System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes

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

***Abstract*—We present a new test-data compression method and decompression architecture based on variable-to-variable-length Golomb codes. The proposed method is especially suitable for encoding precomputed test sets for embedded cores in a system-on-a-chip (SoC). The major advantages of Golomb coding of test data include very high compression, analytically predictable compression results, and a low-cost and scalable on-chip decoder. In addition, the novel interleaving decompression architecture allows multiple cores in an SoC to be tested concurrently using a single automatic test equipment input–output channel. We demonstrate the effectiveness of the proposed approach by applying it to the Internaional Symposium on Circuits and Systems' benchmark circuits and to two industrial production circuits. We also use analytical and experimental means to highlight the superiority of Golomb codes over run-length codes.**

***Index Terms*—Automatic test equipment (ATE), decompression architecture, difference vector, embedded core testing, precomputed test sets, test-set encoding, testing time, variable-to-variable-length codes.**

I. Introduction

CORE-BASED system-on-a-chip (SoC) designs present a number of test challenges [1]. These chips are composed of several reusable intellectual property (IP) cores that together integrate a wide range of functionality on a single die. The volume of test data for an SoC is growing rapidly as IP cores become more complex and an increasing number of these cores are being integrated in a chip. In order to effectively test these systems, each core must be adequately exercised with a set of precomputed test patterns provided by the core vendor (Fig. 1). However, the input–output (I/O) channel capacity, speed and accuracy, and data memory of automatic test equipment (ATE) are limited. Thus, it is becoming increasingly difficult to apply the enormous volume of test data to the SoC, which can be as high as 2.5 Gb for an industrial application-specific integrated circuit [2], without increasing testing time and test cost substantially.

The reduction in test-data volume will not only reduce ATE memory requirements, but also lower testing time. The testing time of an SoC depends on the test-data volume, the time required to transfer the data to the cores, the rate at which the test data is transferred (measured by the cores test-data bandwidth and ATE channel capacity), and the maximum scan chain length. The total test time can be reduced by either reducing the test-data volume or by shortening and reorganizing the scan chains. While test-data volume reduction techniques can be applied to both hard and soft cores, scan chains cannot be modified in hard cores. Lower testing time will increase production capacity as well as reduce test cost and time-to-market for SoCs. Therefore, new techniques are needed for decreasing test-data volume in order to overcome memory bottlenecks and to reduce testing time.

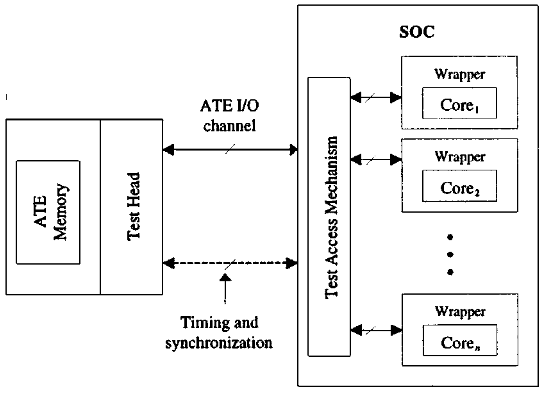


Fig. 1. Conceptual architecture for testing an SoC.

Built-in self test (BIST) has emerged as a useful approach for alleviating the above problems [3]. BIST reduces dependencies on expensive ATEs and it allows precomputed test sets to be embedded in test sequences generated by BIST hardware [4]–[6]. However, BIST can be applied directly to SoC designs only if the embedded cores are BIST-ready. Since most IP cores that are currently available from core vendors are not BIST ready, considerable redesign is necessary for incorporating BIST. This increases time-to-market and, therefore, defeats the very purpose of using IP cores.

Test-data compression offers a promising solution to the problem of reducing the test-data volume for SoCs, especially if the IP cores in the system are not BIST-ready [7]–[10], [13], [14]. In this approach, a precomputed test set T_D for an IP core is compressed (encoded) to a much smaller test set T_E, which is stored in ATE memory. An on-chip decoder is used for pattern decompression to obtain T_D from T_E during test application (Fig. 2).

Manuscript received August 1, 2000. This work was supported in part by the National Science Foundation under Grant CCR-9875324, by a contract from Delphi Delco Electronics Systems, and by an equipment grant from Sun Microsystems. This paper was presented in part at the VLSI Test Symposium, Montreal, QB, Canada, May 2000. This paper was recommended by Associate Editor R. Karri.

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

Publisher Item Identifier S 0278-0070(01)01508-1.

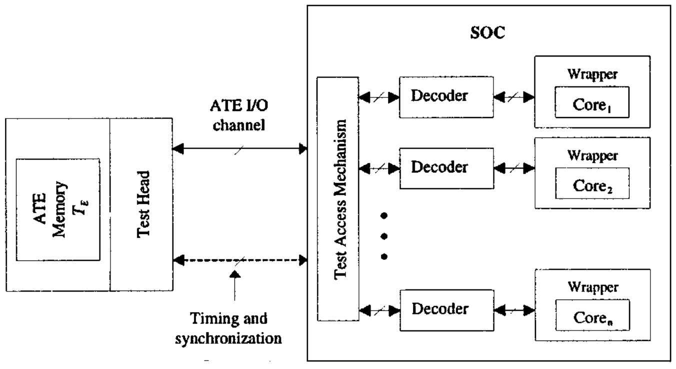


Fig. 2. Conceptual architecture for testing an SoC by storing the encoded test data T_E in ATE memory and decoding it using on-chip decoders.

Test-data compression using statistical coding of test sequences for synchronous sequential (nonscan) circuits was presented in [7] and [8]. Statistical coding was successfully applied to test sets for full-scan circuits in [9]. While the compression method in [7] and [8] is restricted to sequential circuits with a large number of flip flops and relatively few primary inputs, the work presented in [9] does not conclusively demonstrate that statistical coding provides greater compression than standard automatic test pattern generation (ATPG) compression methods for full-scan circuits [11], [12].

Test-data compression was also employed in [13] and [14] to reduce the time needed to download test patterns across a network to a user-interface workstation attached to an ATE. This method employs a combination of Burrows–Wheeler (BW) transformation and run-length coding. The encoding and decoding algorithm are implemented entirely in software. A hardware implementation of the BW decoder is prohibitively complex, thus other methods are required for efficient test-data compression and on-chip decompression.

An alternative approach to test-data compression is motivated by the fact that successive test patterns in a test sequence often differ in only a small number of bits. This was exploited in [10], where instead of compressing the test sequence T_D, a "difference vector" sequence $T_{\rm diff}$ determined from T_D was compressed using run-length coding. Since $T_{\rm diff}$ contains few ones, it can be efficiently compressed using a run-length code. A test architecture employing difference vectors and based on cyclical scan registers (CSRs) is sketched in Fig. 3. Note that existing registers on the SoC may be used as CSRs in order to reduce overhead [10].

A drawback of the compression method described in [10] is that it relies on variable-to-fixed-length codes, which are less efficient than more general variable-to-variable-length codes [15], [16]. Instead of using a run-length code with a fixed block size b, we can achieve greater compression by using Golomb codes that map variable-length runs of zeros in a difference vector to variable-length codewords [15]. Golomb codes have been studied extensively for image processing and data compression [17], [18]. These codes are provably optimal (satisfy the entropy bound) if the run lengths in the data stream are geometrically distributed [15]. Even if this assumption is not satisfied, Golomb codes provide a high degree of compression.

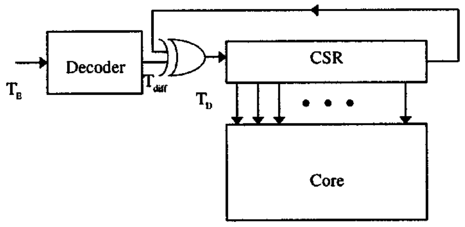


Fig. 3. Decompression architecture based on a CSR.

In this paper, we present a new test-data compression and decompression method based on Golomb codes for testing SoCs using precomputed test sets. The proposed method is applicable to both full-scan and nonscan circuits. Since a number of legacy cores do not use full scan, a practical encoding method should be applicable to both classes of designs. For full-scan circuits, the test patterns in a precomputed test set T_D can be reordered to obtain a difference vector with very few ones. For nonscan circuits, however, the order of pattern application must be preserved; therefore, no reordering of T_D is possible. Nevertheless, we show that Golomb coding is effective for encoding T_D for these circuits. An encoded test set T_E derived using Golomb coding is considerably smaller than the original precomputed test set T_D. Furthermore, we show that T_E is also much smaller than the smallest test sets that have been derived for the Internaional Symposium on Circuits and Systems (ISCAS) benchmark circuits using ATPG compaction.

The main contributions of this paper are summarized as follows.

1) We apply variable-to-variable-length Golomb codes to the problem of compressing test data for SoCs. This saves ATE memory and significantly reduces the testing time.
2) We present a decompression architecture that allows multiple cores to be tested in parallel without requiring additional ATE I/O channels. This benefit is a direct consequence of the structure of the Golomb code.
3) We derive upper and lower bounds on the amount of compression that can be achieved for any given $T_{\rm diff}$. We also derive similar bounds on run-length codes. These simple bounds provide useful guidelines to the designer on whether Golomb codes are suitable for a given problem instance. Moreover, these bounds also reveal the inherent superiority of Golomb codes over run-length codes.
4) We provide experimental results for the ISCAS benchmark circuits and two real industrial designs. For the full-scan ISCAS'89 benchmark circuits, we show that Golomb codes lead to compressed test sets that are significantly smaller than the smallest known test sets for the circuits derived using ATPG compaction.
5) We design a low-cost decoder for decompressing Golomb-encoded test patterns. We implement the decoder using Synopsys design compiler [20] and show that overhead due to the decoder is very small. In addition, the decoder is scalable and independent of the core under test and the precomputed test set T_D.
6) We show that test-data compression not only reduces the volume of test data but it also allows a slower tester to

be used without any penalty on testing time. The Semiconductor Industry Association National Technology Roadmap predicts that the cost of high-speed testers will exceed \$20 million by 2010. While IC speeds have improved at the rate 30% per year, tester accuracy has improved at an annual rate of only 12%. Hence, test methods that can be used with slower low-cost testers are becoming especially important [24].

The organization of the paper is as follows. In Section II, we present the basic concept of Golomb coding. We derive bounds on the amount of compression that can be achieved using Golomb and run-length codes. Section III presents the encoding procedures for full-scan and nonscan circuits. It also describes the decoder that is necessary for on-chip decompression. Section IV presents the overall test architecture and a decompression method for an SoC with multiple cores. Experimental results are reported in Section V, and conclusions are described in Section VI.

II. Golomb Coding

In this section, we describe Golomb coding and analyze its effectiveness for test-data compression. For any give sequence of difference vectors, we derive tight upper and lower bounds on the amount of compression that can be obtained with Golomb codes. We also derive similar bounds for conventional run-length coding in order to highlight the inherent superiority of Golomb codes.

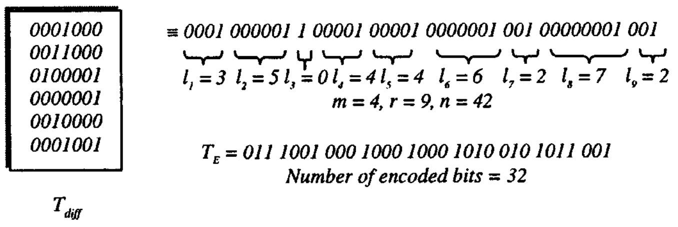
As discussed in Section I, the first step in encoding a test set T_D is to generate its difference vector test set T_{diff}. Let the (ordered) precomputed test set be $T_D = \{t_1, t_2, t_3, \ldots, t_n\}$. Its difference vector is then given by $T_{\text{diff}} = \{t_1, t_1 \oplus t_2, t_2 \oplus t_3, \ldots, t_{n-1} \oplus t_n\}$. This assumes that the CSR starts in the all-zero state. Other starting states can be considered similarly.

The next step in the encoding procedure is to select the Golomb code parameter m, referred to as the group size. The choice of m has received a lot of attention in the information theory literature—for certain distributions of the input data stream (T_{diff} in our case), the group size m can be optimally determined. For example, if the input data stream is random with zero probability p, then m should be chosen such that $p^m \approx 0.5$ [16]. However, since the difference vectors for precomputed test sets do not satisfy the randomness assumption, the best value of m for test-data compression must be determined experimentally. Nevertheless, we show later that the best value of m can be approximated analytically.

Once the group size m is determined, the runs of zeros in T_{diff} are mapped to groups of size m (each group corresponding to a run length). The number of such groups is determined by the length of the longest run of zeros in T_{diff}. The set of run lengths $\{0, 1, 2, \ldots, m-1\}$ forms group A_1; the set $\{m, m+1, m+2, \ldots, 2m-1\}$, group A_2; etc. In general, the set of run lengths $\{(k-1)m, (k-1)m+1, (k-1)m+2, \ldots, km-1\}$ comprises group A_k [16]. To each group A_k, we assign a group prefix of $(k-1)$ ones followed by a zero. We denote this by $1^{(k-1)}0$. If m is chosen to be a power of two, i.e., $m = 2^N$, each group contains 2^N members and a $\log_2 m$-bit sequence (tail) uniquely

Group	Run-length	Group prefix	Tail	Codeword
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
...	...	...	...	...

Fig. 4. Example of Golomb coding for $m = 4$.


0001000
0011000
0100001
0000001
0010000
0001001
T_{diff}
= 0001 000001 1 00001 00001 0000001 001 00000001 001
$l_1 = 3$ $l_2 = 5$ $l_3 = 0$ $l_4 = 4$ $l_5 = 4$ $l_6 = 6$ $l_7 = 2$ $l_8 = 7$ $l_9 = 2$
$m = 4, r = 9, n = 42$
T_E = 011 1001 000 1000 1000 1010 010 1011 001
Number of encoded bits = 32

(a) (b)

Fig. 5. (a) Difference vector test set T_{diff} and (b) its encoded test data T_E.

identifies each member within the group. Thus, the final code word for a run length L that belongs to group A_k is composed of two parts—a group prefix and a tail. The prefix is $1^{(k-1)}0$ and the tail is a sequence of $\log_2 m$ bits. It can be easily shown that for a run of length $l, k = \lfloor l/m \rfloor + 1$. The encoding process is illustrated in Fig. 4 for $m = 4$.

We now analyze the effectiveness of Golomb coding for a given difference vector sequence T_{diff}. We derive upper and lower bounds on $|T_E|$ for any given $m = 2^N$. The patterns in T_{diff} can be considered as a single stream of data as shown in Fig. 5. Let there be n b and r ones in T_{diff}. Also, without loss of generality, let the sequence always end with a one. Therefore, T_{diff} will contain r runs of zeros. Let these runs be of length $l_1, l_2, l_3, \ldots, l_r$, respectively. Thus, T_{diff} can be represented by the sequence $l_1 1 l_2 1 l_3 1 \ldots l_r 1$ such that $(l_1 + l_2 + l_3 + \cdots + l_r) + r = n$. This implies that

$$\sum_{i=1}^{r} l_i = n - r \tag{1}$$

and the number of bits G in the encoded sequence T_E is given by

$$\begin{aligned} G &= \sum_{i=1}^{r} \left(1 + \left\lfloor \frac{l_i}{m} \right\rfloor + \log_2 m\right) \\ &= r + r\log_2 m + \sum_{i=1}^{r} \left\lfloor \frac{l_i}{m} \right\rfloor. \end{aligned} \tag{2}$$

The following theorem provides upper and lower bounds on G, the size of the encoded sequence T_E.

Theorem 1: Let the total number of bits in the difference vector set T_{diff} be n and the total number of ones be r. Then, the size G of the encoded test data T_E is bounded as follows:

$$\frac{n}{m} + r\log_2 m \le G \le \frac{n}{m} + r\log_2 m + r\left(1 - \frac{1}{m}\right).$$

Proof: Let $l_i = Q_i m + R_i$, where l_i is the ith run of zeros in T_{diff} and Q_i (R_i) is the quotient (remainder) when l_i is divided by m. From (1), we get

$$\begin{aligned}\sum_{i=1}^{r} l_i &= \sum_{i=1}^{r}(Q_i m + R_i)\\ &= n - r. \end{aligned}\tag{3}$$

Moreover, by substituting l_i with $Q_i m + R_i$ in (2), we get

$$G = r + r\log_2 m + \sum_{i=1}^{r} Q_i. \tag{4}$$

We first derive a lower bound $G_{\min}$ on G. It follows from (4) that in order to maximize compression, $\sum_{i=1}^{r} Q_i$ must be minimized. Within each group of size m, maximum compression is obtained if $l_i = m - 1 \pmod m$. Our objective here is to minimize the number of run lengths that are factors of m. We note that

$$\begin{aligned}\sum_{i=1}^{r}\left\lfloor \frac{l_i}{m}\right\rfloor &= \sum_{i=1}^{r} Q_i\\ &= \frac{n-r}{m} - \frac{1}{m}\sum_{i=1}^{r} R_i. \end{aligned}\tag{5}$$

For any run length l_i, the maximum value of remainder R_i can be $m-1$. Therefore

$$\begin{aligned}\sum_{i=1}^{r} Q_i &\ge \frac{n-r}{m} - \frac{(m-1)r}{m}\\ &= \frac{n}{m} - r. \end{aligned}\tag{6}$$

Substituting (6) in (5) and using (2), we get

$$\begin{aligned}G_{\min} &= r + r\log_2 m + \frac{n}{m} - rf\\ &= \frac{n}{m} + r\log_2 m. \end{aligned}$$

We next prove the upper bound result. In order to derive an upper bound $G_{\max}$ on G, we need to maximize $\sum_{i=1}^{r} Q_i$. For any run length l_i, the minimum value of R_i can be zero. Combining this with (5), we get

$$\sum_{i=1}^{r} Q_i \le \frac{n-r}{m}. \tag{7}$$

Substituting (7) in (5) and using (2), we get.

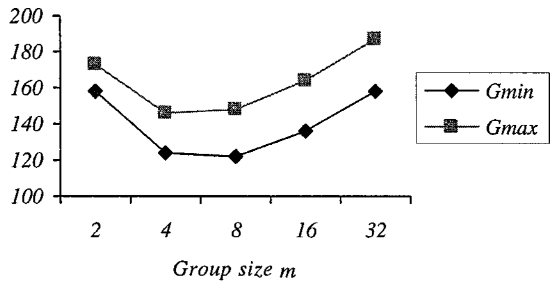
$$\begin{aligned}G_{\max} &= r + r\log_2 m + \frac{n-r}{m}\\ &= \frac{n}{m} + r\log_2 m + r\left(1 - \frac{1}{m}\right). \end{aligned}$$

This completes the proof of the theorem. □

The following corollary shows that Theorem 1 provides tight bounds on G, especially if the number of ones in T_{diff} is small. The proof of the corollary follows from Theorem 1.

Group size m	G_{min}	G_{max}
2	158	173
4	124	146
8	122	148
16	136	164
32	158	187

(a)



(b)

Fig. 6. Example illustrating the variation of the lower and upper bounds with m for n =256 and r =30. (a) Values of the bounds. (b) Plot of the bounds.

Corollary 1: Consider any difference vector set T_{diff} with r ones. Let $G_{\max}(G_{\min})$ be the upper (lower) bound on the size of the encoded test set T_E, as predicted by Theorem 1. The difference between $G_{\max}$ and $G_{\min}$ is bounded as follows:

$$\frac{r}{2} \le G_{\max} - G_{\min} < r.$$

The above corollary illustrates an interesting property of Golomb codes, namely, if the number of ones in T_{diff} is small, Golomb coding provides almost the same amount of compression for different n-bit sequences with r ones. The value of G lies between the values of $G_{\max}$ and $G_{\min}$ derived above and this variation can be at most r.

As an illustration of these bounds, consider a hypothetical example, where $n = 256$ and $r = 30$. The upper and lower bounds for various values of m are shown in Fig. 6(a) and the corresponding graph is plotted in Fig. 6(b). We note that the lower and upper bound on the compression G follows a "bathtub curve" and the best value of m depends on T_{diff}. Also, according to Corollary 1, the difference between $G_{\max}$ and $G_{\min}$ is smallest for $m = 2$ and increases as m increases. These bounds are obtained from the parameters n and r and they do not depend on the distribution of ones in T_{diff}. They can therefore be used as predictors for the effectiveness of Golomb coding for a particular T_D.

We now show how the best code parameter m can also be obtained analytically. This approach yields a value for $m = m_a$ that must be rounded off to the nearest power of two. From (2), we get

$$\begin{aligned}G &= r + r\log_2 m + \sum_{i=1}^{r}\left\lfloor\frac{l_i}{m}\right\rfloor\\ &\approx r + r\log_2 m + \frac{n-r}{r}. \end{aligned}\tag{8}$$

Differentiating (8) with respect to m and equating to zero, we get

$$\frac{r}{m\ln 2} - \frac{n-r}{m^2} = 0$$

which yields $m_a = (0.693(n-r))/(r)$. It can be easily seen that as long as r is sufficiently small compared to $n, (d^2G)/(dm^2) > 0$ for $m = m_a$; hence, m_a provides the best compression. We show in Section V that m_a and the best value of m determined experimentally are very close for all benchmark circuits.

We next derive upper and lower bounds on the compression achieved by run-length coding.

Theorem 2: Let the total number of bits in test set T_{diff} be n and the total number of ones be r. In addition, suppose block size b is used for run-length coding. The size RL of the encoded test data T_E is given by

$$\frac{bn}{2^b-1} \le \text{RL} \le \frac{bn}{2^b-1} + \frac{br(2^b-2)}{2^b-1}$$
$$\approx \frac{bn}{2^b-1} + br \quad \text{for sufficiently large } b.$$

Proof: The total number of compressed bits in a run-length coded (block size b) sequence is given by

$$\text{RL} = \sum_{i=1}^{r} \left\lceil \frac{l_i+1}{2^b-1} \right\rceil b$$
$$= \sum_{i=1}^{r} \left(\frac{l_i+1}{2^b-1} + \delta_i \right) b \quad \text{where, } 0 \le \delta_i \le \frac{2^b-2}{2^b-1}$$
$$= \frac{b}{2^b-1} \sum_{i=1}^{r} l_i + \frac{br}{2^b-1} + b \sum_{i=1}^{r} \delta_i.$$

Since $\sum_{i=1}^{r} l_i = n - r$, we get

$$\text{RL} = \frac{b(n-r)}{2^b-1} + \frac{br}{2^b-1} + b \sum_{i=1}^{r} \delta_i.$$

Therefore, a lower bound RL is given by

$$\text{RL}_{\min} = \frac{bn}{2^b-1} \quad \text{which occurs for } \delta_i = 0 \text{ for all } i.$$

Similarly, an upper bound on RL is given by

$$\text{RL}_{\max} = \frac{bn}{2^b-1} + \frac{br(2^b-2)}{2^b-1}$$
$$\text{which occurs for } \delta_i = \tfrac{2^b-2}{2^b-1} \text{ for all } i.$$

This completes the proof of the theorem. □

We can now compare the efficiency of Golomb coding ($m = 4$) and run-length coding for block size $b = 3$. For run-length coding, a lower bound from Theorem 2 is given by

$$\text{RL}_{\min} = \frac{bn}{2^b-1} = \frac{3n}{7} = 0.428n.$$

Now, an upper bound for Golomb coding from Theorem 1 is given by

$$G_{\max} = \frac{n}{m} + r\log_2 m + r\left(1 - \frac{1}{m} = \right) = \frac{n}{4} + \frac{11r}{4}.$$

If we make the realistic assumption (based on experimental data) that $r \le 0.05n$, we get $G_{\max} = 0.39n$, which is smaller than $\text{RL}_{\min}$. In fact, as r becomes smaller relative to n, $G_{\max} \to 0.25n$. Therefore, we note that as long as r is sufficiently small compared to n, the compression that can be achieved with run-length coding is less than the worst compression with Golomb coding. This provides an analytical justification for the use of Golomb codes instead of run-length codes.

III. Test-Data Compression/Decompression

In this section, we describe the test-data compression procedure, the decompression architecture, and the design of the on-chip decoder. Additional practical issues related to the decompression architecture are discussed in the following section. 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 Golomb codes was outlined in Section II. Let T_D be the test set with p patterns and n primary inputs and T_{diff} be the corresponding difference vector test set. The procedure shown below is used to obtain T_{diff} and the encoded test set T_E.

```
$Code\_procedure(p,n,m)$
begin
  $T_D \cdot Read\_pat(p,n)$            ▷ read the test set
  $T_{\text{diff}} \cdot Addvec(1) = t_1$         ▷ add first pattern to $T_{\text{diff}}$
  $i = 2$ to $p$
    $T_D \cdot Reorder(i,p,n)$          ▷ reorder patterns according to weights
    $T_{\text{diff}} \cdot Addvec(i) = t_i \oplus t_{i-1}$
  $T_{\text{diff}} \cdot Golomb\_code(m)$       ▷ encode $T_{\text{diff}}$ with group size $m$
end
$Reorder(i,p,n)$
begin
  $j = (i+1)$ to $p$                ▷ this loop picks the pattern with largest weight
    If $t_j \cdot pat\_wt(n) \ge t_i \cdot pat\_wt(n)$ ▷ $pat\_wt(n)$ calculates weight for a pattern w.r.t. pattern $(i-1)$
    $Swap(t_j, t_i)$
end
```

A straightforward algorithm is used for generating T_{diff}. For full-scan cores, reordering of the test patterns is allowed; therefore, the patterns can be arranged such that the runs of zeros are long in T_{diff}. The problem of determining the best ordering is equivalent to the NP-Complete Traveling Salesman problem. Therefore, a greedy algorithm is used to generate T_{diff}. Let every pattern in T_D correspond to a node in a complete directed graph G and let the number of zeros in the difference vector obtained from $t_i \oplus t_j$ be defined as the weight (w_{ij}) of the edge from t_i to t_j. Starting from the first pattern t_1, we choose the next pattern that is at the least distance from t_1. (The distance between two nodes is given by $n - w_{ij}$.) We continue this process until all the patterns are covered, i.e., all nodes in G are visited. The procedure $Reorder(i,p,n)$ picks the test pattern with the largest weight and reorders the test set when repeatedly called by $Code\_procedure(p,n,m)$. The procedure $Addvec(i)$ generates T_{diff} by adding the test pattern returned by $Reorder(i,p,n)$. Once T_{diff} is generated, the procedure $Golomb\_code(m)$ generates the encoded test set T_E for the specified m. The same proce-

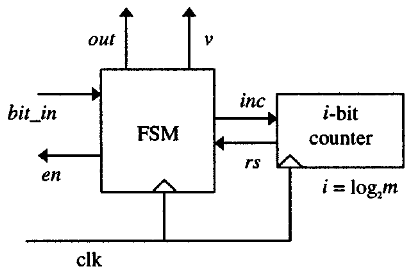


Fig. 7. Block diagram of the decoder used for decompression.

dure can be used to generate T_E for nonscan cores by removing the procedure $Reorder(i, p, n)$. For test cubes, the don't-cares have to be mapped to zeros or ones before they can be compressed. The don't-cares are therefore assigned binary values such that w_{ij} is maximum for the edge between t_i and t_j.

A. Pattern Decompression

The decoder decompresses the encoded test set T_E and outputs T_{diff}. The exclusive-or gate and the CSR are used to generate the test patterns from the difference vectors. Since the decoder for Golomb coding needs to communicate with the tester, proper synchronization must be ensured through careful design. Compared to run-length coding, the synchronization mechanism for Golomb coding is more involved since both the codewords and the decompressed data can be of variable length. For run-length coding, the codewords are of fixed length; nevertheless, a run-length decoder must also communicate with the tester to signal the end of a block of variable-length decompressed data.

The Golomb decoder can be efficiently implemented by a $\log_2 m$-bit counter and a finite-state machine (FSM). The block diagram of the decoder is shown in Fig. 7. The *bit_in* is the input to the FSM and an enable (*en*) signal is used to input the bit whenever the decoder is ready. The signal *inc* is used to increment the counter and *rs* indicates that the counter has finished counting. The signal *out* is the decode output and v indicates when the output is valid. The operation of the decoder is as follows.

1) Whenever the input is one, the counter counts up to m. The signal *en* is low while the counter is busy counting and enables the input at the end of m cycles to accept another bit. The decoder outputs m zeros during this operation and makes the valid signal v high.
2) When the input is zero, the FSM starts decoding the tail of the input codeword. Depending on the tail bits, the number of zeros outputted is different. The *en* and v signals are used to synchronize the input and output operation of the decoder.

The state diagram corresponding to the decoder for $m = 4$ is shown in Fig. 8. The states $S0$–$S3$ and $S4$–$S8$ correspond to the prefix and tail decoding, respectively. We simulated the decoder using very high-speed integrated-circuit hardware description language (VHDL) and Synopsys tools to ensure its correct operation. We also synthesized the FSM using Synopsys design compiler to access the hardware overhead of the decoder. The synthesized circuit is shown in Fig. 9. It contains only four flip flops and 34 combinational gates. For any circuit whose test set is compressed using $m = 4$, the logic shown in the gate level schematic is the only additional hardware required other than the $\log_2 m$-bit counter. Thus, the decoder is independent of not only the core under test, but also its precomputed test set. The extra logic required for decompression is very small and can be implemented very easily. This is in contrast to the run-length decoder, which is not scalable and becomes increasingly complex for higher values of the block length b.

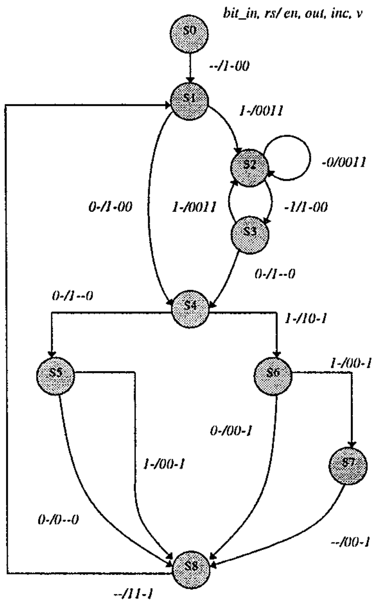


Fig. 8. Decode FSM diagram.

B. Analysis of Test Application Time and Test-Data Compression

We now analyze the testing time for a single scan chain when Golomb coding is employed with the test architecture shown in Fig. 3. From the state diagram of the Golomb decoder, we note that:

1) each "1" in the prefix part takes m cycles for decoding;
2) each separator "0" takes one cycle;
3) the tail part takes a maximum of m cycles and a minimum of $\gamma = \log_2 m + 1$ cycles.

Let n_c be the total number of bits in T_E and r be the number of ones in T_{diff}. T_E contains r tail parts, r separator zeros, and the number of prefix ones in T_E equals $n_c - r(1 + \log_2 m)$. Therefore, the maximum and minimum testing times ($\mathcal{T}_{\max}$ and $\mathcal{T}_{\min}$, respectively) measured by the number of cycles are given by

$$\begin{aligned}\mathcal{T}_{\max} &= (n_c - r(1+\log_2 m))m + r + mr\\ &= mn_c - r(m\log_2 m - 1)\\ \mathcal{T}_{\min} &= (n_c - r(1+\log_2 m))m + r + \gamma r\\ &= mn_c - rm(1+\log_2 m) - (1+\gamma).\end{aligned}$$

Therefore, the difference between $\mathcal{T}_{\max}$ and $\mathcal{T}_{\min}$ is given by

$$\begin{aligned}\delta\mathcal{T} &= \mathcal{T}_{\max} - \mathcal{T}_{\min}\\ &= r(m - \log_2 m - 1).\end{aligned}$$

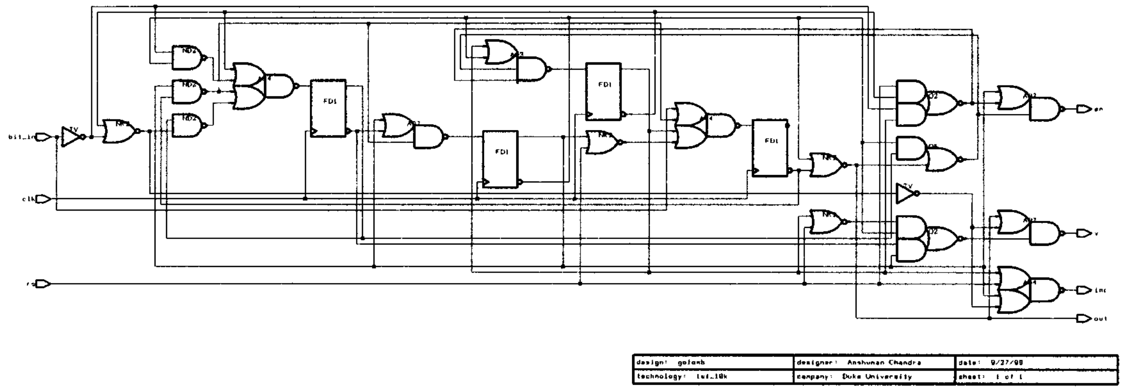


Fig. 9. Gate-level schematic of the decode FSM generated using Synopsys design compiler.

A major advantage of Golomb coding is that on-chip decoding can be carried out at scan clock frequency f_{scan} while T_E can be fed to the core under test with external clock frequency $f_{\text{ext}} < f_{\text{scan}}$. This allows us to use slower testers without increasing the test application time. The external clock and scan clocks must be synchronized, e.g., using the scheme described in [24], [25], and $f_{\text{scan}} = mf_{\text{ext}}$, where the Golomb code parameter m is usually a power of two. This allows the bits of T_{diff} to be generated by the decoder at the frequency of f_{scan}. We now present an analysis of testing time using $f_{\text{scan}} = mf_{\text{ext}}$ and compare the testing time for our method with that of external testing in which ATPG-compacted patterns are applied using an external tester.

Let the ATPG-compacted test set contain p patterns and let the length of the scan chain be n bits. Therefore, the size of the ATPG-compacted test set is pn bits and the testing time $\mathcal{T}_{\text{ATPG}}$ equals pn external clock cycles. Next, suppose the difference vector T_{diff} obtained from the uncompacted test set contains r ones and its Golomb-coded test set T_E contains n_c bits. Therefore, the maximum number of scan clock cycles required for applying the test patterns using the Golomb coding scheme is $\mathcal{T}_{\max} = mn_c - r(m\log_2 m - 1)$.

Now, the maximum testing time τ (seconds) when Golomb coding is used is given by

$$\begin{aligned}\tau &= \frac{\mathcal{T}_{\max}}{f_{\text{scan}}}\\ &= \frac{mn_c - r(m\log_2 m - 1)}{f_{\text{scan}}}\end{aligned}$$

and the testing time τ' (seconds) for external testing with ATPG-compacted patterns is given by

$$\begin{aligned}\tau &= \frac{pn}{f_{\text{ext}}}\\ &= \frac{pnm}{f_{\text{scan}}}.\end{aligned}$$

If testing is to be accomplished in $\tau^\star$ seconds using Golomb coding, the scan clock frequency f_{scan} must equal $\mathcal{T}_{\max}/\tau^\star$, i.e.

$$f_{\text{scan}} = \frac{mn_c - r(m\log_2 m - 1)}{\tau^\star}.$$

This is achieved using a slow external tester operating at frequency $f_{\text{ext}} = f_{\text{scan}}/m$. On the other hand, if only external test is used with the p ATPG-compacted patterns, the required external tester clock frequency f'_{ext} equals $pn/\tau^\star$. Let us take the ratio between f'_{ext} and f_{ext}

$$\begin{aligned}\frac{f'_{\text{ext}}}{f_{\text{ext}}} &= \frac{pn/\tau^\star}{f_{\text{scan}}/m}\\ &= \frac{pn}{n_c - r\log_2 m + r/m}.\end{aligned}$$

Experimental results presented in Section V show that in all cases, the ratio is greater than one, therefore demonstrating that the use of Golomb coding allows us to decrease the volume of test data and use a slower tester without increasing testing time.

IV. Decompression Architecture

In this section, we present a decompression architecture for testing SoC designs when Golomb coding is used for test-data compression. We describe the application of Golomb codes to nonscan and full-scan circuits and we present a new technique for testing several cores simultaneously using a single ATE I/O channel.

A. Application to Sequential (Nonscan) Cores

For sequential cores, a boundary scan register is required at the functional inputs for decompression. This register is usually available for cores that are wrapped. In addition, a two-input exclusive-or gate is required to translate the difference vectors to the patterns of T_D. Fig. 10(a) shows the overall test architecture for the sequential core. 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.

B. Application to Full-Scan Cores

Most cores in use today contain one or more internal scan chains. However, since the scan chains are used for capturing test responses, they cannot be used for decompression. An additional CSR with length equal to the length of the internal scan

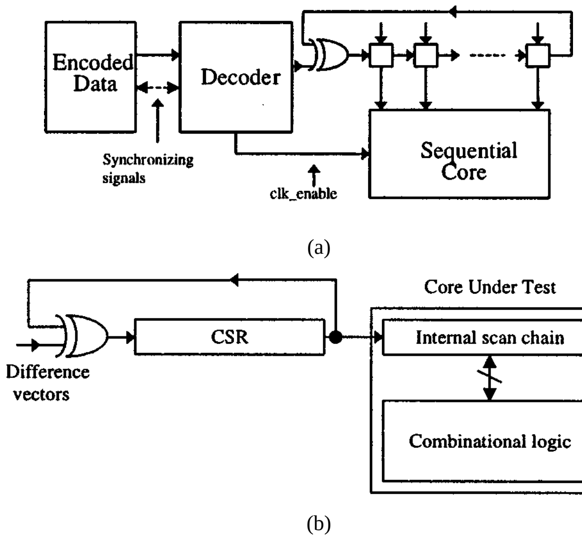


Fig. 10. (a) Decompression architecture with the boundary scan register being used for generating test patterns and applying them to the sequential core. (b) CSR being used to feed the internal scan chain of the core.

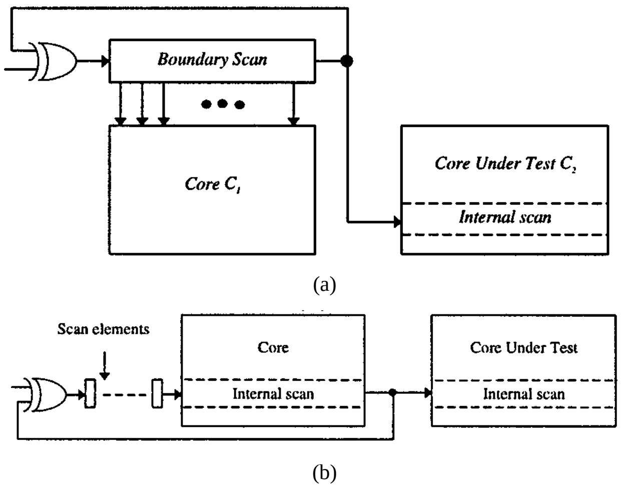


Fig. 11. (a) Configuring the boundary scan register as a CSR for a core. (b) Using internal scan of a core and extra scan elements to form CSR for the core-under-test [10].

chain is required to generate the test patterns. Fig. 10(b) shows the decompression architecture for full-scan cores.

As discussed in [10], there are a number of ways in which the various scan chains in an SoC can be configured to test the cores in the system. If an SoC contains both nonscan and full-scan cores, the boundary-scan register associated with a nonscan core $C1$ can be first used to decompress and apply test patterns to $C1$, and then used to decompress the test patterns and feed the internal scan chain of a full-scan core $C2$ [see Fig. 11(a)]. Similarly, as shown in Fig. 11(b), the internal scan of a core can be used to decompress and feed the test patterns to the internal scan of the core under test if the length of the internal scan chain being used for decompression is smaller than or equal to the internal scan chain being fed. In case the chain length is smaller, extra scan elements can be added to make the lengths of the two scan chains equal. In this way, the proposed scheme provides the designer with flexibility in configuring the various scan chains to minimize hardware overhead.

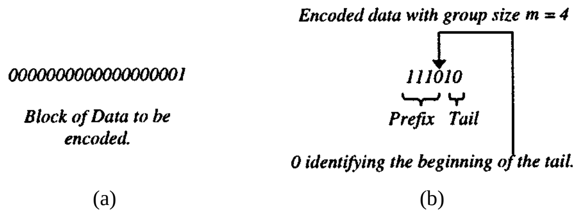

Encoded data with group size $m = 4$
000000000000000001
Block of Data to be encoded.
111010
Prefix
Tail
0 identifying the beginning of the tail.
(a)
(b)

Fig. 12. (a) Run of 14 zeros and (b) its encoded code word.

C. Application to Multiple Cores

We now highlight another important advantage of Golomb coding. In addition to reducing testing time and the size of the test data to be stored in the ATE memory, Golomb coding also allows multiple cores to be tested simultaneously using a single ATE I/O channel. In this way, the I/O channel capacity of the ATE can be increased. This is a direct consequence of the structure of the Golomb coding and such a design is not possible for variable-to-fixed-length (run-length) coding.

As discussed in Section II, when Golomb coding is applied to a block of data containing a run of zeros followed by a single one, the code word contains two parts—a prefix and tail. For a given code parameter m (group size), the length of the tail $(\log_2 m)$ is independent of the run length. Note further that every one in the prefix corresponds to m zeros in the decoded difference vector. Thus, the prefix consists of a string of ones followed by a zero and the zero can be used to identify the beginning of the tail. For example, Fig. 12 shows a run of 14 zeros encoded by a 4-bit prefix and a 2-bit tail.

As shown in Section III, the FSM in the decoder runs the counter for m decode cycles whenever a one is received and starts decoding the tail as soon as a zero is received. The tail decoding takes at most m cycles. During prefix decoding, the FSM has to wait for m cycles before the next bit of the prefix can be decoded. Therefore, we can use interleaving to test m cores together such that the decoder corresponding to each core is fed with encoded prefix data after every m cycles. This can also be used to feed multiple scan chains of a core in parallel as long as the capture cycles of the scan chains are synchronized, for example, by using the same functional clock. For the interleaving to be applicable, the scan chains must be of the same length and the same value of m must be used for encoding each set of scan data. A separate decoder is necessary for each scan chain.

Whenever the tail is to be decoded (identified by a zero in the encoded bit stream), the respective decoder is fed with the entire tail of $\log_2 m$ bits in a single burst of $\log_2 m$ cycles. This interleaving scheme is based on the use of a demultiplexer as shown in Fig. 13. The method works as follows. First, the encoded test data for m cores is combined to generate a composite bit stream T_C that is stored in the ATE. Next, T_C is fed to the demultiplexer and a small FSM with only $i = \log_2 m$ states is used to detect beginning of each tail. An i-bit counter is used to select the outputs to the decoders of the various cores. The only restriction that we impose for now is that the compression of test data corresponding to each core has to be done using the same group size m. This restriction will be removed in the following paragraphs.

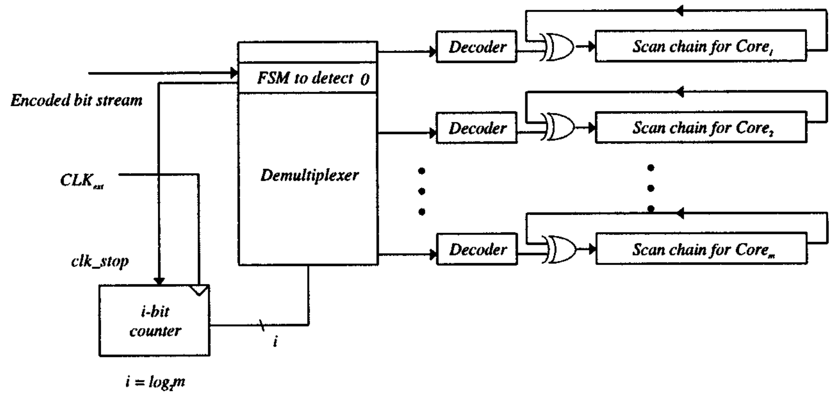


Fig. 13. Test-set decompression and application to multiple cores.

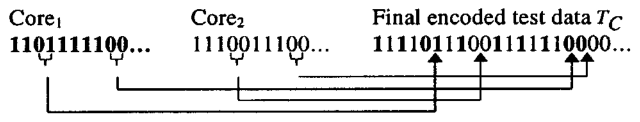


Fig. 14. Composite encoded test data for two cores with group size m =2.

Now we outline how T_C is generated from the different encoded test data. T_C is obtained by interleaving the prefix parts of the compressed test sets of each core, but the tails are included unchanged in T_C. An example is shown in Fig. 14 where compressed data for two cores (generated using group size $m = 2$) have been interleaved to obtain the final encoded test set to be applied through the decompression scheme for multiple cores.

Every scan chain has its dedicated decoder. This decoder receives either a one or the tail of the compressed data corresponding to the various cores connected to the scan chain. The i-bit counter connected to the select lines of the demultiplexer selects a decoder after every m clock cycles. If the FSM detects that a portion of the tail has arrived, the zero that is used to identify the tail is passed to the decoder and then the counter is stopped for $\log_2 m$ (tail length) cycles so that the test data is transferred continuously to the appropriate core.

The tail decoding takes at most m cycles. This is because the number of states traversed by the decode FSM depends on the bits of T_E that it receives; see Fig. 8. This number can be at most m. In order to make the prefix and tail decoding cycles equal, three additional states must be added to the FSM state diagram as shown in Fig. 15. This ensures that the decoder works in synchronization with the demultiplexer. Moreover, now the tail bits may not be passed on to the decoder as a single block. Thus, the interleaving of test data to generate T_C changes slightly. The additional states do not increase the number of flip flops in the decoder.

Consider a simple case where m cores are tested simultaneously using the above decompression scheme. Let p_i be the number of patterns and n_i be the scan length for the ith core. Also, without loss of generality, let $p_1 \leq p_2 \leq p_3 \ldots \leq p_m$ and let $n_1 \leq n_2 \leq n_3 \ldots \leq n_m$. The total testing time T for this system is given by

$$T = \max\{p_1 n_1, p_2 n_2, p_3 n_3, \ldots, p_m n_m\} + \max\{p_1, p_2, p_3, \ldots, p_m\} = p_m n_m + p_m.$$

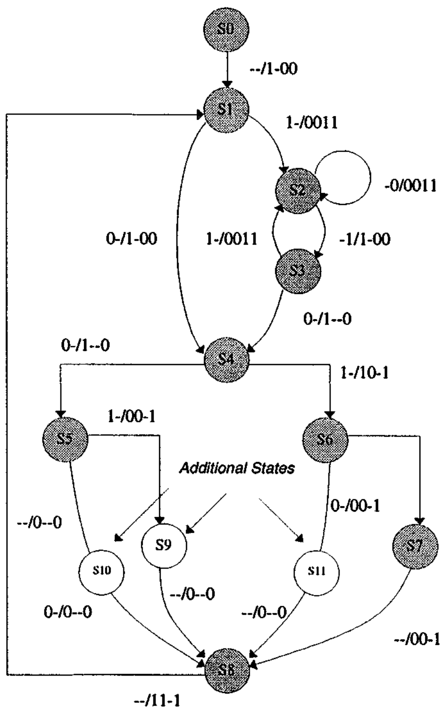


Fig. 15. Modified state diagram of the decode FSM to make the tail and prefix decode cycles equal.

An intuitive interpretation of this is that T will equal the test time of the core with the largest amount of test data. Since all cores do not have the same test-data volume, the proposed decompression scheme can be more efficiently employed by assigning multiple cores to the same system scan chain such that the volume of test data to be fed to the different scan chains are nearly equal (Fig. 16). Even though this increases the lengths of the scan chains in the SoC, it offers the advantage of reducing overhead due to the decoders without increasing system testing time. The encoding procedure now works as follows: the test set for the cores connected to the same scan chain are merged and then encoded. This encoded data is then used to obtain the composite test data T_C as described above.

The test sets for the cores on the different scan chains are compressed more efficiently if the group size m is allowed to vary. Therefore, to derive the maximum benefit of Golomb codes for each core, multiple cores are grouped together if their test sets

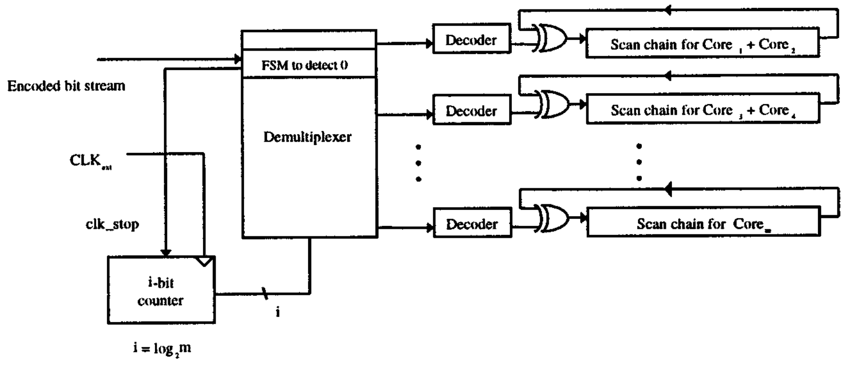


Fig. 16. Decompression architecture for multiple cores assigned to same scan chain.

TABLE I
EXPERIMENTAL RESULTS ON GOLOMB CODING FOR THE COMBINATIONAL AND FULL-SCAN ISCAS BENCHMARK CIRCUITS WITH TEST PATTERNS GENERATED USING MINTEST [11]

Circuit	Percentage compression for various values of m					No of bits in T_D	Best compression (percent)	No of bits G in T_E	No of bits for Mintest
	$m = 2$	$m = 4$	$m = 8$	$m = 16$	$m = 32$				
c1355	34.86	**45.70**	44.58	37.63	28.00	4838	45.70	2627	3444
c1908	30.17	**37.30**	32.63	21.93	8.06	4587	37.30	2876	3498
c2670	38.64	53.34	**56.08**	53.02	47.08	20271	56.08	8903	10252
c3450	23.48	**24.52**	13.90	–2.40	–21.36	6450	24.52	4868	4200
c5315	31.33	**39.02**	35.08	25.46	13.26	15486	39.02	9443	6586
c7552	**15.50**	9.80	-6.99	–29.51	–54.17	25254	15.50	21338	15111
s641	**21.79**	21.58	10.32	–6.83	–26.63	1404	21.79	1098	1134
s713	22.43	**23.07**	11.96	–4.70	–24.07	1404	23.07	1080	1134
s1196	32.80	**42.22**	40.06	31.67	20.21	4448	42.22	2570	3616
s1238	34.21	**44.79**	43.62	36.26	25.94	4864	44.79	2685	3872
s5378	32.11	**40.70**	37.60	28.72	17.19	23754	40.70	14085	20758
s9234	33.44	**43.34**	41.53	33.47	22.33	39273	43.34	22250	25935
s13207	44.78	65.03	72.97	**74.78**	73.44	165200	74.78	41658	163100
s15850	35.37	**47.11**	46.79	40.45	31.07	76986	47.11	40717	57434
s35932*	49.83	74.68	87.04	93.15	96.14	4007299	98.51	59573	19393
s38417	33.55	**44.12**	42.38	35.22	25.20	164736	44.12	92054	113152
s38584	35.65	**47.71**	47.67	41.65	32.71	199104	47.71	104111	161040

are encoded using the same value of m. Each group of cores is assigned a dedicated demultiplexer. For an SoC with a large number of cores, grouping the cores in this fashion gives the maximum benefit without increasing testing time or hardware overhead. The problem of optimally assigning cores to different scan chains however remains an open problem and needs further investigation.

V. EXPERIMENTAL RESULTS

In this section, we experimentally evaluate the proposed test-data compression/decompression method for the ISCAS benchmark circuits and for two industrial circuits. We considered both full-scan and nonscan sequential circuits. For each of the full-scan circuits, we assumed a single scan chain for our experiments. The test set for each full-scan circuit was reordered to increase compression; on the other hand, no reordering was done for the nonscan circuits. The amount of compression obtained was computed as follows:

$$\text{Compression (percent)} = \frac{(\text{Total no. bits in } T_D - \text{Total no. bits in } T_E)}{(\text{Total no. bits in } T_D)} \times 100 = \frac{|T_D| - G}{|T_D|} \times 100.$$

TABLE II
EXPERIMENTAL RESULTS FOR (a) ISCAS'89 BENCHMARK CIRCUITS, (b) VARIOUS TEST SEQUENCES FOR INDUSTRIAL NONSCAN CIRCUIT CKT1, AND (c) VARIOUS TEST SEQUENCES FOR INDUSTRIAL NONSCAN CIRCUIT CKT2

ISACS 89 Circuit (non-scan)	Percentage compression for group size m						Size of T_D (bits)	Best compression (percent)	Size of T_E (bits)
	$m=2$	$m=4$	$m=8$	$m=16$	$m=32$	$m=64$			
s953	41.78	58.56	**63.78**	62.58	57.87	51.62	1168	63.78	423
s5378	41.07	57.68	**62.24**	60.79	–	–	169995	62.24	64176
s13207	49.24	73.36	85.21	90.81	93.30	**94.21**	42284	94.21	2491
s35932	48.10	71.41	82.31	87.03	88.58	**88.74**	147070	88.74	16554
s15850	48.32	71.81	82.97	87.91	89.62	**89.68**	430353	89.68	46872
s38417	47.05	69.08	78.06	81.79	**82.11**	80.34	22624	82.11	4046

(a)

TS1	46.71	68.53	78.12	81.56	**81.71**	80.38	25130	81.71	4595
TS2	47.77	70.68	81.14	85.61	**86.75**	86.38	23230	86.75	3078
TS3	46.06	66.90	75.98	**79.02**	78.40	76.30	5660	79.02	1187
TS4	48.17	71.25	82.12	86.85	**88.23**	88.12	18830	88.23	2216
TS5	47.95	70.71	81.20	85.80	**86.84**	86.47	21550	86.84	2835
TS6	47.05	69.05	79.05	82.88	**83.19**	82.03	18800	83.19	3160

(b)

TS1	46.89	68.81	78.44	82.12	**82.66**	81.34	11079	82.66	1921
TS2	37.60	51.70	**54.27**	50.85	43.16	-	234	54.27	107
TS3	45.94	67.20	76.29	**79.71**	79.55	-	14562	79.71	2954
TS4	46.89	68.81	78.44	82.12	**82.66**	81.34	11079	82.66	1921

(c)

TABLE III
COMPARISON BETWEEN G (OBTAINED EXPERIMENTALLY) WITH THE THEORETICAL BOUNDS $G_{\min}$ AND $G_{\max}$

ISACS circuit	Number of 1s (r)	Lower bound G_{min} (bits)	Size of encoded test set G (bits)	Upper bound G_{max} (bits)
c1355	572	2353	2627	2782
c1908	700	2456	2876	3071
c2670	1728	7717	8903	9229
c3450	1303	4218	4868	5195
c5315	2206	8283	9443	9938
c7552	6475	19102	21338	22339
s641	296	998	1098	1146
s713	290	931	1080	1148
s1196	589	2290	2570	2731
s1238	599	2414	2685	2863
s5378	3239	12416	14085	14845
s9234	5039	19896	22250	23675
s13207	6716	37189	41658	43485
s15850	8702	36650	40717	43177
s35932	5340	55886	59573	61216
s38417	20165	81514	92054	96637
s38584	23320	96416	104111	113906

TABLE IV
COMPARISON BETWEEN THE BEST VALUE OF m OBTAINED EXPERIMENTALLY AND ANALYTICALLY m_a

ISCAS circuit	Number of 1s (r)	No. of bits in T_D (n)	Best compression (m)	m_a
c1355	572	4838	4	5.16
c1908	700	4587	4	3.87
c2670	1728	20271	8	7.47
c3450	1303	6450	4	2.8
c5315	2206	15486	4	4.17
c7552	6475	25254	2	2.00
s641	296	1404	2	2.59
s713	290	1404	4	2.66
s1196	589	4448	4	4.54
s1238	599	4864	4	4.93
s5378	3239	23754	4	4.38
s9234	5039	39273	4	4.70
s13207	6716	165200	16	16.35
s15850	8702	76986	4	5.43
s35932	5340	4007299	512	519
s38417	20165	164736	4	4.97
s38584	23320	199104	4	5.22

The first set of experimental data that we present is based on the use of partially specified test sets (test cubes). The system integrator can determine the best Golomb code parameter and encode test cubes if they are provided by the core vendor. Alternatively, the core vendor can encode the test set for the core and provide the encoded test set along with the value of m to the core user, who can then use m to design the decoder. In a third possible scenario, the core vendor can encode the test set and provide it to the core user without disclosing the value of m used for encoding. Thus, T_E now serves as an encryption of the test data for IP protection and m serves as the "secret key." In this case, however, the core vendor must also design the decoder for the core and provide it to the core user.

Table I presents the experimental results for the ISCAS benchmark circuits with sets of test cubes T_D obtained from the Mintest ATPG program with dynamic compaction and compares the Golomb encoded test set T_E with compacted test set obtained using Mintest [11]. We carried out our experiments using a Sun Ultra 10 workstation with a 333-MHz processor

TABLE V
COMPARISON BETWEEN THE COMPRESSION OBTAINED WITH GOLOMB CODING AND RUN-LENGTH CODING

ISCAS Circuit (full-scan)	Size of T_D (bits)	Percentage compression obtained using Golomb coding	Percentage compression obtained using run-length coding	Difference $G - RL$ (percent)
s5378	23754	40.70	35.57	5.13
s9234	39273	43.34	40.08	3.26
s13207	165200	74.78	55.50	19.28
s15850	76986	47.11	42.10	5.01
s35932	4007299	98.51	62.32	36.19
s38417	164736	44.12	37.16	6.96
s38584	199104	47.71	42.40	5.31

TABLE VI
COMPARISON BETWEEN GOLOMB AND RUN-LENGTH CODING FOR FULLY SPECIFIED TEST SETS

Circuit	Size of T_D (bits)	Size of T_E (Golomb, bits)	Compression (Golomb, percent)	Size of T_E (run-length, bits)	Compression (run-length, percent)	Size of T_E [10]	Compression [10]
s13207	529900	278207 ($m = 4$)	47.49	323088	39.03	313666	15.8
s15850	400205	223356 ($m = 4$)	44.18	254838	36.32	260532	16.1
s38417	3076736	1321185 ($m = 8$)	57.05	1708227	44.48	1673680	16.2
s38584	1742160	1237049 ($m = 4$)	28.99	1346118	22.73	–	–

and 256 MB of dynamic random access memory. The table lists the sizes of the precomputed (original) test sets, the amount of compression achieved for several values of m, and the size of the smallest encoded test set.

As is evident from Table I, the best value of m depends on the test set. Not only do we achieve very high test-data compression with a suitable choice of m, but we also observe that in a majority of cases (e.g., for all but one of the ISCAS'89 circuits), the size of T_E is less than the smallest tests that have been derived for these circuits using ATPG compaction [11]. (These cases are shown shaded in Table I.) Hence, ATPG compaction may not always be necessary for saving memory and reducing testing time. This comparison is essential in order to show that storing T_E in ATE memory is more efficient than simply applying ATPG compaction to test cubes and storing the resulting compact test sets. For example, the effectiveness of statistical coding for full-scan circuits was not completely established in [9] since no comparison was drawn with ATPG compaction in that work.

We next present results on Golomb coding for nonscan circuits. For this set of experiments, we used HITEC [21] to generate test sequences (cubes) for some of the ISCAS'89 benchmark circuits (including the three largest ones) and determined the size of T_E in each case. Table II(a) illustrates the amount of compression achieved for these circuits. We also applied Golomb coding to two nonscan industrial circuits. These production circuits are microcontrollers, whose test data were provided to us by Delphi Delco Electronics Systems. The first circuit CKT1 contains 16.8-K gates, 145 flip flops, and 35 latches. The second (smaller) circuit contains 6.8-K gates, 88 flip flops, and 32 latches. The test sequences for these circuits were fully specified and they were derived using functional methods targeted at single stuck-at faults in their subcircuits. The results on Golomb coding for these circuits are presented in Table II(b) and (c). We achieved significant compression (over 80% on average) in all cases. Thus, the results show that the compression scheme is very effective for the nonscan circuits as well.

We next revisit the lower and upper bounds and the best value of m derived in Section II for test-data compression using Golomb codes. In Table III, we list these bounds and the actual compression obtained for the ISCAS circuits. Table III shows the number of ones in T_{diff}, size of the encoded test set T_E, and lower and upper bounds corresponding to each circuit. In Table IV, we list the best value of m determined experimentally and analytically (m_a). These results show that the experimental results are consistent with the theoretically predicted bounds.

An analytical comparison between run-length coding and Golomb coding was presented in Section II. Here, we present experimental results to reinforce that comparison. Table V compares the amount of compression obtained with run-length coding for $b = 3$ with Golomb coding for the large ISCAS benchmark circuits. Golomb codes give better compression in all cases. For example, the compression is almost 20% better for s13207. While run-length coding may yield slightly better compression (for higher values of b), the complexity of the run-length decoder increases considerably with an increase in b.

If the precomputed test set T_D is already compacted using ATPG methods, then the compression obtained using Golomb codes is considerably less. Nevertheless, we have seen that a significant amount of compression is often achieved if Golomb coding is applied to an ATPG-compacted T_D. Table VI lists the compression achieved for some ISCAS benchmark circuits with test sets derived using SIS [22]. The corresponding compression results achieved with run-length coding (block size $b = 3$) are also shown and are significantly less. Unfortunately, we were unable to directly compare our results with [10] since the test

TABLE VII
COMPARISON BETWEEN THE EXTERNAL CLOCK FREQUENCY f_{ext} REQUIRED FOR GOLOMB-CODED TEST DATA AND THE EXTERNAL CLOCK FREQUENCY f'_{ext} REQUIRED FOR EXTERNAL TESTING USING ATPG-COMPACTED PATTERNS (FOR THE SAME TESTING TIME)

Circuit	m	r	n_c	pn	f'_{ext}/f_{ext}
s9234	4	5039	22250	25935	1.93
s13207	16	6716	41658	163100	9.90
s15850	4	8702	40717	57434	2.25
s38417	4	20165	92054	113152	1.99
s38584	4	23320	104111	161040	2.54

sets used in [10] are no longer available. However, we note that Golomb coding indirectly outperforms [10] since T_E is much smaller and compression is significantly higher for Golomb-coded test sets in all cases.

Table VII demonstrates that Golomb coding allows us to use a slower tester without incurring any testing time penalty. As discussed in Section III, Golomb coding provides three important benefits: 1) it significantly reduces the volume of test data; 2) the test patterns can be applied to the core under test at the scan clock frequency f_{scan} using an external tester that runs at frequency $f_{\text{ext}} = f_{\text{scan}}/m$; and 3) in comparison with external testing using ATPG-compacted patterns, the same testing time is achieved using a much slower tester. The third issue is highlighted in Table VII.

VI. CONCLUSION

We have presented a new test vector compression method and decompression architecture for testing embedded cores in an SoC. The proposed method is based on variable-to-variable-length Golomb codes. We have shown that Golomb codes can be used for efficient compression of test data for SoCs and to save ATE memory and testing time. Golomb coding is inherently superior then run-length coding; we have demonstrated this analytically and through experimental results.

The on-chip decoder is small and easy to implement. In addition, it is scalable and independent of the core under test and the precomputed test set. With careful design, it is possible to ensure proper synchronization between the decoder and the tester. We have also presented a novel decompression architecture for testing multiple cores simultaneously. This reduces the testing time of an SoC further and increases the ATE I/O channel capacity considerably. The novel decompression architecture is a direct consequence of the structure of the Golomb codes.

Experimental results for the ISCAS benchmark show that the compression technique is very efficient for combinational and full-scan circuits. Significant compression is achieved not only for test cubes, but also for compacted fully specified test sets. The results show that ATPG compaction may not be always necessary for saving ATE memory and reducing testing time. We also achieved substantial compression for two nonscan industrial circuits and for the nonscan ISCAS'89 circuits using HITEC test sets. These results show that Golomb coding is also attractive for compressing (ordered) test sequences of nonscan circuits. Finally, we have demonstrated that Golomb is superior to run-length coding for all benchmark circuits.

ACKNOWLEDGMENT

The authors would like to thank Dr. M. C. Hansen of Delphi Delco Electronics Systems for providing test sequences for the industrial circuits and Dr. A. Morosov of the University of Potsdam for generating test sets using SIS.

REFERENCES

[1] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," in *Proc. Int. Test Conf.*, Nov. 1998, pp. 130–143.

[2] G. Hetheringten, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, "Logic BIST for large industrial designs," in *Proc. Int. Test Conf.*, Sept. 1999, pp. 358–367.

[3] B. T. Murray and J. P. Hayes, "Testing ICs: Getting to the core of the problem," *Computer*, vol. 29, pp. 32–38, Nov. 1996.

[4] C.-A. Chen and S. K. Gupta, "Efficient BIST TPG design and test set compaction via input reduction," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 692–705, Aug. 1998.

[5] K. Chakrabarty and B. T. Murray, "Design of built-in test generator circuits using width compression," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1044–1051, Oct. 1998.

[6] K. Chakrabarty, B. T. Murray, and V. Iyengar, "Built-in pattern generation for high-performance circuits using twisted-ring counters," in *Proc. IEEE VLSI Test Symp.*, May 1999, pp. 22–27.

[7] V. Iyengar, K. Chakrabarty, and B. T. Murray, "Built-in self testing of sequential circuits using precomputed test sets," in *Proc. IEEE VLSI Test Symp.*, May 1998, pp. 418–423.

[8] ——, "Deterministic built-in pattern generation for sequential circuits," *J. Electron. Test. Theory Applicat.*, vol. 15, pp. 97–115, Aug./Oct. 1999.

[9] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan vector compression/decompression using statistical coding," in *Proc. IEEE VLSI Test Symp.*, May 1999, pp. 114–120.

[10] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based design," in *Proc. Int. Test Conf.*, Nov. 1998, pp. 458–464.

[11] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1998, pp. 283–289.

[12] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "On compacting test sets by addition and removal of vectors," in *Proc. IEEE VLSI Test Symp.*, May 1994, pp. 202–207.

[13] T. Yamaguchi, M. Tilgner, M. Ishida, and D. S. Ha, "An efficient method for compressing test data," in *Proc. Int. Test Conf.*, Nov. 1997, pp. 79–88.

[14] M. Ishida, D. S. Ha, and T. Yamaguchi, "COMPACT: A hybrid method for compressing test data," in *Proc. IEEE VLSI Test Symp.*, May 1998, pp. 62–69.

[15] S. W. Golomb, "Run-length encoding," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 399–401, Dec. 1966.

[16] H. Kobayashi and L. R. Bahl, "Image data compression by predictive coding, part I: Prediction algorithm," *IBM J. Res. Dev.*, vol. 18, p. 164, 1974.

[17] G. Seroussi and M. J. Weinberger, "On adaptive strategies for an extended family of Golomb-type code," in *Proc. Data Compression Conf.*, 1997, pp. 131–140.

[18] N. Merhav, G. Seroussi, and M. J. Weinberger, "Optimal prefix codes for two-sided geometric distribution," in *Proc. IEEE Int. Symp. Information Theory*, June 1997, p. 71.

[19] Y. Zorian, "Test requirements for embedded core-based systems and IEEE P1500," in *Proc. Int. Test Conf.*, Nov. 1997, pp. 191–199.

[20] "Design Compiler Reference Manual," Synopsys Inc., Mountain View, CA, 1992.

[21] Research in VLSI Circuit Testing, Verification, and Diagnosis, Univ. Illinois. [Online]. Available: www.crhc.uiuc.edu/IGATE

[22] E. M. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis," Electronic Res. Lab., Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M92/41, 1992.

[23] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Dept. of Electrical Eng., Virginia Polytechnic Inst. and State Univ., Tech. Rep. 12_93.

[24] V. D. Agrawal and T. J. Chakraborty, "High-performance circuit testing using slow-speed testers," in *Proc. Int. Test Conf.*, Oct. 1995, pp. 302–310.

[25] D. Heidel, S. Dhong, P. Hofstee, M. Immediato, K. Nowka, J. Silberman, and K. Stawiasz, “High-speed serialiazing/de-serializing design-for-test methods for evaluating a 1 GHz microprocessor,” in *Proc. IEEE VLSI Test Symp.*, May 1998, pp. 234–238.

Anshuman Chandra (S’97) received the B.E. degree in electrical engineering from the University of Roorkee, Roorkee, India, in 1998, the M.S. degree in electrical and computer engineering from Duke University, Durham, NC, in 2000, and is working toward the Ph.D. degree at the same university.

His research interests are in the fields of very large scale integration design, digital testing, and computer architecture. He is currently working in the areas of test-set compression/decompression, embedded core testing, and built-in self test.

Mr. Chandra is a student member of the ACM SIGDA. He received the Test Technology Technical Council James Beausang Student Paper Award for a paper in *Proc. 2000 IEEE VLSI Test Symposium*.

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

He is currently an Assistant Professor of Electrical and Computer Engineering at Duke University, Durham, NC. He has authored or coauthored over 60 papers and holds a U.S. patent in built-in self test. His current research interests (supported by NSF, DARPA, ONR, and industrial sponsors) are in system-on-a-chip test, embedded real-time operating systems, distributed sensor networks, and architectural optimization of microelectrofluidic systems.

Dr. Chakrabarty is a Member of ACM, ACM SIGDA, and Sigma Xi. He received the National Science Foundation Early Faculty (CAREER) Award, the Office of Naval Research Young Investigator Award, and a Mercator Professor Award from the Deutsche Forschungsgemeinschaft, Germany. He is Vice Chair of Technical Activities for the IEEE Test Technology Technical Council and is a member of several program committees for IEEE/ACM conferences and workshops.