluwer, 2002). He has published over 170 papers in journals and refereed conference proceedings, and he holds a U.S. patent in built-in self-test.. He is an Editor of *Journal of Electronic Testing: Theory and Applications (JETTA)*, and a member of the editorial board for *ACM Transactions on Emergering Technologies in Computing Systems*, *Sensor Letters*, and the *Journal of Embedded Computing*

Dr. Chakrabarty is a recipient of the Humboldt Research Fellowship, awarded by the Alexander von Humboldt Foundation, Germany, and is a recipient of the National Science Foundation Early Faculty (CAREER) award, a Best Paper Award at the 2001 Design, Automation, and Test in Europe (DATE) Conference, and the Office of Naval Research Young Investigator Award. He serves as Vice Chair of Technical Activities in IEEE's Test Technology Technical Council and is a member of the program committees of several IEEE/ACM conferences and workshops. He is an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He has also served as an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II—ANALOG AND DIGITAL SIGNAL PROCESSING. He is a member of ACM and ACM SIGDA and a member of Sigma Xi.