JOURNAL OF ELECTRONIC TESTING: Theory and Applications 15, 97–114 (1999)
© 1999 Kluwer Academic Publishers. Manufactured in The Netherlands.

Deterministic Built-in Pattern Generation for Sequential Circuits*

VIKRAM IYENGAR
Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 1308 W Main Street, Urbana IL 61801, USA
vik@crhc.uiuc.edu

KRISHNENDU CHAKRABARTY
Department of Electrical and Computer Engineering, Duke University, Box 90291, 130 Hudson Hall, Durham, NC 27708, USA
krish@ee.duke.edu

BRIAN T. MURRAY
Delphi Automotive Systems, 3900 East Holland Road, Saginaw, MI 48601-9494, USA
bmurray@predator.cs.gmr.com

Received June 10, 1998

Editors: M. Nicolaidis and R. Roy

Abstract. We present a new pattern generation approach for deterministic built-in self testing (BIST) of sequential circuits. Our approach is based on precomputed test sequences, and is especially suited to sequential circuits that contain a large number of flip-flops but relatively few controllable primary inputs. Such circuits, often encountered as embedded cores and as filters for digital signal processing, are difficult to test and require long test sequences. We show that statistical encoding of precomputed test sequences can be combined with low-cost pattern decoding to provide deterministic BIST with practical levels of overhead. Optimal Huffman codes and near-optimal Comma codes are especially useful for test set encoding. This approach exploits recent advances in automatic test pattern generation for sequential circuits and, unlike other BIST schemes, does not require access to a gate-level model of the circuit under test. It can be easily automated and integrated with design automation tools. Experimental results for the ISCAS 89 benchmark circuits show that the proposed method provides higher fault coverage than pseudorandom testing with shorter test application time and low to moderate hardware overhead.

Keywords: BIST, Comma coding, embedded-core testing, Huffman coding, pattern decoding, run-length encoding, sequential circuit testing, statistical encoding

1. Introduction

Built-in self testing (BIST) offers an attractive solution to the problem of testing electronic systems [1]. The design of test generator circuits (TGCs) for BIST has been studied widely and several TGC design techniques have been developed for combinational and full-scan sequential circuits. A number of design automation tools are available today for adding BIST logic to enhance testability. However, these techniques are not directly applicable to non-scan and partial-scan sequential circuits. Many performance-driven designs and embedded-core circuits do not use full scan—in

*This research was supported in part by the National Science Foundation under Grant no. CCR-9875324, a contract from Delphi Delco Electronics Systems, and an equipment grant from Sun Microsystems. A preliminary version of this paper appeared in *Proc. IEEE VLSI Test Symposium*, pp. 418–423, April 1998.

fact, hundreds of non-scan "legacy cores" exist today [2]. To test them using known BIST methods requires substantial redesign to "stitch in" scan chains. This can lead to a considerable increase in the design cycle time, defeating the very purpose of using embedded cores.

A few design automation techniques for non-scan sequential circuit BIST have been proposed recently [3–6]. These methods are based on pseudorandom test sequences and do not always yield the same fault coverage as deterministic sequences. They also require excessively high test application times. Moreover, they require a gate-level circuit model, either for modifying the circuit flip-flops [3, 5], for test-point insertion [4], or for carrying out extensive fault simulation [5, 6]. Such gate-level models may not be readily available for embedded cores and thus the proposed methods cannot be used for these circuits. We present a new approach for designing TGCs that apply precomputed test sets to non-scan and partial scan circuits. Such test sets do not disclose substantial proprietary information and hence may be easily provided by core vendors. The precomputed test sets can be obtained from ATPG tools tailored to any specific fault model. Recent advances in sequential circuit test generation have led to techniques and tools that provide such test sets for single stuck-line (SSL) faults with high fault coverage [7–11].

A straightforward TGC design for sequential circuit testing involves the use of a ROM to store the precomputed test set T_D. However, this is not considered practical because of the silicon area required to store the entire test set in a ROM. A more practical alternative is to encode T_D and store (or generate) only the compressed (encoded) test set T_E, which can then be decoded during test application. A generic TGC that employs encoding is shown in Fig. 1. The sequence generator *SG* feeds a k-bit-wide sequence of compressed (encoded) patterns to a decoder circuit *DC* that expands (decodes) these into n-bit-wide test patterns, where n is the number of controllable primary inputs in the circuit under test (CUT). This decomposition of the TGC is similar to that described in [12] for combinational and full-scan circuits, with the additional requirement that the order of patterns in T_D must be preserved exactly.

Actually, a typical test set T_D for sequential circuits consists of one or more fixed test subsequences, i.e., $T_D = \{T_1, T_2, \ldots, T_k\}$. Each T_i must be preserved exactly, but is independent from T_j, $i \neq j$ for $i, j \in \{1, 2, \ldots, k\}$. Therefore, T_i and T_j may be interchanged and additional test vectors may appear between them. In the following, to simplify the design automation problem and the subsequent analysis, we assume that test sets consist only of a single fixed sequence.

One approach to test sequence encoding is to employ fixed-length codes for each test pattern. Let the number of unique test patterns in the test set be m. Every pattern can then be encoded with $q = \lceil \log_2 m \rceil$ bits. The sequence generator in this case can be a ROM that stores the encoded test set T_E. However, this approach frequently does not provide sufficient compaction of the test data to make on-chip storage practical. Moreover, the unstructured nature of *DC* leads to significant overhead for decoding. The problem with this encoding scheme is that it does not take into account the frequency of occurrence of the various patterns in T_D—significantly more compression is still possible.

Our approach to test set encoding is motivated by the fact that sequential circuit test sets typically contain a large number of repeated patterns. For example, the test set for the s444 ISCAS 89 benchmark circuit [13] obtained from the Gentest ATPG program [7] contains 1881 patterns, of which only 8 are unique. Such test sets are good candidates for statistical encoding, in which variable-length codewords are used to encode the test patterns, more frequent patterns being encoded with fewer bits. We present a deterministic BIST approach based on statistical encoding that exploits this

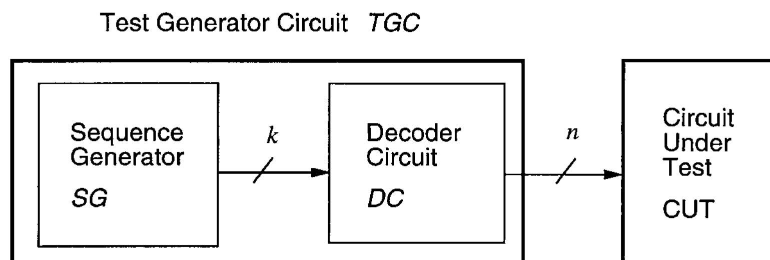

Fig. 1. A generic BIST test generator circuit.

feature of sequential circuit test sets. The proposed approach is especially useful for circuits such as filters, used in digital signal processing (DSP), that have few primary inputs but a large number of flip-flops. These circuits, which require long test sequences, are typical of embedded cores used in complex DSP designs [14]. A number of ISCAS 89 benchmark circuits also belong to this category. Full scan is often not practical for these circuits because of the relatively large number of flip-flops; the overhead is substantial and there are few combinational gates to test between scan chains.

The paper is organized as follows. In Section 2, we discuss statistical encoding and motivate its use for encoding a given precomputed test set T_D. In Section 3, we describe two specific statistical encoding techniques: optimal Huffman codes and near-optimal Comma codes. We also describe run-length encoding, a technique for compressing an already encoded test set further. For each encoding method, we develop simple and efficient decoding methods and, as an example, present the decoder implementations for the s444 circuit from the ISCAS 89 benchmark set. Section 4 presents experimental results on test set compression and decoder design for the ISCAS 89 circuits. We also compare our approach with a recently-proposed pseudorandom BIST method for sequential circuits. These results demonstrate that Huffman, Comma and run-length coding of test patterns lead to effective encoding of T_D and practical, deterministic TGCs for sequential circuits.

2. Statistical Encoding

Statistical encoding methods exploit the unequal probabilities of occurrence of patterns to minimize the average length of a codeword. Short codewords are used to represent frequently-occurring patterns while longer words are used for less frequently encountered patterns. Such lossless data compression codes have been widely employed in digital communication and in such computer applications as instruction encoding [15], but not typically in BIST because deterministic testing is widely assumed to be too costly. Examples of statistical encoding methods include Huffman coding, Shannon-Fano coding and Lempel-Ziv string encoding [16, 17]. We review Huffman and Comma coding here, since these seem to be particularly useful for encoding T_D. We also compare optimal Huffman coding to fixed length codes, to more precisely analyze the advantages of statistical encoding.

Optimal Test Set Encoding: Huffman Codes

An optimal encoding technique minimizes the average length of a codeword. Consider a test set T_D containing m unique patterns $X_1, X_2, \ldots, X_m$ with probabilities of occurrence $p_1, p_2, \ldots, p_m$, respectively. The entropy, defined intuitively as the minimum average number of bits required to represent a pattern, is given by $H(T_D) = -\sum_{i=1}^{m} p_i \log_2 p_i$. Therefore, an optimal encoding technique is one in which the average number of bits needed to represent a pattern is closest to the entropy bound. The Huffman code is provably optimal since it results in the shortest average codeword length among all statistical encoding techniques that assign a unique binary codeword to each pattern [18, 19]. In fact, if l_H is the average length of a Huffman-encoded pattern in T_D, then $H(T_D) \leq l_H \leq H(T_D) + 1$ [18]. In addition, Huffman codes possess the prefix-free property, i.e., no codeword is the prefix of a longer codeword. We will show that this property is important for test sequence decoding [20].

Table 1 illustrates the Huffman code for an example test set T_D with four unique patterns out of a total of eighty. Column 1 of Table 1 lists the four patterns, column 2 lists the corresponding number of occurrences f_i of each pattern X_i, and column 3 lists the corresponding probability of occurrence p_i, given by $f_i/|T_D|$. Finally, column 4 gives the corresponding Huffman code for each unique pattern. Note that the most common pattern X_1 is encoded with a single 0 bit; that is $e(X_1) = 0$, where $e(X_1)$ is the codeword for X_1. Since no codeword appears as a prefix of a longer codeword (the prefix-free property), if a sequence of encoded test vectors is treated as a serial bit-stream, decoding can be done as soon as the last bit of a codeword is read. This property is essential since variable-length codewords cannot be read from memory as words in the usual fashion.

The Huffman code illustrated in Table 1 can be constructed by generating a binary tree (Huffman tree) with

Table 1. Test set encoding for a simple example test sequence of 80 test patterns.

Unique patterns	Occurrences	Probability of occurrence	Huffman codeword	Comma codeword
X_1: 0000	45	0.5625	0	0
X_2: 0101	15	0.1875	10	10
X_3: 1010	15	0.1875	110	110
X_4: 1111	5	0.0625	111	1110

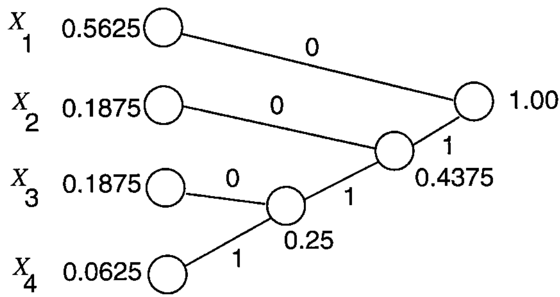

Fig. 2. An example illustrating the construction of the Huffman code.

edges labeled either 0 or 1 as illustrated in Fig. 2. Each unique pattern X_i of Table 1 is associated with a (leaf) node of the tree, which initially consists only of these unmarked nodes. The Huffman coding procedure iteratively selects two nodes v_i and v_j with the lowest probabilities of occurrence, marks them, and generates a parent node v_{ij} for v_i and v_j. If these two nodes are not unique, then the procedure arbitrarily chooses two nodes with the lowest probabilities. The edges (v_{ij}, v_i) and (v_{ij}, v_j) are labeled 0 and 1. The 0 and 1 labels are chosen arbitrarily, and do not affect the amount of compression [18]. The node v_{ij} is assigned a probability of occurrence $p_{ij} = p_i + p_j$. This process is continued until there is only one unmarked node left in the tree.

Each codeword $e(X_i)$ is obtained by traversing the path from the root of the Huffman tree to the corresponding leaf node v_i. The sequence of 0–1 values on the edges of this path provides the $e(X_i)$. The Huffman coding procedure has a worst case complexity of $O(m^2 \log_2 m)$, thus the encoding can be done in reasonable time. The average number of bits per pattern l_{H} (average length of a codeword) is given by $l_{\mathrm{H}} = \sum_{i=1}^{m} w_i p_i$, where w_i is the length of the codeword corresponding to test pattern X_i. The average length of a codeword in our example is therefore given by $l_{\mathrm{H}} = 1 \times 0.5625 + 2 \times 0.1875 + 3 \times 0.1875 + 3 \times 0.0625 = 1.68$ bits.

We next compare Huffman coding with equal-length coding. Let l_{H} (l_{E}) be the average length of a codeword for Huffman coding (equal-length coding). Since Huffman coding is optimal, it is clear that $l_{\mathrm{H}} \leq l_{\mathrm{E}}$. We next show that $l_{\mathrm{H}} = l_{\mathrm{E}}$ under certain conditions.

Theorem 1. *If all unique patterns have the same probability of occurrence and the number of unique patterns m is a power of* 2, *then* $l_H = l_E$.

Proof: If all the unique patterns in T_{D} have the same probability of occurrence $p = 1/m$, the entropy $H(T_{\mathrm{D}}) = -\sum_{i=1}^{m} p_i \log_2 p_i = \log_2 m$ bits. For equal-length encoding, $l_{\mathrm{E}} = \lceil \log_2 m \rceil$, and if m is a power of 2 then $l_{\mathrm{E}} = \log_2 m$, which equals the entropy bound. Therefore, $l = l_{\mathrm{E}}$ for this case. ■

The above theorem can be restated in a more general form in terms of the structure of the Huffman tree.

Theorem 2. *If the Huffman tree is a full binary tree, then* $l_H = l_E$.

Proof: A full binary tree with k levels has $2^k - 1$ vertices, out of which 2^{k-1} are leaf vertices. Therefore if the Huffman tree is a full binary tree with m leaf vertices, then m must be a power of 2 and the number of levels must be $\log_2 m + 1$. It follows that every path from the root to a leaf vertex is then of length $\log_2 m$. ■

If all unique patterns in T_{D} have the same probability of occurrence and m is a power of 2, then the Huffman tree is indeed a full binary tree; Theorem 2 therefore implies Theorem 1. Note that Theorem 2 is sufficient but not necessary for l_{H} to equal l_{E}. Figure 3 shows a Huffman tree for which $l_{\mathrm{H}} = l_{\mathrm{E}} = 2$, even though it is not a full binary tree.

The practical implication of Theorem 1 is that Huffman encoding will be less useful when the probabilities of all of the unique test patterns are similar. This tends to happen, for instance, when the ratio of the number of flip-flops to the number of primary inputs in the CUT, denoted γ in Section 4, is low. Theorem 2 suggests that even when γ is high, the probability distribution of the unique test patterns should be analyzed to

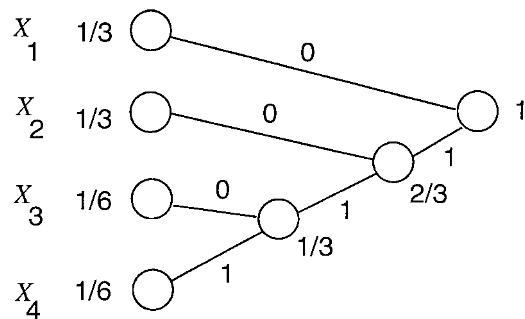

Fig. 3. An example of a non-full binary tree with $l = l_{\mathrm{E}}$.

determine if statistical encoding is worthwhile. However, in all cases we analyzed where γ was high, statistical encoding was indeed effective.

Comma Codes

Although Huffman codes provide optimal test set compression, they do not always yield the lowest-cost decoder circuit. Therefore, we also employ a non-optimal code, namely the Comma code, which often leads to more efficient decoder circuits. The Comma code, also prefix-free, derives its name from the fact that it contains a terminating symbol, e.g. 0, at the end of each codeword.

The Comma encoding procedure first sorts the unique patterns in decreasing order of probability of occurrence, and encodes the first pattern (i.e., the most probable pattern) with a 0, the second with a 10, the third with a 110, and so on. The procedure encodes each pattern by adding a 1 to the beginning of the previous codeword. The codeword for the ith unique pattern X_i is thus given by a sequence of $(i-1)$ 1s followed by a 0. Comma codewords for the unique patterns in the example test set of Section 2.1 are listed in Column 5 of Table 1. This procedure has complexity $O(m \log_2 m)$ and is simpler than the Huffman encoding procedure. The Comma code also requires a substantially simpler decoder *DC* than the Huffman code. Since each Comma codeword is essentially a sequence of 1s followed by a zero, the decoder only needs to maintain a count of the number of 1s received before a 0 signifies the end of a codeword. The 1s count can then be mapped to the corresponding test pattern.

For a given test sequence T_{D} with m unique patterns having probabilities of occurrence $p_1 \geq p_2 \geq \cdots \geq p_m$, the average length of a Comma codeword is given by $l_{\mathrm{C}} = \sum_{i=1}^{m} i p_i$. Since the code is non-optimal, $l_{\mathrm{C}} \geq l_{\mathrm{H}}$. However, the Comma code provides near-optimal compression, i.e., $\lim_{m\to\infty}(l_{\mathrm{C}} - l_{\mathrm{H}}) = 0$, if T_{D} satisfies certain properties. These hold for typical test sequences that have a large number of repeated patterns. We first present the condition under which Comma codes are near-optimal and then the property of T_{D} required to satisfy the condition.

A binary tree with leaf nodes $X_1, X_2, \ldots, X_m$ is *skewed* if the distance d_i of X_i from the root is given by

$$d_i = \begin{cases} i, & i \in \{1, 2, \ldots, m-1\} \\ m-1, & \text{otherwise} \end{cases} \tag{1}$$

For instance, the Huffman tree of Fig. 2 is a skewed binary tree with four leaf nodes.

Theorem 3. *Let $p_1 \geq p_2 \geq \cdots \geq p_m$ be the probabilities of occurrence of the m unique patterns in T_D. Let l_H and l_C be the average length of the codewords for Huffman and Comma codes, respectively. If the Huffman tree for T_D is skewed then $l_C - l_H = p_m$ and $\lim_{m\to\infty}(l_C - l_H) = 0$.*

Proof: If the Huffman tree for T_{D} is skewed then $l_{\mathrm{H}} = \sum_{i=1}^{m-1} i p_i + (m-1)p_m$ and $l_{\mathrm{C}} = \sum_{i=1}^{m} i p_i$. Therefore, $l_{\mathrm{C}} - l_{\mathrm{H}} = p_m$. We also know that $p_1 \geq p_2 \geq \cdots p_m$. Therefore, $0 \leq p_m \leq 1/m$, which implies that $\lim_{m\to\infty} p_m = 0$. Hence $l_{\mathrm{C}} - l_{\mathrm{H}}$ is vanishingly small for a skewed Huffman tree and the Comma code is near-optimal. ∎

Next we derive a necessary and sufficient condition that T_{D} must satisfy in order for its Huffman tree to be skewed.

Theorem 4. *Let $p_1 \geq p_2 \geq \cdots \geq p_m$ be the probabilities of occurrence of the unique patterns in T_D. The Huffman tree for T_D is skewed if and only if, for $1 \leq i \leq m-2$, the probabilities of occurrence satisfy the condition*

$$p_i \geq \sum_{k=i+2}^{m} p_k. \tag{2}$$

Proof: We prove sufficiency of the theorem. The necessity can be proven similarly. Generate the Huffman tree for the m patterns $X_1, X_2, \ldots, X_m$ in T_{D} whose probabilities of occurrence satisfy (2). Let the leaf node corresponding to the ith pattern be v_i. The two leaf nodes v_m and v_{m-1} corresponding to the patterns X_m and X_{m-1} with the lowest probabilities p_m and p_{m-1} are first selected, and a parent node $v_{m(m-1)}$, with the probability $(p_m + p_{m-1})$, is generated for them. Now, $p_{m-3} \geq p_{m-2}$, and from (2), $p_{m-3} \geq \sum_{k=m-1}^{m} p_k$. Thus, $p_{m-3} \geq p_{m-1} + p_m$. Therefore, the leaf node v_{m-2} and $v_{m(m-1)}$ are now the two nodes with the lowest probabilities, and a parent $v_{m(m-1)(m-2)}$ with probability $(p_m + p_{m-1} + p_{m-2})$ is generated for them.

Similarly, a parent $v_{m(m-1)\cdots i}$ is generated for nodes v_i and $v_{m(m-1)\cdots(i+1)}$, $i \in \{(m-3)\cdots 1\}$. The process terminates when the root $v_{m(m-1)\cdots 1}$ is generated for leaf node v_1 and $v_{m(m-1)\cdots 2}$. The distance d_1 of v_1 to the root is therefore 1. Similarly $d_i = i$, $i \in \{2, 3, \ldots, m-1\}$. Leaf nodes v_m and v_{m-1} are equidistant from the root,

since they share a common parent $v_{m(m-1)}$, thus $d_m = m-1$. Therefore, d_i satisfies (1) for $i \in \{1, 2, \ldots, m\}$, and the Huffman tree is skewed. ■

We next determine the relationship between $|T_D|$ and m, the number of unique patterns in the test set when the Huffman tree is skewed. We show that $|T_D|$ must be exponential in m for the Huffman tree to be skewed. This property is often satisfied by deterministic test sets for sequential circuits with a large number of flip-flops but few primary inputs.

Theorem 5. *Let $|T_D|$ and m be the total number of patterns and the number of unique patterns in T_D, respectively. If the Huffman tree for T_D is skewed then $|T_D| = \omega(1.62^m)$, where ω denotes an asymptotic lower bound in the sense that $f(m) = \omega(g(m))$ implies $\lim_{m\to\infty} \frac{f(m)}{g(m)} = \infty$.*

Proof: Let $f_1 \geq f_2 \geq \cdots \geq f_m$ be the numbers of occurrence of the unique patterns in T_D. Then $|T_D| = \sum_{i=1}^{m} f_i$. We know that $f_{m-1} \geq f_m \geq 1$, and $f_{m-2} \geq f_m \geq 1$. From (2), $f_{m-3} \geq f_{m-1} + f_m \geq 2$. Similarly, $f_{m-4} \geq 3$ and $f_{m-5} \geq 5$. The lower bounds on $f_{m-1}, \ldots, f_1$ thus form the Fibonacci series $1, 1, 2, 3, 5, \ldots$; therefore, $|T_D| \geq 1 + \sum_{i=1}^{m-1} s_i$, where s_i is the ith Fibonacci term, given by $s_i = \frac{1}{\sqrt{5}}(\phi^{i+1} - \hat{\phi}^{i+1})$, where $\phi = \frac{1}{2}(1+\sqrt{5})$ and $\hat{\phi} = \frac{1}{2}(1-\sqrt{5})$ [21].

Now, $|T_D| > s_{m-1} = \frac{1}{\sqrt{5}}(\phi^m - \hat{\phi}^m)$. For even m,

$$\begin{aligned}|T_D| &> \frac{1}{\sqrt{5}}(1.62^m - 0.62^m)\\ &= \frac{1}{\sqrt{5}}(1.62 - 0.62)(1.62^{m-1}\\ &\quad + 1.62^{m-2}0.62 + \cdots),\end{aligned}$$

from which it follows that $|T_D| = \omega(1.62^m)$. The proof for odd m is similar. ■

Comma codes, being non-optimal, do not always yield better compression than equal-length codes. The following theorem establishes a sufficient condition under which Comma codes perform worse than equal-length codes.

Theorem 6. *Let $p_1 \geq p_2 \geq \cdots \geq p_m$ be the probabilities of occurrence of the unique patterns in T_D. If $p_m > \frac{2\lceil \log_2 m \rceil}{m(m+1)}$, then $l_C > l_E$, where l_C (l_E) is the average codeword length for Comma (equal-length) coding.*

Proof: We know that $l_E = \lceil \log_2 m \rceil$ and $l_C = p_1 + 2p_2 + \cdots + mp_m$. Since $p_1 \geq p_2 \geq \cdots \geq p_m$, $l_C \geq p_m(1 + 2 + \cdots + m) = \frac{1}{2}p_m m(m+1)$. Thus $l_C > l_E$ if $\frac{1}{2}p_m m(m+1) > \lceil \log_2 m \rceil$, from which the theorem follows. ■

For example, in the test set for s35932 obtained using Gentest, $m = 86$ and $p_m = 0.012$, while $2\lceil \log_2 m \rceil/86(87) = 0.0019$. Hence, Comma coding performs worse than equal-length coding for this test set.

Note that Theorem 6 not provide a necessary condition for which $l_C > l_E$. In fact, it is easy to construct data sets for which $l_C > l_E$ even though $p_m \leq \frac{2\lceil \log_2 m \rceil}{m(m+1)}$. The following theorem provides a tighter condition under which Comma codes perform worse than equal-length codes.

Theorem 7. *Let $p_1 \geq p_2 \geq \cdots \geq p_m$ be the probabilities of occurrence of the unique patterns in T_D, and let $l_C(l_E)$ be the average codeword length for Comma coding (equal-length coding). Let $\alpha = \min_i\{\frac{p_i}{p_{i+1}}\}$, $1 \leq i \leq m-1$. Then $l_C > l_E$ if*

$$p_m \geq \frac{(\alpha - 1)^2 \lceil \log_2 m \rceil}{\alpha^{m-1} - \alpha(m+1) + m}. \tag{3}$$

Proof: We first note that $p_i \geq \alpha p_{i+1} \geq \alpha^2 p_{i+2} \geq \cdots \geq \alpha^{m-i} p_m$, $1 \leq i \leq m$. Since $l_C = \sum_{i=1}^{m} i p_i$, it follows that $l_C \geq \alpha^{m-1}p_m + 2\alpha^{m-2}p_m + 3\alpha^{m-3}p_m + \cdots + (m-1)\alpha p_m + mp_m$. Let E be defined as

$$\begin{aligned}E &= \alpha^{m-1}p_m + 2\alpha^{m-2}p_m + 3\alpha^{m-3}p_m + \cdots\\ &\quad + (m-1)\alpha p_m + mp_m,\end{aligned} \tag{4}$$

such that $l_C \geq E$. From (4), we get

$$\begin{aligned}\alpha E &= \alpha^m p_m + 2\alpha^{m-1}p_m + 3\alpha^{m-2}p_m + \cdots\\ &\quad + (m-1)\alpha^2 p_m + m\alpha p_m,\end{aligned} \tag{5}$$

From (4) and (5), we obtain

$$\begin{aligned}(\alpha - 1)E &= \alpha^m p_m + \alpha^{m-1}p_m + \cdots + \alpha p_m - mp_m\\ &= \alpha p_m\left(\frac{\alpha^m - 1}{\alpha - 1}\right) - mp_m\end{aligned} \tag{6}$$

Now, if $E > l_E$, i.e. $E > \lceil \log_2 m \rceil$, then $p_m \geq \frac{(\alpha-1)^2\lceil \log_2 m \rceil}{\alpha^{m-1} - \alpha(m+1) + m}$ and the theorem follows. ■

If $m \gg 1$ then (3) can be simplified to $p_m \geq \frac{(\alpha-1)^2\lceil \log_2 m \rceil}{\alpha^{m-1} - m(\alpha-1)}$. In addition, if $\min_i\{\frac{p_i}{p_{i+1}}\}$ exceeds 2, i.e.

Table 2. Huffman and Comma code words for the patterns in the test set of s444.

Test pattern	Occurrences	Probability of occurrence	Huffman codeword	Comma codeword
000	1631	0.8671	0	0
010	139	0.0738	10	10
001	93	0.0494	110	110
011	7	0.0037	1110	1110
110	5	0.0026	11110	11110
101	3	0.0015	111110	111110
111	2	0.0011	1111110	1111110
100	1	0.0005	1111111	11111110

every data pattern occurs twice as often the next most-frequent data pattern, then we can replace α by 2 in (3) to obtain the following simpler sufficient condition under which Comma codes perform worse than equal-length codes.

Corollary 1. *Let $p_1 \geq p_2 \geq \cdots \geq p_m$ be the probabilities of occurence of the unique patterns in T_D, and let l_C (l_E) be the average codeword length for Comma coding (equal-length coding). If $\alpha = \min_i\{\frac{p_i}{p_{i+1}}\} > 2$ and $p_m > \frac{\lceil \log_2 m \rceil}{2^{m-1}-m-2}$, then $l_C > l_E$.*

However, the skewing probability distribution property of Theorem 4 appears to be easy to satisfy in most cases. The probabilities of occurrence of patterns for a typical case (the s444 test set) are shown in Table 2 in Section 3. The decrease in compression resulting from the use of Comma codes, instead of optimal (Huffman) codes to compress such test sets, which is given by $l_C - l_H = p_m$ from Theorem 3, is extremely small in practice. For the s444 test set, $p_m = 0.0005, l_H = 1.2121$, and therefore $l_C = 1.2126$, and the compression loss is only one bit. Therefore, both Huffman and Comma codes can efficiently encode sequential circuit test sets.

3. TGC Design

In this section, we illustrate our methods for constructing TGCs employing statistical encoding of precomputed test sequences. We illustrate the steps involved in encoding and decoding with the test set for the s444 benchmark circuit as an example.

Huffman Coding

The first step in the encoding process is to identify the unique patterns in the test set. A codeword is then developed for each unique pattern using the Huffman code construction method outlined in Section 2. The Huffman tree used to construct codewords for the patterns of s444 is shown in Fig. 4. The unique test patterns and the corresponding codewords for s444 are listed in Table 2. The original (unencoded) test set T_D, which contains 1881 test patterns of 3 bits each, requires $1881 \times 3 = 5643$ bits of memory for storage. On the other hand, the encoded test set T_E has only 1.2121 bits per codeword, and hence requires only 2280 bits of memory. Therefore, Huffman encoding of T_D leads to 59.59% saving in storage, while both the order as well as the contiguity of test patterns are preserved.

Once the encoded test set T_E is determined by applying the Huffman encoding procedure to T_D, it is

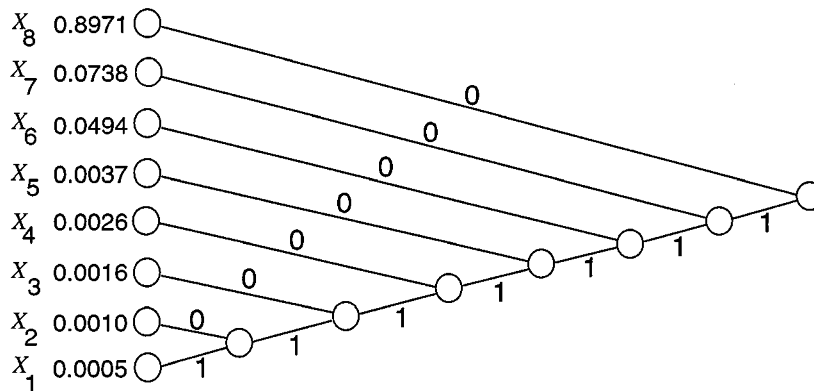

Fig. 4. Huffman tree for the test set of s444.

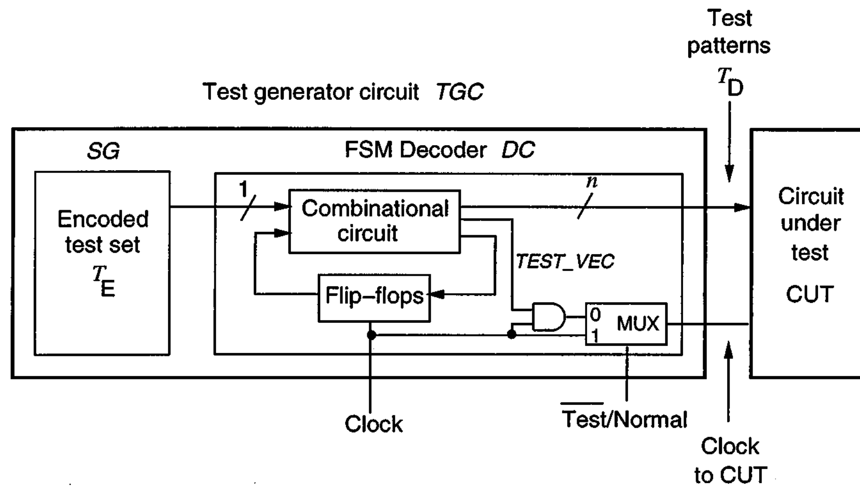

Fig. 5. Illustration of the proposed test application technique.

stored on-chip and read out one bit at a time during test application. The sequence generator *SG* of Fig. 1 is therefore a ROM that stores T_E. The test patterns in T_D can be obtained by decoding T_E using a simple finite-state machine (FSM) [20]. Table-lookup based methods that are typically used for software implementations of Huffman decoding are inefficient for on-chip, hardware-implemented decoding.

The decoder *DC* is therefore a sequential circuit, unlike for combinational and full-scan circuits where a combinational decoder can be used [12, 22]. Figure 5 outlines the proposed test application scheme. We exploit the prefix-free property of the Huffman code; thus patterns can be decoded immediately as the bits in the compressed data stream are encountered. We next describe the state diagram of the FSM decoder *DC* using the s444 example.

Figure 6 shows the state transition diagram of *DC*. The number of states is equal to the number of non-leaf nodes in the corresponding Huffman tree. For example, the Huffman tree of Fig. 4 has seven non-leaf nodes, hence the corresponding FSM of Fig. 6 has seven states—$S_1, S_2, \ldots, S_7$. The FSM receives a single-bit input from *SG*, and produces n-bit-wide test patterns, as well as a single-bit control output *TEST_VEC*. The control output is enabled only when a valid test pattern for the CUT is generated by the decoder—this happens whenever a transition is made to state S_1. The use of the *TEST_VEC* signal ensures that the test sequence T_D is preserved and no additional test patterns are applied to the CUT. Hence Huffman codes provide an efficient encoding of the test patterns

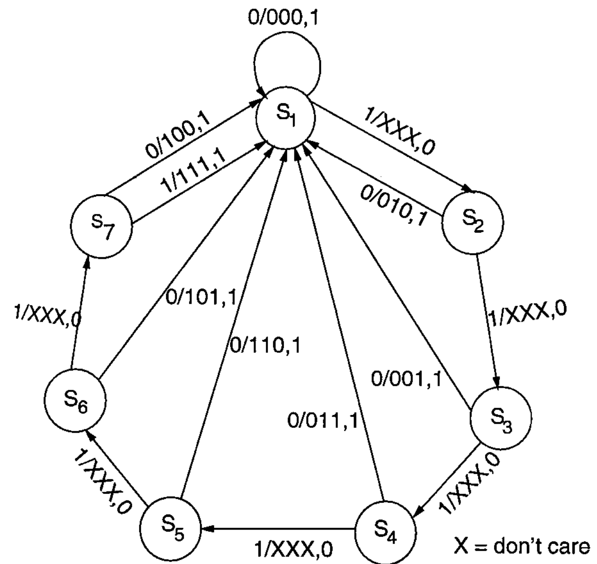

Fig. 6. State transition diagram for the FSM decoder of s444.

and a straightforward decoding procedure can then be used during test application. The trade-off involved is the increase in test application time t since the decoder examines only one bit of T_E in each clock cycle. Fortunately, the increase in t is directly related to the amount of test set compression achieved—the higher the degree of compression, the lesser is the impact on t.

Theorem 8. *The test application time t increases by a factor l_H, where l_H is the average length of a Huffman codeword.*

Proof: The state transition diagram of Fig. 6 shows that w_i clock cycles are required to apply a test pattern X_i which is mapped to a codeword of w_i bits. Hence the test application time (number of clock cycles) is given by $t = \sum_{i=1}^{|T_{\mathrm{D}}|} w_i$, where $|T_{\mathrm{D}}|$ is the total number of patterns in the test set T_{D}. The test application time therefore increases by a factor $\sum_{i=1}^{|T_{\mathrm{D}}|} w_i/|T_{\mathrm{D}}| = l_{\mathrm{H}}$, the average length of a codeword. ■

Experimental results on test set compression in Section 4 show that the average length of a Huffman codeword for typical test sets is less than 2. This implies that the increase in test application time rarely exceeds 100%. Since test patterns are applied in a BIST environment, this increase in testing time is acceptable, and it has little impact on testing cost or test quality.

Figure 7 shows the netlist of the decoder circuit for s444. This circuit was generated for a test set obtained using Gentest. The design is simplified considerably by the presence of a large number of don't-cares in the decoder specification, which a design automation tool can exploit for optimization.

The cost of the on-chip decoder *DC* can be reduced by noting that it is possible to share the same decoder on a chip among multiple CUTs. The encoding problem is now reformulated to encode the test sets of the CUTs together. We do this by combining these test sets to obtain a composite test set T_{D}' and applying the encoding procedure to T_{D}' to obtain an encoded test set T_{E}'. Figure 8 illustrates a single sequence generator SG' and pattern decoder DC' used to apply test sets to multiple CUTs that have the same number of primary inputs. Note that such sharing of the pattern decoder is also possible if the CUTs have an unequal number of primary inputs. The sharing is, however, more efficient if the difference in the number of primary inputs is small. The slight increase in the size of T_{E}' (compared to T_{E}) is offset by the hardware saving obtained by decoder sharing. We next present upper and lower bounds on the Huffman codeword length for two test sets that are encoded jointly.

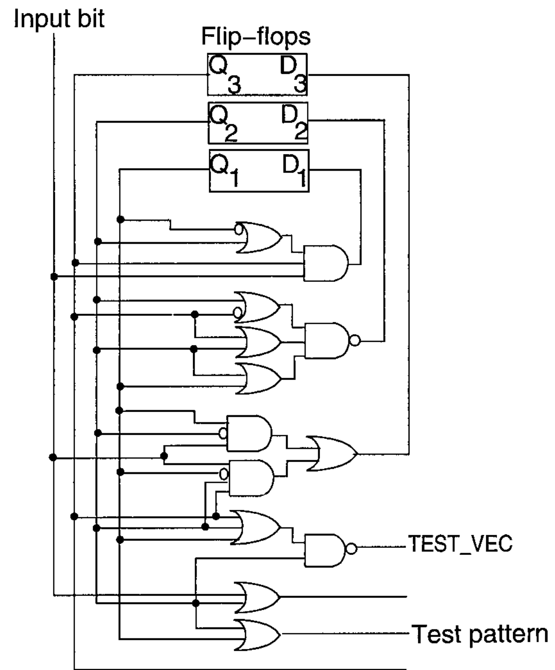

Fig. 7. Gate-level netlist of the FSM decoder for s444.

Theorem 9. *Let T_{D1} and T_{D2} be test sets for two CUTs with the same number of primary inputs and let T_D' be obtained by combining T_{D1} and T_{D2}. Let m_1, N_1, l_1, and m_2, N_2, l_2 be the number of unique patterns, total number of patterns, and average Huffman codeword length for T_{D1} and T_{D2}, respectively. Let l' be the average Huffman codeword length for T_D' and let m' and N' be defined as: $m' = \max\{m_1, m_2\}$ and $N' = N_1 + N_2$. In addition, let $p_i(q_i)$, $1 \le i \le m'$, be the probability of occurrence of the ith unique pattern*

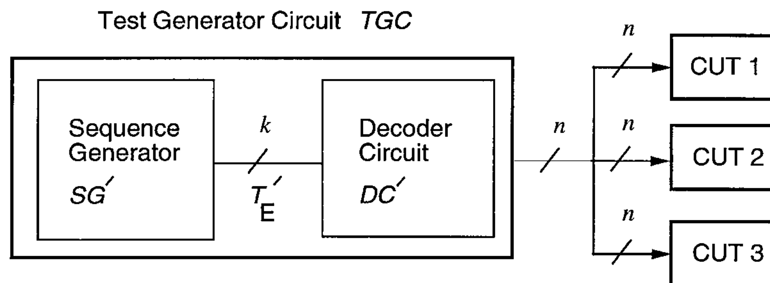

Fig. 8. A BIST sequence generator and decoder circuit used to test multiple CUTs.

in $T_{D1}(T_{D2})$. Then

$$\frac{N_1 l_1 + N_2 l_2}{N'} - \log_2 \beta_{\min} - 1 \le l' \le \frac{N_1 l_1 + N_2 l_2}{N'} - \log_2 \alpha_{\max} + 1,$$

where (i) $\alpha_{\max}$ is the largest value of α such that $N_1 p_i + N_2 q_i \ge N'\alpha p_i$ and $N_1 p_i + N_2 q_i \ge N'\alpha q_i$, and (ii) $\beta_{\min}$ is the smallest value of β such that $N_1 p_i + N_2 q_i \le N'\beta p_i$ and $N_1 p_i + N_2 q_i \le N'\beta q_i$, $1 \le i \le m'$.

Proof:hskip10ptWe use the fact that $H(T_{D1}) \le l_1 \le H(T_{D1})$ $+ 1$ and $H(T_{D2}) \le l_2 \le H(T_{D2}) + 1$, where $H(T_{D1})$ and $H(T_{D2})$ are the entropies of T_{D1} and T_{D2}, respectively. The probability of occurrence of the ith unique pattern in T'_D is $(N_1 p_i + N_2 q_i)/N'$. The entropy of T'_D is therefore given by

$$H(T'_D) = -\sum_{i=1}^{m'} \frac{N_1 p_i + N_2 q_i}{N'} \log_2\left(\frac{N_1 p_i + N_2 q_i}{N'}\right)$$

$$= \frac{N_1}{N'} \sum_{i=1}^{m'} p_i \log_2\left(\frac{N'}{N_1 p_i + N_2 q_i}\right) + \frac{N_2}{N'} \sum_{i=1}^{m'} q_i \log_2\left(\frac{N'}{N_1 p_i + N_2 q_i}\right)$$

It follows from the theorem statement that

$$\alpha_{\max} = \min_i\left\{\min\left\{\frac{N_1 + N_2(q_i/p_i)}{N'}, \frac{N_2 + N_1(p_i/q_i)}{N'}\right\}\right\}.$$

Therefore,

$$H(T'_D) \le \frac{N_1}{N'} \sum_{i=1}^{m'} p_i \log_2\left(\frac{1}{\alpha_{\max} p_i}\right) + \frac{N_2}{N'} \sum_{i=1}^{m'} q_i \log_2\left(\frac{1}{\alpha_{\max} q_i}\right)$$

$$\le \frac{N_1}{N'}(l_1 - \log_2 \alpha_{\max}) + \frac{N_2}{N'}(l_2 - \log_2 \alpha_{\max})$$

$$= \frac{N_1 l_1 + N_2 l_2}{N'} - \log_2 \alpha_{\max}$$

Now, $l' \le H(T'_D) + 1$. Therefore, $l' \le [(N_1 l_1 + N_2 l_2)/N'] - \log_2 \alpha_{\max} + 1$.

Next we prove the lower bound. Once again from the theorem statement,

$$\beta_{\min} = \max_i\left\{\max\left\{\frac{N_1 + N_2(q_i/p_i)}{N'}, \frac{N_2 + N_1(p_i/q_i)}{N'}\right\}\right\}.$$

Note that the lower bound is meaningful only if $p_i \ne 0$, $1 \le i \le m'$, and $q_i \ne 0$, $1 \le i \le m'$, which requires that T_{D1} and T_{D2} have the same set of unique patterns and therefore $m' = m_1 = m_2$.

Therefore,

$$H(T'_D) \ge \frac{N_1 H(T_{D1}) + N_2 H(T_{D2})}{N'} - \log_2 \beta_{\min}.$$

Now, $H(T_{D1}) \ge l_1 - 1$ and $H(T_{D2}) \ge l_2 - 1$. Therefore, $l' \ge H(T'_D) \ge [(N_1 l_1 + N_2 l_2)/N'] - \log_2 \beta_{\min} - 1$. ■

A tighter lower bound on l' is given by the following corollary to Theorem 9.

Corollary 2. *Let $l_1 = H(T_{D1}) + \delta_1$ and $l_2 = H(T_{D2})$ $+ \delta_2$, and let $\beta_{\min}$ be defined as in Theorem 9, where $0 \le \delta_1, \delta_2 \le 1$. Then $l' \ge [(N_1 l_1 + N_2 l_2)/N'] - [(N_1 \delta_1$ $+ N_2 \delta_2)/N'] - \log_2 \beta_{\min}$.*

As a special case, if $N_1 = N_2$ then $\frac{1}{2}(l_1 + l_2) - \log_2 \beta_{\min} - 1 \le l' \le \frac{1}{2}(l_1 + l_2) - \log_2 \alpha_{\max} + 1$.

For example, let T_{D1} and T_{D2} be test sets for two different CUTs with five primary inputs each. Suppose they contain the unique patterns shown in Fig. 9, with $N_1 = N_2$. The probabilities of occurrence of patterns in T_{D1} and T_{D2} satisfy (2), therefore $l_1 = \sum_{i=1}^{4} i p_i + 4p_5 = 1.25$. Similarly, $l_2 = 1.31$. From Theorem 9, $\alpha_{\max} = 0.58$, and $\beta_{\min} = 3.5$. Since $N_1 = N_2$, the bounds on l' are given by $\frac{1}{2}(l_1 + l_2) - \log_2 \beta_{\min} - 1 \le l' \le \frac{1}{2}(l_1 + l_2) - \log_2 \alpha_{\max} + 1$. Therefore, $1.03 \le$

Unique pattern	Probability of occurrence		
	T_{D1}	T_{D2}	T'_D
10110	0.80	0.75	0.775
00111	0.10	0.20	0.15
10100	0.05	0.04	0.045
00000	0.03	0.005	0.0175
11111	0.02	0.005	0.0125

Fig. 9. Unique patterns and their probabilities of occurrence for the example illustrating Theorem 9.

$l' \leq 3.55$. Now, the patterns in T'_D also satisfy (2), and therefore $l' = \sum_{i=1}^{4} i p'_i + 4p'_5 = 1.33$, which clearly lies between the calculated bounds, where p'_i is the probability of occurrence of pattern X_i in T'_D.

Experimental results on test set encoding and decoder overhead in Section 4 show that it is indeed possible to achieve high levels of compression while reducing decoder overhead significantly if test sets for two different CUTs with the same number of primary inputs are jointly encoded and a single decoder DC' is shared among them.

Comma Coding

We next describe test set compression and test application using Comma encoding. Once again, we illustrate the encoding and decoding scheme using the s444 example.

The unique patterns in the test set are first identified, and sorted in decreasing order of probability of occurrence. Codewords are then generated for the patterns according to the Comma code construction procedure described in Section 2. Comma codewords generated for the unique patterns in the s444 test set are listed in Table 2. The probabilities of occurrence of test patterns, shown in Table 2 clearly satisfy (2) in Theorem 3, and therefore the encoding is near-optimal. The Comma encoded test set has 1.2126 bits per codeword, and requires 2281 bits for storage, an increase of only one bit from the optimally (Huffman) encoded test set described in Section 3. Hence the reduction in test set compression arising from the use of Comma codes instead of Huffman codes for this example is only 0.02%.

The slight decrease in test set compression due to the use of the Comma code is offset by the reduced complexity of the pattern decoder DC. Figure 10 illustrates the pattern decoder for the s444 circuit test set. The decoder is constructed using a binary counter and combinational logic that maps the counter states to the test patterns. The test application scheme is the same as that in Fig. 5 for the Huffman decoder.

The inverted input bit is used to generate the *TEST_VEC* signal which ensures that the CUT is clocked only when a 0 is received. *TEST_VEC* is also gated with the clock to the CUT and used to reset the the counter on the falling edge of the clock. Bits with value 1 received from *SG* therefore result in the flip-flops of the counter being clocked to the next state, while 0s (the terminating "commas" present at the end of each code word) reset the flip-flops to the initial (000)

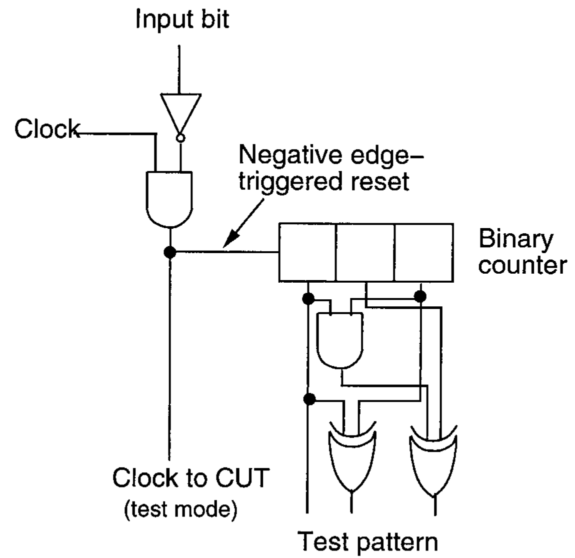

Fig. 10. The Comma pattern decoder for the s444 test set.

state after half a clock cyle. The test pattern can thus be latched by the CUT before the counter is reset. Comma decoders are simpler to implement than Huffman decoders, and binary counters already present for normal operation can be used to reduce overhead. As in the case of Huffman coding (Theorem 8), the increase in testing time due to Comma coding equals the average length of a codeword.

Run-Length Encoding

Finally, we describe run-length encoding of the statistically encoded test sequence T_E to achieve further compression. We exploit the fact that sequences of identical test patterns (runs) are common in test sets for sequential circuits having a high ratio of flip-flops to primary inputs. For example in the test set for s444, runs of the pattern 000 occur with lengths of up to 70. Huffman and Comma encoding exploit the large number of repetitions of patterns in the test sequence without directly making use of the fact that there are many contiguous, identical patterns. Run-length encoding exploits this property of the test sequences—it therefore complements statistical encoding. Huffman and comma encoding transform the sequence of test patterns to a compressed serial bit stream, and in the s444 test set, each occurrence of the test pattern 000 is replaced by a 0 (Table 2). Therefore, long runs of 0s are present in the statistically compressed bit stream,

Table 3. Distribution of runs in the Huffman encoded test set for the s444 circuit.

Run-length	No. of runs		Run-length	No. of runs	
	0s	1s		0s	1s
1	90	114	5	8	0
2	52	104	6	10	0
3	33	10	7	37	0
4	11	18	8	145	0

which can be further compressed using run-length coding.

Run-length coding is a data compression technique that replaces a sequence of identical symbols with a copy of the repeating symbol and the length of the sequence. For example, a run of 5 0s (00000) can be encoded as (0,5) or (0,101). Run-length encoding has been used recently to reduce the time to download test sets to ATE across a network [23, 24]. We improve upon the basic run-length encoding scheme by considering only those runs that have a substantial probability of occurrence in the statistically encoded bit stream. A unique symbol representing a run of a particular length (and the corresponding bit) is then stored. The value of the repeating bit is generated from the bits representing the length of the run during decoding. We therefore obviate the need to store a copy of the repeating bit.

We describe our run-length encoding process using the s444 example. An analysis of its Huffman encoded test set yields the distribution of runs shown in Table 3. Encoding all 16 runs would obviously be expensive (4 bits would be required for each run) since very few instances of (0,4), (0,5) and (1,3), and no instances of (1,5), (1,6), (1,7) or (1,8) exist. We therefore use combinations of 3 bits $(000, 001, \ldots, 111)$ to encode the 8 most frequently occurring runs–(0,1), (0,2), (0,3), (0,7), (0,8), (1,1), (1,2) and (1,4). The less-frequently occurring runs (0,4), (0,5), (0,6) and (1,3) are divided into smaller consecutive runs for encoding. For example (0,5) is encoded as (0,3) followed by (0,2). Figure 11 illustrates run-length encoding applied to a portion of the Huffman encoded s444 test set. The encoded runs are stored in a ROM and output to a run-length decoder. The run-length decoder provides a single bit in every clock cycle to the Huffman (or Comma) decoder for test application.

The run-length decoder consists of a binary down counter, and a small amount of combinational logic. Figure 12 illustrates the run-length decoder for the s444 test set. The bits used to encode a run (e.g., 011 for (0,7)—Fig. 11(a)) are first mapped to the run-length (110 for 7 bits). The run-length is loaded into the counter which outputs the first bit of the run. The counter then counts down from the preset value 110 to 000, sending a bit (0, for this example) to the Huffman decoder in every clock cycle. When the counter reaches 000, the NOR gate output becomes 1, enabling the ROM to output the bits representing the next run. Since one bit is received by the Huffman decoder in every clock cycle, run-length decoding does not add to testing time.

4. Experimental Results

In this section, we present experimental results on test set encoding for several ISCAS 89 benchmark circuits to demonstrate the saving in on-chip storage achieved using Huffman, Comma and run-length encoding. We

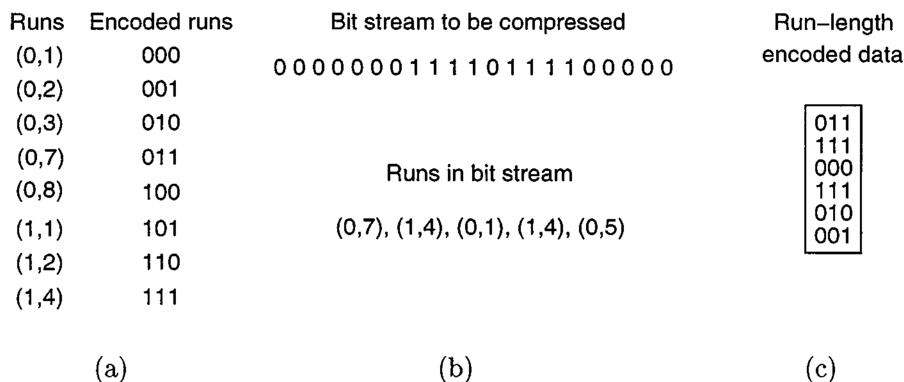

Fig. 11. Run-length encoding applied to a portion of the Huffman encoded s444 test set: (a) 3-bit encoding for 8 types of runs, (b) bit stream to be encoded, and (c) run-length encoded data.

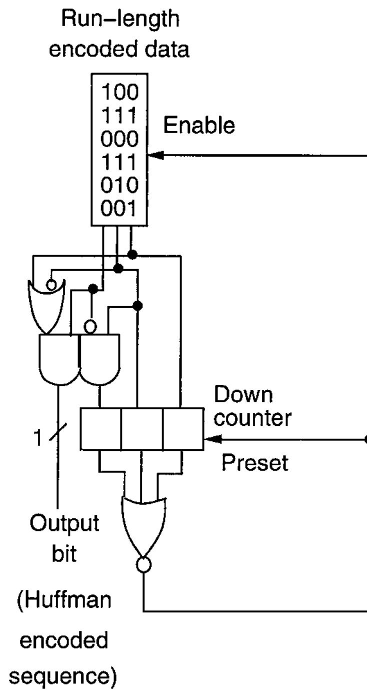

Fig. 12. Run-length decoder for the s444 test set.

consider circuits in which the number of flip-flops f is considerably greater than the number of primary inputs n; we denote the ratio f/n by γ. Table 4 lists the values of γ for the ISCAS 89 circuits, with circuits having a high value of γ shown in bold. Such circuits are especially hard to test because of the relatively large number of internal states and few primary inputs. From Table 4

Table 4. The ratio of the number of flip-flops to the number of primary inputs γ, and $|T_D|$, the length of the HITEC test sequences for the ISCAS 89 circuits.

CUT	γ	$\|T_D\|$	CUT	γ	$\|T_D\|$
s1196	1.28	435	**s400**	**7**	**2208**
s1238	1.28	475	s420	0.84	166
s1423	4.35	150	**s444**	**7**	**2240**
s298	**4.66**	**322**	**s526**	**7**	**2250**
s344	1.66	127	s641	0.54	209
s349	1.66	134	s713	0.54	173
s35932	**49.37**	**496**	s820	0.625	1115
s382	**7**	**2074**	s838	0.94	26
s386	0.85	286	s953	1.81	13

we see that these circuits typically require longer sequences of test patterns. On the other hand, they are excellent candidates for our encoding approach.

Several other ISCAS 89 benchmark circuits do not have a high value of γ, and are therefore more suitable for scan-based testing, than for the proposed approach of encoding non-scan test sets. We do not present results for these circuits, however, statistical encoding of full-scan test sets for these circuits, on the lines of the proposed approach, has recently been shown to be effective in reducing the amount of memory required for test storage [25].

We performed experiments on test sets for single-stuck line (SSL) faults obtained from the Gentest ATPG program, as well as the HITEC, GATEST, and STRATEGATE test sets from the University of Illinois [26]. We measured the fault coverage of these test sets using the PROOFS fault simulator [27] and ensured that the coverage is comparable to the best-known fault coverage for these circuits. We next present results on the compression achieved using Huffman and Comma coding for all four test sets. Table 5 compares the number of bits required to store the encoded test set T_E with that required to store the corresponding unencoded test set T_D. The number of bits required by our scheme is moderate, substantially less than that required to store unencoded test sets, and reduces significantly when the same test set can be shared among multiple CUTs of the same type included on a chip, as in core-based DSP circuits [14]. The saving in *SG* memory presented in Table 5 is substantial, and in most cases, the difference in compression due to the use of Comma coding instead of Huffman coding is very small. In Table 6, we show that further compression is achieved by applying run-length coding to T_E. We present results on run-length encoding for the s382 and s444 circuits using the Gentest test set.

The test application time required is considerably less than that required for pseudorandom testing, even though the number of clock cycles $\mathcal{C}$ is greater than the number of patterns in T_D ($\mathcal{C} = l_H|T_D|$ for Huffman coding and $\mathcal{C} = l_C|T_D|$ for Comma coding). Table 7 compares the number of test patterns applied, the number of clock cycles required, and the fault coverage obtained for our method, with the corresponding figures reported recently for two pseudorandom testing schemes [5, 6]. The test application time required by our method is much less than for the pseudorandom testing method of [5]. We also achieve higher fault coverage for all circuits.

Table 5. Experimental results on test set compression for ISCAS 89 circuits with a high value of γ.

ISCAS 89 circuit	n	m	$\|T_D\|$	T_{bits}	Average codeword length l_H	l_C	H_{bits}	C_{bits}	Percentage compression HE	CE
Gentest										
s298	3	7	162	486	1.7901	1.7963	290	291	40.32	40.12
s382	3	8	2463	7389	1.2026	1.2030	2962	2963	59.91	59.89
s400	3	8	1282	3846	1.4180	1.4188	1818	1819	52.73	52.70
s444	3	8	1881	5643	1.2121	1.2126	2280	2281	59.59	59.57
s526	3	8	754	2262	1.9031	1.9058	1435	1437	36.60	36.47
s35932	35	86	86	3010	6.65	—[a]	572	—	81.00	—
HITEC										
s298	3	8	322	966	2.04	2.05	657	660	31.88	31.36
s382	3	8	2074	6222	1.46	1.47	3028	3049	51.35	51.33
s400	3	8	2208	6624	1.46	1.47	3224	3246	51.43	51.43
s444	3	8	2240	6720	1.47	1.47	3293	3297	45.19	45.19
s526	3	8	2250	6750	1.89	1.90	4253	4275	41.38	41.38
s35932	35	496	496	17360	8.97	—[a]	4449	—	74.38	—
STRATEGATE										
s298	3	8	194	582	2.46	2.72	447	528	18.04	9.10
s382	3	8	1486	4458	1.92	1.92	2853	2854	36.00	35.80
s400	3	8	2424	7272	1.82	1.83	4412	4436	39.36	39.06
s444	3	8	1945	5835	1.64	1.64	3190	3191	45.19	45.19
s526	3	8	2642	7924	1.76	1.76	4450	4652	41.38	41.38
s35932	35	257	257	8995	7.79	—[a]	2002	—	77.73	—
GATEST										
s298	3	7	147	441	2.34	2.38	344	350	22.00	20.40
s382	3	7	331	993	2.23	2.40	771	794	25.58	19.83
s400	3	6	324	972	2.25	2.47	729	800	25.00	17.48
s444	3	6	254	762	2.22	2.34	564	594	25.85	21.78
s526	3	5	371	1113	2.21	2.38	820	883	26.24	20.66
s35932	35	256	256	8960	8.00	—[a]	2048	—	77.14	—

n: No. of primary inputs; m: No. of unique test patterns; T_{bits}: Total no. of bits in T_D; H_{bits}: No. of bits in T_E after Huffman encoding; C_{bits}: No. of bits in T_E after comma encoding;
[a]Comma coding is not applicable for the test set of s35932, because the probabilities of occurrence of the test patterns do not satisfy (2) given in Theorem 4.

Table 6. Percentage compression achieved by run-length coding after applying Huffman and Comma encoding to T_D.

ISCAS 89 circuit	T_{bits}	Number of bits in T_E: H_{bits}	C_{bits}	HR_{bits}	CR_{bits}	Percentage compression HE	CE	HRE	CRE
s382	7389	2962	2963	2268	2421	59.91	59.89	69.31	67.24
s444	5643	2280	2281	1953	2013	59.59	59.57	65.39	64.33

HR_{bits}: No. of bits in the encoded test set after Huffman and run-length encoding; CR_{bits}: No. of bits in the encoded test set after Comma and run-length encoding.

Table 7. Number of clock cycles $\mathcal{C}$ required and fault coverage obtained using pseudorandom testing compared with the corresponding figures using precomputed deterministic test sets.

ISCAS 89 circuit	Number of patterns $\|T_D\|$			Number of clock cycles $\mathcal{C}$			Fault coverage (%)		
	[5][a]	[6][a]	Det[a]	[5]	[6]	Det	[5]	[6]	Det[b]
s298	—[c]	899	194	—	899	477	—	86.04	86.04
s382	34,807	4,694	1,486	100,000	4,694	2,853	86.00	89.47	91.22
s444	34,807	4,603	1,881	100,000	4,603	2,280	80.40	87.76	89.45
s526	—	19,864	2,642	—	19,864	4,650	—	79.28	81.80
s35932	41,667	—	257	100,000	—	2,002	87.00	—	89.78

[a][5, 6]: Recently proposed pseudorandom BIST methods; Det: Deterministic testing using precomputed test sets.
[b]The best fault coverage achieved by precomputed deterministic testing.
[c]Results for these circuits were not reported in [5, 6].

Table 8. Literal counts of the Huffman and Comma decoders for the four test sets.

ISCAS 89 circuit	Decoder cost in literals							
	Huffman decoders				Comma decoders			
	Gen	HIT	GAT	STRAT	Gen	HIT	GAT	STRAT
s298	46	44	42	41	24	32	27	28
s382	42	34	34	43	30	27	33	29
s400	44	33	33	37	26	27	27	27
s444	50	46	30	47	27	29	27	25
s526	43	47	18	43	29	29	28	27
s35932	2220	4002	4002	4150	—	—	—	—

Gen: Gentest; HIT: HITEC; GAT: GATEST; STRAT: STRATEGATE.

We next present experimental results on the Huffman and Comma decoder implementations. We designed and synthesized the FSM decoders using the Epoch CAD tool from Cascade Design Automation [28]. The low to moderate decoder costs in Table 8 show that the decoding algorithm can be easily implemented as a BIST scheme. Note that the largest benchmark circuit s35932 requires an extremely small overhead (synthesized ROM area is 0.53% of CUT area, and decoder area is 6.18% of CUT area) to store the encoded test set and decoder, thus demonstrating that the proposed approach is scalable and it is feasible to incorporate the encoded test set on-chip for larger circuits.

Note that, while Huffman and Comma encoding reduce the number of bits to be stored, the serialization of the ROM may increase the hardware requirements for ROM address generation. In a conventional fixed-length encoding scheme, the size of the counter required for ROM address generation is $\lceil \log_2 |T_D| \rceil$, while an encoded ROM requires a $\lceil \log_2(|T_D| l) \rceil$-bit counter for address generation, where l is the average codeword length. However, since l is small, this logarithmic increase in counter size is also small, e.g., the size of the counter does not change for s444, while it increases from 7 to 10 for s35932. The hardware overhead figures in Table 8 do not include this small increase in counter size.

It may be argued that a special-purpose, minimal-state FSM may be used to produce a precomputed sequence. However, we have seen that the overhead of such FSMs is prohibitive, especially for long test sequences. In addition, such a special-purpose FSM would be specific to a single CUT; on the other hand, the decoder *DC* for the proposed scheme is shared among multiple CUTs, thereby reducing overall TGC overhead.

Table 9 compares the overhead of the proposed deterministic BIST scheme with the overhead of a pseudorandom BIST method [6] for several circuits. The overhead for the pseudorandom method was obtained

Table 9. Literal counts for the proposed technique compared with pseudorandom testing.

ISCAS 89 circuit	Deterministic TGC cost: Decoder cost	Deterministic TGC cost: Total cost	Pseudorandom TGC cost [6]	Number of test points [6]
s298	24	53	47	9
s382	27	111	57	8
s444	25	85	57	8
s526	18	114	61	9

by mapping the gate count figures from [6] to the literal counts of standard cells in the Epoch library. While the deterministic TGC requires greater area than the pseudorandom TGCs, the difference is quite small, and thus may be acceptable if higher fault coverage and shorter test times are required. Note also that the pseudorandom method requires the addition of a large number of observability test points. These require a gate-level model of the CUT, as well as additional primary outputs and routing. Moreover, they may also increase the size of the response monitor at the CUT outputs. The proposed TGCs require no circuit modification, thus making them more applicable to testing core-based designs using precomputed test sequences.

Finally, we present experimental results for test set compression and decoder overhead, using a single decoder to test several CUTs on a chip. Table 10 shows that the levels of compression obtained for combined test sets are comparable to those obtained for the individual test sets. In fact, in several cases the overall compression is higher than that obtained for one of the individual test sets. The percentage area overhead required for the decoder reduces significantly, because a single decoder can now be shared among several CUTs. Note that in the case of the Comma decoders, a major part of the overhead is contributed by the binary counters. For example, in the Comma decoder for the pair {s444, s526}, the binary counter represents 3.14% overhead, while the combinational logic represents only 0.18% overhead. If the counter is also used for normal operation of the system, then the BIST overhead will reduce further. The test application technique is therefore clearly scalable with increasing circuit complexity. The decoder overhead also tends to decrease with an increase in γ. This clearly demonstrates that the proposed test technique is well suited to circuits for which γ is high.

5. Conclusion

We have presented a novel technique for deterministic built-in pattern generation for sequential circuits. This approach is especially suited to sequential circuits that have a large number of flip-flops and relatively few

Table 10. Percentage compression for test sets encoded jointly.

Circuit pair	Percentage Huffman compression				Percentage Comma compression			
	Gen	HIT	GAT	STRAT	Gen	HIT	GAT	STRAT
{s298, s400}	49.24	47.87	17.41	35.43	49.21	47.76	11.04	33.79
{s382, s444}	58.65	49.25	12.48	36.00	58.60	49.16	2.56	33.59
{s444, s526}	51.85	42.42	25.39	42.96	51.81	42.33	18.08	42.72

Table 11. Decoder cost in literals and percentage decoder overhead for a single decoder shared among several CUTs.

Circuit pair		Decoder cost in literals				Percentage decoder overhead			
		Gen	HIT	GAT	STRAT	Gen	HIT	GAT	STRAT
{s298, s400}	Huffman	44	46	33	45	8.15	8.45	6.19	8.44
{s382, s444}		48	52	44	44	6.72	7.29	6.18	6.21
{s444, s526}		42	36	33	39	5.11	4.33	4.03	4.77
{s298, s400}	Comma	28	32	25	26	5.22	5.87	4.71	4.83
{s382, s444}		37	34	29	36	5.20	4.83	4.02	5.01
{s444, s526}		33	32	27	31	3.98	3.87	3.32	3.82

primary inputs, and for circuits such as embedded cores, for which gate-level models are not available. We have shown that statistical encoding of precomputed test sequences leads to effective compression, thereby allowing on-chip storage of encoded test sequences. We have also shown that the average codeword length for the non-optimal Comma code is nearly equal to the average codeword length for the optimal Huffman code if the test sequence satisfies certain proeprties. These are generally satisfied by test sequences for typical sequential circuits, therefore Comma coding is near-optimal in practice.

Our results show that Huffman and Comma encoding of test sequences, followed by run-length encoding, can greatly reduce the memory required for test storage. The small increase in testing time is offset by the high degree of test set compression achieved. Furthermore, testing time is considerably less than that for pseudorandom methods. We have developed efficient low-overhead pattern decoding methods for applying the test patterns to the CUT. We have also shown that the overhead can be reduced further by using a single decoder to test multiple CUTs on the same chip. The proposed technique thus offers a promising BIST methodology for complex non-scan and partial-scan circuits for which precomputed test sets are readily available.

References

1. B.T. Murray and J.P. Hayes, "Testing ICs: Getting to the Core of the Problem," *IEEE Computer*, Vol. 29, pp. 32–38, Nov. 1996.
2. D.J. Holden, "Focus Report: Design for Test Tools," *Integrated Systems Design*, pp. 36–52, Sep. 1997.
3. F. Muradali, T. Nishida, and T. Shimizu, "A Structure and Technique for Pseudorandom-Based Testing of Sequential Circuits," *Journal of Electronic Testing: Theory and Applications*, Vol. 6, pp. 107–115, Feb. 1995.
4. F. Muradali and J. Rajski, "A Self-Driven Test Structure for Pseudorandom Testing of Non-Scan Sequential Circuits," *Proc. IEEE VLSI Test Symposium*, 1996, pp. 17–25.
5. L. Nachman, K.K. Saluja, S. Upadhyaya, and R. Reuse, "A Novel Approach to Random Pattern Testing of Sequential Circuits," *IEEE Transactions on Computers*, Vol. 47, pp. 129–134, Jan. 1998.
6. I. Pomeranz and S.M. Reddy, "Built-In Test Generation for Synchronous Sequential Circuits," *Proc. Int. Conf. on Computer Aided Design*, 1997, pp. 421–426.
7. W.T. Cheng and T. Chakraborty, "Gentest—An Automatic Test-Generation System for Sequential Circuits," *IEEE Computer*, Vol. 22, pp. 43–49, April 1989.
8. M.C. Hansen and J.P. Hayes, "High-Level Test Generation Using Symbolic Scheduling," *Proc. Int. Test Conf.*, 1995, pp. 586–595.
9. M.S. Hsiao, E.M. Rudnick, and J.H. Patel, "Alternating Strategies for Sequential Circuit ATPG," *Proc. European Design and Test Conf.*, 1996, pp. 368–374.
10. T.M. Niermann and J.H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. European Design Automation Conf.*, 1991, pp. 214–218.
11. D.G. Saab, Y.G. Saab, and J.A. Abraham, "Automatic Test Vector Cultivation for Sequential VLSI Circuits Using Genetic Algorithms," *IEEE Trans. on Computer Aided Design*, Vol. 15, pp. 1278–1285, Oct. 1996.
12. K. Chakrabarty, B.T. Murray, J. Liu, and M. Zhu, "Test Width Compression for Built-in Self Testing," *Proc. Int. Test Conf.*, 1997, pp. 328–337.
13. F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. Int. Symp. on Circuits and Systems*, 1989, pp. 1929–1934.
14. M.S.B. Romdhane, V.K. Madisetti, and J.W. Hines, *Quick-Turnaround ASIC Design in VHDL: Core-Based Behavioral Synthesis*, Kluwer Academic Publishers, Boston, MA, 1996.
15. J.P. Hayes, *Computer Architecture and Organization*, 3rd ed., McGraw-Hill, New York, NY, 1998.
16. G. Held, *Data Compression Techniques and Applications: Hardware and Software Considerations*, John Wiley, Chichester, West Sussex, 1991.
17. M. Jakobssen, "Huffman Coding in Bit-Vector Compression," *Information Processing Letters*, Vol. 7, No. 6, pp. 304–307, Oct. 1978.
18. T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley, New York, NY, 1991.
19. M. Mansuripur, *Introduction to Information Theory*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.
20. V. Iyengar and K. Chakrabarty, "An Efficient Finite-State Machine Implementation of Huffman Decoders," *Information Processing Letters*, Vol. 64, No. 6, pp. 271–275, Jan. 1998.
21. D.H. Greene and D.E. Knuth, *Mathematics for the Analysis of Algorithms*, Birkhauser, Boston, MA, 1981.
22. K. Chakrabarty and B.T. Murray, "Design of Built-in Test Generator Circuits Using Width Compression," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, pp. 1044–1051, Oct. 1998.
23. T. Yamaguchi, M. Tilgner, M. Ishida, and D.S. Ha, "An Efficient Method for Compressing Test Data to Reduce the Test Data Download Time," *Proc. Int. Test Conf.*, 1997, pp. 79–88.
24. M. Ishida, D.S. Ha, and T. Yamaguchi, "COMPACT: A Hybrid Method for Compressing Test Data," *Proc. IEEE VLSI Test Symposium*, 1998, pp. 62–69.
25. A. Jas, J. Ghosh-Dastidar, and N.A. Touba, "Scan Vector Compression/Decompression Using Statistical Coding," *Proc. IEEE VLSI Test Symposium*, 1999, pp. 114–120.
26. *IGATE Genetic Framework for Test & Diagnosis*, The University of Illinois at Urbana-Champaign. WWW site *http://www.crhc.uiuc.edu/IGATE/*
27. W.T. Cheng and J.H. Patel, "PROOFS: A Super Fast Fault Simulator for Sequential Circuits," *Proc. IEEE European Conf. on Design Automation*, March 1990, pp. 475–479.
28. *Epoch Finesse User and Reference Manual*, Cascade Design Automation, Bellevue, WA, 1993.

Vikram Iyengar received the B.E. degree from the Birla Institute of Technology, India in 1996, in Electrical and Electronics Engineering,

and the M.S. degree from Boston University in 1998, in Computer Engineering.

While at Boston University, he was a research assistant at the Reliable Computing Laboratory of the Department of Electrical and Computer Engineering. Mr. Iyengar is currently a research assistant at the Center for Reliable and High-Performance Computing at the University of Illinois at Urbana-Champaign. His research interests are in computer-aided design of VLSI circuits and systems, design verification and built-in self test.

Mr. Iyengar is a member of IEEE.

Krishnendu Chakrabarty 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 now Assistant Professor of Electrical and Computer Engineering at Duke University. During 1995–1998, he was Assistant Professor of Electrical and Computer Engineering at Boston University. Dr. Chakrabarty is a recipient of the National Science Foundation Early Faculty (CAREER) award. His current research projects are in embedded core testing, BIST, and real-time operating systems, and he has published over 30 papers on these topics in archival journals and referred conference proceedings. He is a member of IEEE and Sigma Xi, and serves as Vice Chair of Technical Activities in IEEE's Test Technology Technical Council (TTTC).

Brian T. Murray received the A.B. degree in Physics and Mathematics from Albion College in 1982. He received the M.S. degree in Electrical Engineering from Duke University in 1984, and the Ph.D. degree in Computer Science and Engineering from the University of Michigan in 1994. From 1984 to 1998, he was a member of the technical staff at General Motors Research and Development, where he led projects in testing, high-dependability distributed embedded systems and computer architecture. He has also been an Adjunct Lecturer at the University of Michigan since 1995. He is currently Project Manager of Dependable Embedded Systems in the Advanced Development organization of Delphi Automotive Systems. His research interests include drive-by-wire, architectures for high-dependability distributed embedded systems, and methods and tools for system safety engineering. Dr. Murray is a member of the IEEE, the Association for Computing Machinery, Sigma Xi, and Phi Beta Kappa. He currently serves on the editorial board of the Journal of Electronic Testing: Theory and Applications.