Scan Vector Compression/Decompression Using Statistical Coding

Abhijit Jas, Jayabrata Ghosh-Dastidar, and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084
E-mail: {jas, dghosh, touba}@cat.ece.utexas.edu

Abstract

A compression/decompression scheme based on statistical coding is presented for reducing the amount of test data that must be stored on a tester and transferred to each core in a core-based design. The test vectors provided by the core vendor are stored in compressed form in the tester memory and transferred to the chip where they are decompressed and applied to the core. Given the set of test vectors for a core, a statistical code is carefully selected so that it satisfies certain properties. These properties guarantee that it can be decoded by a simple pipelined decoder (placed at the serial input of the core's scan chain) which requires very small area. Results indicate that the proposed scheme can use a simple decoder to provide test data compression near that of an optimal Huffman code. The compression results in a two-fold advantage since both test storage and test time are reduced.

1. Introduction

One of the increasingly difficult challenges in testing systems-on-a-chip is dealing with the large amount of test data that must be transferred between the tester and the chip [Chandramouli 96], [Zorian 97]. Systems-on-a-chip are commonly constructed from pre-designed functional blocks called *cores*. Each core has a specified set of test vectors that must be applied to it. The test vectors must be stored on the tester and then transferred to the inputs of the core during testing. As more and more cores (each with its own test set) are placed on a single chip, the amount of test data is growing rapidly. This poses a serious problem because of the cost and limitations of ATE (automated test equipment). Testers have limited speed, channel capacity, and memory. The amount of time required to test a chip depends on how much test data needs to be transferred to the cores and how fast the data can be transferred (i.e., the test data bandwidth to each core). This depends on the speed and channel capacity of the tester and the organization of the scan chains on the chip. Both test time and test storage are major concerns for systems-on-a-chip.

One solution to this problem is to use built-in self-test (BIST) where on-chip hardware is used to test the cores. However for pre-designed logic cores, this is only practical if the core was originally designed to achieve high fault coverage with BIST. Many legacy designs cannot be efficiently tested with BIST without significant re-design effort. Currently there are few commercially available cores that include BIST features. Usually only a set of test vectors for the core is available. The amount of BIST hardware required to apply a large set of deterministic test vectors is generally prohibitive.

This paper presents a statistical compression/ decompression scheme to reduce the amount of test data that must be stored on the tester and transferred to a core. The idea is to store the test vectors for a core in the tester memory in compressed form, and then transfer the compressed vectors to the chip where a small amount of on-chip circuitry is used to decompress the test vectors. Instead of having to transfer each entire test vector from the tester to the core, a smaller amount of compressed data is transferred instead. The approach presented here significantly reduces both test storage requirements and the overall test time.

Transferring compressed test vectors takes less time than transferring the full vectors at a given bandwidth. However, in order to guarantee a reduction in the overall test time, the decompression process should not add additional delay (which would subtract from the time saved in transferring the test data). Moreover, the on-chip decompression circuitry must be small so that it doesn't add significant area overhead. Given a set of test vectors, a method is presented here for choosing a statistical code (similar to Huffman coding) which can be decoded with a simple pipelined decoder. The properties of the code are chosen such that the pipelined decoder has a very small area and is guaranteed to be able to decode the test data as fast as the tester can transfer it. Thus, if the amount of test data is compressed by a factor of c, then that means *both* the tester memory requirements and the total test time are reduced by a factor of c. The test data is compressed by exploiting inherent correlations in the test vectors due to the structural relationship among the faults.

The compression/decompression scheme presented in this paper can be used for generating any set of deterministic scan vectors. It preserves the sequence of the vectors and requires no modifications to the circuit-under-test. It does not require any knowledge of the internal design of the circuit-under-test, and thus is suitable for testing intellectual property cores where the core supplier does not provide any information about the internal structure of the core.

2. Related Work

Novel approaches for compressing test data using the Burrows-Wheeler transform and run-length coding were presented by Yamaguchi, *et al.*, [Yamaguchi 97], [Ishida 98]. These schemes were developed for reducing the time to transfer test data from a workstation across a network to a tester (not for use on chips). The decompression process is implemented with software. It is much too complex and slow for an on-chip hardware implementation as described here.

Iyengar, *et. al* [Iyengar 98] presented the idea of statistically encoding test data. They described a BIST scheme for non-scan circuits based on statistical coding using comma codes (very similar to Huffman codes) and run-length coding. It takes advantage of the fact that sequential test sets often contain many repeated patterns which can be encoded efficiently with statistical encoding. Results indicate that the approach works well for circuits with a small number of primary inputs.

A scheme for compression/decompression of test data using cyclical scan chains is described in [Jas 98]. It uses careful ordering of the test set and formation of cyclical scan chains to achieve compression with run-length codes. A drawback of this approach is the need to configure the cyclical scan chains during testing.

This paper presents a statistical coding scheme for testing cores with internal scan. One of the novel features of this approach is that the code that is used for a particular core is carefully chosen such that only a small decoder circuit is required. There are no restrictions on the order of the test set, and no modifications need to be made to the core-under-test. The small decoder circuit is simply placed at the serial input of the core's scan chain. As will be shown, the decoder provides a significant reduction in the amount of test data that must be transported from the tester to the core.

3. Statistical Coding

The compression/decompression scheme described in this paper is based on statistical coding. In statistical coding, variable length codewords are used to represent fixed-length blocks of bits in a data set. For example, if a data set is divided into 4-bit blocks, then there are 2^4 or 16 unique 4-bit blocks. Each of the 16 possible 4-bit blocks can be represented by a binary *codeword*. The size of each codeword is variable (it need not be 4 bits). The idea is to make the codewords that occur most frequently have a smaller number of bits, and those that occur least frequently to have a larger number of bits. This minimizes the average length of a codeword. The goal is to obtain a coded representation of the original data set that has the smallest number of bits.

A Huffman code [Huffman 52] is an optimal statistical code that is proven to provide the shortest average codeword length among all uniquely decodable variable length codes. A Huffman code is obtained by constructing a Huffman tree. The path from the root to each leaf gives the codeword for the binary string corresponding to the leaf. An example of constructing a Huffman code can be seen in Table 1 and Figs. 1 and 2. An example of a test set divided into 4-bit blocks is shown in Fig. 1. Table 1 shows the frequency of occurrence of each of the possible blocks (referred to as symbols). There are a total of 60 4-bit blocks in the example in Fig. 1. Figure 2 shows the Huffman tree for this frequency distribution and the corresponding codewords are shown in Table 1.

```
0010 0100 0010 0110 0000 0010 1011 0100 0010 0100 0110 0010
0010 0100 0010 0110 0000 0110 0010 0100 0110 0010 0010 0000
0010 0110 0010 0010 0010 0100 0100 0110 0010 0010 1000 0101
0001 0100 0010 0111 0010 0010 0111 0111 0100 0100 1000 0101
1100 0100 0100 0111 0010 0010 0111 1101 0010 0100 1111 0011
```

Figure 1. Example of Test Set Divided into 4-Bit Blocks

Table 1. Statistical Coding Based on Symbol Frequencies for Test Set in Fig. 1

Sym.	Freq	Pat.	Huff. Code	Sel. Code
S_0	22	0010	10	**1**0
S_1	13	0100	00	**1**10
S_2	7	0110	110	**1**11
S_3	5	0111	010	**0**0111
S_4	3	0000	0110	**0**0000
S_5	2	1000	0111	**0**1000
S_6	2	0101	11100	**0**0101
S_7	1	1011	111010	**0**1011
S_8	1	1100	111011	**0**1100
S_9	1	0001	111100	**0**0001
S_{10}	1	1101	111101	**0**1101
S_{11}	1	1111	111110	**0**1111
S_{12}	1	0011	111111	**0**0011
S_{13}	0	1110	-	**-**
S_{14}	0	1010	-	-
S_{15}	0	1001	-	-

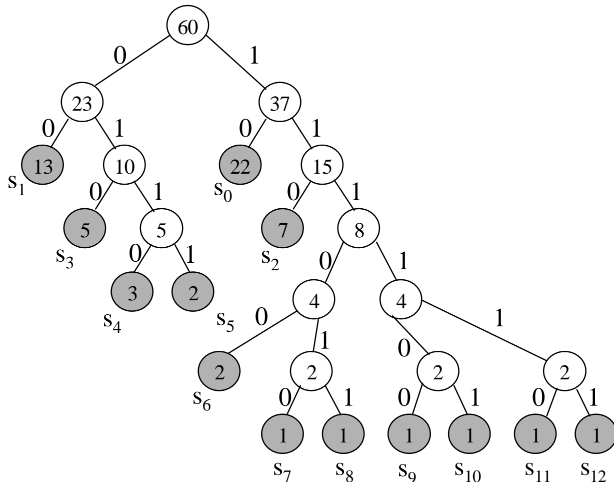

Figure 2. Huffman Tree for the Code Shown in Table 1

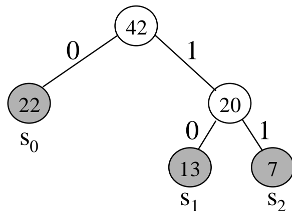

Figure 3. Huffman Tree for the 3 Highest Frequency Symbols in Table 1

An important property of Huffman codes is that they are prefix-free. No codeword is a prefix of another codeword. This greatly simplifies the decoding process. The decoder can instantaneously recognize the end of a codeword uniquely without any lookahead.

The amount of compression that can be achieved with statistical coding depends on how skewed the frequency of occurrence is for the different codewords. If all of the codewords occur with equal frequency, then no compression can be achieved. It is well known, however, that the test vectors in a test set tend to have a lot of correlations. This arises from the fact that faults in the CUT that are structurally related require similar input value assignments in order to be provoked and sensitized to an output. This often results in skewed frequency of occurrence for different codewords. Moreover, for *test cubes* (which are test vectors that are not fully specified, i.e., contain X's), the compression can be very large. The X's provide flexibility to allow a block to be encoded with more than one possible codeword. The shortest possible codeword can be chosen for each block to maximize the compression.

To fully exploit the correlations in a test set, the number of bits in each scan vector should be a multiple of the fixed-length block size used for the statistical code. When dividing the test set into *b*-bit blocks for coding, if the size of the scan vectors is not a multiple of *b*, then X's can be added to pad the start of the vectors (first bits shifted into the scan chain) to make the length a multiple of *b*. Shifting some extra bits (at the start of the vector) into the scan chain doesn't matter provided the final contents of the scan chain contains the correct test vector when it is applied to the core-under-test. Having each scan vector be a multiple of the block size aligns the blocks within the vectors so that the correlations between the bits will skew the frequencies.

4. Overview Of Proposed Scheme

The compression/decompression scheme proposed here involves statistically coding the scan vectors for a core and then placing an on-chip decoder at the serial input of the core's scan chain to decompress the vectors. A block diagram illustrating the scheme is shown in Fig. 4. The tester channel shifts a constant stream of variable length codewords (corresponding to compressed scan data) to the decoder. The decoder generates the corresponding fixed-length blocks. The number of bits in the fixed-length block will often be greater than the number of bits in the codeword, so the rate at which data is coming out of the decoder is higher than the rate at which the data is coming into the decoder. This means that the decompressed scan vectors must be shifted into the core's scan chain faster than the compressed scan vectors are shifted into the decoder. There are two ways to achieve this:

1. <u>Use scan chain with faster clock than tester clock</u>
This is illustrated in Fig. 4. If the system clock rate is faster than the tester clock rate, then it may be possible to clock the scan chain at a faster clock rate than the tester's clock rate (as described in [Heidel 98]). A serializer is placed between the decoder and the core's scan chain. The serializer is loaded in parallel by the decoder (allowing the decoder to generate multiple bits of data in a slower tester clock cycle) and serially shifted out into the core's scan chain at a faster clock rate. One advantage of this approach is that it can be used to provide at-speed scan with a slow tester [Heidel 98].

2. <u>Use single tester channel to feed multiple scan chains</u>
This is illustrated in Fig. 5. If it is not possible to clock the scan chain with a faster clock than the tester clock, then another approach is to have the tester channel rotate between *n* scan chains (each scan chain has its own decoder). Each clock cycle, the tester shifts in a bit for a different decoder for each of the *n* scan chains. Each of the *n* decoders simply samples its input once every *n* clock cycles in a different phase from the other decoders. For example, if there are two scan chains ($n = 2$), then

the decoder for scan chain 1 would sample its input on even tester clock cycles and the decoder for scan chain 2 would sample its input on odd tester clock cycles. With this approach, the "effective clock rate" for each of the decoders is divided by *n*. However, the scan chain corresponding to each decoder is still clocked at the normal tester clock rate and thus its clock rate is *n* times faster than the decoder. Each time the decoder is clocked once, the scan chain is clocked *n* times.

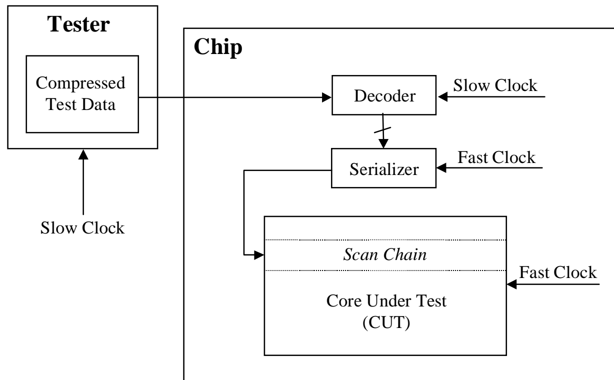

Figure 4. Block Diagram Illustrating Scheme for a Slower Tester Clock

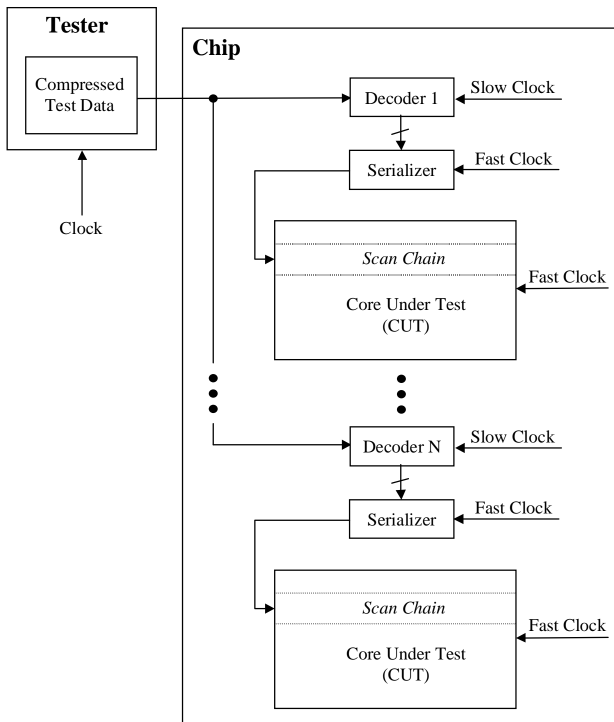

Figure 5. Block Diagram Illustrating Scheme Using Single Tester Channel to Feed Multiple Cores

In the remainder of this paper, without loss of generality, it will be assumed that the scan clock is faster than the tester clock (i.e., corresponding to scenario #1 above). However, all of the concepts apply equally as well for scenario #2 where the tester channel feeds multiple scan chains such that the "effective clock rate" seen by each decoder is slower than the clock rate of the scan chain.

To illustrate how the decoder and serializer work, consider the following example. Suppose the scan vectors are divided into 4-bit blocks, and each 4-bit block is replaced by a variable length codeword. The compressed test data stored on the tester consists of the variable length codewords. These codewords are shifted into the decoder as a continuous stream of bits. If the codewords are prefix-free, than the decoder can easily recognize when it has received a complete codeword. When the decoder has received a complete codeword, it loads the corresponding 4-bit block in parallel into the serializer. The contents of the serializer are then shifted into the core's scan chain. If the core's scan chain is clocked at twice the clock rate that the tester operates at, then after two tester clock periods the entire contents of the serializer will have been shifted into the core's scan chain. During the two tester clock periods that the serializer is being shifted into the scan chain, the decoder can be receiving the next codeword.

The key to making the scheme work is careful selection of the statistical code that is used for compressing the test set. There are two important issues that must be considered in selecting the code: one is that the decoder must be small in order to keep the area overhead down, and the other is that the decoder must not output the decompressed bits into the serializer faster than they can be shifted out into the core's scan chain. While a Huffman code gives the optimal compression for a test set divided into a particular fixed-length block size, it generally requires a very large decoder. A Huffman code for a fixed-length block size of b bits requires a finite state machine (FSM) decoder with 2^b-1 states. Thus the size of the decoder for a Huffman code grows exponentially as the block size is increased. A method for selecting an efficient statistical code for the proposed scheme is described in the following section.

5. Statistical Code Selection For Scheme

Given the test set for a core, a statistical code for compressing the test set must be selected. There is a tradeoff in selecting the code between the amount of compression that is achieved and the complexity of the decoder. Moreover, if the clock rate of the tester is *Clock_Rate(tester)* and the clock rate of the core's scan

chain is *Clock_Rate(scan)* then the ratio of the clock rates, *Clock_Rate(tester) / Clock_Rate(scan)*, limits the minimum size of a codeword. If the test set is divided into fixed-length blocks of *b* bits, then the serializer will hold *b* bits, and thus it takes *b* scan clock cycles to shift the serializer's contents into the core's scan chain. During the time that the contents of the serializer are being shifted into the core's scan chain, the tester is shifting bits into the decoder. When the decoder receives a complete codeword, then it needs to output the corresponding block of *b* bits into the serializer. If the codeword is too short, then the serializer may not have been emptied yet which would cause a problem for the decoder. So, in order to ensure that the serializer is always empty when the decoder finishes decoding a codeword, the minimum size of a codeword, *Min_Size(codeword)*, must be no smaller than the ratio of the tester and scan clock rates times the size of each block:

$$Min\ Size(codeword) \geq \frac{Clock_Rate(tester)}{Clock_Rate(scan)}\ x\ b$$

For example, if the block size is 8 and the scan clock rate is twice the tester clock rate, then the minimum size of a codeword is 4. (Remember that one way to make the scan clock rate be twice as fast as the "effective clock rate" as seen by the decoder is to simply have the tester channel feed two scan chains so that the rate that the decoder receives data from the tester is half as fast as the rate at which data can be shifted into the scan chain.)

Note that there is no constraint on the maximum size of a codeword. When the serializer finishes shifting the data into the scan chain, it will wait (hold the scan clock) until the decoder is ready to load the next block into the serializer.

Using a Huffman code would provide the maximum compression, however, it would require a complex decoder and may not satisfy the constraint on the minimum size of a codeword. Therefore, some alternative statistical code must be selected. The approach taken here involves using a *selective* coding approach for which a very simple decoder can be constructed. Consider the case where the test set is divided into fixed-length blocks of *b* bits, then there will be 2^b codewords. The first bit of each codeword will be used to indicate whether the following bits are coded or not. If the first bit of the codeword is a '0', then the next *b* bits are not coded and can simply be passed through the decoder as is (hence the complete codeword has $b+1$ bits). If the first bit of the codeword is a '1', then the next variable number of bits form a prefix-free code that will be translated by the decoder into a *b*-bit block. The idea is to only code the most frequently occurring *b*-bit blocks using codewords with small numbers of bits (less than *b*, but greater than or equal to *Min_Size(codeword)*). Compression is achieved by having the most common *b*-bit blocks be represented by codewords with less than *b*-bits. The decoder is simple because only a small number of blocks are coded. The vast majority of the blocks are not coded and can be simply passed through the decoder. If *n* blocks are coded, then the decoder can be implemented with an FSM with no more than $n + b$ states (compared with a Huffman code which requires 2^b-1 states).

An example to illustrate the proposed approach for selecting a statistical code is shown in Fig. 3. Consider the test set in Fig. 1. If we divide the entire test set into 4-bit blocks then we get the frequency distribution as shown in the second column of Table 1. As can be seen from Table 1, the patterns having the highest frequencies are **0010**, **0100** and **0110**. So these are the patterns that are coded while the rest of them will be left unchanged. We construct a Huffman tree for the three patterns to get their codewords (as shown in Fig. 3). The codewords for the remaining 13 symbols is simply a 0 followed by the symbol itself (as shown in the last column of Table 1).

The two important parameters in selecting the code are the block size *b* and the number of coded blocks *n*. Once those have been chosen, then the procedure for constructing the code is mechanical. A Huffman tree is formed for the *n* most frequently occurring *b*-bit blocks. The codewords for the most frequently occurring blocks is simply a '1' followed by the Huffman code obtained from the Huffman tree. The codewords for the remaining blocks are simply a '0' followed by the *b*-bit block itself. The amount of area overhead for the decoder can be controlled by placing an upper bound on the values of *n* and *b*. For a particular value of *b*, the amount of compression that will be achieved can be computed in linear time with respect to the size of the test set. Thus, the best value of *b* can be efficiently determined through experimentation. Several values of *b* can be tried for a particular test set to determine which gives the best compression.

To summarize, the statistical code that is selected using the approach described here has the following properties:

1. No codeword is smaller than the *Min_Size(codeword)* constraint based on the ratio of the tester clock rate to the scan clock rate.
2. No codeword is larger than $b+1$ bits.
3. Any codeword that is $b+1$ bits long is always a '0' followed by the *b*-bit block that it corresponds to.
4. A decoder for the code can be implemented by an FSM with $n + b$ states.

6. Implementation

Once the statistical code has been chosen, then a decoder for the code is synthesized. One way to implement the decoder is to use a simple FSM. There are two inputs to the decoder, one is the tester clock and the other is the serial input from the tester channel. For a block size of *b*, the decoder has *b* data outputs and two control outputs. The two control outputs are: *parallel_load* (**Par**) and *serial_load* (**Ser**). These two signals control the buffering and loading of data into the serializer when the data has been decoded. The state transition diagram for the decoder FSM can be easily formed from the tree representation of the code. An example is shown in Fig. 6. If the first bit of the codeword is a '0' indicating that the next *b* bits are not coded, then the decoder simply passes the bits through by serially loading them into an internal buffer and when done loads them in parallel into the serializer. If the first bit of the codeword is a '1' indicating that the subsequent bits form a prefix-free variable length code, then the decoder branches on each bit one at a time until it reaches the end of the codeword at which point it does a parallel load of the appropriate *b*-bit block into the serializer.

Note than each codeword has a bit indicating whether the pattern that follows is encoded or not. In our scheme we use a '0' to indicate "no coding" and a '1' to indicate coding. The state transition diagram of the FSM for the decoder is shown in Fig. 6.

Another way to implement the decoder is to use a ROM or RAM by placing a further restriction on the selection of the statistical code. The additional restriction is that all codewords must be one of two fixed sizes. If the first bit of the codeword is a '0', then the next *b*-bits are the block itself. If the first bit of the codeword is a '1', then the next *a*-bits form an address into a ROM/RAM whose contents contain the corresponding *b*-bit block. For example, if the block size is 8 (b = 8), and the address size is 4 (a = 4), then there are 2^8=64 different 8-bit blocks and 2^4=16 of them can be encoded with (a+1=5) bit codewords while the remaining 48 of them would have (b+1 = 9) bit codewords. The decoding would be done using a 16x8 bit ROM or RAM. If a RAM is used, it could be reused for different cores by simply changing its contents to correspond to the appropriate code for each core. Note also that a RAM that is already present in the functional design could be adapted for this purpose during testing. If the RAM is bigger than what is needed, it can be used by simply padding the high-order address bits with 0's and looking only at the low-order data bits when decoding the statistical codes.

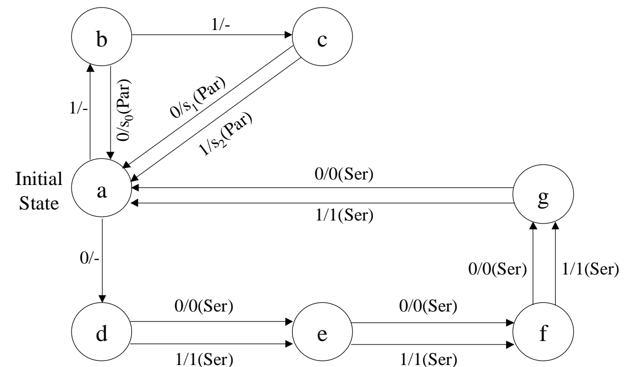

Figure 6. State Transition Diagram of the FSM Decoder for Selected Code

7. Experimental Results

The proposed compression/decompression scheme was used to compress test sets for the largest ISCAS benchmark circuits. An ATPG tool was used to generate test vectors that provided 100% coverage of detectable faults in each circuit. Static compaction of the test set was performed by merging the test cubes when possible and doing reverse 3-valued fault simulation to remove superfluous test cubes that aren't needed to detect faults. Unspecified input assignments were left as X's when selecting the statistical code to enable better compression. When encoding a block containing X's, the value of the X's can be chosen to match the codeword whose representation requires the smallest number of bits. After all of the X's have been filled to minimize the statistically encoded data, then fault simulation can be done to remove any superfluous test vectors. After specifying the X's, some of the vectors may detect more faults than they did before thus rendering other vectors superfluous which can thus be removed. Normally, the X's in test cubes are filled with random values to try to reduce the number of vectors needed to detect all the faults thereby providing some compression. However, as can be seen in Table 2, the proposed statistical coding approach provides a much greater compression than doing a random fill. For each circuit, Table 2 shows the amount of compression that can be achieved with random fill, and then compares it with the results for using statistical coding with 3 different block sizes: 4, 6, and 8. For each block size, the compression achieved by using a normal Huffman coding is shown followed by the compression achieved using the scheme described here. The percentage of compression is computed as:

[(Original Bits - Compressed Bits /(Original Bits)] x 100

Table 2. Compression Achieved for Test Cubes Providing 100% Fault Coverage of Detectable Faults

Circuit Name	R-Fill Comp. (%)	Huffman Code Blk Size	Huffman Code FSM States	Huffman Code Comp. (%)	Selected Code Blk Size	Selected Code FSM States	Selected Code Comp. (%)
c2670	15.9	4	15	59.0	4	5	41.5
		6	63	67.3	6	8	58.3
		8	255	72.1	8	16	61.3
c3540	17.4	4	15	34.8	4	5	26.7
		6	63	42.7	6	12	35.7
		8	255	49.5	8	21	40.6
c5315	11.5	4	15	60.1	4	5	43.9
		6	63	65.4	6	8	57.2
		8	255	71.4	8	10	63.6
c6288	7.3	4	15	45.8	4	5	36.1
		6	63	55.3	6	12	43.2
		8	255	56.0	8	24	44.6
c7552	17.5	4	15	44.9	4	7	31.0
		6	63	61.7	6	8	53.1
		8	255	66.2	8	17	56.4
s5378	0	4	15	61.2	4	5	45.6
		6	63	67.8	6	8	59.5
		8	255	73.1	8	20	62.0
s9234	23.2	4	15	50.8	4	6	40.1
		6	63	76.7	6	5	70.9
		8	255	76.8	8	9	67.1
s13207	6.0	4	15	61.2	4	6	48.8
		6	63	82.0	6	5	80.2
		8	255	86.0	8	9	83.5
s15850	10.6	4	15	53.7	4	5	35.8
		6	63	81.1	6	7	78.9
		8	255	78.8	8	10	70.5
s38417	21.3	4	15	53.9	4	8	32.7
		6	63	60.1	6	13	47.0
		8	255	63.5	8	17	53.6

The number of states required in a FSM decoder for each code is shown. As can be seen, the code selected for the proposed scheme provides slightly less compression than a Huffman code, but it allows the use of a much simpler decoder. This is essential for making the scheme an area efficient approach for reducing test time and test storage requirements.

While the number of states for the Huffman decoder grows exponentially, the number of states for the proposed scheme grows linearly. The block size provides an easy way to tradeoff between area overhead and compression with the proposed scheme. Larger block sizes generally give greater compression, but require a more complex decoder and larger serializer.

8. Conclusion

Statistical coding provides a powerful way to compress test data. It provides a two-fold advantage in both reducing the amount of test data that needs to be stored on the tester and reducing the time for transferring test data from the tester to the circuit-under-test. The drawback of using statistical coding is the on-chip circuitry needed for decompressing the test vectors. The scheme described in this paper addresses this problem by selecting a "simple-to-decode" statistical code for a particular test set. Results indicate that a small FSM decoder can be used to provide compression near that of an optimal Huffman code.

The key ideas presented in this paper can be applied to other types of compression codes as well. One area for further research is to consider variable-to-variable length codes for compressing test data. The complexity of the decoding process could be constrained by careful selection of the code.

Acknowledgements

This material is based on work supported in part by the National Science Foundation under Grant No. MIP-9702236, and in part by the Texas Advanced Research Program under Grant No. 1997-003658-369.

References

[Chandramouli 96] Chandramouli, R., and S. Pateras, "Testing Systems on a Chip," *IEEE Spectrum*, pp. 42-47, Nov. 1996.

[Heidel 98] Heidel, D., S. Dhong, P. Hofstee, M. Immediato, K. Nowka, J. Silberman, K. Stawiasz, "High Speed Serializing/De-Serializing Design-For-Test Method for Evaluating a 1GHz Microprocessor," *Proc. of VLSI Test Symposium*, pp. 234-238, 1998.

[Huffman 52] Huffman, D.A., "A Method for the Construction of Minimum Redundancy Codes," *Proc. of IRE*, Vol. 40, No. 9, pp. 1098-1101, Sep. 1952.

[Ishida 98] Ishida, M., D.S. Ha, T. Yamaguchi, "COMPACT: A Hybrid Method for Compressing Test Data," *Proc. of VLSI Test Symposium*, pp. 62-69, 1998.

[Iyengar 98] Iyengar, V., K. Chakrabarty, and B.T. Murray, "Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets," *Proc. of VLSI Test Symposium*, pp. 418-423, 1998.

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

[Yamaguchi 97] Yamaguchi, T., M. Tilgner, M. Ishida, and D.S. Ha, "An Efficient Method for Compressing Test Data," *Proc. of International Test Conference*, pp. 191-199, 1997.

[Zorian 97] Zorian, Y., "Test Requirements for Embedded Core-based Systems and IEEE P1500," *Proc. of International Test Conference*, pp. 191-199, 1997.