

# Test Resource Partitioning for SOCs

Anshuman Chandra and Krishnendu Chakrabarty

Duke University

A new test-resource-partitioning approach, based on test data compression and on-chip decompression, reduces data volume, decreases testing time, and accommodates slower (less expensive) testers without decreasing test quality.

■ **SHRINKING PROCESS TECHNOLOGIES** and increasing design sizes have led to highly complex, billion-transistor integrated circuits. Testing these complex ICs to weed out defective parts has become a major challenge. To reduce design and manufacturing costs, testing must be quick and effective. The time it takes to test an IC depends on test data volume. The rapidly increasing number of transistors in ICs has spurred enormous growth in test data volume. Techniques that decrease test data volume and testing time are necessary to increase production capacity and reduce test cost.

The latest system-on-a-chip designs integrate multiple ICs (microprocessors, memories, DSPs, and I/O controllers) on a single piece of silicon. SOCs consist of several reusable embedded intellectual-property (IP) cores provided by third-party vendors and stitched into designs by system integrators. Testing all these circuits when they are embedded in a single device is far more difficult than testing them separately. Achieving satisfactory SOC test quality at an acceptable cost and with minimal

effect on the production schedule is also becoming increasingly difficult. High transistor counts and aggressive clock frequencies require expensive automatic test equipment (ATE). More important, they introduce many problems into test development and manufacturing test that decrease product quality and increase cost and time to market.

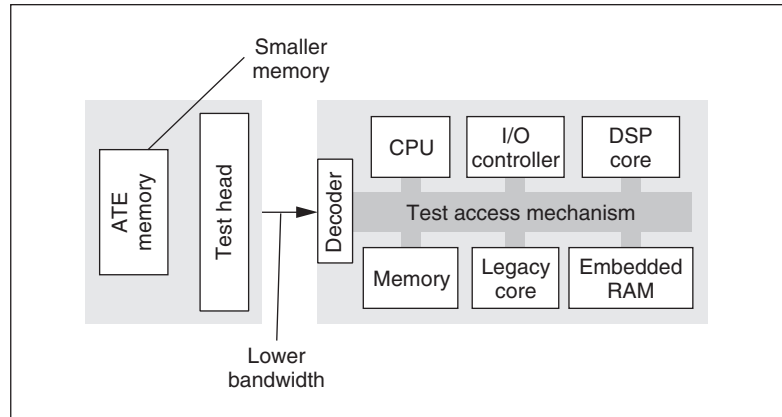
ATE costs have been rising steeply. A tester that can accurately test today's complex ICs costs several million dollars. According to the 1999 *International Technology Roadmap for Semiconductors* ([http://public.itrs.net/Files/1999\\_SIA\\_Roadmap/Home.htm](http://public.itrs.net/Files/1999_SIA_Roadmap/Home.htm)), the cost of a high-speed tester will exceed \$20 million by 2010, and the cost of testing an IC with conventional methods will exceed fabrication cost. Conventional direct-probe testing methods have become inadequate and are no longer commercially practical. The increasing ratio of internal node count to external pin count makes most chip nodes inaccessible from system I/O pins, so controlling and observing these nodes and exercising the numerous internal states in the circuit under test is difficult. ATE I/O channel capacity, speed, accuracy, and data memory are limited. Therefore, design and test engineers need new techniques for decreasing data volume.

Test resource partitioning offers a promising solution to these problems by moving some test resources from ATE to chip. Our new TRP approach, based on test data compression and on-chip decompression, reduces test data volume, decreases testing time, and uses slower testers without decreasing test quality.

## Overview

There are three main TRP techniques:

- **Test set compaction.** This technique reduces test data volume by compacting the partially specified test cubes generated by automatic test pattern generation (ATPG) algorithms. It requires no additional hardware investment. The test set is compacted through dynamic or static compaction procedures.<sup>1,2</sup> However, test set compaction results in the application of fewer patterns to the SOC. Because every modeled fault is thus detected by fewer patterns, this approach can reduce unmodeled-fault coverage.<sup>3</sup>
- **Built-in self-test.** BIST, an alternative to ATE-based external testing, offers several advantages: It lets precomputed test sets be embedded in test sequences generated by on-chip hardware, supports test reuse and at-speed testing, and protects intellectual property. Although BIST is now extensively used for memory testing, it is not as common for logic testing. This is particularly true for nonscan and partial-scan designs in which test vectors cannot be reordered and applying pseudorandom vectors can lead to serious bus contention problems during testing. Moreover, BIST can be applied to SOC designs only if the IP cores are BIST-ready. Because most currently available IP cores are not BIST-ready, BIST insertion in SOCs containing these circuits is expensive and requires considerable redesign.
- **Test data compression.** Another way to reduce test data volume is through data compression techniques such as statistical, run-length, Golomb, and frequency-directed run-length (FDR) coding.<sup>4,7</sup> These techniques



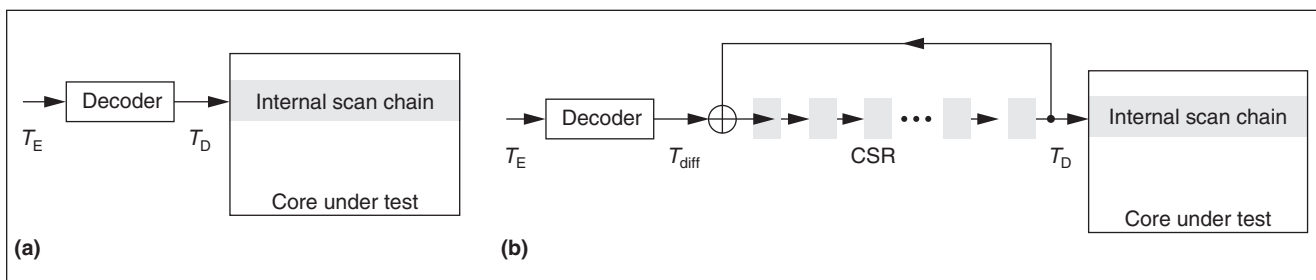
**Figure 1. TRP scheme for testing a SOC by storing encoded test data in ATE memory and decoding it with on-chip decoders.**

compress precomputed test set  $T_D$ , provided by the core vendor, into the much smaller test set  $T_E$ , which is stored in ATE memory. Figure 1 shows a TRP scheme using test data compression. An on-chip decoder performs pattern decompression to generate  $T_D$  from  $T_E$  during pattern application. Compressing difference-vector sequence  $T_{diff}$  determined from  $T_D$  decreases test set size and reduces testing time.<sup>5,6</sup> Figure 2 shows test architectures based on  $T_D$  and  $T_{diff}$  and cyclical scan registers. However, using  $T_{diff}$  and CSRs is not always necessary. Directly encoding  $T_D$  can also achieve significant compression.

Our TRP approach uses the third technique, which reduces test data volume more than test set compaction and is less expensive than BIST.

### Run-length codes

To encode SOC test data, we first decompose it into either fixed-length or variable-length blocks. We then assign each block a code



**Figure 2. Decompression architectures using precomputed test set  $T_D$  (a) and a cyclical scan register (CSR) and difference-vector test set  $T_{diff}$  (b).**

**Table 1. An example of conventional run-length encoding for block size  $b = 3$ .**

Group	Run length	Code word
$A_1$	0	000
	1	001
	2	010
	3	011
	4	100
	5	101
	6	110
$A_2$	7	111
	8	111000
	9	111001
	10	111010
...	11	111011
	...	...

word, also of either fixed or variable length. Assigning a fixed-length code word to fixed-length data blocks doesn't lead to significant compression, so we must consider variable-to-fixed-length and variable-to-variable-length encoding.

Variable-to-fixed-length: conventional run-length coding

The first step in encoding test set  $T_D$  is to generate a fully specified test set with long runs of 0s followed by a single 1. Run-length codes can be used to compress both difference-vector sequence  $T_{diff}$  and  $T_D$ . Let  $T_D = \{t_1, t_2, t_3, \dots, t_n\}$  be the (ordered) precomputed test set. A straightforward heuristic procedure determines the ordering.<sup>6</sup> We say that  $T_{diff} = \{d_1, d_2, \dots, d_n\} = \{t_1, t_1 \oplus t_2, t_2 \oplus t_3, \dots, t_{n-1} \oplus t_n\}$ , where a bitwise exclusive-or operation is carried out between patterns  $t_i$  and  $t_{i+1}$ . If uncompact 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.

The next step is to select block size  $b$ . Once  $b$  is determined, the runs of 0s are mapped to groups of size  $M + 1 = 2^b$ . The length of the longest run of 0s determines the number of groups. The set of run lengths  $\{0, 1, 2, \dots, M - 1\}$  and a run of  $M$  0s form group  $A_1$ ; the set  $\{M, M + 1, M + 2, \dots, 2M - 1\}$  and a run of  $2M$  0s form group  $A_2$ ; and so on. In general, the set of run

lengths  $\{(k - 1)M, (k - 1)M + 1, (k - 1)M + 2, \dots, kM - 1\}$  and a run of  $kM$  0s comprise group  $A_k$ . The code word size for the  $k$ th group is  $k(M + 1)$ . Table 1 shows the encoding.

Variable-to-variable-length:

Golomb coding

The first step in the encoding procedure is to select Golomb code parameter  $m$ . For certain distributions of the input data stream ( $T_{diff}$ , in our case), group size  $m$  can be optimally determined. For example, if the input data stream is random with 0-probability  $p$ , then  $m$  should be chosen such that  $p^m \approx 0.5$ . However, because 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.

Once group size  $m$  is determined, the runs of 0s in the precomputed test set are mapped to groups of size  $m$  (each group corresponding to a run length). The length of the longest run of 0s in the precomputed test set determines the number of groups. The set of run lengths  $\{0, 1, 2, \dots, m - 1\}$  forms group  $A_1$ ; the set  $\{m, m + 1, m + 2, \dots, 2m - 1\}$  forms group  $A_2$ ; and so on. In general, the set of run lengths  $\{(k - 1)m, (k - 1)m + 1, (k - 1)m + 2, \dots, km - 1\}$  comprises group  $A_k$ .

To each group  $A_k$ , we assign a group prefix of  $(k - 1)$  1s followed by a 0. We denote this by  $1^{(k-1)}0$ . If  $m$  is determined to be a power of 2 (that is,  $m = 2^N$ ), each group contains  $2^N$  members, and a sequence (a tail) of  $\log_2(m)$  bits uniquely identifies each member in the group. Thus, the final code word for 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. Thus,  $(k - 1) = (L \bmod m)$ —that is,  $k = (L \bmod m) + 1$ . Table 2 shows an example of Golomb encoding.

Variable-to-variable length: FDR coding

The need for FDR coding arises from the distribution of runs of 0s in typical test sets. We conducted a series of experiments on the large benchmark circuits from the International Symposium on Circuits and Systems (ISCAS) and studied the distribution of runs of 0s in  $T_{diff}$

obtained from complete single stuck-at test sets for these circuits. Figure 3 illustrates this distribution for benchmark s9234. We found that the distributions were similar for other circuits' test sets.

Figure 3 shows that the frequency of runs of 0s of length  $l$

- is high for  $0 \leq l \leq 20$ ,
- is very low for  $l \geq 20$ , and
- decreases rapidly with decreasing  $l$  even within the range  $0 \leq l \leq 20$ .

If we use conventional run-length coding with block size  $b$  for compressing such test sets, every run of  $l$  0s,  $0 \leq l \leq 2^{b-1}$ , is mapped to a  $b$ -bit code word. This is clearly inefficient for the large number of short runs of 0s. Likewise, if we use Golomb coding with code parameter  $m$ , a run of  $l$  0s is mapped to a code word with  $\lfloor l/m \rfloor + 1 + \log_2(m)$  bits. This is also inefficient for short runs of 0s. Clearly, test data compression is more efficient if the more frequently occurring runs of 0s are mapped to shorter code words. This leads us to the notion of FDR codes.

FDR code is constructed as follows: The runs of 0s are divided into groups  $A_1, A_2, A_3, \dots, A_k$ , where  $k$  is determined by length  $l_{\max}$  of longest run ( $2^k - 3 \leq l_{\max} \leq 2^{k+1} - 3$ ). Also, a run of length  $l$  is mapped to group  $A_j$ , where  $j = \lceil \log_2(l+3) - 1 \rceil$ . The  $i$ th group's size equals  $2^i$ —that is,  $A_i$  contains  $2^i$  members. Each code word consists of two parts—a group prefix and a tail. The group prefix identifies the group to which the run belongs, and the tail identifies the group's members. Table 3 shows an example of FDR encoding.

FDR code has the following properties:

- For any code word, prefix and tail are of equal length. For example, they are each one bit long for  $A_1$ , two bits long for  $A_2$ , and so on.
- The length of the prefix for group  $A_i$  equals  $i$ . For example, the prefix is 2 bits long for group  $A_2$ .
- For any code word, the prefix is identical to the binary representation of the run length corresponding to the group's first element. For example, run-length 8 is mapped to group  $A_3$ , and this group's first element is run-length 6. Hence, the prefix of the code word for run-length 8 is 110.

Table 2. An example of Golomb encoding for group size  $m = 4$ .

Group	Run length	Group prefix	Tail	Code word
$A_1$	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
...	...	...	...	...

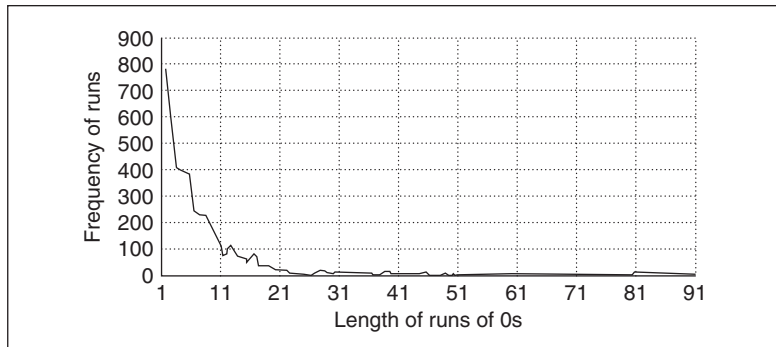
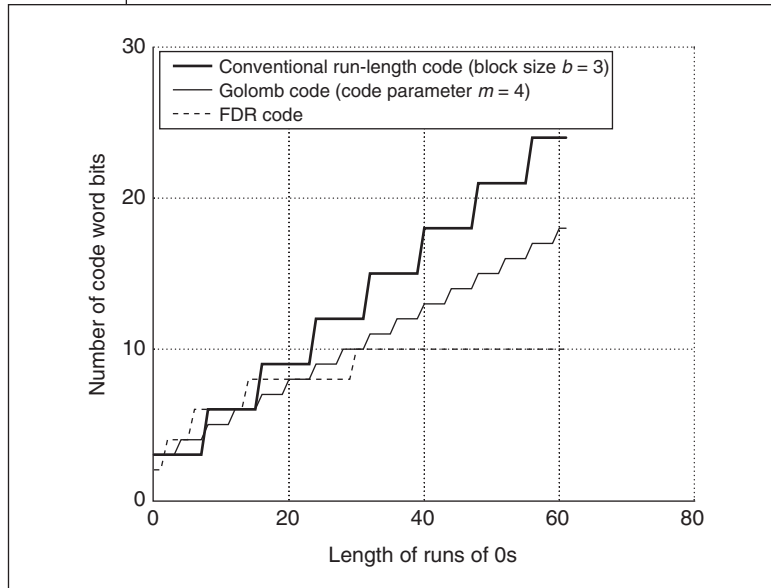


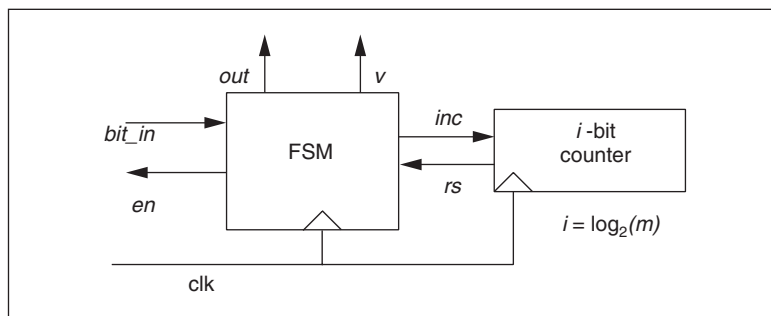
Figure 3. Distribution of runs of 0s for ISCAS benchmark circuit s9234.

Table 3. An example of FDR encoding.

Group	Run length	Group prefix	Tail	Code word
$A_1$	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		
...	...	...	...	...



**Figure 4. Comparison of code word size (bits) for different run lengths for FDR code, Golomb code, and conventional run-length code.**



**Figure 5. Block diagram of the decoder used for decompression.**

- Code word size increases by 2 bits (1 bit for the prefix and 1 bit for the tail) as we move from group  $A_i$  to group  $A_{i+1}$ .

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 the runs of 0s grow—that is,  $A_i$  is smaller than  $A_{i+1}$ . However, tails for Golomb code words in different groups are of equal length ( $\log_2(m)$ , where  $m$  is the code parameter), and the prefix

increases by only 1 bit as we move from one group to another. Hence, Golomb coding is less effective when the runs of 0s spread far from an effective range determined by  $m$ .

Figure 4 compares the three codes, showing the number of bits per code word for different-length runs of 0s. Conventional run-length code's performance is worse than that of Golomb code when run-length  $l$  exceeds 7. Golomb code's performance is worse than that of FDR code for  $l \geq 24$ . FDR code outperforms the other two types for runs of lengths 0 and 1. Since these runs' frequencies are very high for precomputed test sets (Figure 3), FDR codes outperform run-length and Golomb codes for SOC test data compression.

#### Test data compression and decompression

Although the on-chip decoder designs are similar for the three codes we've described, we discuss only the Golomb decoder in this article. The decoder is simple, scalable, and independent of the core under test and the precomputed test set. Moreover, because it is small, it does not introduce significant hardware overhead.

The decoder decompresses encoded test set  $T_E$  and outputs  $T_{diff}$ . The exclusive-or gate and the CSR generate test patterns from the difference vectors. A counter of  $\log_2(m)$  bits and a finite-state machine can efficiently implement the decoder. Figure 5 shows the decoder's block diagram. The input to the FSM is  $bit\_in$ , and enable signal  $en$  is used to input the bit whenever the decoder is ready. Signal  $inc$  increments the counter, and  $rs$  indicates that the counter has finished counting. Signal  $out$  is the decoder output, and  $v$  indicates when the output is valid.

The decoder operates as follows:

- When the input is 1, the counter counts up to  $m$ . 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  $m$  0s during this operation and makes  $v$  high.
- When the input is 0, the FSM starts decoding the input code word's tail. The number of output 0s depends on the binary value of tail bits. The  $en$  and  $v$  signals synchronize the decoder's input and output operations.

Figure 6 shows the FSM state diagram corresponding to the decoder for  $m = 4$ . States S0 to S3 and S4 to S8 correspond to prefix and tail decoding, respectively. We synthesized the FSM using the Synopsys Design Compiler to access the decoder's hardware overhead. The synthesized circuit contains only four flip-flops and 34 combinational gates. For a 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 counter. Thus, the decoder is independent of not only the core under test but also its pre-computed test set. The amount of extra logic required for decompression is small and can be implemented easily—in contrast to the run-length decoder, which is not scalable and becomes increasingly complex for higher values of block length  $b$ .

#### Test application time

Here, we analyze the testing time for a single scan chain using Golomb coding with the test architecture shown in Figure 2. The Golomb decoder's state diagram indicates that

- each 1 in the prefix takes  $m$  cycles for decoding,
- each separator 0 takes one cycle, and
- the tail takes a maximum of  $m$  cycles and a minimum of  $\gamma = \log_2(m) + 1$  cycles.

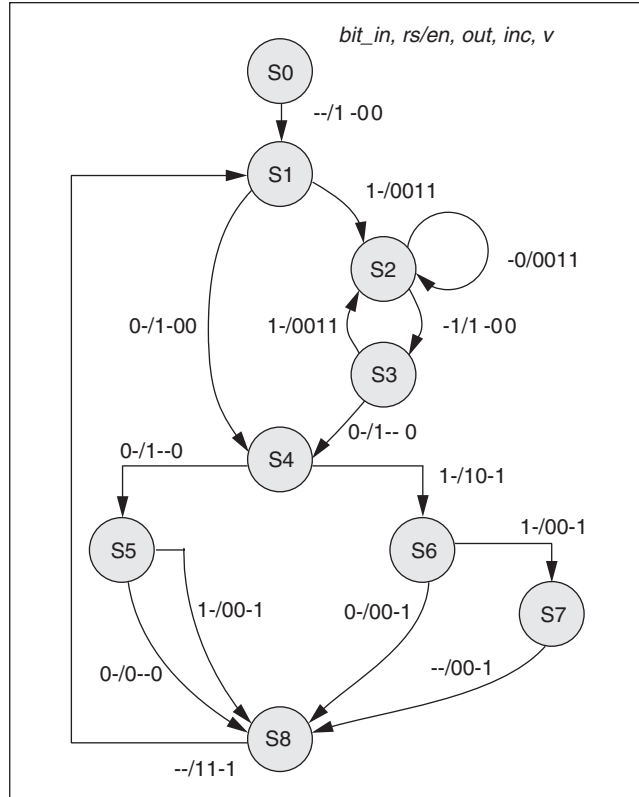
We let  $n_c$  be the total number of bits in  $T_E$ , and  $r$  the number of 1s in  $T_{diff}$ .  $T_E$  contains  $r$  tails and  $r$  separator 0s, and the number of prefix 1s in  $T_E$  equals  $n_c - r[1 + \log_2(m)]$ . Therefore, maximum and minimum testing times  $T_{max}$  and  $T_{min}$ , measured by the number of cycles, are

$$\begin{aligned} T_{max} &= \{n_c - r[1 + \log_2(m)]\}m + r + mr \\ &= mn_c - r[m\log_2(m) - 1] \end{aligned}$$

$$\begin{aligned} T_{min} &= \{n_c - r[1 + \log_2(m)]\}m + r + \gamma r \\ &= mn_c - rm[1 + \log_2(m)] - (1 + \gamma) \end{aligned}$$

Therefore, the difference between  $T_{max}$  and  $T_{min}$  is

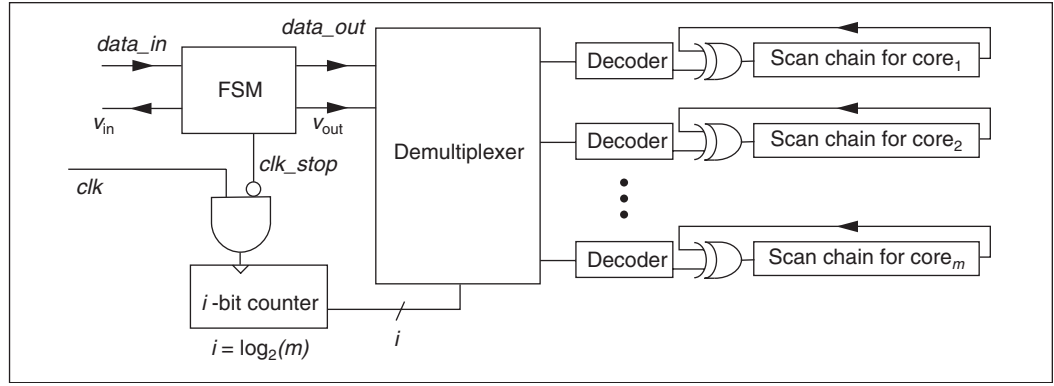
$$\begin{aligned} \delta T &= T_{max} - T_{min} \\ &= r[m - \log_2(m) - 1] \end{aligned}$$



**Figure 6. Decoder FSM state diagram.**

A major advantage of Golomb coding is that on-chip decoding can be carried out at scan clock frequency  $f_{scan}$  while  $T_E$  is fed to the core under test with external clock frequency  $f_{ext} < f_{scan}$ . This lets us use slower testers without increasing test application time. The external clock and the scan clocks must be synchronized such that  $f_{scan} = mf_{ext}$ , where Golomb code parameter  $m$  is usually a power of 2.<sup>8</sup> Therefore, the decoder can generate the bits of  $T_{diff}$  at  $f_{scan}$ .

Next, we analyze testing time, using  $f_{scan} = mf_{ext}$ , and compare our method's testing time with that of external testing applying ATPG-compacted patterns. We let the ATPG-compacted test set contain  $p$  patterns and the scan chain's length be  $n$  bits. Therefore, the ATPG-compacted test set's size is  $pn$  bits, and testing time  $T_{ATPG}$  equals  $pn$  external clock cycles. Next, we let difference-vector  $T_{diff}$  obtained from the uncompact test set contain  $r$  1s and its Golomb-coded test set  $T_E$  contain  $n_c$  bits. Therefore, the maximum number of scan clock cycles required for applying test patterns in the Golomb coding scheme is  $T_{max} = mn_c - r[m\log_2(m) - 1]$ .



**Figure 7. SOC channel selector for application to multiple cores and multiple scan chains.**

The maximum testing time  $\tau$  (seconds) with Golomb coding is

$$\begin{aligned}\tau &= T_{\max}/f_{\text{scan}} \\ &= \{mn_c - r[m\log_2(m) - 1]\}/f_{\text{scan}}\end{aligned}$$

and testing time  $\tau'$  (seconds) for external testing with ATPG-compacted patterns is

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

If we are to accomplish testing in  $\tau^*$  seconds with Golomb coding, scan clock frequency  $f_{\text{scan}}$  must equal  $T_{\max}/\tau^*$ . That is,  $f_{\text{scan}} = \{mn_c - r[m\log_2(m) - 1]\}/\tau^*$ . We meet this requirement using a slow external tester operating at frequency  $f_{\text{ext}} = f_{\text{scan}}/m$ . On the other hand, if we use only external testing with  $p$  ATPG-compacted patterns, the required external tester clock frequency  $f'_{\text{ext}}$  is  $pn/\tau^*$ .

The ratio between  $f'_{\text{ext}}$  and  $f_{\text{ext}}$  is

$$\begin{aligned}f'_{\text{ext}}/f_{\text{ext}} &= (pn/\tau^*)/(f_{\text{scan}}/m) \\ &= pn/[n_c - r\log_2(m) + r/m]\end{aligned}$$

Our experimental results show that in all cases the ratio is greater than 1, demonstrating that Golomb-code-based TRP lets us decrease test data volume and use a slower tester without increasing testing time.

### Interleaving decompression architecture

Our interleaving decompression architecture based on Golomb coding enables testing

of multiple cores in parallel with a single ATE I/O channel. The architecture reduces testing time and test data volume stored in ATE memory and increases ATE I/O channel capacity.

As discussed earlier, when Golomb coding is applied to a data block containing a run of 0s followed by a single 1, the code word contains two parts—prefix and tail. For given code parameter  $m$  (group size), the tail's length  $\log_2(m)$  is independent of the run length. Every 1 in the prefix corresponds to  $m$  0s in the decoded difference vector. Thus, the prefix consists of a string of 1s followed by a 0, and the 0 identifies the tail's beginning.

Whenever the decoder FSM receives a 1, it runs the counter for  $m$  decode cycles and starts decoding the tail as soon as it receives a 0. Tail decoding takes at most  $m$  cycles. During prefix decoding, the FSM must wait  $m$  cycles before the next prefix bit can be decoded. Therefore, we can test  $m$  cores in parallel by using interleaving to feed each core's decoder with encoded prefix data every  $m$  cycles. Interleaving can also be used to feed multiple scan chains in parallel, as long as their capture cycles are synchronized. Whenever a tail is to be decoded, the respective decoder is fed with the entire tail of  $\log_2(m)$  bits in a single burst of  $\log_2(m)$  cycles.

Figure 7 shows the SOC channel selector used for interleaving. It consists of a demultiplexer, an  $i$ -bit counter, and an FSM. Interleaving proceeds as follows: First, the SOC integrator combines the encoded test data for  $m$  cores to generate composite bit stream  $T_c$ , which is stored in the ATE. Next,  $T_c$  is fed to the FSM, which detects the

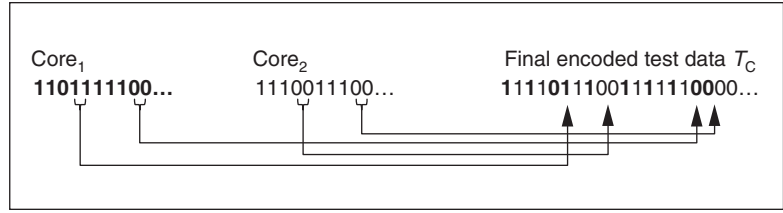
beginning of each tail and feeds the demultiplexer. An  $i$ -bit counter, where  $i = \log_2(m)$ , selects the outputs to the various cores' decoders.

$T_C$  is obtained by interleaving the prefixes of each core's compressed test sets, but the tails are included unchanged. In the example in Figure 8, compressed data for two cores (generated using group size  $m = 2$ ) have been interleaved to obtain  $T_C$ . The final encoded test set will be applied to multiple cores through decompression.

Let's describe the SOC channel selector in greater detail. The FSM detects the tail's beginning and generates  $clk\_stop$  to stop the  $i$ -bit counter. Signals  $v_{in}$  and  $v_{out}$  indicate that  $data\_in$  and  $data\_out$  are valid. The  $i$ -bit counter connects to the demultiplexer's select lines, and the demultiplexer's outputs connect to the different scan chains' decoders. Each scan chain has a dedicated decoder. This decoder receives either a 1 or the tail of the compressed data corresponding to the various cores connected to the scan chain. If the FSM detects that a portion of the tail has arrived, the 0 that identifies the tail passes to the decoder, and  $clk\_stop$  goes high for the next  $m$  cycles. The demultiplexer's output doesn't change during this period, and the entire tail passes continuously to the appropriate core.

Figures 9a and 9b (next page) show the FSM state diagram for  $m = 4$  and the corresponding timing diagram. The FSM is fed  $T_C$  corresponding to four different cores. It remains in state S0 as long as it receives 1s corresponding to the prefixes. As soon as it receives a 0, it outputs the entire tail unchanged and makes  $clk\_stop$  high. This stops the  $i$ -bit counter and prevents any change at the demultiplexer output. The timing diagram shows that whenever the FSM receives a 0, SOC channel selection remains unchanged for the next  $m + 1$  cycles.

The difference between maximum and minimum testing times for a single tail is  $\delta t = [m - \log_2(m) - 1]$ . If we restrict  $m$  to be small, then  $m \leq 8$  and  $\delta t \leq 4$ . In this case, we can easily modify the decoder FSM by introducing additional states to the Golomb decoder FSM such that the tail decoding always takes  $m$  cycles and  $\delta t = 0$ . As Figure 10 shows, three additional states are required to equalize tail and prefix



**Figure 8. Interleaving composite encoded test data for two cores with group size  $m = 2$ .**

decoding for  $m = 4$ . The additional states don't increase testing time and hardware overhead significantly. There are  $m$  cores in parallel, and each separator 0 and tail takes  $m + 1$  cycles to decode. For  $m$  cores, therefore, the decoding time for the separator and tail is

$$\begin{aligned} t_{tail} &= \sum_{j=1}^m (r_j + mr_j) \\ &= (m+1) \sum_{j=1}^m r_j \\ &= (m+1)R \end{aligned}$$

where

$$R = \sum_{j=1}^m r_j$$

Because all cores' prefixes are decoded in parallel, the number of cycles  $t_{prefix}$  required for decoding all the prefixes in  $T_C$  equals the number of 1s in the prefix of the core with the largest amount of encoded test data. Therefore,

$$\begin{aligned} t_{prefix} &= \max(m\{n_{C,i} - r_i[1 + \log_2(m)]\}) \\ &= m\{n_{C,max} - r_{max}[1 + \log_2(m)]\} \end{aligned}$$

where  $n_{C,i}$  and  $r_i$  are the number of encoded bits in  $T_E$  and the number of 1s in  $T_{diff}$  for the  $i$ th core, respectively. Moreover,  $n_{C,max}$  and  $r_{max}$  are the number of encoded bits in  $T_E$  and the number of 1s in  $T_{diff}$  for the core with the largest amount of encoded test data. Therefore, total testing time for  $m$  cores tested in parallel with interleaving is

$$\begin{aligned} T_1 &= t_{prefix} + t_{tail} \\ &= m\{n_{C,max} - r_{max}[1 + \log_2(m)]\} + R(1 + m) \end{aligned}$$

In contrast, we now find the testing time  $T_{NI}$

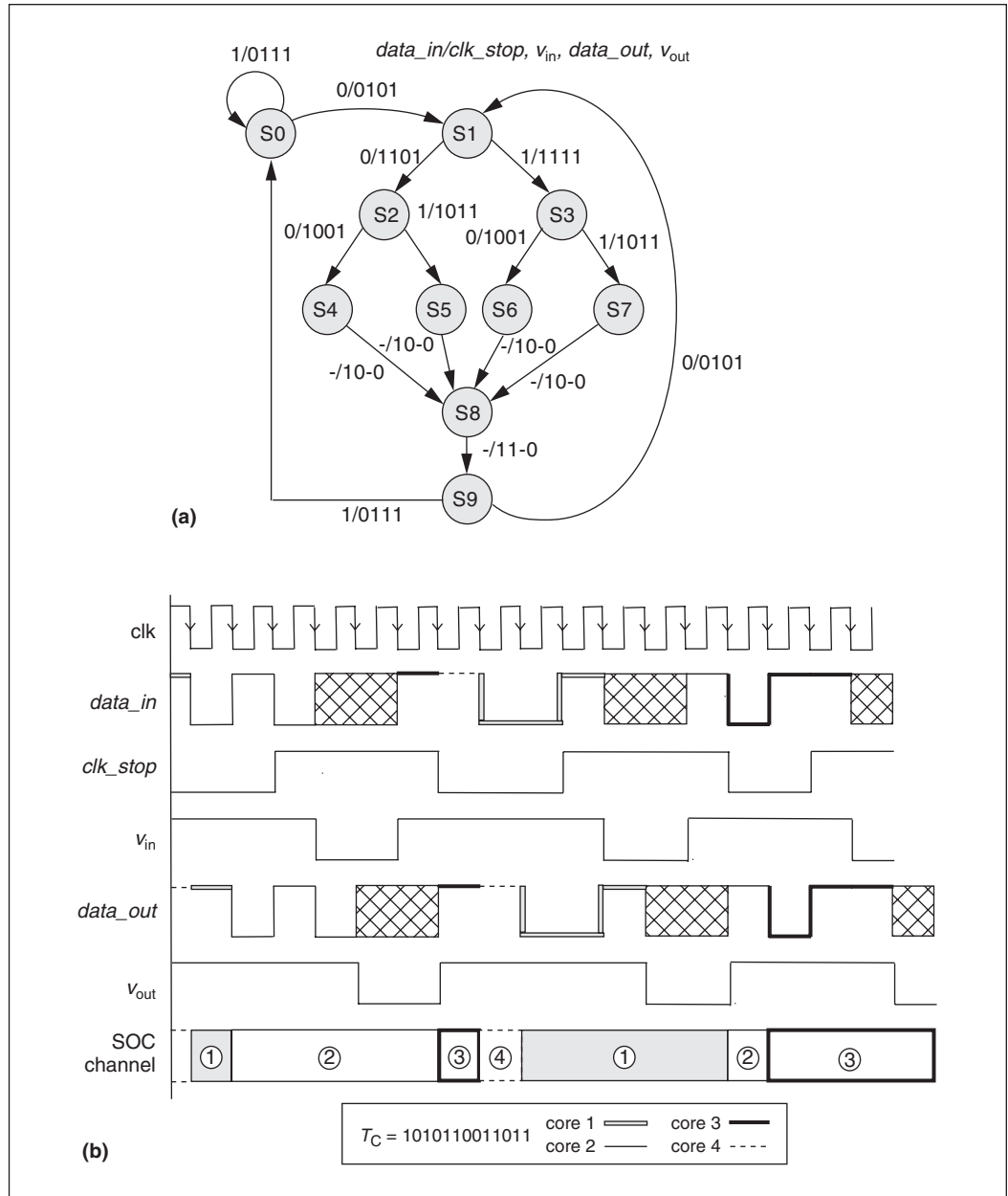


Figure 9. State diagram (a) and timing diagram (b) for the SOC channel selector FSM ( $m = 4$ ).

(NI denotes noninterleaved) required if we test all the cores one by one using a single ATE I/O channel:

$$\begin{aligned}
 T_{NI} &= \sum_{j=1}^m \left( \{n_{C,j} - r_j [1 + \log_2(m)]\} m \right) + (m+1)R \\
 &= m |T_C| - m \sum_{j=1}^m r_j \log_2(m) - \sum_{j=1}^m r_j + (m+1)R \\
 &= m |T_C| - mR \log_2(m) + R \\
 &= m |T_C| - R [m \log_2(m) - 1]
 \end{aligned}$$

where  $|T_C|$  denotes the number of bits in  $T_C$ . The difference between the interleaved and noninterleaved testing times is

$$\begin{aligned}
 T_{NI} - T_i &= m |T_C| - Rm \log_2(m) + R - n_{C,\max} + \\
 &\quad mr_{\max} [1 + \log_2(m)] - R - mR \\
 &= m (|T_C| - n_{C,\max}) - m [1 + \\
 &\quad \log_2(m)] (R - r_{\max}) \\
 &= m \{ (|T_C| - n_{C,\max}) - [1 + \log_2(m)] \\
 &\quad (R - r_{\max}) \} \\
 &\approx m (|T_C| - n_{C,\max}) \gg 0
 \end{aligned}$$

because  $n_{c,max} \gg r_{max}$  and  $T_c \gg R$ .

Consider a hypothetical example of four cores with an encoded test data size equal to  $n_{c,1} = 40$ ,  $n_{c,2} = 60$ ,  $n_{c,3} = 80$ ,  $n_{c,4} = 100$ , and  $r_1 = 4$ ,  $r_2 = 6$ ,  $r_3 = 8$ , and  $r_4 = 10$ . Therefore,  $n_{c,max} = 100$ ,  $r_{max} = 10$ ,  $m = 4$ ,  $R = 28$ , and  $|T_c| = 280$ . Finally,  $T_{NI} - T_i = 4[(280 - 100) - (1 + 2)(28 - 10)] = 504$ . This analysis shows that the interleaving architecture reduces testing time and increases ATE channel bandwidth.

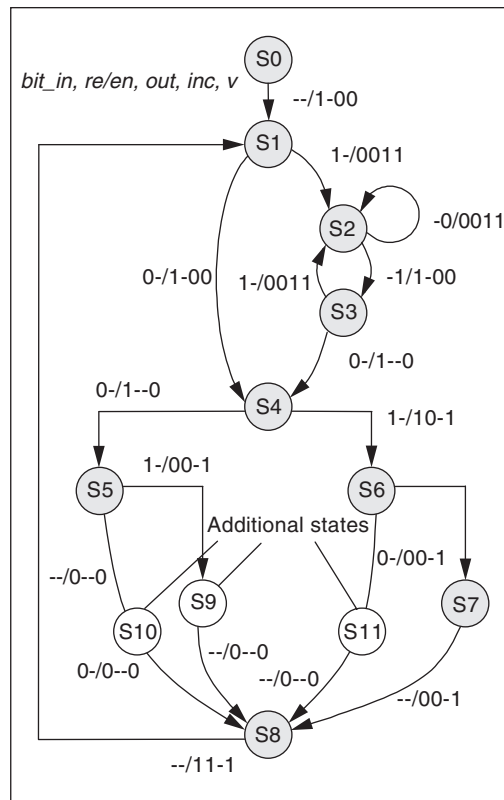
We developed a Verilog model of the FSM for  $m = 4$  and simulated it using  $T_c = 1010110011011$ . We also synthesized the gate-level circuit of the channel selector FSM with the Synopsys Design Compiler. It contains only four flip-flops and 17 gates. Thus, additional hardware overhead is small.

### Experimental results

We performed TRP on the large ISCAS benchmark circuits. We considered full-scan circuits for the proposed compression and decompression schemes. For full-scan circuits, we reordered patterns to achieve higher compression. For all full-scan circuits, we considered a single scan chain. We computed the compression percentage as  $C_p = (|T_D| - |T_E| / |T_D|) \times 100$ , where  $|T_D|$  is the test set size and  $|T_E|$  is the encoded test set size.

For our first experiment, we used difference-vector sequences ( $T_{diff}$ ) obtained from partially specified test sets (test cubes). Table 4 presents results for test cubes obtained using dynamic compaction with the Mintest ATPG program.<sup>9</sup> The table compares the fully compacted Mintest test sets with the compression obtained from FDR, Golomb, and conventional run-length coding. The table lists precomputed (original) test set sizes ( $T_D$ ), encoded test set sizes ( $T_E$ ), and smallest ATPG-compacted (Mintest) test set sizes. We used a Sun Ultra 10 workstation with a 333-MHz processor and 256 Mbytes of DRAM.

Table 4 shows that FDR codes provide better compression than Golomb and conventional run-length codes in all cases. (Golomb code results reported here are better than those reported in an earlier publication<sup>6</sup> because we used an improved pattern-reordering heuristic for these experiments.) For circuit s38417, the increase for FDR codes was as much as 7% over



**Figure 10. Modified state diagram of the decoder FSM equalizing tail and prefix decoding cycles.**

Golomb codes. In all but one case, the encoded test set ( $T_E$ ) size is much smaller than that of the Mintest-compacted test set.

The test cubes we used for s35932 were already highly compacted, so we didn't obtain high compression for this circuit. Nevertheless, in contrast to FDR codes, Golomb codes provided insignificant compression, and run-length codes provided no compression for this circuit. On average, the compression obtained with FDR codes was 7.49% higher than that obtained with Golomb codes and 19.56% higher than that obtained with conventional run-length codes. Test data compression always leads to encoded test sets smaller than ATPG-compacted test sets.<sup>6</sup> Moreover, test data compression decreases testing time by several orders of magnitude,<sup>10</sup> and substantially reduces power consumption during scan testing.

Table 5 demonstrates that using test cubes  $T_D$  (with all the don't-care bits mapped to 0s) also yields high compression. The advantage

Table 4. Compression obtained using various  $T_{diff}$  sequences.

Circuit	$T_D$ size (bits)	Run-length compression (%) for $b = 3$	Golomb compression (%)	FDR compression (%)	$T_E$ size with FDR (bits)	Mintest test set size (bits)
s5378	23,754	44.49	53.73	61.32	9,188	20,758
s9234	39,273	49.63	59.85	60.63	15,460	25,935
s13207	165,200	58.75	84.33	87.67	20,368	163,100
s15850	76,986	52.15	66.55	71.95	21,590	57,434
s35932	28,208	None	2.27	25.74	20,946	19,393
s38417	164,736	46.82	58.08	65.35	57,066	113,152
s38584	199,104	48.52	59.61	64.67	70,328	161,040

Table 5. Compression obtained using  $T_D$ .

Circuit	$T_D$ size (bits)	Run-length compression (%) for $b = 3$	Golomb compression (%)	FDR compression (%)	$T_E$ size with FDR (bits)	Mintest test set size (bits)
s5378	23,754	35.72	37.11	48.02	12,346	20,758
s9234	39,273	42.12	45.25	43.59	22,152	25,935
s13207	165,200	56.83	79.74	81.30	30,880	163,100
s15850	76,986	47.98	62.82	66.22	26,000	57,434
s35932	28,208	None	None	19.37	22,744	19,393
s38417	164,736	32.53	28.37	43.26	93,466	113,152
s38584	199,104	42.21	57.17	60.91	77,812	161,040

Table 6. Comparison between external clock frequency  $f_{ext}$  required for Golomb-coded test data, and external clock frequency  $f'_{ext}$  required for external testing with ATPG-compacted patterns (for the same testing time).

Circuit	$m$	$r$	$n_c$	$pn$	$f'_{ext}/f_{ext}$
s9234	4	5,039	22,250	25,935	1.93
s13207	16	6,716	41,658	163,100	9.90
s15850	4	8,702	40,717	57,434	2.25
s38417	4	20,165	92,054	113,152	1.99
s38584	4	23,320	104,111	161,040	2.54

of using  $T_D$  for compression is that the decompression architecture for on-chip pattern generation doesn't require a separate CSR. For circuits with long scan chains, equally long additional CSRs increase hardware overhead significantly. Therefore, compressing  $T_D$  to generate the encoded test set not only yields smaller test sets but also reduces hardware overhead.

Table 6 shows that Golomb coding lets us use a slower tester without incurring a time penalty. In comparison with external testing using ATPG-compacted patterns, we achieved the same testing time using a much slower tester.

Overall, our experimental results for the ISCAS benchmarks show that the compression technique is very efficient for full-scan circuits and that ATPG compaction is not always necessary to save ATE memory and reduce testing time.

**TEST DATA COMPRESSION** offers a solution to the TRP problem for SOC designs. We are currently working on reduced-pin-count-testing (RPCT) and BIST techniques using test data compression. ■

## Acknowledgments

This research was supported in part by National Science Foundation grant CCR-9875324 and in part by an Intel equipment grant.

## ■ References

1. A. Raghunathan and S.T. Chakradhar, "Acceleration Techniques for Dynamic Vector Compaction," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 310-317.
2. S. Bommu, S.T. Chakradhar, and K.B. Doreswamy, "Static Compaction Using Overlapped Restoration and Segment Pruning," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 140-146.
3. 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.*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 289-294.
4. V. Iyengar, K. Chakrabarty, and B.T. Murray, "Deterministic Built-in Pattern Generation for Sequential Circuits," *J. Electronic Testing: Theory and Applications (JETTA)*, vol. 15, Aug.-Oct. 1999, pp. 97-115.
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.*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 458-464.
6. 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, Mar. 2001, pp. 355-368.
7. A. Chandra and K. Chakrabarty, "Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression," *Proc. IEEE VLSI Test Symp.*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 42-47.
8. D. Heidel et al., "High-Speed Serializing/De-serializing Design-for-Test Methods for Evaluating a 1 GHz Microprocessor," *Proc. IEEE VLSI Test Symp.*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 234-238.
9. I. Hamzaoglu and J.H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 283-289.
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 01) Conf.*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 145-149.



**Anshuman Chandra** is a PhD candidate in electrical and computer engineering at Duke University. His research interests include VLSI design, digital testing, and computer architecture. Chandra has a BE in electrical engineering from the University of Roorkee, Roorkee, India, and an MS in electrical and computer engineering from Duke University. He is a student member of the IEEE and ACM SIGDA.



**Krishnendu Chakrabarty** is an assistant professor of electrical and computer engineering at Duke University. His research interests include system-on-a-chip testing, embedded real-time operating systems, distributed sensor networks, and architectural optimization of microelectrofluidic systems. Chakrabarty has a BTech from the Indian Institute of Technology, Kharagpur, and an MSE and a PhD from the University of Michigan, Ann Arbor, all in computer science and engineering. He is a senior member of the IEEE and a member of the ACM SIGDA and Sigma Xi. He is the vice chair of technical activities in the IEEE Computer Society's Test Technology Technical Council.

■ Direct questions or comments about this article to Anshuman Chandra, Duke University, Dept. of Electrical and Computer Engineering, 130 Hudson Hall, Box 90291, Durham, NC 27708; [achandra@ee.duke.edu](mailto:achandra@ee.duke.edu).

**For further information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.**