Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-Directed Run-Length (FDR) Codes

Anshuman Chandra, *Student Member*, *IEEE*, and Krishnendu Chakrabarty, *Senior Member*, *IEEE*

Abstract—Test data compression and test resource partitioning (TRP) are necessary to reduce the volume of test data for system-on-a-chip designs. We present a new class of variable-to-variable-length compression codes that are designed using distributions of the runs of 0s in typical test sequences. We refer to these as frequency-directed run-length (FDR) codes. We present experimental results for ISCAS 89 benchmark circuits and two IBM production circuits to show that FDR codes are extremely effective for test data compression and TRP. We derive upper and lower bounds on the compression expected for some generic parameters of the test sequences. These bounds are especially tight when the number of runs is small, thereby showing that FDR codes are robust, i.e., they are insensitive to variations in the input data stream. In order to highlight the inherent superiority of FDR codes, we present a probabilistic analysis of data compression for a memoryless data source. Finally, we derive entropy bounds for the benchmark test sets and show that the compression obtained using FDR codes is close to the entropy bounds.

Index Terms—Automatic test equipment (ATE), decompression architecture, embedded core testing, precomputed test sets, test set encoding, system-on-a-chip test, variable-to-variable-length codes.

1 Introduction

Test data volume is a major problem encountered in the testing of system-on-a-chip (SOC) designs [1]. A typical SOC consists of several intellectual property (IP) blocks, each of which must be exercised by a large number of precomputed test patterns. The increasingly high volume of SOC test data is not only exceeding the memory and I/O channel capacity of commercial automatic test equipment (ATEs), it is also leading to excessively high testing times.

The testing time of an SOC directly impacts test cost. It is determined by several factors, including the test data volume, the time required to transfer test data to the cores, the rate at which the test patterns are transferred (measured by the test data bandwidth and the ATE channel capacity), and the maximum scan chain length. For a given ATE channel capacity and test data bandwidth, reduction in testing time can be achieved by reducing the test data volume and by redesigning the scan chains. While test data volume reduction techniques can be applied to both soft and hard cores, scan chains cannot be modified in hard (IP) cores. New techniques are therefore needed to reduce the test data volume, decrease testing time, and overcome ATE memory limitations for SOCs containing IP cores.

Built-in self-test (BIST) has emerged as an alternative to ATE-based external testing [2]. BIST offers a number of key advantages. It allows precomputed test sets to be embedded in the test sequences generated by on-chip hardware, supports test reuse and at-speed testing, and protects intellectual property. While BIST is now extensively used for memory testing, it is not as common for logic testing. This is particularly the case for nonscan and partial-scan designs in which test vectors cannot be reordered and application of pseudorandom vectors can lead to serious bus contention problems during test application. Moreover, BIST can be applied to SOC designs only if the IP cores in it are BIST-ready. Since most currently available IP cores are not BIST-ready, BIST insertion in SOCs containing these circuits is expensive and requires considerable redesign.

An alternative approach for reducing test data volume for SOCs is based on the use of data compression techniques such as statistical coding, run-length coding, and Golomb coding [3], [4], [5], [6], [7]. In this approach, the precomputed test set T_D provided by the core-vendor is compressed (encoded) to a much smaller test set T_E and stored in the ATE memory; see Fig. 1. An on-chip decoder is used for pattern decompression to generate T_D from T_E during pattern application. This is an example of test resource partitioning (TRP) in which ATE complexity is reduced by moving some of the test resources from the ATE to the SOC. It was shown in [5], [6], [7] that compressing a "difference vector" sequence T_{diff} determined from T_D results in smaller test sets and reduced testing time. Fig. 2 shows the test architecture based on T_{diff} and cyclical scan registers (CSRs).

While previous research has clearly demonstrated that data compression offers a practical solution to the problem of reducing test data volume via TRP, the compression codes used in prior work were derived from other application areas. For example, the statistical codes used in [3] and [4] are motivated by pattern repetitions in large text files. Similarly, the run-length and Golomb codes used in [5], [6], [7] are more effective for encoding large files

• *The authors are with the Department of Electrical and Computer Engineering, Duke University, 130 Hudson Hall, Box 90291, Durham, NC 27708. E-mail: {achandra, krish}@ee.duke.edu.*

Manuscript received 8 May 2002; revised 28 Feb. 2002; accepted 4 June 2002. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114113.

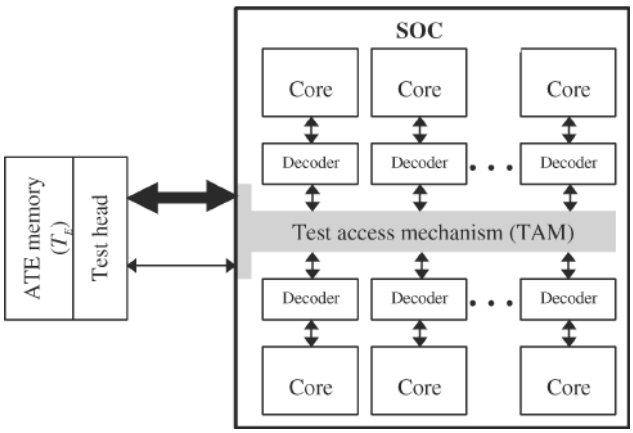

Fig. 1. A conceptual architecture for testing a system-on-chip by storing the encoded test data T_E in ATE memory and decoding it using on-chip decoders.

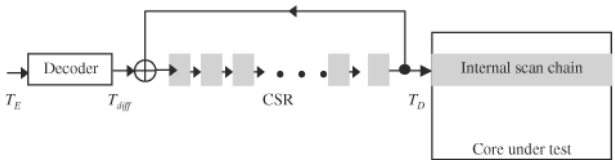

Fig. 2. Decompression architecture based on a cyclical scan register (CSR).

containing image data. None of these codes are tailored to exploit the specific properties of precomputed test sets for logic circuits. The Huffman code is known to be provably optimal under certain well-defined conditions for data compression [8] and it has been proposed for test data compression [3], [4], [9]; however, its decoding complexity for large block sizes makes it unsuitable for on-chip decompression in a TRP scheme. While an attempt was made in [6], [7] to customize the Golomb code by choosing an appropriate code parameter, the basic structure of the code was still independent of the test set. We can therefore expect even greater reduction in test data volume by crafting compression codes that are based on the generic properties of test sets.

In this paper, we present a class of variable-to-variable-length compression codes that are designed using the distributions of the runs of 0s in typical test sequences. In this way, the code can be tailored to our application domain, i.e., SOC test data compression. We refer to these as frequency-directed run-length (FDR) codes. For simplicity, we also refer to an instance of this class of codes as an FDR code. We show that the FDR code outperforms both Golomb codes and conventional run-length codes. We also show that the FDR code can be effectively applied to both the difference vector sequence T_{diff} and the precomputed test set T_D. The latter is especially attractive since it eliminates the need for a separate CSR for decompression. Additional contributions of this paper include a novel decompression architecture for FDR codes and an analytical characterization of the amount of data compression that can be expected using these codes. We also derive entropy bounds for the benchmark test sets and show that the compression obtained using FDR codes is quite close to the theoretical bounds.

A major advantage of test data compression lies in the fact that the patterns obtained after on-chip decompression can target a large number of nonmodeled faults. Test set compaction methods typically employed in automatic test pattern generation (ATPG) drastically reduce the number of patterns that detect any given fault from a fault model. Recent research has, however, shown that n-detection test sets, in which every fault is detected by at least n ($n > 1$) tests, are more effective in detecting physical defects [15], [16], [17]. When test data compression is applied to a set of test cubes containing t patterns, all the t patterns are applied to the circuit under test (CUT) at scan clock frequency after on-chip decompression. Thus, test data compression not only reduces tester memory requirements and decreases testing time, but it also increases the likelihood of detecting nonmodeled faults.

The proposed compression approach for reducing test data volume is especially suitable for SOCs containing IP cores since it does not require gate-level models for the embedded cores. Precomputed test sets can be directly encoded without any fault simulation or subsequent test generation. This is in contrast to other recent techniques, such as LFSR-based reseeding for BIST [18], bit-flipping BIST [19], bit-fixing BIST [20], and scan broadcast [21], which require structural models for fault simulation and test generation. For example, the mixed-mode BIST technique in [18] relies on fault simulation for identifying hard faults and test generation to determine test cubes for these faults. The scan broadcast technique in [21] also requires test generation.

The organization of the rest of this paper is as follows: In Section 2, we first motivate the new FDR code and describe its construction. In Section 3, we determine the best-case and the worst-case compression that can be achieved given some generic parameters of the precomputed test set. We also present a probabilistic analysis for a memoryless data source and compare FDR codes to Golomb codes, run-length codes, and entropy bounds. We then describe some extensions to the basic FDR code, the data compression procedure, and the decompression architecture in Section 4. Finally, in Section 5, we present experimental results for the six largest ISCAS 89 benchmark circuits as well as the scan vectors for two production circuits from IBM. We do not include results for the smaller benchmark circuits since they are small enough to be considered trivial. Results on Golomb coding for the smaller circuits can be found in [7]. We also derive fundamental entropy bounds and show that the FDR codes provide almost as much compression as the entropy bounds for the benchmark circuits.

2 FDR Codes

In this section, we describe FDR coding and compare it with conventional run-length coding and variable-to-fixed-length Golomb coding. An FDR code is a variable-to-variable-length code which maps variable-length runs of 0s to codewords of variable length. It corresponds to a special case of the exponential Golomb code with code parameter $k = 1$ [22].

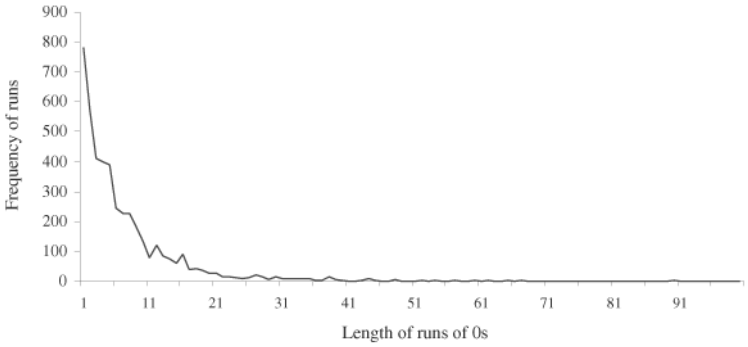

Fig. 3. Distribution of runs of 0s for the ISCAS benchmark circuit s9234.

An FDR code can be used to compress both the difference vector sequence T_{diff} and the test set T_D. Let $T_D = \{t_1, t_2, t_3, \ldots, t_n\}$ be the (ordered) precomputed test set. The ordering is determined using a heuristic procedure described later. T_{diff} is defined as follows:

$$T_{diff} = \{d_1, d_2, \ldots, d_n\} = \{t_1, t_1 \oplus t_2, t_2 \oplus t_3, \ldots, t_{n-1} \oplus t_n\},$$

where a bit-wise exclusive-or operation is carried out between patterns t_i and t_{i+1}. This assumes that the CSR starts in the all-0 state. (Other starting states can be considered similarly.) The successive test patterns in a test sequence often differ in only a small number of bits. Therefore, T_{diff} contains few 1s and it can be efficiently compressed using the FDR code. However, the test architecture requires additional CSR and an exclusive-or gate for pattern decompression. If the uncompacted test set T_D is used for compression, all the don't-care bits in T_D are mapped to 0s to obtain a fully specified test set before compression.

A run length l is defined as a stream of 0s terminating with a 1. Therefore, 000001 is a run length of five ($l = 5$) and a single 1 is a run length of zero ($l = 0$). We now present some important observations about the distribution of runs of 0s in typical test sets which motivate the need for an FDR code. We conducted a series of experiments for the large ISCAS benchmark circuits and IBM test data and studied the distribution of the runs of 0s in T_{diff} obtained from complete single stuck-at test sets for these circuits. Fig. 3 illustrates this distribution for the s9234 benchmark circuit. We found that the distributions of runs of 0s were similar for the test sets of the other circuits.

The key observations from Fig. 3 are as follows:

- The frequency of runs of 0s of length l is high for $0 \leq l \leq 20$.
- The frequency of runs of 0s of length l is very small for $l \geq 20$.
- Even within the range $0 \leq l \leq 20$, the frequency of runs of 0s of length l decreases rapidly with decreasing l.

If conventional run-length coding with block size b is used for compressing such test sets, every run of l 0s, $0 \leq l \leq 2^{b-1}$, is mapped to a b-bit codeword. This is clearly inefficient for the large number of short runs of 0s. Likewise, if Golomb coding with code parameter m is used, a run of l 0s is mapped to a codeword with $\lfloor \frac{l}{m} \rfloor + 1 + \log_2 m$ bits. Since Golomb code is a variable-to-variable-length code, each codeword consists of two parts—a group prefix of $\lfloor \frac{l}{m} \rfloor + 1$ bits and a tail of $\log_2 m$ bits. This is also inefficient for short runs of 0s. Clearly, test data compression is more efficient if the runs of 0s that occur more frequently are mapped to shorter codewords. This leads us to the notion of FDR codes.

The FDR code is constructed as follows: The runs of 0s are divided into groups $A_1, A_2, A_3, \ldots, A_k$, where k is determined by the length l_{max} of the longest run ($2^k - 3 \leq l_{max} \leq 2^{k+1} - 3$). Note also that a run of length l is mapped to group A_j where $j = \lceil \log_2(l + 3) - 1 \rceil$. The size of the ith group is equal to 2^i, i.e., A_i contains 2^i members. Each codeword consists of two parts—a group prefix and a tail. The group prefix is used to identify the group to which the run belongs and the tail is used to identify the members within the group. The encoding procedure is shown in Fig. 4 and the encoding of an input data stream is illustrated in Fig. 5. The FDR code has the following properties:

- For any codeword, the prefix and tail are of equal length. For example, the prefix and the tail are each one bit long for A_1, two bits long for A_2, etc.
- The length of the prefix for group A_i equals i. For example, the prefix is two bits long for group A_2.
- For any codeword, the prefix is identical to the binary representation of the run length corresponding to the first element of the group. For example, run length 8 is mapped to group A_3 and the first

Group	Run-length	Group prefix	Tail	Codeword
A_1	0	0	0	00
	1		1	01
A_2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A_3	6	110	000	110000
	7		001	110001
	8		010	110010
	9		011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111
...	...	...	...	...

Fig. 4. An example of FDR coding.

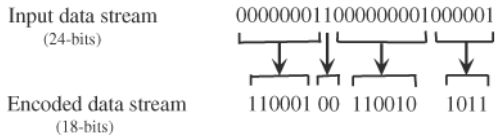

Fig. 5. FDR encoding procedure for an input data stream.

element of this group is run length 6. Hence, the prefix of the codeword for run length 8 is 110.

- The codeword size increases by two bits (one bit for the prefix and one bit for the tail) as we move from group A_i to group A_{i+1}.

Note that run lengths are also mapped to groups in conventional run length and Golomb coding. In run-length coding with block size b, the groups are of equal size, each containing 2^b elements. The number of code bits to which runs of 0s are mapped increases by b bits as we move from one group to another. On the other hand, in Golomb coding, the group size increases as we consider larger runs of 0s, i.e., A_i is smaller in size than A_{i+1}. However, the tails for Golomb codewords in different groups are of equal length ($\log_2 m$, where m is the code parameter) and the prefix increases by only one bit as we move from one group to another. Hence, Golomb coding is less effective when the runs of zeros are spread far from an "effective" range determined by m.

We now present a comparison between the three codes—conventional run-length code with block size $b = 3$, Golomb code with parameter $m = 4$, and the new FDR code. Fig. 6 shows the number of bits per codeword for runs of 0s of different lengths. It can be seen from the figure that the performance of the conventional run-length code is worse than that of the Golomb code when the run length l exceeds seven. The performance of the Golomb code is worse than that of the FDR code for $l \geq 24$. We also note that the new FDR code outperforms the other two types of codes for runs of length zero and one. Since the frequencies of runs of length zero and one are very high for precomputed test sets (Fig. 3) and FDR codes are significantly more efficient for $l \geq 24$, they outperform run length and Golomb codes for SOC test data compression. This is demonstrated by experimental results in Section 5.

3 Analysis of FDR Codes

In this section, we first develop an analysis technique to determine the worst-case and best-case compression that can be achieved using FDR codes for some generic parameters of precomputed test sets. We then present a probabilistic analysis for a memoryless data source and compare FDR codes to Golomb codes, run-length codes, and entropy bounds.

Suppose T_{diff} (or T_D if it is encoded directly) contains r 1s and a total of n bits. We first determine C_{max}, the number of bits in the encoded test set T_E in the worst case, i.e., when the compression is the least effective. In doing so, we also determine the distribution of the runs of 0s that gives rise to this worst-case compression.

Suppose T_{diff} contains k_i runs of length i with maximum run length l_{max}. Let the size of the encoded test set T_E be F bits and let $\delta = F - (n - r)$ measure the amount of compression achieved using FDR codes. To make the

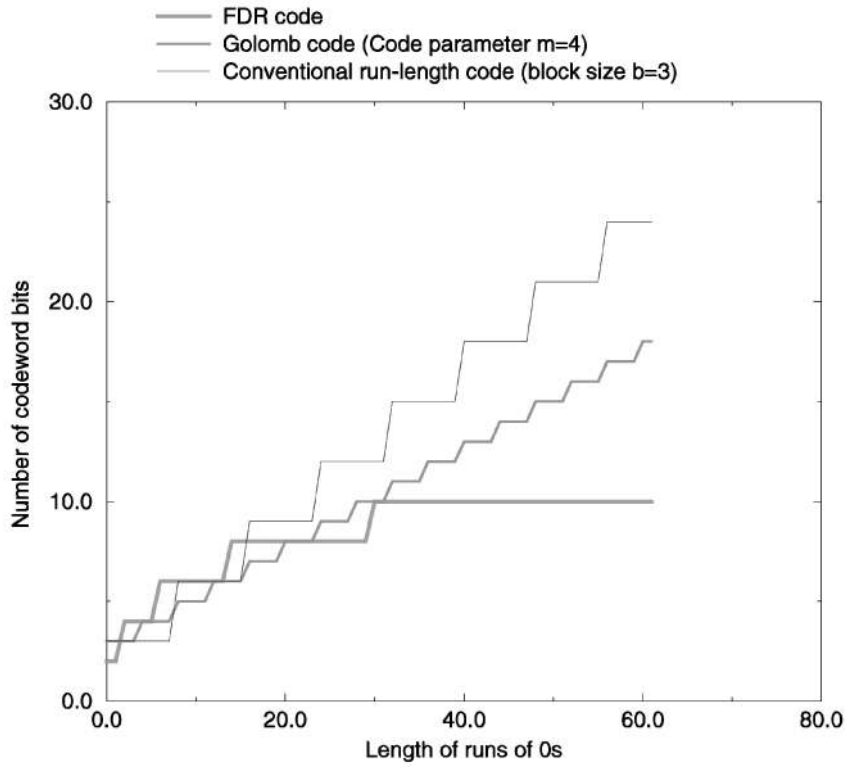

Fig. 6. Comparison of codeword size (bits) for different run lengths for the FDR code, Golomb code ($m = 4$), and conventional run-length code ($b = 3$).

TABLE 1
Worst-Case Compression Using FDR Codes

n	r	C_{max}	Percentage compression $(1 - C_{max}/n) \times 100$	Worst-case distribution of runs
1000	200	1000	0	100/2, 100/6
	100	674	32.6	59/6, 4/7, 37/14
	75	568	43.2	13/6, 3/7, 59/14
	50	400	60	30/14, 10/25, 10/28
	45	360	64	21/14, 1/17, 23/28
	40	320	68	10/14, 2/18, 28/28
2000	500	2250	Negative	375/2, 125/6
	200	1350	37.5	125/6, 75/14
	100	800	60	63/14, 2/21, 1/24, 34/28
	75	600	70	12/14, 1/21, 62/28

TABLE 2
Best-Case Compression Using FDR Codes

n	r	C_{min}	Percentage compression $(1 - C_{min}/n) \times 100$	Best-case distribution of runs
1000	200	530	47	178/1, 1/13, 21/29
	100	374	62.6	71/1, 1/17, 28/29
	75	334	66.6	44/1, 1/11, 30/29
	50	294	70.6	17/1, 1/5, 32/29
	45	282	71.8	11/1, 1/4, 32/29
	40	278	72.2	7/1, 1/25, 32/29
2000	500	1216	39.2	464/1, 1/21, 35/29
	200	744	62.8	142/1, 1/5, 57/32
	100	588	70.6	35/1, 1/9, 64/32
	75	548	72.6	8/1, 1/3, 66/32

presentation simpler, we subtract a constant term from F for all distributions of runs, given a fixed n and r. If the FDR coding procedure of Fig. 4 is applied to T_{diff}, we have $\delta = 2k_0 + k_1 + 2k_2 + k_3 - k_5 - k_7 - 2k_8 - 3k_9 \pm \cdots$ (up to l_{max}). This can be explained as follows: For each run of 0 of length i, we compare the size of the run length (i) with the size of the corresponding codeword. For example, the codeword corresponding to a run of length 0 contains two bits (one more than the original run), the codeword for run length 1 is of the same size as the original run length, etc. The difference between these two quantities contributes to δ and it appears as the coefficient of the appropriate k_i term in the equation for δ.

We next use the following simple integer linear programming (ILP) model to determine the maximum value of δ. This yields the worst-case compression (C_{max}) using FDR codes.

Maximize: $\delta = 2k_0 + k_1 + 2k_2 + k_3 - k_5 - k_7 - 2k_8 - 3k_9 \pm \cdots$ (up to l_{max}) **subject to**: 1) $\sum_{i=1}^{l_{max}} ik_i = n - r$ and 2) $\sum_{i=1}^{l_{max}} k_i = r$.

This ILP model can be easily solved, e.g., using a solver such as *lpsolve* [14], to obtain the worst-case values for the k_i's. Note that, even though l_{max} appears in the above ILP model, we do not make any explicit use of it. Our goal here is to determine a worst-case distribution of the runs of 0s. Generally, short run lengths yield the worst-case compression; however, if l_{max} must exceed a minimum value to satisfy constraints 1) and 2) above, we can use *lpsolve* to determine the minimum l_{max} by incrementally increasing l_{max} until the optimization problem becomes feasible.

Table 1 lists the size C_{max} of the encoded data set for worst-case compression for various values of n and r. The last column shows a distribution of runs for which the worst-case compression is achieved (a/b indicates a runs of length b). Note that this distribution is not unique since a number of run lengths can yield the worst-case distribution. Note also that the worst-case percentage compression is negative when r is high relative to n—this is unlikely to be the case for test sets (don't-cares mapped to 0s) or difference vector sequences for which r is generally very small.

Next, we analyze the best-case compression achieved using FDR codes for any given n and r. Since the compression is better for longer run lengths, we also need to constrain the maximum run length in this case. As before, we formulate this problem using ILP and the following model can be solved using *lpsolve* to obtain a best-case distribution of runs and C_{min}, the number of bits in the encoded test set in the best case.

Minimize: $\delta = 2k_0 + k_1 + 2k_2 + k_3 - k_5 - k_7 - 2k_8 - 3k_9 \pm \cdots$ (upto l_{max}) **subject to**: 1) $\sum_{i=1}^{l_{max}} ik_i = n - r$ and 2) $\sum_{i=1}^{l_{max}} k_i = r$.

Table 2 lists the run-length distributions corresponding to the best case compression using FDR codes. The corresponding percentage compression values are also listed. In Fig. 7, a plot shows the lower and upper bounds on the percentage compression as the number of runs r is varied (for $n = 1,000$). We note that, for small values of r, the bounds are very close to each other, hence, the FDR code is robust, i.e., its efficiency is relatively insensitive to variations in the distributions of the runs.

Next, we analyze FDR codes for a memoryless data source that produces 0s and 1s with probabilities p and $(1 - p)$, respectively. The purpose of this analysis is to examine the fundamental limits of the FDR code and to demonstrate its effectiveness for all values of p, $0 < p < 1$. The entropy $H(p)$ of this memoryless source is given by the following equation [24]:

$$H(p) = -p \log_2 p - (1 - p) \log_2 (1 - p).$$

We first analyze Golomb codes with group parameter m. This is necessary to determine a baseline for evaluating FDR codes. (The reader is referred to [7] for a review of Golomb codes.) The smallest and the longest run lengths that belong to group A_k are given by $(k - 1)m$ and $(km - 1)$,

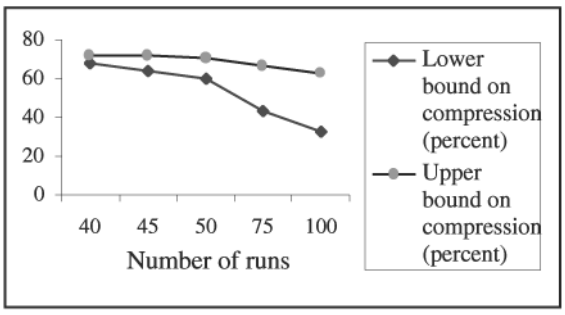

Fig. 7. Comparison between the upper and lower bounds on percentage compression for $n = 1,000$.

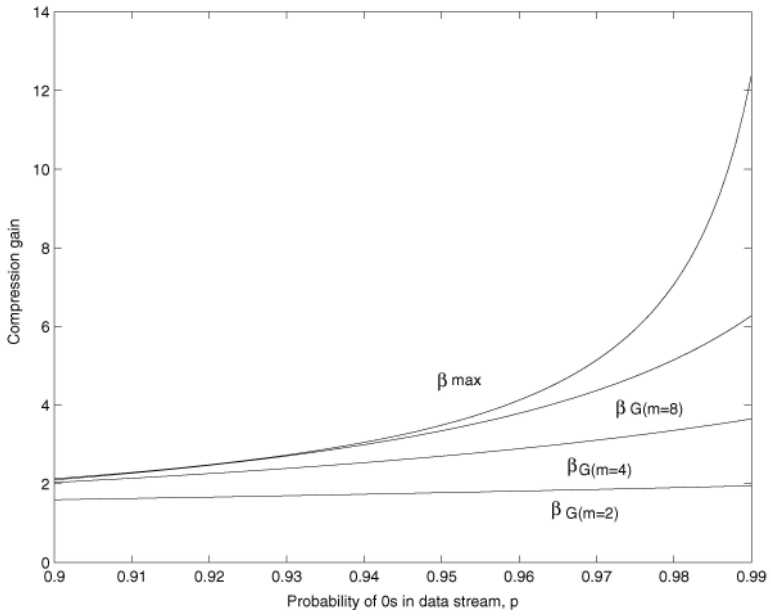

Fig. 8. Compression gain obtained with Golomb codes.

respectively. Therefore, the probability that an arbitrarily chosen run of length i belongs to group A_k is given by:

$$P(i,k) = \sum_{i=(k-1)m}^{(km-1)} p^i(1-p) = (1-p^m)p^{(k-1)m}.$$

The codewords in group A_k consist of $(\log_2 m + k)$ bits [7]. Therefore, the average codeword length $\bar{G}$ for Golomb codes is given by:

$$\bar{G} = \sum_{k=1}^{\infty}(1-p^m)p^{(k-1)m}(\log_2 m + k) = \left(\log_2 m + \frac{1}{(1-p^m)}\right).$$

We next determine λ, the average number of bits in any run generated by the data source. It can be easily shown that:

$$\lambda = 1 + \sum_{i=1}^{\infty} ip^i(1-p) = \frac{1}{1-p}.$$

The effectiveness of compression is measured by the compression gain β_G, which is defined as the ratio of the average number of bits in any run to the average codeword size, i.e., $\beta_G = \frac{\lambda}{\bar{G}}$. This yields

$$\beta_G = \frac{1}{(1-p)\left(\log_2 m + \frac{1}{1-p^m}\right)}.$$

An upper bound on the compression gain is obtained from the entropy $H(p)$ of the source using the following equation:

$$\beta_{max} = \frac{1}{H(p)}.$$

Fig. 8 shows the relationship between β_G and p for three values of m. The upper bound β_{max} is also shown in the figure. The figure shows that, while the compression gain for Golomb codes is significant, especially for large values of p, there is a significant difference between β_G and the upper bound β_{max}. This motivates the need for FDR codes.

We next analyze the effectiveness of conventional run-length codes for a memoryless data source. Let group A_k for run-length codes contain $(M+1)$ members such that $M = 2^N - 1$ for some positive number N. The parameter M must be kept small, e.g., $M = 15$, in order to keep the decoder simple. The smallest and the longest run length that belong to group A_k are given by $M(k-1)$ and Mk, respectively. Therefore, the probability that an arbitrarily chosen run of length i belongs to group A_k is given by:

$$P(i,k) = \sum_{i=(k-1)M}^{(kM-1)} p^i(1-p) + p^M k = \frac{p^{kM}}{p^M}.$$

The codewords in group A_k consist of $k\log_2(M+1)$ bits. Therefore, the average codeword length $\bar{R}$ for run-length codes is given by:

$$\bar{R} = \sum_{k=1}^{\infty}\frac{p^{kM}}{p^M}k\log_2(M+1) = \frac{\log_2(M+1)}{(1-p^M)^2}.$$

The compression gain β_R for run-length codes is given by:

$$\beta_R = \frac{(1-p^M)^2}{(1-p)\log_2(M+1)}.$$

Finally, we analyze the effectiveness of FDR codes for a memoryless data source. The smallest and the longest run

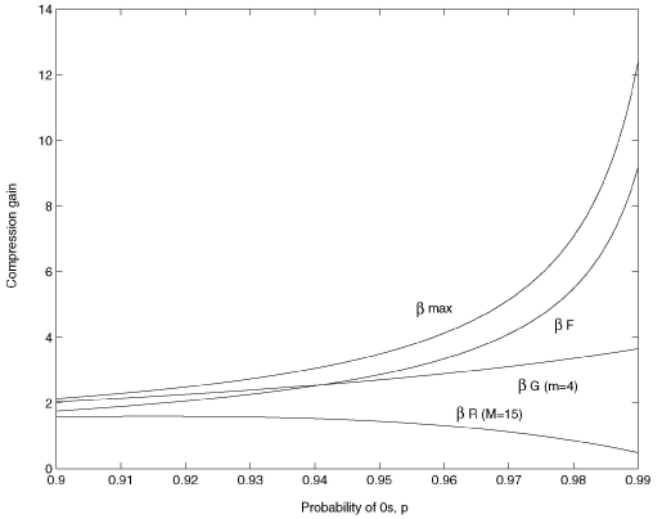

Fig. 9. Comparison of compression gain obtained with FDR codes, Golomb codes, and run-length codes for $0.9 \leq p \leq 0.99$.

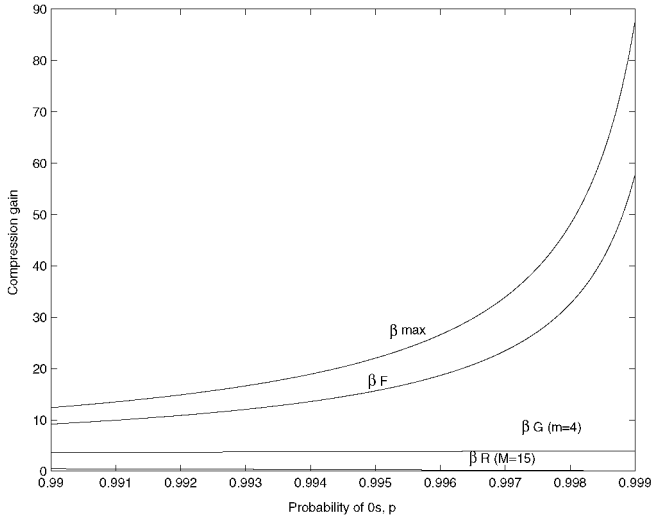

Fig. 10. Comparison of compression gain obtained with FDR codes, Golomb codes, and run-length codes for $0.99 \leq p \leq 0.999$.

lengths that belong to group A_k are given by $(2^k - 2)$ and $(2^{k+1} - 3)$, respectively. Therefore, the probability $P(i,k)$ that an arbitrarily chosen run of length i belongs to group A_k is given by:

$$P(i,k) = \sum_{i=2^k-2}^{2^{k+1}-3} p^i(1-p) = p^{2^k-2}(1-p^{2^k}).$$

The codeword in group A_k consists of $2k$ bits. Therefore, the average codeword length $\bar{F}$ for FDR codes is given by:

$$\bar{F} = \sum_{k=1}^{\infty} 2kp^{2^k-2}(1-p^{2^k}) = 2\sum_{k=1}^{\infty} p.^{2^k-2}.$$

Even though we do not have a closed-form expression for $\bar{F}$, the above equation can be used to evaluate the effectiveness of FDR codes. The compression gain β_F for FDR codes is given by

$$\beta_F = \frac{1}{2(1-p)\sum_{k=1}^{\infty} p^{2^k-2}}.$$

Fig. 9 shows a comparison between the compression gain β_F, β_G, and β_R, where β_R is the compression gain corresponding to run-length codes. The upper bound β_{max} is also shown in the figure. The figure shows that compression gain for FDR codes is always higher than that for Golomb codes for all values of $p > 0.942$. Fig. 10 shows that, for large values of p, there is a significant difference between β_F and β_G. The figures also show how closely the FDR gain curve follows the upper bound β_{max}. Hence, these results show that FDR codes are inherently superior to Golomb codes and run-length codes and they allow us to approach the fundamental entropy bounds.

4 Extensions to the FDR Code and Test Data Decompression

In this section, we describe some extensions to the basic FDR code described in Section 2 and then present the data compression/decompression method for testing SOCs. Additional practical issues related to the decompression architecture are discussed in this section. We design the on-chip decoder and show that it is independent of the core under test and the precomputed test set.

The FDR coding algorithm described in Section 2 represents an instance of a code belonging to the class of more general FDR codes. This instance is especially suitable when the frequencies of runs decreases monotonically, i.e., the number of runs of length l is greater than the number of runs of length $l+1$. It is also effective when the cumulative frequency of the runs in any group A_i exceeds the cumulative frequency of the runs in group A_{i+1}. However, for precomputed test sets, the run-length frequencies do not always decrease monotonically. For such nonmonotonically decreasing run lengths, the compression can be increased by extending the basic FDR code as described below.

For each group A_i, we calculate the cumulative frequency of the run lengths in that group. This is done by simply adding the frequencies of the run lengths in that group. Next, instead of assigning the group prefix as shown in Fig. 4, we assign the prefix based on the cumulative frequency of that group. A group with a large cumulative frequency is assigned a short prefix. In this way, the size of the encoded test set can be reduced by carrying out a small amount of preprocessing and by using a mapping logic block (outlined later) in the decoder.

Let us consider a hypothetical test set with the distribution of run lengths as shown in Fig. 11. Let C_i denote the cumulative frequencies for group A_i. For our example, these cumulative frequencies are: $C_1 = 45$, $C_2 = 40$, and $C_3 = 150$. Fig. 11 shows the group prefix assignment for the FDR code described in Section 2, as well as the more efficient prefix assignment based on the cumulative frequencies. Since

Group	Run-length	Frequency	Group prefix	Tail	Codeword size (bits)	Modified prefix	Codeword size (bits)
	0	25	0	0			
A_1	1	20		1	2	10	3
	2	7		00			
A_2	3	13	10	01	4	110	5
	4	12		10			
	5	8		11			
	6	25		000			
	7	45		001			
	8	30		010			
A_3	9	5	110	011	6	0	4
	10	7		100			
	11	8		101			
	12	10		110			
	13	20		111			

Fig. 11. An example of an extended FDR code for nonmonotonically decreasing run lengths of 0s.

$C_3 > C_1 > C_2$, the group prefixes for A_1, A_2, and A_3 are 10, 110, and 0, respectively.

For this example, the size of the test set is 1,624 bits. If the cumulative frequencies are not used for prefix assignment, the size of the encoded test set is 1,150 bits, which implies a compression of 29.68 percent. On the other hand, if the cumulative frequencies are used as described above, the encoded test set contains only 935 bits, which implies 42.42 percent compression. Hence, the compression increases by 12.74 percent when prefix assignment is done based on cumulative frequencies.

4.1 Test Data Compression/Decompression

We next describe the test data compression procedure, the decompression architecture, and the design of the on-chip decoder. We show that the decoder is simple and scalable and independent of both the core under test and the precomputed test set. Moreover, due to its small size, it does not introduce significant hardware overhead.

The encoding procedure for a block of data using FDR codes was outlined in Section 1. Let T_D be the uncompacted test set and let T_{diff} be the corresponding difference vector test sequence. If T_D is used for compression, all don't-cares in it are carefully mapped to 0 to obtain long runs of 0s. If T_{diff} is used for compression, then the don't-cares are mapped to 0 or 1 so as to obtain long runs of 0s in the difference vector sequence. For full-scan cores, the patterns can be reordered. However, since the ordering problem is equivalent to the NP-Complete Traveling Salesman problem, a heuristic algorithm is used to reorder the patterns [6], [7].

For sequential cores, a boundary scan register is required at the functional inputs for decompression [7]. This register is usually available for cores that are wrapped. The encoded data is fed bitwise to the decoder, which produces a sequence of difference vectors. The decompression hardware then translates the difference vectors into the test patterns, which are applied to the core. If an existing boundary-scan register or the P1500 test wrapper is used to decompress the test data, the decoder and a small amount of synchronizing logic are the only additional logic required.

We first design the decoder for the basic FDR code presented in the Section 2 and then describe the mapping logic that allows cumulative frequencies to be used for prefix assignment. The design is similar to the FSM-based decoder in [6], [7]. Issues related to data synchronization are described in [7]. The decoder decompresses the encoded

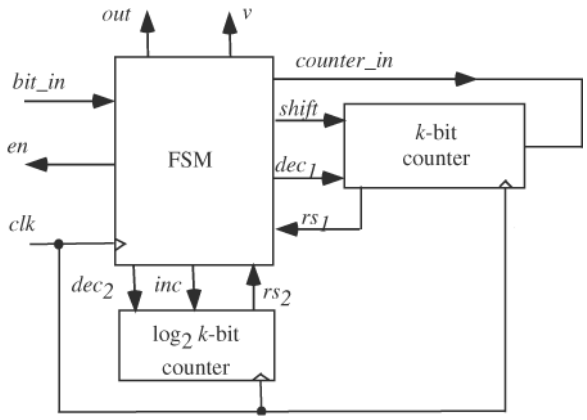

Fig. 12. Block diagram of the decoder used for on-chip pattern decompression.

test set T_E and outputs T_D. It can be efficiently implemented by a k-bit counter, a $\log_2 k$-bit counter, and a finite-state machine (FSM). The block diagram of the decoder is shown in Fig. 12. The bit_in is the input to the FSM and an enable (en) signal is used to input encoded data when the decoder is ready. The FSM output $counter\_in$ is used to shift in the prefix or the tail into the k-bit counter and the signals $shift$, dec_1, and rs_1 are used to shift the data in, to decrement, and to indicate the reset state of the counter, respectively. The second counter of $\log_2 k$-bit is used to count the length of the prefix and the tail so as to identify the group. The signals inc and dec_2 are used to increment and decrement the counter, respectively, and rs_2 indicates that the counter has finished counting. Finally, the signal out is the decoder output and v indicates when the output is valid. The operation of the decoder is as follows:

- The FSM feeds the k-bit counter with the prefix. The end of the prefix is identified by the separator 0. The en, $shift$, and inc signals are high till the 0 is received.
- The FSM output 0s and decrements the k-bit counter and makes the signal dec_1 high. It continues to output 0s until rs_1 goes high. The signal v is used to indicate a valid output.
- The tail part is shifted in until the $\log_2 k$-bit counter resets to zero. The dec_2 signal then goes high, the counter is decremented, and the signal rs_2 indicates when it is in the zero state.
- The FSM output 0s corresponding to the tail followed by a 1 at the end of tail decoding.

The state diagram for the FSM used for pattern decompression is shown in Fig. 13. We note that the state diagram consists of only nine states. We synthesized the FSM using Synopsys design compiler [26]. The synthesized circuit contains only four flip-flops and 38 gates. Therefore, the additional hardware needed for the decoder is very small and existing counters on the SOC can be reused for decompression.

The above decoder can be easily modified for decompressing data encoded using cumulative frequencies for the groups. Since the use of the cumulative frequencies affects only the prefix and not the tail, we only need to add a

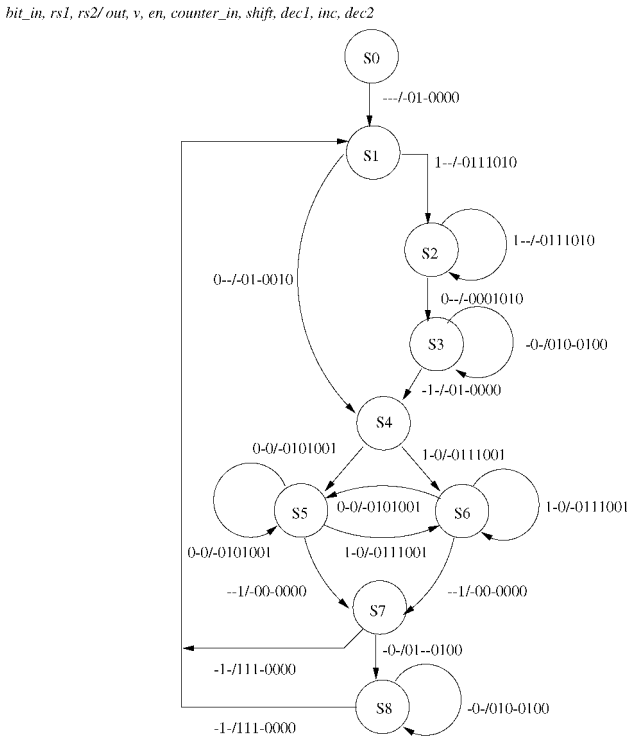

Fig. 13. State diagram of the FSM used for on-chip pattern decompression.

mapping logic block between the encoded data stream and the decode FSM. Thus, the mapping logic feeds the decode FSM and transforms the prefixes in the encoded data to the prefix assignment of Fig. 4. Fig. 14 sketches the position of the mapping logic relative to the decoder. For the hypothetical test case considered in Fig. 11, we have the following mapping: $10 \Rightarrow 0, 110 \Rightarrow 10, 0 \Rightarrow 110$. For example, if the mapping logic receives 110, the output to the decode FSM is 10.

In our experiments with ISCAS 89 benchmark circuits, we observed that the run lengths were never long enough to exceed group A_{10}. Therefore, in the worst case, the mapping logic is required for only 10 prefixes. We show in Section 5 that a small amount of additional compression is achieved using the mapping logic. Thus, if area overhead is a major concern, then the decoder can be designed without the mapping logic, thereby trading off area overhead with the amount of compression.

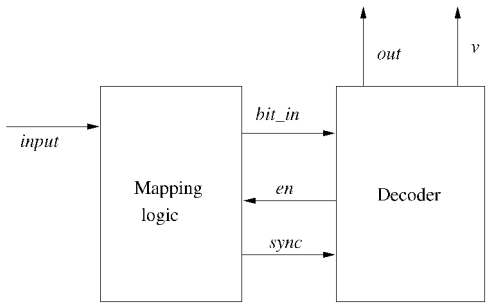

Fig. 14. The mapping logic and the decoder used for pattern decompression.

TABLE 3
Compression Obtained Using T_{diff}

Circuit	Percentage compression (Golomb)	Percentage compression (FDR)	Size of T_D (bits)	Size of T_E (bits)	No. of bits for Mintest
s5378	53.73	61.32	23754	9188	20758
s9234	59.85	60.63	39273	15460	25935
s13207	84.33	87.67	165200	20368	163100
s15850	66.55	71.95	76986	21590	57434
s38417	58.08	65.35	164736	57066	113152
s38584	59.61	64.67	199104	70328	161040

5 Experimental Results

We now present experimental results on test data compression for the large ISCAS benchmark circuits. We considered both full-scan and nonscan circuits for the proposed compression/decompression scheme. For full-scan circuits, patterns were reordered to achieve higher compression whereas no ordering was done for the nonscan circuits. For all the full-scan circuits, we considered a single scan chain. The compression percentage was computed as follows:

$$\begin{aligned} &Percentage\ compression \\ &= \frac{Size\ of\ the\ test\ set - Size\ of\ encoded\ test\ set}{Size\ of\ the\ test\ set} \times 100 \\ &= \frac{|T_D| - |T_E|}{|T_D|} \times 100. \end{aligned}$$

The first set of experimental data that we present is based on the use of difference vector sequences T_{diff} obtained from partially-specified test sets (test cubes). Table 3 presents results for test cubes obtained using the Mintest ATPG program [23] with dynamic compaction. We compare the compression obtained using FDR coding with Golomb coding and also with fully compacted test sets generated using Mintest. Note that there is no loss in fault coverage due to on-chip decompression. We carried out our experiments using a Sun Ultra 10 workstation with a 333 MHz processor and 256 MB of DRAM. The table lists the percentage compression, sizes of the precomputed (original) test sets, sizes of the encoded test sets, and the sizes of the smallest ATPG-compacted test sets.

Table 3 shows that FDR codes provide better compression than Golomb codes in all cases.[1] For the benchmark circuit s38417, there is as much as a 7 percent increase in compression. We also note that that, in all cases, the size of the encoded test set T_E is much smaller than the compacted test set obtained using Mintest. On average, the percentage compression using FDR codes was 4.9 percent higher than that obtained using Golomb codes. Note that test data compression leads to encoded test sets that are always smaller than ATPG compacted test sets [7]. Moreover, the testing time is reduced significantly [10], [11] and substantial reduction is obtained in power consumption during scan testing [12]. Testing time is decreased by employing faster on-chip decompression of encoded test data. The compressed data can be transferred at a slower rate from

1. The percentage compression results for Golomb codes reported here are better than those reported in [6], [7] due to the use of an improved pattern reordering heuristic.

TABLE 4
Compression Obtained Using T_D

Circuit	Percentage compression (Golomb)	Percentage compression (FDR)	Size of T_D (bits)	Size of T_E (bits)	No. of bits for Mintest
s5378	37.11	48.02	23754	12346	20758
s9234	45.25	43.59	39273	22152	25935
s13207	79.74	81.30	165200	30880	163100
s15850	62.82	66.22	76986	26000	57434
s38417	28.37	43.26	164736	93466	113152
s38584	57.17	60.91	199104	77812	161040

the ATE to the SOC. This allows the use of low-end ATEs with less memory and slower clock rates [11]. Test power is reduced by decreasing the power dissipation during scan shifting operation. Scan power consumption during scan-in and scan-out has been shown to be a dominant part of the total power dissipation during scan-based testing [13]. Scan power can be decreased by carefully mapping the don't-care bits in the test cubes. Therefore, significant savings in average and peak power can be obtained using the methods based on test data compression [12].

Table 4 demonstrates that the use of test cubes T_D (with all the don't-cares mapped to 0) also yields very high compression. The advantage of using T_D for compression is that the decompression architecture for on-chip pattern generation does not require a separate CSR. For circuits with long scan chains, additional CSRs of lengths equal to the scan chain lengths increase the hardware overhead significantly. Therefore, compressing T_D to generate the encoded test set not only yields smaller test sets but also reduces the hardware overhead.

Next, we present experimental results on test data compression for nonscan circuits. We obtained the test sequences for these circuits from HITEC [27]. No reordering of test patterns was done during compression. Table 5 lists the sizes of the precomputed test sequences and the percentage compression obtained in each case for the basic FDR code and the modified code using mapping logic. Not surprisingly, we found out that more compression is obtained using the mapping logic. The results also show that very high compression is achieved for nonscan circuits using FDR codes.

Next, we compare the experimental results to the theoretical upper bounds on the compression predicted by the "entropy" of the test data. Let S be a data sequence with patterns $s_1, s_2, s_3, \ldots, s_k$ and let $p_1, p_2, p_3, \ldots, p_k$ be the relative frequencies of the patterns in S, respectively. An entropy measure of S is given by:[2]

$$E(S) = \sum_{i=1}^{k} p_i \log_2(1/p).$$

Intuitively, $E(S)$ provides a lower bound on the average number of bits required to encode each pattern in S [25]. If b is the sum of the relative frequencies of $s_1, s_2, s_3, \ldots, s_k$, a lower bound on the encoded data stream for S is given by $bE(S)$.

2. An explicit distinction is being made here between the formal notion of entropy for a probabilistic data source and the entropy measure for a deterministic test set with relative frequencies of test patterns.

TABLE 5
Compression Obtained Using T_D and FDR Codes for Nonscan Circuits with and without Mapping Logic

Circuit	Size of T_D (bits)	FDR code (without mapping logic)		FDR code (with mapping logic)	
		Percentage compression	Size of T_E (bits)	Percentage compression	Size of T_E (bits)
s953	1168	73.45	310	75.94	281
s5378	169995	81.16	32020	81.45	31529
s13207	42284	93.44	2770	95.17	2039
s15850	147070	73.93	38330	82.41	25869
s35932	430353	83.35	71650	86.84	56618
s38417	22624	84.58	3488	89.26	2428

Table 6 lists the sizes of the precomputed test sequences lower bounds on the size of encoded data and percentage compression based on entropy analysis and the actual size of encoded test data and percentage compression obtained in each case for the FDR code using T_D. We find that, in almost all the cases, the percentage compression obtained is very close to the entropy bound. Table 7 shows the lower bounds on the size of encoded data and percentage compression based on entropy analysis and the actual size of encoded test data and percentage compression obtained in each case for the FDR code using T_{diff}. The results show that the difference between the lower bound on the size of the encoded data obtained using entropy and FDR codes is less than 2 percent in all cases.

We next present experimental results for two real test sets from industry. The test set for the first circuit (CKT1) from IBM consists of 32 statically compacted scan vectors (a total of 362,922 bits of test data per vector). This microprocessor design consists of 3.6 million gates and 726,000 latches. The compression results using the Golomb and the FDR code and the entropy bounds for the 32 scan vectors are shown in Table 8. Note that we

TABLE 6
Comparison of Compression Predicted by Entropy of Test Data and FDR Codes for T_D

Circuit	Size of T_D (bits)	Entropy bounds		FDR code	
		Percentage compression	Size of encoded data (bits)	Percentage compression	Size of T_E (bits)
s5378	23754	52.42	11301.3	48.02	12346
s9234	39273	47.90	20547.9	43.59	22152
s13207	165200	83.65	27009.7	81.30	30880
s15850	76986	68.2	24477	66.22	26000
s38417	164736	54.49	74957.4	43.26	93466
s38584	199104	62.48	74684.2	60.91	77812

TABLE 7
Comparison of Compression Predicted by Entropy of Test Data and FDR Codes for T_{diff}

Circuit	Size of T_D (bits)	Entropy bounds		FDR code	
		Percentage compression	Size of encoded data (bits)	Percentage compression	Size of T_E (bits)
s5378	23754	63.21	8738.4	61.32	9188
s9234	39273	63.06	14505.9	60.63	15460
s13207	165200	88.77	18543.9	87.67	20368
s15850	76986	73.51	20387.8	71.95	21590
s38417	164736	67.88	52903.9	65.35	57066
s38584	199104	65.98	67733.1	64.67	70328

TABLE 8
Compression Obtained for CKT1 from IBM

Scan vector	Size of T_D (bits)	Size of encoded test set (bits)			Percentage compression		
		Golomb ($m = 128$)	FDR	Entropy bound	Golomb ($m = 128$)	FDR	Entropy bound
1	362922	27708	23800	20509.3	92.36	93.44	94.34
2	362922	20406	17410	14523.6	94.37	95.20	95.99
3	362922	20406	17410	14523.6	94.37	95.20	95.99
4	362922	20244	16366	13736	94.42	95.49	96.21
5	362922	18267	14802	12195.7	94.96	95.92	96.63
6	362922	14081	12152	9671.14	96.12	96.65	97.33
7	362922	14418	10432	8218.14	96.02	97.12	97.73
8	362922	14370	10058	7899.4	96.04	97.22	97.82
9	362922	12875	9718	7543.95	96.45	97.32	97.92
10	362922	10414	6866	5071.56	97.13	98.10	98.60
11	362922	9729	7622	5589.85	97.31	97.89	98.45
12	362922	11199	7908	5920.32	96.91	97.82	98.36
13	362922	8618	6240	4527.83	97.62	98.28	98.75
14	362922	8675	6380	4622.78	97.60	98.24	98.72
15	362922	8123	5326	3845.45	97.76	98.53	98.94
16	362922	6508	4000	2749.55	98.20	98.89	99.24
17	362922	7629	4982	3582.11	97.89	98.62	99.01
18	362922	7974	5996	4293.5	97.80	98.34	98.81
19	362922	7668	4842	3471.37	97.88	98.66	99.04
20	362922	6791	4256	2852.64	98.12	98.82	99.21
21	362922	8149	6022	4242.26	97.75	98.34	98.83
22	362922	7120	4892	3388.64	98.03	98.65	99.06
23	362922	6959	4266	3049.34	98.08	98.82	99.15
24	362922	5856	2836	1955.32	98.38	99.21	99.46
25	362922	5911	3330	2188.08	98.37	99.08	99.39
26	362922	6254	3994	2625.88	98.27	98.89	99.27
27	362922	6903	4258	2989.75	98.09	98.82	99.17
28	362922	9212	5220	3925.03	97.46	98.56	98.91
29	362922	6460	3982	2726.91	98.22	98.90	99.24
30	362922	5580	2386	1655.87	98.46	99.34	99.54
31	362922	5994	3418	2282.86	98.34	99.05	99.37
32	362922	6118	3208	2211.5	98.31	99.11	99.39

obtain a staggering 97.10 percent compression on average. Table 8 also shows the entropy bounds for the test vectors. The difference between the entropy-based lower bound on the size of the encoded data and the size of FDR-coded data is less than 1 percent in all cases.

Table 9 shows experimental results for a second microprocessor circuit (CKT2) from IBM. T_D for this consists of a set of four scan vectors (a total of 1,031,072 bits of test data per vector); this design contains 1.2 million gates and 32,200 latches. Over 95 percent compression is obtained for the test cubes of CKT2. The compression results here are also within 1 percent of the entropy bounds.

Finally, we compare the compression obtained using the FDR code to the Unix file compression utilities *gzip* and *compress*. In order to carry out a fair comparison, we converted the encoded test sets obtained using the FDR code to a binary format. Table 10 shows the size of the encoded test set and the percentage compression obtained using *gzip*, *compress*, and the FDR code. We note that, in almost all cases, the compression obtained using the FDR code is close to the compression obtained using the two Unix utilities. For s9234, the FDR code outperforms both *gzip* and *compress*. The *gzip* and *compress* utilities employ far more complex encoding algorithms than the FDR code. Hence, it is inconceivable that they can be decoded using simple hardware techniques for TRP; the corresponding decompression utilities (*gunzip* and *uncompress*) are usually implemented in software. It is therefore particularly noteworthy that the simpler FDR code, which can be easily used for on-chip decompression, provides almost as much compression as *gzip* and *compress*.

6 Conclusions

We have presented a new class of variable-to-variable-length compression codes that are designed using the

TABLE 9
Compression Obtained for CKT2 from IBM

Scan vector	Size of T_D (bits)	Size of encoded test set (bits)			Percentage compression		
		Golomb ($m = 128$)	FDR	Entropy bound	Golomb ($m = 128$)	FDR	Entropy bound
1	1031072	82242	47998	41393.8	92.02	95.34	95.98
2	1031072	79927	49612	42938.4	92.24	95.18	95.83
3	1031072	74902	46986	40961.1	92.73	95.44	96.02
4	1031072	73013	43396	37650.4	92.91	95.79	96.34

TABLE 10
Comparison of Compression Obtained Using *gzip*, *compress*, and FDR Code

Circuit	Size of encoded test set (bytes)			Percentage compression		
	gzip	*compress*	FDR code	*gzip*	*compress*	FDR code
s9234	5913	6398	4910	46.19	41.77	55.31
s13207	8747	9279	20650	94.70	94.38	87.50
s15850	8135	8096	9624	89.43	89.48	87.49
s38417	10096	16833	20592	93.87	89.78	87.50
s38584	23948	21467	24888	87.97	89.21	87.50

distributions of the runs of 0s in typical test sequences. We refer to these as frequency-directed run-length (FDR) codes. We have presented experimental results for the ISCAS 89 benchmark circuits and two production circuits from IBM to show that FDR codes outperform Golomb codes for test data compression. We have presented a decompression architecture for FDR codes, as well as an analytical characterization of the amount of compression that can be expected using these codes. Our analysis provides lower and upper bounds on the compression expected for some generic parameters of the test set. These bounds are especially tight when the number of runs is small. This shows that FDR codes are robust, i.e., they are insensitive to variations in the input data stream.

We have also presented a probabilistic analysis of the FDR code for a memoryless data source in order to highlight its inherent superiority for all data sources. Experimental results show that the compression for FDR codes is quite close to the fundamental entropy bounds for the benchmark circuits. We are currently reviewing tester technology to determine practical ways to adapt and configure testers to apply FDR-coded test data to an SOC under-test. This work will pave the way for easy adoption of test data compression in the semiconductor industry.

ACKNOWLEDGMENTS

The authors would like to thank Brion Keller of IBM Corporation for providing scan vectors for a production circuit. They also thank Rafael Medina for helping us in our experiments with the IBM test data. This research was supported in part by the US National Science Foundation under grant number CCR-9875324 and in part by an equipment grant from Intel Corporation. Preliminary versions of this paper appeared in the *Proceedings of the IEEE VLSI Test Symposium*, pp. 42-47, Los Angeles, April-May 2001 and will appear in the *Proceedings of the IEEE VLSI Test Symposium*, pp. 91-96, Monterey, California, April-May 2002.

REFERENCES

[1] Y. Zorian, E.J. Marinissen, and S. Dey, "Testing Embedded-Core Based System Chips," *Proc. Int'l Test Conf.,* pp. 130-143, 1998.
[2] B.T. Murray and J.P. Hayes, "Testing ICs: Getting to the Core of the Problem," *Computer,* vol. 29, no. 11, pp. 32-38, Nov. 1996.
[3] V. Iyengar, K. Chakrabarty, and B.T. Murray, "Deterministic Built-In Self Testing of Sequential Circuits Using Precomputed Test Sets," *J. Electronic Testing: Theory and Applications (JETTA),* vol. 15, pp. 97-114, Aug./Oct. 1999.
[4] A. Jas, J. Ghosh-Dastidar, and N.A. Touba, "Scan Vector Compression/Decompression Using Statistical Coding," *Proc. IEEE VLSI Test Symp.,* pp. 114-120, 1999.
[5] A. Jas and N.A. Touba, "Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Design," *Proc. Int'l Test Conf.,* pp. 458-464, 1998.
[6] A. Chandra and K. Chakrabarty, "Test Data Compression for System-on-a-Chip Using Golomb Codes," *Proc. IEEE VLSI Test Symp.,* pp. 113-120, 2000.
[7] A. Chandra and K. Chakrabarty, "System-On-a-Chip Test Data Compression and Decompression Architectures Based on Golomb Codes," *IEEE Trans. Computer-Aided Design,* vol. 20, no. 3, pp. 355-368, Mar. 2001.
[8] D. Salomon, *Data Compression,* second ed. Springer-Verlag, 2000.
[9] H. Ichihara, K. Kinoshita, I. Pomeranz, and S.M. Reddy, "Test Transformation to Improve Compaction by Statistical Encoding," *Proc. Int'l Conf. VLSI Design,* pp. 294-299, 2000.
[10] A. Chandra and K. Chakrabarty, "Efficient Test Data Compression and Decompression for System-On-a-Chip Using Internal Scan Chains and Golomb Coding," *Proc. Design, Automation, and Test in Europe (DATE) Conf.,* pp. 145-149, 2001.
[11] A. Chandra and K. Chakrabarty, "Test Resource Partitioning and Reduced Pin-Count Testing Based on Test Data Compression," *Proc. Design, Automation and Test in Europe (DATE) Conf.,* pp. 598-603, 2002.
[12] A. Chandra and K. Chakrabarty, "Combining Low-Power Scan Testing and Test Data Compression for System-On-a-Chip," *Proc. IEEE/ACM Design Automation Conf. (DAC),* pp. 166-169, June 2001.
[13] J. Saxena, K. Butler, and L. Whetsel, "An Analysis of Power Reduction Techniques in Scan Testing," *Proc. Int'l Test Conf.,* pp. 670-677, 2001.
[14] M. Berkelaar, *lpsolve.* version 3.0. Eindhoven Univ. of Technology, Design Automation Section, Eindhoven, The Netherlands, ftp://ftp.ics.ele.nl/pub/lp_solve, 2000.
[15] S.C. Ma, P. Franco, and E.J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experimental Results," *Proc. Int'l Test Conf.,* pp. 663-672, 1995.
[16] S.M. Reddy, I. Pomeranz, and S. Kajihara, "On the Effects of Test Compaction on Defect Coverage," *Proc. IEEE VLSI Test Symp.,* pp. 430-435, 1996.
[17] I. Pomeranz and S.M. Reddy, "Stuck-At Tuple-Detection: A Fault Model Based on Stuck-At Faults for Improved Defect Coverage," *Proc. IEEE VLSI Test Symp.,* pp. 289-294, 1998.
[18] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, "A Mixed-Mode BIST Scheme Based on Reseeding of Folding Counters," *Proc. Int'l Test Conf.,* pp. 778-784, 2000.
[19] H.-J. Wunderlich and G. Kiefer, "Bit-Flipping BIST," *Proc. Int'l Conf. Computer-Aided Design,* pp. 337-343, 1996.
[20] N.A. Touba and E.J. McCluskey, "Altering a Pseudo-Random Bit Sequence for Scan Based BIST," *Proc. Int'l Test Conf.,* pp. 167-175, 1996.
[21] I. Hamzaoglu and J.H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," *Proc. Int'l Symp. Fault-Tolerant Computing,* pp. 260-267, 1999.
[22] J. Teuhola, "A Compression Method for Clustered Bit-Vectors," *Information Processing Letters,* vol. 7, pp. 308-311, Oct. 1978.
[23] I. Hamzaoglu and J.H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *Proc. Int'l Conf. Computer-Aided Design,* pp. 283-289, 1998.
[24] H. Kobayashi and L.R. Bahl, "Image Data Compression by Predictive Coding, Part I: Prediction Algorithm," *IBM J. Research and Development,* vol. 18, p. 164, 1974.
[25] J.A. Storer, *Data Compression: Methods and Theory.* New York: Computer Science Press, 1988.
[26] *Design Compiler Reference Manual.* Synopsys Inc., 1992.
[27] Univ. of Illinois IGATE website, www.crhc.uiuc.edu/IGATE, 2000.

Anshuman Chandra received the BE degree in electrical engineering from the University of Roorkee, Roorkee, India, in 1998, and the MS degree in electrical and computer engineering from Duke University, Durham, North Carolina, in 2000. He defended his PhD thesis in May 2002 and graduated with the PhD degree from Duke University in August 2002. His research interests include VLSI design, digital testing, and computer architecture. His PhD thesis research was on test set compression/decompression and embedded core testing. He is a student member of ACM SIGDA. He received the IEEE Test Technology Technical Council James Beausang Memorial Best Student Paper award for a paper in the *Proceedings of the 2000 IEEE VLSI Test Symposium*. He is a also a recipient of a best paper award at the 2001 Design, Automation, and Test in Europe (DATE) Conference. He is a student member fo the IEEE and the IEEE Computer Society.

Krishnendu Chakrabarty received the BTech degree from the Indian Institute of Technology, Kharagpur, in 1990, and the MSE and PhD degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively, all in computer science and engineering. He is now an associate professor of electrical and computer engineering at Duke University, Durham, North Carolina. He was also a Mercator Visiting Professor at the University of Potsdam in Germany from 2000 to 2002. He is a recipient of the US National Science Foundation Early Faculty (CAREER) award and the US Office of Naval Research Young Investigator award. His current research projects are in system-on-a-chip test, embedded real-time operating systems, distributed sensor networks, and the modeling, simulation, and optimization of microelectrofluidic systems. Dr Chakrabarty is a coauthor of two books: *Microelectrofluidic Systems: Modeling and Simulation* (CRC Press, 2002) and *Test Resource Partitioning for System-on-a-Chip* (Kluwer, 2002). He has published more than 120 papers in archival journals and refereed conference proceedings, and he holds a US patent in built-in self-test. He is a recipient of a best paper award at the 2001 Design, Automation and Test in Europe (DATE) Conference. Dr Chakrabarty is an associate editor of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, an editor of the *Journal of Electronic Testing: Theory and Applications (JETTA)*, and the guest editor of a special issue of *JETTA* on system-on-a-chip testing, scheduled for publication in August 2002. He was also a guest editor in 2001 of a special issue of the *Journal of the Franklin Institute* on distributed sensor networks. He is a member of the ACM and ACM SIGDA and a member of Sigma Xi. He serves as the 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 a senior member of the IEEE and a member of the IEEE Computer Society.