Journal of
Electrical & Electronic Systems

Lakshmi et al., J Electr Electron Syst 2019, 8:1
DOI: 10.4172/2332-0796.1000300

Research Article Open Access

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

Lakshmi K, Robert Theivadas J* and Markkandan S

M.E VLSI Design, Anand Institute of Higher Technology, Chennai, Tamil Nadu, India

Abstract

The enhancement of integration capability in semiconductor technology requires a large amount of test data, resulting increase in memory, transition time and test time. In this paper, a novel lossless data compression technique is proposed to reduce test data, time and memory, based on variable to variable run length encoding scheme. In this scheme, a test data is partitioned into variable length test patterns and by applying compression algorithm, the bits are compressed into variable length codes. The encoding technique enhances the test data reduction with a limited number of code words. The compression technique is effective, especially when the runs of 0s and 1s in the test set are high and efficiently compress the data streams which is composed of runs of 0's and 1's. The variable to variable run length code algorithm is used to make changes in test vectors and adaptable for compressing precomputed test sets to test the embedded cores of System-on-chip (SOC). The decompression architecture for proposed technique was presented in this paper. Experimental results of ISCAS 85 and ISCAS 89 benchmark circuit's results in the significant reduction of test data with better compression ratio.

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

Introduction

Over the last two decades, the electronics industry had a phenomenal growth due to the advent of integration in semiconductor technology. Integrated circuits consist of different types of active and passive components, which is fabricated on a single chip to form systems. The different modules and intellectual property (IP) cores present in a system-on-chip (SOC) wraps different functions such as a processor, memory and different technologies such as CMOS logic to analog circuits. With the increase of technology, the large integrated devices are composed of millions of transistors and different hardware modules. With today's technology, the system-on-chip complexity is increased because the numerous components are implemented on a single chip.

However, the SOC presents many challenges such as increased test data, memory storage in automatic test equipment (ATE), transition time, test time, test power and test application time (TAT). Usually, the test data is generated and stored on workstations. Individual types of application-specific integrated circuit (ASIC) requires a more frequent download of test data from workstations to ATE. The Test sets for ASIC will be as large as gigabytes and the time taken to download the test data is more significant because the download takes from several minutes to hours. The Throughput of ATE is affected by download time of test data. To enhance the throughput of ATE, it is very important to reduce the download time of test data. A High volume of test data is directly proportional to higher transition time and memory. The Transfer of large test data between the ATE and chip is a bottleneck because of limited bandwidth, memory, and I/O channels [1].

The Limited bandwidth increases the test time and test cost during data transfer from ATE to the device under test (DUT). The Techniques widely used to reduce SOCs test data volume and test application time are Built-in self-test (BIST) and test-data compression.

When the test data increases, some of the challenges are as follows: Limited memory on ATE, Long upload time, Limited I/O bandwidth. Test data compression is a promising solution to store and transmits the compressed data from ATE to chip also it is used to speed up the interaction between ATE and SOC during the test. Data compression is a procedure of reducing the size of data file. The Use of test data compression is to compress the pre-computed test set (T_D) provided by the core vendor to a smaller test set (T_E) and then stored in automatic test equipment memory. Reduction of test data reduces the size of memory requirements in ATE, testing time, and test power [2]. A Novel test data compression is required to test the cores of SOC without exceeding limits of memory, bandwidth and power. Additional on-chip hardware is added before and after scan chains. An on-chip decoder decompresses the compressed test data from memory and delivers the original data to the device under test. The lossless test data compression reproduces all the bits, after decompression [3].

The Test vector compression consists of three categories as follows [4,5]: Code-based schemes use a data compression technique to encode the test cubes, Linear-decompression-based schemes decompress the data by using only linear operations [6], Broadcast-scan-based schemes broadcasts the same value to multiple scan chain. The Code-based scheme approach partitions the pre-computed test sets (T_D) into symbols and then replaces a symbol with codeword by applying a compression algorithm to form encoded data. The decompression is performed using a decoder, which simply converts each codeword into the original data. The structural information about circuit under tests (CUT) is not required in these methods but they are well suitable for IP core based SOCs. Based on this scheme, the different categories are described as follows [7,8]: Run-Length based codes, Dictionary based codes, Statistical codes [9].

***Corresponding author:** Robert Theivadas J, Associate Professor, Anand Institute of Higher Technology, Chennai, Tamil Nadu, India, Tel: 9840487566; E-mail: roberttheivadas@gmail.com

Received February 28, 2019; **Accepted** March 27, 2019; **Published** April 05, 2019

Citation: Lakshmi K, Robert Theivadas J, Markkandan S (2019) Variable-to-Variable Run Length Encoding Technique for Testing Low Power VLSI Circuits. J Electr Electron Syst 8: 300. doi: 10.4172/2332-0796.1000300

In the current generation, the Run-Length coded scheme is a very effective method for data compression for testing SOCs with a large number of IP cores. The Simple run length code scheme is used in [10], to encode runs of 0s into fixed length code words. The cyclical scan chain architecture is used to increase the frequency of 0s by allowing application of different vectors and reordering of test cubes. The difference vectors between test cubes are equal to XOR of two test cubes and encoded with a run-length code. The compression technique based on Golomb codes is proposed [11-13] to encode runs of 0s with variable-length code words. Here, each group consists of unique symbols for identification. The variable length code words are used for efficient encoding of longer runs of data. The Frequency-Directed Run-Length codes are similar to Golomb codes which are proposed in [14,15] and the difference in both methods is variable group size. FDR is a variable to variable length coded scheme and it is a method of mapping variable-length runs of 0s to variable length code words after the compression algorithm is applied. If runs of 1s are high on test sets, the FDR codes are not much effective. This coding scheme is efficient for few 1s and long runs of 0s for compressing data, but it is inefficient for data streams which consist of both runs of 0s and 1s. The on-chip decoder has to identify the prefix and tail to decompress FDR code. FDR requires a complex decoder with high area overhead. To overcome the complexity of the decoder in FDR, Huffman method and FDR is combined to use the variable length pattern as input to the Huffman algorithm instead of using fixed length pattern [16,17]. So, this retains the compression ratio due to FDR method and reduces the area overhead using selective Huffman. The 0s and 1s are filled in the place of X-bits to improve occurrence of frequencies of the blocks [18]. In the zero-fill algorithm is used to maximize the runs of 0s and it fills the 0s in the place of unspecified bits to reduce the scan-in test power [19]. The Extended Frequency Directed Run-Length Coding [EFDR] and Alternating Run-Length code shown in [7,20] describes that EFDR is suitable for test data streams consists of both runs of 0s followed by runs of 1s and vice versa. In EFDR method, the runs of 0s followed by 1 is encoded as in FDR, but the difference here is an extra bit is added at the beginning of FDR code word. The Alternating Run-Length code is a variable to variable length code and here, the test set consists of alternating runs of 0s and runs of 1s. By adding variable 'a' to the core, the runs of data can be identified. If a=0, the run-length is considered as runs of 0s, if a=1, the run-length is considered as runs of 1s. In [21] only nine code words are used to encode the test data and it is flexible coding technique. For each pattern, variable nine coded compression uses a variable length block to get a higher compression ratio. The Multistage encoding technique, namely alternating frequency-directed equal run-length (AFDER) and run-length based Huffman coding (RLHC) is proposed in [22,23] to reduce test data and test application time. The multistage encoding along with nine-coded compression technique improves the reduction of test data. The test data, scan power consumption, test application time (TAT) is reduced using a method named as alternating variable run-length codes (AVR) in [24]. A proper mapping of don't cares in test sets to 0s and 1s results in saving average and peak power consumption without slower scan-clocks. A Test data compression scheme based on the fixed to variable length coding with a limited number of code words is proposed using extended variable length codes [25] to reduce data, time, and memory.

The main objective in many of the code-based compression techniques is to reduce the test data volume without giving any importance to test power reductions, for example, the test compression techniques which is detailed in [19,20,22,26] has mainly focused on reduction of test data. The Test power as well as test data is reduced in some of the test independent compression techniques, for example, techniques detailed in [22,27].

Main contribution

In this paper, a new test data compression and decompression method has been presented based on variable to variable run length code for testing embedded processor core using pre-computed test sets. This method provides effective way to reduce the test data and memory required for automatic test equipment (ATE). The compression technique efficiently compresses the runs of 0's and 1's by applying an algorithm to each variable test pattern in order to enhance the compression ratio. Here, a different code word pattern is applied to runs of 0's which is followed 1 and also for runs of 1's which is followed by 0, in order to identify whether the run length pattern is either runs of 0's or runs of 1's. If the test pattern is runs of 0's followed by 1, then the compressed code starts with a bit '0'. If the test pattern is runs of 1's followed by 0, then the compressed code starts with a bit '1'.

This paper is organized as follows. Section 2 explains the detailed concept of variable to variable run length compression technique and describes the data compression procedure using an algorithm, flowchart and decompression. Section 3 describes the experimental results obtained for ISCAS benchmark circuits. Section 4 concludes the paper.

Variable to Variable Run Length Code

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

Variable to variable run length code

The Proposed code scheme is a method of mapping an input variable length test pattern (T_D) to variable length code words. The block diagram of variable to variable run length method is shown in Figure 1. Let T_D be the pre-computed test sets which are provided by core vendors.

Consider $T_D = \{t_1, t_2, t_3, \ldots t_n\}$, where n is the number of bits present in pre-computed test sets (T_D). Partition the test sets into variable run-length pattern and compress the test sets by applying code word. The code word will be different for both runs of 0's and runs of 1's.

Table 1 illustrates the encoding example of variable to variable run length coding scheme. This method consists of two types of run length pattern (i.e.) runs of 0's followed by 1 and runs of 1's followed by 0. For each type of runs, a code word is allocated separately. In each group, the code word for runs of 0's is the inverted data of the code word for runs of 1's. The code word present in each group determines a 2^n pattern, where n is 0, 1, 2, 3… In this method, as per test data, the runs of 0's were considered as strings of 0's followed by a bit '1' and the runs of 1's were considered as strings of 1's followed by a bit '0'. For example, 000001 and 00001 is a pattern of runs of 0's and its run length is 5, 4. 11111110 and 11110 is a pattern of runs of 1's and its run length is 7, 4. The start bit of code word identifies which type of runs has been processed and length of the code word is used to identify the group. The difference between the proposed method and other variable run length codes illustrates that, no prefix and tail is considered here, separate code words are assigned for both runs of 0's and 1's

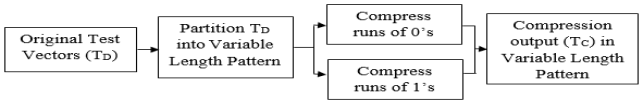

Figure 1: Block Diagram of Variable to Variable Run Length Coding.

Group	Run	Code Word Runs of 0's	Code Word Runs of 1's	Code Word Length
B0	1	0	1	1
B1	2 3	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	01111 … 00000	10000 … 11111	5
B5	32 … 63	011111 … 000000	100000 … 111111	6
…..	…..	…..	…..	…..

Table 1: Example of Proposed Coding Scheme.

From observation of Table 1, the test pattern is partitioned into runs of 0's followed by 1 and runs of 1's followed by 0. The test patterns are mapped to corresponding code word based on the type of run and number of repeated bits or run length. The size of the group is determined by the group number and the members of each group are equal to 2^m, where m is the group number, m = 0, 1, 2, 3... The bit size of the code word for both runs of 0's and 1's is equal to n.

A group B0 consists of 1 (2^0) member. Note that, the output of B0 will be $2^1(2^n)$ pattern (i.e.) 0 and 1. In B0, the test pattern of run length is 1 (viz) 01(runs of 0's) and 10 (runs of 1's). For 10, run length is 1 and the code word is 1. For 01, the run length is 1 and the code word is 0 which is inverted data of code word 1.

A group B1 consists of 2 (2^1) members. Note that, the output of B1 will be 2^2 patterns (i.e.) 00, 01, 10 and 11. In B1, the test pattern of run length is 2 (viz) 001 (runs of 0's) and 110 (runs of 1's). For 110, the run length is 2 and the code word is 10. For 001, the run length is 2 and the code word is 01 which is inverted data of code word 10. So for 110 and 001 pattern, the output is 10 and 01. If the test pattern of run length is 3 (viz) 0001 (runs of 0's) and 1110 (runs of 1's). For 1110, the run length is 3 and the code word is 11. For 0001, the run length is 3 and the code word is 00 which is inverted data of code word 11. So, for 1110 and 0001 pattern, the output is 11 and 00. The bit size for run length of 2 and 3 is same.

A group B2 consists of 4 (2^2) members. Note that, the output of B2 will be 2^3 patterns (i.e.) 000, 001, 010… 110, 111. In B2, the test pattern of run length is 4 (viz) 00001 (runs of 0's) and 11110 (runs of 1's). For 11110, the run length is 4 and the code word is 100.

For 00001, the run length is 4 and the code word is 011 which is inverted data of code word 100. So for 11110 and 00001 pattern, the output is 100 and 011.

If the test pattern of run length is 5 (viz) 000001 (runs of 0's) and 111110 (runs of 1's). For 111110, the run length is 5 and the code word is 101. For 000001, the run length is 5 and the code word is 010 which is inverted data of code word 101. So, for 111110 and 000001 pattern, the output is 101 and 010. For 1111110, the run length is 6 and the code word is 110. For 0000001, the run length is 6 and the code word is 001 which is inverted data of code word 110. So for 1111110 and 0000001 pattern, the output is 110 and 001. For 11111110, the run length is 7 and the code word is 111. For 00000001, the run length is 7 and the code word is 000 which is inverted data of code word 111. So for 11111110 and 00000001 pattern, the output is 111 and 000. The bit size for run length of 4, 5, 6 and 7 is same. This process is continued upto m group numbers. Here, a run of 0's and 1's is mapped to shorter code words in order to reduce test data.

Figure 2 shows the illustration of encoding example for variable to variable run length coding scheme. The test vector is considered as example from benchmark circuit. An algorithm is applied to each pattern (refer Table 1). From the example of encoding procedure, note that, the start bit of code word for runs of 1's is 1 and the start bit of code word for runs of 0's is 0. The original number of bits is 55 whereas the compressed bits is 28.

Data compression procedure using algorithm and flow chart

The Algorithm 1 and Figure 3 describe the process of compression algorithm using proposed coding scheme. If the test pattern is runs of 1's, then the run length of 1's is encoded as 2^n code word. If the test pattern is runs of 0's, then the run length of 0's is encoded as inverted data of 2^n code word. For 2^m run length pattern, 2^n code word is assigned as shown in case, where m = 0, 1, 2, 3... n = 1, 2, 3… For 2^0 run length pattern (01 or 10), 2^1 bit pattern is assigned as output. For 2^1 run length patterns (001 or 110, 0001 or 1110), 2^2 bit pattern is assigned as output. For 2^2 run length patterns (00001 or 11110, 000001 or 111110, 0000001 or 1111110, 00000001 or 11111110), 2^3 bit pattern is assigned as output. This is continued upto m patterns and n code word.

Algorithm 1: Variable to Variable Run Length Coding Algorithm

1. Generate pre-computed test vectors (T_D).
2. Let x be the input test vector, i be the start position and i+1 be the successive position.
3. Find the length of T_D.
4. Assign count = 0.
5. If x[i] == x[i+1], count the number of repeated bits (count = count + 1).
6. If test pattern is runs of 1's, assign count value.
7. If test pattern is runs of 0's, assign transition (0 as 1, 1 as 0) of count value.
8. Case:

i. If 1 ≤ count < 2, assign count as 2^1 code word.

ii. If 2 ≤ count < 4, assign count as 2^2 code word.

iii. If 4 ≤ count < 8, assign count as 2^3 code word.

iv. If 8 ≤ count < 16, assign count as 2^4 code word.

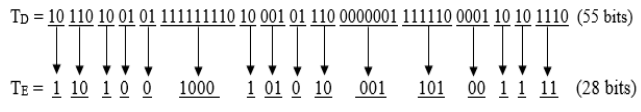

Figure 2: Example of encoding procedure of variable to variable run length code.

v. If $16 \leq count < 32$, assign count as 2^5 code word.

vi. If $32 \leq count < 64$, assign count as 2^6 code word.

9. Repeat the algorithm till end of data stream.

10. Calculate T_C = Total compressed bits.

Decompression architecture

Figure 4 shows the decompression architecture, which is used to decompress the encoded data. The decoder is simple and scalable. The architecture consists of finite state machine, counters and exclusive OR gates. The bit-in is the input to the FSM. When the decoder was ready, the enable signal is used to control the encoded data. The signal shift is used to control the codeword to shift in to the m-bit counter via ex-or gate. Signal dec is used to decrement the counter and rs is used to indicate reset state of the counter.

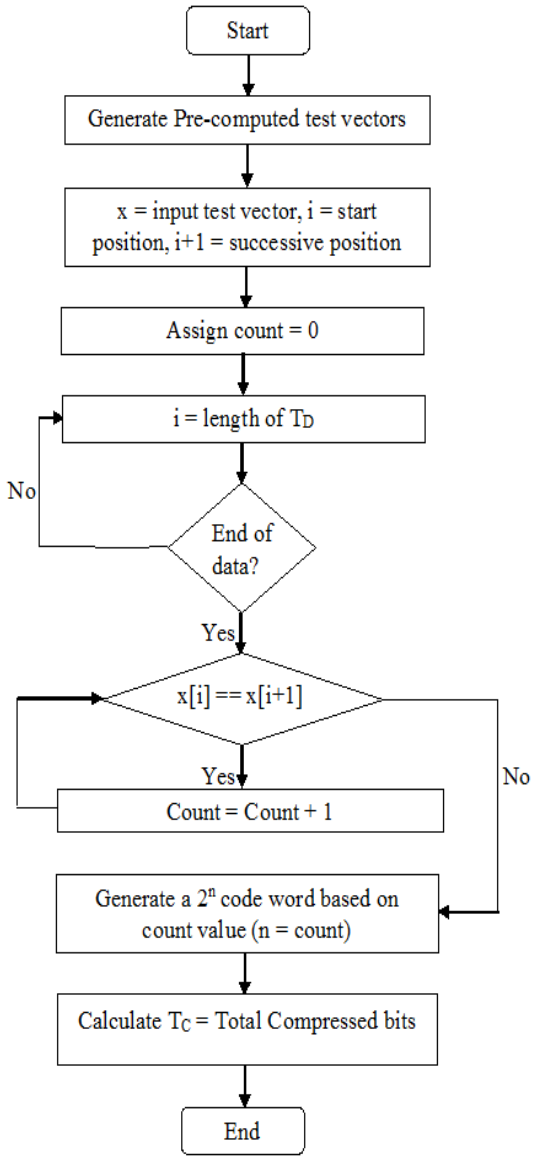

Figure 3: Flow Chart of Variable to Variable Run Length Coding.

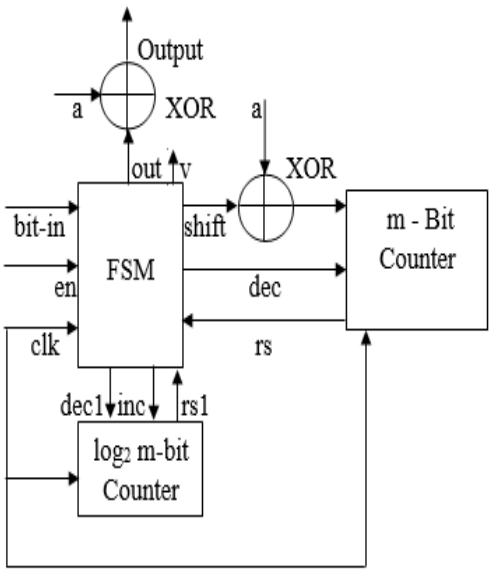

Figure 4: Conceptual architecture of Decompression.

The counter of $\log_2$ m-bits was used to count the length of the code word in order to decode the code word into run length pattern. The inc and dec1 is used to increment and decrement the counter and rs1 is used to indicate counting has finished. The FSM output signal out controls the ex-or gate and indicates if it finishes the decoding of runs of 1's. The signal v indicates the valid output. The sequence is detected using FSM and output of FSM will be code word. For run type 0's, the code word starts with a bit '0' and for run type 1's, the code word starts with a bit '1'. If bit-in is '0', the code word is a compressed code of run type 0's and if bit-in is '1', the code word is a compressed code of run type 1's.

The operation of the decoder is explained as follows:

Initially, signal en will be high and ready to receive data from bit-in. When bit-in input is 1, a will be 0 and if bit-in input is 0, a will be equal to 1. When the signal shift is high, the data fed to the counter, after the process of ex-or operation. If bit-in is 1, and a = 0, the code word of run type 1's does not get changed.

It remains as the original compressed code. For example, if compressed code is 1011 ex-or with 0. So, the output will be 1011. If bit-in is 0, and a = 1, the code word of run type 0's get changed. This is because to achieve the corresponding run length pattern.

For example, if compressed code is 0100, by using reference of compression algorithm, the output should be 11 zeros followed by 1. But, the length of 0100 is 4 and this is the inverted data of code word runs of 1's. So, the 0100 should be inverted again to reach correct output. For example, if compressed code is 0100 ex-or with 1, the output will be 1011. The correct run length pattern can be decoded. The shift, inc, en signal will be high, while the data fed to ex-or gate until the sequence detected code word is received. Then, the m-bit counter is decremented and allows signal dec goes high until rs was high. The signal v indicates a valid output.

The data from output of ex-or gate was shifted to m-bit counter until $\log_2$m-bit counter value was 0. The $\log_2$m-bit counter controls

the length of code word. Then, dec1 goes high and the counter was decremented. The signal rs1 went high, when the $\log_2$m-bit counter reaches the state 0. This indicates that the code word has been transferred to m-bit counter. The FSM outputs the 1's corresponding to code word and signal v indicates valid output. The signal out will be high when the m-bit counter value was 0.

When bit-in = 1 and a = 0, the data from the FSM output such as 11110 is ex-or with 0. So that, the output will be 11110. This indicates that the data decoded is run type 1's. When bit-in = 0 and a = 1, the data from the FSM output such as 11110 is ex-or with 1. So that, the output will be 00001. This indicates that the data decoded is run type 0's.

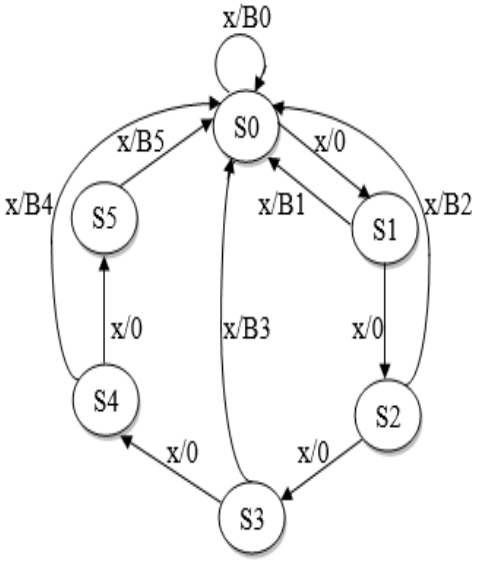

Figure 5: Finite State Machine for Decompression Architecture.

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

Experimental Results

The algorithm was analysed using ISCAS benchmark circuit. The compression results were obtained using variable to variable length compression technique. In order to prove the effectiveness of proposed technique, the result is compared with other compression techniques such as Multistage encoding technique [24], Golomb [20], FDR[15], EFDR[24], 9C[22], VIHC[18], EVRL[26]. The compression ratio is calculated using the formula CR (%) = $((T_D - T_E) / T_{E)} *100$ where T_D is the pre-computed test bits of given benchmark circuits and T_E is the encoded test data. From Table 2, the column 2 shows the compression ratio of the various benchmark circuits. The encoded bits are lesser when compared with original test vectors. So, by using proposed algorithm, the reduction of test data is achieved. Note that, from Table 2, the encoded bits are smaller for all the benchmark circuit compared with original test vectors. The c2670 combinational circuit shows the highest percentage 83.84%. The Average compression of various benchmark circuit obtained is 79.80%. Table 3 shows the comparison of compression ratio with other compression techniques. From the observation of Table 3, the proposed algorithm shows a good compression ratio by comparing with other compression techniques [28,29].

Circuit	Compression ratio	Size of T_D (bits)	Size of T_E (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 2: Compression ratio for proposed technique.

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

Table 3: Comparison of compression ratio with other compression techniques.

Conclusion

Test data compression is a best solution to reduce larger test data volume. A New compression and decompression method is presented in this paper for testing embedded cores in SOC. The proposed method is variable to variable length coding technique and this method proves that it is efficient compression method for test data in order to save memory and testing time. In this technique, the runs of 0's and runs of 1's will have different code word, so that while decoding the type of run can be identified. The decompression architecture is presented. This technique results in reduction test data, saves ATE memory and channel capacity requirements. Experimental results of ISCAS bench mark circuit's shows 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.