A MODIFIED COMPRESSION TECHNIQUE FOR TRACE DATA USING DICTIONARY METHOD

R.AGALYA[#1], S.SARAVANAN[#2]
[#] School of Computing, SASTRA UNIVERSITY,
Thirumalaisamudram, India.
[1] agalya@sastra.ac.in
[2] saran@core.sastra.edu

Abstract –In VLSI design circuits, System-On-Chip consumes more area due to huge data volume in the hardware. In order to reduce that constraint, the size of the trace buffer should be maintained constant. The trace data should be compressed to acquire huge amount of data. To compress those data several conventional techniques has been followed. Here in the proposed system, an approach called Look-Up-Table based dictionary compression method for System-On-Chip is used. It implements dictionary in such way that the most of the test vector frequencies are calculated which is to compress the test data in an efficient manner. This compression technique holds three different test vectors of all zero's, one's and X's value where most of benchmark circuitry frequency can be captured. This technique is implemented in those circuits using hardware description language and it is verified by XILINX Spartan-3E to know utilization of devices. Hence, it achieves more compression ratio with less area overhead without loss of data.

Key words-System-On-Chip, Look-Up-Table, Dictionary, Compression and area overhead

I. INTRODUCTION

To certify that the complexity of the design increases, System-On-Chip (SOC) needs a verification process before the chip is released to market. For this, a variety of checkpoints are forces at various stages of implementation. Time-to-market is the major demand now-a-days. It's not possible to spend more time in verification process. It should be done as soon as possible with high quality. Therefore, Silicon debug has come out as a most powerful technique for detecting and locating the design bugs in recent years. The limited observability of internal signals is the major key issue of silicon debug during manufacturing and the restorability of signals. Embedded logic analyzer is the Design-For-Debug (DFD) hardware. On-chip buffers are used to improve the observability of these signals states during runtime. The obtained data are transferred via low bandwidth device pins in which the post-processing algorithms were used to analyze and identify the design bugs off-chip. In Silicon debug, there are two methods for verification. One such method is Pre-silicon debug and the other is Post-silicon debug. In earlier days, Pre-silicon verification was the most popular method to identify and detect the design bugs when the chip is fabricated. This verification method has two approaches. It may be either formal verification or extensive simulation for the designers to check the functionality against its specification. In earlier stages, the physical parameters cannot be represented correctly. To detect the defects like opens and shorts, the manufacturing test is applied to the fabrication of each circuit [1]. The testing techniques of manufacturing are used to detect only manufacturing defects. As discussed earlier, because of the rising complexity in IC's, to validate the circuit thoroughly will take more time in pre-silicon phase is unbearable. And also, it is inadequate to see the accuracy when the first silicon is available. To avoid the increasing cost of mask and also to reduce time-to-market, many bugs are escaped or undetected in this phase. Hence, the pre-silicon validation is failed to detect and fix the bugs.

The design bugs are captured by one of the debugging techniques of silicon debug technique called post-silicon validation even after the chip is fabricated. It is not possible to store the internal values of each and every signal. To observe some internal signals, several techniques have been proposed [2-4]. During execution, trace buffers are used to capture the states of internal signals. At the time of debug, the content of the trace buffer is analysed to find the possible errors. As mentioned earlier, the trace buffer size should be maintained constant. So as to keep its size constant, the amounts of data which are stored in the trace buffer are to be compressed. For this several compression techniques have been followed. An efficient compression technique has to be taken without having loss of data. The compression should be efficiently high and the speed of the compression is more. A small area overhead is considerable.

Different compression techniques were proposed on the depth of the trace buffer [5-6] and its width [7]. Depth compression deals with the selection of cycles in which the data are erroneous. The major problem of depth compression algorithms in above papers are: rerunning same set of test controls on the same system which

produces same information. The output of the system is not correct because of its repeatable condition. Instead of that, the optimized trace data which are generated can be re-run. The compression of width runs just once without regenerating the trace data. Usually, the compression technique has done based on three methods. In the proposed work, the trace data are compressed using LUT based compression technique which holds three different vectors where most of the circuit frequencies are captured. It achieves high compression rate without loss of data.

Section II shows the reassessing of prior works. Section III and IV shows the proposed work and experimental results respectively followed by conclusion.

II. PRIOR WORKS

The primary goal of silicon debug technique is to achieve the limited observability of internal signals. Formerly the signal values are identified, it is examined by the algorithms such as failure propagation tracing for recognizing the bugs in the circuits [8]. To attain the real time observabilty, chip pins are used in [9] to observe those internal signals. But it is difficult to implement in the limitation of chip pins and in the occurrence of high frequency internal clocks. Consequently, this was overcome by using trace buffer techniques [10]. As discussed in introduction, Embedded Logic Analyzer (ELA) is a design for debug technique to obtain the signal samples and keep them in on-chip buffers. The data are loaded to the processor from the trace buffer through low bandwidth interface which examines to locate the bugs in circuit. It is healthier to keep the size of the trace buffer constant for reducing the overall cost, area and its energy needs when debugging. When the amount of data increases which are to be stored in the trace buffer are needed to be compressed. Several techniques have been proposed for data compression from [5-7]. These techniques are used to compress the data before storing it in trace buffer and it should provide a good compression with less area overhead.

Lossy compression techniques are of four types. It is based on variable or fixed sizes. In [11] the Frequency-Directed Run-Length (FDR) codes achieves good compression ratio. Papers [4] and [12] proposed a compression technique for increasing the observation window with small area overhead. Without increasing the chip area, it needs to dig out more data from the debug sessions by compressing the width and the depth of the trace buffer. These two techniques have been used for compressing the debug data before it is being stored in the tracefer. In [12], the compression ratio is up to 4X and improves the FDR with a small area overhead compared to [3]. Usually, dictionary based techniques consumes more area. But the paper [4] proposed some dictionary based techniques takes the account on the fact of difference between erroneous and golden trace data are small. It gives better compression than the dynamic dictionary method of [7] and reduces it hardware size by 84% with 60% better compression ratio.

The compression technique may be of either statistical or dictionary based algorithms. Usually, statistical algorithms are complex and it can provide best compression. This leads to more area overhead. Statistical may be of either arithmetic or Huffman coding. This Huffman coding leads to get the optimal code length and therefore it achieves a good compression ratio. In that, variable length coding is used to reinstate the common symbols with shorter code. Static implementation of these methods needs to scan the same data two times. For the real-time requirement, this method is not apt for embedded logic analysis (ELA) and also the area will be more when embedded into an on-chip debug module.

Another technique is known as dictionary based algorithm, it will give acceptable area overhead but the compression is sacrificial. With the acceptable area, this dictionary based algorithms are improved to get better compression ratio. In [13], it can be used to improve with the help of bitmask for code compression. For test compression, bitmask based compression were used. The dictionary based compression algorithms such as BSTW, Lempel-Ziv (LZ77) [14] were proposed. This dictionary based algorithm can attain competitive compression ratio than the statistical algorithms. While implementing in hardware, these methods can achieve higher throughput and compression ratio [14-17]. The area overhead is still large but it is acceptable for ELA. When it is constructed statically achieves real-time compression with good compression ratio and less area overhead than the dynamic method in existing methods. In the existing methods, it was explored three standard dictionary based compression. In the proposed work, look up table based dictionary selection compression technique is used to get better compression ratio with less area overhead.

III. PROPOSED WORK

The compression technique is used to compress the trace data during execution without loss of data. It is classified into either dictionary coding algorithm or statistical coding algorithm. In the proposed work, the dictionary coding technique has taken based on Look-Up-Table (LUT).

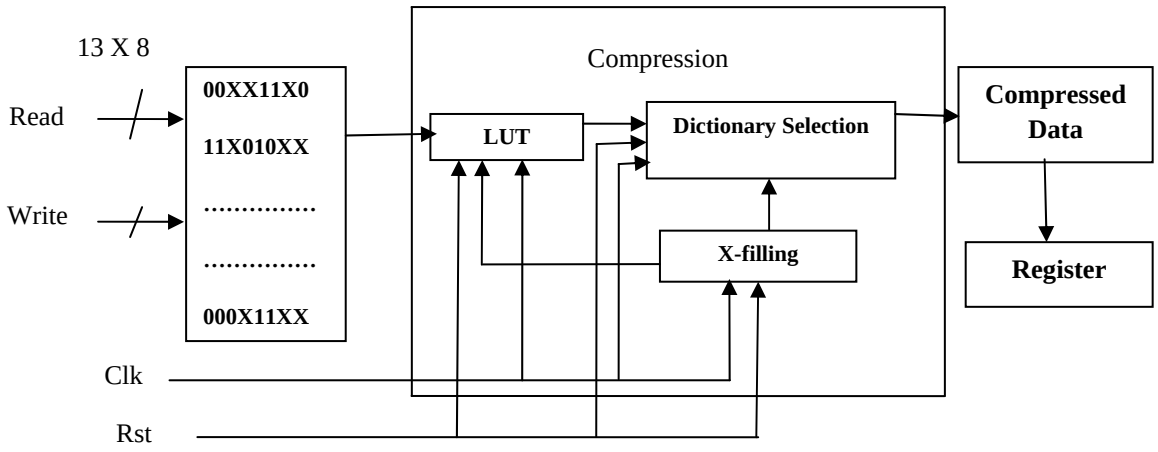

Fig.1. Proposed trace data compression method

Fig. 1 shows an impression of proposed compression technique. The test vectors of benchmark circuits are taken as input. For example, it contains 13x8 as input. Those input files are taken from the memory and it gives to the compression block. For the observability of internal signals, the compression techniques have been followed to store more number of data. The compression technique used in this system contains LUT, dictionary selection and X-filling blocks to compress the given data. Those blocks are described in the following subsections. After filling up the data in X-filling block, it is stored in LUT for finding the repeated number of data bits. By using the dictionary selection method, the given data are compressed. These compressed data can be used for the silicon debug applications.

Initially the input file has taken to read the data. Set the reset and rising edge of clk pins to 1 for enabling the write mode to change the location of input file. The input file is stored in the temporary register or memory. After this, set lut to 1. Usually the trace signals require 0's and 1's value. Here in the proposed compression technique, the inputs may have X's value. It may be used for trace buffer [18-19] to improve their observability with large number of data. Keep the test vectors which is having all 0's, all 1's and all X's in the LUT's at three different address locations. E.g LUT (0) is assigned with 'X' value. If the input is having all 0's then it is compressed to 0. For e.g one of the inputs of 8 bit is having all 0's 0r 1's or X's values then those bits are compressed to a single 0 or 1 or X respectively. These can be done by using LUT. After the LUT process, the data or signal should be selected via dictionary selection method. It is done by following methods. The following example shows how the LUT will perform.

1. *When LUT is equal to temp then*
2. *Select any of the cases of X or 0 or 1*
3. *If it is other than that set null*

A. X-filling

Initialize the operation of X-filling. The function will check the input values whether it is having 'X' value or not. If it is found 'X', then it is filled by the either the value of 1 or 0. It is done for throughout the operation. The subsequent steps can be taken for the process. For example, 1000XX01 will become 10001101 if it is filled by the value 1.This X-filling can be applicable to LUT also. If it so, it will give better compression ratio. The steps involved are:

1. *if X is found in the given array then*
2. *Replace it with 1or 0*
3. *Else it remains same*

B. Dictionary Selection

The dictionary can be selected by the following methods. Initially, the frequency of the test vectors should be calculated. The patterns which are having more number of frequencies or repetition will be considered for the calculation. For example, 1000XX01 is repeated more number of times than the other vectors, it will be given as the most priority. The highest frequency vector is set at first. After finding all the frequencies, those vectors are arranged in the priority order for the selection of signals. For example, if the highest frequency of the given test vector is 4, it will be stored in the first location of array. It can be done by the subsequent steps.

1. *if temp_array(i) is greater than temp_array(j) then*
2. *temp_array(i) is stored in any other temp_array*
3. *temp_array(j) is stored in temp_array(i)*

4. end if

For example, while selecting the dictionaries, the number of frequencies is noted as 4. At first, select the first two greatest values of frequency for dictionary selection. Search for the match with that frequency. If it matches with that then store it in the dictionary register. Then search for the next vector. X-filling can be done after this or before this. If it is done before calculating the number of frequencies, it can achieve higher compression ratio. For e.g., if the temp_array(i) is greater than the temp_array(j), i^{th} location is stored in any other temp_array. Here i and j comparison shows the frequency comparison.

C. Compression

While compressing the test vectors, it may be any one of the following combinations. It may be both are 1 or 0 or X, 0 and X, etc. By this way the compressed bits are stored in the register. For ex., comp_dic (addr1) = (compressed &No_action& '0'). Here, the expression states that the 'compressed' indicates the compression data; 'No_action' indicates it is not present in the look-up-table and '0' can be replaced by 1. If the test vector doesn't have frequency, those vectors are referred to as uncompressed bits. From this method, can find the compressed and uncompressed data bits. The compression ratio will be more by using this LUT based compression method and its experimental results were shown in section IV.

IV. EXPERIMENTAL RESULTS

The compression technique is implemented using hardware description language and verified by XILINX Spartan-3E for the area utilization. In the existing methods, it was verified by using Object Oriented Programming Language. The experimental result shows that the functional verification waveform & its synthesis output of LUT based compression technique. In fig. 2, it shows the functional verification of 13 inputs. In that, red mark indicates compressed and uncompressed data.

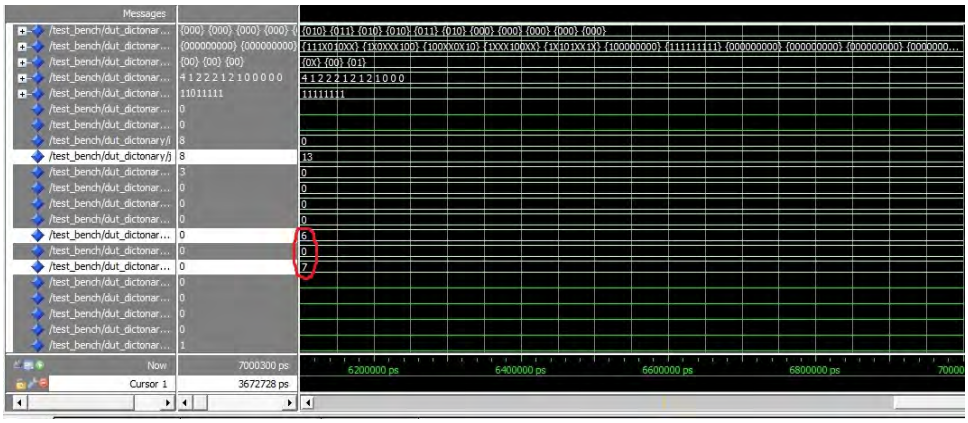

Fig. 2. Functional verification of given input

Table I and II shows that synthesis results which are obtained from XILINX Spartan-3E. The number of utilization percentage is shown in Table I to know how much area it can be occupied and timing reports were shown in Table II. In the proposed work, the utilization of the devices is acceptable and achieves good compression ratio by the following expression. Table III shows the compression ratio of various benchmark circuits of ISCAS'89 from the proposed work. The compression ratio can be calculated by the following means.

Compression Ratio = Uncompressed Size of the Data / Compressed Data Size

TABLE I
Synthesis results obtained from Spartan-3E

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1356	4656	29%
No. of Slice Flip Flops	122	9312	1%
No. of 4-input LUTs	2657	9312	28%
No. of bonded IOBs	10	232	4%
No. of GCLKs	1	24	4%

TABLE III
Timing Report

Min. Period	212.449ns
Max. Frequency	4.707MHz
Min. input arrival time before clock	9.896ns
Max. output required time after clock	4.310ns

TABLE IIIII
Compression ratio of various circuits

Circuit Name	Compression Ratio
s298	77%
s382	87%
s713	97%

The uncompressed data size and the compressed data size represents that, the test vectors which are not compressed from the given test vectors to the compressed vectors. When the number of inputs is huge, the compression ratio will increase. This LUT based dictionary selection method gives more compression when we go for larger circuits which consumes huge amount of data. Retrieving the exact data can be identified by comparing the inputs with the same area. It will be used for any applications which need to be compress.

V. CONCLUSION

Earlier, the existing methods lead to more area overhead due to more number of data. In silicon debug techniques, it needs to trace more signals with more data. For that, this technique needs to dig out more data. The trace buffers are used to store the trace data which are obtained from the silicon debugging techniques. With the limited size of trace buffer, we cannot keep more data in that. For that, several compression techniques have been followed to improve the compression rate and area overhead. In this proposed work, Look Up Table based dictionary method was constructed and it achieves more compression ratio with high quality and less area overhead.

REFERENCES

[1] M. L. Bushnell and V. D. Agrawal, “Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits”, Boston, MA: Kluwer, 2000.

[2] H. Ko and N. Nicolici, “Algorithms for state restoration and trace signal selection for data acquisition in silicon debug,” IEEE TCAD, vol. 28, no. 2, pp. 285–297, Feb. 2009.

[3] X. Liu and Q. Xu, “Trace signal selection for visibility enhancement in post-silicon validation”, in DAC, 2009.

[4] K. Basu and P. Mishra, “Efficient Trace Signal Selection for Post Silicon Validation and Debug,” in International Conference on VLSI Design, 2011.

[5] J. Yang and N. Touba, “Expanding trace buffer observation window for in-system silicon debug through selective capture,” in VTS, 2008.

[6] E. Anis and N. Nicolici, “Low cost debug architecture using lossy compression for silicon debug,” in DATE, 2007, pp. 225–230.

[7] E. Anis and N. Nicolici, “On using lossless compression of debug data in embedded logic analysis,” in Proceeding IEEE International Test Conference, 2007, pp.1–10.

[8] O. Caty, P. Dahlgren, and I. Bayraktaroglu, “Microprocessor silicon debug based on failure propagation tracing,” in ITC, 2005, pp. 10–293.

[9] B. Vermeulen and S. Goel, “Design for debug: Catching design errors in digital chips,” IEEE Design Test, vol. 19(3), pp. 37–45, 2002.

[10] R. Leatherman and N. Stollon “An embedded debugging architecture for socs,” IEEE Potentials, vol. 24, no. 1, pp. 12–16, February 2005.

[11] A. Chandra, K. Chakrabarty, “Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression”, in Proceeding VLSI Test Symposium, 2001, pp. 42-47.

[12] S. Prabhakar, R. Sethuram and Michael S. Hsiao, “Trace Buffer-Based Silicon Debug with Lossless Compression” in 24th Annual Conference on VLSI Design, 2011.

[13] S. Seong and P. Mishra, “Bitmask-based code compression for embedded systems”, IEEE TCAD, vol. 27(4), pp. 673–685, April 2008.

[14] J.-M. Cheng, L. M. Duyanovich, and D. J. Craft,“A Fast, Highly Reliable Data Compression Chip and Algorithm for Storage Systems” IBM Journal of Research and Development, 40(6), 1996.

[15] M.-B. Lin, J.-F.Lee, and G. Jan. A Lossless Data Compression and Decompression Algorithm and Its Hardware Architecture, IEEE Transactions on VLSI Systems, 14(9):925–936, September 2006.

[16] J. L. Nunez and S. Jones, Gbit/s Lossless Data Compression Hardware, IEEE Transactions on VLSI Systems, 11(3):499 –510, 2003.

[17] J. L. Nunez-Yanez and V. A. Chouliaras, Gigabyte per Second Streaming Lossless Data Compression Hardware Based on a Configurable Variable-Geometry CAM Dictionary. IEE Proceedings, Computers and Digital Techniques, 153(1):47–58, January 2006.

[18] Agalya.R and S.Saravanan, “Modified Trace Buffer for Error detection Using Efficient Compaction Method” in International Journal of Applied Engineering Research, vol. 9, pp.1399-1410, 2014.

[19] J-S Yang and N.A. Touba, “Improved Trace Buffer Observation via Selective Data Capture Using 2-D Compaction for Post-Silicon Debug” in IEEE Transaction on very large scale integration, vol. 21, pp.320-328, 2013.

AUTHOR PROFILE

Agalya R received B.E degree in Electronics and Communication Engineering from Periyar Maniammai University in 2008 and M.Tech degree in VLSI Design from SASTRA University in 2014. From August 2014, she has been a PhD candidate at SASTRA University. Her research interest includes VLSI design, System on Chip design testing and test compression.

Saravanan S is an Assistant Professor at School of Computing (SoC), SASTRA University, Thanjavur, Tamilnadu. He did his Ph.D in the domain of VLSI Design Testing and M.Tech in Computer Science Engineering. His research interests include VLSI testing, Low Power design and System on Chip and Embedded Systems. So far he has published 30+ research articles in SCOPUS indexed National & International journals.