Copyright

by

Najad Borhan Baltaji

2012

The Report committee for Najad Borhan Baltaji
Certifies that this is the approved version of the following report:

Scan Data Compression Using Alternate Huffman Coding

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: ______________________________
Nur A. Touba

____
Vijay K. Garg

Scan Test Data Compression using Alternate Huffman Coding

by

Najad Borhan Baltaji, B.S.E.E.

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2012

To my Parents

Acknowledgements

I am thankful to Dr. Nur A. Touba for his guidance, accessibility, and willingness to aid with answers during the course of this report. I am also grateful for Vijay K. Garg for agreeing to be in the supervising committee.

Scan Data Compression Using Alternate Huffman Coding

by

Najad Borhan Baltaji, M. S. E.

The University of Texas at Austin, 2010

SUPERVISOR: Nur A. Touba

Huffman coding is a good method for statistically compressing test data with high compression rates. Unfortunately, the on-chip decoder to decompress that encoded test data after it is loaded onto the chip may be too complex. With limited die area, the decoder complexity becomes a drawback. This makes Huffman coding not ideal for use in scan data compression. Selectively encoding test data using Huffman coding can provide similarly high compression rates while reducing the complexity of the decoder. A smaller and less complex decoder makes Alternate Huffman Coding a viable option for compressing and decompressing scan test data.

TABLE OF CONTENTS

LIST OF TABLES.. viii

LIST OF FIGURES.. ix

Chapter 1: INTRODUCTION 1

1.1 Background of Test.. 1

1.2 ATE Storage.. 3

1.3 Organization of Report.. 4

Chapter 2: PREVIOUS WORK 5

2.1 Previous Work... 5

Chapter 3: SELECTIVE HUFFMAN CODING 8

3.1 Huffman Coding... 8

3.2 Selective Coding.. 13

3.3 Selective Huffman Coding.. 13

3.4 Implementation.. 14

Chapter 4: RESULTS 17

4.1 Experimental Results.. 17

Chapter 5: CONCLUSIONS 23

REFERENCES... 24

LIST OF TABLES

Table 1 Huffman frequency distribution – step 1 8

Table 2 Huffman frequency distribution – step 2 9

Table 3 Huffman frequency distribution – step 3 10

Table 4 Total encoded blocks percentages for each circuit test data set 15

Table 5 Comparison of compression results for s35932 circuit 17

Table 6 Comparison of compression results for s13207 circuit 17

Table 7 Comparison of compression results for s15850 circuit 18

Table 8 Comparison of compression results for s38417 circuit 18

Table 9 Comparison of compression results for s38584 circuit 19

Table 10 Comparison of compression results for s9234 circuit 20

Table 11 Comparison of compression results for s5378 circuit 20

LIST OF FIGURES

Figure 1 Huffman tree after first node construction ………………………………………… 9

Figure 2 Huffman tree after second node construction ……………………………………. 10

Figure 3 Final Huffman tree for frequency distribution in table 1 …………………….. 11

INTRODUCTION

Chapter 1

1.1 Background of Test

Microprocessors design is a large and expensive undertaking filled with high risk and high reward. One of the main mechanisms for reducing risk and ensuring a successful part is performing extensive verification of the microprocessor design using simulators that verify the functional validity of the design. And microprocessor designers are smart to heavily invest in this area to reduce the risk of producing faulty parts. Now even after the investment is made in fully verifying a design, there remains the risk the chip does not perform as expected. The sources of these failures are manufacturing defects. Microprocessor designers need a way to ensure that the functionality of the design as intended in the spec is not corrupted by manufacturing defects. And this must be done in an efficient manner. This is where scan data testing comes in. Scan testing is the addition of specialty logic in the chip whose sole purpose is to efficiently verify the manufacturing validity of the chip post-manufacture.

Scan testing encompasses and covers a large portion of the chip area. It mainly targets all the logic on the chip that is not memory. The scan architecture enables the detection of several types of manufacturing, not logical, defects in the design. It does this by employing chains of flops that utilize existing flops in the design. Modern processors contain millions of flops most of which need to be part of a scan chain in the design. Each of these chains is accessible through a single input and output port on the chip: one input and one out per scan chain. The idea behind scan testing is to be able to scan in certain chains of data onto each of the scan chains in the design and then employing one or more functional clock cycles that allow the capture of data from flops on the scan chains onto other flops on the scan chains. Then the resultant captured data is shifted out from all the scan chains for external observation and to determine whether any manufacturing defects exist in the design. This process needs to be repeated for every chip produced. The number of chips produced for a design can easily number in the tens of millions. One can then easily see the need to reduce the amount of time that is spend testing each produced chip in order to keep production costs low. One way to do this is to find the smallest set of scan data that gives the highest amount of test coverage for the design.

1.2 ATE Storage

The tools that are used to administer this test data to each produced chip is called automatic test equipment or ATE. By virtue of the limited amount of storage that exists on an ATE, it turns out that the test data set that was produced for a design will likely not all fit into the ATE's memory storage at once. It then becomes time consuming to have to load two or more sets of scan data into an ATE's memory for each chip being tested. This causes an unacceptable waste of time which pushes up production costs. So there then comes the need to be able to load all the test data into the ATE's memory storage at once.

The common and obvious way to solve this problem is to compress the test vector data that was produced for a design enough so that it fits into the ATE's storage. But the implication of compressing the data is that the data then needs to be uncompressed before it can be shifted onto the chip's chains of scan flops. For reasons beyond the discussion of this report, the logic that uncompresses the data exists on the chip. And given the limited area for gates on a chip die, the decoder must be implemented with a limited amount of gates. This means that the solution to this problem doesn't just lie in finding the highest compression algorithm for the test data. The solution lies in finding the best compression that is balanced out against the complexity of the decoder required to uncompress this data. This is the

issue that this report attempts to address. Namely, finding good compression rates that require the less complex on-chip decoders.

1.3 Organization of Report

This report is organized into five main sections. The first section gave a background on the area of silicon manufacturing test. It also introduced the problem of test data storage and the implications of decompressing compressed ATE storage data and defined the problem which this report attempts to tackle. Section 2 discusses previous work involved in test data compression techniques. Section 3 goes into detail about the new method of test data compression introduced in this report along with some implementation details. Section 4 then discusses some experimental results that compare the new compression scheme in this report against previous compression schemes. And finally section 5 concludes this report with comments on the results discussed in the previous section.

PREVIOUS WORK

CHAPTER 2

2.1 Previous Work

There are many techniques for test data compression. [2] provides a survey of these techniques. The various techniques generally fall into three broad categories: code based, linear decompressor, and broadcast scan based techniques. The compression technique presented in this paper falls under the category of code based compression techniques and specifically under the sub-category of statistical coding techniques. Test data that is used for scan is generally unique in that it contains many repeated patterns. These repeated patterns usually come in the form of X or “don’t care bits”. This means that for a given block size of data bits from the overall data set, the frequency distribution of each block of data is highly skewed with certain block values appearing far more frequently that other block values. Statistical coding data compression techniques take advantage of this statistical distribution to improve compression.

The first paper to explore using statistical encoding to for test data is [6]. The paper takes advantage of the highly skewed frequency distribution of test vector

data sets to achieve compression. But this paper mainly discussed a BIST scheme that doesn't directly address scan chain data compression and decompression.

Huffman coding is a form of statistical coding. It was published in a 1952 paper by David A. Huffman: "A Method for the Construction of Minimum-Redundancy Codes". The method employed involves dividing test data vectors into fixed-length blocks of bits of size b. Then forming a frequency distribution of the 2^b different values of the blocks. Then forming a binary tree based on the frequency distribution of the block values. Once the tree is complete, test data vectors can be compressed by encoding each block in the vector set using the constructed Huffman tree. The end result is that block values that are most frequent in the test vector set have the shortest encoded value and the least frequent prevalent block values have the longest encoded value. The result is that the encoded test vector data set contains far fewer data bits than the original unencoded data set.

Yet Huffman compression method is not suitable for compressing data for the purposes of scan since the number of FSM states for the logic required on the chip to decode the data is on the order of 2^b-1 states where b is the chosen size for length of each block of bits. And given the area requirements of chips, Huffman encoding can't be used in its original form.

In [1], an alternate form of statistical coding is presented which can be applied to test vector data compression. The method involves sacrificing some of the compression potential of Huffman coding in order to reduce the complexity of the on-chip decoder logic in order to reduce the amount of gates that would be required to implement the decoder. The result is moderately less compression but with highly reduced complexity for the decoder, which makes this a good option for scan test data compression.

This report addresses a variant of the statistical encoding presented in [1]. It employs an alternate method of encoding which uses a slightly larger decoder but with the benefit of improved compression rates. The next section introduces the basics of Huffman coding, then explain the selective coding introduced in [1], and finally introduces the alternate Huffman coding technique which is the topic of this report.

SELECTIVE HUFFMAN ENCODING

Chapter 3

3.1 Huffman Coding

Before introducing the alternate Huffman encoding technique, we first discuss Huffman coding. Huffman coding is a form of lossless data compression. Huffman coding starts of by dividing the test data set into fixed length block sizes of length b. Once the value of b is chosen, the data is divided into a series of b-length blocks whose values can be one of any 2^b values. The next step is to create a frequency distribution for each of the 2^b possible values. Let us consider the example frequency-sorted distribution in table 1 below:

Frequency	Value
6	1
8	2
11	3
16	4
21	5
46	6

Table 1 : Huffman frequency distribution – step 1

After the frequency-value distribution is determined, the goal becomes to construct a Huffman tree from these frequency-value pairs. In the first step of this example, we take the least two frequent pairs in the distribution and create a node of the tree whose two leaves point to the least two frequent pairs in the current distribution. In this example, we would create a node with value 14 which points to leaf nodes which are the pairs 6-1 and 8-2:

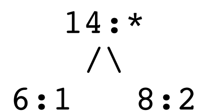

Figure 1. Huffman tree after first node construction

After removing the 6:1 and 8:2 pairs since they are now part of the Huffman tree, the frequency-value pair sorted distribution now looks like this:

Frequency	Value
11	3
14	*
16	4
21	5
46	6

Table 2 : Huffman frequency distribution – step 2

We then repeat the same step of choosing the least two frequent values and creating a node which has the sum of their frequencies and points to two leaf nodes which are the chosen least frequent nodes. In this step, those would be the values 3 with frequency 11 and the node formed in the previous step which has a total frequency of 14:

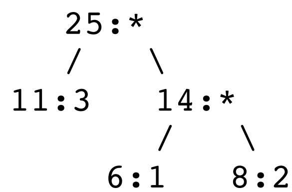

Figure 2. Huffman tree after second node construction

The frequency-value pair distribution now looks like this:

Frequency	Value
16	4
21	5
25	*
46	6

Table 3 : Huffman frequency distribution – step 3

The process is then repeated until we are left with a single frequency-value pair in the table. This last pair has a frequency which is the sum of all blocks in the test data set and will be the root of the constructed Huffman tree which is used to compress b-sized blocks of data. In our example, the final Huffman tree will have 108:* as its root:

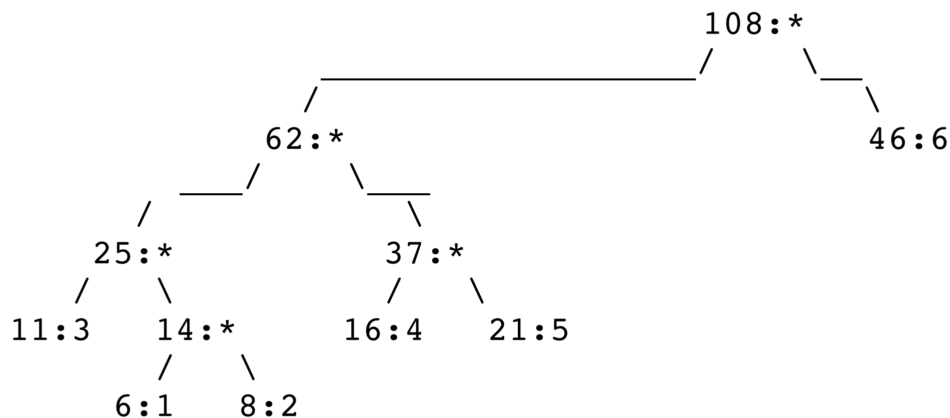

Figure 3. Final Huffman tree for frequency distribution in Table 1

One can see then that resultant tree has all the original frequency:value pairs as leaves of the tree. Also one can notice that the most frequent values appear at low depths in the tree whereas the least frequent values appear at the largest depths in the tree. Its is this difference in depth that is utilized in encoding the most frequent values with the smallest bit-length encoded representations and vice versa for the least frequent values.

The significance of this difference of depth will be apparent when the data encoding method is discussed. For block of size b in the data set, we begin traversing the Huffman tree from its root until we reach the leaf whose value matches that of the block value we are encoding. Each time we traverse a node to its left sub-tree, we output a value of 0 and we output a 1 for traversing the right sub-tree. As mentioned earlier, the difference in tree depths on values in the Huffman tree causes the most frequent values to have shorter encoded length because they exist at a shallow depth in the tree and vice versa for least frequent values in the tree.

In our running example, we encode the block value of 4. Starting at the root, we take a left turn to reach 62:* then right turn to reach 37:* and then finally a left turn to reach the leaf with a value of 4. Based on the turns taken, the encoding for the block value of 4 is 010. However, since the value 6 is the most frequently appearing value in the data set, its encoding is obtained by taking one right turn or simply 1.

Given the skewed frequency distribution seen in test data compression, Huffman coding then can be ideal at compressing this type of data. However the main drawback is the complexity of the decoder required to uncompress the test data. According to [1], the number of states of the decoder FSM is 2^b -1 states for pure Huffman coding.

3.2 Selective Coding

In order to address the issue of the complex decoder that results from Huffman coding, [1] proposes a selective encoding scheme which utilizes Huffman coding but vastly reduces the complexity of the decoder. Instead of encoding the entire space of 2^b values, selective encoding only encodes the n most frequent values in the frequency distribution. The chosen n values are encoded using the same Huffman encoding method described in the previous section. And then encoded blocks are prefixed with a value of 1 and unencoded blocks are prefixed with a value of 0. The end result is that the optimal compression rates seen with Huffman coding are applied to only the most frequent block value occurrences in the test data set and the complexity of the decoder is reduced from 2^b-1 states to b+n states. This reduced decoder complexity makes selective encoding presented in [1] a good candidate for use in test data compression.

3.3 Alternate Huffman Coding

Now we explain the alternate Huffman coding method presented in this report. The goals behind this method are two-fold. The first is to achieve compression rates that near those of pure Huffman coding. And the second is to achieve the reduced decoder complexity seen with selective coding in [1].

The idea behind selective Huffman coding is to also choose the n most frequent block values to encode. But instead of using 1 bit to indicate which blocks are encoded and which are not, we use a Huffman code prefix to denote unencoded blocks. The idea becomes then to form a Huffman tree for the n selected most frequent values and to add one more leaf to the Huffman tree whose frequency equals the sum of all unencoded block occurrences.

One can then see that with Huffman selective coding, we are saving 1 bit per block in the data but expending several bits per unencoded block when compared against [1]. The overall bits savings seems like it could go either way. But let us go over some experimental test data with typical values for n to get a better idea of the bit costs or savings.

For the experiments and example data, we use the following ISCAS benchmark circuits: s5378, s9234, s38584, s38417, s35932, s15850, s13207 and for each we have a set of test data vectors that provide 100% fault coverage in each circuit. The table below shows the percentage of 4-bit values (b typically chosen as 4) that are a part of the most frequently applicable n values for n values of 3, 4, and 5:

	s5378	s9234	s38584	s38417	s35932	s15850	s13207
n=3	76%	77%	85%	76%	92%	89%	94%
n=4	81%	83%	89%	80%	95%	92%	96%
n=5	84%	90%	93%	84%	97%	95%	98%

Table 4: Total encoded blocks percentages for each circuit test data set

Analyzing the data in the above table, we can see that frequency distribution for block values is highly skewed whereby typically just 3 of the most frequent values out of 2^b possible values (16 in our example) form on average about 85% of the test data set.

This example shows the potential gains that can be seen with forgoing the single bit per b-length block of data bits. Compared against [1], we are saving 1 bit per block for 85% of blocks in the data set and expending and additional 2-3 bits per block for 15% of blocks in the data set. Therefore there is likely a potential overall savings in bits and overall improved compression rates.

3.4 Implementation

In order to provide discrete proof for the type of bit savings that can be seen from the previous example, we provide and compare compression results for each

of three algorithms. In order to provide the compression results, I have implemented the three compression algorithms using the PERL language: Huffman coding [3], Selective coding [1], and Alternate Huffman Coding presented in this report. The three implementations allow obtaining compression results for the test data sets each of the nine ISCAS benchmark circuits: s5378, s9234, s38584, s38417, s35932, s15850, s13207. Furthermore, each implementation is parameterized with the size of the block of data chosen (b) and the number of the encoded blocks (n). This allows obtaining compression rates for multiple b and n value combinations.

RESULTS

Chapter 4

4.1 Experimental Results

Now that we've discussed each of the algorithms in the previous section, we can predict that the compression rate for each algorithm will vary somewhat depending on the values of b and n chosen and depending on the test vector data that is used.

This section shows the compression results for all three algorithms across each of the available test vector data sets for the following ISCAS benchmark circuits: s5378, s9234, s38584, s38417, s35932, s15850, s13207. Furthermore, each algorithm is run on each test data set for various n (3, 4, 5) and b (4, 6, 8) value combinations. The compression rate is defined as

*Compression Rate = [(Original_bits – Compressed_bis) / Original_bits] * 100*

The compression rates for the data set for all circuits are shown in the tables below.

	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
s35932	4	3	58	16	36	7	55	7
		4	58	16	36	8	56	8
		5	58	16	35	9	57	9
	6	3	65	35	50	9	60	9
		4	65	35	51	10	61	10
		5	65	35	51	11	62	11
	8	3	72	63	57	11	63	11
		4	72	63	57	12	63	12
		5	72	63	60	13	67	13

Table 5: Comparison of compression results for s35932 circuit

	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
s13207	4	3	68	16	45	7	66	7
		4	68	16	45	8	67	8
		5	68	16	45	9	68	9
	6	3	75	35	58	9	71	9
		4	75	35	59	10	72	10
		5	75	35	59	11	73	11
	8	3	78	63	65	11	73	11
		4	78	63	65	12	74	12
		5	78	63	66	13	75	13

Table 6: Comparison of compression results for s13207 circuit

	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
s15850	4	3	59	16	39	7	56	7
		4	59	16	39	8	56	8
		5	59	16	38	9	58	9
	6	3	63	35	48	9	57	9
		4	63	35	49	10	59	10
		5	63	35	50	11	60	11
	8	3	66	63	51	11	56	11
		4	66	63	53	12	58	12
		5	66	63	54	13	60	13

Table 7: Comparison of compression results for s15850 circuit

	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
s38417	4	3	45	16	30	7	41	7
		4	45	16	30	8	42	8
		5	45	16	29	9	42	9
	6	3	50	35	39	9	44	9
		4	50	35	40	10	45	10
		5	50	35	40	11	46	11
	8	3	53	63	41	11	43	11
		4	53	63	42	12	45	12
		5	53	63	43	13	46	13

Table 8: Comparison of compression results for s38417 circuit

	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
s38584	4	3	55	16	36	7	52	7
		4	55	16	36	8	53	8
		5	55	16	36	9	54	9
	6	3	60	35	46	9	53	9
		4	60	35	47	10	55	10
		5	60	35	48	11	56	11
	8	3	61	63	47	11	51	11
		4	61	63	49	12	53	12
		5	61	63	50	13	55	13

Table 9: Comparison of compression results for s38584 circuit

There are several things to point out from the above results. The first observation is that the larger the value of b and n, the higher the compression rates because given the frequency distributions for typical test data sets shown in table 4, the Huffman coding provides increased bit savings the longer the block length. This trend naturally applies across all three algorithms, but the cost being that the complexity of the decoder also increases with larger values of b and n. And the more important observation is that the compression rates for the Alternate coding method shows compression rates that rival that of the optimal compression rates of Huffman Coding.

s9234	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
	4	3	45	16	29	7	39	7
		4	45	16	29	8	41	8
		5	45	16	29	9	44	9
	6	3	48	35	35	9	37	9
		4	48	35	37	10	40	10
		5	48	35	38	11	43	11
	8	3	48	63	32	11	32	11
		4	48	63	34	12	34	12
		5	48	63	36	13	36	13

Table 10: Comparison of compression results for s9234 circuit

s5378	b	n	Huffman Coding		Selective Coding		Alternate Coding	
			Compress Rate %	FSM States	Compress Rate %	FSM States	Comress Rate %	FSM States
	4	3	44	16	30	7	41	7
		4	44	16	30	8	41	8
		5	44	16	29	9	42	9
	6	3	49	35	39	9	45	9
		4	49	35	40	10	45	10
		5	49	35	40	11	46	11
	8	3	50	63	41	11	43	11
		4	50	63	42	12	44	12
		5	50	63	42	13	45	13

Table 11: Comparison of compression results for s5378 circuit

The results for the s9234 and s5378 circuit test data sets are shown above. The same general results from the previous test data sets also apply to these two. However, one different trend is that for b value of 8, the compression rates for selective coding and for alternate coding become similar. This is because of the block value frequency distribution in the s9234 and s5378 test data sets. As discussed earlier in section 3, alternate coding leverages the highly skewed frequency distribution of block values. As can be seen in table 4, the frequency distribution of the s9234 and s5378 circuit data set is more evenly distributed between the n selected block values and the non-selected block values. This means that the bit savings seen with alternate coding against those seen with selective coding become less pronounced and the compression rates between the two coding techniques converge.

CONCLUSION

Chapter 5

In conclusion, the experimental results for the six benchmark circuits show the benefits of the Alternate Huffman Coding method presented in this report. According to the experimental results, Alternate Huffman Coding provides compression rates that rival those of optimal Huffman coding while using simpler decoder complexities. The results also show improved compression rates against those obtained by Selective Encoding. The reduced decoder complexity coupled with improved compression rates make Alternate Huffman Coding a strong candidate for use in test data compression.

References

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

[2] N.A. Touba, "Survey of Test Vector Compression Techniques", *IEEE Design & Test Magazine,* Vol. 23, Issue 4, pp. 294-303, Jul. 2006

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

[4] A.L. Crouch, "Design-for-Test for Digital IC's and Embedded Core Systems," 1999

[5] Wang, Wu, Wen, "VLSI Test Principles and Architectures Design for Testability," 2006

[6] 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.