ISSN 2278 – 0211 (Online)

Generation of Test Data Compression and Decompression Using Efficient Bitmask and Dictionary Selection Method

S. Sivaganesan
Assistant Professor, Department of Electronics and Communication Engineering
Kalaignar Karunanidhi Institute of Technology, Coimbatore, Tamil Nadu, India
K. Nirmala
Assistant Professor, Department of Electronics and Communication Engineering
Kalaignar Karunanidhi Institute of Technology, Coimbatore, Tamil Nadu, India
S. Tamilselvan
Assistant Professor, Department of Electronics and Communication Engineering
Kalaignar Karunanidhi Institute of Technology, Coimbatore, Tamil Nadu, India

Abstract:
In higher order SOC (System On Chip) circuit, designs have led to drastic increase in test data volume. Larger test data size demands not only higher memory requirements, but also an increase in testing time. Test data compression addresses this problem by reducing the test data volume without affecting the overall system performance. In this, testable input data (test data) is generated by using Automatic test pattern generation (ATPG) then it is compressed and compressed data stored to memory. To test the particular circuit that time we will decompress the stored memory test data and then decompressed test data given to the Design Under Test (DUT). Finally DUT fault is tested and identified. It proposes a test compression technique using efficient dictionary selection and bitmask method to significantly reduce the testing time and memory requirements. This algorithm giving a best possible test compression of 92% when compared with other compression methods.

***Keywords**: Test data compression, Decompression, Dictionary selection, Bitmask, Run length, Golomb*

1. Introduction

In system on chip (SOC) designs, higher circuit densities have larger volume of test data. So in this larger test data applying in a particular circuit, the occupied memory volume and testing time consumption is very large. Test data compression method is reducing the testing time and memory requirements.

An Automatic Test pattern Generation (ATPG) is used to generate the Design Under test patterns. After then test data is compressed and the data will be stored to memory. When we want to test a circuit, that time, compressed data is immediately decompressed and given to Design Under Test (DUT), After words fault is tested and identified [1].

This paper deals with the comparison of the Bitmask and Dictionary selection method test data compression, Run length compression [6] and Golomb method test data compression [8] and then best compression method output is stored to memory. The bitmask and dictionary selection method [1][2] is a very efficient compression method and also fast compression, decompression mechanism.

In this compression technique, input test data is compared with created dictionary. When the input data is matched with dictionary, no need for bitmask operation, otherwise, it needs.

In this method of compression, data is stored to memory and then when we want to test a particular circuit that time immediately compressed data is decompressed and given to circuit (DUT) as shown in the fig.1.

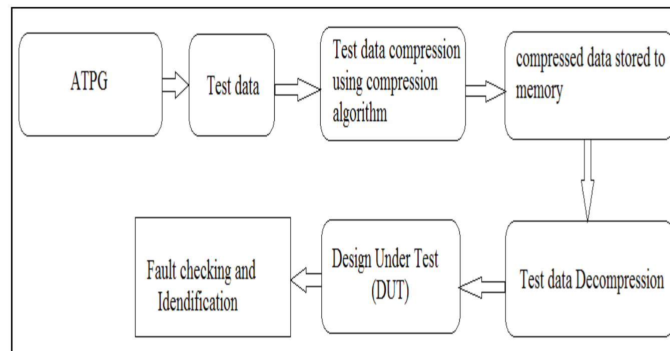
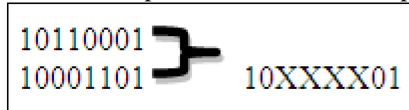


Figure 1: Block Diagram of Compression and Decompression

2. Automatic Test Pattern Generation (ATPG)

[1] Generates test pattern by using Pseudo Random ATPG method. Generated the test pattern for Magnitude comparator circuit, the magnitude comparator input data's are generated using LFSR (Linear Feedback Shift Register) method as shown in the fig.2. In this method, initially load the 8 bit value then 7th, 5th, 4th and 3rd bits are XOR-ed and its output is given to 0th bit, then bits are shifted in reverse direction. In this manner 2^n-1 = 255 input data (8 bit) are generated and this input is given to the magnitude comparator actual and fault(fault injected) circuit. The comparator outputs are compared and the mismatched output is identified. In this mismatched output of corresponding input data is called as "Test Pattern". Finally, the generated test patterns are combined to reduce the total test pattern width. For example,



The combined test pattern is two-two sets, it means P(1)&P(2), P(3)&P(4)………………

The generated test data's are compressed using Bit mask and Dictionary selection method. This method is efficient compare with other compression techniques.

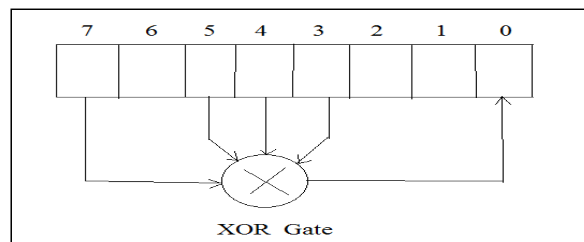


Figure 2: Block Diagram of ATPG (LFSR method)

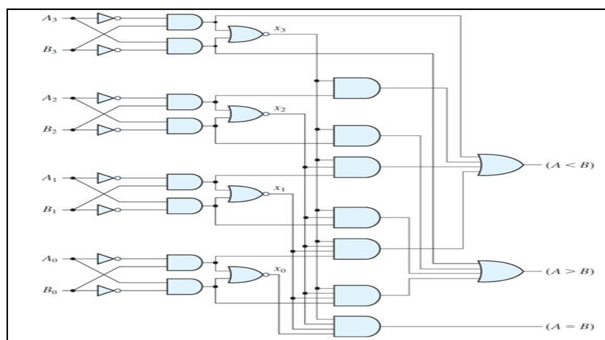


Figure 3: Magnitude Comparator

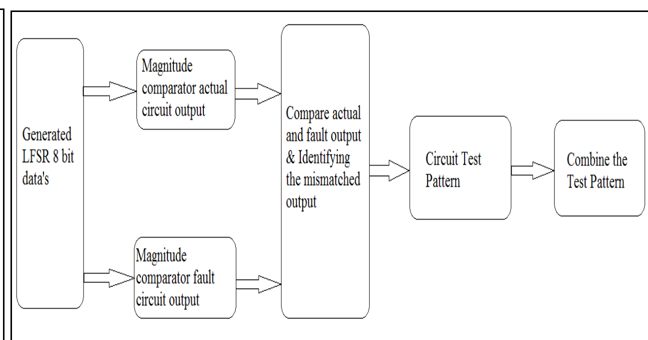


Figure 4: Block Diagram of ATPG

3. Run Length Test Data Compression

This is the one of the technique of test data compression. In this method, test data compression uses an algorithm "same value set of count and then represent the type of bit". This is the algorithm of output representation of Run length compression [6].

It means repeated same value total count and then represent the type of bit (character), but in this method is efficient if, there is a continuous repeated sequence and also if one or two bit changes in middle value [7]. For example,

If the Input is WWWWWWWWWWWWWBWWWWWWWWW WWWBBBWWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW,

Then the Compressed output will be

12W1B12W3B24W1B14W.
Suppose if we are using binary value, it will be represented as,
Test data Input : 11110000
Compressed output : 100 “1”100 “0”

Simulated output

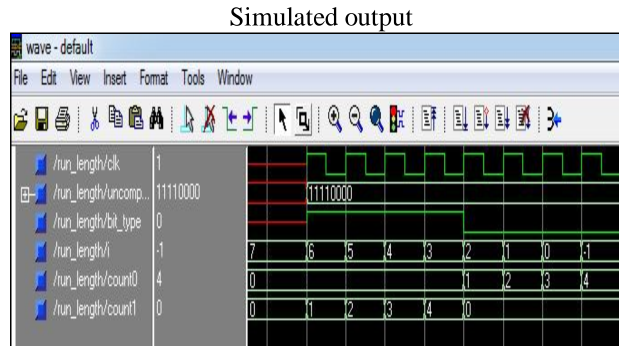


Figure 5: Continuous Sequence and its Output

Otherwise,
Test data Input : 00110101
Compressed output :
010 ‘0’010 ‘1’001 ‘0’001 ‘1’00‘0’001‘1’

Simulated output

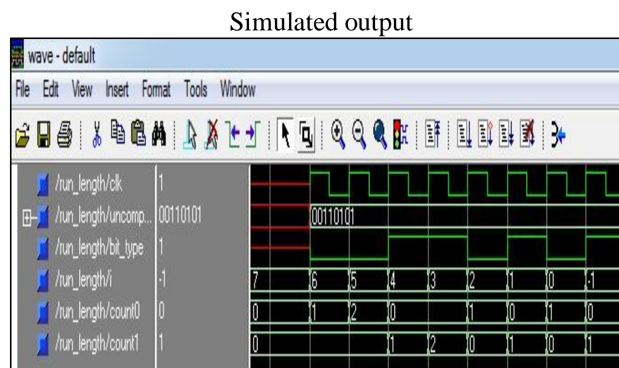


Figure 6: Continuous Changes in input and its Output

Actually Total number of bits in input is ‘8’ but a total bit in compressed output is ‘24’. So, this method is not efficient because in this method compressed output value strength (number of bits) exceeds the inputs. So the run length test data compression is not efficient.

4. Golomb Method Test Data Compression

Golomb coding is the one of the test data compression and it is a optimal prefix code, making Golomb coding highly suitable for situations in which the occurrence of small values in the input stream is significantly more likely than large values [9].

It is totally different from the run length test data compression method. This method uses splited groups and it is having group prefixes. In this each group prefix having increased value (0, 10,110) and also uses number of repeated occurrence. So it is named as run length. Each group of run length binary count named as tail. The codeword depend upon above causes because, each group splitting the run length wise then, each group and run length wise writing the group prefix then, each and every group, run length, group prefix having separate tail count values finally each group prefix and tail count are joined [8] so, they will get the code word of each run length.

The table I shows the code word of golomb method test data compression. Here if the test data is single bit, it is represented run length is “0” & codeword is “000” and if continuous two bits are same it is represented run length is “1” & codeword is “001” so same procedure is followed by the run length of 2 to 11 and it’s codeword shown in the table. For example,
Input : 00001111
Compressed output : 011011
In this example, input value is “00001111” here continuous repeated total number of 1’s (last four) count is “4” so it is mentioned codeword is “011” (because it is run length is 3) then input comes continuous repeated total number of 0’s (first four) count is “4” so it is mentioned also “011” (run length is 3) finally repeated 1’s and 0’s compressed output is “011011”. The fig.7 is shows the simulated output.

Group	Run length	Group prefix	Tail	Code word
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

Table 1: Golomb Method Test Data Compression Codeword

Simulated output

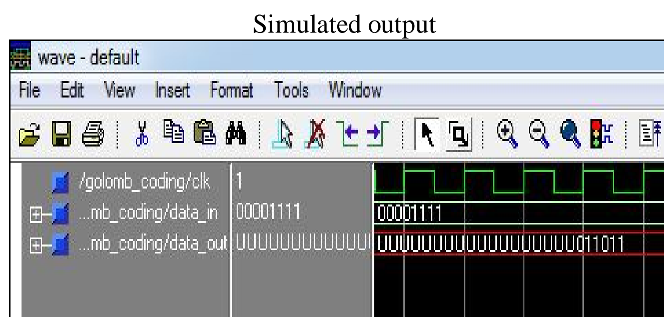


Figure 7: Continuous Repeated Input and Its Output

If the input value is continuously repeated (i.e. 00001111), then this method is efficient; otherwise it is not efficient. For example,
Input : 01010101
Compressed output: 000000000000000000000000
Simulated output:
Here the input is 8 bits (01010101) but compressed output total bits are 24. Actually compressed output total bit exceeds the input total bits. So this method is not efficient.

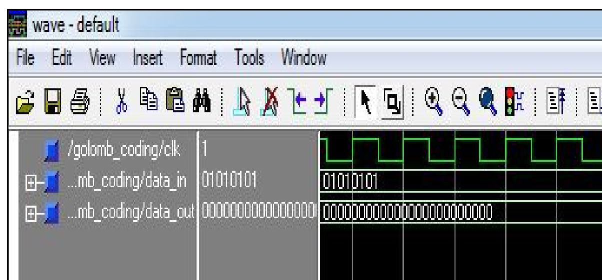


Figure 8: Continuously Changing Input Sequence and Its Output

5. Dictionary Selection and Bitmask Method Test Data Compression

This is the one of the test data compression method. Here the first step is to create the dictionary depending upon the test data input value (LZSS method) [1]. It means that all the input data sets are compared and most matching and most similarity 8 bit data is taken and kept in a content box. The dictionary having '0' and '1' index and corresponding 8 bits content value is as shown in the fig.9.

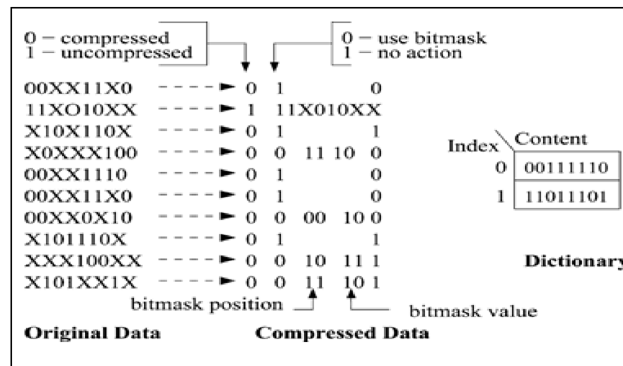


Figure 9: Dictionary Selection and Bitmask Method Test Data Compression

Then compare the input test data value (original data) with dictionary value [1] [2] [3]. If the input is matched with dictionary that is either with the index '0' content or with the index '1' then it is said to be compressed (0) that is the 1st position of compressed data. If it is not matched it is said to be uncompressed (1). That is, the input value is written as it is. Otherwise, the matched index value is written in the 5th position of compressed data. If the input test data is matched in dictionary but when one or two bits are differ then we make use of mask operation. When we use mask operation we write '0' in the 2nd position. Then split the input data in order to identify the bit change and in this splitting of each data assumed positions are 00, 01, 10, 11 then compare the input data with the dictionary. Since the mismatch of the bit is identified then write the corresponding mismatched position of the original data at its 3rd position.
Then finally the mismatched dictionary data and the input test data values perform EX-OR operation and placed in the 4th position of the compressed data i.e. bitmask value [1]. This is the method of dictionary selection and bitmask method test data compression and it is giving a high compressed output. For this method, test data compression simulated output is shown in the fig.10.

Simulated output

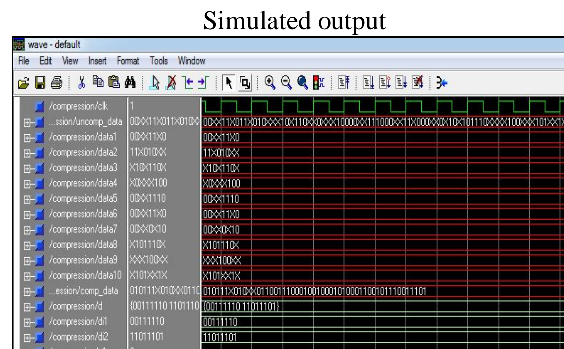


Figure 10: Dictionary Selection and Bitmask Method Test Data Compression Simulation Result

6. Comparison of Compression Methods

The table II shows the comparison of compression methods. The run length compression method gives 60% compression result and golomb gives 65% compression result but dictionary selection and bitmask method compression gives 92% compression result.

S. No	COMPRESSION METHODS	COMPRESSION PERCENTAGE (%)
1	Run Length Compression	60
2	Golomb method compression	65
3	Dictionary Selection and Bitmask Method Compression	92

Table 2

The dictionary selection and bitmask method of test data compression gives the high compressed output when compared with Run length compression and Golomb compression. So it is the best method of test data compression. At the same time, it is the fast and easy compression method.

7. Compressed Data Stored to Memory

After test data compression using dictionary selection and Bitmask method, the compressed data is stored to memory [1] [2]. The memory is created by using array format of VHDL program then compressed data is stored in this memory.

```
type memory is array(0 to 1) of std_logic_vector(51 downto 0);
signal m : memory:=("010111X010XX0110011100010010001010001100101110011101",
                    "0000000000000000000000000000000000000000000000000000");
```

Figure 11: Compressed Data Stored To Memory Using

Array Format Of VHDL Program

The fig.11 gives VHDL program how the compressed data is directly stored to memory using array format.

8. Test Data Decompression

In test data decompression when we want to test circuits (DUT) that time immediately decompress the data from memory. In this purpose used memory READ and READY signals.

The Decompression logic is shown in fig.12. In Decompression logic, the compressed data is checked whether the first two bit are ‘01’ or ‘00’or ‘1’ from memory [1] [2]. Here if the first two bits are ‘01’, Data is compressed without bitmask operation. So the index value is read. Directly taking the dictionary index corresponding 8 bit data is given to output buffer. When the first two bits are ‘00’, Data is compressed with bitmask operation. So it refers next two bits (i.e. bitmask position) and goes to next mentioned location. It EX-ORs the bitmask value with given corresponding dictionary indexed value to get the original uncompressed data. It is given to output buffer [1] [3].

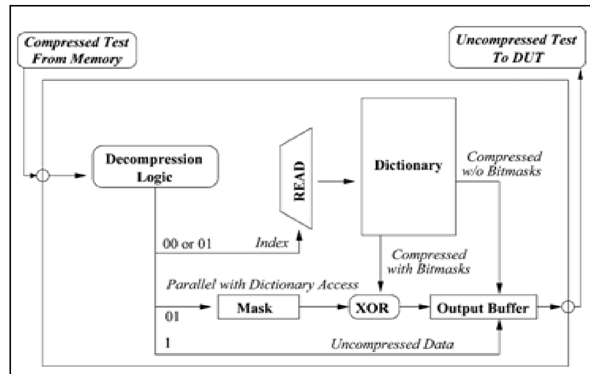


Figure 12: Decompression Logic

Otherwise, when the first bit is ‘1’, then the data is uncompressed and it is directly given to output buffer. Finally we get the decompression methodology. Output is exactly equal to the original uncompressed data.

Simulated output

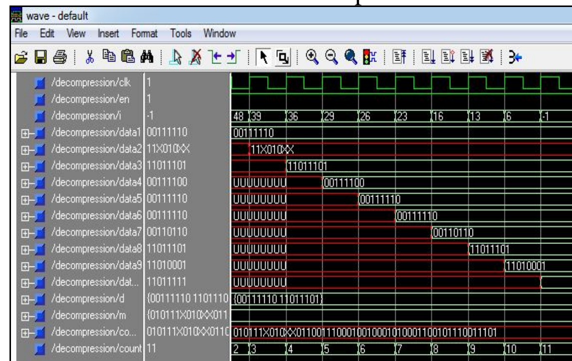


Figure 13: Decompressed Output

9. Fault Analysis

Here test data is compressed using efficient dictionary selection and bitmask method and stored to memory. When we want to test a circuit that time compressed data immediately decompressed and given to design under test (DUT). In future, the test data will be generated using Automatic Test Pattern Generation method (ATPG).

It means that an automatic test pattern generation will generate the test data by its own depending upon the circuit. Then the test data will be given to test data compression operation and stored to memory and then the decompression operation. Then the circuit(DUT) will be tested and fault will be identified. The faults are,

- Stuck at faults,
- Delay faults,
- Short circuit and open circuit faults,
- Bridge faults.

These faults to be tested and identified.

Stuck at 1 Fault: Whatever input is given to the gate, output is "1" it is called stuck at 1 fault.

Stuck at 0 Fault: Whatever input is given to the gate, output is "0" it is called stuck at 0 fault**.**

Short & Open Circuit Fault and Delay Fault: In a circuit any two or more data lines are short circuited it is called as short circuit fault. In a circuit any one data line was opened it is called as open circuit fault. The circuit output is getting exactly but after some time (delay –After some ns) it is called as delay fault.

Bridge Fault: The nearest two or more circuit lines creates the attracted function. It creates some resistive function between the nearest circuit lines. The resistive function operate highest value is forced (applied) to nearest circuit lines it is called bridge fault.

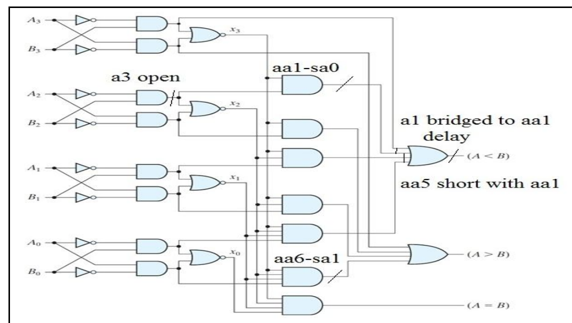


Figure 14: Fault injected Magnitude Comparator circuit

Simulated output

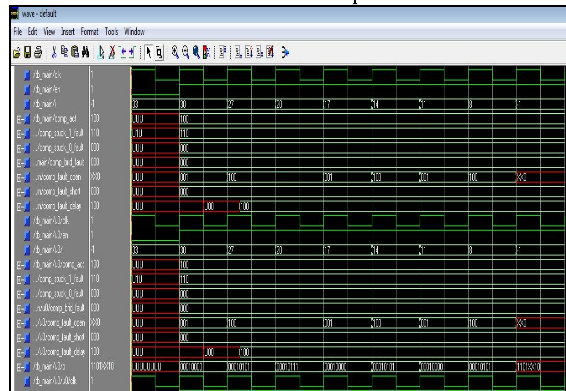


Figure 15: Fault analyzed simulated output

10. Future Plan

In future of this paper after completion of fault indentification step to check the fault coverage and correct the identified faut in Design Under Test (DUT).

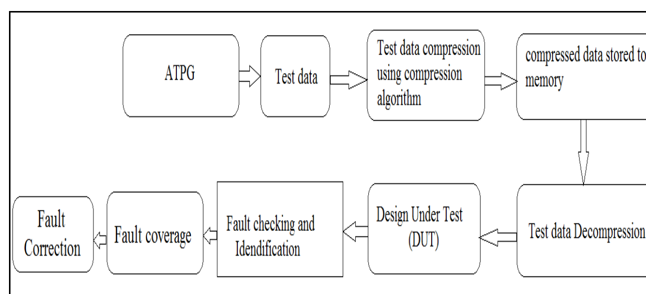


Figure 16: Future plan-Fault coverage and Fault correction

11. Conclusion

A test data compression algorithm that combines the advantages of dictionary based compression and bitmask based compression. This paper developed efficient bitmask and dictionary selection techniques for test data compression in order to create maximum matching patterns in the presence of don't cares. Our test compression technique used the dictionary and bitmask selection methods to significantly reduce the testing time and memory requirements. The test data compressed using dictionary selection and Bitmask method is reduced the maximum bit volume that same time without affecting the test data overall performance and stored memory content decompressed at anytime immediately. The test data compression and decompression done is very fast. So this compression and decompression method reduces the higher order SOC circuit testing time consumption and occupied memory volume. At the same time, the compression efficiency is 92%.

12. References

1. Kanad Basu and Prabhat Mishra, "Test Data Compression using Efficient Bitmask and Dictionary Selection Methods", IEEE Transaction on Very Large Scale Integration(VLSI) Systems, Vol. 18, NO. 9, September 2010.
2. K. Basuand P. Mishra, "A novel test data compression technique using application aware bitmask and dictionary selection methods", in Proc.ACM Great Lakes Symp. VLSI, 2008, pp. 83–88.
3. S. Seong and P. Mishra, "Bitmask based code compression for Embedded systems", IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 27, no. 4, pp. 673–685, Apr. 2008.
4. F. Wolffand C. Papachristou, "Multiscan based test compression and hardware decompression using LZ77", in Proc. Int. Test Conf., 2002, pp.331–339.
5. A. Wurtenberger, C. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections", in Proc. Int. Test Conf., 2004, pp. 926–935.
6. A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed Run length (FDR) codes", IEEE Trans. Computers, vol.52, no.8, pp.1076–1088, Aug.2003.
7. N. Ranganathan, S. Henriques, "High-Speed VLSI Designs for Lempel-Ziv-Based Data Compression", Trans. on Circuits & Systems- Analog and Digital DSP, Vol. 40,No. 2, Feb. 1993.
8. G. H. H'ng, M. F. M. Salleh and Z. A. Halim, "Golomb Coding Implementation in FPGA" , Faculty of Electrical Engineering University Teknologi Malaysia VOL. 10, NO. 2, 2008, 36-40.
9. T. Silva, et. al., "FPGA based design of CAVLC and Exp-Golomb coders for H.264/AVC baseline entropy coding", Proc 3rd IEEE Southern Conference on Programmable Logic, pp. 161-166, Feb 2007.
10. A. Chandra and K. Chakrabarty, "System on-a-chip test data compression and decompression Architectures based on Golomb codes," IEEE Trans. Computer-Aided Design, vol. 20, pp.355-368, Mar.2001.
11. A. Chandra and K. Chakrabarty, "Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding", IEEE Trans. Computer-Aided Design, vol. 21, pp. 715-722, June 2002.
12. L. Li, K. Chakrabarty, and N. Touba, "Test data compression using dictionaries with selective entries and fixed-length indice", ACM Trans. Des. Autom. Electron.syst.vol.8, no.4, pp.470-490, 2003.
13. I.HamzaogluandJ.Patel, "Test set compaction algorithm for combinational circuits", in Proc. Int. Conf. Comput. - Aided Des., 1998, pp. 283-289