Journal of Theoretical and Applied Information Technology
15 May 2012. Vol. 39 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DOUBLE COMPRESSION OF TEST DATA USING HUFFMAN CODE

[1]R.S.AARTHI, [2]D. MURALIDHARAN, [3]P. SWAMINATHAN

[1] School of Computing, SASTRA University, Thanjavur, Tamil Nadu, India

[2]Assistant professor, School of Computing, SASTRA University, Thanjavur

[3]Dean, School of Computing, SASTRA University, Thanjavur

E-Mail: [1]aarthirs@ymail.com , [2]murali@core.sastra.edu , [3]deanpsw@sastra.edu

ABSTRACT

Increase in design complexity and fabrication technology results in high test data volume. As test size increases memory capacity also increases, which becomes the major difficulty in testing System-on-Chip (SoC). To reduce the test data volume, several compression techniques have been proposed. Code based schemes is one among those compression techniques. Run length coding is one of the most popular coding methodology in code based compression. Run length codes like Golomb code, Frequency directed run Length Code (FDR code), Extended FDR, Modified FDR, Shifted Alternate FDR and OLEL coding compress the test data and the compression ratio increases drastically. For further reduction of test data, double compression technique is proposed using Huffman code. Compression ratio using Double compression technique is presented and compared with the compression ratio obtained by other Run length codes.

Keywords:– *Test Data Compression, Run Length Codes, Golomb Code, OLEL Code, Huffman Code*

1. INTRODUCTION

Compression is possible for data that are redundant or repeated in a given test set. Compression method is broadly divided into lossless and lossy methods, lossless method which can reconstruct the original data exactly from the compressed data and lossy method which can only reconstruct an approximation of the original data. Lossless methods are commonly used for text and lossy methods used for images and sound where a little bit of loss in resolution is often undetectable or at least acceptable.

Test data volume is a major problem encountered in the testing for System-on-Chip (SoC) design. The volume of test data for a SoC is increasing rapidly. The clock rate of the channel, number of channels, amount of test data of the channel are some of the important parameters for the Automatic Test Equipment (ATE) to function. Since all these parameters are large in number, the test time is also high and low on the other case. Therefore, to mitigate the test time, the amount of test data must be decreased. One way to achieve it is to compress the test data.

$$\text{Test time} \geq \frac{\text{Amount of test data on Tester}}{\text{Number of tester channels X tester clock rates}}$$

Test data compression tells about, compressing the test data to reduce the test volume and increasing the compression ratio. Test data compression is divided into three categories: 1. Code-based schemes, 2. Linear-decompression-based schemes, 3. Broadcast-scan-based schemes. Code based schemes use data compression codes to encode the test data. This performs partitioning the original data into groups, and each group is replaced with a codeword to produce the compressed data. Linear - Decompression - based schemes decompress the data using only linear operations with wires, LFSR and XOR networks. Broadcast - scan - based schemes are based on the idea of broadcasting the same value for multiple scan chains, i.e., a single channel drives multiple scan chains [1].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Among the code-based techniques, run length coding is an important methodology. Run length coding is one of the lossless type data compressions like Huffman and Arithmetic coding technique that is based on a statistical model (occurrence probabilities). Universal coding consists of Fibonacci coding, Elias coding, Levenstein coding. Elias coding has Elias Delta, Elias Gamma, and Elias Omega coding. Dictionary based compression also involves LZW coding. More number of efficient compression techniques have been proposed, such as Golomb code [16] [17] [18] Variable-length Input code codes (VIHC)[15], Frequency Directed run length code (FDR) [19] [20], Extended FDR [21], Modified FDR [22], Shifted Alternate FDR [23] [24], which will reduce the total number of test data and also increases the compression ratio.

2. RUN LENGTH BASED CODES

Run length code is a very simple form of data compression technique, in which sequences or runs of consecutive repeated data values are replaced by a single data value. There are several run length based codes like Golomb code, Frequency Directed run length code (FDR), Extended FDR, Modified FDR, and Shifted Alternate FDR. Other than this, we have an OLEL coding methodology. From these techniques, better compression ratio is achieved. To find out the compression ratio the equation is formulated as,

$$\%\text{compression} = \frac{(\text{original test data in bits} - \text{compressed test data in bits})}{\text{original test data in bits}} X\ 100$$

3. HUFFMAN CODING

Huffman code is mapped to the fixed length symbols to variable length codes. Huffman algorithm begins, based on the list of all the symbols or data which are arranged in descending order of probabilities. Next it generates a binary tree, by the bottom-up approach with a symbol at every leaf. This has some steps, in each step two symbols with the smallest frequencies are chosen, added to the top of the partial tree. The selected smallest frequencies symbols are deleted from the list, which are replaced by a secondary symbol denoting the two original symbols. Therefore, the list is reduced to one secondary symbol, which denotes the tree is complete. Finally, assign a codeword for each leaf or symbol based on the path from the root node to symbols in the list [3] [4].

4. SHANON-FANO CODING

The procedure is done by a more frequently occurring string which is encoded by a shorter encoding vector and a less frequently occurring string is encoded by a longer encoding vector. Shannon-Fano coding relied on the occurrence of each character or symbol with their frequencies in a list and is also called as a variable length coding. According to their frequency, the list of symbols is sorted, by the most frequently occurring symbols at the left side and the least most is takes place at the right. Divide the symbols into two sets, with the total frequency of the left set being as close to the total of the right set as possible and assign 0 in the left set of the list whereas assign 1 in right set of the list . Repeatedly divide the sets and assigning 0 and 1 to the left and right set of the lists until each symbol has a unique coding [5].

5. DICTIONARY BASED COMPRESSION

Lempel-Ziv-Welch (LZW) is a category of a dictionary–based compression method [6] [7] [8]. It maps a variable number of symbols to a fixed length code. LZW places longer and longer repeated entries into a dictionary. It emits the code for an element, rather than the string itself, if the element has already been placed in the dictionary.

In a dictionary-based data compression technique, a symbol or a string of symbols generated from a source alphabet is represented by an index to a dictionary constructed from the source alphabet. In a dictionary based coding to build a dictionary that has frequently occurring symbols and string of symbols. When a new symbol or string is found and that is contained in the dictionary, it is encoded with an index to the dictionary. Or else, the new symbol is not in the dictionary, the symbol or strings of symbols are encoded in a less efficient manner.

6. ARITHMETIC CODING

One of the powerful techniques is called arithmetic coding [9]. This converts the entire input data into a single floating point number. Between source symbols and code word, there is no one-to-one correspondence. Here a single arithmetic code word is assigned for an entire sequence of symbols. The interval of real numbers between 0 and 1 is defined by the codeword. The interval denoting it become smaller and the number of bits which are required to represent the interval become larger. According

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

to the probability of occurrence, each bit symbol reduces the interval size.

7. UNIVERSAL CODING

Elias Gamma code, Elias Delta code, Elias Omega code [11] [12] the positive integers are encoded by the universal code, developed by Peter Elias. Levenstein code is also a universal code that encodes the non negative integers by Vladimir Levenshtein. An advantage of universal code is behalf of Huffman codes. It is not mandatory to find the probabilities of the source string exactly.

7.1. Elias Gamma coding:

In Elias Gamma coding, binary values are always expressed as x, $2\lfloor \log_2(x) \rfloor + 1$ and it will begin with 1, in this, few bits as possible. From this, the result will be the decodable code; when the number of zeros in the prefix is doubled, it gives the total code word length that is exactly greater than it by one. After completion of an encoder, the first 1 of codeword length is known. Hence the code is not repeated for minimum number of times. The codeword length will keep on increasing; hence, the entropy approach goes to infinity [12].

7.2. Elias Delta Coding:

Then Elias Delta coding will divide the integer into the highest power of 2. The highest power of 2 contains ($2^{N'}$). And the remaining N' denotes the binary digits of the integer. Encode function where N= N' + 1 is used to encode the data with Elias Gamma coding. Finally combine the remaining N' binary digits to the N. With the representation of its order of magnitude in a universal code, it works by prefixing the integer like Gamma and Delta coding [12].

7.3. Elias Omega Coding:

Other than the two codes, Elias Omega code encodes the prefix recursively. Sometimes it is also called as a recursive Elias code. Omega coding is used to compress the data in which smaller values are much more frequent than larger values. To encode a number, first place a zero at the end of the binary code representation.

Secondly, if the code contains one, stop at that point .Otherwise we need to include the binary code representation of the number as group to the beginning of the code representation. Finally, repeat the second step, and then subtract one from the number of previously calculated digits, after the new encoded number is found.

Fibonacci code is a one type of universal code that converts positive integer numbers into binary code words. Each binary code word ends with 11 and it doesn't have other instances of "11" before the end of the binary codeword [13] [14].

8. GOLOMB CODE [16]

Golomb code was invented by Solomon W. Golomb in 1960. Golomb code is variable-variable length code. The main advantage of Golomb code is the very high compression ratio. Encoding procedure has two steps, in first step a test set T_D is to produce its difference vector test set T_{diff}. So the pre-computed test set is $T_D = \{t_1, t_2, t_3 \ldots t_n\}$. Its difference vector is then given by,

$$T_{diff} = \{t_1, t_1 \oplus t_2, t_2 \oplus_{3,\ldots} t_{n-1} \oplus_n\}.$$

Another step in encoding procedure is to select the Golomb code parameter *m*, this indicates the group size.

Once group size *m* is concluded, the runs of 0s in T_{diff} is mapped to groups of size *m* whereas each group corresponding to a run length. The numbers of such groups are determined by the length of the longest runs of 0s in T_{diff}.

Finally, the code words are separated into equal sized groups m ($m=2^{N,}$ any power of 2). Each group A_k is assigned a group prefix (k-1) runs of 0s ended with 1 and as each group symbols which are unique. The group prefix and a tail of N bit which identifies the members in a group comes under the final code word.

The Table 1 shows the encoding process of Golomb code by the test data. A simple example is explained, the test data T_{diff} is

0000000100000001
0000000100100000
0010000000100000
1000000010000000
1000001000000010
0000001000000011
0000000100000001

Table 1- Golomb Code

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Group	Run length	Group prefix	Tail	Codeword
A1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
A4	12	1110	00	111000
	13		01	111001
	14		10	111010
	15		11	111011

Grouped test data: 00000001 00000001 00000001 001 00000001 00000001 000001 00000001 00000001 000001 00000001 00000001 00000001 1 00000001 00000001 – 112 bits

Compressed data: 1011 1011 1011 010 1011 1011 1001 1011 1011 1001 1011 1011 1011 000 1011 1011 – 62 bits.

% compression = ((112-62)/62) * 100 = 44.6%

Therefore, the compression ratio increases and test data will be reduced.

9. FREQUENCY DIRECTED RUN LENGTH CODE

An FDR code is a variable-to-variable length code which is mapped as variable-length runs of 0s to codeword of variable length. A run length l is defined as a runs of 0s terminating with a 1. Hence, 00000001 is run length of seven (l=7) and a single 1 is a run length of zero (l=0). Each codeword consists of two parts, a group prefix and a tail. Group prefix is used to identify the group to which the run belongs. Tail is used to identify the members within the group [19].

FDR code has some characteristics:

1. The prefix and tail should have equal length in codeword. So, each prefix and the tail are one bit long for group A1, two bits long for A2 etc.
2. The length of the prefix for group Ai equals i. For example, the prefix length is two bits for group A2, three bits for group A3.
3. For any codeword, the prefix is similar to the binary representation of the run length corresponding to the first element of the group. For example, run length 8 belongs to group A3 and the first element of this same group has run length 6. So the prefix of the codeword for run length 8 is 110 and also run length 6 has the prefix value 110.
4. The codeword size increases by two bits, one bit for the prefix and one bit for the tail. As we move from group Ai to group Ai+1. The Table 2 tells the encoding process of FDR.

Table 2 – FDR Code

Group	Run length	Group prefix	Tail	Codeword
A1	0	0	0	00
	1		1	01
A2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A3	6	110	000	110000
	7		001	110001
	--		--	--
	12		110	110110
	13		111	110111

Grouped test data: 00000001 00000001 00000001 001 00000001 00000001 000001 00000001 00000001 000001 00000001 00000001 00000001 1 00000001 00000001 – 112 bits

Compressed data: 110001 110001 110001 1000 110001 110001 1011 110001 110001 1011 110001 110001 110001 00 110001 110001 – 86 bits

%compression = ((112-86)/112)*100 = 23.20%

10. EXTENDED FDR

The extended form of FDR is EFDR, which is used to compress the data as efficient as FDR. So to encode both runs of 0s which ends with bit 1 and runs of 1s which ends with bit 0 are encoded like FDR but an extra bit is included at the beginning of the code word. If the test data encodes runs of 0s, then add '0' bit at the beginning of the code word whereas runs of 1s, then add '1' bit at the beginning of the code word. So that the appended bit of the codeword is 0, then the run is of 0 type and if the appended bit of the codeword is 1, then the run is of 1 type. However, in this code there is no run length of size zero, because we encode both runs of 0s and 1s. In this code when moving from group Ai to group Ai+1, the length of the codeword increases by two bits, one for the prefix and one for the tail. In EFDR, the codeword which has one extra bit is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

included, when compared to FDR [21]. The Table 3 shows an encoding procedure for extended form of FDR. This will reduce the given test data and named as compressed data.

Table 3 - EFDR Code

Group	Run length	Group prefix	Tail	Codeword Runs of 0s	Codeword Runs of 1s
A1	1	0	0	000	100
	2		1	001	101
A2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	9		010	0110010	1110010
	10		011	0110011	1110011
	11		100	0110100	1110100
	12		101	0110101	1110101
	13		110	0110110	1110110
	14		111	0110111	1110111

Grouped test data: 00000001 00000001 00000001 001 00000001 00000001 000001 00000001 00000001 000001 00000001 00000001 00000001 10 0000001 00000001 – 112 bits

Compressed data: 0110000 0110000 0110000 001 0110000 0110000 01010 0110000 0110000 01010 0110000 0110000 0110000 100 01011 0110000 – 98 bits

% compression = ((112-98)/112)*100 = 12.5%

10. MODIFIED FDR

The Modified FDR code is a class of variable-to-variable length encoding method, which maps variable-length runs of zeros to code words of variable length. MFDR code can acquire better compression efficiency than FDR code. On the basis of analysis, when the probability of 0s in the test set is greater than 0.856, a better compression ratio than FDR is obtained [22].

MFDR code has some properties:
1. MFDR codes are separated into groups A1, A2, and so on. Each code word is combined with the group prefix and tail.
2. A parameter is denoted as r (≥1), is given to promote the flexibility and the compression ratio.
3. MFDR codes consist of three class,

a) First class represents the group A1, which consists of set of run length L, where $0 \le L \le 2^{r+1} - 1$. The prefix is set as 01 and the tail length is r+1.
b) Second class represents the group as A2, A4, A6, A8…A2k, where k≥1. The group prefix can be indicated as $p_k p_{k-1} \ldots p_0$, where p_m = 1when $1 \le m \le k$ and p_0 = 0. The tail part of a codeword contains k+r bits.
c) Third class includes the groups as A_3, A_5, A_7, $A_9 \ldots A_{2k+1}$, where k≥1. The group prefix can be denoted as $p_{k+1}\ p_k\ p_{k-1} \ldots p_0$, where p_m = 0when $1 \le m \le k+1$ and p_0 = 1. The tail part contains k+r bits.

Table 4 - MFDR Code

Group	Run length	Group prefix	Tail	Codeword
A1	0	01	00	0100
	1		01	0101
	2		10	0110
	3		11	0111
A2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A3	8	001	00	00100
	9		01	00101
	10		10	00110
	11		11	00111
A4	12	110	000	110000
	13		001	110001
	--		--	--
	18		110	110110
	19		111	110111
A5	20	0001	000	0001000
	21		001	0001001
	--		--	--
	--		--	--
	26		110	0001110
	27		111	0001111

Thereby, the encoding procedure is illustrated in the Table 4. The given test data is compressed by the codeword and hence reduced bits are acquired.

Grouped test data: 00000001 00000001 00000001 001 00000001 00000001 000001 00000001 00000001 000001 00000001 00000001 00000001 1 00000001 00000001 – 112 bits

Compressed data: 1011 1011 1011 0110 1011 1011 1001 1011 1011 1001 1011 1011 1011 0100 1011 1011 - 64 bits

% compression = ((112-64)/112)*100 = 42.8%

11. SAFDR

Shifted alternate FDR is an evolution of Alternating run length. It consists of alternate runs of zeroes or runs of ones. In shifted alternate FDR, only one bit will be added i.e., add '0' or '1' at the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

beginning to indicate the type of first run length. Then the successive run lengths are automatically allocated.

In SAFDR, there is no run length of size '0'. So code word starts with the run length as 1 and so on. From the alternating run length coding, the run length is shifted upward by one position. This SAFDR performs higher compression compared to alternate FDR. It is also a variable-to-variable length encoding method and comprise of two parts, group prefix and tail. The prefix identifies the group in which the run length occurs, and the tail denotes the member in the group. Prefix and tail are of equal length [23] [24].

Table 5 – SAFDR

Group	Run length	Group prefix	Tail	Codeword
A1	1	0	0	00
	2		1	01
A2	3	10	00	1000
	4		01	1001
	5		10	1010
	6		11	1011
A3	7	110	000	110000
	8		001	110001
	9		010	110010
	10		011	110011
	11		100	110100
	12		101	110101
	13		110	110110
	14		111	110111

Normally, this SAFDR is same as the FDR codeword, but the difference is to start from the run length 1. The Table 5 shows an encoding procedure for compressing the test data and the original test data is reduced to form a compressed data.

Grouped test data: 0000000 1 0000000 1 0000000 1 00 1 0000000 1 0000000 1 00000 1 0000000 1 0000000 1 00000 1 0000000 1 0000000 1 0000000 11 0000000 1 0000000 1 – 112 bits

Compressed data: 0 110000 00 110000 00 110000 00 01 00 110000 00 110000 00 1010 00 110000 00 110000 00 1010 00 110000 00 110000 110000 01 110000 00 110000 00 – 110 bits

% compression = ((112-110)/112)*100 = 1.7%

12. OLEL CODING

OLEL coding is also a variable-to-variable length encoding method. The codeword is divided into two parts according to the position, odd bits and even bits. The odd bits of codeword represent the length of the runs and the even bits of codeword represents whether a run is finished.

The encoding procedure for OLEL coding as, the column 1 specifies the length of the runs and the column 2 has the number of groups. The column 3 indicates codes for corresponding runs and the column 4 has improved codes for corresponding runs.

The every value of column 3 is greater than the length of the corresponding runs. The Codes (Column 3) are calculated by adding two to the run length. For example, the run length 5 is added with number 2 and converting it to the binary number $(5+2)_{10} = (7)_{10} = (111)_2$. In the column 3 in the Table VI, every code has the highest bit in the codes that is 1, so to reduce the 1 bit from the codes and to form improved codes; the eliminating 1 is no need to be encoded.

The last column is codeword; all odd bits of the codeword are corresponding improved codes. And all even bits of the codeword are labels, which indicate whether the codeword ends. When the label is 1, the corresponding codeword ends. Otherwise it represents that the corresponding codeword continues [25].

Finally, the Table 6 gives an encoding procedure for OLEL coding and produces the compressed bits.

Grouped test data: 00000001 00000001 00000001 001 00000001 00000001 000001 00000001 00000001 000001 00000001 00000001 00000001 1 00000001 00000001 – 112 bits.

Compressed test data: 000011 000011 000011 0001 000011 000011 1011 000011 000011 1011 000011 000011 000011 01 000011 000011 – 86 bits.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 6 – OLEL Code

Run length	Group	Codes	Improved codes	Code word
0	A1	10	0	01
1		11	1	11
2	A2	100	00	00 01
3		101	01	00 11
4		110	10	10 01
5		111	11	10 11
6	A3	1000	000	00 00 01
7		1001	001	00 00 11
8		1010	010	00 10 01
9		1011	011	00 10 11
10		1100	100	10 00 01
11		1101	101	10 00 11
12		1110	110	10 10 01
13		1111	111	10 10 11

%compression = ((112-86)/112)*100 = 23.21%

13. PROPOSED DOUBLE COMPRESSION USING HUFFMAN CODE TECHNIQUE

In this method, we perform a double compression method. First, the data is compressed by any one of the run length based codes like Golomb code, FDR code, EFDR, MFDR, SAFDR, and OLEL coding.

From the compressed data, make another compression by Huffman code. Huffman code is constructed by the frequency of more number of repeated data values from the first compressed data. Based on the frequency to make a Huffman tree and form a variable length code word. The second compressed data is obtained by Huffman code and the first compressed data is obtained by Golomb code or any other run length codes are tailored together and the final double compression is performed. Golomb gives the highest compression ratio compared to other run length codes, so we choose Golomb code for the double compression technique. While adding Golomb code word, we must add a flag. This flag will separate the Golomb code word and Huffman code word in second compressed data. This flag should not be available in either Golomb code word or Huffman code word.

The first compressed data is normally generated by Golomb code.

Grouped test data: 00000001 00000001 00000001 001 00000001 00000001 000001 00000001 00000001 000001 00000001 00000001 00000001 1 00000001 00000001 – 112 bits

First Compressed data: 1011 1011 1011 010 1011 1011 1001 1011 1011 1001 1011 1011 1011 000 1011 1011 – 62 bits

Table 7 – Frequency of each symbol

Golomb Codeword	Frequency
1011	12
1001	2
010	1
000	1

The repeated codeword data value of Golomb compressed data that is, first compressed data is assigned in an above Table 7 with their probabilities, which represents the redundancy of the each value. Based on this probability, a Huffman tree and a new codeword for double compression technique are constructed.

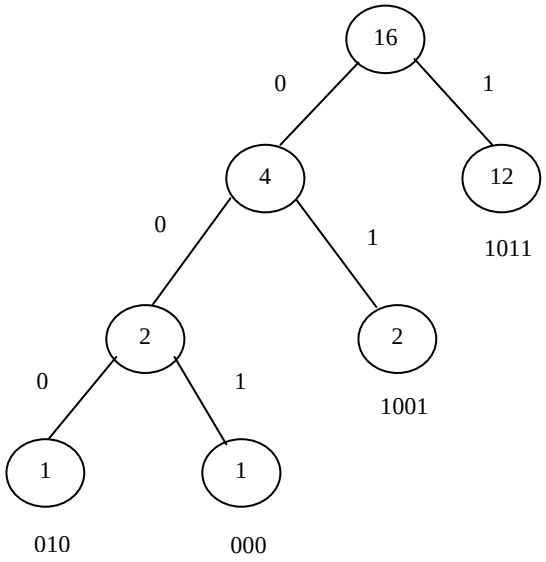

Figure 1 - Huffman Tree

The Huffman code word is generating from the above tree Figure.1. Tree is traversed in a bottom up approach. The smallest probability symbols are selected and are summed up to the top of partial tree. Then the smallest possibility symbols are deleted from list and the partial tree top node is replaced by the sum of the smallest possible symbols. If the list is reduced to just one symbol,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

then the tree is complete. Then the tree is travelled across to determine code word of the symbols.

Codeword is formed by the tree, the left side path of the tree is assigned as 0 and right side path of the tree is assigned as 1. The codeword is written by the path from root node to specified symbol in the list.

Table 8 – Huffman codeword and their frequency

Golomb Codeword	Frequency	Huffman Codeword
1011	12	1
1001	2	01
010	1	000
000	1	001

Finally the second compressed data of double compression is formulated by the Table 8,

Compressed data: 1 1 1 000 1 1 01 1 1 01 1
1 1 001 1 1 00000
1011 1001 010 000 00000
1 01 000 001 - 55 bits

The second compressed data should be the combination of Huffman codeword and also Golomb codeword. The flag is assigned as five zeroes (00000), in order to differentiate the Golomb codeword and Huffman codeword. This double compression method will reduce the test data.

%compression= ((112-55)/112) * 100 = 50.8%

14. COMPARISON RESULTS

The comparison results of various run length based codes and double compression technique using Huffman code.

Table 9 – Comparison of run length codes and double compression using Huffman code

Run Length Code	% Compression Ratio
Golomb Code	44.60%
FDR Code	23.20%
EFDR Code	12.50%
MFDR Code	42.80%
SAFDR Code	1.70%
OLEL Code	23.21%
Double Compression Technique	50.80%

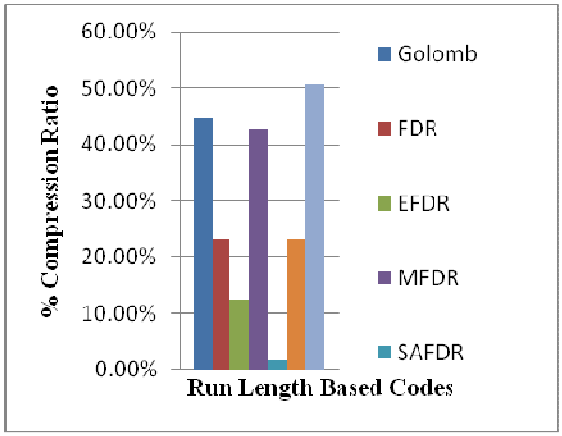

Figure 2 – compression ratio of run length codes and double compression technique

15. RECOMMENDATION

From the above-mentioned compression techniques, the test data compression can be achieved in a better way using all the run length based code technique. The main aim of the double compression technique proposed is to reduce the test data volume and area even further. When the data set has redundant data then compression will be better when compared to the other sets. To give better compression ratio and reduce the volume of test data the number of redundant data should be high for the proposed compression technique. This can be further enhanced by improvising the compression technique.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] Usha S. Mehta, Kankar S. Dasgupta, Niranjan M. Devashrayee, "Hamming Distance Based Reordering and Columnwise Bit Stuffing with Difference Vector: A Better Scheme for Test Data Compression with Run Length Based Codes", *IEEE Computer Society 2010, 23rd International Conference on VLSI Design*.

[2] N. A. Tauba, "Survey of Test Vector Compression Techniques", *IEEE transaction Design & Test of Computers, 2006.*

[3] David a. Huffman, Associate, IRE "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the I.R.E.*

[4] Reza Hashemian, Senior Member, IEEE, "Memory Efficient and High-speed Search Huffman Coding", *IEEE Transactions on Communications,* October 1995, VOL. 43. NO**.** IO.

[5] Luis *g.* Rueda and b. John oommen, "Enhanced Static Fano Coding" *IEEE 2001*.

[6] P. Sismanoglou and D. Nikolos, "Test Data Compression based on the Reuse of Parts of the Dictionary Entries", *IEEE 2011*.

[7] S. Kwong and Y. F. Ho, "A Statistical Lempel-Ziv Compression Algorithm for Personal Digital Assistant (PDA)" *IEEE Transactions on Consumer Electronics,* Vol. 47, No. 1, February 2001.

[8] Lei Li and Krishnan Chakrabarty, "Test Data Compression Using Dictionaries with Fixed-Length Indices", *Proceedings of the 21st IEEE VLSI Test Symposium (VTS.03).*

[9] Paul G. Howard and Jeffrey Scott Vitter, "Practical Implementations of Arithmetic Coding" *Providence, R.I. 02912{1910.*

[10] Shengtian Yang, Member, IEEE, and Peiliang Qiu, Member, IEEE, "Efficient Integer Coding for Arbitrary Probability Distributions", *IEEE Transactions On Information Theory*, VOL. 52, NO. 8, August 2006.

[11] P. Elias, Universal codeword sets and representations of the integers." *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 194-203, March 1975.

[12] Jiri Walder, Michal Kratky, and Jan Platos, "Fast Fibonacci Encoding Algorithm", *Dateso 2010*, pp. 72{83, ISBN 978-80-7378-116-3.

[13] Shmuel T. Klein, Miri Kopel Ben-Nissan, "On the Usefulness of Fibonacci Compression Codes", *the Computer Journal, 2005.*

[14] X. Kavousianos, E. Kalligeros and D. Nikolos, "Multilevel-Huffman test-data compression for IP cores with multiple scan chains", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* , Volume 16 Issue 7, July 2008.

[15] Jas and N.A. Touba, "Test Vector Compression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs," *Proc. Int'l Test Conf. (ITC 98), IEEE CS Press*, 1998, pp. 458-464.

[16] Chandra and K. Chakrabarty, "Test Data Compression for System-on-Chip Using Golomb Codes" VTS '00**:** *Proceedings of the 18th IEEE VLSI Test Symposium, 2000.*

[17] Chandra and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes", *IEEE transaction on Computer –Aided Design of Integrated Circuits and Systems,* Vol. 20, No. 3, March 2001.

[18] Chandra and K. Chakrabarty, "Efficient test data compression and decompression for system-on-a-chip using internal scan chains and Golomb coding", DATE '01**:** *Proceedings of the conference on Design, automation and test in Europe,* March 2001.

[19] Chandra and K. Chakrabarty, "Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression" in the *Proceedings of the 19th IEEE VLSI Test Symposium*, March 2001.

[20] Chandra and K. Chakrabarty, "Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-Directed Run-Length (FDR) Codes" *IEEE Transactions*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

on Computers, Volume 52 Issue 8, August 2003.

[21] El-Maleh and R. Al-Abaji., "Extended frequency-directed run-length code with improved application to system-on-a-chip test data compression", *Proc. Int. Conf: on Electronics, Circuits and Systems*, 2:449-452, Sep 2002.

[22] J. Feng and G. Li, "A Test Data Compression Method for System-on-a- Chip" *4th IEEE International Symposium on Electronic Design, Test and Applications, 2008. DELTA 2008.*

[23] Chandra and K. Chakrabarty, "Reduction of SOC test data volume, scan power and testing time using alternating run-length codes", DAC '02: *Proceedings of the 39th conference on Design automation*, June 2002.

[24] S. Hellebrand and A. Würtenberger, "Alternating Run-Length Coding – A Technique for Improved Test Data Compression", *Handouts 3rd IEEE International Workshop on Test Resource Partitioning, Baltimore, MD, USA*, October 10 – 11, 2002.

[25] Wenfa Zhan. "An Efficient Collaborative Test Data Compression Scheme Based on OLEL Coding" *IEEE 2008.*