RL-Huffman Encoding for Test Compression and Power Reduction in Scan Applications

Mehrdad Nourani
and
Mohammad H. Tehranipour
Center for Integrated Circuits & Systems,
The University of Texas at Dallas, Richardson, TX 75083
{nourani,mht021000}@utdallas.edu

Categories and Subject Descriptors: B.7.3 [**Integrated Circuits**]: Reliability and Testing—*Test Compression*; B.7.m [**Integrated Circuits**]: Miscellaneous—*Power Reduction in Scan Application*; B.6.2 [**Logic Design**]: Reliability and Testing—*Test Compression*; B.6.m [**Logic Design**]: Miscellaneous—*Power Reduction During Test*

General Terms: Test Compression, Power Reduction, Scan Applications

Additional Key Words and Phrases: Compression Ratio, Decompression, Huffman Encoding, Run-Length Encoding, Scan-in Test Power, Switching Activities, Test Pattern Compression.

Abstract– This paper mixes two encoding techniques to reduce test data volume, test pattern delivery time and power dissipation in scan test applications. This is achieved by using Run-Length encoding followed by Huffman encoding. This combination is especially effective when the percentage of don't cares in a test set is high which is a common case in today's large SoCs. Our analysis and experimental results confirm that achieving up to 89% compression ratio and 93% scan-in power reduction is possible for scan testable circuits such as ISCAS89 benchmarks.

1. INTRODUCTION

Design for testability (DFT) based on scan and automatic test pattern generation (ATPG) is a reliable technique to achieve high fault coverage. Unfortunately, for large circuits the number of scan cells and test patterns become huge. This growth can be especially seen in today's system-on-chips (SoCs). For large SoCs, test data volume increases which in turn increases test cost due to rise in test time and increased memory requirement.

Power dissipation is an important factor in today's chip design. Power dissipation in CMOS circuits is proportional to the switching activity in the circuit [1]. During normal operation of a circuit, often a small number of flip flops change values in each clock cycle. However, during test operation, large numbers of flip flops switch, especially when test patterns are scanned into the scan chain. Compacting the test set often requires replacing (mapping) don't cares with specified bits '0' or '1' [2]. This process may increase

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works, requires prior specific permission and/or a fee.
© 1995 by the Association for Computing Machinery, Inc.

switching activity of scan flip flops and eventually the scan-in power dissipation. There are usually plenty of don't cares in test patterns generated for scan which provides an opportunity for compression and power reduction.

1.1 Prior Work

•**Test Data Volume Reduction:** There are several techniques to address ATE-SoC interaction problems in terms of test data volume and application time. Built-in self-test (BIST) methodology reduces the need for an expensive ATE [3]. In BIST, on-chip pseudo-random pattern generators and signature compressors are used. In practice, pseudo-random BIST cannot replace other test methods, because, especially for large chips, the existence of random pattern resistant faults may lead to unacceptably long test time. To overcome these difficulties, deterministic test patterns need to be transferred from the ATE to the SoC under test. Several methods have been reported to reduce test volumes stored in the tester's memory [4][5][6][7].

Compression techniques are used to speed up the ATE-SoC interaction during test. A data compression and decompression architecture for testing embedded cores in SoCs using Golomb coding [8] is presented in [9]. A variable-length compression coding is presented in [10] which carefully considers distributions of 0's in a test sequence. Reference [11] addresses alternating run-length codes followed by frequency-directed run-length (FDR) coding for test data compression. A simultaneous reduction in volume of test data and power dissipation to generate minimum transition count (MTC) is presented in [12] [13]. Variable-length input Huffman compression (VIHC) method [14] uses a maximum acceptable length to improve compression ratio, area overhead and test application time. Authors in [15] show a compression/decompression based on Huffman coding of fixed-length blocks to reduce test data. The Illinois scan architecture provides a mechanism to reduce test application time and data storage requirements [16]. An embedded deterministic test technology for a low cost test that reduces scan test data volume and scan test time is presented in [17].

Reducing the number of scan chain input pins fed by the ATE using an on-chip decoder is proposed in [18]. Reseeding the Linear Feedback Shift Register (LFSR) [19] is another method that finds one or multiple seeds for an on-chip LFSR for every test vector. The generated bit sequence for LFSR matches the test vector at specified bit positions. The size of LFSR is significantly smaller than the scan chain. Reduction of scan test data in designs with multiple scan chains using a combinational decoder is presented in [20]. The basic approach proposed in [21] is a fixed packet-based compression in which don't cares are filled appropriately in the test vectors such that they need not be stored in the ATE memory. Reordering test patterns is used to reduce the overall test application time [22] [23]. LZ77 [24] and LZW [25] methods use dictionary-based algorithms to compress the input test set with a large number of don't cares. A hybrid coding strategy, using a combination of run-length and dictionary-based methods, is proposed in [26] to improve compression ratio.

•**Scan Power Reduction:** The excessive switching activity during scan tests may cause larger peak and average power dissipation than those during normal operation. Therefore, the power constraint of the circuit needs to be considered in test mode. Applying scan test vectors generated by an ATPG is very time consuming and thus compacting test vectors is unavoidable. Being a feature of the test set, overall power dissipation of cores remain the same. However, the compression/decompression procedure may intensively increase

the power dissipation of scan elements in the chain. This is often called the scan-in and scan-out power consumption of the chain.

Several techniques have been proposed to reduce power consumption during test application. Test vector ordering [27], gated clock scheme [28], scan latch partitioning [29], test generation for low power scan testing [30], static compression to reduce power [2], mixed compression/decompression and low power test application techniques based on Golomb codes [31], FDR codes [11] and MTC [12] [13] are among the proposed techniques.

1.2 Main Contribution and Paper Organization

In this paper we present a compression/decompression technique to reduce test data volume, test application time and switching activities in scan cells. The proposed technique combines two well known methods, Run-Length (RL) and Huffman encodings [32]. Essentially, RL encoding performs variable-length grouping to utilize the don't cares for: 1) minimizing bit-transition and 2) compressing data by sending the length of running (similar) bits. Huffman encoding further enhances compression. While scan power reduction is not the main focus of this paper, creating the minimum bit-transition by RL has a positive effect in reducing scan-in power consumption of the scan components. The compressed test data is scanned in and decompressed by an inexpensive on-chip decoder to generate the exact test pattern which is finally scanned into the scan chain.

Our work is similar to [11] [12] in using Run-Length encoding and to [14] in using Huffman encoding. However, our technique is more flexible by: 1) dealing with large runs of 0's or 1's effectively, 2) filling don't cares to maximize occurrence frequency and thus compression ratio and 3) being able to tradeoff between compression ratio and decoder cost.

This paper is organized as follows. Section 2 describes the proposed compression technique. In Section 3, we explain the analytical foundation of our compression technique. The decompression method and architecture are discussed in Section 4. Power analysis is presented in Section 5. The experimental results are shown in Section 6. Finally, Section 7 concludes the paper.

2. RL-HUFFMAN COMPRESSION TECHNIQUE

Typically, an ATPG generates test patterns in several steps [33]. First, it generates test patterns by pseudo-random pattern generation. When the fault coverage does not increase by generating more random patterns, it stops. The second step is to generate deterministic test patterns to detect all leftover random pattern resistant faults to achieve higher fault coverage. There are often large numbers of don't cares in each test pattern. The ATPG may use don't care bits to compact (collapse) test vectors. Random bits are sometimes assigned to don't cares in test patterns to detect non-modeled faults. In most CAD tools, the user has options for filling the don't cares. For example, in Synopsys' Tetramax [33], the *"norandomfill"* option can be used to leave don't cares unchanged in the generation phase.

2.1 Step 1: Run-Length Encoding

In today's SoCs, the percentage of don't cares can be quite high, sometimes more than 80% for large circuits [24]. IBM reported 98% of bits in test patterns for some of their designs were filled with don't cares [34]. In our compression technique we take advantage of the presence of don't cares in test vectors. Our technique uses a variable encoding of

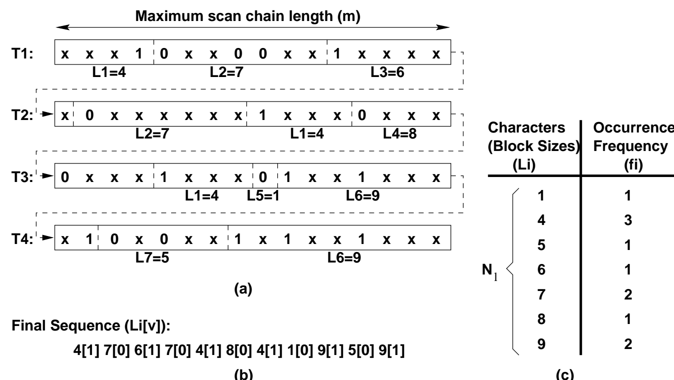


Fig. 1. Applying Run-Length algorithm to a small example.

test vectors. The basic idea in RL encoding is a careful replacement of don't cares with '0' or '1' to find the minimum number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. Minimizing such transitions also has a significant impact in reducing dynamic switching activities of test components (i.e. scan cells) to which the vectors are sent.

Figure 1 shows an example of filling don't care bits with '0' or '1' based on RL strategy. As shown, the filling process starts from the most significant bit of vector sequence (e.g. T_1). The don't cares are filled with 1's until we reach a '0'. In the next step, don't cares are filled with 0's until the next '1'. This process continues until it replaces all don't care bits with '0' or '1'. In this example, the number of transitions is ten and the test set is partitioned into eleven blocks (N_b=11) each of which is filled with only 1's or 0's. L_i shows the length of blocks in the test set. The corresponding block lengths are shown in Figure 1(b). Block lengths are shown using $L_i[v]$ notation, where L_i is block length and $[v]$ is bit value '0' or '1' that is filled in that block. The characters are stored in the lookup table for decoding purpose. As shown, the number of characters (N_l) stored in the lookup table is seven. Figure 1(c) summarizes the occurrence frequency (f_i) of these characters. In this example, no limit on the block size is defined and we simply replace don't cares with 0/1 values to maximally enlarge a block. More sophisticated approaches can be devised for other objectives such as minimizing N_l or increasing f_i values. Some techniques are discussed in Section 3.4.

Figure 2 shows RL encoding algorithm, where n and m are the number of test patterns and the scan chain size, respectively. Again, we did not define any limit on the block size. In the next section we will elaborate further on the role of maximum block size. In RL algorithm T_j refers to the jth bit in a test sequence (combined patterns) when it is checked to form the blocks. In our method, within each block we have only 0's or 1's. If RL is the only compression step, instead of scanning actual data we can scan the length of that block showing how many bits in the block are '0' or '1'. Most of the other techniques focus on probability of the occurrence of fixed-length blocks which may include '0', '1' or 'x' (don't care) [15][21]. In our case, the length of blocks (L_i values in Figure 1) are encoded in the first step of compression. Note carefully that, as lines 9-12 in Figure 2 indicate, the algorithm always generates alternating 0-blocks and 1-blocks. Lines 15-16 store the final block size and frequency in a test set. Therefore, a single bit T_0 to start alternation is sufficient and the values of each block (e.g. [v] in Figure 1(b)) are not explicitly sent.

```
RL Encoding()
Input: Test set {T}
Output: L_i and f_i, i ≥ 1

01:T_0=First specified bit;
02:i=1;
03:for (j=1 to n·m)
04:{
05:    if (T_j = T_0 or T_j ='x')
06:        i++;
07:    else
08:    {
09:        L_i = i;
10:        f_i++;
11:        T_0=~T_0;
12:        i=1;
13:    }
14:}
15:L_i = i;
16:f_i++;
```

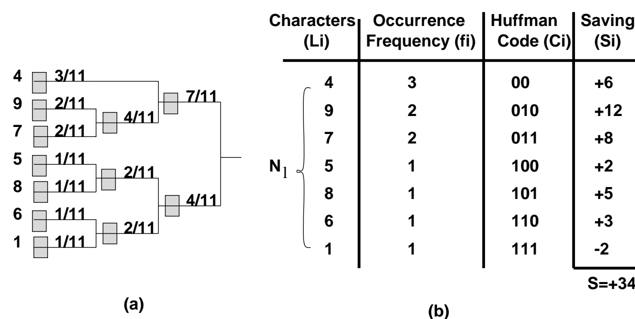
Fig. 2. RL encoding algorithm.

2.2 Step 2: Huffman Encoding

To get the best compression rate in the second step we also apply Huffman coding (a variable-length encoding by nature) [35] to encode block length values (characters). The idea is to assign a smaller number of bits to codewords that occur most frequently and a larger number of bits to those that occur less frequently. Huffman codes are obtained by constructing a Huffman tree. Figure 3(a) shows the Huffman tree for the example of Figure 1(c) with the occurrence frequency annotated. The generated Huffman code is shown in Figure 3(b). Clearly, blocks with higher occurrence frequencies will get shorter codewords. For this example, instead of sending 64 bits of original data, the ATE sends only 64-34=30 bits. This is a 53.12% overall saving in test data volume. The saving in transfer time depends on working frequencies of the ATE and SoC under test. In Section 4 we will express the upper bound of transfer time saving based on compression saving. The last column in Figure 3(b) shows saving of data sent into the chip. Instead of sending L bit 0's or 1's, the l bit codeword is sent. Saving for the ith codeword is $L_i - l_i$, where l_i is the length of the ith codeword (C_i). Saving for sending the ith codeword with frequency f_i is $S_i = f_i(L_i - l_i)$. Total saving for all codewords is $S = \sum_{i=1}^{N_l} f_i(L_i - l_i)$. Then, the compression ratio (percentage of data reduction) for n vectors sent to a scan chain of m cells (overall $n \cdot m$ bits) will be:

$$CR = \frac{S}{n \cdot m} = \frac{\sum_{i=1}^{N_l} f_i(L_i - l_i)}{\sum_{i=1}^{N_b} L_i}$$

Eventually, the RL-Huffman code is scanned into the chip. An on-chip decoder decompresses the encoded bits to recognize how many 0's or 1's need to be shifted into the scan chain. In general, the Huffman tree is needed not only for encoding but also for decoding. To make decoder independent of the test set, the information of this tree can be transmitted with the compressed test data. However, this approach may result in higher cost and lower compression ratio.



Characters (Li)	Occurrence Frequency (fi)	Huffman Code (Ci)	Saving (Si)
4	3	00	+6
9	2	010	+12
7	2	011	+8
5	1	100	+2
8	1	101	+5
6	1	110	+3
1	1	111	-2
			S=+34

Fig. 3. Huffman code for the example in Figure 1.

Our decompression technique is quite straightforward and will be discussed in details in Section 4. Also in Section 5 we will show the positive impact of RL encoding on minimizing the bit transitions, and hence, power consumptions of the scan cells. Note that Huffman encoding provides further reduction and does not deteriorate the minimum transition property achieved by RL encoding.

2.3 Comparing to Other Techniques

To clarify why RL-Huffman encoding is so efficient, we compare our technique with techniques using FDR code [11], statistical coding [15] and VIHC [14]. Figure 4 shows a comparison between these techniques for one 20-bit test pattern.

In FDR code, don't care bits are replaced with 0's and 1's. The blocks may have different lengths but they use a fixed code to make the decoder independent of test set. Such replacement and code reduce the cost of a decoder [11] but the highest compression ratio may not be achieved (see Figure 4(a)). The statistical coding method proposed in [15] uses a fixed length coding combined with Huffman coding based on the occurrence frequency of codewords. Figure 4(b) shows the result of this method for 4-bit lengths. But, the chance of having identical fixed-length blocks (and thus occurrence frequency) will be decreased when the size of blocks increases. In VIHC method [14] a maximum acceptable length of runs of 0's is defined ($m_h = 4$ in Figure 4(c)). The test vector is divided into runs of 0's of lengths smaller or equal to m_h and then coded using Huffman encoding.

The final compressed data in our technique is shorter than other techniques as shown in Figure 4(d). Note carefully that in the conventional RL coding we need to send both the characters and their lengths, which means 9[1]5[0]6[1] (i.e. nine 1's, five 0's and six 1's) should be sent for pattern in Figure 4(d). However, in our method we guarantee alternating transitions between a 0-block and a 1-block and therefore the value of '0' and '1' are not explicitly needed to be sent along with the code. In section 4 we will explain how a decoder takes this property into account and easily generates this alternation. Our method is similar to the method presented in [14] in using a combination of RL and Huffman encoding. However, the application techniques are different. Specifically, in dealing with large block sizes, being able to target the occurrence frequency to maximize compression and tradeoff between the compression ratio and the decoder cost differentiate our method from similar techniques.

We acknowledge that comparing methods with only one pattern may not be fair as the average compression ratio for large test sets matters. This example is presented to differ-

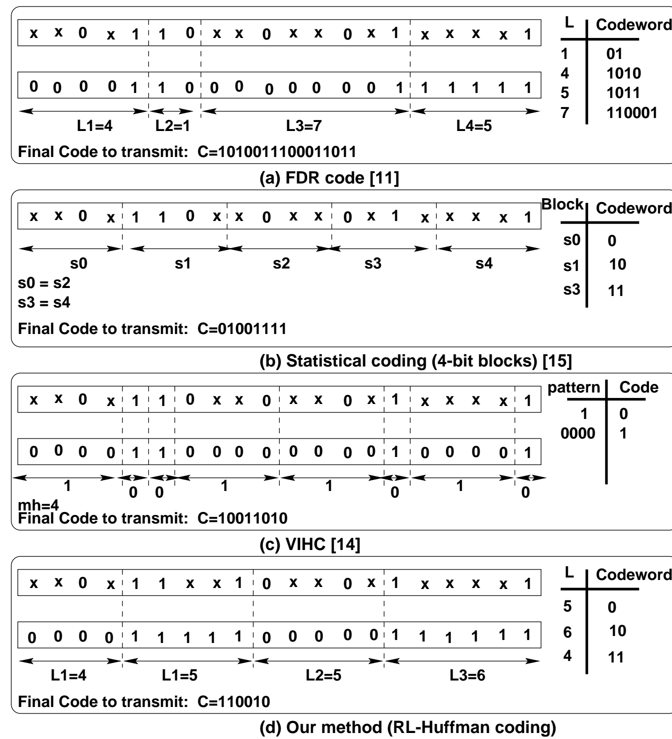


Fig. 4. Comparison between several techniques.

entiate our approach from others. In Section 6 we will show that the compression ratio of our method is also quite good for large test sets.

3. ANALYSIS OF RL-HUFFMAN COMPRESSION TECHNIQUE

RL encoding takes advantage of don't cares by creating blocks of identical data values. The Huffman code is a variable length code that can do much better than a fixed-length code in terms of compression [32]. Huffman coding can be done in $O(nlogn)$ and therefore is not very time consuming [41]. Additionally, no codeword is prefix to the others, a property that makes decoding easy and inexpensive. Huffman coding gives frequent and infrequent characters short and long codewords, respectively.

3.1 Compression Efficiency

The lower bound of the average length of a codeword, used for encoding an information source, can be expressed by its *entropy* [32]. The entropy of a test set with N_l unique characters is given by:

$$H = -\sum_{i=1}^{N_l}(p_i \cdot log_2 p_i)$$

where p_i is the occurrence probability of character L_i. As the above formula shows, entropy is independent of character lengths. The average length of a codeword is given by:

$$l_{avg} = \sum_{i=1}^{N_l} p_i \cdot l_i$$

where l_i is the codeword length for the character L_i. For a general compression technique we have $l_{avg} \geq H$ and efficiency of the compression technique is expressed as:

$$E = l_{avg}/H$$

The closer E is to 1, the more efficient is the compression technique. The average length of an Huffman codeword is the closest to the entropy of a test set [32]. In the example of Figure 1, $H = 2.663$, $l_{avg} = 2.727$ and $E = 1.023$.

3.2 Compression Ratio

The overall compression ratio depends on saving that RL (S_{RL}) and Huffman (S_H) encodings achieve in the first and second step, respectively. Briefly, we use this formula in computing the overall compression ratio (CR):

$$CR = \frac{S_{RL} + S_H}{\sum_{j=1}^{N_b} L_j}$$

More specifically, the compression ratio achieved in the first step (RL) will be: $CR_{RL} = S_{RL}/\sum_{j=1}^{N_b} L_j$. The compression ratio in the second step (Huffman) with respect to the result of the first step can be computed as: $CR_H = S_H/(\sum_{j=1}^{N_b} L_j - S_{RL})$. S_{RL} metric depends on the number and distribution of x's and also on the maximum block size (K). Briefly, in the RL step instead of K-bit data, we send only $\lceil log_2 K \rceil$ bits. Therefore, if maximum block size (maximum Run-Length) is K and there are N_b characters to transmit:

$$S_{RL} = \sum_{j=1}^{N_b} L_j - (\lceil log_2 K \rceil \times N_b)$$

where $\sum_{j=1}^{N_b} L_j = n \cdot m$. In the above formula N_b bits need to be added to $\lceil log_2 K \rceil$ if '1' and '0' (value for each block) need to be sent. In our method we don't include these values because alternating 0-blocks and 1-blocks are guaranteed. For the example of Figure 1(b) we have: $S_{RL} = 64 - 4 \times 11 = 20$.

Since we use Huffman coding in the second step, the compression ratio of this step depends on the occurrence frequency of characters. The characters in our method are L_i values. Total length (bits) of data will be $B = \sum_{i=1}^{N_l} (f_i \times l_i)$, where N_l is the number of distinct characters (different L_i values), f_i and l_i are occurrence frequency and length of codeword C_i, respectively.

Using fixed-length binary coding, we have $l_i = \lceil log_2 N_l \rceil \; \forall i$. Therefore, for a total number of characters in the data set $N_b = \sum_{i=1}^{N_l} f_i$ we have:

$$B_{fixed-length} = \lceil log_2 N_l \rceil \times \sum_{i=1}^{N_l} f_i = \lceil log_2 N_l \rceil \times N_b$$

It has been shown that for Huffman coding, the average length of codewords (l_{avg}) satisfies this inequality: $H \leq l_{avg} \leq H + 1$, where $H = -\sum_{i=1}^{N_l} [(f_i/N_b) \times log_2(f_i/N_b)]$ [32]. Note

that f_i/N_b shows the occurrence probability of codeword C_i in the data set. Therefore:

$$B_{Huffman} \leq \{1 - \sum_{i=1}^{N_l}[(f_i/N_b) \times log_2(f_i/N_b)]\} \times N_b$$

The upper bound of the Huffman compression ratio, compared to the fixed-length coding will be:

$$\begin{aligned} CR_H &\leq \frac{B_{fixed-length} - B_{Huffman}}{B_{fixed-length}} \\ &\leq \frac{\lceil log_2 N_l \rceil - \{1 - \sum_{i=1}^{N_l}[(f_i/N_b) \times log_2(f_i/N_b)]\}}{\lceil log_2 N_l \rceil} \end{aligned}$$

Researchers have shown that CR_H is in the range of 20% to 90% depending on the occurrence frequencies [41]. In our example of Figure 3: $S_H = 4 \times 11 - 30 = 14$ and overall $CR = \frac{S_{RL}+S_H}{\sum_{j=1}^{N_b} L_j} = \frac{20+14}{64} = 53.12\%$.

3.3 Maximum Run-Length (K)

The overall compression ratio depends on distribution of don't cares and occurrence frequencies of characters to be transmitted. Additionally, the maximum block size (K) affects number of characters (N_l), number of blocks (N_b) and indirectly occurrence frequencies (f_i). More specifically, in our approach we need to choose K such that a reasonable probability of having a block of that size exists in a test data set. Very small and very large values for K both hurt the compression ratio. Intuitively, if we choose a very large K, the probability of having that many 0's or x's (or that many 1's and x's) running through K consecutive bits may be very small and thus we will not get high frequencies. On the other hand, if we choose K to be very small, distribution of frequencies will be close to uniform and we will not get that much compression.

Note that there is no magic number for K. K can be chosen more efficiently based on the amount of don't cares in test data set. Suppose, $p(0)$, $p(1)$ and $p(x)$ show the probability of being '0', '1' and 'x' when one bit is chosen from the data set, respectively. Obviously, these probabilities depend on the total number of 0's, 1's and x's in a set and can be computed a priori. Let's assume $p(x) = D$ and $p(0) = p(1) = (1-D)/2$, where D is the ratio of don't care bits to the total bits in a test set. The probability that a K-bit block is formed can be computed [39]:

$$\begin{aligned} p_K(D) &= p(K\ bits \in \{0,x\}) + p(K\ bits \in \{1,x\}) \\ &\quad - p(K\ bits \in \{x\}) \\ &= [p(0)+p(x)]^k + [p(1)+p(x)]^k - [p(x)]^k \\ &= 2^{1-K}(1+D)^K - D^K \end{aligned}$$

In Figure 5, $p_K(D)$ is drawn for various D% (don't care percentage) values between 60% to 90%. As K increases, the probability of having such block will be smaller. Due to the complicated relationship of factors and unavailability of frequencies, no magical K can work for all data sets. However, for a given data set , the ratio of don't care D can be computed and an appropriate K can be chosen such that the probability remains reasonable. A large value of K hurts RL encoding because for a fixed D, $p_K(D)$ becomes

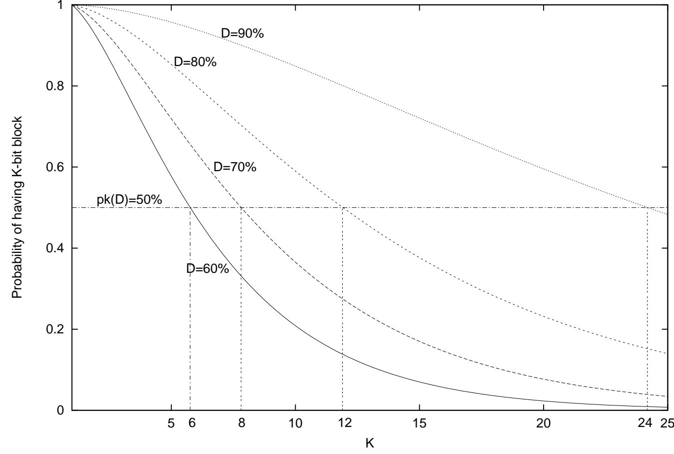


Fig. 5. Probability of having K-bit block (p_K) as a function of $D\%$.

small. But it helps Huffman encoding as a large K implies larger saving. Similarly, a small value of K makes Huffman encoding ineffective because the block size distribution becomes almost uniform. But it helps RL encoding as $p_K(D)$ will be higher. Optimizing K is beyond the scope of this paper. Empirically, we observed that choosing K such that the probability of having a block size of K (or very close to K) remains in the range of 0.4 to 0.6 produces a good compromise between the compression ratio and the decoder cost. In such situation $CR\%$ is often upper bounded by $D\%$. Some experimental evidence will be shown in Section 6.

3.4 Techniques to Tradeoff Compression Ratio and Decoder Cost

When RL encoding is applied, the length of blocks may vary from 1 to $n \cdot m$. Therefore, the decoder needs to decode all codewords between 1 and $n \cdot m$. For large circuits, the scan chain is too long, may be around a couple of hundreds. In such cases, the occurrence frequencies of f_i may not be too high because it is distributed from 1 to $n \cdot m$ and all block lengths between 1 to $n \cdot m$ may happen. The advantage of a long block is that a large amount of data is sent by a very small codeword. The disadvantage is that it needs a more costly decoder. In our technique, we use the maximum run length (K) to tradeoff complexity of the decoder and compression ratio.

We have investigated three techniques of which the first two limit block sizes to a predefined number K and the third technique has no limit on block size. The compression algorithm, shown in Figure 2, needs to be slightly revised to limit block sizes to a pre-defined value K. Our experimentation on tradeoff between K, compression ratio and decoder cost will be reported in Section 6.

- **Technique 1:**

$$\forall L_i : L_i > K \Rightarrow L^*_{i1} + L^*_{i2} = L_i \ni \begin{cases} L^*_{i1} = K \\ L^*_{i2} = L_i - K \end{cases}$$

In Technique 1 if the size of a block (L_i) is greater than K, it is divided into two new blocks (with the size of K and $L_i - K$) and the process continues until no block size is greater

than K. A 0 (blank) character is inserted between the new blocks. Obviously, zero length never happens for actual data blocks in our technique. The 0 character is used to guarantee alternation between 0's and 1's for consecutive blocks. This feature reduces the decoder cost with little sacrifice of the compression ratio.

- **Technique 2:**

$$\forall L_i : L_i > K \Rightarrow L^*_{i1} + L^*_{i2} = L_i \ni$$
$$S^*_{i1} + S^*_{i2} = \max_{L^*_{i1}+L^*_{i2}=L_i} \{S_{i1} + S_{i2}\}$$

Technique 2 revisits the blocks and if a block size is larger than a given K, decomposes it such that the break increases the occurrence frequencies of other characters to achieve maximum saving. Blank character (0) is still required to be added between each two blocks to ensure alternating 0-blocks and 1-blocks. This technique keeps the size of the decoder proportional to K, but may enhance the compression ratio.

- **Technique 3:**

$$\forall L_i : L_i > K \Rightarrow L^*_{i-1} + L^*_i + L^*_{i+1} = L_{i-1} + L_i + L_{i+1} \ni$$
$$S^*_{i-1} + S^*_i + S^*_{i+1} = \max_{L^*_{i-1}+L^*_i+L^*_{i+1}=L_{i-1}+L_i+L_{i+1}} \{S_{i-1} + S_i + S_{i+1}\}$$

In Technique 3, blocks generated by RL Encoding algorithm are revisited to find the minimum number of blocks. Reducing the number of blocks reduces the entropy of a test set which eventually increases saving and compression ratio. Moreover, the cost of a decoder is reduced since less number of states need to be handled by the decoder. No blank character needs to be added to the final sequence. This technique uses a graph-based heuristic for entropy optimization and is more complicated than Techniques 1 and 2. Details of these techniques are beyond the scope of this paper and can be found in [39].

Figure 6 shows these three techniques applied to a small example. Figure 6(a) is the original RL-Huffman with no limit on K and no block size adjustment. Figure 6(b) shows the length of blocks computed with $K = n \cdot m = 48$ using original RL technique. In this case, the lookup table size, which indicates decoder cost, is proportional to 6 entries and the compression ratio is 62.50%. Assuming K=9, Figure 6(c) shows the first technique. Length 11[0] is decomposed to 9[0]0[1]2[0] and length 10[1] is decomposed to 9[1]0[0]1[1]. The lookup table size is 7 and the compression ratio is 37.50%.

Figure 6(d) shows the second technique for K=9. The length 11[0] is decomposed to 7[0]0[1]4[0] and 10[1] is decomposed to 7[1]0[0]3[1]. The lookup table size is 5 and the compression ratio is 47.91%. Overall, Technique 2 produces better results due to the increasing of occurrence frequencies of leftover characters while reducing the lookup table size. Figure 6(e) shows the best results based on Technique 3. In this case, 11[0]3[1] changes to 10[0]4[1] making that the frequencies of blocks 10 and 4 increase. The final sequence is shorter than other techniques, compression ratio is CR=72.92% and lookup table size is 4.

4. DECOMPRESSION

Huffman codes are prefix-free [41]. This is an important property compared to other coding techniques used for compression and it significantly simplifies the decoding process.

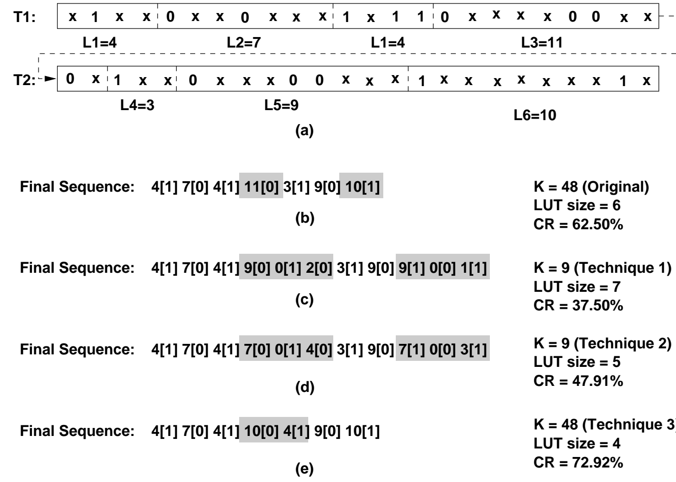


Fig. 6. Techniques to tradeoff compression ratio and decoder cost.

In Huffman code, the decoder easily recognizes the end of each codeword. An on-chip decoder at the serial input of the SoC under test is used to decompress test vectors. The decoder decodes input codes to recognize how many 1's or 0's need to be shifted into the scan chain. The test vectors can be shifted into the scan chain with a higher rate, e.g. at the SoC clock speed. The ATE sends the compressed data and clock for synchronization to the decoder. In our method the compressed data is a stream of variable-length (Huffman) codewords corresponding to the block lengths.

4.1 FSM-Based Decoder

Several Huffman decoders have been proposed in recent years which are independent of the circuit and data set [36] [37] [38]. The drawback, however, is that these decoders are expensive. To make decoder independent of the test set, the information of the Huffman tree can be transmitted with the compressed test data. However, such overhead has adverse effect on the compression ratio. Moreover, such generic implementation is still expensive. In this paper we propose using an inexpensive decoder that is test set-dependent. Dependency of the decoder on the test data is not desirable in general but can be well justified by its low cost and generic architecture. Empirical results are shown in Section 6.

The block diagram of one possible implementation of a decoder is shown in Figure 7. The compressed data (input codeword) come from the ATE to Huffman FSM where the input codewords are decoded. The decoded code addresses a small lookup table to find the block length L_i that indicates the number of 1's or 0's required to be shifted into the scan chain. The other outputs of the FSM are *sel* (to shift appropriate 0 or 1 into the scan chain), *load* (to enable of the counter) and *stop* (to stop the ATE when the counter has not finished its job while the FSM has generated a new address).

When the FSM generates an address pointer (AP), it enables *load* and the counter starts counting. At the same time *sel* ('0' or '1') is shifted into the scan chain through an open buffer. The FSM receives the new codeword for decoding. When the value of the counter and the output of the lookup table L_i become equal, output of the counter (ripple carry out (rco)) becomes 1 and shifting *sel* value is stopped. It also signals the FSM to put the next

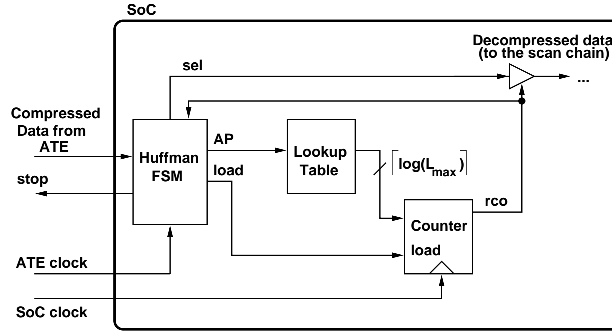


Fig. 7. RL-Huffman decoder architecture.

address pointer on the input line of lookup table. If the FSM generates the address before receiving *rco* from the counter, it stops the ATE with *stop* signal and waits for *rco*. When it receives *rco=1* signal, it puts the new address on the output and deactivates the *stop* signal (*stop=0*). Every time that a new AP is provided, a T flip-flop inside the FSM toggles to generate alternating blocks of 0's and 1's on the *sel* line.

4.2 CAM-Based Decoder

The decoder architecture of Figure 7 is one of many ways to implement the decoder. We would like to comment that other architectural styles are also efficient because the size of the decoder is quite small. In particular, Content Addressable Memory (CAM) architecture can be used instead of the FSM-Table pair to translate the codeword l_i to the block length L_i. Various efficient implementations of CAM have been proposed so far [42]. For large CAMs, the cost and power consumption of the large priority encoder inside the unit are matters of concern [42]. However, for applications such as ours that use small CAMs (up to a few hundred entries), these issues will not be problematic. In Section 6 we show the results of FSM-based implementation only. The CAM implementation of a few RL-Huffman decoders that we tried were only 7 to 12% more costly than their FSM-based counterparts [39].

4.3 Test Time Reduction

Reducing overall test application time is the ultimate goal of test pattern compression. In general, the amount of time reduction depends on the compression ratio and decompression method. In this section we estimate the overall time reduction ratio (TR).

Suppose the ATE and SoC under test work with frequencies f_{ATE} and f_{SoC}, respectively. When there is no compression, the test time is the same as the transfer time. That is:

$$T_{no_comp} = (\sum_{i=1}^{N_b} L_i) \cdot (\frac{1}{f_{ATE}})$$

When we compress the test data, such as in our method, the overall time is made of three portions:

$$T_{comp} = T_{transfer} + T_{decode} + T_{idle}$$

In our method, the transfer and decode part is quite straightforward. Essentially, codewords (C_i) are transferred with a speed of f_{ATE} and counting toward L_i is done with a speed

of f_{SoC}. In other words:

$$T_{transfer} = (\sum_{i=1}^{N_b} l_i) \cdot (\frac{1}{f_{ATE}})$$

$$T_{decode} = (\sum_{i=1}^{N_b} L_i) \cdot (\frac{1}{f_{SoC}})$$

Based on the decoder architecture shown in Figure 7, when the counter has not finished its counting for a block length L_i, the FSM cannot function on a new codeword C_i. The FSM receives l_i bit in $\frac{l_i}{f_{ATE}}$ cycles and counter counts up to L_i in $\frac{L_i}{f_{SoC}}$ cycles. We may have T_{idle}=0 only if the following inequality is satisfied:

$$(\max_{1 \le i \le N_b} \{L_i\}) \cdot (\frac{1}{f_{SoC}}) \le (\min_{1 \le i \le N_b} \{l_i\}) \cdot (\frac{1}{f_{ATE}})$$

When this relation is not satisfied for the (l_i,L_i) pair, the idle time will be $\frac{L_i}{f_{SoC}} - \frac{l_i}{f_{ATE}}$ and therefore:

$$T_{idle} = \sum_{i=1}^{N_b} (\frac{L_i}{f_{SoC}} - \frac{l_i}{f_{ATE}}) \quad \forall i \ni \frac{L_i}{f_{SoC}} > \frac{l_i}{f_{ATE}}$$

Although the above relation can be approximated as $T_{idle} \simeq N_b \cdot (\frac{L_{avg}}{f_{SoC}} - \frac{l_{avg}}{f_{ATE}})$, we prefer to use the upper bound to combine T_{idle} with $T_{transfer}$ and T_{decode}. According to the above equation for T_{idle}, the upper bound will be $T_{idle} \le \sum_{i=1}^{N_b} \frac{L_i}{f_{SoC}}$ and therefore:

$$T_{comp} \le \sum_{i=1}^{N_b} (\frac{l_i}{f_{ATE}} + 2 \cdot \frac{L_i}{f_{SoC}})$$

Finally, the time-reduction ration can be computed as:

$$\begin{aligned} TR &= \frac{T_{no_comp} - T_{comp}}{T_{no_comp}} \\ &\ge \frac{\sum_{i=1}^{N_b} \frac{L_i}{f_{ATE}} - \sum_{i=1}^{N_b} \frac{l_i}{f_{ATE}} - 2\sum_{i=1}^{N_b} \frac{L_i}{f_{SoC}}}{\sum_{i=1}^{N_b} \frac{L_i}{f_{ATE}}} \\ &\ge \frac{\sum_{i=1}^{N_b} (L_i - l_i)}{\sum_{i=1}^{N_b} L_i} - 2\frac{\frac{1}{f_{SoC}}}{\frac{1}{f_{ATE}}} \\ &\ge CR - 2\frac{f_{ATE}}{f_{SoC}} \end{aligned}$$

The above formula shows a lower bound for the TR metric. For the example in Figure 1, if f_{ATE}=100MHz, f_{SoC}=1GHz and CR=53.12% we estimate $33.12\% \le TR \le 53.12\%$. In Section 6 we will show empirical evidence indicating that this lower bound of TR is not tight and in practice, the relation $CR - 1.5\frac{f_{ATE}}{f_{SoC}} \le TR \le CR - 1.3\frac{f_{ATE}}{f_{SoC}}$ holds for large test sets.

5. POWER SAVING

The main goal of compression is to reduce the volume of test data. Each compression method decides on filling don't cares in the test set. Such decision (sometimes called

test set mapping) affects the power consumption of those components that finally receive uncompressed data from the decoder unit. In other words, the decoder simply performs the reverse job and creates the same data (including those that replaced don't cares) as decided in the compression process.

Reducing the number of transitions in test vectors reduces the switching activity and eventually power during scan-in operation. Several techniques have been reported to reduce test application power. Test vector reordering [27] reorders scan cells such that the test sequence shifted in has the minimum switching activity during test application. The gated clock scheme [28] reduces clock rate on scan cells during shift operations to reduce scan power consumption. Scan latch partitioning in multi-scan chains [29] reduces power dissipation by eliminating the spurious transitions which occur in the combinational part of the circuit via sending an extra test vector.

While scan power reduction is not the main focus of this paper, our compression technique also reduces the *scan-in power*. This refers to power consumption of elements in the scan chain during scan-in operation. This desirable side effect (i.e. the reduced scan-in power) exists due to the inherent feature of our RL encoding technique that generates the minimum number of transitions. We have not pursued scan-out power (due to the core's output responses) in this paper. Reducing scan-out power requires scan cell reordering [40] or scan latch partitioning [29] which is beyond the scope of this paper.

To analyze scan-in power, we have used the power estimation relation proposed in [2] and [31]. These analytical relations only estimate bit transitions (proportional to power consumption) due to input test patterns traveling through the chain. Suppose n and m are the number of test patterns and scan chain length, respectively. Assume T_i=(b_{i1} b_{i2} ... b_{im}) is the ith test pattern ($1 \leq i \leq n$), where b_{i1} is the first bit scanned into the chain. Power dissipated in the scan cell elements (due to input test patterns) is estimated by counting the number of weighted transitions (WT) in the pattern as presented in [2]. The transition weight for a bit indicates how many times this bit is replaced with its complemented value when it is scanned into the scan chain.

In [31], the authors showed an analytical formula for scanning in pattern T_i, WT_i:

$$WT_i = \sum_{j=1}^{m-1} (m-j)(b_{ij} \oplus b_{i(j+1)})$$

The total, average and peak weighted transitions, due to input patterns, are:

$$\begin{cases} WT = \sum_{i=1}^{n} WT_i \\ WT_{avg} = WT/n \\ WT_{peak} = MAX_{1 \leq i \leq n}\{WT_i\} \end{cases}$$

As mentioned earlier, the RL encoding step in our technique builds minimum transitions in each test vector. The minimum bit transition for a test vector does not necessarily mean the minimum WT_i (as each bit enters the chain) in that test vector. WT_i depends on the position of the transitions. Note that in WT_i formula $m-j$ is the weight for the jth transition position in T_i. To reduce WT_i we need to minimize $m-j$. If we push the transition position towards the least significant bit, the weighted transitions of a test vector (WT_i) and eventually the overall weighted transitions (WT) are reduced.

As an example, assume that T_1=(b_{11},b_{12}, $\cdots$, b_{18})=(0xx11x0x) when m=8 and obviously the minimum number of transitions is two. Suppose that b_{11} is the first bit scanned into the scan chain. As shown in Section 2, after filling don't cares, RL encoding produces

Table I. Comparing WT for an example with different fillings of don't cares

i	T_1=0xx11x0x	*Bit Transition*	WT
1	0xx11x01	≥ 2	≥ 8
2	00011000	2	8
3	00111000	2	9
4	00111100	2	8
5	01011x00	≥ 4	≥ 15
6	01111000	2	10
7	01111100	2	9
8	**00011100**	**2**	**7**

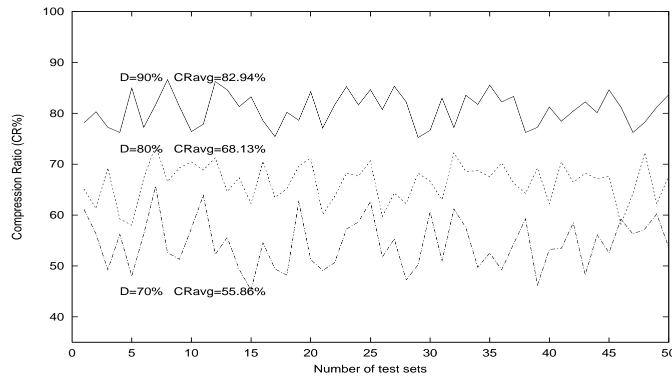


Fig. 8. CR for 50 random test sets.

T_1=(00011100). This filling method gives the minimum weighted transition in each test vector. Fortunately, it is an inherent property of RL encoding process to push minimum transitions towards the least significant bit. Any other different fillings with the same number of transitions cause larger WT. Table I shows the comparison between all possible fillings of don't cares while keeping the minimum number of transitions for test vector T_1. The last row in the table shows our filling process which has the minimum WT. The other rows either show more WT or indicate more transitions which eventually result in more WT. In Section 6 we compare average and peak scan-in power estimate with the random-fill technique.

6. EXPERIMENTAL RESULTS

The proposed technique is used to compress test data for ISCAS89 and PMC-Sierra's [43] benchmarks. Test patterns for PMC Sierra's benchmarks are generated using FastScan [44] with static compaction. Test patterns for ISCAS89 benchmarks are identical to those used by other researchers [11] [14] [24].

Figure 8 shows the results of our compression technique applied to 50 randomly generated test sets when we tuned the random filling to have 90%, 80% and 70% don't cares. CR_{avg} shows average compression ratios among 50 examples in each case and is close to D% especially when D% is high. This experiment confirms our conjecture that CR% is generally upper bounded by D% (see Section 3.3).

Table II. Test pattern compression analysis and compression ($K = n \cdot m$)

Circuit	N_{bits_before}	$D\%$	N_{bits_after}	*Compression Ratio (CR%)*			
				RL-Huffman(+Tech.3)	FDR[11]	MTC(+SLR)[12]	VIHC [14]
PMC1	164800	90	24209	85.31	–	–	–
PMC2	4557	88	712	84.36	–	–	–
PMC3	16830	83	3519	79.09	–	–	–
PMC4	89154	82	17295	80.60	–	–	–
PMC5	13311	81	2678	79.89	–	–	–
s420	1785	51	1026	42.52	–	–	–
s838	5762	59	2215	61.55	–	–	–
s1196	4448	56	3016	32.19	–	–	–
s1238	4864	56	3263	32.90	–	–	–
s1423	3276	46	2225	32.08	–	–	–
s5378	23754	71	10986	53.75 (**58.23**)	50.77	38.49 (46.01)	51.52
s9234	39273	72	20582	47.59 (52.46)	44.96	39.06 (47.20)	**54.84**
s13207	165200	92	28893	82.51 (**86.31**)	80.23	77.00 (81.07)	83.21
s15850	76986	83	25143	67.34 (**69.89**)	65.83	59.32 (64.59)	60.68
s38417	164736	68	59024	64.17 (**66.25**)	60.55	55.42 (58.56)	54.51
s38584	199104	82	74863	62.40 (**68.97**)	61.13	56.63 (63.41)	56.97
s35932	28208	35	3029	89.26 (92.07)	–	83.77 (**95.75**)	66.47

Table II shows the compression results of five PMC Sierra's SoCs and twelve ISCAS89 benchmarks assuming $K = n \cdot m$ that implies unlimited block size. The total number of bits before (N_{bits_before}) and after (N_{bits_after}) compression and don't care percentage ($D\%$) are also provided. This table also compares our results with the best results presented in [11], [12] and [14]. Specifically, for the last seven ISCAS89 benchmarks, we reported our RL-Huffman method with and without using Technique 3. Recall that Technique 3 requires revisiting blocks to find higher occurrence frequencies (equivalent to minimum number of blocks). The numbers in bold faces indicate the highest compression ratio among those reported in the table. In most cases, Technique 3 improves the result of the original RL-Huffman by 2 to 7%.

The FDR, MTC and VIHC methods in Table II were chosen for comparison due to their similarities in using run-length encoding. We acknowledge that other compression methods, such as dictionary-based LZ77 [24] or LZW [25] approaches, have reported higher compression ratios for some of these benchmarks. However, their results in terms of consistency to achieve a high compression ratio, cost of memory-demanding decoder and scan-in power due to high bit transitions are still inconclusive.

Table III shows the entropy analysis of ISCAS89 benchmarks. As shown in the last three columns, using our technique l_{avg}, H and E are very close to their corresponding lower bound.

Table IV shows the test time analysis of ISCAS89 benchmarks for different frequencies of the system. It shows that when $f_{SoC} >> f_{ATE}$, the test application time reduction ratio (TR) is close to CR confirming our analysis in Section 4.3.

As Figure 7 shows, the decoder consists of mainly three components, one FSM, a lookup table and a counter. The counter size is $\lceil log_2 L_{max} \rceil$ bit, where $L_{max} = \max_{1 \le i \le N_l}\{L_i\}$. The counter size/cost is not sensitive to compression factors. Therefore, we report only the cost of the main part of the decoder, i.e. the FSM with the lookup table. This cost (*FSM-Table*

Table III. Entropy analysis

Circuit	*RL-Huffman*	*Entropy Analysis*		
	CR%	l_{avg}	*H*	*E*
s420	42.52	3.800	3.774	1.006
s838	61.55	4.343	4.321	1.005
s1196	32.19	3.081	3.011	1.023
s1238	32.90	3.160	3.098	1.020
s1423	32.08	3.387	3.319	1.020
s5378	53.75	3.599	3.556	1.012
s9234	47.59	4.223	4.180	1.010
s13207	82.51	5.328	5.299	1.005
s15850	67.34	4.466	4.429	1.008
s38417	64.17	4.121	4.104	1.004
s38584	62.40	4.271	4.249	1.005
s35932	89.26	3.908	3.873	1.009

Table IV. Test Time analysis

Circuit	*RL-Huffman*	*Time Reduction Ratio (TR%)*		
	CR%	$\frac{f_{ATE}}{f_{SoC}} = \frac{1}{20}$	$\frac{f_{ATE}}{f_{SoC}} = \frac{1}{10}$	$\frac{f_{ATE}}{f_{SoC}} = \frac{1}{5}$
s420	42.52	35.62	28.72	14.92
s838	61.55	54.55	47.55	33.55
s1196	32.19	25.54	18.89	5.59
s1238	32.90	25.75	18.60	4.30
s1423	32.08	25.18	18.28	4.48
s5378	53.75	46.95	40.15	26.55
s9234	47.59	41.09	34.59	21.59
s13207	82.51	75.26	68.01	53.51
s15850	67.34	60.34	53.34	39.34
s38417	64.17	57.46	50.88	37.05
s38584	62.40	55.90	49.40	36.40
s35932	89.26	82.61	75.96	62.66

Table V. Decoder (*FSM-Table*) cost analysis

Circuit	L_{max}	*Cost [NAND]*
PMC2	217	481
PMC3	165	373
s420	47	173
s1196	32	161
s1423	214	432
s5378	281	551
s9234	296	589
s15850	611	769

Table VI. Tradeoff between compression ratio and decoder cost using different maximum block size (K)

Circuit	*Techniques*	*Factors*	*Maximum Run Length (K)*					
			4	*8*	*12*	*16*	*32*	$K = n \cdot m$
PMC2	1	*RL-Huffman CR%*	52.5	65.4	68.0	75.7	78.2	84.3
		FSM-Table Cost	48	91	147	251	379	481
	2	*RL-Huffman CR%*	59.0	76.5	**80.2**	79.6	79.1	**84.3**
		FSM-Table Cost	51	70	**82**	121	239	**481**
s5378	1	*RL-Huffman CR%*	31.2	39.5	45.9	49.2	51.1	53.7
		FSM-Table Cost	43	93	167	301	412	551
	2	*RL-Huffman CR%*	37.1	41.8	47.1	50.2	52.4	53.7
		FSM-Table Cost	49	89	148	277	385	551
s9234	1	*RL-Huffman CR%*	22.8	29.1	34.7	42.7	42.4	47.6
		FSM-Table Cost	44	101	158	337	431	589
	2	*RL-Huffman CR%*	28.5	34.2	38.9	40.6	43.2	47.6
		FSM-Table Cost	51	103	155	301	417	589

pair) is summarized in Table V for some of ISCAS89 and PMC Sierra's benchmarks based on the number of equivalent *NAND* gates reported by Synopsys synthesizer [33].

Table VI shows the effect of maximum run-length K on compression ratio and cost of *FSM-Table* pair. To limit the maximum run-length we explored two techniques (Techniques 1 and 2) discussed in Section 3. As shown, the second technique achieves overall higher compression rate and smaller decoder size than the first technique. This is because the second technique decomposes blocks intelligently to increase the occurrence frequency of characters. Compression ratio in the case of $K = n \cdot m$ for each circuit is more than the case when K is limited. When $K = n \cdot m$, the longer block lengths will be sent by much smaller codewords. This results in more compression, but the cost of the decoder slightly increases.

Table VI clearly shows the tradeoff possibility between CR and *FSM-Table* cost for Techniques 1 and 2 by choosing various K's. By choosing a smaller K we significantly reduce the cost of a decoder with slight sacrificing of compression ratio. For example, consider the statistics in bold face obtained for PMC2 benchmark. We can tradeoff 4.1% compression ratio by reducing the size of decoder by a factor of 6.

Figure 9 shows the reduction of average (ΔWT_{avg}) and peak (ΔWT_{peak}) switching activities for various random test sets. Statistics are reported based on the results of 50 randomly generated test sets for average and peak values. Each test set includes 100 test vectors and each vector is 32 bit for different don't care percentages (70%, 80% and 90%). When D% increases, our method is more efficient in reducing average and peak power.

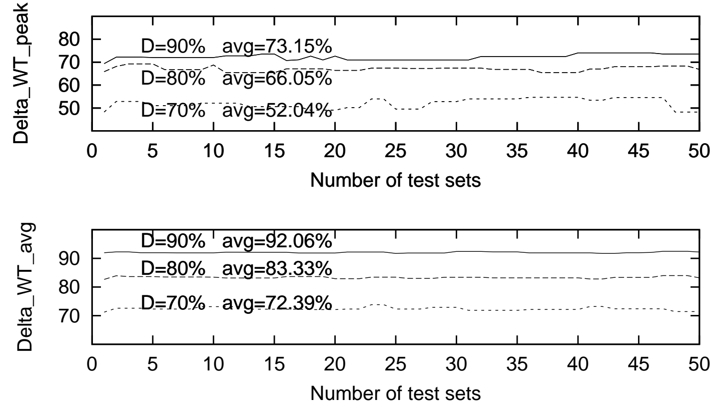


Fig. 9. ΔWT_{peak} and ΔWT_{avg} for 50 random test sets.

Table VII. Comparing switching activities (estimate of scan-in power dissipation)

Circuit	$N_{patterns}$	*Randomly Filled*		*RL-Huffman*		*Scan-in Power Reduction*	
		WT_{avg}	WT_{peak}	WT_{avg}	WT_{peak}	ΔWT_{avg}%	ΔWT_{peak}%
s420	1785	200	314	95	238	52.36	24.20
s838	5762	700	1310	219	1022	68.72	21.98
s1196	4448	244	393	52	234	78.69	40.45
s1238	4864	241	356	49	177	79.47	50.28
s1423	3276	1860	2551	831	2098	55.28	17.75
s5378	23754	11409	13370	3433	11519	69.90	13.84
s9234	39273	15085	18416	3957	14092	73.76	23.47
s13207	165200	122034	137455	7734	94879	93.66	30.97
s15850	76986	92212	102816	13513	70875	85.34	31.06
s38417	164736	581505	65560	116301	411718	80.51	37.26
s38584	199104	527871	572499	85655	481158	83.77	15.95
s35932	28208	316548	787718	39874	107226	87.40	86.38

Table VII shows the comparison between scan-in power dissipated (due to the input test patterns) in two cases. First, we replace the don't care bits randomly for each ISCAS89 benchmark. This process is performed 50 times and results shown in the table are the average of the 50 times compilations. Second, the don't cares are replaced according to our technique. The average and peak switching activities are shown in the table. The reduction percentage for weighted transition average and peak (ΔWT_{avg} and ΔWT_{peak}) are shown in the last two columns with respect to random filling. The average and peak power reductions reported in Table VII are very close to those reported in [11] [12]. This is expected as in all three methods the run length encoding produces the minimum bit transition.

7. CONCLUSION

We presented a new compression and decompression technique based on Run-Length and Huffman codings for scan testing to reduce test data volume, test application time and scan-in power. The proposed technique takes advantage of don't cares generated by the ATPG. The method can be tuned by limiting the maximum run-length to tradeoff compression

ratio and decoder cost. Experimental results show that up to 89% compression ratio and 93% scan-in power reduction is achievable for the benchmarks that we have tried so far.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation CAREER Award #CCR-0130513. The authors also wish to thank Karim Arabi (PMC Sierra), Christos Papachristou and Francis Wolff (Case Western Reserve University) and Michael Knieser (Purdue University Indianapolis) for generously sharing their test pattern sets with us. We also thank Krishnendu Chakrabarty (Duke University) for his fruitful discussion on scan power analysis.

References

[1] J. Rabaey, *Digital Integrated Circuits*, Prentice-Hall, 2002.

[2] R. Sankaralingam, R. Oruganti and N. A. Touba, "Static Compaction Techniques to Control Scan Vector Power Dissipation," in Proc. *VLSI Test Symp. (VTS'00)*, pp. 35-40, 2000.

[3] Y. Zorian, E. Marinissen and S. Dey, "Testing Embedded-Core-Based System Chips," *Computer*, 32(6), pp. 52-60, 1999.

[4] V. Iyengar, K. Chakrabarty and B. Murray, "Built-In Self Testing of Sequential Circuits Using Precomputed Test Sets," in Proc. *VLSI Test Symp. (VTS'98)*, pp. 418-423, 1998.

[5] D. Das and N. Touba, "Reducing Test Data Volume Using External/LBIST Hybrid test patterns," in Proc. *Int. Test Conf. (ITC'00)*, pp. 115-122, 2000.

[6] I. Hamzaoglu and J. Patel, "Reducing Test Application Time for Built-In Self-Test Patterns," in Proc. *VLSI Test Symp. (VTS'00)*, pp. 369-376, 2000.

[7] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "Built-In Test for Circuit with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Trans. On Computer*, vol. 44, pp. 223-233, 1995.

[8] S. Golomb, "Run-Length Encoding," *IEEE Trans. Inform. Theory*, vol. IT. 12, pp. 399-401, Dec. 1966.

[9] A. Chandra and K. Chakrabarty, "Test Data compression and Decompression for System-on-a-Chip Using Golomb Codes," in Proc. *VLSI Test Symp. (VTS'00)*, pp. 113-120, 2000.

[10] A. Chandra and K. Chakrabarty, "Frequency-Directed Run-Length (FDR) Codes with Applications to System-on-a-Chip Test Data Compression," in Proc. *VLSI Test Symp. (VTS'01)*, pp. 42-47, 2001.

[11] A. Chandra and K. Chakrabarty, "Reduction of SOC Test Data Volume, Scan Power and Testing Time Using Alternating Run-Length Codes," *Design Automation Conf. (DAC'02)*, pp. 673-678, 2002.

[12] P. Rosinger, P. Gonciari, B. Al-Hashimi and N. Nicolici, "Simultaneous Reduction in Volume of Test Data Power Dissipation for Systems-on-a-Chip," *Electronics Letters*, vol. 37, no. 24, pp. 14341436, 2001.

[13] P. Rosinger, P. Gonciari, B. Al-Hashimi and N. Nicolici, "Analysing Trade-offs in Scan Power and Test Data Compression for System-on-a-Chip," in IEE Proceedings, Computers and Digital Techniques, 149(4):188 196, July 2002.

[14] P. Gonciari, B. Al-Hashimi and N. Nicolici, "Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-chip Test Data Compression/Decompression," *Design, Automation and Test in Europe (DATE'02)*, pp. 604-611, 2002.

[15] A. Jas, J. Ghosh-Dastidar and N. Touba, "Scan Vector Compression/Decompression Using Statistical Coding," in Proc. *VLSI Test Symp. (VTS'99)*, pp. 114-120, 1999.

[16] F. Hsu, K. Butler and J. Patel, "A Case Study on the Implementation of the Illinoise Scan Architecture," in Proc. *Int. Test Conf. (ITC'01)*, pp. 538-547, 2001.

[17] J. Rajski, et. al, "Embedded Deterministic Test for Low Cost Manufacturing Test," in Proc. *Int. Test Conf. (ITC'02)*, pp. 301-310, 2002.

[18] I. Bayraktaroglu and A. Orailoglu, "Test Volume and Application Time Reduction Through Scan Chain Concealment," in Proc. *Design Automation Conf. (DAC'01)*, pp. 151-155, 2001.

[19] C. Krishna, A. Jas and N. Touba, "Test Vector Encoding Using Partial LFSR Reseeding," in Proc. *Int. Test Conf. (ITC'01)*, pp. 885-893, 2001.

[20] S. Reddy, K. Miyase, S. Kajihara and I. Pomeranz, "On Test Data Volume Reduction for Multiple Scan Chain Designs," in Proc. *VLSI Test Symp. (VTS'02)*, pp. 103-108, 2002.

[21] A. Khoche, E. Volkerink, J. Rivoir and S. Mitra, “Test Vector Compression Using EDA-ATE Synergies,” in Proc. *VLSI Test Symp. (VTS’02)*, pp. 97-102, 2002.

[22] S. Wang and S. Chiou, “Generating Efficient Tests for Continuous Scan,” in Proc. *Design Automation Conf. (DAC’01)*, pp. 162-165, 2001.

[23] M. H. Tehranipour, N. Ahmed, M. Nourani, “Testing SoC Interconnects for Signal Integrity Using Boundary Scan,” in Proc. *VLSI Test Symposium (VTS’03)*, pp. 4A.13-4A.18, 2003.

[24] F. Wolff and C. Papachristou, “Multiscan-Based Test Compression and Hardware Decompression Using LZ77,” in Proc. *Int. Test Conf. (ITC’02)*, pp. 331-339, 2002.

[25] M. Knieser, F. Wolff, C. Papachristou, D. Weyer and D. McIntyre, “A Technique for High Ratio LZW Compression,” *Design, Automation and Test in Europe (DATE’03)*, pp. 116 -121, 2003.

[26] A. Wurtenberger, C. Tautermann and S. Hellebrand, “A Hybrid Coding Strategy for Optimized Test Data Compression,” in Proc. *Int. Test Conf. (ITC’03)*, pp. 451-459, 2003.

[27] P. Girard, C. Landrault, S. Pravossoudovtch and D. Severac, “Reducing Power Consumption During Test Application by Test Vector Ordering,” in Proc. *Int. Symp. on Circuits and Systems (ISCAS’98)*, vol. 2, pp. 296-299, 1998.

[28] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault and S. Pravossoudovtch, “A Gated Clock Scheme for Low Power Scan Testing of Logic ICs or Embedded Cores,” in Proc. *VLSI Test Symposium (VTS’01)*, pp. 253-258, 2001.

[29] N. Nicolici and B. Al-Hashemi, “Scan Latch Partitioning into Multiple Scan Chains for Power Minimization in Full Scan Sequential Circits,” in Proc. *Design, Automation and Test in Europe (DATE’00)*, pp. 715-722, 2000.

[30] S. Wang and S. K. Gupta, “ATPG for Heat Dissipation Minimization During Scan Testing,” in Proc. *Design Automation Conf. (DAC’97)*, pp. 614-619, 1997.

[31] A. Chandra and K. Chakrabarty, “Low-Power Scan Testing and Test Data Compression for System-on-a-Chip,” *IEEE Trans. On Computer-Aided Design (TCAD)*, vol. 21, no. 5, pp. 597-604, 2002.

[32] T. Cover and J. Thomas, *Elements of Information Theory*, Wiley, 1991.

[33] Synopsys Inc., “User Manuals for SYNOPSYS Toolset Version 2002.05,” Synopsys, Inc., 2002.

[34] B. Koenmann, “SMARTBIST,” Presentation by IBM in ITC2000.

[35] D. Huffman, “A Method for the Construction of Minimum Redundancy Codes,” in Proc. *IRE*, vol. 40, no. 9, pp. 1098-1101, 1952.

[36] L. Chia-Hsing and J. Chein-Wei, “Low Power Parallel Huffman Decoding,” *Electronics Letters*, vol. 34, no. 3, pp. 240-241, 1998.

[37] R. Benes, S. Nowick and A. Wolf, “A Fast Asynchronous Huffman Decoder for Compressed-Code Embedded Processor,” *Int. Advanced Research in Asynchronous Circuits and Systems*, pp. 43-56, 1998.

[38] M. Rudberg and L. Wanhammar, “High Speed Pipelined Parallel Huffman Decoding,” *Int. Symp. on Circuits and Systems (ISCAS’97)*, vol. 3, pp. 2080-2083, 1997.

[39] M. H. Tehranipour and M. Nourani, “Compression Techniques for Scan,” *Technical Report*, UTD-10-05-2002, Dept. of EE, University of Texas at Dallas, 2002.

[40] V. Dabholkar, S. Cakravarty, I. Pomeranz and S. Reddy, “Techniques for Minimizing Power Dissipation in Scan and Combinational Circuits During Test Application,” *IEEE Trans. On Computer-Aided Design (TCAD)*, vol. 17, no. 12, pp. 1325-1333, 1998.

[41] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithms*, McGraw Hill, 2001.

[42] A. McAuley and P. Francis, “Fast Routing Table Lookup Using CAMs,” in Proc. *INFOCOM’93*, 1993.

[43] PMC-Sierra, www.PMC-Sierra.com, 2002.

[44] Mentor Graphics Corporation, “User Manuals for FastScan,” Mentor Graphics Co., 2002.