A Comprehensive Test Compression Scheme based on Precomputed Test Sets

Zhijian Tian, Fayong Zhao
School of Physics and Electronic Science
Fuyang Normal College
Fuyang, 236037, China
fytzj1999@163.com

Abstract—To cope with increasingly rigorous challenges that large scale digital integrated circuit testing is confronted with, a comprehensive compression scheme consisting of test-bit rearrangement algorithm, run-length assignment strategy and symmetrical code is proposed. The presented test-bit rearrangement algorithm can fasten don’t-care bits, 0s or 1s in every test pattern on one of its end to the greatest extent so as to lengthen end-run blocks and decrease number of short run-lengths. A dynamical don’t-care assignment strategy based on run-lengths can be used to specify the remaining don’t-care bits after the test-bit rearrangement, which can decrease run-length splitting and maximize length of run-lengths. The symmetrical code benefits from long run-lengths and only uses 2 4-bit short code words to identify end-run blocks almost as long as a test pattern, and hence the utilization ratio of code words can be heightened. The presented experiment results show that the proposed comprehensive scheme can obtain very higher data compression ratios than other compression ones published up to now, especially for large scale digital integrated circuits, and considerably decrease test power dissipations.

Keywords-Terms—Run-length code, run-length assignment symmetrical code, test data compression.

I. INTRODUCTION

All manuscripts must be in English. These guidelines include complete descriptions of the fonts, spacing, and related information for producing your proceedings manuscripts. Please follow them and if you have any questions, direct them to the production editor in charge of your proceedings at Conference Publishing Services (CPS): Phone +1 (714) 821-8380 or Fax +1 (714) 761-1784.

with the continuous improvement of semiconductor manufacturing technologies and the rapid development of integrated circuits (ICs) design, the density and functional complexity of ICs are further heightened, which increase failure probability and naturally enhance amount of test data and difficulty of testing them. To solve a series of testing questions such as data storage, test pattern exertion and response analysis, researchers have proposed combinational logic testing, sequential logic testing and design for testability, and so on, where the design for testability becomes one main research focus and is divided into boundary-scan test, self-test and test resource partitioning and so on [1]. For test resource partitioning, a precomputed test set is compressed using code techniques and is stored in automatic test equipments (ATEs). Decode circuit in chip decompresses the compressed data from the ATE and applies them to ICs during testing. In recent years, there have been many classical test resource partition schemes, such as single run-length coding [1]-[4], dual run-length coding [5], [6] and blocking coding [7]-[14].

References [1]-[4] describe a type of code schemes for run-lengths of 0, which obtain certain compression ratios. References [5] and [6] encode not only run-lengths of 0 but also run-lengths of 1, and hence they obtain higher compression effect. Blocking code schemes mainly divide test data into blocks and compress them in different code techniques [7]-[14], several of which [13], [14] emerging recently increase compression effect, even exceed [1]-[6].

In order to improve compression effects of [1]-[4], researchers propose an algorithm of pattern difference optimizing [1]-[4], which looks for a pattern sorting order so as to make sum of different bits between all adjacent test patterns littler. It does help to improve compression ratios of [1]-[4], and make them beyond those of [5]-[14]. However, its implementation brings sharp increase at the cost of decompression hardware, without any positive effect for test power reduction [1]-[4].

The algorithm does very well in compression ratio for single run-length code compression schemes, but it does not fit for others. Mehla et al. [15] also proposes a treatment technique for test set, which further improves compression ratio of run-length code. However, these test set processing programs can not fundamentally solve the contradiction between test power and compression ratio.

To fully take advantage of don’t-care bits and high-quality double-length compression techniques to reduce amount of test data, additional hardware overhead, test power consumption and so on, it is necessary to probe a novel treatment algorithm of test set and a corresponding compression technique. This paper proposes a comprehensive compression scheme consisting of test-bit-rearrangement algorithm, run-length assignment strategy and symmetrical code.

Firstly, the test-bit-rearrangement algorithm can fasten as many bits of certain deterministic value and don’t-care in every pattern as possible on one of its ends so as to increase length of end-run blocks (an end-run block is a part of continuous same value on one end of a test pattern) and decrease number of short run-lengths. The test set after treatment is more fitted for dual run-length code schemes. Secondly, the run-length assignment strategy introduced can
dynamically specify remaining don’t-care bits after test-bit-rearrangement so as to further reduce short run-lengths and lengthen long run-lengths. Finally, symmetrical code technique proposed benefits from the test set treatment, which can get twice the result with half the effort. One reason is that long run-lengths get longer and more, short run-lengths less. The other is that bit number of end-run block is considerably larger, and even close to that of a test pattern. For example, end-run blocks of circuit s35932 from ISCAS’89 benchmarks may attain 1762 bits after test set treating, 1 bit less compared with bit number of its test pattern. Symmetrical code technique only uses a 4-bit code word to directly identify end-run blocks without considering bit number of end-run blocks. Therefore, symmetrical code technique can effectively save number of code words.

The proposed comprehensive compression scheme has several advantages as follows: 1) Test data compression ratios are farther improved, which are not only higher than those of similar test compression schemes, but also higher than those of other types; 2) Transition numbers of test bits and test power dissipations are reduced, since a large number of same level test bits are put together; 3) The transition numbers of test bits are reduced, since a large number of similar test compression schemes, but also higher ratios are farther improved, which are not only higher than those of other types; 4) Transition numbers of test bits and test power dissipations are reduced, since a large number of same level test bits are put together; 5) The comprehensive compression scheme has universal applicability, particularly suitable for large scale integrated circuit testing.

II. TEST-BIT-REARRANGEMENT ALGORITHM AND ANALYSIS

A. Description of Test-Bit-Rearrangement Algorithm

Corresponding test bits in others must be synchronously changed when some test bits in one of test patterns are moved, since a test set contains a few of test patterns, each containing a large number of test bits, and corresponding test bits between different test patterns share common inputs or outputs. Thus, when some deterministic bits in a test pattern are moved to reduce amount of run-lengths and to lengthen end-run block, it may increase amount of run-lengths in other patterns in return. For this, an algorithm need be studied to ensure that new short run-lengths are not increased while a large number of test bits are concentrated. Taking into account the characteristics of test set and the test process, the proposed algorithm adopts fastening a large number of same deterministic and don’t-care bits on the right ends of test patterns (Suppose that each test pattern displays horizontally). Basic idea of the algorithm is as follows.

As illustrated in procedure 1. First, set a boundary position signal (e.g. division) to record number of test bits on the left of boundary position, which do not be counted and shifted after every cyclic operation. Then, count number of 0s and 1s on the right of the boundary position in other test patterns except benchmark patterns, and record minimum bit amounts between 0s or 1s (indicated by ‘number’), their types (indicated by ‘flag’) and sums of 0s and 1s (indicated by ‘sum_number’), respectively. Next, look for a pattern as benchmark one in view of two conditions as below: 1) Select a pattern whose number value is the minimum. 2) Select a pattern whose sum_number value is the maximum when the number values of several patterns are equal. After that, for the benchmark pattern, those deterministic bits equal to the flag value on the right of the boundary position are bit by bit moved in circles to its boundary position according to the number value, and corresponding bits in other test patterns are synchronously moved. Then replace the number value with the number+division value, and fill don’t-care bits on the right of boundary position of the benchmark pattern with inverse code of the flag value. Finally, label the benchmark pattern. So the cycle continues, until all patterns are labeled.

```
Procedure 1: Test-Bit Rearrangement

Begin
readfile(n, m);  // Read n m-bit test patterns.
division=0;
for(h=0; h<n-1; h++) // Set cycle time with variable h.
{
counter(division); // Count and store numbers of 0s and
  // Is in unmarked patterns
  // respectively, and store their types.
compare(s); // Select benchmark pattern, and
  // store its order in variable s.
shift(s); // Move test bits of all patterns
  // according to parameters of
  // benchmark pattern.
filling(s, division); // Amend division value in
  // view of // number value. Fill don’t-care
  // bits
  // on the right of its boundary
  // position, and label the
  // benchmark // pattern.
}
End
```

Table I is used to describe basic idea of the algorithm, where there are 3 test patterns, each with 32 test bits. The first - fifth columns list order, original test pattern, result after rearrangement, boundary position and bit number of end-run block (i.e. bold font in the third column), respectively. In addition, ‘Number’ row marks position changing of every test bit before and after rearrangement, ‘Total’ row total number of bits. First, let division = 0, and count numbers of 0s and 1s in all patterns. For example, the first test pattern has 8 0s and 14 1s, and then, number = 8, sum_number = 22 and flag = 0. Similarly, corresponding values for the second and third test patterns are 7, 20 and 0.7.18 and 1, respectively. Note that the number values of the second and third patterns are equal, and smaller than that of the first, whereas the sum_number value of the second is largest, so the second pattern is chosen as a benchmark pattern. After that, the values from the first to sixth positions in the benchmark pattern are Afterwards...
moved 1 bit in cycle, and so do synchronously corresponding bits in other patterns until 7 0s in the benchmark pattern are all moved close to the boundary position. At last, division = 0 + 7 = 7, and don’t-care bits on the right of the seventh bit in the benchmark pattern are all replaced by inverse code of the flag value, i.e., ‘1’. At the same time, the benchmark pattern is labeled so that 0s and 1s in it are not counted during later cycles. The division = 7 during the second cycle. Number, sum_number and flag of the first and the third patterns on the right of the seventh bit are 4, 15 and 0, 2, 11 and 1, respectively. It is obvious that the third pattern is benchmark one. 2 1s on the right of the seventh bit of the benchmark pattern are cyclically shifted to the eighth and ninth positions, and corresponding bits in other 2 patterns are synchronously moved. Amend division = 7 + 2 = 9, replace don’t-care bits by ‘0’, and label the benchmark pattern. Treat the last cycle in the same method.

As illustrated in Table I: 1) For each test pattern, number of short run-lengths is reduced, long run-lengths lengthened, and total number of run-lengths is lowered. For example, the second pattern has 7 run-lengths before rearrangement, but it has only 2 run-lengths after rearrangement. 2) Bit number of end-run block gets higher. For example, end-run block of the second pattern has 1 ‘0’ before rearrangement, but does 25 ‘1’ after rearrangement, close to 32 bits. 3) The test set has 96 bits, however, end-run blocks owns 68 bits and exceeds half of the total test bits. All the facts show that the proposed algorithm should be effective.

The ‘number’ row in Table I shows that every test bit could change after the test-bit rearrangement. Therefore, in order to apply test patterns to circuits under test (CUT) with hardware overhead.

B. Analysis of Effectiveness

The test-bit-rearrangement algorithm is mainly fitted for dual run-length code schemes, and hence it must be effective as long as average length of run-lengths is increased after test-bit rearrangement.

Taking into account arbitrary assignment of don’t-care bits, let a memoryless test pattern has m bits, with s 0s, where the probability of 0 is p = s / m, the probability of 1 1-p = (m-s) / m. Thus, the average length of run-lengths is

\[ \lambda = \sum_{i=1}^{s} ip_i + \sum_{i=1}^{m-s} i(1-p)^i \]

(1)

Allow for the s and m-s to be very large. For example, a pattern of circuit s13207 in ISCAS89 benchmark has 700 bits. Hence (1) is amended as follows:

\[ \lambda = \frac{p}{1-p} + \frac{1-p}{p} = \frac{1-3p + 3p^2}{p^2 (1-p)^2} \]

(2)

Without loss of generality, suppose d 0s are moved to one end, since a large number of 0s or 1s are moved to one end of test patterns after test-bit rearrangement. At this time, the original test data apart from end-run block has s-d 0s, where the probability of 0 is p_0 = (s-d) / (m-d), the probability of 1 p_1 = 1-p_0 = (m-s) / (m-d). So, average length of run-lengths after the rearrangement is given by:

\[ \lambda = \sum_{i=1}^{s-d} ip_0^i + \sum_{i=1}^{m-s-d} i(1-p_0)^i \]

(3)

The s-d may be very large still, although the d is large. This is because m is considerably large. Equation (3) is approximately written into:

\[ \lambda = \frac{p_0}{(1-p_0)^2} + \frac{1-p_0}{p_0^2} = \frac{1-3p_0 + 3p_0^2}{p_0^2 (1-p_0)^2} \]

(4)

Although (2) and (4) are the same in the form, probabilities of 1 or 0 are different before and after rearrangement. Numbers of 0s and 1s in original test set are approximately equal. Without loss of generality, suppose p is close to 0.5. A large number of test bits are moved to one end after rearrangement, i.e., d is very large. So p_0 = (s-d) / (m-d) < p, may be less than 0.1, even close to 0.0. Relation curve of average length of run-lengths and probability of 0 is plotted in Fig. 1.

![Figure 1. Relation between probability and average length of run-length](image)

The average length of run-lengths sharply increases, while probability of 0 lowers from 0.5 to 0.0. For example, p = 0.5, thus \( \lambda = 4 \), p_0 = 0.1, thus \( \lambda = 90 \), which shows average length of run-lengths is increased by 22 times. For the case that d 1s are moved to one end, likewise, average length of run-lengths is sharply heightened, while p_0 > p ≥ 0.5. In short, it is effective, since the proposed algorithm can sharply heighten average length of run-lengths.

Table II lists numbers of run-lengths of 7 circuits in ISCAS89 benchmark before and after rearrangement. Principle of filling don’t-care bits is as follows [5], [6]: Let ‘1’ replace don’t-care bits bounded by 1s from both sides in test data stream, otherwise let ‘0’ replace them. Table II shows that numbers of runs of 0s and runs of 1s, and the total of run-lengths all drop sharply after the rearrangement. The last row shows that total of run-lengths is reduced to 7736, which occupies 21.32% of run-lengths before rearrangement. This result also indicates that the test-bit-rearrangement algorithm is able to lengthen long run-lengths, reduce total of run-lengths, and hence it is a very effective pretreatment method of test set.
As seen above, the proposed strategy can reduce number of short run-lengths, in particular, for 1 bit run-length. Naturally the strategy is superior to the three ones introduced above.

### III. RUN-LENGTH ASSIGNMENT STRATEGY

For the remaining don't-care bits after test-bit rearrangement, some effective assignment strategies may still be exploited to lengthen long run-lengths and to decrease number of short run-lengths. There are three assignment techniques [1]-[6]. 1) Fill all of don’t-care bits with ‘0’ or ‘1’ [1]-[4]. For example, if fill don’t-care bits with ‘0’, ‘0X011X10X110’ is changed into ‘000110100110’, which has 5 run-lengths. Else, ‘0X011X10X110’ is changed into ‘010111101110’, which has 4 run-lengths. 2) Let ‘1’ replace don’t-care bits bounded by 1s from both sides, otherwise let ‘0’ replace them [5], [6]. For example, ‘0X011X10X110’ is changed into ‘001111100110’, which has 4 run-lengths. 3) If beginning bit in a test data stream is a don’t-care bit, all of don’t-care bits before the first deterministic bit emerging are replaced by value of the deterministic bit, otherwise don’t-care bits are filled with value of deterministic bit before them. For example, ‘0X011X10X110’ is changed into ‘000111101110’, which has 4 run-lengths.

The above three assignment strategies of don’t-care bits are not ideal. In order to improve compression effect of dual run-length code schemes, a new assignment strategy of don’t-care bits is given, which can dynamically fill don’t-care bits based on requirement of run-lengths in data stream. If the first bit is a deterministic value during constructing a run-length, all of don’t-care bits between it and the first dissimilar deterministic bit emerging after it are replaced with it, otherwise, all of don’t-care bits before the first dissimilar deterministic bit emerging after the first determinate bit are filled with the first determinate value. For example, test pattern ‘0X011X10X110’ is changed into ‘000111101110’ according to the proposed assignment strategy, which has 3 run-lengths. The first and the second Xs are filled based on their previous determinate values, whereas the third X must be filled with value of the first determinate bit after it since it is the beginning bit of a new run-length.

### IV. PROPOSED CODE TECHNIQUE AND ITS DECOMPRESSION CIRCUIT

#### A. Symmetrical Code

Test set after test-bit rearrangement and run-length assignment can well be fitted for dual run-length codes, however, the existing dual run-length code schemes do not fully take advantage of the characteristic of test set, i.e., longer end-run blocks. For this, a symmetrical code is proposed, and shown in table III.

### TABLE III SYMMETRY CODE

<table>
<thead>
<tr>
<th>Group</th>
<th>Length of Run-length</th>
<th>Prefix</th>
<th>Tail</th>
<th>Code Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Runs of 0s</td>
<td>Runs of 1s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>10</td>
<td>01</td>
<td>001001000</td>
</tr>
<tr>
<td>A2</td>
<td>4</td>
<td>110</td>
<td>000</td>
<td>0010001100</td>
</tr>
</tbody>
</table>

‘0100’ and ‘1000’ on the first row of the first group in table III are used to represent end-run blocks of 0s and 1s, respectively, whereas other elements identify different run-lengths. For example, ‘0101’ indicates 1-bit run-length of 0, ‘110001’ 5-bit run-length of 1. The proposed technique is named as symmetrical code because of the symmetries of the code words and types of run-lengths.

The third test pattern in table I is used to show encoding process of the proposed scheme. As illustrated in table IV, results of EFDR code[5] and the proposed code are given on the second and third rows, respectively. For the symmetry code, the first run-length ‘111110’ is encoded with code word ‘110001’, ‘01’ with code word ‘0101’, ‘10’ with code word ‘1001’, the remaining run block of 0s with code word ‘0100’.

#### B. Decompression Circuit

Decompression circuit of symmetry code includes a finite state machine (FSM), three counters and a D-flipflop, shown in Fig. 2. Counter 1 is similar to a counter surplus 3,
which consists of $K$×2-bit flipflops (Where $K$ is maximum number of grouping for a test set) and is used to store tail of code word. The signal $rst1$ is high level only when value of counter 1 is 3. Counter 2 consists of $\log 2K+1$-bit flipflops, which is used to count bit number of prefix so as to control bit number of tail moved into counter 1. Counter 3 is a counter module $m$ ($m$ is equal to bit number of a test pattern.), which can count bit number of each test pattern so as to determine stop of a test pattern, i.e., stop of an end-run block. The D-flipflop is used to capture the first letter of prefix, which can identify type of run-lengths and control their output.

Principle and working process of counter 1 are introduced here. For a run-length of 0s or 1s, assume it at the group $k$, then its length can be expressed as $l=2k+1+M-(100)b$, where $M$ is binary number of tail of code word. Counter 1 decreases by 1 when the first bit of prefix is received, and then it does by 1 too and surplus 1 when end signal of the prefix arrivals. Next, the tails are shifted bitwise into it. When the tail finishes, its value is $2k+1+M$. Subsequently, counter 1 further reduces by 1. If its value is 3 at the time, $rst1 = 1$, which indicates beginning of an end-run block. Otherwise, if it is more than 3, $rst1 = 0$, which shows beginning of a run-length. As illustrated in Fig. 2 and Fig. 3, the decompression circuit works as follows.

1) After initialization, value of counter 1 is 3, and then let $en = 1$, the circuit into state $s1$.

2) Counter 2 counts bit number of prefix after plus 1. Let counter 1 decrease by 1 and the circuit latch data on the $bit\ in$. The circuit goes into state $s2$, and continues to valid the $en$.

3) If $flag = 0$, i.e., the input value is not end signal of prefix, let counter 2 increase by 1 and the circuit continue state $s2$. Otherwise, not only let counter 2 increase by 1, but also make counter 1 decrease by 1, the circuit into state $s3$. Next, counter 2 minus 1, the tail is moved into counter 1 until $rst2 = 1$. Once again let counter 1 decrease by 1, the circuit into state $s4$. At the time, the state order is determined according to value of the $rst1$.

4) If $rst1 = 0$, the circuit enters state $s5$. Counter 1 minus 1, let $valid= 1$, $contr= 0$, output a run-length until $rst1 = 1$. Enable the $en$, $contr$ and $valid$ again, output an anti-code, and the circuit enters state $s1$, i.e., transfers to the step (2).

5) If $rst1 = 1$, the circuit enters state $s6$. Let $valid= 1$, $contr= 0$, output a end-run block until $rst3 = 1$. Enable the $en$, and the circuit enters state $s1$, i.e., transfers to the step (2).

V. EXPERIMENTAL RESULTS AND ANALYSIS

In order to verify compression effect of the proposed scheme, we present experimental results on test data compression for several larger ISCAS89 benchmark circuits, as illustrated in table V. The first and second schemes indicate compression results using EFDR code before and after test-bit rearrangement, respectively [5]. The third and forth schemes show compression effects using the symmetry code before and after run-length assignment after test-bit rearrangement, respectively. For the first and second schemes, although they use the same compression method, the second scheme obtains very high compression ratios because of using test-bit rearrangement algorithm. The largest difference of their compression ratios arrives 14.62%, the smallest also 3.54%. As far as average value is concerned, the second scheme raises 9.30%, reduces by 9779 bits, which occupies 30.65% of volume of the first scheme. This indicates that the proposed test-bit rearrangement algorithm is successful and effective in improving compression effect of test data. For the second and third schemes, symmetry code scheme obtains very higher compression ratios than EFDR code for any circuits. Symmetry code scheme raises 0.50% at average compression ratio, reduces by 497 bits. This indicates that the proposed Symmetry code scheme possesses higher advantage than EFDR code. Finally, the third and forth schemes are analyzed. Symmetry code scheme can obtain higher compression ratios by using run-length assignment strategy, and get 10.87% more than the first scheme, 1.57% the second, 1.06% the third at average compression ratio. This indicates that the run-length assignment strategy is effective in improving compression effect of test data. In addition, run-length assignment strategy has also an advantage that it only affects value of don’t-care bits in test set, without increasing test hardware dissipation.

Table VI lists compression ratios of the proposed

---

![Diagram of decompression](Image 1)

![State transition diagram of FSM](Image 2)
comprehensive compression scheme and several representative classic code compression ones. The second-fourth columns present compression ratios of run-length code schemes combining with pattern difference optimization algorithm (PD[1, 2, 5]) and hamming distance treatment (HD[15]), respectively. Besides the circuit s5378, the proposed scheme obtains the highest compression ratio than others, and its average value is the highest, more than 6% of that of others. In addition, the reason that the proposed scheme can not obtain ideal compression effect for the circuit s5378 is that volume of its test set is less than that of others. This fact shows that the proposed scheme is particularly fit for testing VLSI.

### TABLE VI COMPARISON OF COMPRESSION SCHEMES BASED ON TEST SET

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s13207</td>
<td>84.33</td>
<td>70.03</td>
<td>87.67</td>
<td>74.74</td>
<td>82.49</td>
<td>86.40</td>
</tr>
<tr>
<td>s15850</td>
<td>66.55</td>
<td>62.55</td>
<td>71.95</td>
<td>72.84</td>
<td>68.66</td>
<td>70.43</td>
</tr>
<tr>
<td>s38417</td>
<td>58.06</td>
<td>56.09</td>
<td>65.35</td>
<td>66.18</td>
<td>62.02</td>
<td>65.67</td>
</tr>
<tr>
<td>s38584</td>
<td>59.11</td>
<td>55.87</td>
<td>64.67</td>
<td>64.79</td>
<td>64.28</td>
<td>63.10</td>
</tr>
<tr>
<td>s5378</td>
<td>53.73</td>
<td>52.97</td>
<td>61.32</td>
<td>61.50</td>
<td>63.10</td>
<td>63.20</td>
</tr>
<tr>
<td>s9234</td>
<td>59.85</td>
<td>56.05</td>
<td>60.63</td>
<td>61.06</td>
<td>64.66</td>
<td>57.56</td>
</tr>
<tr>
<td>Average</td>
<td>63.69</td>
<td>58.93</td>
<td>68.60</td>
<td>69.11</td>
<td>70.15</td>
<td>70.15</td>
</tr>
</tbody>
</table>

Reduction in average power consumption is

\[ P_{\text{avg}} = \frac{P_{\text{avg}} - P_{\text{avg}}^T}{P_{\text{avg}}} \times 100 \%
\]

Reduction in peak power consumption is

\[ P_{\text{peak}} = \frac{P_{\text{peak}} - P_{\text{peak}}^T}{P_{\text{peak}}} \times 100 \%
\]

### VI. CONCLUSION

This paper not only proposes a test set treatment algorithm, but also gives a code method and a don’t-care bit assignment strategy. Experimental results and analysis show that the comprehensive compression scheme obtains higher compression ratios than other schemes, more than 6% on average. The proposed scheme also reduces scan test power consumption, and reaches 71.85% at maximum scan power reduction. In addition, the proposed scheme has higher general applicability, especially for large scale integrated circuits, and meets today’s development trends of ICs. Therefore, the comprehensive compression scheme would be an ideal option for testing digital ICs with deterministic fault sets.

### REFERENCES


Here to discuss testing power consumption of the proposed scheme. Weighted transition metric(WTM) [2] is used to estimate scan test power consumption, i.e., given a l-bit test pattern \( t_j = t_{j,1}t_{j,2} \ldots t_{j,l} \), where \( t_{j,i} \) is moved in scan chain before \( t_{j,2} \), and then,

\[ WTM_j = \sum_{i=1}^{l-3} (l-i) \cdot (t_{j,i} \oplus t_{j,i+1}) \]

Let a test set contain \( n \) test patterns, i.e., \( t_1, t_2, \ldots, t_n \), and its average and peak scan power consumptions are estimated as follows:

\[ P_{\text{avg}} = \sum_{j=1}^{n} \sum_{i=1}^{l-3} (l-i) \cdot (t_{j,i} \oplus t_{j,i+1}) \]

\[ P_{\text{peak}} = \max_{i \in \{1, 2, \ldots, n\}} \sum_{i=1}^{l-3} (l-i) \cdot (t_{j,i} \oplus t_{j,i+1}) \]

Table VII lists scan power consumptions of Golomb code [2] and the proposed scheme, where the percentage reduction in power was computed as follows:

Reduction in peak power consumption is

\[ P_{\text{peak}} = \frac{P_{\text{peak}} - P_{\text{peak}}^T}{P_{\text{peak}}} \times 100 \%
\]

Reduction in average power consumption is

\[ P_{\text{avg}} = \frac{P_{\text{avg}} - P_{\text{avg}}^T}{P_{\text{avg}}} \times 100 \%
\]


### Table 1: Process of Test-Bit Rearrangement

<table>
<thead>
<tr>
<th>Order</th>
<th>Original Test Pattern</th>
<th>Rearrangement Result</th>
<th>Boundary Position</th>
<th>Bit Number of Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X00X1XX11XX00XXXX011011101100101</td>
<td>0010011100011111111111111111111</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>X1X100X0000XXXX011011011011010100</td>
<td>0011111111111111111111111111111</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>X1XXXX001XXXX100010010001001001</td>
<td>000000000000000000000000000000000</td>
<td>9</td>
<td>23</td>
</tr>
<tr>
<td>Number</td>
<td>abcdefghijklmnopqrstuvwxyz</td>
<td>96</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>96</td>
<td>28</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>

### Table 5: Comparison of Compression Schemes Before and After Test-Bit Rearrangement, and Under Run-Length Assignment (%)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s13207</td>
<td>165200</td>
<td>28930</td>
<td>82.49</td>
<td>16853</td>
<td>89.80</td>
<td>14704</td>
<td>91.10</td>
<td>14414</td>
<td>91.27</td>
</tr>
<tr>
<td>s15850</td>
<td>76986</td>
<td>24127</td>
<td>68.66</td>
<td>15128</td>
<td>80.35</td>
<td>14700</td>
<td>80.91</td>
<td>14152</td>
<td>81.62</td>
</tr>
<tr>
<td>s35932</td>
<td>28208</td>
<td>5415</td>
<td>80.80</td>
<td>3045</td>
<td>89.21</td>
<td>2846</td>
<td>89.91</td>
<td>2808</td>
<td>90.05</td>
</tr>
<tr>
<td>s38417</td>
<td>164736</td>
<td>62508</td>
<td>62.02</td>
<td>43566</td>
<td>72.34</td>
<td>43566</td>
<td>72.46</td>
<td>43314</td>
<td>73.11</td>
</tr>
<tr>
<td>s38584</td>
<td>199104</td>
<td>71121</td>
<td>64.28</td>
<td>55782</td>
<td>73.49</td>
<td>52482</td>
<td>73.64</td>
<td>50054</td>
<td>74.86</td>
</tr>
<tr>
<td>s5378</td>
<td>23754</td>
<td>11066</td>
<td>53.67</td>
<td>10164</td>
<td>57.21</td>
<td>10042</td>
<td>57.73</td>
<td>9598</td>
<td>59.59</td>
</tr>
<tr>
<td>s9234</td>
<td>39273</td>
<td>20162</td>
<td>48.66</td>
<td>14421</td>
<td>63.28</td>
<td>14340</td>
<td>63.49</td>
<td>13508</td>
<td>65.60</td>
</tr>
<tr>
<td>Average</td>
<td>99609</td>
<td>31904</td>
<td>65.80</td>
<td>22566</td>
<td>75.10</td>
<td>22069</td>
<td>75.61</td>
<td>21121</td>
<td>76.67</td>
</tr>
</tbody>
</table>