International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 5 Issue: 2 December 2019

Variable-to-Variable Run Length Encoding Technique for Testing Low Power VLSI Circuits

[1]Zarul Fitri Zaba, [2]Idyawati Hussein
[1]School of Computer Sciences, Universiti Sains Malaysia, 11800 Minden, Pulau Pinang, Malaysia
[2]School of Computing, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah, Malaysia

Abstract: Enhancing the integration capability in semiconductor technology requires a large amount of test data, resulting in higher memory, time for transition and time for testing. A novel lossless data compression technique is proposed in this paper to reduce test data, time and memory based on the encoding scheme factor to variable run size. A test data are partitioned into variable length test patterns in this scheme and the bits are compressed into variable length codes by applying the compression algorithm. With a limited number of code words, the encoding technique enhances the reduction of test data. The compression technique is effective, especially when the 0s and 1s runs in the test set are high and compress the data streams composed of 0's and 1's runs efficiently. The variable to variable run length code algorithm is used to make changes to test vectors and can be adapted to compress pre-computed test sets to test system-on-chip (SOC) embedded cores.

Keywords: Pre-computed test sets; Compression codes; Decompression; Embedded core testing; SOC testing

1. INTRODUCTION

Due to the advent of integration in semiconductor technology, the electronics industry has experienced a phenomenal growth over the past two decades. Integrated circuits consist of various types of active and passive components that are manufactured to form systems on a single chip. The multiple modules and IP cores in a system-on-chip (SOC) wrap functions such as a processor, memory and various technologies such as CMOS logic into analog circuits. The large integrated devices are made up of millions of transistors and numerous hardware modules with the increase of technology. The SOC, however, presents many challenges such as increased test data, automatic test equipment (ATE) memory storage, transition time, test time, test power, and TAT. Test data is usually generated and stored on workstations. Individual types of application-specific integrated circuit (ASIC) require a more frequent download from workstation to ATE of test data. The ASIC test sets will be as large as gigabytes, and the time taken to upload the test data is more significant as it takes several minutes to hours to download. ATE's performance is affected by uploading test data time. To improve ATE's throughput, reducing the download time of test data is very important. A high volume of test data is directly proportional to higher memory and transition time. Because of limited bandwidth, memory, and I / O channels, the transfer of large test data between the ATE and chip is a bottleneck [1]. During data transfer from ATE to the test device (DUT), the Limited bandwidth increases the test time and test costs. Built-in self-test (BIST) and test-data compression are the techniques generally used to minimize SOCs test data size and test request time. Some of the problems are as follows when the test data increases: Limited space on ATE, Long upload time, Limited I / O bandwidth. Compression of test data is a promising solution for storing and transmitting compressed data from ATE to chip as well as speeding up the interaction between ATE and SOC during the test. Data compression is a data file size reduction procedure. The use of compression test data is to compress the pre-computed test set (TD) provided by the core vendor to a smaller test set (TE) and then stored in the memory of the automatic test device. Test data reduction reduces the size of ATE memory requirements, test time, and test power [2]. A new compression of test data is required to test the SOC core without exceeding memory, bandwidth and power limitations. Before and after scanning chains, additional on-chip hardware is added. An on-chip decoder will decompress the compressed memory test data and supply the original data to the test device. After decompression, the lossless test data compression reproduces all the bits [3]. Test vector compression consists of three categories as follows [4,5]: code-based schemes use a data compression technique to encode test cubes, linear-decompression-based schemes

decompress data using only linear operations [6], broadcast-based schemes transmit the same value to multiple scan chain. The code-based scheme approach partitions the pre-computed test sets (TD) into symbols and replaces a symbol with codeword by applying a compression algorithm to form encoded data. Decoding is done using a decoder that simply converts each codeword to the original data. In these methods, structural circuit information under tests (CUT) is not required but is well suited for IP core-based SOCs. The different categories are described as follows [7, 8] based on this scheme: run-length-based codes, dictionary-based codes, statistical codes [9]. The Run-Length coded scheme is a highly effective data compression method for testing SOCs with a large number of IP cores in the current generation. In [10], the Simple run length code scheme is used to encode 0s runs into words with fixed length code. The architecture of the cyclic scan chain is used to increase the frequency of 0s by allowing different vectors to be applied and test cube reordered. The vectors of difference between test cubes are equivalent to two test cubes XOR and encoded with a run-length code. The Golomb-based compression technique is proposed to encode 0s runs with variable-length code words [11-13]. Here, each group is made up of specific identification symbols. The words of the variable length code are used to encode longer data runs efficiently. The frequency-directed run-length codes are similar to the Golomb codes suggested in [14,15] and variable group size is the difference in both methods. FDR is a variable to variable length encoded scheme and is a method of mapping variable length runs of 0s to variable length code words after application of the compression algorithm. The FDR codes are not very effective when runs of 1s are high on test sets. This coding scheme is effective for compressing data for several 1s and long runs of 0s, but it is inefficient for data streams consisting of both 0s and 1s runs. To decompress FDR code, the on-chip decoder must identify the prefix and the tail. FDR requires a powerful, high-range overhead decoder. In order to overcome the decoder difficulty in FDR, the Huffman approach and FDR were combined to use the variable length pattern as input to the Huffman algorithm instead of using fixed length pattern [16, 17]. This therefore maintains the compression ratio due to the FDR process and uses limited Huffman to reduce the field overhead. In place of X-bits, 0s and 1s are filled to improve the occurrence of block frequencies [18]. Used to maximize the runs of 0s in the zero-fill algorithm, it fills the 0s instead of unspecified bits to reduce the scan-in test power [19]. The Extended Frequency Directed Run-Length Coding [EFDR] and Alternating Run-Length coding shown in [7, 20] defines that EFDR is ideal for test data streams consisting of both 0s runs followed by 1s runs and vice versa. The 0s runs followed by 1 are encoded in the EFDR method as in FDR, but the difference here is an additional bit is added at the beginning of the FDR code word. The Alternating Run-Length code is a variable to variable length code, and here the test set is composed of alternating 0s runs and 1s runs. It is possible to identify the data runs by adding variable ' a ' to the core. When a=0, the run-length is considered to be runs of 0s, when a=1 is considered to be run-lengths of 1s. Only nine code words are used in [21] to encrypt the test data, and the technique of encoding is flexible. In order to obtain a higher compression ratio, variable nine coded compression uses a variable length block for each pattern. In [22,23], the multi-stage encoding technique, i.e. alternating frequency-directed equal run-length (AFDER) and run-length-based Huffman encoding (RLHC), is proposed to reduce test data and application time. Together with nine-coded compression technique, multi-stage encoding enhances the reduction of test data. A method called alternating variable run-length codes (AVR) in [24] reduces the test data, scan power consumption, test application time (TAT). In test sets up to 0s and 1s, proper mapping of don't care results in average and peak power consumption savings without slower scan clocks. Using extended variable length codes [25] to reduce data, time and memory, a test data compression scheme based on fixed to variable length coding with a limited number of code words is proposed. In many of the code-based compression techniques, the main objective is to reduce the volume of test data without giving any priority to the reduction of test power, for example, the test compression techniques outlined for [19, 20, 22, 26] focused mainly on the reduction of test data. In some of the independent compression techniques, for example, techniques detailed in [22, 27], the test power and test data are reduced.

2. MAIN CONTRIBUTION

A new method for testing embedded processor core using pre-computed test sets was presented in this paper based on variable to variable run length code. This method provides an efficient way of reducing the required test data and memory for automatic test equipment (ATE). By applying an algorithm to each variable test pattern to enhance the compression ratio, the compression technique efficiently compresses the runs of 0s and 1s. Here, a different code word pattern is applied to runs of 0 followed by 1 and also to runs of 1 followed by 0, in order to determine whether the run length pattern is either run of 0's or run of 1's. If the test pattern is 0's followed

by 1's, and then the compressed code starts with a bit' 0.'If one runs the test pattern followed by 0, then the compressed code starts with a bit ' 1. 'The following is the organization of this paper. Section 2 explains the detailed compression technique of variable to variable run length and describes the process of data compression using an algorithm, flowchart and decompression. Section 3 explains the ISCAS reference circuit experimental results obtained.

Variable to Variable Run Length Code

In this section, the detailed compression technique of variable to variable run length is presented and also describes the process of data compression using the architecture of algorithm, flowchart and decompression.

Variable to variable run length code

In this section, the detailed compression technique of variable to variable run length is presented and also describes the process of data compression using the architecture of algorithm, flowchart and decompression. Take TD = {t1, t2, t3 ... tn}, where n is the number of bits in pre-computed test sets (TD). Partition the test sets into variable run length patterns and use code word to compact the test sets. For both 0's runs and 1's runs, the code word will be different. Table 1 illustrates the variable-to-variable-length coding scheme encoding example. This method consists of two types of run length pattern (i.e. 0 runs followed by 1 runs followed by 1 runs followed by 0 runs). A code word is allocated separately for each type of runs. The code word for 0's runs in each group is the inverted code word data for 1's runs. The code word present in each group determines a 2n pattern, where n is 0, 1, 2, 3... According to the test data, 0's runs were considered as 0's strings followed by a bit' 1' and 1's runs were considered as 1's strings followed by a bit' 0.'For instance, 000001 and 00001 are a 0's run sequence and their run length is 5, 4. 11111110 and 11110 is a 1's run pattern with a running length of 7, 4. The start bit of code word specifies which type of runs were processed and the code word length is used to identify the unit. The difference between the proposed method and other variable run length codes shows that separate code words are assigned for both 0's and 1's runs, since no prefix and tail are considered here.

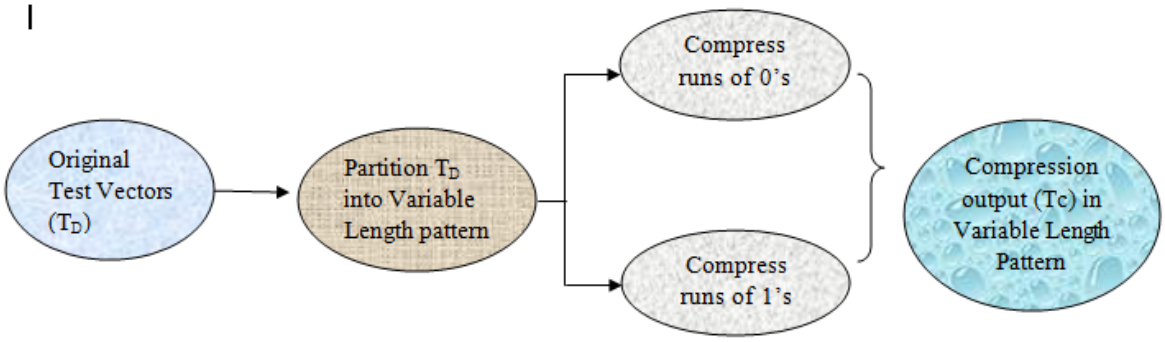

Figure 1. Block Diagram of Variable to Variable Run Length Coding

Table 1. Example of Proposed Coding Scheme

Group	Run	Code World Runs of 0' s	Code word Runs of 1's	Code word Length
B0	1	0	1	1
B1	2	01 00	10 11	2
B2	4 5 6 7	011 010 001 000	100 101 110 111	3

B3	8 9 10 11 12 13 14 15	0111 0110 0101 0100 0011 0010 0001 0000	1000 1001 1010 1011 1100 1101 1110 1111	4
B4	16 … 31	011111 … 00000	10000 … 11111	5
B5	32 … 63	011111 … 000000	100000 … 111111	6
….	…..	….	…..	…..

From Table 1 observation, the test pattern is divided into 0's runs followed by 1 and 1's runs followed by 0. The test patterns are mapped to the appropriate code word based on run type and number of repeated bits or run length. The group size is determined by the group number, and each group's members are equal to 2 m, where m is the group number, m = 0, 1, 2, 3 ... For both runs of 0's and 1's, the code word bit size is equal to n. A group B0 is made up of one (20) member. Note that B0 output will be the pattern of 21(2n) (i.e.) 0 and 1. In B0, the run length test pattern is 1 (viz) 01 (0's runs) and 10 (1's runs). The run length is 1 for 10 and the code word is 1. The run length for 01 is 1 and the code word for 0 is inverted code word data 1. A B1 group is made up of two (21) members. Notice that B1 production will consist of 22 patterns (i.e. 00, 01, 10, and 11). For B1, the run-length test pattern is 2 (viz) 001 (0's runs) and 110 (1's runs). The run length is 2 for 110 and the code word is 10. The run length for 001 is 2 and the code word is 01, which is code word 10 inverted data. For patterns of 110 and 001, the output is 10 and 01. If the running length test pattern is 3 (viz) 0001 (0's runs) and 1110 (1's runs). The run length is 3 for 1110 and the code word is 11. The run length for 0001 is 3 and the code word is 00 which is inverted code word 11 info. So, the output is 11 and 00 for pattern 1110 and 0001. The running bit size of 2 and 3 is the same. A group B2 is made up of 4 (22) members. Note that B2 will result in 23 patterns (i.e. 000, 001, 010 ... 110, 111). For B2, the run-length test pattern is 4 (viz) 00001 (0's runs) and 11110 (1's runs). The run length is 4 for 11110 and the code word is 100. The run length for 00001 is 4 and the code word is 011, which is code word 100 inverted data So the output for the pattern of 11110 and 00001 is 100 and 011. If the running period test pattern is 5 (viz) 000001 (0's runs) and 111110 (1's runs). The run length for 111110 is 5 and the code word is 101. The run length for 000001 is 5 and the code word is 010, which is code word 101 inverted info. So, the production is 101 and 010 for sequence 111110 and 000001. The run length is 6 for 1111110 and the code word is 110. The run length for 0000001 is 6 and the code word is 001, which is code word 110 inverted information. The output for the pattern 1111110 and 0000001 is 110 and 001. The run length is 7 for 111111110 and the code word is 111. The run length is 7 for 111111110 and the code word is 111. The run length for 00000001 is 7 and the code word is 000 that is inverted code word 111 data. So the output for the pattern 11111110 and 00000001 is 111 and 000. The running bit size of 4, 5, 6 and 7 is the same. Up to m group numbers are continuing this cycle. Here, in order to reduce test data, a run of 0's and 1's is mapped to shorter code words. Figure 2 demonstrates the example of encoding set to fixed run length encoding scheme. As an example from the benchmark circuit, the test vector is considered. For each pattern, an algorithm is applied (see Table 1). From the encoding procedure example, note that for runs of 1's the start bit of code word is 1 and for runs of 0's the start bit of code word is 0. The original bits number is 55, while the compressed bits are 28.

Data compression procedure using algorithm and flow chart

The Algorithm 1 and Figure 3 describe the compression algorithm process using the proposed scheme of coding. If the test pattern is 1's run, then 1's run length is encoded as 2n code word. If the test pattern is running from 0's, then 0's run length is encoded as 2n code word inverted information. 2n code word is allocated for 2 m run length pattern as shown in the case where m = 0, 1, 2, 3 ... N= 1,

International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 5 Issue: 2 December 2019

2, 3 ... The performance is assigned to the pattern of 20 run lengths (01 or 10). 22 bit pattern is assigned as output for 21 run-length patterns (001 or 110, 0001 or 1110). 23 bit pattern is allocated as output for 22 run length patterns (00001 or 11110, 000001 or 111111, 0000001 or 1111110). This goes on up to m patterns and n word code.

Algorithm 1.Variable to Variable Run Length Coding Algorithm

1. Generate test vectors (TD) pre-computed.
2. Let x be the vector of the input test, I'm the starting position, and i+1 is the next position.
3. Find the TD's size.
4. Attribute count = 0.
5. If x [i] = = x [i+1], count multiple bits (count = count + 1).
6. If the test pattern is 1's ride, give count interest.
7. If test pattern is 0's, give count meaning transformation (0 as 1, 1 as 0)
8. Case:
   - If 1 ≤ count < 2, assign count as 21 code word
   - If 2 ≤ count < 4, assign count as 22 code word
   - If 4 ≤ count < 8, assign count as 23 code word
   - If 8 ≤ count < 16, assign count as 24 code word
   - If 16 ≤ count < 32, assign count as 25 code word
   - If 32 ≤ count < 64, assign count as 26 code word
9. Repeat the algorithm till end of data stream
10. Calculate TC = Total compressed bits

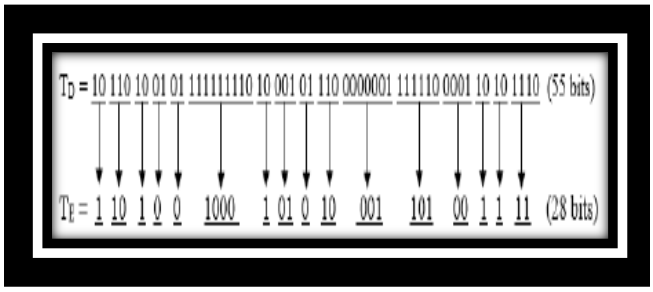

Figure 2. Example of encoding procedure of variable to variable run length code

3. DECOMPRESSION ARCHITECTURE

Figure 4 demonstrates the design of decompression, which is used to decompress the encoded data. The decoder is scalable and simple. The architecture is made up of finite state machines, clocks, and exclusive OR gates. The bit-in is the FSM's entry. The activate signal is used when the decoder is ready to monitor the encoded information. The signal shift is used to control the codeword to be passed via exor gate to the m-bit counter. Signal dec is used to decrease counter and rs is used to indicate counter reset status.

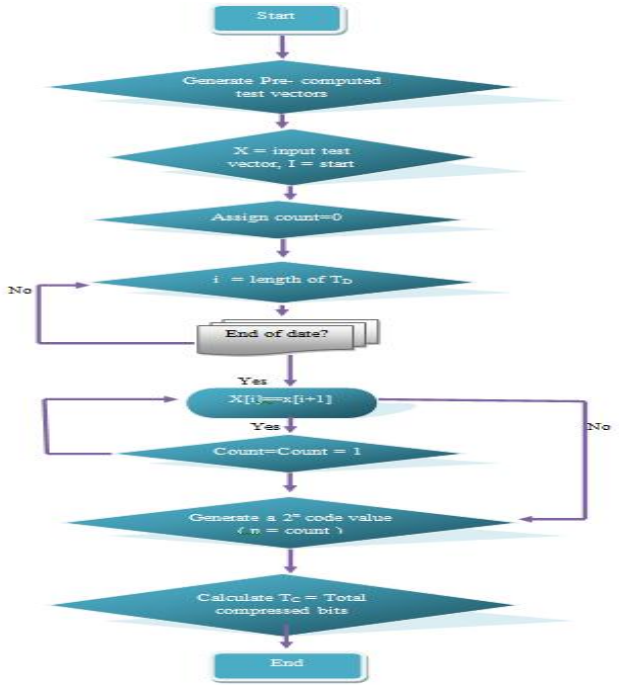

Figure 3. Flow Chart of Variable to Variable Run Length Coding

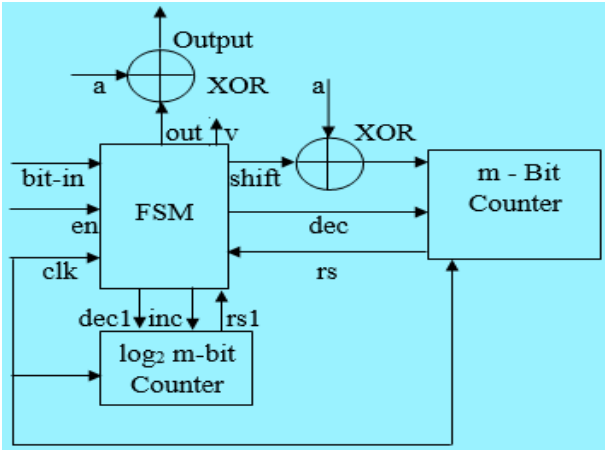

Figure 4. Conceptual architecture of Decompression

To decode the code word into run-length pattern, the counter of log2 m-bits was used to count the length of the code word. The inc and dec1 are used to increase and decrease the counter and rs1 is used to indicate the end of counting. The output signal from the FSM controls the ex- or gate and indicates if the decoding of runs of 1's is complete. The signal v shows the valid output. Using FSM, the sequence is detected and FSM output is code word. The code word starts with a bit ' 0 ' for run type 0's

and the code word begins with a bit ' 1 ' for run type 1's. If bit-in is ' 0 ' the code word is a compressed run type 0's code and if bit-in is ' 1 ' it is a compressed run type 1 code.

The operation of the decoder is explained as follows:

Signal en will initially be high and ready to receive bit-in data. If the input bit-in is 1, it will be 0 and if the input bit-in is 0, it will be 1. After the ex-or operation phase, when the signal shift is high, the data fed to the counter. If bit-in is 1, and a= 0, the execute type 1 code word will not be changed. It remains as the compressed original code. If the compressed code is 1011 ex-or 0, for example. The performance is going to be 1011. If bit-in is 0, and a = 1, the execute sort 0 code word will be modified. This is because the corresponding pattern of run length can be achieved. For example, if the compressed code is 0100, then the output should be 11 zeros followed by 1. However, 0100's length is 4 and this is 1's inverted code word runs info. In order to reach the correct output, the 0100 should be reversed again. For example, the output will be 1011 if the compressed code is 0100 ex-or with 1. In order to reach the correct output, the 0100 should be reversed again. For example, the output will be 1011 if the compressed code is 0100 ex-or with 1. The m-bit counter is then decreased; allowing signal dec to go high until rs was high. The signal v shows a valid output. The data from the ex-or gate output was moved to the m-bit counter until the counter value of log2m-bit was 0. So dec1 goes up and decrement of the counter. The rs1 signal went up when the counter log2m-bit reached state 0. It implies moving the code word to the m-bit counter. The FSM outputs the 1's corresponding to the word code and shows the correct output of the signal v. The FSM output data such as 11110 is ex-or 0 when bit-in = 1 and a = 0. So the output is going to be 11110. This indicates that form 1 runs the decoded data. If bit-in = 0 and a = 1, the FSM output data like 11110 is ex-or 1. So that's going to be 00001 output. This indicates that type 0's are running the decoded data.

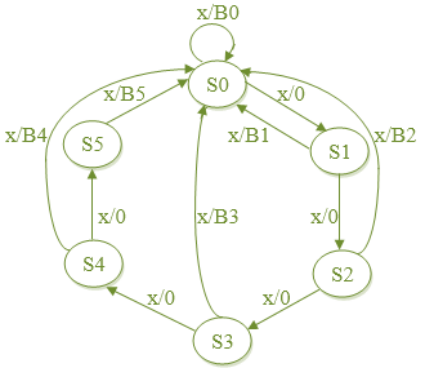

Figure 5. Finite State Machine for Decompression Architecture

Figure 5 shows the state diagram for the FSM used to classify sequence patterns. The FSM consists of six states in Figure 5. The process of State S0 is a 1-bit decoding word (i.e.) 1 or 0. The State S0 querS1 method is a 2-bit word decoding (i.e. 00, 01, 10, and 11). The state process S0 →S1→ is a 3-bit decoding word (i.e.) 000, 001 .... 111. Meanwhile, 111. The state process S0 →S1 → is a 4-bit decoding word (i.e. 0000, 0001 .... 1111. The State S0 →S2 →S3 →S4 process is a 5-bit word (i.e., 00000, 00001) decoding code. 11111. The State S0S1 → S3→S4 →S5 method is a 6-bit word decoding (i.e. 000000, 000001). 111111.

4. EXPERIMENTAL RESULTS

Using ISCAS benchmark circuit, the algorithm was analyzed. The tests of compression were obtained using the compression technique of variable to variable length. The result is compared to other compression techniques such as multi-stage encoding technique [24], Golomb [20], FDR [15], EFDR [24], 9C [22], VIHC [18], EVRL [26] to show the efficacy of the proposed technique. The compression ratio is calculated using the formula CR (percent) = ((TD-TE)/TE) * 100 where TD is the pre-computed test bits of certain benchmark circuits and TE is the encoded test data. Column 2 shows the compression ratio of the different benchmark circuits from Table 2. Compared to the original test vectors, the encoded bits are less. Therefore, the reduction of test data is accomplished through the use of the proposed algorithm. Note that, compared to the original test vectors; the encoded bits are smaller for all the benchmark circuit. The combination circuit of c2670 shows the highest 83.84 percentage. The average compression of different benchmark circuits is 79.80%. The compression ratio relation with other compression methods is shown in Table 3. The proposed algorithm shows a good compression ratio relative to other compression strategies from the analysis of Table 3[28, 29].

Table 2. Compression ratio for proposed technique

Circuit	Compression ratio	Size of TD (bits)	Size of TE (bits)	No. of bits for Mintest
c2670	83.84	20271	3276	10252
c7552	81.12	25254	4767	15111
s5378	78.25	23754	5167	20758
s9234	82.68	39273	6804	25935
s13207	75.25	165200	40885	163100
s15850	80.68	76986	14870	57434
s38417	76.93	164736	38010	113152
s38584	79.80	199104	40224	161040

Table 3. Comparison of compression ratio with other compression techniques

Circuit	Compression ratio (proposed)	Multistage encoding technique	Golomb	FDR	EFDR	9C	VIHC	EVRL
c2670	83.84	-	56.08	-	55.53	-	-	-
c7552	81.12	-	15.50	-	43.02	-	-	-
s5378	78.25	73.2	54.7	48.4	44.2	45.6	25.29	59.9
s9234	82.68	64.4	37.1	36.8	34.2	27.4	28.29	58.8
s13207	75.25	86	44.3	24.9	22.7	30.5	56.16	59.37
s15850	80.68	74.7	52.1	25	20.9	24.7	52.35	58.84
s38417	76.93	69.4	45.2	46.1	22.4	22.3	60.92	68.34
s38584	79.80	70.3	43.3	24.1	20	13.9	46.76	59.3
Average	79.82	74.2	43.53	34.3	32.86	22.9	44.96	60.76

5. CONCLUSION

Compression of test data is the best solution for minimizing larger volume of test data. This paper presents a new method of compression and decompression to test embedded cores in SOC. The proposed method is set to fixed length coding technique and this method proves to be an efficient method of testing data compression to save space and time for testing. The runs of 0's and runs of 1's will have different code word in this technique, so that the run type can be defined when decoding. The structure of the decompression is presented. This technique leads to decreased test data, saves the requirements of ATE memory and channel capacity. The ISCAS benchmark circuit experimental results show that the method is very efficient in reducing test data.

REFERENCES

[1] Rau JC, Wu PH, Li WL (2012) Test Slice Difference Technique for Low- Transition Test Data Compression. J Inform Sci Eng 15: 157-166.

[2] Wu HF, Cheng YS, Zhan WF, Cheng YF, Wu Q, et al. (2014) A Test Data Compression Scheme Based on Irrational Numbers Stored Coding. Scientific World Journal.

[3] Yeh PS (2002) Implementation of CCSDS lossless data compression for space and data archive applications. NASA/ Goddard space flight centre.

[4] Yamaguchi T, Tilgner M, Ishida M, Ha DS (1997) An Efficient method for compressing data. Int Test conf.

[5] Biswas NS, Das SR, Petriu EM (2014) On System-On-Chip Testing Using Hybrid Test Vector Compression. IEEE Trans Instrum Meas 63: 2611-2619.

[6] Yang JS, Lee J, Touba NA (2014) Utilizing ATE Vector Repeat with Linear Decompressor for Test Vector Compression. IEEE Trans Comput Aided Des Integr Circuits Sys 33: 1219-1230.

[7] Touba NA (2006) Survey of test vector compression techniques. IEEE Des Test Comput 23: 294 303.

[8] Mehta U, Dasgupta KS, Devashrayee NM (2009) Survey of Test Data Compression Techniques Emphasizing Code Based Schemes. 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools

[9] Jas A, Ghosh-Dastidar J, Touba NA (1999) Scan Vector Compression/ Decompression Using Statistical Coding. VLSI Test Symposium pp: 114-120.

[10] Jas A, Touba NA (1998) Test vector compression via cyclical scan chains and its application to testing core-based designs. Proceedings of the IEEE International Test Conference (ITC) pp: 458-464.

[11] Chandra, Chakrabarty K (2000) Test data compression for system-on-a-chip using Golomb codes. Proceedings of the 18th IEEE VLSI Test Symposium (VTS '00) pp: 113-120.

[12] Robert Theivadas J, Ranganathan V, Perinbam JRP (2016) System-on-Chip Test Data Compression based on Split-Data Variable Length (SDV) Code. Circuits and Sys 7: 1213-1223.

[13] Chandra A, Chakrabarty K (2001) Efficient test data compression and decompression for system-on-a-chip using internal scan chains and Golomb coding. Proceedings of the Conference on Design, Automation and Test in Europe, Munich.

[14] Li L, Chakrabarty K (2004) On using an exponential—Golomb codes and sub exponential codes for system-on-chip test data compression. J Electro Testing 20: 667-670.

[15] Chandra A, Chakrabarty K (2001) Frequency-directed run length (FDR) codes with application to system-on-a-chip test data compression. Proceedings of the 19th IEEE VLSI Test Symposium pp: 42-47.

[16] Chandra A, Chakrabarty K (2003) Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-directed Run-length (FDR) Codes. IEEE Trans Computer 52: 1076-1088.

[17] Li L, Chakrabathy K (2003) Test Data Compression Using Dictionaries with Fixed-Length Indices. Proceedings of the 21st IEEE VLSI Test Symposium (VTS'03) pp: 219-224.

[18] Gonciari PT, Al- Hashimi BM, Nicolici N (2003) Variable-length input Huffman coding for system-on-chip test. IEEE Trans Comput Aided Des Integr Circuits Sys 22: 783-796.

[19] Jas A, Ghosh- Dastidar J, Mom-Eng Ng, Touba NA (2003) An efficient test vector compression scheme using selective Huffman coding. IEEE Trans Comput Aided Des Integr Circuits Sys 22: 797-806.

[20] Chandra A, Chakrabarty K (2001) System-on-a-Chip Data Compression and Decompression Architecture Based on Golomb Code. IEEE Trans Comput Aided Des Integr Circuits Sys 20: 355-368.

[21] Mehta US, Dasgupta KS, Devashrayee NM (2010) Run-Length-Based Test Data Compression Techniques: How Far from Entropy and Power Bounds?-A Survey. Hindawi Publishing Corporation, VLSI Design.

[22] Tehranipoor M, Nourani M, Chakrabarty K (2005) Nine-coded compression technique for testing embedded cores in SoCs. IEEE Trans Very Large Scale Integrated Syst 13: 719-731.

[23] Tsai PC, Wang SJ, Lin CH, Yeh TH (2007) Test data compression for minimum test application time. J Inf Sci Eng pp: 1901-1909.

[24] Sivanantham S, Padmavathy M, Gopakumar G, Mallick PS, Perinbam JRP (2014) Enhancement of test data compression with multistage encoding. Integration VLSI J 47: 499-509.

[25] Bo Ye, Zhao Q, Zhou D, Wang X, Luo M (2011) Test data compression using alternating variable run-length code. Integration the VLSI Journal 44: 103-110.

[26] Robert Theivadas J, Ranganathan V (2014) Test Data Compression Using a New Scheme Based on Extended Variable Length Codes. World Appl Sci J 32: 2297-2302.

[27] Kavousianos X, Kalligeros E, Nikolos D (2008) Test Data Compression Based on Variable-to-Variable Huffman Encoding With Codeword Reusability. IEEE Trans Comput Aided Des Integr Circuits Sys 27: 1333-1338.

[28] Kalode P, Khandelwal R (2012) test data compression based on golomb coding and two-value golomb coding. Signal Image Process: Int J vol: 3.

[29] Luo Z, Li X, Li H, Yang S, Min Y (2002) Test Power Optimization Techniques for CMOS Circuits. Proceedings of the 11th Asian Test Symposium.

