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 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;              // Let division= 0.
    for(h=0; h<n-1; h++) // Set cycle time with variable
h.
      {
        counter(division);    // Count and store numbers of
                                      0s and
                              // 1s 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 rightwards 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 design for testability, scan chain structure must be constructed in view of the test bit order after the rearrangement, which does not remarkably increase test 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

$$\lambda = \sum_{i=1}^{s} ip^i + \sum_{i=1}^{m-s} i(1-p)^i \tag{1}$$

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:

$$\lambda = \frac{p}{(1-p)^2} + \frac{1-p}{p^2} = \frac{1-3p+3p^2}{p^2(1-p)^2} \tag{2}$$

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:

$$\lambda = \sum_{i=1}^{s-d} ip_0^i + \sum_{i=1}^{m-s} i(1-p_0)^i \tag{3}$$

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:

$$\lambda = \frac{p_0}{(1-p_0)^2} + \frac{1-p_0}{p_0^2} = \frac{1-3p_0+3p_0^2}{p_0^2(1-p_0)^2} \tag{4}$$

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 larger. 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.

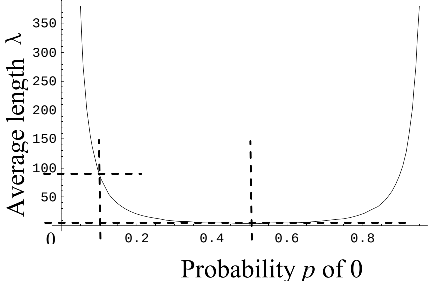


Figure 1. Relation between probability and average length of run-length

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 λ = 4. p_0 = 0.1, thus λ = 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 $p_0 > p \geq 0.5$. 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.

TABLE II. COMPARISON FOR RUN-LENGTH

Circuit	Test Set(bit)	Before Test-bit Rearrangement[5]			After Test-bit Rearrangement		
		Runs of 0	*Runs of 1*	*Sum of run-Length*	*Runs of 0*	*Runs of 1*	*Sum of run-Length*
s13207	165200	2581	1210	3791	1636	950	2586
s15850	76986	2644	1202	3846	1631	1086	2717
s35932	28208	235	346	581	189	207	396
s38417	164736	5773	4834	10607	4926	3445	8371
s38584	199104	7585	4074	11659	6043	3689	9732
s5378	23754	1237	1001	2238	1115	886	2001
s9234	39273	2347	1212	3559	1623	1119	2742
Total				36281			28545

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 '000111100110', 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 '000111100110', 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 determinate bit emerging after it are replaced with it, otherwise, all of don't-care bits before the first dissimilar determinate 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.

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.

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

Group	Length of Run-length	Prefix		Tail	Code Word	
		Runs of 0s	*Runs of 1s*		*Runs of 0s*	*Runs of 1s*
End-run Block	***Uncertain***	01	10	00	0100	1000
A1	1			01	0101	1001
	2			10	0110	1010
	3			11	0111	1011
A2	4	001	110	000	001000	110000
	11			111	001111	110111
…						

'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 symmetric 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'.

TABLE IV CODE AND COMPARISON

Pattern after Rearrangement	11111001100000000000000000000000	32
EFDR Code	11010000100011100111	20
Symmetry Code	110001010110010100	18

As described above, the proposed technique only uses a 4-bit code word to identify end-run blocks without considering their bit number, and hence it can decrease amounts of code words. For example, 22 bits of end-run block (except the tail of previous run-length) in table Ⅳ need 9-bit code word with EFDR code, whereas only do 4 bits with the symmetrical code. Table Ⅳ shows that the test pattern is reduced to 20 bits with EFDR code, whereas leaves 18 bits with symmetric code, saving 2 bits. Therefore, symmetric code can reduce amount of code words and heighten their utilization.

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 *rst*1 is high level only when value of counter 1 is 3. Counter 2 consists of $\lceil \log 2K+1 \rceil$-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 $l=2k+1+N$t-(100)b, where Nt 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+Nt. Subsequently, counter 1 further reduces by 1. If its value is 3 at the time, *rst*1 = 1, which indicates beginning of an end-run block. Otherwise, if it is more than 3, *rst*1 = 0, which shows beginning of a run-length.

As illustrated in Fig. 2 and Fig. 3, the decompression circuit works as follows.

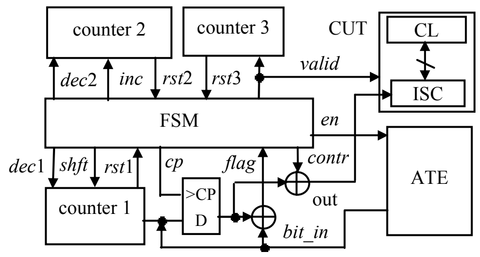
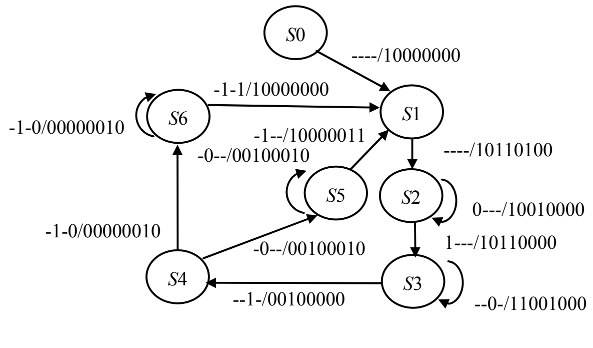


Figure 2. Diagram of decompression



flag, rst1, rst2, rst3 / en, shift, dec1, inc, dec2, cp, valid, contr

Figure 3. State transition diagram of FSM

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

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 *s*2, 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 *s*2. Otherwise, not only let counter 2 increase by 1, but also make counter 1 decrease by 1, the circuit into state *s*3. Next, counter 2 minus 1, the tail is moved into counter 1 until *rst*2 = 1. Once again let counter 1 decrease by 1, the circuit into state *s*4. At the time, the state order is determined according to value of the *rst*1.

4) If *rst*1 = 0, the circuit enters state *s*5. Counter 1 minus 1, let *valid* = 1, *contr* = 0, output a run-length until *rst*1 = 1. Enable the *en*, *contr* and *valid* again, output an anti-code, and the circuit enters state *s*1, i.e., transfers to the step (2).

5) If *rst*1 = 1, the circuit enters state *s*6. Let *valid* = 1, *contr* = 0, output a end-run block until *rst*3 = 1. Enable the *en*, and the circuit enters state *s*1, 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 comprehensive compression scheme and several representative classic code compression ones. The second - fourth columns present compression ratios of run-length code schemes combinating with pattern difference optimization algorithm (PDO)[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 TREATMENT(%)

Circuit	**Golomb Code**		**FDR Code**		**EFDR Code**		**SCCD [12]**	**BDSM [13]**	**Symmetry Code**
	PDO [2]	***HD [15]***	***PDO [1]***	***HD [15]***	***PDO [5]***	***HD [15]***			
s13207	84.33	70.03	87.67	87.47	82.49	86.40	90.39	82.09	**91.27**
s15850	66.55	62.55	71.95	72.84	68.66	70.43	76.17	66.84	**81.62**
s38417	58.06	56.09	65.35	66.18	62.02	65.67	67.95	64.05	**73.71**
s38584	59.61	55.87	64.67	64.79	64.28	63.10	68.64	68.28	**74.86**
s5378	53.73	52.97	61.32	**62.33**	53.67	60.03	60.16	-	59.59
s9234	59.85	56.05	60.63	61.06	48.66	57.56	57.61	45.82	**65.60**
Average	63.69	58.93	68.60	69.11	63.30	67.20	70.15		**76.67**

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 $t_j = t_{j,1}t_{j,2}\dots t_{j,l}$, where $t_{j,1}$ is moved in scan chain before $t_{j,2}$, and then,

$$WTM_j = \sum_{i=1}^{l-1}(l-i)\bullet(t_{j,i}\oplus t_{j,i+1})$$

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

$$P_{avg} = \frac{\sum_{j=1}^{n}\sum_{i=1}^{l-1}(l-i)\bullet(t_{j,i}\oplus t_{j,i+1})}{n}$$

$$P_{peak} = \max_{j\in\{1,2,\cdots,n\}}\{\sum_{i=1}^{l-1}(l-i)\bullet(t_{j,i}\oplus t_{j,i+1})\}$$

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

$$\frac{P_{peak}-P_{peak}^{T}}{P_{peak}}\times 100.$$

Reduction in average power consumption is

$$\frac{P_{avg}-P_{avg}^{T}}{P_{avg}}\times 100.$$

TABLE Ⅶ COMPARISON OF TEST POWER CONSUMPTION

Circuit	**Golomb Code[2]**		**Proposed Scheme**			
	$\mathbf{P_{peak}}$	$\mathbf{P_{avg}}$	P_{peak}^{T}	**Reduction (percent)**	P_{avg}^{T}	**Reduction (percent)**
s5378	10127	3336	7800	22.98	1359	59.26
s9234	12994	5692	10674	17.85	1601	71.87
s13207	101127	12416	40360	60.10	1946	84.33
s15850	81832	20742	43055	47.39	4483	78.39
s38417	505295	172665	303293	39.98	49168	71.52
s38584	531321	136634	451115	15.10	46869	65.70
Average				33.90		71.85

Table Ⅶ shows that the peak and average powers of the proposed scheme are significantly less than those of Golomb code. On average, its peak (average) power is 33.9% (71.85%) less than Golomb code. This indicates the proposed scheme can lower test power consumption by a large margin, and has stronger practicability.

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

[1] A. Chandra, and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency- directed run-length (FDR) codes," *IEEE Transactions on Computers*, vol. 52, no. 8, pp. 1076 -1088, Aug. 2003.

[2] A. Chandra, and K. Chakrabarty, "Low-power scan testing and test data compression for system-on a-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems*, vol. 21, no. 5, pp. 597-604, May 2002.

[3] Y.H. Han, Y.J. Xu, and X.W. Li, "Co-optimization for test data compression and testing power based on variable-tail code," *Proc. the 5th International Conference on ASIC*, pp. 105 – 108, Oct. 2003.

[4] J.H. Feng, and G.L. Li. "A test data compression method for system-on-a-chip," *Proc. the 4th IEEE International Symposium on Electronic Design, Test and Applications*, pp. 270 – 273, Jan. 2008.

[5] A. H. El-Maleh, "Test data compression for system-on-a-chip using extended frequency-directed run-length code," *IET Computers & Digital Techniques*, vol. 2, no. 3, pp. 155 – 163, May 2008.

[6] A. Chandra, and K. Chakrabarty, "A unified approach to reduce SOC test data volume, scan power and testing time," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on.* Vol. 22, no. 3, pp. 352 – 363, march 2003.

[7] P. T. Gonciari, and B. M. Al-Hashimi. "Variable-length input Huffman coding for system-on-a-chip test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, 783- 796, June 2003.

[8] A. Jas, J. Ghosh-Dastidar, N. Mom-Eng, and N. A. Touba, "An efficient test vector compression scheme using selective Huffman coding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 797-806, June 2003.

[9] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Optimal Selective Huffman Coding for Test-Data Compression," *IEEE Transactions on Computers*，vol. 56, no. 8, pp. 1146 – 1152, June 2007.

[10] A. H. El-Maleh, "An efficient test vector compression technique based on block merging," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 1447-1450, May 2006.

[11] T. Kim, S. Chun, Y. Kim, M.H. Yang, and S. Kang, "An effective hybrid test data compression method using scan chain compaction and dictionary-based scheme," *Proc. Asian Test Symposium*, pp. 151 – 156, Nov. 2008.

[12] K. Basu, and P. Mishra, "Test data compression using efficient bitmask and dictionary selection methods," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 9, pp. 1277 – 1286, Sept. 2009.

[13] C.Y. Lin, H.C. Lin, and H.M. Chen. "On Reducing Test Power and Test Volume by Selective Pattern Compression Schemes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 18 , no. 8, pp. 1220 – 1224, Aug. 2010.

[14] M.X. Yi, H.G. Liang, L. Zhang, and W.F. Zhan, "A Novel X-ploiting Strategy for Improving Performance of Test Data Compression," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on.*Vol. 18, no. 2, pp. 324 – 329, Feb. 2010.

[15] U. S. Mehla, K. S. Dasgupta, and N. M. Devashrayee, "Hamming distance based reordering and columnwise bit stuffing with difference vector: A better scheme for test data compression with run length based codes," *Pro. 23rd International Conference on VLSI Design*, pp. 33 – 38, Jan. 2010.

TABLE I PROCESS OF TEST-BIT REARRANGEMENT

Order	**Original Test Pattern**	**Rearrangement Result**	**Boundary Position**	**Bit Number of Block**
1	X00X11XX11XXX0XXX101110111100110	00100111000011111111111111111111	12	20
2	X1X110XX0XXXXXXXX101110101110110	00000001111111111111111111111111	7	25
3	X1XXX0XX01XXXXXXXX10001010001001	11111001100000000000000000000000	9	23
Number	abcdefghijklmnopqrstuvwxyz123456	63ywsifjb2ncadeghklmopqrtuvxz145		
Sum	96	96	28	68

TABLE V COMPARISON OF COMPRESSION SCHEMES BEFORE AND AFTER TEST-BIT REARRANGEMENT, AND UNDER RUN-LENGTH ASSIGNMENT (%)

Circuit	**Test Set(bit)**	**Scheme 1**		**Scheme 2**		**Scheme 3**		**Scheme 4**	
		Code Words(bits)	***Com. Ratio***	***Code Words(bits)***	***Com. Ratio***	***Code Words(bits)***	***Com. Ratio***	***Code Words(bits)***	***Com. Ratio***
s13207	165200	28930	82.49	16853	89.80	14704	91.10	14414	91.27
s15850	76986	24127	68.66	15128	80.35	14700	80.91	14152	81.62
s35932	28208	5415	80.80	3045	89.21	2846	89.91	2808	90.05
s38417	164736	62568	62.02	45566	72.34	45366	72.46	43314	73.71
s38584	199104	71121	64.28	52782	73.49	52482	73.64	50054	74.86
s5378	23754	11006	53.67	10164	57.21	10042	57.73	9598	59.59
s9234	39273	20162	48.66	14421	63.28	14340	63.49	13508	65.60
Average	99609	31904	65.80	22566	75.10	22069	75.61	21121	76.67