Relating Entropy Theory to Test Data Compression

Kedarnath J. Balakrishnan and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712
Email: {kjbala, touba}@ece.utexas.edu

Abstract

The entropy of a set of data is related to the amount of information that it contains and provides a theoretical bound on the amount of compression that can be achieved. While calculating entropy is well understood for fully specified data, this paper explores the use of entropy for incompletely specified test data and shows how theoretical bounds on the maximum amount of test data compression can be calculated. An algorithm for specifying don't cares to minimize entropy for fixed length symbols is presented, and it is proven to provide the lowest entropy among all ways of specifying the don't cares. The impact of different ways of partitioning the test data into symbols on entropy is studied. Different test data compression techniques are analyzed with respect to their entropy bounds. Entropy theory is used to show the limitations and advantages of certain types of test data encoding strategies.

1. Introduction

The rapid increase in test data volume is a major challenge for testing system-on-a-chip (SOC) designs. Reducing the test data volume by compression techniques is an attractive approach for dealing with this problem. The test data can be stored in compressed form on the tester and then decompressed using an on-chip decompressor.

A number of test vector compression schemes have been developed using a variety of codes. Codes can be classified into four categories depending on whether they encode a fixed or variable number of bits in the original data using either a fixed or variable number of bits in the encoded data. Each of the four categories are listed below:

Fixed-to-Fixed Codes: These codes encode fixed size blocks of data using smaller fixed size blocks of encoded data. Conventional LFSR reseeding [Könemann 91] falls into this category where each fixed size test vector is encoded as a smaller fixed size LFSR seed. Techniques that use combinational expanders with more outputs than inputs to fill more scan chains with fewer tester channels each clock cycle fall into this category. These techniques include using linear combinational expanders such as broadcast networks [Hamzaoglu 99] and XOR networks [Bayraktaroglu 01], as well as non-linear combinational expanders [Reddy 02], [Li 03]. If the size of the original blocks is n bits and the size of the encoded blocks is b bits, then there are 2^n possible symbols (original block combinations) and 2^b possible codewords (encoded block combinations). Since b is less than n, obviously not all possible symbols can be encoded using a fixed-to-fixed code. If $S_{dictionary}$ is the set of symbols that can be encoded (i.e., are in the "dictionary") and S_{data} is the set of symbols that occur in the original data, then if $S_{data} \subseteq S_{dictionary}$, it is a complete encoding, otherwise it is a partial encoding. For LFSR reseeding, it has been shown in [Könemann 91], that if b is chosen to be 20 bits larger than the maximum number of specified bits in any n-bit block of the original data, then the probability of not having a complete encoding is less than 10^{-6}. The technique in [Reddy 02] constructs the non-linear combinational expander so that it will implement a complete encoding. For techniques that do not have a complete encoding, there are two alternatives. One is to constrain the ATPG process so that it only generates test data that is contained in the dictionary (this is used in [Bayraktaroglu 01]), and the other is to bypass the dictionary for symbols that are not contained in it (this is used in [Li 03]). Bypassing the dictionary requires adding an extra bit to each codeword to indicate whether it is coded data or not.

Fixed-to-Variable Codes: These codes encode fixed size blocks of data using a variable number of bits in the encoded data. Huffman codes are in this category. The idea with a Huffman code is to encode symbols that occur more frequently with shorter codewords and symbols that occur less frequently with longer codewords. A method was shown in [Huffman 52] to construct the code in a way that minimizes the average length of a codeword. The problem with a Huffman code is that the decoder grows exponentially as the block size is increased. In [Jas 99, 03], the idea of a selective Huffman code was introduced where partial coding is used and the dictionary is selectively bypassed. This allows larger block sizes to be efficiently used.

Variable-to-Fixed Codes: These codes encode a variable number of bits using fixed size blocks of encoded data. Conventional run-length codes are in this category. A method for encoding variable length runs of 0's using fixed size blocks of encoded data was proposed in [Jas 98]. The LZ77 based coding method in [Wolff 02] also falls in this category. Note that in [Wolff 02], it is a partial encoding that uses a bypass mode.

Variable-to-Variable Codes: These codes encode a variable number of bits in the original data using a variable number of bits in the encoded data. Several techniques that use run-length codes with a variable number of bits per codeword have been proposed including using Golomb codes [Chandra 01a], frequency directed codes [Chandra 01b], and VIHC codes [Gonciari 02]. One of the difficulties with variable-to-variable codes is synchronizing the transfer of data from the tester. All of the techniques that have been proposed in this category require the use of a synchronizing signal going from the on-chip decoder back to the tester to tell the tester to stop sending data at certain times while the decoder is busy. Fixed-to-fixed codes do not have this issue because the data transfer rate from the tester to the decoder is constant.

This paper investigates the fundamental limits of test data compression, i.e., it looks to answer the question: *how much can we compress?* This is done by looking at the entropy of test sets and its relation to the compression limit. The idea of using entropy for calculating the test data compression limits was first studied in [Chandra 02]. However, in that work, the entropy calculations were limited to a special case related to FDR codes. All the don't cares were specified as 0's and only one set of symbols was considered. Different ways of specifying the don't cares and different symbol sets will lead to different entropies for the same test set. In this paper, we investigate entropy including the additional degrees of freedom that were not explored in [Chandra 02]. In this paper, a procedure for calculating the minimum entropy of a test set over all possible ways of specifying the don't cares is described. Also, the relationship between entropy and symbol partitioning is studied.

Using entropy theory, we can derive theoretical limits on the compression that can be achieved using various types of coding techniques. This is useful in identifying how much room for improvement there is for the various test data compression techniques that have been proposed. It is also useful to identify the compression techniques that have a lot of room for improvement and offer scope for fruitful research.

2. Entropy Analysis for Test Data

Entropy is a measure of the disorder in a system. The entropy of a set of data is related to the amount of information that it contains which is directly related to the amount of compression that can be achieved. Entropy is equal to the minimum average number of bits needed to represent a codeword and hence presents a fundamental limit on the amount of data compression that can be achieved [Cover 91]. The entropy for a set of test data depends on how the test data is partitioned into symbols and how the don't cares are specified. These two degrees of freedom are investigated in this section.

2.1 Partitioning Test Data into Symbols

For fixed-to-fixed and fixed-to-variable codes, the test data is partitioned into fixed length symbols (i.e., each symbol has the same number of bits). The entropy for the test data will depend on the symbol length. Different symbol lengths will have different entropy. For a given symbol length, the entropy for the test data can be calculated and will give a theoretical limit on the amount of compression that can be achieved by any fixed-to-fixed or fixed-to-variable code that uses that symbol length. This is illustrated by the following example. Consider the test set *T* in Table 1 which consists of four test vectors each partitioned into 4-bit blocks. The probability of occurrence of each unique 4-bit symbol can be calculated. Note that the term "probability" here refers to the actual frequency of the symbol with respect to the total number of symbols in the data rather than the classical definition of probability as the chance of occurrence. The entropy of this test set is calculated from the probabilities of occurrence of unique symbols using the formula $H = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i$, where p_i is the probability of occurrence of symbol x_i in the test set and n is the total number of unique symbols. This entropy is calculated to be 2.64. The entropy gives the minimum average number of bits required for each codeword. Thus, the maximum compression that can be achieved is given by *(symbol_length – entropy) / (symbol_length)* which in this case is equal to (4 - 2.64) / 4 = 34%. Now, if the test set is partitioned into 6-bit blocks instead of 4-bit blocks, then the corresponding probabilities of occurrence of unique symbols will change. In this case, the entropy calculated using the above formula is 3.25. Hence, the entropy for 6-bit blocks is different than 4-bit blocks. The maximum compression in this case is equal to (6 – 3.25) / 6 = 45.8%.

Table 1. Test set divided into 4-bit blocks

test 1	1111	0001	0011	0000	1100	0111
test 2	0000	0011	0001	0111	0100	0000
test 3	0001	0111	0111	1100	1100	0000
test 4	0011	0000	0000	0100	1100	0111

For variable-to-fixed and variable-to-variable codes, the test data is partitioned into variable length symbols (i.e., the symbols can have different numbers of bits). An example of this is a run-length code where each symbol consists of a different size run-length. Given the set of variable length symbols, the entropy can be calculated the same way as for fixed length symbols. If the test set shown in Table 1 is partitioned into symbols corresponding to different size runs of 0's, The entropy for this partitioning is 2.07. Calculating the maximum compression for variable length symbols is different than for fixed length symbols. For variable length symbols, the maximum compression is equal to *(avg_symbol_length – entropy) / (avg_symbol_length)*. The average symbol length is computed as $\sum_{i=1}^{n} p_i \cdot |x_i|$ where p_i is the probability of occurrence of symbol x_i, $|x_i|$ is the length of symbol x_i, and n is the total number of unique symbols. For this example, the average symbol length was calculated as 2.49, so the maximum compression is (2.53 – 2.07)/2.53 = 18.2%. This is the maximum compression that any code that encodes runs of 0's can achieve for the test data in Table 1.

2.2 Specifying the Don't Cares

While computing entropy for fully specified data is well understood, the fact that test data generally contains don't cares makes the problem of determining theoretical limits on test data compression more complex. Obviously, the entropy will depend on how the don't cares are filled with 1's and 0's since that affects the frequency distribution of the various symbols. To determine the maximum amount of compression that can be achieved, the don't cares should be filled in a way that minimizes the entropy.

While the number of different ways the don't cares can be filled is obviously exponential, for fixed length symbols, a greedy algorithm is presented here for filling the don't cares in an optimum way to minimize entropy. A proof is given that it minimizes entropy across all possible ways of filling the don't cares. However, for variable length symbols, the problem is more difficult and remains an open problem. In [Wurtenberger 03], this problem is discussed for run length codes. An optimization method based on simulated annealing is then used to find the solution. Since a heuristic based approach is used, it is not guaranteed to obtain the optimum solution.

2.2.1 Greedy Fill Algorithm

For fixed length symbols, we describe an algorithm called "greedy fill" below and claim that it is optimal, i.e., it results in minimum entropy among all possible ways of filling the don't cares given a specific partitioning of the test set into fixed length symbols.

The greedy fill algorithm is based on the idea that the don't cares should be specified in a way such that the frequency distribution of the patterns becomes as skewed as possible. This algorithm was first described in [Jas 03] where the goal was to maximize the compression for Huffman coding, but it also applies for minimizing entropy. If the fixed symbol length is n, then the algorithm identifies the minterm, a fully specified symbol of length n (there are 2^n such minterms), that is contained in as many of n-bit blocks of the unspecified test data as possible, i.e., having the highest frequency of occurrence. This minterm is then selected, and all the unspecified n-bit blocks that contain this minterm are specified to match the minterm. The algorithm then proceeds in the same manner by finding the next most frequently occurring minterm. This continues until all the don't cares have been specified.

Theorem 1: The probability distribution obtained from the greedy fill algorithm described above results in minimum entropy.

Proof: The proof of this theorem can be found in [Balakrishnan 04].

To illustrate this algorithm, an example test set consisting of four test vectors with 12 bits each is shown in Fig. 1(a). The test vectors are partitioned into 4-bit blocks resulting in a total of twelve 4-bit blocks. Of the 16 (2^4 = 16) possible minterms for a block, the most frequently occurring minterm is 1111 as seven of the unspecified blocks contain 1111. After the first iteration, only five blocks remain and the most frequently occurring minterm now is 1000. All the five blocks contain this minterm and hence the algorithm terminates. The resulting test set after filling up the unspecified bits in the above manner is shown in Fig. 1(b).

```
X0X0 10XX 1XXX      1000 1000 1111
XX1X X1X1 111X      1111 1111 1111
XX00 1XX0 X111      1000 1000 1111
XXXX X1XX 1000      1111 1111 1000
```

(a) Unspecified test set (b) After greedy fill algorithm

Figure 1. Specifying don't cares to minimize entropy

2.2.2 Experimental Results for Greedy Fill Algorithm

Experiments were performed using the greedy fill algorithm to calculate the theoretical limit on test data compression for the dynamically compacted test cubes generated by MINTEST [Hamzaoglu 98] for the largest ISCAS'89 benchmark circuits. These are the same test sets used for experiments in [Chandra 01ab], [Gonciari 03], and [Jas 03]. The compression values in Table 2 are calculated from the exact values of minimum entropy that were generated using the greedy fill algorithm. As can be seen from the table, the percentage compression that can be achieved increases with the symbol length. No fixed-to-fixed or fixed-to-variable code using these particular symbol lengths can achieve greater compression than the bounds shown in Table 2 for these particular test sets. Note, however, that these entropy bounds would be different for a different test set for these circuits, e.g., if the method in [Kajihara 02] was used to change the location or number of don't cares. However, given any test set, the proposed method can be used to determine the corresponding entropy bound for it.

The greedy fill algorithm is exponential in the symbol length because the number of minterms grows exponentially. Thus, it is not practical to calculate the entropy for larger symbol lengths using this algorithm. An approximate algorithm that is not guaranteed to find the minimum entropy but can scale to larger symbol lengths is described in [Balakrishnan 04]. Experimental results show that the limits obtained by this algorithm are very close to the exact entropy limits while greatly reducing the computation complexity.

Table 2. Exact entropy compression limits using greedy fill algorithm

Circuit	Original Test Data	Symbol Length			
		4	6	8	10
s5378	23754	52.0 %	56.3 %	59.2 %	63.8 %
s9234	39723	54.1 %	57.4 %	60.7 %	63.7 %
s13207	165200	83.6 %	85.0 %	85.6 %	87.1 %
s15850	76986	68.9 %	70.5 %	71.9 %	73.5 %
s38417	164736	59.8 %	63.2 %	65.3 %	67.7 %
s38584	199104	65.9 %	67.6 %	68.8 %	70.8 %

3. Symbol Length versus Compression Limits

In this section, the relationship between symbol length, compression limits, and decoding complexity is investigated. The compression limit was calculated from minimum entropy for the benchmark circuit *s9234* as the symbol length is varied. The percentage compression varies from 50% for a symbol length of 2 to 92% for a symbol length of 32; this corresponds to a compression ratio range of 2 to 12. The reason why greater compression can be achieved for larger symbol lengths is that more information is being stored in the decoder. This becomes very clear when the compression limit is graphed for all possible symbol lengths as shown in Fig. 2. As can be seen, the compression ratio goes towards infinity as the symbol length becomes equal to the number of bits in the entire test set. When the symbol length is equal to the number of bits in the entire test set, then the decoder is encoding the entire test set. This is equivalent to built-in self-test (BIST) where no data is stored on the tester.

Consider the simple case where the information in the decoder is simply stored in a ROM without any encoding. The decoder ROM translates the codewords into the original symbols, so at a minimum, it needs to store the set of original symbols. For a symbol length of 2, there are 2^2 possible symbols, so the decoder ROM would have to store 4 symbols each requiring 2 bits, thus it requires 8 bits of storage. For a symbol length of 4, there are 2^4 possible symbols, so the decoder ROM would have to store 16 symbols each requiring 4 bits, thus it requires 64 bits of storage. Having a larger decoder ROM allows greater compression. For a symbol length of 32, there are 2^{32} possible symbols, but it is not likely that the test data would contain all of them. The decoder would only need to store the symbols that actually occur in the test data. In

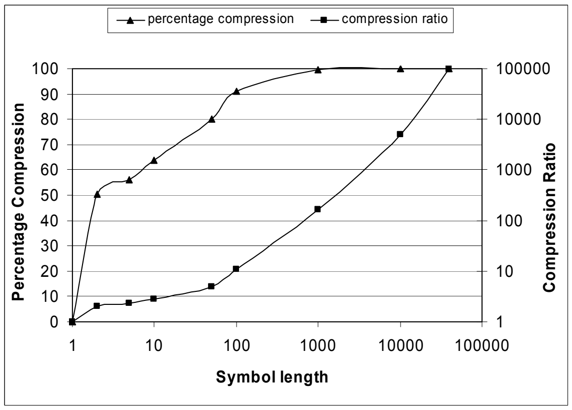

Figure 2. Compression for all symbols lengths for *s9234*

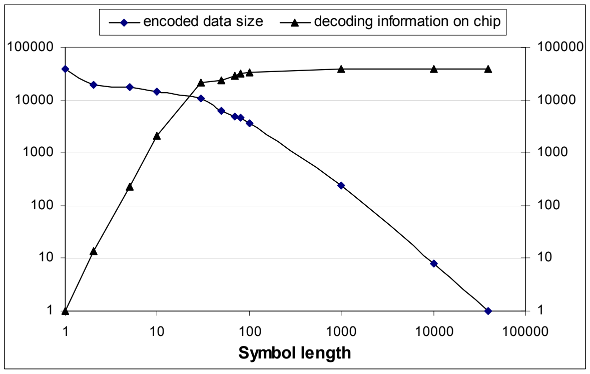

Figure 3. Encoded data size versus simple decoder ROM size for *s9234*

Fig. 3, the size of a simple decoder ROM that is required for different symbols lengths is graphed for different symbol lengths. As can be seen, the decoder information goes up exponentially as the symbol length goes from 1 to 10 as in this range all possible symbols are occurring in the test data and the number of symbols is growing exponentially. After this, the decoder information goes up less than exponentially because not all possible symbols of that length occur in the test data. After the symbol length exceeds about 20, the decoder information is nearly equal to the entire test set as there is very little repetition of the symbols in the test data (i.e., almost all the symbols in the test data are unique at that point). One way to view the graph in Fig. 3 is that as the symbol length is increased, essentially more information is being stored in the on-chip decoder, and less information is being stored off-chip in the tester memory. A symbol length of 1 corresponds to conventional external testing with no compression, and a symbol length equal to the size of the test set corresponds to conventional BIST where no data is stored on the tester.

The real challenge in test data compression is in the design of the decoder. The entropy of the test data places a fundamental limit on the amount of compression that can be achieved for a particular symbol length, but the real key is to achieve something close to the maximum compression using a small decoder. The next two sections evaluate existing test data compression schemes with respect to their compression limits based on entropy theory.

4. Analysis of Test Data Compression Schemes

In this section, different test data compression schemes proposed in the literature are compared with their entropy bounds. Both schemes that operate on fixed length symbols as well as those that work on variable length symbols are considered. One major constraint is that we are only able to report results for schemes for which we have access to the exact test set that was encoded. For this reason, we have limited the results to only those schemes that reported results for the MINTEST test cubes [Hamzaoglu 98]. Discussion of LFSR reseeding schemes is deferred to Sec. 5.

4.1 Fixed Symbol Length Schemes

Table 3 shows results for two compression schemes. One is a fixed-to-variable scheme based on selective Huffman coding [Jas 99, 03], and the other is a fixed-to-fixed scheme that uses a selective dictionary with fixed length indices [Li 03]. The first two columns show the circuit name and the size of the original test set. Then results are shown for [Jas 03] comparing the actual percentage of compression achieved with the entropy limit for a symbol length of 8 and 12. The same is done for the method in [Li 03] for a block size of 16 and 32. The method in [Jas 03] is closer to the entropy limit because it uses variable length codewords whereas the method in [Li 03] uses fixed length codewords. However, the method in [Li 03] has the advantage of an easier interface to the tester since it is a fixed-to-fixed code.

Table 3. Fixed symbol length schemes versus entropy limits

Circuit	Test Set Size	Selective Huffman [Jas 03]				Dictionary Based Compression [Li 03]			
		8		12		16		32	
		Entropy Limit	Actual Comp.	Entropy Limit	Actual Comp.	Entropy Limit	Actual Comp.	Entropy Limit	Actual Comp.
s5378	23754	59.2	50.1	66.2	55.1	71.6	45.3	96.0	67.2
s9234	39723	60.7	50.3	65.0	54.2	68.7	46.1	91.8	65.9
s13207	165200	85.6	69.0	88.0	77.0	88.8	49.2	98.3	73.4
s15850	76986	71.9	60.0	75.1	66.0	77.8	47.4	94.0	67.6
s38417	164736	65.3	55.1	68.5	59.0	71.7	44.0	89.2	49.0
s38584	199104	68.8	58.3	72.2	64.1	75.1	45.8	93.7	64.3

4.2 Variable Symbol Length Schemes

Table 4 shows results for three compression schemes that are based on variable-to-variable codes. Each of these schemes are based on encoding runs of 0's. As was discussed in Sec. 2, filling the don't cares to minimize entropy for variable size symbols is an open problem. These three schemes all fill the don't cares by simply replacing them with specified 0's. For simplicity, the entropy limits in Table 4 were calculated by also filling the don't cares with specified 0's.

The results in Table 4 indicate that the VIHC method in [Gonciari 03] gets very close to the entropy limit. This is because this method is based on Huffman coding. One conclusion that can be drawn from these results is that there is not much room for improvement for research in variable-to-variable codes that encode runs of 0's. The only way to get better compression than the entropy limit that is shown here would be to use a different automatic test pattern generation (ATPG) procedure or fill the don't cares in a different way that changes the entropy limit.

Table 4. Variable symbol length schemes versus entropy limits

Circuit	Test Data Size	Entropy Limit	Golomb [Chandra 01a]	FDR [Chandra 01b]	VIHC [Gonciari 03]
s5378	23754	52.4 %	40.7 %	48.0 %	51.8 %
s9234	39723	47.8 %	43.3 %	43.6 %	47.2 %
s13207	165200	83.7 %	74.8 %	81.3 %	83.5 %
s15850	76986	68.2 %	47.1 %	66.2 %	67.9 %
s38417	164736	54.5 %	44.1 %	43.3 %	53.4 %
s38584	199104	62.5 %	47.7 %	60.9 %	62.3 %

5. Analysis of LFSR Reseeding Schemes

Conventional LFSR reseeding [Könemann 91] is a special type of fixed-to-fixed code in which the set of symbols is the output space of the LFSR and the set of codewords is the seeds. As was seen in the graphs in Sec. 3, larger symbol lengths are needed in order to achieve greater amounts of compression due to the entropy limits on compression. The difficultly with larger symbol lengths is that the decoder needs to be able to produce more symbols of greater length. The power of LFSR reseeding is that an LFSR is used for producing the symbols. An *r*-stage LFSR has a maximal output space as it produces 2^r-1 different symbols as well as having a very compact structure resulting in low area. Thus, an LFSR is an excellent vehicle for facilitating larger symbols lengths with a small area decoder. The only issue is whether the set of symbols that occur in the test data, S_{data}, are a subset of the symbols produced by the LFSR, S_{LFSR}, (i.e., $S_{data} \subseteq S_{LFSR}$). Fortunately, the symbols produced by the LFSR have a pseudo-random property. If *n* is the symbol length, then as was shown in [Könemann 91], if the number of specified bits in an *n*-bit block of test data is 20 less than the size of the LFSR, then the probability of the *n*-bit block of test data not matching one of the symbols in S_{LFSR} is negligibly small (less than 10^{-6}). In [Hellebrand 95], a multiple-polynomial technique was presented which reduces the size of the seed information to just 1 bit more than the maximum number of specified bits in an *n*-bit block. Thus, the compression that is achieved with this approach for a symbol length of *n* is equal to the ratio of *n* to the maximum number of specified bits in an *n*-bit block plus 1. Fig. 4 shows a graph of the compression for LFSR reseeding for all possible symbol lengths. As can be seen, no compression is achieved for short symbol lengths where the maximum number of specified bits is equal to the symbol length. Compression does not start to occur until the symbol length becomes larger than 20. The compression steadily improves as the symbol length is increased, but in general, it cannot exceed the total percentage of don't cares in the test data. For the test set in Fig. 4, the percentage of don't cares is 94%.

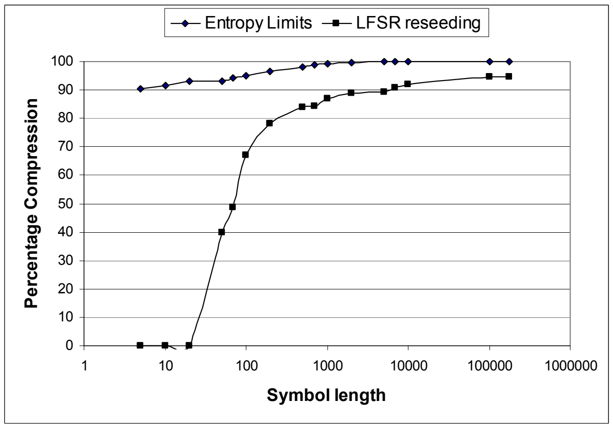

Figure 4. Compression for LFSR reseeding versus symbol length for *s13207*

6. Conclusions

This paper shows how entropy theory can be used to calculate theoretical limits to the amount of test data compression. These limits are useful as a measure of how much improvement is possible over existing test data compression schemes. They can be used to identify coding techniques where there is not much scope for improvement as well as identify coding techniques that hold promise for fruitful research.

This paper studied the relationship of symbol partitioning and don't care filling to entropy. An area for future research is to look at new coding techniques that can exploit this to achieve greater compression.

Acknowledgements

This material is based on work supported in part by the Intel Corporation and in part by the National Science Foundation under Grant No. CCR-0306238.

References

[Balakrishnan 04] Balakrishnan, K. J., "New Approaches and Limits to Test Data Compression for Systems-on-chip", Ph.D. dissertation, University of Texas, Austin, TX, August 2004.

[Bayraktaroglu 01] Bayraktaroglu, I., and Orailoglu, A., "Test Volume and Application Time Reduction through Scan Chain Concealment", *Proc. Design Automation Conference*, pp. 151-155, 2001.

[Chandra 01a] Chandra A., and Chakrabarty, K., "System-on-a-chip test data compression and decompression architectures based on Golomb Codes", *IEEE Trans. on Computer-Aided Design* Vol. 20, No. 3, pp. 355-368, March 2001.

[Chandra 01b] Chandra, A., and Chakrabarty, K., "Frequency-Directed Run-Length Codes with Application to System-on-a-chip Test Data Compression", *Proc. VLSI Test Symposium*, pp. 42-47, 2001.

[Chandra 02] Chandra, A., Chakrabarty, K., and Medina, R. A., "How Effective are Compression Codes for Reducing Test Data Volume?", *Proc. VLSI Test Symposium*, pp. 91-96, 2002.

[Cover 91] Cover, T. M., and Thomas J. A., *Elements of Information Theory*, John Wiley and Sons, Inc., 2nd Edition.

[Gonciari 02] Gonciari, P. T., Al-Hashimi, B M., Nicolici, N., "Improving compression ratio, area overhead, and test application time for systems-on-a-chip test data compression/ decompression", *Proc. Design Automation and Test in Europe*, pp. 604-611, 2002.

[Gonciari 03] Gonciari, P. T., Al-Hashimi, B M., Nicolici, N., "Variable-Length Input Huffman Coding for System-on-a-chip Test", *IEEE Trans. on Computer-Aided Design*, Vol. 22, No 6, pp. 783-796, 2003.

[Hamzaoglu 98] Hamzaoglu I., Patel J. H., "Test Set Compaction Algorithms for Combinational Circuits", *Proc. of Int. Conference on Computer Aided Design*, pp. 283-289, 1998.

[Hamzaoglu 99] Hamzaoglu, I., and Patel, J. H. , "Reducing Test Application Time for Full Scan Embedded Cores," *Proc. Int. Symp. on Fault-Tolerant Computing*, pp. 260-267, 1999.

[Hellebrand 95] Hellebrand, S., Rajski J., Tarnick S., Venkataraman S., and Courtois B., "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers", *IEEE Transactions on Computers*, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Huffman 52] Huffman, D. A., "A method for the construction of minimum redundancy codes", *Proc. IRE*, Vol. 40, pp. 1098-1101, 1952.

[Ichihara 00] Ichihara, H., Kinoshita, K., Pomeranz, I.,Reddy, S. M., "Test Transformation to Improve Compaction by Statistical Encoding", *Proc. Int. Conf. on VLSI Design*, pp. 294-299, 2000.

[Jas 98] Jas, A., and Touba, N. A. ,"Test Vector Decompression Via Cyclical Scan Chains and Its Application to Testing Core-Based Designs", *Proc. of Int.. Test Conference*, pp. 458-464, 1998.

[Jas 99] Jas, A., Dastidar, J. G., and Touba, N. A., "Scan Vector Compression/Decompression Using Statistical Coding," *Proc. VLSI Test Symposium*, pp. 114-120, 1999.

[Jas 03] Jas, A., Dastidar, J. G., Ng M-E.,Touba, N. A., "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding", *IEEE Trans. on Computer-Aided Design*, Vol. 22, No 6, pp 797-806, 2003.

[Kajihara 02] Kajihara, S., Taniguchi K., Miyase K., I. Pomeranz, and Reddy S. M., "Test Data Compression Using Don't Care Identification and Statistical Encoding", *Proc. of Asian Test Symposium*, pp. 67-72, 2002.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," *Proc. European Test Conference*, pp. 237-242, 1991.

[Li 03] Li, L., and Chakrabarty, K., "Test data compression using dictionaries with fixed-length indices", *Proc. VLSI Test Symposium*, pp. 219-224, 2003.

[Reddy 02] Reddy, S. M., Miyase K., Kajihara S., Pomeranz, I., "On test data volume reduction for multiple scan chain designs", *Proc. VLSI Test Symposium*, pp. 103-108,2002.

[Wolff 02] Wolff, FG, Papachristou, C., "Multiscan Based Test Compression and Hardware Decomposition Using LZ77," *Proc. International Test Conference*, pp. 331-338, 2002

[Wurtenberger 03] Wurtenberger, A., Tautermann, C., "Hellebrand. S., A Hybrid Coding Strategy for Optimized Data Compression," *Proc. International Test Conference*, pp. 451-459, 2003.