Hindawi Publishing Corporation
VLSI Design
Volume 2010, Article ID 670476, 9 pages
doi:10.1155/2010/670476

Review Article

Run-Length-Based Test Data Compression Techniques: How Far from Entropy and Power Bounds?—A Survey

Usha S. Mehta,[1] Kankar S. Dasgupta,[2] and Niranjan M. Devashrayee[1]

[1] *Department of Electronics and Communication, Nirma University, Ahmedabad 382481, India*
[2] *Space Application Center, ISRO, Ahmedabad 380015, India*

Correspondence should be addressed to Usha S. Mehta, usha.mehta@nirmauni.ac.in

Received 23 July 2009; Revised 17 November 2009; Accepted 11 January 2010

Academic Editor: Avi Ziv

Copyright © 2010 Usha S. Mehta et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The run length based coding schemes have been very effective for the test data compression in case of current generation SoCs with a large number of IP cores. The first part of paper presents a survey of the run length based codes. The data compression of any partially specified test data depends upon how the unspecified bits are filled with 1s and 0s. In the second part of the paper, the five different approaches for "don't care" bit filling based on nature of runs are proposed to predict the maximum compression based on entropy. Here the various run length based schemes are compared with maximum data compression limit based on entropy bounds. The actual compressions claimed by the authors are also compared. For various ISCAS circuits, it has been shown that when the X filling is done considering runs of zeros followed by one as well as runs of ones followed by zero (i.e., Extended FDR), it provides the maximum data compression. In third part, it has been shown that the average test power and peak power is minimum when the don't care bits are filled to make the long runs of 0s as well as 1s.

1. Introduction

As a result of the emergence of new fabrication technologies and design complexities, standard stuck-at scan tests are no longer sufficient. The number of tests, corresponding to data volume and test time, increases with each new fabrication process technology just to maintain test quality requirements.

Conventional external testing involves storing all test vectors and test response on ATE. But these testers have limited speed, memory, and I/O channels. Testing cannot proceed any faster than the amount of time required to transfer the test data:

> Test time ≥ (amount of test data on tester)/(number of tester channels ×tester clock rate) [1].

As a result, some companies are looking for compression well beyond 100X tester cycle reduction [2–4].

The paper is organized as follows. Section 2 describes the test data compression techniques and the qualities of a good technique. Section 3 presents existing run-length-based codes. Section 4 introduces the different methods of do not care bit filling for run-length-based code. Section 5 introduces entropy. Sections 6 and 7 present the experimental results of test data compression and test power with different methods of X filling. Section 8 compares the actual data compression for various methods claimed in literature with maximum possible compression predicted on the basis of entropy. Section 9 analyzes the nature of test data on the basis of various experimental results. Finally conclusions and future work discussion are presented in Section 10.

2. Code-Based Data Compression Techniques

Test data compression involves adding some additional on-chip hardware before and after the scan chains. This additional hardware decompresses the test stimulus coming from the tester. This permits storing the test data in a compressed form on the tester. With test data compression, the tester still applies a precise deterministic (ATPG-generated) test set to the circuit under test (CUT).

The quantity of test data rapidly increases, while, at the same time, the inner nodes of dense SoCs become less accessible than the external pins. The testing problem is further exacerbated by the use of *intellectual property* (IP) cores, since their structure is often hidden from the system integrator. In such cases, no modifications can be applied

to the cores or their scan chains, whereas neither automatic test pattern generation nor fault simulation tools can be used. Only precomputed test sets are provided by the core vendors, which should be applied to the cores during testing. In this context, code-based test data compression technique seems more interesting. One more advantage is that by generating difference patterns and reordering test patterns, higher compression can be achieved in some cases.

The Code-based schemes use data compression codes to encode the test cubes. This involves partitioning the original data into symbols, and then replacing each symbol with a code word to form the compressed data. To perform decompression, a decoder simply converts each code word in the compressed data back into the corresponding symbol.

A few important quality factors [5] to be considered for any compression technique are as follows.

(i) The amount of compression possible.

(ii) The area overhead of the decoding architecture.

(iii) The reduction in test time. Transferring compressed test vectors takes less time than transferring the full vectors at a given bandwidth. However, in order to guarantee a reduction in the overall test time, the decompression process should not add much delay.

(iv) The scalability of compression with various design sizes, scan channels, and design types

(v) Test compression method should effectively use do not cares for compression as well as power reduction.

(vi) The robustness in the presence of X states (can the design maintain compression while handling X states without losing coverage?).

(vii) The ability to perform diagnostics of failures when applying compressed patterns.

(viii) Type of decoder: data-independent decoder or data-dependant decoder.

During these years, the researchers have developed a large number of variants of the above schemes.

3. Run-Length-Based Codes

The first data compression codes that researchers investigated for compressing scan vectors were encoded by runs of repeated values.

3.1. Simple-Run-Length Code. In Table 1, a variable number of bits are encoded by a fixed number of bits. Jas and Touba [6] used the above scheme to encode runs of 0s. To increase the prevalence of runs of 0s, this scheme uses cyclical scan architecture to allow the application of difference vectors, where the difference vector between test cubes t_1 and t_2 is equal to t_1 XOR t_2. Careful ordering of the test cubes maximizes the number of 0s in the difference vectors, thereby improving the effectiveness of run-length coding.

3.2. Golomb Codes. As shown in Table 2, Chandra and Chakrabarty [7, 9] and Li and Chakrabarty [8] proposed a technique based on Golomb codes that encode runs of 0s

Table 1: 3-Bits Run Length Code [6].

Code Word	Symbol for 3-bit run length code	Symbol for modified 3-bit run length code
000	1	10
001	01	11
010	001	01
011	0001	001
—	—	—
111	0000000	000000

Table 2: Golomb code for group length = 4 [7].

Group	Run length	Group prefix	Tail	Code word
A_1	0	0	00	000
	1		01	001
	3		11	011
A_2	4	10	00	1000
	—		—	—
	7		11	1011
—	—	—	—	—

with variable-length code words. The code words are divided into groups of equal size m (m is any power of 2). Each group A_k is assigned a group prefix "$(k-1)$ 1s followed by a 0" and as each group contains uniquely identifiable symbols, the final code word consists of a group prefix and a tail of N bit which identifies the member in the group. The use of variable-length code words allows for efficient encoding of longer runs, although it requires a synchronization mechanism between the tester and the chip. This scheme is applied to difference vector derived the same way as in [6].

Encoded sequence corresponding to 001 0000001 001.... is 010 1010 010....

3.3. Frequency-Directed Run-Length Codes.

Original Test Data: 0 0 1 1111 0 0 0 0 111111.

Run Length: 2 0000 4 0 0 0 0 0.

Encoded Test Data: 1000 00 00 00 00 1010 00 00 00 00 00.

In 2001, Chandra and Chakrabarty [10, 11] proposed a new scheme based on the observation that the frequency of runs of 0s with run length less than 20 is high and even within the range of 0 to 20, the frequency of runs of length l decreases rapidly with increasing l. So test data compression can be more efficient if the runs of 0s with shorter run length are mapped to shorter code words. So further optimization can be achieved using frequency-directed run-length (FDR) codes. The FDR is similar to Golomb code but the difference is the variable group size. The size of the ith group is equal to 2^i, that is, that group contains 2^i members. Table 3 demonstrates the code words for different run lengths.

Table 3: Frequency-directed run-length code [10].

Group	Run length	Group prefix	Tail	Code word
1	0	0	0	00
	1		1	01
2	2	10	00	1000
	—		—	—
	5		11	1011
3	6	110	000	110000
	7		001	110001
	—		—	—

Table 4: Extended FDR code [12].

Group	Run length	Group prefix	Tail	Code word Runs of 0s	Code word Runs of 1s
1	1	0	0	000	100
	2		1	001	101
2	3	10	00	01000	11000
	—		—	—	—
	6		11	01011	11011
3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	—		—	—	—

3.4. Extended FDR. The FDR code is very efficient for compressing data that has few 1s and long runs of 0s but inefficient for data streams that are composed of both runs of 0s and runs of 1s**.** Generally, test vectors contain 0s and 1s in group, that is, there will be a run of 1s followed by a run of 0s and vice versa. Maleh and Abaji proposed an extension of FDR (EFDR) [12]. Here, the run of 0s followed by bit "1" and the run of 1s followed by bit "0" are the coded same way as FDR but adding an extra bit at the beginning of FDR code word. Code words for this method are shown in Table 4.

3.5. Alternating Run Length Code. Generally, the test set T is composed of alternating runs of zeros and runs of ones. The alternating run-length code is also a variable-to-variable-length code. An additional parameter associated with this code is the alternating binary variable *a*. The encoding produced by the alternating run-length code for a given run length depends on the value of *a*. If $a = 0$, the run length is treated as a run of 0s**.** On the other hand, if $a = 1$, the run length is treated as a run of 1s. Note that the values of *a* for the different runs are not added to the encoded data stream. The *a* is inverted after each run is encoded and it keeps alternating between 0 and 1 thereafter. The default initial value of $a = 0$, that is, the input data stream starts with a run of 0s. The following example shows the encoded data obtained using this code for a data stream composed of interleaved run of 0*s* and 1s. [13]

Original Test Data: 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 10.

Run Length: 2 4 3 5.

Encoded Test Data: 1000 1010 1001 1011.

a: 0 1 0 1.

3.6. Shifted Alternating FDR Code (SAFDR). One more scheme for alternate runs of 1s and 0s has been proposed by Hellebrand and Würtenberger [14]. Each symbol is made up of only 1s or only 0s. The first bit of encoded data will indicate the type of the first run, that is, 0 or 1, then each code word will indicate the run length of alternate runs. Based on the fact that, in the alternating FDR, there is no run length of 0 size, code word for run length size 0 is assigned to run length size 1. This way, each code word is shifted to one position higher. This helps in achieving higher compression

Table 5: Modified FDR code [18].

Group	Run length	Group prefix	Tail	Code word
A_1	0	01	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	—		—	—
	7		11	1011
A_3	8	001	00	11000
	—		—	—
	11		11	11011
A_4	12	110	000	110000
	13		001	110001
	—		—	—
	19		111	110111
A_5	20	0001	000	0001000
	—		—	—

compared to Alternate FDR. The following example shows coding for Alternating as well as Shifted Alternating FDR.

Original test data: 00 11111 0000 111111.

Run of length: 2 5 4 6.

Encoded test data:

For Alternating FDR: 0 1000 1011 1010 110000.

For Shifted Alternating FDR: 0 01 1010 1001 1011.

3.7. Variable Length Input Huffman Code (VIHC). In order to decompress an FDR code, the on-chip decoder has to identify the group prefix as well as the tail. Because the code is not dependent on a group size as Golomb codes, the decoder has to detect the length of the prefix in order to decode the tail. So, the FDR code requires a more complicated decoder with higher area overhead. So a mix of Huffman and FDR is proposed which instead of using only patterns of *fixed-length* uses patterns of *variable-length* as input to the Huffman algorithm (VIHC) [15, 16]. Here, the compression ratio is retained because of FDR and the area overhead is reduced using selective Huffman Coding.

3.8. Split VIHC. Spilt VIHC [16, 17] approach demonstrates that before going to the VIHC, if the test file is divided into two or more equal parts and the vectors are reordered in a specific way, the compression ratio can be still improved.

3.9. Modified Frequency-Directed Run-Length Code (MFDR). One more scheme based on probability of 0s and FDR is MFDR (Modified Frequency-Directed Run-length) [18]. In this scheme, the groups of FDR are further modified in such a way that gives better compression ratio than FDR if the probability of 0s in the test set is greater than 0.8565. Table 5 presents the code words for this method.

3.10. Selective Relaxation of Bits with FDR Code. In 2003, Kajihara et al. [19] proposed a scheme based on selectively relaxing some of bits of test vector before encoding it using FDR or Golomb code. By changing a specified bit with value 1 to a do not care, two consecutive runs of 0s in the test sequence can be concatenated into a longer run of 0s, thereby facilitating run-length coding. This procedure retains the fault coverage of the test set. Since the increase in compression depends on the lengths of the two runs that are concatenated with each bit relaxation, a lookup table, referred to as the gain table, is precomputed and used during the test set relaxation procedure to maximize the likelihood of increasing the amount of test data compression. The gain table is used to pinpoint the bit positions with value 1, which when relaxed to do not-cares, will yield the maximum compression.

3.11. Data-Independent Pattern Run-Length Code. Ruan and Katti [20] have proposed data-independent run-length coding. This scheme explores the do not care bits in test patterns. It transmits the first segment of the pattern as it is and then compares all other subsequent segments with the first segment and decides either the next segment is equal to the first or complement of the first segment. If segment is equal, it sends "0" and if complement, it sends "11." The code word is ended with "10."

3.12. Run-Based Reordering with EFDR. Stuck-at fault based-test patterns can be reordered without any loss of fault coverage. The test patterns are reordered based on the minimum hamming distance between them. The Run-Based Reordering approach [21] is based on reordering the test patterns to give the bigger run lengths of 0s. As this bigger run lengths are than coded with Extended FDR it gives better compression to normal Extended FDR.

3.13. Fixed-Plus-Variable-Length Code. In 2007, Zhan et al. [22] proposed a test data compression based on fixed-plus-variable-length (FPVL) coding. This scheme divides code word into two parts: fixed-length head section and variable-length tail section. The width of head section is k bits where maximum. possible run length is $2^{2^k+1} - 3$. The value of the presenting tail is two-times bigger than the length of runs in the original test data. In order to obtain further compression, the highest bit of the tail section is reduced from the code words because all of the highest bits in the tail section of the tail are "1." The code words for different run lengths is given in Table 6.

Table 6: Fixed plus variable length coding [22].

Group	Run length	Head (K=2)	Tail	Code word
1	0	00	10	000
	1		11	001
2	2	01	100	0100
	—		—	—
	5		111	0111
3	6	10	1000	10000
	—		—	—
	13		1111	10111
4	14	11	10000	110000
	—		—	—
	29		11111	111111

3.14. Overlapped Vectors with FDR. In 2008, Sheng et al. [23] proposed that in a given test set, there exists such a vector, from which parts of each test vector from the different test vectors can be sought. Based on this, a vector named overlapped vector which contains parts of each test vector and has shorter length than that of the sum of each test vector length is decided. Secondly, the overlapped test vectors are further compressed utilizing Frequency-Directed Run-Length (FDR) coding.

4. Do not Care Bit Filling for Run-Length-Based Codes

To get the maximum compression, the do not care bits should be filled to get the longer runs. The way do not care bits should be filled with 1s or 0s depends upon the different natures of code. The nature of run-length codes can be broadly classified into three categories.

(1) Codes considering the runs of zeros followed by bit one, that is, simple run-length code, Golomb Code, FDR Code, and MFDR Code

(2) Codes considering runs of zeros followed by one as well as runs of ones followed by zero, that is, Extended FDR

(3) Codes considering alternate runs of zeros followed by ones and runs of ones followed by zeros, that is, Alternate FDR

(4) Codes considering the alternate runs of only zeros and only ones without any follower bit, that is, Shifted Alternate FDR

(5) Codes considering the runs of ones followed by zero. This is a hypothetical case only. No such code is proposed in literature but we have taken this case to compare it with the above four styles and analyze the results.

Table 7: Comparison of total no. of symbols needed to be encoded and % compression for various schemes of do not care bits filling.

	Total No. of Symbols Needed to be Encoded						% Compression				
ISCAS circuits	Original test data bits	Runs of 0s followed by 1	Runs of 1s followed by 0	Runs of 0s followed by 1 and 1s followed by 0	Alternating runs of only 0s and only 1s	Alternating runs of 0s followed by 1 and 1s followed by 0	Runs of 0s followed by 1	Runs of 1s followed by 0	Runs of 0s followed by 1 and 1s followed by 0	Alternating runs of only 0s and only 1s	Alternating runs of 0s followed by 1 and 1s followed by 0
S5378	23754	3538	2969	**2094**	3129	3130	52.36	54.33	**56.38**	40.73	38.94
S9234	39273	4817	5786	**3216**	4904	4905	47.80	44.34	**53.37**	35.92	35.33
S13207	165200	5021	6294	**3550**	5427	5428	83.65	81.12	**85.55**	79.72	79.59
S15850	76986	5330	7329	**3647**	5628	5629	68.18	62.49	**71.90**	60.72	60.32
S38417	164736	29473	23110	**9548**	13751	13752	54.50	57.44	**65.84**	55.77	55.14
S38584	199104	16814	18474	**10771**	16275	16275	62.49	60.09	**66.67**	54.65	54.21

Table 8: Comparison of entropy and total no. of unique symbols for various schemes of do not care bits filling.

		Entropy					Total No. of unique symbols				
ISCAS circuits	Original test Data bits	runs of 0s followed by 1	runs of 1s followed by 0	Runs of 0s followed by 1 and 1s followed by 0	Alternating runs of only 0s and only 1s	Alternating runs of 0s followed by 1and 1s followed by 0	runs of 0s followed by 1	runs of 1s followed by 0	Runs of 0s Followed by 1 and 1s followed by 0	Alternating runs of only 0sand only 1s	Alternating runs of 0s followed by 1 and 1s followed by 0
S5378	23754	3.1983	3.6537	**4.9546**	4.4995	4.6338	79	87	**141**	131	136
S9234	39273	4.2556	3.7778	**5.6939**	5.1312	5.1780	74	86	**141**	133	134
S13207	165200	5.3792	4.9558	**6.7272**	6.1745	6.2113	211	178	**331**	314	314
S15850	76986	4.5955	3.9401	**5.9319**	5.3736	5.4267	173	145	**234**	230	229
S38417	164736	2.5432	3.0332	**5.8934**	5.2987	5.3742	121	154	**236**	233	238
S38584	199104	4.4418	4.3018	**6.1614**	5.5474	**5.6021**	214	191	**345**	333	333

Considering the above five cases, the fillings of do not care bits of partially filled test vectors are done as per the following schemes.

4.1. X Filling for Codes Considering Runs of Zeroes Only. For codes like Golomb [7], FDR [10], or MFDR [18], the symbols are made of runs of 0s followed by bit "1." So we have applied a simple technique of replacing all the Xs with 0s. So the overall of the runs of 0s will increase, the number of symbols will decrease and hence entropy will decrease, and data compression will increase.

4.2. X Filling for Codes Considering Runs of Zeros Followed by One As Well As Runs of Ones Followed by Zero. The code Extended FDR [12] is a case which accepts runs of 0s as well as runs of 1s. Here, each symbol is a run of 0s followed by the bit "1" or run of 1s followed by the bit "0." If the last symbol is a run of 0s without any follower bit "1" (run of 1s only without the follower bit "0"), in that case, it would be counted as a symbol of the run length which is equal to the number of 0s (1s). The X filling is done in such a way that it should maximize the run length as well as it should not introduce any new symbol. While filling the X, the logic is that if just before the position of X, if the symbol has ended, the X should be filled with reference to next symbol. But if there is a continuous symbol going on at the position of X, X should be filled such that it increases the run length of the current symbol. Proposed algorithm needs to do back tracking as well as forth tracking.

4.3. X Filling for Codes Considering Alternating Runs of Zeros Followed by One and Runs of Ones Followed by Zero. For code like Alternate FDR [12], the symbols are made of alternate runs of 0s followed by one and runs of 1s followed by zero. Here, the first run must be of zero type. So if there is any X at first bit position, it is replaced by 0. If first bit is "1," then the first run is of 0 length and then it starts with 1. After that all the X bits are filled with last non-X value.

4.4. X Filling for Codes Considering Runs of Ones only. This is a hypothetical case introduced to analyze the compression results for VLSI test data. The symbols are made of runs of 1s followed by bit "0." Here all Xs are replaced by 1s. So the runs of 1s will increase.

Considering the above cases, we can divide these methods in two categories: (1) considering the runs of one type only, that is, either runs of 0s or runs of 1s, and (2) considering runs of both types, that is, runs of 0s as well

Table 9: Comparison of % compression predicted by entropy with corresponding actual compression claimed in literature.

ISCAS circuits	Original test data bits	Codes based on run length of zeroes				Code based on run length of zeros and ones		Code based on aternate runs of zeros and ones	
		Golomb code [7]	FDR [10]	MFDR [18]	% compression predicted by entropy	EFDR [12]	% compression predicted by entropy	alternating FDR [13]	% compression predicted by entropy
S5378	23754	40.70	48.02	51.47	52.36	51.93	56.32	-NA-	51.44
S9234	39273	43.34	43.59	**57.74**	47.80	45.89	53.37	44.96	47.26
S13207	165200	74.78	81.30	83.42	83.65	81.85	85.54	80.23	82.45
S15850	76986	47.11	66.22	66.93	68.18	67.99	71.90	65.83	67.25
S38417	164736	44.12	43.26	**57.95**	54.50	60.57	65.84	60.55	62.93
S38584	199104	47.71	60.91	59.32	62.49	62.91	66.67	61.13	62.23

as 1s. For the second category, after X filling, there may be runs of similar run length but of different run type. While identifying the unique symbols, such runs are taken as two different symbols in this paper.

5. Entropy

Entropy is an important concept to data compression. The entropy of a symbol $E(s)$ is the minimum number of bits needed to encode that symbol. The entropy of the test set is calculated from the probabilities of the occurrence of unique symbols using the formula $E(s) = \sum_{i=1}^{k} pi\log_2(1/Pi)$, where pi is the probability of the occurrence of symbol xi in the test set and k is the total number of unique symbols. In case of fixed symbol length, the formula for the maximum compression that can be achieved is given by ((symbol length − entropy)/symbol length), and in case of variable symbol length, the maximum compression is equal to ((avg symbol length − entropy)/avg symbol length). The average symbol length is computed as $\sum_{i=1}^{n} pi|Xi|$, where pi is the probability of the occurrence of symbol xi, $|xi|$ is the length of symbol xi, and n is the total number of unique symbols [24]. Mathematically it can be proved that the following formula for maximum compression is valid for fixed symbol length as well as variable symbol length. %compression $= ([T - (S \times E)]/T) \times 100$, where T is the total number of bits in original uncoded test data, S is the total number of symbols needed to be encoded, and E is the entropy. For all further discussions, the above formula of %compression is used.

6. Experimental Results with X Filling for Maximum Test Data Compression

We have implemented all the X filling techniques using MATLAB7.0 language. The experiments are conducted on a workstation with a 3.0 GHz Pentium IV processor and 1GB memory. Experiments were performed for X filling to calculate the theoretical limit on test data compression for the dynamically compacted test cubes generated by MINTEST for the largest ISCAS89 benchmark circuits. These are the same test sets used for experiments in [7, 9, 11, 13, 17]. The compression values in Table 7 are predicted from the exact values of entropy that were generated after the X filling. As can be seen in Table 7, the percentage compression that can be achieved is maximum where the runs are considered of both types, that is, runs of 0s and runs of 1s. Note, however, that these entropy bounds would be different for a different test set for these circuits. If the reordering or any other method is used to change the location or the number of do not cares, the entropy can be different. However, given any test set, the proposed method can be used to determine the corresponding entropy bound for it.

7. Experimental Results with X Filling for Minimum Test Power

The goal of X filling was to reduce the number of runs. As the number of runs will decrease, the number of transitions should be reduced, which should lower the test power. In this paper, a widely used weighted transitions metric (WTM) introduced in [25] is used to estimate the average and peak power consumption. Test data $T = \{T_1, T_2, \ldots, T_m\}$ has m patterns, and the length of the pattern is n bits. Each test pattern $T_i = \{t_{i1}, t_{i2}, \ldots t_{in}\}$, $1 \le i \le m$, $1 \le j \le n$ denotes the jth bit in the ith pattern. Weighted transitions metric WTM_j for T_j, the average test power P_{avg} and peak power P_{peak} are estimated as per the formulae in [26]:

$$\mathrm{WTM}_j = \sum_{i=1}^{n-1} (n-i) * \left(t_{j,i} \oplus t_{j,i+1}\right),$$
$$P_{\mathrm{avg}} = \frac{\sum_{j=1}^{m} \mathrm{WTM}_j}{m}, \quad (1)$$
$$P_{\mathrm{peak}} = \max_{1 \le j \le m} \mathrm{WTM}_j.$$

Intuitively, the average power and peak power for test data should be minimum when there are long runs of ones as well as zeros. This is proved in Table 11. Peak power and average

Table 10: Comparison of actual compression claimed in literature for various run based coding schemes.

ISCAS circuits	Original test data	Golomb code [7]	FDR [10]	EFDR [12]	Alternating FDR [13]	MFDR [18]	data independent pattern run length [20]	run based reordering with EFDR [21]	FPVL[22]
S5378	23754	40.70	48.02	51.93	NA	51.47	60.53	**68.56**	52.15
S9234	39273	43.34	43.59	45.89	44.96	57.74	61.46	**70.14**	45.82
S13207	165200	74.78	81.30	81.85	80.23	83.42	88.66	**92.83**	81.58
S15850	76986	47.11	66.22	67.99	65.83	66.93	75.53	**85.09**	67.70
S38417	164736	44.12	43.26	60.57	60.55	57.95	58.72	**84.98**	43.06
S38584	199104	47.71	60.91	62.91	61.13	59.32	75.45	**80.36**	32.29

Table 11: Comparison of peak power and average power for various methods of do not care bit filling.

		Peak power				Average power			
ISCAS circuits	Original test data bits	Runs of 0s	Runs of 1s	Alternating runs of 0s and 1s	Runs of 0s followed by 1 and 1s followed by 0	Runs of 0s	Runs of 1s	Alternating runs of 0s and 1s	Runs of 0s followed by 1 and 1s followed by 0
S5378	23754	12085	12375	**11732**	11522	4300	4087	**3524**	3526
S9234	39273	15395	15640	**14092**	14103	6706	6521	**4002**	4022
S13207	165200	110129	126820	**94879**	94886	12318	1453	**8073**	7887
S15850	76986	84360	88794	**70875**	70894	19448	25636	**13611**	13659
S38417	164736	514716	539019	**437884**	437935	194843	193140	**118100**	118080
S38584	199104	530464	533975	**481158**	481171	133320	142220	**86135**	86305

power are minimum when the X filling is done for alternating runs of 0s and 1s without any follower bit.

8. Method of Do not Care Bit Filling, Total Number of Symbols and Nature of Test Data

For the given test set, the different method of X filling gives the different numbers of total symbols, entropy, and hence compression. For the ISCAS89 benchmark circuits, Table 7 compares the total number of symbols needed to be encoded and % compression for various X filling methods. When the X filling is done to make runs of zeros followed by "1" as well as runs of ones followed by "0" both, the total number of symbols needed to be encoded is minimum, hence %compression is maximum. But as shown in Table 8, for the same methodology of X filling, the entropy is maximum. The reason of higher entropy is the higher number of unique symbols. It can be concluded that for the don't care bit method of "runs of zeros followed by 1 as well as runs of ones followed by 0", in spite of higher number of unique symbols and higher value of entropy, the total number of runs are minimum. This results into maximum compression. This comparison can be further explored to investigate the nature of test data. If the partially specified test data has maximum numbers of zeros, the "Runs of Only Zeros" method must give the maximum compression and vice versa if the test data has maximum numbers of ones, the "Runs of Only Ones" method should give the maximum compression. So the first conclusion is that as for approximately 2/3 of the total cases, the "runs of zeros" has given better compression, so the probability of "0" can be higher than the probability of "1." If "alternating runs of 0s and 1s" method gives the better compression, it can be said that in test data, 1s and 0s are distributed in groups like 11100001110011 and so on. If "runs of 0s and runs of 1s both" method gives better compression, it can be said that in test data, 1s and 0s are not distributed in groups but they may be distributed like a zero sitting between the group of ones or vice versa like 111011100001000 and so on. It means that there may be a large number of cases where "1" is followed as well as preceded by group of 0s and "0" is followed as well as preceded by group of 1s. The comparison of the "alternating runs of 0s and runs of 1s" method with the "runs of 0s and runs of 1s both" method shows that the second method gives more compression. So it can be concluded that in test data, the probability of sitting 1(0) sitting between the group of 0s (1s) is high. This conclusion is further enforced by the results shown in columns 3 to 7 of Table 7. Here, it has been shown that the number of symbols needed to be encoded is minimum in case of "runs of 0s and runs of 1s both" method.

9. Comparison of Compression Based on Entropy with Actual Compression Claimed in Literature

Golomb, FDR, and MFDR scheme are based on run of 0s. EFDR is based on runs of 0s followed by "1" and 1s followed by "0." Alternating FDR is based on alternate runs of 0s followed by "1" and 1s followed by "0." Table 9 compares the % compression claimed in literature for each of these categories of coding with its theoretical upper limit predicted

by entropy. It should be noted that the %compression claimed here as the upper bound predicted by entropy is achieved after filling all do not care bits with an appropriate method of bit filling but without applying any technique like reordering of test vector or difference vector. Considering %compression, MFDR seems to give the best compression for codes based on runs of zeros. It can be seen that because of reordering and other techniques, in some of the cases of EFDR, the %compression achieved is even higher than the predicted by entropy.

Table 10 compares the % compression in case of ISCAS89 benchmark circuits for different coding schemes described in literature. Run-based reorder used with extended FDR gives better compression ratio for ISCAS circuits compared to other schemes.

10. Conclusion

In this survey paper, we have covered the wide variety of test data compression techniques based on run-length scheme and their variants. Five different techniques of do not care bit filling based on the nature of the runs are used to increase the run length and hence the % compression. The entropy-based % compression for each of these five techniques is calculated and the analysis proves that run-length-based code which includes run of ones followed by zero as well as run of zeros followed by one gives the best compression for VLSI test data. The same conclusion is further emphasized by comparison of actual compression claimed by literature where EFDR gives the maximum compression. The run-based reordering and other techniques used to enhance the run length further improve-compression which is proven by Run-Based Reordering with Extended FDR scheme. The researchers can start with this method and explore the possibilities of further compression with consideration of area overhead of on-chip decoder and overall test time and test power.

Acknowledgments

We are thankful to Prof. Nur A. Tauba for providing test sets. Our thanks are also due to Prof. Virendra Singh for his valuable suggestions.

References

[1] U. Mehta, N. Devashrayee, and K. Dasgupta, "Survey of test data compression techniques emphasizing code based schemes," in *Proceedings of the 12th IEEE Euromicro Conference on Digital System Design (DSD '09)*, pp. 617–620, Patras, Greece, August 2009.

[2] J. Rajski, J. Tyszer, M. Kassab, et al., "Embedded deterministic test for low cost manufacturing test," in *Proceedings IEEE International Test Conference (ITC '02)*, pp. 301–310, Baltimore, Md, USA, October 2002.

[3] B. Koenemann, C. Barnhart, B. Keller, et al., "A SmartBIST variant with guaranteed encoding," in *Proceedings of the 10th Asian Test Symposium (ATS '01)*, pp. 325–330, Kyoto, Japan, November 2001.

[4] http://www.reed-electronics.com/tmworld.

[5] N. Tauba, "Survey of test vector compression techniques," *IEEE Transaction Design & Test of Computers*, pp. 294–303, 2006.

[6] A. Jas and N. Touba, "Test vector compression via cyclical scan chains and its application to testing core-based designs," in *Proceedings of the IEEE International Test Conference (ITC '98)*, pp. 458–464, IEEE CS, Washington, DC, USA, October 1998.

[7] A. Chandra and K. Chakrabarty, "Test data compression for system-on-a-chip using Golomb codes," in *Proceedings of the 18th IEEE VLSI Test Symposium (VTS '00)*, pp. 113–120, Montreal, Canada, May 2000.

[8] L. Li and K. Chakrabarty, "On using exponential—Golomb codes and subexponential codes for system-on-chip test data compression," *Journal of Electronic Testing*, vol. 20, no. 6, 2004.

[9] A. Chandra and K. Chakrabarty, "Efficient test data compression and decompression for system-on-a-chip using internal scan chains and Golomb coding," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '01)*, Munich, Germany, March 2001.

[10] A. Chandra and K. Chakrabarty, "Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression," in *Proceedings of the 19th IEEE VLSI Test Symposium (VTS '01)*, pp. 42–47, Marina Del Rey, Calif, USA, March 2001.

[11] 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, 2003.

[12] A. El-Maleh and R. Al-Abaji, "Extended frequency-directed run-length code with improved application to system-on-a-chip test data compression," in *Proceedings of the 8th IEEE International Conference on Electronic Circuits and Systems (ICECS '02)*, vol. 2, pp. 449–452, Dubrovnik, Croatia, September 2002.

[13] A. Chandra and K. Chakrabarty, "Reduction of SOC test data volume, scan power and testing time using alternating run-length codes," in *Proceedings of the 39th Design Automation Conference (DAC '02)*, pp. 673–678, New Orleans, La, USA, June 2002.

[14] S. Hellebrand and A. Würtenberger, "Alternating run-length coding—a technique for improved test data compression," in *Proceedings of the 3rd IEEE International Workshop on Test Resource Partitioning (TRP '02)*, Baltimore, Md, USA, October 2002.

[15] P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Variable-length input Huffman coding for system-on-a-chip test," *IEEE Transactions on Computer-Aided Design*, vol. 22, no. 6, pp. 783–796, 2003.

[16] 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," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '02)*, Paris, France, March 2002.

[17] C. Giri, B. Rao, and S. Chattopadhyay, "Test data compression by spilt-VIHC (SVIHC)," in *Proceedings of the International Conference on Computing: Theory and Applications (ICCTA '07)*, Kolkata, India, March 2007.

[18] J. Feng and G. Li, "A test data compression method for system-on-a-chip," in *Proceedings of the 4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA '08)*, Hong Kong, January 2008.

[19] S. Kajihara, et al., "On combining pinpoint test set relaxation and run-length codes for reducing test data volume," in

Proceedings of the 21st International Conference on Computer Design (ICCD '03), San Jose, Calif, USA, October 2003.

[20] X. Ruan and R. Katti, "Data-independent pattern run-length compression for testing embedded cores in SoCs," *IEEE Transactions on Computers*, vol. 56, no. 4, pp. 545–556, 2007.

[21] H. Fang, C. Tong, and X. Cheng, "RunBasedReordering: a novel approach for test data compression and scan power," in *Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC '07)*, Yokohama, Japan, January 2007.

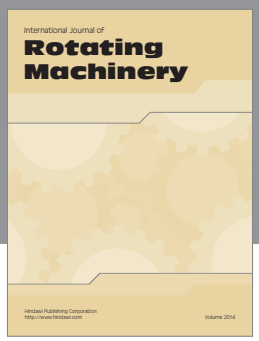
[22] W. Zhan, H. Liang, F. Shi, et al., "Test data compression scheme based on variable-to-fixed-plus-variable-length coding," *Journal of Systems Architecture*, vol. 53, no. 11, pp. 877–888, 2007.

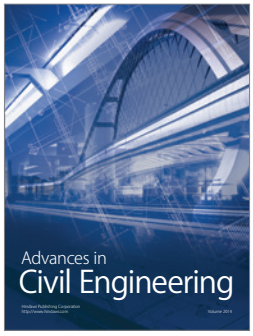
[23] G. Sheng, et al., "Combined partial test vector reuse and FDR coding for two dimensional SoC test compression," in *Proceedings of the International Conference on Internet Computing in Science and Engineering (ICICSE '08)*, Harbin, China, January 2008.

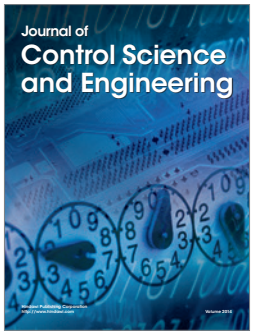
[24] K. Balakrishnan and N. Touba, "Relating entropy theory to test data compression," in *Proceedings of the European Test Symposium (ETS '04)*, Corsica, France, May 2004.

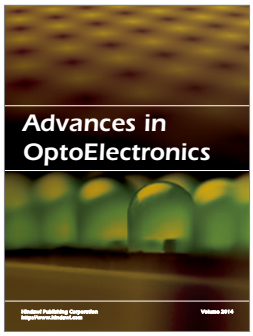
[25] A. Chandra, et al., "How effective are compression codes for reducing test data volume?" in *Proceedings of the VLSI Test Symposium (VTS '02)*, Monterey, Calif, USA, May 2002.

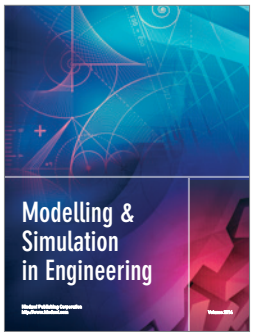
[26] R. Sankaralingam, R. Orugani, and N. Touba, "Static compaction techniques to control scan vector power dissipation," in *Proceedings of the IEEE VLSI Test Symposium (VTS '00)*, pp. 35–40, Montreal, Canada, May 2000.

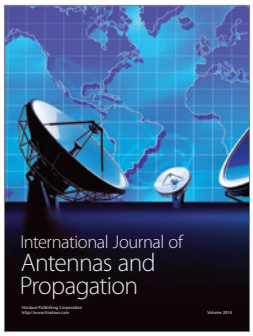


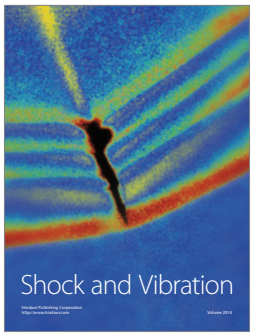


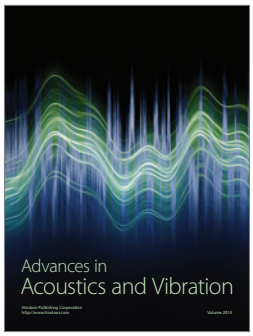












Hindawi

Submit your manuscripts at
http://www.hindawi.com