International Journal of Engineering Research and Development
e-ISSN: 2278-067X, **p**-ISSN : 2278-800X, www.ijerd.com
Volume 5, Issue 5 (December 2012), PP. 08-18

A Novel Test Data Compression Algorithm

Ms.K.Hemalatha[1], Ms R.Kalpana[2], Mr N.SrikanthPrasad[3]
[1]M.Tech.Student, Dept of ECE,, Sri Indu College of Engineering & Technology, Hyderabad,
[2]Associate Professor, Dept of ECE Sri Indu College of Engineering & Technology, Hyderabad
[3]Associate Professor, Dept of ECE Aurora's Engineering College, Bhuvanagiri

Abstract: - The circuit sizes grow ever larger, test data volume and test application time grow unwieldy in system on chip(SOC) designs. Larger test data size demands not only increase in testing time but also requires large memory. Code compression techniques address this issue by reducing the code size. We present a novel test data compression technique using bitmask and dictionary selection to significantly reduce the testing time and memory requirements, which significantly improves the compression efficiency without introducing any decompression penalty. To demonstrate the importance of our approach, we have performed test data compression using applications from various domains and compiled for a wide variety of architectures. Our algorithm outperforms the existing dictionary based approaches by up to 33% giving a best possible test data compression of 92%.

Keywords: - Testing time, memory, bitmask, test compression, decompression.

I. INTRODUCTION

In system on chip(SOC) designs, the circuit grow ever lead to lager volume of test data which demands larger memory requirement and also increase in testing time. Memory is one of the key factor in embedded systems. Because a larger memory indicates the increased chip area, more power dissipation and higher cost. Test data compression place a crucial role reducing the testing time and memory requirements. It also overcomes the automatic test equipment bandwidth limitations, BIST. However, BIST is not appropriate for logic testing because of its inadequate test coverage. There are other alternatives for test coverage, such as bit-flipping and bit-fixing, with the disadvantage that structural information has to be provided. Test data can be reduced with the structural information, but there should be modification of the design. Test data compression algorithms can reduce the test data efficiently with out facing any problems mentioned above. The over view of a test compression methodology is shown in fig. 1.

The original test data is compressed and stored in memory thus the memory size is reduced. The test data is compressed using the compression techniques. An on chip decoder decodes the compressed test data from the memory and delivers the original uncompressed set of test vectors to the design under test circuit.

.

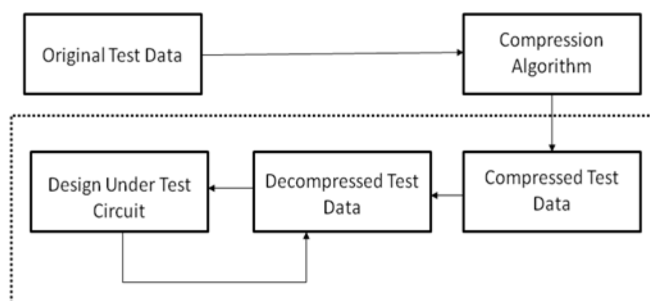


Fig. 1. Test data compression methodology

Dictionary-based test data compression has been used by Li et al. [1]. To improve compression efficiency as well as fast decompression mechanism, embedded systems uses the test data compression techniques. However the efficiency of these techniques increases only when the number of bits allowed to mismatch are high. Many recently proposed techniques [5] have tried to improve the dictionary based compression techniques by considering mismatches. Bitmask based compression [6] creates more matching patterns with the aid of bitmasks, while taking care of size of compressed code. We propose a test data compression algorithm that combines the dictionary selection [1] and bitmask based [6] techniques.

In bitmask based compression of test data, the primary concern is the presence of don't cares in test data set. Since bitmask based compression technique [6] was not designed for data with don't care values. This technique on test data does not result in a good compression efficiency. Our approach solves these problems by selecting dictionary selection and effective bitmask algorithms to improve compression efficiency and testing time. We demonstrate these techniques in the coming sections with the existing techniques [1], [6]. Our experimental results shows the compression efficiency as 33% greater than existing approaches.

In this paper the remaining papers describes about the reduction of test data into scan chains, dictionary selection method, bitmask based compression and decompression of test data, experimental results and finally conclusion of this paper.

II. RELATED WORK

Test data compression is an important factor in higher density circuits. Hence several techniques came for reduction of test data. Some of them are heterogeneous compression technique [15], multilevel Huffman coding [17], variable to variable Huffman coding [19], selective Huffman coding [10], run length Huffman coding [11], tunstall coding [12], FDR coding [16]. We have compared our technique with these approaches using ISCAS'89 benchmarks.

The test data volume in SOCs can be reduced by using dictionary-based compression technique. Reddy et al. [22] and Li et al. [1] used fixed length dictionary entries to reduce test data volume. Hellebrand et al. [24] have proposed a test data compression method by remembering the mismatches with the dictionary entries. We have proposed a bitmask based compression technique, which gives better result than the existing Li et al. [1]. Bitmask based compression was developed by seong et al. [6] for code compression in embedded systems. We propose an efficient bitmask based code compression technique in our test data compression. From our results it shows that the compression efficiency is improved on comparing with existing bitmask based compression.

Hillenbrand et al. [24] have proposed a test data compression technique which is some what similar to our approach to remember the mismatches from the dictionary entries. However there are some differences between the two. While they try to remember the positions of the bit changes. Our method uses the bitmask and it can also compress the test data which is having the don't cares. Our approach performs better compression efficiency while uses simpler decompression engine compared to them.

III. DICTIONARY BASED CODE COMPRESSION

This section describes about the bitmask based code compression . usin the dictionary methods and highlight the challenges evolved in bitmask based technique for test data compression to reduce the testing time. Following figure describes about the bitmask selection.

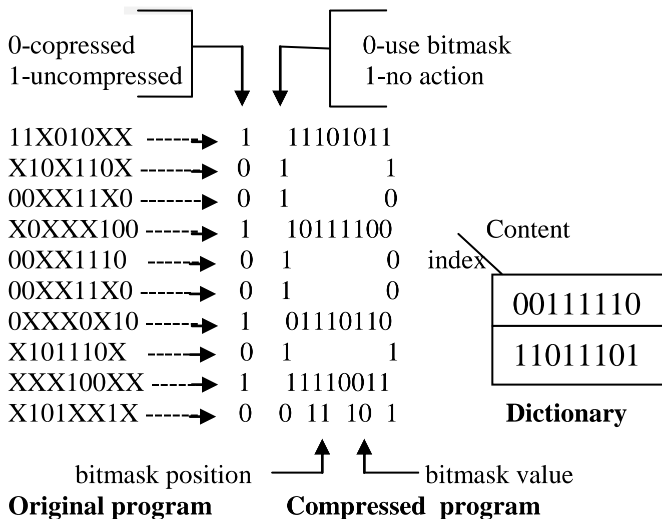


Fig. 2 Bitmask-based test data compression.

A. Test data compression using bitmask.

Dictionary based compression is one of the compression technique. With this compression technique test data size is not completely reduced. So further best compression is the combination of bitmask and dictionary selection methods.

In dictionary based compression, each vector is compressed only if it is completely matches with a dictionary entry. We can compress the test data up to six data entries using bitmask based compression. Original test data is compressed is compressed as fallows. Those test vectors that match directly are compressed with 3

bits. Whether the data is compressed or not is represented with the first bit. The second bit indicates whether it is compressed using bitmask or not. The last bit indicates the dictionary index. In our paper we discuss about the bitmask based test compression technique. In this the test data can also include the don't cares. Data that are compressed using bitmask requires 7 bits. The first two bits, as before, represent if the data is compressed, and whether the data is compressed using bitmasks. The next two bits indicate the bitmask position and followed by two bits that indicate the bitmask pattern.

We are using the fixed bitmasks, which are always employed on even-bit positions and hence only2two bits are sufficient to represent the four positions in a 8-bit data. Dictionary index is provided by the last bit. Original data is produced by XORing the dictionary entry with the bitmask. The compression efficiency is improved with this method on comparing with the proposed one.

Since existing approach does not handle don't cares, but in this paper we can use don't cares also to improve compression efficiency. The compression efficiency is reduced if all don't cares are replaced with 0's. hence instead of replacing all with 0's replace either with 0's or 1's.

B. Major role in bitmask based compression

There are three major roles in bitmask based test data compression. These three roles are independent and cant not me solved separately.

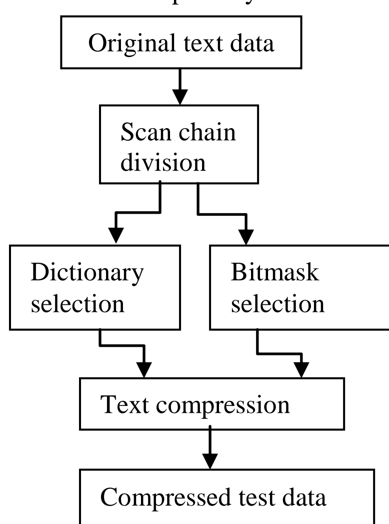


Fig. 3 Bitmask-based test data compression

1) *Dictionary selection*: A profitable dictionary is to be selected which can reduce the number of bits due to frequency matching as well as bitmasks.
2) *Selection of bitmask:* Suitable number and type of bitmasks are to be selected for compression.
3) *Don't care decision*: It is necessary to selectively replace each don't care with either “0” or “1”.

These factors place an important role in test data compression and to determine the performance of an algorithm. At first we have to consider the dictionary selection. For this the frequency of occurrence provided a good base for selction, if any mismatches found that could be simply ignored as un compressed data. In this case bitmask based compression is used. Where mismatched data could also be compressed using bitmasks.

Hence to choose a number and type of bitmasks for compression a particular data vector is important. There are different types of bitmasks we have to select particular bitmasks that are most suitable for compressing a particular group of data. Ther is an another important role in this bitmask that is selective don't care resolution in bitmask and dictionary selection. If we consider two vectors even though there is an lot of dissimilarity in those that can be matched and finally test data can be compressed. This is because the don't care can be either “0” or “1”.

IV. TEST DATA COMPRESSION

This chapter we are going to discuss about the test data compression using bitmask. We have developed an efficient test data compression using bitmask and dictionary selection methods. The test data compression is expained with the flow chart as fallows. It has four major steps, first the input test data is devided into number of scan chains i. e the uncompressed test data into equal number of slices. Second step is to generate the dictionary entries and third is bitmask selection algorithmprovides the number and type of profitable bitmasks. Then finally test compression using don't care resolution, dictionary and bitmask selection.

A. *Compression algorithm*

Algorithm 1 shows the four basic steps of our algorithm.

Inputs: Original test data

Output: Dictionary and compressed data

Begin

1: Scan chain division

2: Bitmask selection

3: Perform dictionary selection

4: Test data compression using bitmask selection and dictionary

Return compressed test data and dictionary

End

Input test data is divided into scan chains of predetermined length. If the test data consists of n test patterns, those are divided into m scan chains in the best balanced manner possible. That means each vector is divided into m sub-vectros, each of length l. if the length of the sub-vectros are not same then append zeros at the end of the each sub-vector. If ther are n vectors at the beginning, we obtain a total of n x l m-bit slices, which is our uncompressed data set that needs to be compressed.

If we consider an example consisting of two test patterns 1110 and 1XX1 for a design with two scan chains. Therefore, length of each sub-vector l = 2. In this case, padding of don't cares is not required. The fallowing figure explains about slices formation with the two vectors obtained by scan chain based partitioning of the two original test patterns. These are the required four slices that need to be compressed.

B. *Methods in bitmask selection*

In bitmask selection compressed data stores information regarding the bitmask type, bitmask location and bitmask pattern. The bitmask can be applied on different places on a vector and the number of bits required for indicating the position varies depending on the bitmask type. We need 5-bits to indicate any starting position on a 32-bit vector, if we do not restrict the placement of the vector. But in this paper we propose an bitmask selection with only byte boundaries requires 2-bits to represent four locations.

There are two methods of bitmasks sliding and fixed. Fixed bitmasks are referred with the letter f while sliding bitmasks are referred with the letter s. to represent the length of the bitmask that is the coefficient of that reference. For example 2s means sliding bitmask with bitmask length 2.A fixed bitmask is one which can be applied to fixed locations where as sliding bitmask can be applied to anywhere in the test vector. So the number of bits required to represent the fixed bitmask are less on compared with the sliding bitmask.

The performance of the test data compression is reduced if we use the bitmask type of length 4. This is because the probability that four corresponding contiguous bits will differ in a pair of test data is only 0.25%, which can easily be neglected. Therefore it is not profitable to use bitmask type of length 4. Hence bitmask type of length below 4 is used to get better performance. The number of bitmask selected is dependet on the lemma rule.

The number of bitmasks is dependent on vector length and dictionary entries. Let l be the number of dictionary entries and N be the vector length. If y be the number of bitmasks allowed, then the number of bits required is less than N.

The first two bits are used to represent whether the data is compressed or not, and if compressed , whether bitmask is used or not. The maximum number of bitmasks allowed can be found by equating the above expression with N, so that the worst case condition holds. If y < N then the value of y can be found easily by equating the right most value with 1. After this we can find the maximum number of bitmasks that are profitable.

C. *Dictionary selection*

Dictionary selection is a critical part in bitmask based test data compression. We used the clique partitioning algorithm of graph theory[12]. The clique partitioning is already discussed in the previous methods it is basically refers to the breaking up of a graph into several cliques, such that the nodes within one clique are all interconnected.

A graph G is drawn with nodes that are separated with the scan chain as n X l nodes, where each node signifies a m-bit test vector. If they are compatible then draw an edge between the two nodes. Two nodes are compatible if and only if they meet the following requirements. First for all positions, the corresponding characters in the two vectors are either equal or one of them is a don't care; or second condition is two vectors can be matched by predetermined profitable bitmasks. Each edge also contains weight information. The weight is determined based on the number of bits that can be saved by using that edge.

Algorithm 2 Selection of dictionary entries

Inputs: 1. Number of entries allowed, N

2. Dictionary entries, S
3. Original test data vectors, V

Output: Dictionary entries
Begin
Number of profitable entries = { };
1: for each dictionary entry compute savings by frequency and bitmasks.
2: for count from 1 to N
2.1 Select the entry with maximum savings.
2.2 Profitable entries+ = D;
2.3 S = S-D.
2.4 V = V- entries composed by D.
2.5 Recompute the savings of S using V
Return Profitable entries
End

In this partitioning method, we developed three dictionary selection techniques: 1) two step method (TSM), 2) method using compatible edges (MCE). 3) method using compatible dges with edge weights. All these techniques uses a clique partitioning algorithm. We will discuss these techniques in detail as follows.

1) *Two-step method*: in this method dictionary selection

Is done based on the edges that are formed by direct matching. The graph will not have any edges corresponding to bitmask-based matching. then a clique partitioning algorithm[5] is performed on the graph. In this procedure it selects the node with the largest connectivity and is entered as the first entry to a clique. Now, the nodes connected with it are analysed, and the node having the largest connectivity is selected. This procedure have to be followed until no node remains to be selected. That means the nodes present in the partitioning are completed. The entries present in the clique are deleted from the graph. This algorithm is repeated until the graph becomes empty.

We have already some predefined number of dictionary entries, hence two possibilities may arise. The number of cliques selected may be greater than the predefined number of entries or vice versa. In the next case we need to fill in the dictionary entries with those obtained from clique partitioning.
If the number of cliques is larger, we have to select the best dictionary entries, that is explained in the coming algorithms in this paper. With this method we have selected the dictionary based on the maximum overall savings.

2) *MCE without edge weights*: in this method weight of

All the edges are considered equal. Clique partitioning is performed as the same procedure followed in the two-step method.

3) *MCE with edge weights*: Method using compatible

Edges with edge weights is same as without edge weights. In MCE the edge weight is determined based on the number of bits saved if that edge is used for bitmask or direct matching. The total bits saved is taken into account in an clique partitioning. The total savings by each node is obtained as the sum of weights of edges originating from that node.

Table I: Test data sets

Data set	Entry
1	001X011101100110
2	001X101101100110
3	XX01X11X01100110
4	XXX101X101100110
5	1X00X10101100110
6	X1000X0101100110
7	11X001XX01100110
8	0XXX00X101100110
9	001X011101100110
10	000X0XX101100110

These methods are illustrated by considering an example, test data set in table 1.the resultant clique partitioning graph is shown in the figure 6. Direct matching of test data is shown with the straight lines. And the matching with the bitmask is shown with the dotted lines.

The dotted lines will be absent in case of two-step method. The dotted lines will have the same weight as the straight lines for MCE. There are different weights in case of MEW.

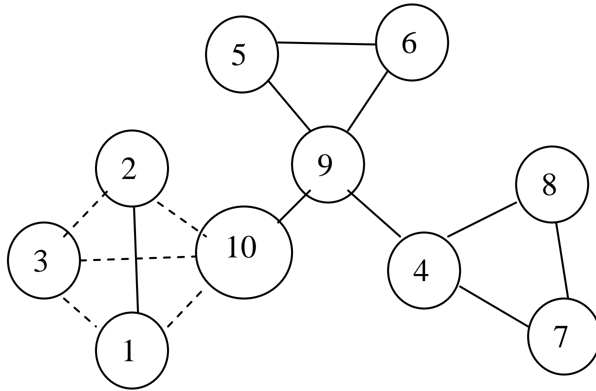


Fig. 4 Graph model for the test data in table I.

Test data taken in an example is compressed using TSM, the cliques selected are {5,6,9} and {2,7, 8}. The compression efficiency is obtained as 47.2%. if we compress using MCE or MEW then clique partitioning selection are {10,2,3,1} and {5,6,9} then the compression efficiency is improved as 48%.

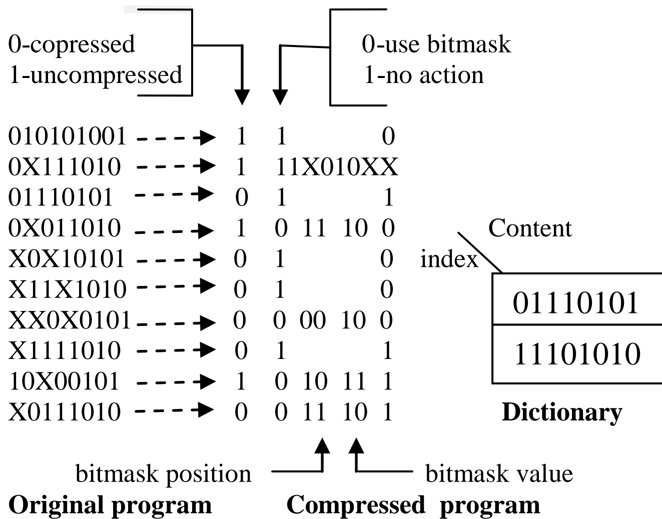


Fig. 5 Compression using bitmask

D. *An example for test data compression*

Test data is compressed now with the selection of the dictionary and bitmask methods and the performance is improved with the usage of these two techniques. Which are already discussed in the previous example. In this example we have taken 10 test vectors and the clique partitioning is applied with these vectors by using one of the methods. Then the performance is improved as 35% more on comparing with the existing methods. From the fig 5. It is described that the original data is an uncompressed test data and the compressed data is an data after applying the methods for test data compression.

V. DECOMPRESSION METHOD

For bitmask encoding decompression engine is proposed, it is shown in the following figure that it can provide fast decompression mechanism on comparing with the proposed one. In this XOR gate is introduced in addition with the decompression scheme for dictionary based compression. The decompression system generates a test data length bitmask, which is then XoRed with the dictionary entry. The generation of bitmask is done in parallel with the dictionary access. With this time consumption is reduced.

The overview of this decompression system is explained with the algorithm as follows. The decompression system takes the compressed vector as input. It checks first bit to see whether the data is compressed or not. If the first bit is “1” it implies that the data is uncompressed. Then it directly sends the uncompressed data to output buffer. If the first bit is “0”, means the data is com[pressed. Now there are two possibilities, they are the data is compressed using bitmask or it is directly compressed with dictionary entry.

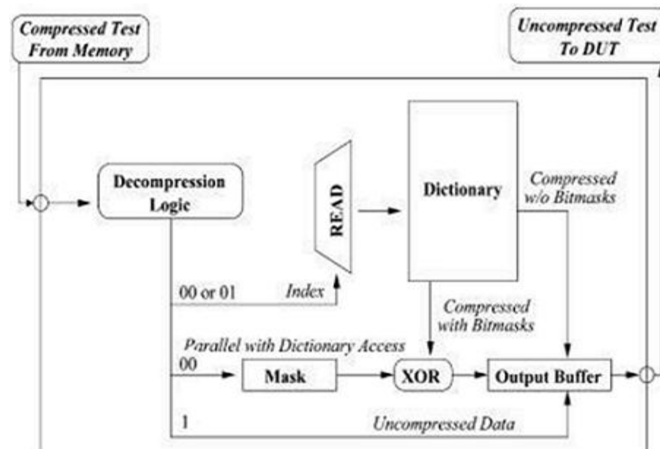


Fig 6.Decompression engine for bitmask-based encoding

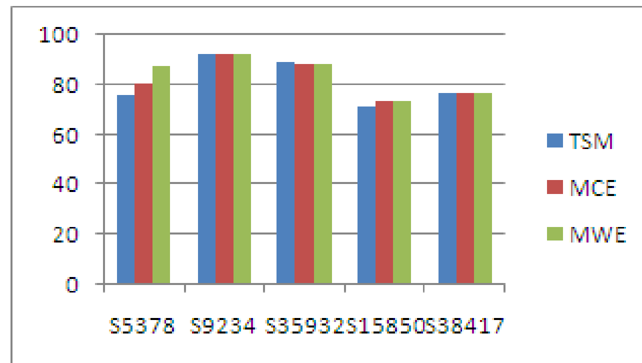
The second bit discuss about these two scenarios. If the second bit is "0", it signifies that it has been matched using only dictionary entry. The index of the dictionary entry is used to read from the dictionary, and the data is transferred to the output buffer. If the second bit is "1", then the system understands that the data is compressed using bitmasks. If it is compressed using bitmask then it needs to know about the bitmask value and bitmask position. Then it extracts the data corresponds to the dictionary entry. Then it creates a bitmask. Finally it creates a vector of length of the uncompressed data. It then finds the position and types of bitmasks and inserts the bitmasks in those positions. Along with that it lookup in the same way as dictionary based compression. Now these two results are XORed to get the uncompressed data, and send it to the output buffer. The result is finally sent to the design under test circuit.

VI. RESULTS

We compare our results with the popular existing methods of compression and decompression methods in accordance with the performance of the system. To demonstrate the usefulness of our method, we have applied it on the tests which were obtained from the MINTEST ATPG program[9] for the five largest ISCAS'89 circuits. Then our results shows an improvement in the performance.

Table II: Compression efficiency

circuits	TSM	MCE	MWE
S5378	75.8	80.6	87.5
S9234	91.9	92	92.3
S35932	89.1	88	88.2
S15850	71.2	73.6	74
S38417	76.5	76.7	76.6



A. *Compression overhead*

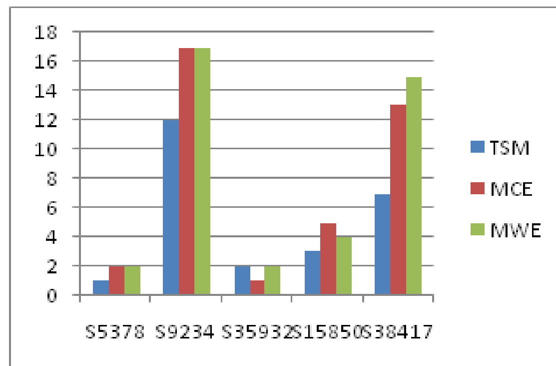
Compression performance of an TSM, MCE and MEW

Using the five largest ATPG programs is shown in the table II. MCE and MEW generates better compression by exploiting compatible edges and thereby selecting larger cliques. However larger cliques may not generate better results when TSM is able to directly match with a large number of test vectors. The performance of the TSM is better on comparing with the others like MCE and MEW. The performance of the TSM is 88% while the performance of the MCE and MEW are 65%.

The performance of the MCE and MEW are analysed by using the s38417. As we see the results the performance of the MEW is better than the MCE, which in turn performs better than TSM. However when we consider matching by one ortwo bitmasks, TSM only covers 8%, while MCE and MEW matches 31.5% and 34% respectively. In some scenarios like s9234, MCE is seen to perform 0.1% better than MEW because MCE is almost 91% of the test vectors are compressed either directly or using bitmasks.

Table III: Time taken for an compression

circuits	TSM	MCE	MWE
S5378	1	2	2
S9234	12	17	17
S35932	2	1	2
S15850	3	5	4
S38417	7	13	15



Run time information for compression of test data cases obtained from the five largest ATPG programs as shown in the table III. MEW is expected to give the best compression performance, but it takes the largest compression time. We compare the compression performance of MEW with those obtained by employing the algorithms of Seong et al.[8] and Li et al.[12]. Fig.6 shows the comparison between the three techniques . We have compared the test data obtained from the largest circuits using the 128 dictionary entries because we have considered only 128 scan chains.

From the figure the first bar represents the compression by using bitmask based compression technique[1] is directly applied for compression the given test data. The second bar represents the compression using dictionary entries of fixed
length. The third bar represents our method, gives the best compression efficiency.

On considering the differences between the applications of our approach and the proposed methods, our approach outperforms the conventional bitmask based approach[1] by 35% -60%. This can be attributed to the introduction of bitmasks in our approach. Bitmasks can allow bit changes and thus matches mor e vectors in the given text data .

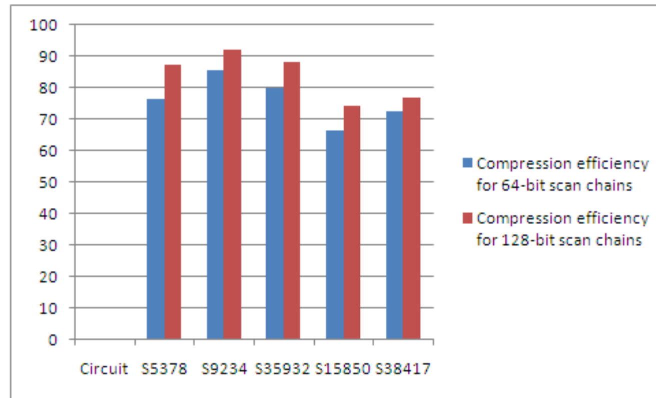
We now compare the compression efficiencies obtained by previous method Li et al.[6] with our approach for the test cases obtained from the five ATPG programs using both 64-bit and 128-bit scan chains. From the table IV it is clear that in all cases our approach provides better compression efficiency than existing methods of compression[6].

The performance of our approach is compared with the selective Huffman coding[14] on test vectors obtained from one of the largest circuits, shown with the results of comparison of the two compression schemes for 64 dictionary entries and variable number of scan chains. It compares the two approaches for 128 dictionary entries and variable number of scan chains. On observing the figure it is clear that our approach works better with greater number of scan chains .

Table IV: Compression efficiency for different scan chians

	Compression efficiency for	Compression efficiency

	64-bit scan chains	for 128-bit scan chains
Circuit	Our approach	Our approach
S5378	76.73	87.55
S9234	85.86	92.22
S35932	80.21	88.31
S15850	66.31	74.51
S38417	72.51	77.12



The compression efficiency is reduced with smaller number of scan chains and larger dictionary entries. It is important to note that our approach performs better than the dictionary based compression scheme in all cases. Even though the selective Huffman coding gives better performance ,its decompression area and performance overhead .

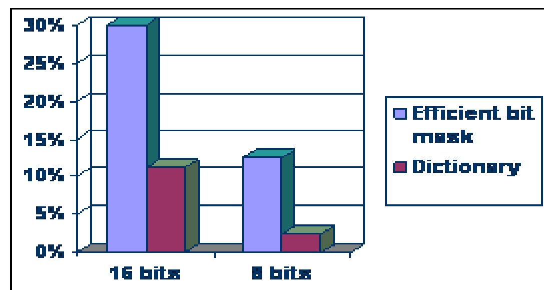
For test data compression our approach outperforms the existing approaches in the presence of don't cares also. Previous proposed methods do not have with the don't cares. We have compared results of our approach with those obtained when compressed using the existing bitmask based compression method[1]. Table V presents the results of test data compression, if we compare these results then it is clear that our approach provides better compression than the bitmask based compression.

The test data sets are combined to get the same number of scan chains and then compress them. This is done in order to test the performance of our compression algorithm on really huge data sets which are formed by the combination of a pair of test data. We create three sets of test data by combining the test data of the circuits in each set. It can be seen that our approach performs up to 30% better than Li et al.[6].

If we compare our approach with the algorithm proposed by Wurtenberger et al.[15] that compress test data by remembering the mismatches from the dictionary entries. Table V shows the comparison results, then it represents our approach performs better compression on compared to them.

Table V: Compression efficiency for different methods

Vector length	Dictionary method(%)	Efficient bitmask method(%)
16	11.25%	30%
8	2.5%	12.5%



Comparison of our approach with various

existing compression techniques is shown in table V. It shows the comparison of our approach with the selective Huffman coding.

B. *Decompression overhead*

We have used the decompression engine to decompress

The compressed data obtained from s9234. These results are compared with those obtained using dictionary based compression and selective Huffman coding compression then it shows the improvement in performance. The results for the 64-bit dictionary entries. And the results for 128-bit dictionary entries are explained with the compression efficiency. If we compare these results, our method requires less area compared to dictionary-based compression.

VII. CONCLUSION

We have shown how bitmask-based compression and dictionary-based compression are used in test data compression to reduce the test data volume for SOC circuits. This paper developed an efficient bitmask and dictionary selection techniques for test data compression in order to create maximum matching patterns in the presence of don't cares. Our test compression technique used bitmask and dictionary selection methods to significantly reduce the testing time and memory requirements. We have compared our results with the existing test data compression algorithms, our algorithm outperforms existing dictionary based compression[6] by up to 35%, giving a best possible compression of 92%. Our approach also generates up to 60% improvement in compression efficiency compared to the existing bitmask-based compression[1] without introducing any additional penalty.

In order to achieve the optimal compression results, it is necessary to provide the data with better compression. This is achieved in our paper with high performance and reducing testing time on comparison with the previous methods of bitmask-based and dictionary-based selections.

REFERENCES

[1]. S. Seong and P. Mishra, "Bitmask-based code compression for embedded systems, "IEEE Trans. Comput.-Aided Des. Integer. Circuits Syst., vol. 27,no. 4,pp.673-685,Apr.2008.

[2]. X. Kavousianos, E. Kalligeros, and D. Nikolos, "Optimal selective Huffman coding for test-data compression," IEEE Trans.Computers,vol.56,no.8,pp.1146-1152,Aug.2007.

[3]. H. Hashempour, L. Schiano,and F.Lombardi,"Error-resilient test data compression using tunstall codes,"in proc. IEEE Int. Symp. Defect Fault Tolerence VLSI Syst.,2004,pp.316-323.

[4]. M. Ros and P. Sutton,"A hamming distance based VLIW/EPIC code compression technique,"in proc. Compilers, Arch., Synth. Embed. Syst., 2004,pp.132-139.

[5]. N. Touba and E. McCluskey,"Altering a Pseudo-random bit sequence for scan based bist," in proc. Int. Test Conf.,1996,pp.167-175.

[6]. L.Li,K.Chakrabarty,and N.Touba,"Test data compression using dictionaries with selective entries and fixed-length indices,"ACM Trans. Des. Autom. Electron. Syst., vol.8,no. 4,pp.470-490,2003.

[7]. S. Reda and A. Orailoglu,"Reducing test application time through test data mutation encoding," in Proc.Des. Autom. Test Eur.,2002,pp. 387-393.

[8]. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, introduction to Algorithms. Boston, MA: MIT Press,2001.

[9]. I. Hamzaoglu and J. Patel,"Test set compaction algorithm for combinational circuits,"in Proc. Int. Conf. Comput.-Aided Des., 1998, pp. 283-289.

[10]. H. Wunderlich and G. Kiefer,"Bit-flipping BIST," in Proc. Int. Conf. Comput.-Aided Des., 1996,pp.337-343.

[11]. S. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz,"On test data volume reduction for multiple Scan chain desaign, in Proc. VLSI Test Symp., 2002, pp. 103-108.

[12]. K. Basu and P. Mishra,"A novel Test-data compression technique using application-aware bitmask and dictionary selection methods," in Proc. ACM Great Lakes Symp. VLSI, 2008,pp.83-88.

[13]. A. Wurtenberger, C. Tautermann, and S. Hellebrand,"Data compression for multiple Scan Chains using dictionaries with correlations," in Proc. Int. Test Conf.,2004,pp.926-935.

[14]. A. Al-Yamani and E. J. McCluskey, "Seed encoding for LFSRs and cellular automata," in *Proc. M/IEEE Des. Autom. Conf.*, Jun. 2003, pp. 560–565.

[15]. 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 Trans. Computers*, vol. 52, no. 8, pp. 1076–1088, Aug. 2003.

[16]. M.-E. N. A. Jas, J. Ghosh-Dastidar, and N. Touba, "An efficient test vector compression scheme using elective Huffman coding," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 6, pp. 797–806, Jun. 2003.

[17]. A. Jas and N. Touba, "Test vector decompression using cyclical scan chains and its application to testing core based design," in *Proc. Int. Test Conf.*, 1998, pp. 458–464.

[18]. M. Nourani and M. Tehranipour, "RL-Huffman encoding for test compression and power reduction in scan pplications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 91–115, 2005.

[19]. X. Kavousianos, E. Kalligeros, and D. Nikolos, "Test data compression based on variable-to-variable Huffman encoding with codeword reusability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*,vol. 27, no. 7, pp. 1333–1338, Jul. 2008.

[20]. L. Lingappan, S. Ravi, A. Raghunathan, N. K. Jha, and S. T. Chakradhar, "Test-volume reduction in systems-on-a-chip using heterogeneous and multilevel compression techniques," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2193–2206, Oct. 2006.

[21]. M. Garey and D. Johnson*, Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[22]. M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in SOCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, pp. 719–731, Jun. 2005.