A Novel Test-Data Compression Technique using Application-Aware Bitmask and Dictionary Selection Methods

Kanad Basu[1] and Prabhat Mishra[2]
Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL-32611
E-mail: [1]kbasu@cise.ufl.edu, [2]prabhat@cise.ufl.edu

ABSTRACT
Higher circuit densities in System-on-Chip (SOC) designs have led to enhancement in the test data volume. Larger test data size demands not only greater memory requirements, but also an increase in the testing time. Test data compression addresses this problem by reducing the test data volume without affecting the overall system performance. This paper proposes a novel test data compression technique using bitmasks which provides a significant enhancement in the compression efficiency without introducing any additional decompression penalty. The major contributions of this paper are as follows: i) it develops an efficient bitmask selection technique for test data in order to create maximum matching patterns; ii) it develops an efficient dictionary selection method which takes into account the speculated results of compressed codes and iii) it proposes a suitable code compression technique using dictionary and bitmask based code compression that can reduce the memory and time requirements. We have used our algorithm on various test data sets and compared our results with other existing test compression techniques. Our algorithm outperforms the best known existing compression technique up to 30%, giving a best possible compression of 92.2%.

Keywords
Test Data, Compression, Decompression.

1. INTRODUCTION

In case of system-on-chip, higher circuit densities have led to larger volume of test data generation, which demands large memory requirement in addition to an increased testing time. Test data compression plays a crucial role, reducing the memory and time requirements. It also overcomes the Automatic Test Equipment (ATE) bandwidth limitation.

The test compression mechanism should allow small number of ATE channels to transfer the compressed data from tester to the chip and be able to drive a large number of internal scan chains. Thus, it would be suitable for reduced pin count and low cost Design for Testability (DFT) environment. Our algorithm is able to achieve all the above advantages without introducing any additional decompression penalty. Compression ratio, widely accepted as a primary metric for measuring the efficiency of code compression, is defined as, Compression ratio = (compressed program size) / (original program size). Clearly, smaller compression ratio indicates better compression.

Dictionary based compression techniques are extremely popular in embedded systems domain since they provide a dual advantage of good compression ratio as well as a fast decompression mechanism. The basic idea is to take advantage of a number of commonly occurring sequences. Test data compression using dictionaries of fixed length entries was proposed by [11]. Many recently proposed techniques [10, 12] have tried to improve the dictionary based compression techniques by considering mismatches. However, the efficiency of these techniques depends on the number of bits allowed to mismatch. It is obvious that if more number of bit changes is allowed, more matching patterns will be generated. However, a serious problem might arise since there is a chance that the final size of the compressed data becomes greater than the original data size. Bitmask based code compression [17] addresses this issue by creating more matching patterns with the aid of bitmasks, while taking care of the size of the compressed code.

Application of bitmask based compression in test data might seem to be attractive, but it presents various challenges. The primary concern is the presence of don't cares ('X') in the test data set. Since bitmask based compression techniques [17] were not designed for these data, application of those techniques to test data does not result in a good compression ratio. We have to determine not only the effective bitmasks, but also select a dictionary that would suffice us with the most optimized result. We demonstrate in Section 5 that selection of bitmasks or dictionary using existing procedures [11, 17] are not appropriate in the case of test data compression using bitmasks. This paper addresses both issues by selecting suitable bitmasks, as well as proposing a suitable dictionary selection algorithm, which improves the compression ratio without introducing any additional decompression penalty. Our experimental results demonstrate that it produces up to 30% better compression compared to the existing dictionary based compression approach [11], which is the best known result on test data compression.

The rest of the paper is organized as follows. Section 2 describes related works in the area of test data compression. Section 3 describes our compression technique. Section 4

GLSVLSI'08, May 4–6, 2008, Orlando, Florida, USA.

describes our decompression mechanism. Section 5 presents the experimental results. Finally, section 6 concludes the paper.

2. RELATED WORKS

Code compression in embedded systems has been an issue of concern for a long time. Different researchers have come up with different compression techniques. The first code compression technique for embedded processors was proposed by Wolfe and Chanin [1], which compressed codes using Huffman-coding. The compressed code was stored in the main memory. A line address table (LAT) was used to map compressed block addresses to the original code addresses. A similar code compression scheme CodePack was introduced by IBM for PowerPC [8] architectures. A code compression using Markov Model was proposed by Lekatsas and Wolf [7]. Lie et al. [6] proposed a LZW-based code compression for VLIW processors using variable sized block method. Dictionary-based compression techniques were explored by Liao [15] and Lefurgy [5]. Ishiura and Yamaguchi [13] proposed a technique where a VLIW instruction is split up into multiple fields and each field is compressed using a dictionary based approach. Nam et al. [16] presented a dictionary based method for an isomorphic VLIW instruction word scheme.

Various techniques have been proposed to improve the standard dictionary based compression by considering mismatches. The basic idea is to store the mismatch information during encoding. Prakash et al [10] considered one bit change for 16-bit vectors. Ros et al [12] considered a generic scheme for 32 bit vectors and stated that up to 3-bit changes are profitable. Bitmask based code compression was used in embedded systems by Seong et al. [17]

Test data compression has also manifested itself as a serious problem for a long time. Different compression techniques have been proposed over the years to reduce the test data volume. Some of them are statistical coding [18], run length coding [4], Golomb coding [3], FDR coding [2] and VIHC coding [14]. Several commercial tools have also been available in the market that utilizes test data compression, some of them being OPMISR, SmartBIST and TestKompress. Dictionary based compression techniques have been recently used to reduce the test data volume in SOCs. Li et al. [11] used fixed length dictionary entries to reduce test data volume. Our algorithm performs bitmask based compression to obtain significant better compression ratio than [11], as demonstrated in Section 5.

3. BITMASK BASED TEST DATA COMPRESSION

We have developed an efficient test data compression algorithm using bitmasks. Though bitmask based test data compression might seem to be a promising scheme, there are some major challenges in this method like selection of appropriate bitmasks, dictionary and development of the compression technique. Subsection 3.1 illustrates why bitmask based test compression performs better than ordinary dictionary based test data compression [11]. The other subsections describe three important components of our approach: determination of uncompressed data sets, bitmask selections and dictionary selection techniques respectively.

3.1 A Motivational Example

We consider a test data set of 8-bit entries. The total number of entries is 10. Therefore, the total test set is of 80 bits. Figure 1 shows the data set as well as the compressed data set under the application of dictionary based compression. In this case, the dictionary has 2 entries, each of 8-bits length. Each repeating pattern is replaced with a dictionary index. (In this example, an index of 0 refers to the first dictionary entry and an index of 1 refers to the second one.) The final compressed test data set is reduced to 55 bits and the dictionary requires 16 bits. Thus, the compression ratio obtained is 68.75%.

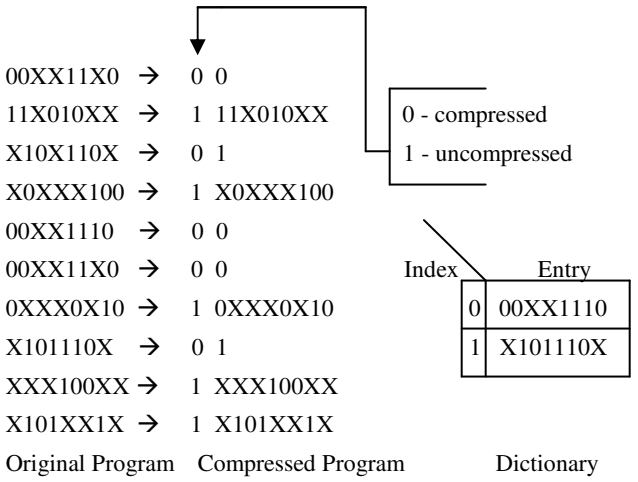

Figure1. Dictionary based Test Data Compression

We now attempt to compress the same data set using our bitmask based code compression. The compressed data as well as the dictionary entries are shown in Figure 2 We have used a 2-bit mask, only on quarter-byte boundaries. It is seen that such a mask is able to create 90% matching patterns. The compression ratio is found to be 65%, which is better than the dictionary based compression method as was described earlier.

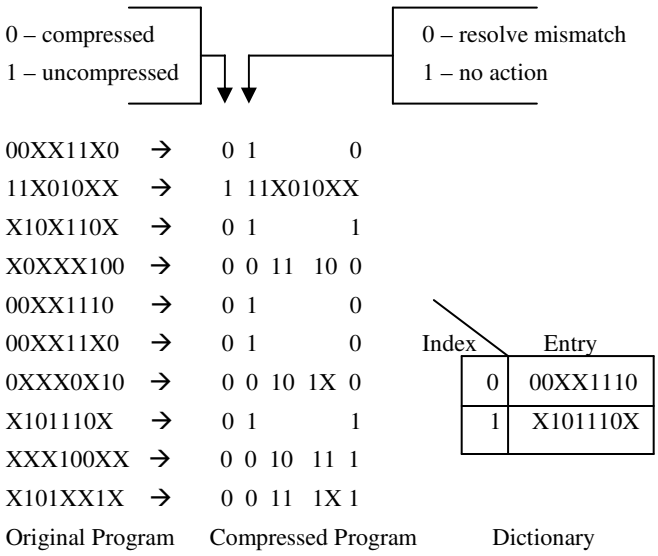

Figure 2. Bitmask Based Test Data Compression

3.2 Uncompressed Data Set Determination

Once we get the total test data, our next task would be to divide them into scan chains of pre-determined length. We perform this step in accordance with the method prescribed by Li et al. [11]. Let us assume that the test data T_D consists of n test patterns. If we choose to have the uncompressed data as a group of m-bit words, we divide the scan elements into m-scan chains in the best balanced manner possible. This results in each vector being divided into m sub-vectors. Dissimilarity in the lengths of the sub-vectors are resolved by padding don't cares to the end of the shorter sub-vectors. Thus, all the sub-vectors are of equal length, which is denoted by l. The m-bit data which is present at the same position of each sub-vector constitute an m-bit word. Thus, we obtain a total of nxl m-bit words, which is our uncompressed data set that needs to be compressed. Figure 3 shows how two 4-bit words are obtained from a 8-bit long test pattern.

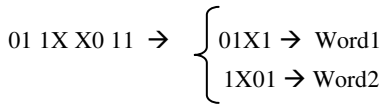

Figure 3. Determination of two 4-bit words from a 8-bit word

In this example, m = 4 and l = 2. It is to be noted that since the words were balanced, padding of don't cares was not necessary here.

3.3 Mask Selection

Figure 4 shows the generic encoding scheme of bitmask based compression technique.

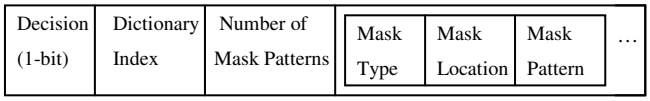

Figure 4. Generic Encoding Format

A compressed code stores information regarding the mask type, mask location and the mask pattern itself. The mask can be applied on different places on a vector and the number of bits required for indicating the position varies depending on the mask type. For instance, if we consider a 32-bit vector, an 8-bit mask applied on only byte boundaries requires 2-bits, since it can be applied on four locations. If we do not restrict the placement of the mask, it will require 5 bits to indicate any starting position on a 32-bit vector.

Bitmasks may be sliding or fixed. A fixed bit mask always operates on half-byte boundaries while a sliding bitmask can operate anywhere in the data. It is obvious that generally sliding bitmasks require more bits to represent themselves compared to fixed bitmasks. In this paper, we use the alphabets `s` and `f` to represent sliding and fixed bitmasks respectively.

As shown by Seong et al. [17], the optimum bitmasks to be selected for code compression are 2s, 2f, 4s and 4f. However, in the case of test data compression, the last two need not be considered. This is because as per Lemma 1, the probability that 4 corresponding contiguous bits will differ in a set of test data is only 0.02%, which can easily be neglected. Thus, we perform our compression by using only 2s and 2f bimtasks. The number of masks selected depends on the word length and the dictionary entries and is found out using Lemma 2.

Lemma 1: The probability that 4 corresponding contiguous bits differ in two test data is 0.2 %.

Proof: For two corresponding bits to differ in a set of test data, none of them should be don't cares. Let us consider the scenario in which they really differ, and find out the probability of such an event. We can see that any position in a test data can be occupied by 3 different symbols, 0, 1 and X. However, as already mentioned, to differ, the positions should be filled up with 0 or 1. Hence, the probability that a certain portion is occupied by either 0 or 1 is 2/3 = 0.67. Therefore, the probability that all the four positions have either 0 or 1 is

$P1 = (0.67)^4 = 0.20.$

For the other vector, the same rule applies. The additional constraint here is that the bits in the corresponding positions are fixed due to difference in the two vectors, that is, the bits in the second vector has to be exact complement of those of the first vector. Therefore, the probability of occupancy of a single position is 1/3 = 0.33

Therefore, the probability of 4 mismatches in the second vector =

$P_2 = (0.33)^4 = 0.01$

The cumulative probability of the 4-bit mismatch is a product of the two probabilities P_1 and P_2 and is given by:

$P = P_1 \text{ X } P_2 = 0.2\ \%$

***Lemma 2*:** The number of masks used is dependent on the word length and dictionary entries.

Proof: Let L be the number of dictionary entries and N be the word length. If y is the number of masks allowed, then in the worst case (when all the masks are 2s), the number of bits required is,

$$\text{no\_bits} = 2 + \log(L) + \frac{\log(y)}{\log(2)} + y \text{ X } \left(2 + \left(\frac{\log(N)}{\log(2)}\right)\right)$$

and this should be less than N. The first two bits are required to check whether the data is compressed or not, and if compressed, mask is used or not. So, the maximum number of bitmasks allowed is

$$y = \frac{N - 2 - \log(L)}{2 + \frac{\log(N)}{\log(2)}} - \frac{\frac{\log(y)}{\log(2)}}{2 + \frac{\log(N)}{\log(2)}}$$

We can see that it is not easy to compute y from here since both sides of the equation contain y related terms. To ease our calculation, we can replace the y-related term on the right hand side of the equation with a constant. It is to be noted that since y<N, a safe measure would be to use 1 as this constant. Therefore, the final equation for y is:

$y = \left(\frac{N - 2 - \log(L)}{2 + \frac{\log(N)}{\log(2)}} - 1 \right)$, floored to the nearest integer.

3.4 Dictionary Selection

The dictionary selection algorithm is a critical part in bitmask based code compression. Our dictionary selection algorithm is a two-step process. The first step is similar to that used by [11]. Our dictionary selection algorithm uses the classical clique partitioning algorithm of graph theory. A graph G is drawn with nxl nodes, where each node signifies a m-bit test word. We now check for the compatibility between the words. Two words are said to be compatible if for a particular position, the corresponding characters in the two words are either equal or one of them is a don't care. If two nodes are mutually compatible, an edge is drawn between them. Cliques are now selected from this set. The clique-partitioning algorithm for our purpose is described as follows:

1. Copy the graph G to a temporary data structure G'.
2. The vertex in G' which has the maximum number of edges is selected. Let's denote it by v.
3. We create a subgraph that contains all the vertices connected to v.
4. This subgraph is copied to G' and v is added to a set C.
5. If (G'==NULL), the clique C has been formed, else go to step 2.
6. G = G-C
7. If (G==0) STOP, else go to Step 1.

At this point, two possibilities may arise. We have a predefined number as to the count of the dictionary entries. The number of cliques selected may be greater than that or vice versa. In the latter case, we just need to fill in the dictionary entries with those obtained from clique partitioning.

However, if the number of cliques is larger, we have to select the best dictionary entries out of them. To accomplish this, we perform the following steps:

1. For each entry, calculate the number of bits saved over the entire data set by compression if that entry was present in the dictionary. The number of bits saved should account those due to bitmask based compression as well.
2. For each entry in the dataset, choose the dictionary entry which gives the maximum compression. If two entries give the same compression, the one which has the maximum saved bits over the entire dataset is given preference. For all the other dictionary entries, the bit savings are deducted. This step is used to prevent aliasing.
3. Sort the dictionary entries in descending order of bits saved.
4. If the dictionary was predefined to have L entries, choose the best L dictionary entries.

The following example shows our dictionary selection algorithm. Table 1 shows the different data sets we have taken into consideration. As seen, there are 16 sets of data, each of 8-bits.

Table 1. Data Sets

Data Set	Entry	Data Set	Entry
1	11X001XX	9	0XX0X10X
2	01X00X1X	10	1X11X01X
3	1101XXX1	11	1XX10001
4	01X01X1X	12	X1X0XX11
5	XX10001X	13	11X000XX
6	X110X0XX	14	01XX0110
7	0101XX1X	15	010X0X01
8	0X00X110	16	1XXX0011

We proceed to find the dictionary by performing the clique partitioning algorithm. The graph drawn for this purpose is shown in Fig. 5.

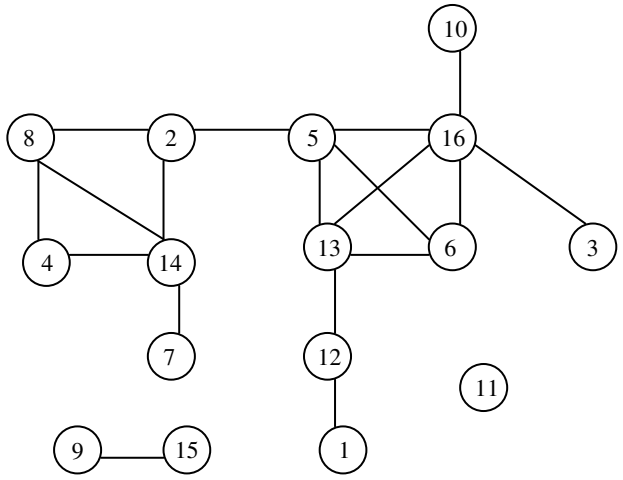

Figure 5. Graph for dictionary selection algorithm

The cliques selected in this case are {5, 6, 13, 16} and {2, 8, 14}. The dictionary entries obtained are {11100011, 01000110}. The original data was of 128 bits. The data when compressed using ordinary dictionary selection algorithm as proposed by Li et al. [11] was of 95 bits, which corresponds to a compression ratio of 74.21%. However, when it is compressed using bitmask based compression, using 2-bit fixed bitmask, the compressed data obtained is of 86 bits, which corresponds to a compression ratio of 67.19%, thus providing a significant advantage in compression.

4. DECOMPRESSION MECHANISM

We have proposed the design of a decompression engine (DCE) that can easily handle bitmasks and provide fast decompression. The design of our engine is based on the one cycle decompression engine proposed by Seong et al. [17]. The decompression engine has also been motivated by that of Li et al. [11], the only addition being the introduction of XOR gate in addition to the decompression scheme for dictionary based compression. The

most interesting feature of the decompression engine is the generation of a test data length mask, which is then XOR-ed with the dictionary entry. The test data length mask is created by applying the bitmask on the specified position in the encoding. The generation of our mask is done in parallel with accessing the dictionary, thus reducing additional penalty. The DCE can decode more than one instruction in one cycle.

5. EXPERIMENTS

To find out the efficacy of our algorithm, we have applied it on the ISCAS-89 circuits which were obtained from the MINTEST ATPG program [9]. We have compared our results with those obtained by employing the algorithms of Li et al. [11] and Seong et al. [17], which are the best known related works on test data compression and code compression in embedded systems respectively. Figure 6 shows the bar chart obtained for comparison between the 3 techniques. Here, we have considered only 128 bits words. We have compared for the 5 largest circuits. All the dictionaries have been selected to have 128 entries each.

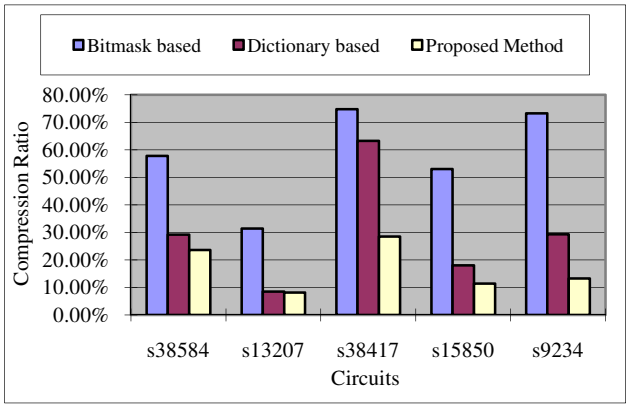

Figure 6. Compression Ratio for different circuits

The first bar represents the compression if bitmask based code technique [17] is directly applied for compressing the test data. The second bar represents the compression using dictionaries of fixed-length indices [11]. The third bar, which represents our proposed method, shows that it gives the best compression ratio. The key point to be noted here is the difference between the applications of ours and the algorithm of Seong et al. [17]. Although both rely on bitmasks, yet, due to differences in the dictionary selection algorithm, our approach outperforms the conventional bitmask based approach [17] by 30 – 60%. It is also evident that our proposed algorithm significantly outperforms (up to 30%) the existing dictionary based compression algorithm [11]. This can be attributed to the introduction of bitmasks in our technique. Bitmasks allow bit changes (which was absent in the dictionary based compression) and thus matches a more number of words. Clearly, more number of words can be compared as compared to the dictionary based approach.

We would now like to compare the compression ratios obtained by Li et al. [11] with ours for all the 7 ATPG circuits that they have considered. Table 2 shows the exact comparison for 64 and 128 bit words.

Table 2. Comparison of compression ratio for different circuits

Circuit	64-bit word		128-bit word	
	Dictionary Based	Our Method	Dictionary Based	Our Method
s5378	26.71 %	20.52 %	37.02 %	11.69 %
s9234	32.54 %	24.68 %	29.28 %	13.22 %
s13207	14.57 %	15.27 %	8.47 %	8.12 %
s15850	24.12 %	21.44 %	18.02 %	11.35 %
s35932	26.24 %	21.32 %	11.07 %	7.80 %
s38417	57.27 %	37.09 %	63.25 %	28.50 %
s38584	29.23 %	25.92 %	29.23 %	23.57 %

The results show that in almost all cases, bitmask based compression gives a better compression ratio than dictionary based compression, the best known result in test data compression.

The previous experiments have demonstrated that our method outperforms the existing approaches for test data compression in the presence of don't cares in test data. We have also applied our algorithm in test data sequences which do not have don't cares. These circuits are some of the MINTEST ATPG circuits. We have compared our results with those obtained when compressed using the ordinary bitmask based compression method [17].

Table 3. Compression ratio for different circuits.

Circuit	Number of Dictionary Entries	Best Compression Ratio	
		Ordinary bitmask based method	Proposed Algorithm
c1355	16	65.73%	55.89%
c499	16	40.85%	38.32%
c6288	16	66.36%	60.63%

Table 3 presents the results of test data compression (without don't cares). Our algorithm provides much better compression ratio than the bitmask based compression [17]. This shows the

efficacy of our algorithm as compared to that proposed by [17], even when dealing with data having only binary values.

6. CONCLUSION

Test data compression is a very critical problem. Different researchers have proposed different compression algorithms over the years to compress test data. In this paper, we proposed a compression algorithm which utilizes the bitmask based compression technique to compress test data. The contribution of the paper is two-fold. It describes a suitable bitmask selection technique for test data, and proposes a dictionary selection algorithm which is appropriate for test data compression. This algorithm ensures that no additional decompression penalty is introduced. We applied this algorithm on test data containing don't cares, as well as those not containing them. In both cases, our technique achieved significant improvements compared to existing approaches – up to 30% better than the best known compression algorithms used in this domain, giving a best known compression of 92.2%. Matching sequences of upto 99% have been achieved using our algorithm. In the future, we plan to study the impact of bitmask-based compression on overall cost, power and performance.

7. REFERENCES

[1] A.Wolfe and A.Chanin. Executing compressed programs on an embedded RISC architecture. In Proceedings of International Symposium on Microarchitecture (MICRO), pages 81-91, 1992.

[2] A. Chandra and K. Chakrabarty. Frequency-directed run-length (FDR) codes with application to system-on-chip test data compression. In Proceedings of the VLSI test Symposium, pages 42-47, 2001.

[3] A. Chandra and K. Chakrabarty, System on a chip test data compression and decompression architectures based on golomb codes. IEEE Transactions on Computer Aided Design, 20: 355-368, 2001.

[4] A. Jas and N.A. Touba. Test vector decompression using cyclical scan chains and its application to testing core based design. In Proceedings of International Test Conference, pages 458-464, 1998.

[5] C. Lefurgy, P. Bird, I. Chen and T. Mudge. Improving code density using compression techniques. In Proceedings of International Symposium on Microarchitectures (MICRO), pages 194-203, 1997.

[6] C. Lin, Y. Xie and W. Wolf. LZW- based code compression for VLIW embedded systems. In Proceedings of Design Automation and Test in Europe (DATE), pages 76-81, 2004.

[7] H. Lekatsas and W. Wolf. SAMC: A code compression algorithm for embedded processors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18(12): 1689-1701, December 1999.

[8] http://www.ibm.com. CodePack PowerPC Code Compression Utility User's Manual. Version 3.0, 1998.

[9] I. Hamzaoglu and J. H. Patel. Test set compaction algorithm for combinational circuits. In Proceedings of the International conference on CAD, pages 283-289, 1998.

[10] J. Prakash, C. Sandeep, P. Shankar and Y. Srikant. A simple and fast scheme for code compression for VLIW processors. In Proceedings of Data Compression Conference (DCC), page 444, 2003.

[11] L. Li, K. Chakrabarty and N. Touba. Test data compression using dictionaries with selective entries and fixed-length indices. ACM Transactions on Design Autmation of Electronic Systems (TODAES), 8(4): 470-490, October 2003.

[12] M Ros and P. Sutton. A hamming distance based VLIW/EPIC code compression technique. In Proceedings of Compilers, Architectures, Synthesis for Embedded Systems (CASES), pages 132-139, 2004.

[13] N. Ishiura and M. Yamaguchi. Instruction code compression for application specific VLIW processors based on automatic field partitioning. In Proceedings of Synthesis and System Integration of Mixed Technologies (SASIMI), pages 105-109, 1997.

[14] P.T. Gonciari , B. Al-Hashimi and N. Nicolai. Improving compression ratio, area overhead, and test application time for system-on-chip test data compression/decompression. In Proceedings of Design, Automation and Test in Europe (DATE), pages 604-611, 2002.

[15] S. Liao, S. Devadas and K. Keutzer. Code density optimization for embedded DSP processors using data compression techniques. In Proceedings of Advanced Research in VLSI, pages 393-399, 1995.

[16] S. Nam, I. Park and C. Kyung. Improving dictionary-based compression in VLIW architectures. IEICE Transactions Fundamentals, E82-A(11):2318-2324, November 1999.

[17] Seok-Won Seong and Prabhat Mishra. An Efficient code compression technique using application aware bitmask and dictionary selection methods. In Proccedings of Design, Automation and Test in Europe (DATE) , 2007.

[18] V. Iyengar, K. Chakrabarty and T. B. Murray. Deterministic built in pattern generation for sequential circuits. Journal of Elcect. Test: Theory and Applications, 15: 97-115, 1999.