Design of Test Data Compressor/Decompressor Using Xmatchpro Method

C. Suneetha*[1],V.V.S.V.S.Ramachandram*[2]
[1]Research Scholar, Dept. of E.C.E., Pragathi Engineering College, JNTU-K, A.P. , India [2] Associate Professor, Dept.,of E.C.E., Pragathi Engineering College, E.G.Dist., A.P., India

Abstract
Higher Circuit Densities in system-on-chip (SOC) 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/Decompression addresses this problem by reducing the test data volume without affecting the overall system performance. This paper presented an algorithm, XMatchPro Algorithm that combines the advantages of dictionary-based and bit mask-based techniques. Our test compression technique used the dictionary and bit mask selection methods to significantly reduce the testing time and memory requirements. We have applied our algorithm on various benchmarks and compared our results with existing test compression/Decompression techniques. Our approach results compression efficiency of 92%. Our approach also generates up to 90% improvement in decompression efficiency compared to other techniques without introducing any additional performance or area overhead.

Keywords - XMatchPro, decompression, test compression, dictionary selection, SOC.

I. INTRODUCTION

1.1. Objective

With the increase in silicon densities, it is becoming feasible for compression systems to be implemented in chip. A system with distributed memory architecture is based on having data compression and decompression engines working independently on different data at the same time. This data is stored in memory distributed to each processor. The objective of the project is to design a lossless data compression system which operates in high-speed to achieve high compression rate. By using the architecture of compressors, the data compression rates are significantly improved. Also inherent scalability of architecture is possible.The main parts of the system are the Xmatchpro based data compressors and the control blocks providing control signals for the Data compressors allowing appropriate control of the routing of data into and from the system. Each Data compressor can process four bytes of data

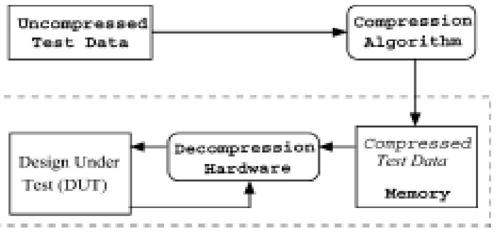


Fig. 1. Test data compression methodology.

into and from a block of data in every clock cycle. The data entering the system needs to be clocked in at a rate of 4 bytes in every clock cycle. This is to ensure that adequate data is present for all compressors to process rather than being in an idle state.

1.2. Goal of the Thesis

To achieve higher decompression rates using compression/decompression architecture with least increase in latency.

1.3. Compression Techniques

At present there is an insatiable demand for ever-greater bandwidth in communication networks and forever-greater storage capacity in computer system. This led to the need for an efficient compression technique. The compression is the process that is required either to reduce the volume of information to be transmitted – text, fax and images or reduce the bandwidth that is required for its transmission – speech, audio and video. The compression technique is first applied to the source information prior to its transmission. Compression algorithms can be classified in to two types, namely

- Lossless Compression
- Lossy Compression

1.3.1. Lossless Compression

In this type of lossless compression algorithm, the aim is to reduce the amount of source information to be transmitted in such a way that, when the compressed information is decompressed, there is no loss of information. Lossless compression is said, therefore, to be reversible. i.e., Data is not altered or lost in the process of compression or decompression. Decompression generates an exact replica of the original object. The Various lossless Compression Techniques are,

- Packbits encoding
- CCITT Group 3 1D
- CCITT Group 3 2D
- Lempel-Ziv and Welch algorithm LZW
- Huffman
- Arithmetic

Example applications of lossless compression are transferring data over a network as a text file since, in such applications, it is normally imperative that no part of the source information is lost during either the compression or decompression operations and file storage systems (tapes, hard disk drives, solid state storage, file servers) and communication networks (LAN, WAN, wireless).

1.3.2 Lossy Compression

The aim of the Lossy compression algorithms is normally not to reproduce an exact copy of the source information after decompression but rather a version of it that is perceived by the recipient as a true copy. The Lossy compression algorithms are:

- JPEG (Joint Photographic Expert Group)
- MPEG (Moving Picture Experts Group)
- CCITT H.261 (Px64)

Example applications of lossy compression are the transfer of digitized images and audio and video streams. In such cases, the sensitivity of the human eye or ear is such that any fine details that may be missing from the original source signal after decompression are not detectable.

1.3.3 Text Compression

There are three different types of text – unformatted, formatted and hypertext and all are represented as strings of characters selected from a defined set. The compression algorithm associated with text must be lossless since the loss of just a single character could modify the meaning of a complete string. The text compression is restricted to the use of entropy encoding and in practice, statistical encoding methods. There are two types of statistical encoding methods which are used with text: one which uses single character as the basis of deriving an optimum set of code words and the other which uses variable length strings of characters. Two examples of the former are the Huffman and Arithmetic coding algorithms and an example of the latter is Lempel-Ziv (LZ) algorithm. The majority of work on hardware approaches to lossless data compression has used an adapted form of the dictionary-based Lempel-Ziv algorithm, in which a large number of simple processing elements are arranged in a systolic array [1], [2], [3], [4].

II. Previous Work On Lossless Compression Methods

A second Lempel-Ziv method used a content addressable memory (CAM) capable of performing a complete dictionary search in one clock cycle [5], [6], [7]. The search for the most common string in the dictionary (normally, the most computationally expensive operation in the Lempel-Ziv algorithm) can be performed by the CAM in a single clock cycle, while the systolic array method uses a much slower deep pipelining technique to implement its dictionary search. However, compared to the CAM solution, the systolic array method has advantages in terms of reduced hardware costs and lower power consumption, which may be more important criteria in some situations than having faster dictionary searching. In [8], the authors show that hardware main memory data compression is both feasible and worthwhile. The authors also describe the design and implementation of a novel compression method, the XMatchPro algorithm. The authors exhibit the substantial impact such memory compression has on overall system performance. The adaptation of compression code for parallel implementation is investigated by Jiang and Jones [9]. They recommended the use of a processing array arranged in a tree-like structure. Although compression can be implemented in this manner, the implementation of the decompressor's search and decode stages in hardware would greatly increase the complexity of the design and it is likely that these aspects would need to be implemented sequentially. An FPGA implementation of a binary arithmetic coding architecture that is able to process 8 bits per clock cycle compared to the standard 1 bit per cycle is described by Stefo et al [10]. Although little research has been performed on architectures involving several independent compression units working in a concurrent cooperative manner, IBM has introduced the MXT chip [11], which has four independent compression engines operating on a shared memory area. The four Lempel-Ziv compression engines are used to provide data throughput sufficient for memory compression in computer servers. Adaptation of software compression algorithms to make use of multiple CPU systems was demonstrated by research of Penhorn [12] and Simpson and Sabharwal [13].

Penhorn used two CPUs to compress data using a technique based on the Lempel-Ziv algorithm and showed that useful compression rate improvements can be achieved, but only at the cost of increasing the learning time for the dictionary. Simpson and Sabharwal described the software implementation of compression system for a multiprocessor system based on the parallel architecture developed by Gonzalez and Smith and Storer [14].

Statistical Methods

Statistical Modeling of lossless data compression system is based on assigning values to events depending on their probability. The higher the value, the higher the probability. The accuracy with which this frequency distribution reflects reality determines the efficiency of the model. In Markov modeling, predictions are done based on the symbols that precede the current symbol. Statistical Methods in hardware are restricted to simple higher order modeling using binary alphabets that limits speed, or simple multi symbol alphabets using zeroeth order models that limits compression. Binary alphabets limit speed because only a few bits (typically a single bit) are processed in each cycle while zeroeth order models limit compression because they can only provide an inexact representation of the statistical properties of the data source.

Dictionary Methods

Dictionary Methods try to replace a symbol or group of symbols by a dictionary location code. Some dictionary-based techniques use simple uniform binary codes to process the information supplied. Both software and hardware based dictionary models achieve good throughput and competitive compression. The UNIX utility 'compress' uses Lempel-Ziv-2 (LZ2) algorithm and the data compression Lempel-Ziv (DCLZ) family of compressors initially invented by Hewlett-Packard[16] and currently being developed by AHA[17],[18] also use LZ2 derivatives. Bunton and Borriello present another LZ2 implementation in [19] that improves on the Data Compression Lempel-Ziv method. It uses a tag attached to each dictionary location to identify which node should be eliminated once the dictionary becomes full.

XMatchPro Based System

The Lossless data compression system is a derivative of the XMatchPro Algorithm which originates from previous research of the authors [15] and advances in FPGA technology. The flexibility provided by using this technology is of great interest since the chip can be adapted to the requirements of a particular application easily. The drawbacks of some of the previous methods are overcome by using the XmatchPro algorithm in design. The objective is then to obtain better compression ratios and still maintain a high throughput so that the compression/decompression processes do not slow the original system down.

Usage of XMatchPro Algorithm

The Lossless Data Compression system designed uses the XMatchPro Algorithm. The XMatchPro algorithm uses a fixed-width dictionary of previously seen data and attempts to match the current data element with a match in the dictionary. It works by taking a 4-byte word and trying to match or partially match this word with past data. This past data is stored in a dictionary, which is constructed from a content addressable memory. As each entry is 4 bytes wide, several types of matches are possible. If all the bytes do not match with any data present in the dictionary they are transmitted with an additional miss bit. If all the bytes are matched then the match location and match type is coded and transmitted, this match is then moved to the front of the dictionary. The dictionary is maintained using a move to front strategy whereby a new tuple is placed at the front of the dictionary while the rest move down one position. When the dictionary becomes full the tuple placed in the last position is discarded leaving space for a new one. The coding function for a match is required to code several fields as follows:

A zero followed by:

1). Match location: It uses the binary code associated to the matching location. 2). Match type: Indicates which bytes of the incoming tuple have matched. 3). Characters that did not match transmitted in literal form. A description of the XMatchPro algorithm in pseudo-code is given in the figure below.

Pseudo Code for XMatchPro Algorithm

With the increase in silicon densities, it is becoming feasible for XMatchPros to be implemented on a single chip. A system with distributed memory architecture is based on having data compression and decompression engines working independently on different data at the same time.This data is stored in memory distributed to each processor. There are several approaches in which data can be routed to and from the compressors that will affect the speed, compression and complexity of the system. Lossless compression removes redundant information from the data while they are transmitted or before they are stored in memory. Lossless decompression reintroduces the redundant information to recover fully the original data. There are two important contributions made by the current compression & decompression work, namely, improved compression rates and the inherent scalability. Significant improvements in data compression rates have been achieved by sharing the computational requirement between compressors without significantly compromising the contribution made by individual compressors. The scalability feature permits future bandwidth or storage demands to be met by adding additional compression engines.

The XMatchPro based Compression system

The XMatchPro algorithm uses a fixed width dictionary of previously seen data and attempts to match the current data element with a match in the dictionary. It works by taking a 4-byte word and trying to match this word with past data. This past data is stored in a dictionary, which is constructed from a content addressable memory. Initially all the entries in the dictionary are empty & 4-bytes are added to the front of the dictionary, while the rest move one position down if a full match has not occurred. The larger the dictionary, the greater the number of address bits needed to identify each memory location, reducing compression performance. Since the number of bits needed to code each location address is a function of the dictionary size greater compression is obtained in comparison to the case where a fixed size dictionary uses fixed address codes for a partially full dictionary.In the XMatchPro system, the data stream to be compressed enters the compression system, which is then partitioned and routed to the compressors.

The Main Component- Content Addressable Memory

Dictionary based schemes copy repetitive or redundant data into a lookup table (such as CAM) and output the dictionary address as a code to replace the data.

The compression architecture is based around a block of CAM to realize the dictionary. This is necessary since the search operation must be done in parallel in all the entries in the dictionary to allow high and data-independent

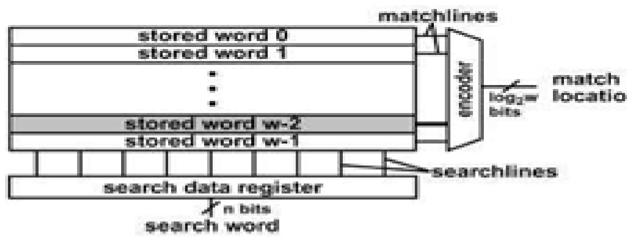


Fig..Conceptual view of CAM

The number of bits in a CAM word is usually large, with existing implementations ranging from 36 to 144 bits. A typical CAM employs a table size ranging between a few hundred entries to 32K entries, corresponding to an address space ranging from 7 bits to 15 bits. The length of the CAM varies with three possible values of 16, 32 or 64 tuples trading complexity for compression. The no. of tuples present in the dictionary has an important effect on compression. In principle, the larger the dictionary the higher the probability of having a match and improving compression. On the other hand, a bigger dictionary uses more bits to code its locations degrading compression when processing small data blocks that only use a fraction of the dictionary length available. The width of the CAM is fixed with 4bytes/word. Content Addressable Memory (CAM) compares input search data against a table of stored data, and returns the address of the matching data. CAMs have a single clock cycle throughput making them faster than other hardware and software-based search systems. The input to the system is the search word that is broadcast onto the searchlines to the table of stored data. Each stored word has a matchline that indicates whether the search word and stored word are identical (the match case) or are different (a mismatch case, or miss). The matchlines are fed to an encoder that generates a binary match location corresponding to the matchline that is in the match state. An encoder is used in systems where only a single match is expected.

The overall function of a CAM is to take a search word and return the matching memory location.

Managing Dictionary entries

Since the initialization of a compression CAM sets all words to zero, a possible input word formed by zeros will generate multiple full matches in different locations.The Xmatchpro compression system simply selects the full match closer to the top. This operational mode initializes the dictionary to a state where all the words with location address bigger than zero are declared invalid without the need for extra logic.

Iii. Xmatchpro Lossless Compression System

DESIGN METHODOLOGY

The XMatchPro algorithm is efficient at compressing the small blocks of data necessary with cache and page based memory hierarchies found in computer systems. It is suitable for high performance hardware implementation. The XMatchPro hardware achieves a throughput 2-3 times greater than other high-performance hardware implementation. The core component of the system is the XMatchPro based

Compression/ Decompression system. The XMatchPro is a high-speed lossless dictionary based data compressor. The XMatchPro algorithm works by taking an incoming four-byte tuple of data and attempting to match fully or partially match the tuple with the past data.

FUNCTIONAL DESCRIPTION

The XMatchPro algorithm maintains a dictionary of data previously seen and attempts to match the current data element with an entry in the dictionary, replacing it with a shorter code referencing the match location. Data elements that do not produce a match are transmitted in full (literally) prefixed by a single bit. Each data element is exactly 4 bytes in width and is referred to as tuple. This feature gives a guaranteed input data rate during compression and thus also guaranteed data rates during decompression, irrespective of the data mix. Also the 4-byte tuple size gives an inherently higher throughput than other algorithms, which tend to operate on a byte stream. The dictionary is maintained using move to front strategy, where by the current tuple is placed at the front of the dictionary and the other tuples move down by one location as necessary to make space. The move to front strategy aims to exploit locality in the input data. If the dictionary becomes full, the tuple occupying the last location is simply discarded.

A full match occurs when all characters in the incoming tuple fully match a dictionary entry. A partial match occurs when at least any two of the characters in the incoming tuple match exactly with a dictionary entry, with the characters that do not match being transmitted literally.

The use of partial matching improves the compression ratio when compared with allowing only 4 byte matches, but still maintains high throughput. If neither a full nor partial match occurs, then a miss is registered and a single miss bit of '1' is transmitted followed by the tuple itself in literal form. The only exception to this is the first tuple in any compression operation, which will always generate a miss as the dictionary begins in an empty state. In this case no miss bit is required to prefix the tuple.

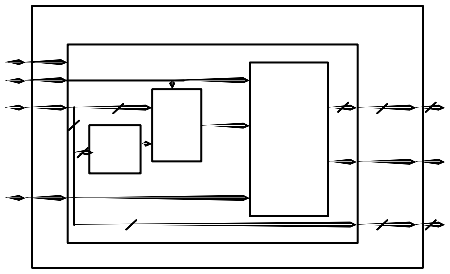
At the beginning of each compression operation, the dictionary size is reset to zero. The dictionary then grows by one location for each incoming tuple being placed at the

front of the dictionary and all other entries in the dictionary moving down by one location. A full match does not grow the dictionary, but the move-to-front rule is still applied. This growth of the dictionary means that code words are short during the early stages of compressing a block. Because the XMatchPro algorithm allows partial matches, a decision must be made about which of the locations provides the best overall match, with the selection criteria being the shortest possible number of output bits.

Implementation of Xmatchpro Based Compressor

The block diagram gives the details about the components of a single 32 bit Compressor. There are three components namely, COMPARATOR, ARRAY, CAMCOMPARATOR. The comparator is used to compare two 32-bit data and to set or reset the output bit as 1 for equal and 0 for unequal. The CAM COMPARATOR searches the CAM dictionary entries for a full match of the input data given. The reason for choosing a full match is to get a prototype of the high throughout Xmatchpro compressor with reduced complexity and high performance.
If a full match occurs, the match-hit signal is generated and the corresponding match location is given as output by the CAM Comparator.. If no full match occurs, the corresponding data that is given as input at the given time is got as output.

32 BIT COMPRESSIONS



Array is of length of 64X32 bit locations. This is used to store the unmatched incoming data and when a new data comes, the incoming data is compared with all the data stored in this array. If a match occurs, the corresponding match location is sent as output else the incoming data is stored in next free location of the array & is sent as output. The last component is the cam comparator and is used to send the match location of the CAM dictionary as output if a match has occurred. This is done by getting match information as input from the comparator.

Suppose the output of the comparator goes high for any input, the match is found and the corresponding address is retrieved and sent as output along with one bit to indicate that match is found. At the same time, suppose no match occurs, or no matched data is found, the incoming data is stored in the array and it is sent as the output. These are the functions of the three components of the Compressor

IV.Design of Lossless Compression/Decompression System

DESIGN OF COMPRESSOR / DECOMPRESSOR

The block diagram gives the details about the components of a single 32-bit compressor / decompressor. There are three components namely COMPRESSOR, DECOMPRESSOR and CONTROL.

The compressor has the following components - COMPARATOR, ARRAY, and CAMCOMPARATOR.

The comparator is used to compare two 32-bit data and to set or reset the output bit as 1 for equal and 0 for unequal.

Array is of length of 64X32bit locations. This is used to store the unmatched in coming data and when the next new data comes, that data is compared with all the data stored in this array. If the incoming data matches with any of the data stored in array, the Comparator generates a match signal and sends it to Cam Comparator.

The last component is the Cam comparator and is used to send the incoming data and all the stored data in array one by one to the comparator. Suppose output of comparator goes high for any input, then the match is found and the corresponding address (match location) is retrieved and sent as output along with one bit to indicate the match is found. At the same time, suppose no match is found, then the incoming data stored in the array is sent as output. These are the functions of the three components of the XMatchPro based compressor.

The decompressor has the following components – Array and Processing Unit.

Array has the same function as that of the array unit which is used in the Compressor. It is also of the same length. Processing unit checks the incoming match hit data and if it is 0, it indicates that the data is not present in the Array, so it stores the data in the Array and if the match hit data is 1, it indicates the data is present in the Array, then it instructs to find the data from the Array with the help of the address input and sends as output to the data out.

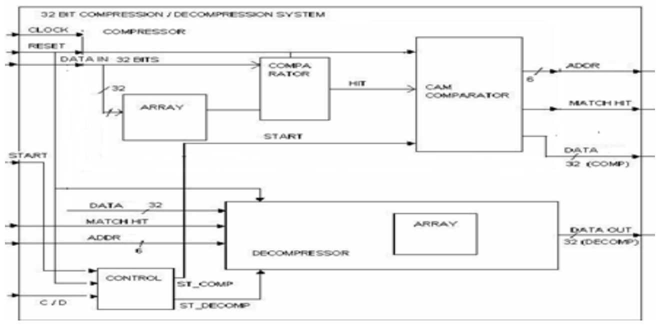


Fig. Block Diagram of 32 bit Compressor/Decompression

The Control has the input bit called C / D i.e., Compression / Decompression Indicates whether compression or decompression has to be done. If it has the value 0 then omcpressor is stared when the value is 1 decompression is done

V. Result Analysis

1 SYNTHESIS REPORT

Release 8.2i - xst I.31

Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.

--> Parameter TMPDIR set to ./xst/projnav.tmp

CPU : 0.00 / 0.17 s | Elapsed : 0.00 / 0.00 s

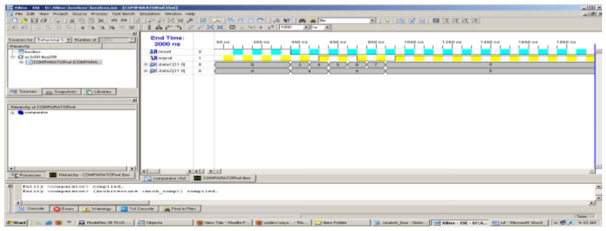
--> Parameter xsthdpdir set to ./xst

CPU : 0.00 / 0.17 s | Elapsed : 0.00 / 0.00 s

--> Reading design: ds.prj

Table of Contents

COMPARATOR



COMPARATOR waveform explanation

Whenever the reset signal is active low then only the data is compared otherwise i.e. if it is active high the eqout signal is zero. Here we are applying 3 4 5 6 7 8 as data1 and 4 5 8 as data2. So after comparing the inputs the output signal eqout raises at the data 4 and 8 instants which indicate the output signal.

COMPARATOR function

Suppose the output of the comparator goes high for any input, the match is found and the corresponding address is retrieved and sent as output along with one bit to indicate that match is found. At the same time, suppose no match occurs, or no matched data is found, the incoming data is stored in the array and it is sent as the output.

CAM

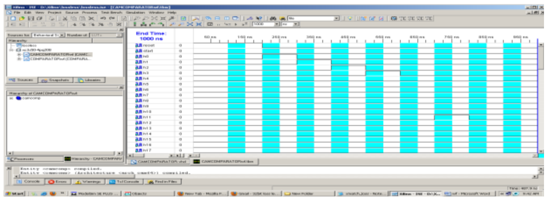


Fig. CAM data input Waveform

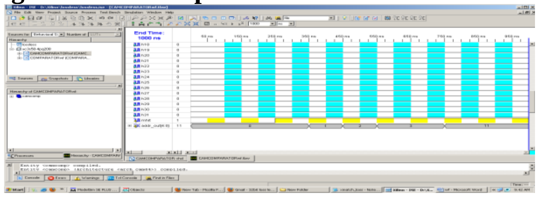


Fig. CAM data output Waveform

CAM waveform explanation

Whenever the reset signal is active low then only the cam produces the corresponding address and the match hit signal by individually raise the corresponding hit otherwise i.e. if it is active high it doesn't produces any signal. In order to obtain the outputs the start should be in active low state.

CAM function

The cam is used to send the match location Of the CAM dictionary as output if a match has occurred. This is done by getting match Information as input from the comparator.

Compression

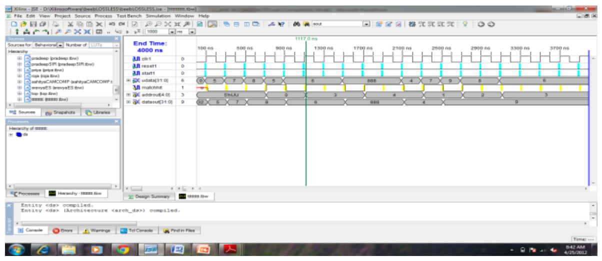


Fig. Compression of INPUT DATA waveform

Compression waveform explanation

Whenever the reset1 and start1 signal is active low then only the compression of data is occurred. Here every signal reacts at the clk1 rising edge only because it is meant for positive edge triggering. The data which has to be compressed is applied at the udata signal and the resultant data is obtained at the dataout signal which is nothing but compressed data. Whenever there is a redundancy data is there in the applied data then the matchhit signal will be in active high state by generating the corresponding address signal addrout.

Compression function

The compression engine main intension is to compress the incoming content for memory reduction purpose and to transfer the data very easily.

Decompression

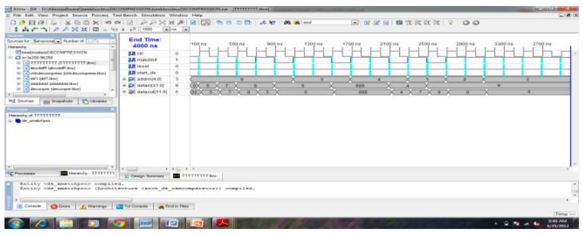


Fig. Decompression of the data

Decompression waveform explanation

Whenever the reset and start_de signal is active low then only the decompression of data is occurred. Here every signal reacts at the clk1 rising edge only because it is meant for positive edge triggering. Here for retrieving the original data the outputs of the compressor engine has to be applied as inputs for the decompression engine.

Decompression function

The decompression engine main intension is to retrieve the original content.

SCHEMATIC

Compression

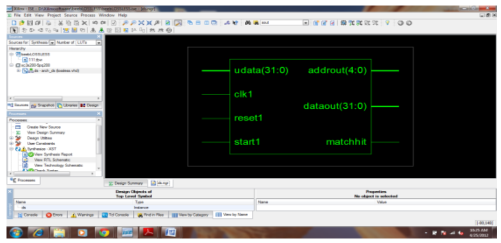


Fig. RTL Schematic of compression waveform of component level

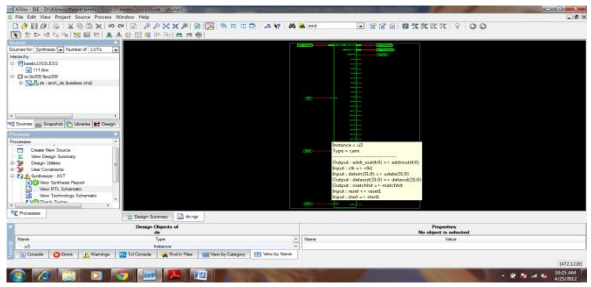


Fig. RTL Schematic of compression waveform of circuit level

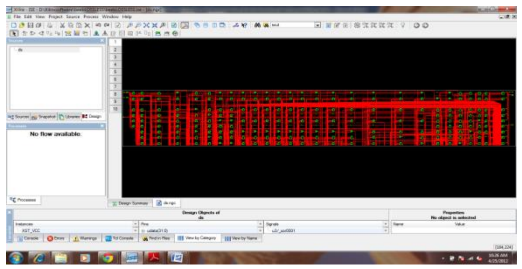


Fig.RTL Schematic of compression waveform of chip level

Decompression

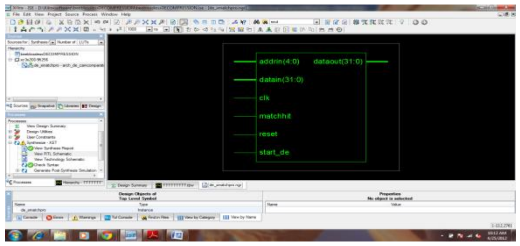


Fig RTL Schematic of decompression waveform of component level

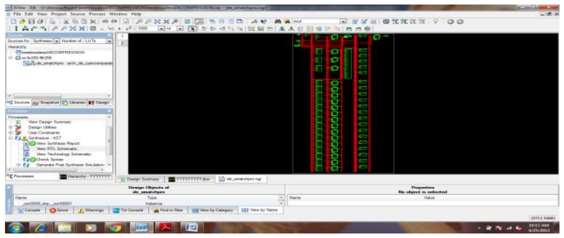


Fig. RTL Schematic of decompression waveform of circuit level

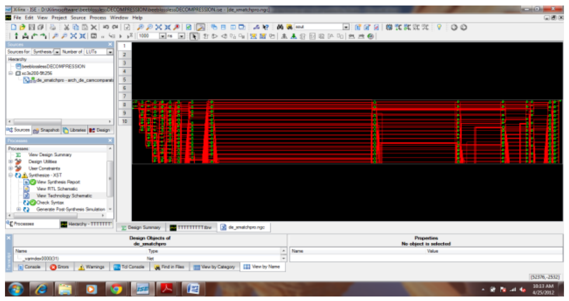


Fig. RTL Schematic of decompression waveform of chip level

VI.CONCLUSION

The various modules are designed and coded using VHDL. The source codes are simulated and the various waveforms are obtained for all the modules. Since the Compression/Decompression system uses XMatchPro algorithm, speed of compression throughput is high.

The Improved Compression ratio is achieved in Compression architecture with least increase in latency. The High speed throughput is achieved. The architecture provides inherent scalability in future.

The total time required to transmit compressed data is less than that of

Transmitting uncompressed data. This can lead to a performance benefit, as the bandwidth of a link appears greater when transmitting compressed data and hence more data can be transmitted in a given amount of time.

Higher Circuit Densities in system-on-chip (SOC) 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/Decompression addresses this problem by reducing the test data volume without affecting the overall system performance. This paper presented an algorithm, XMatchPro Algorithm that combines the advantages of dictionary-based and bit mask-based techniques. Our test compression technique used the dictionary and bit mask selection methods to significantly reduce the testing time and memory requirements. We have applied our algorithm on various benchmarks and compared our results with existing test compression/Decompression techniques. Our approach results compression efficiency of 92%. Our approach also generates up to 90% improvement in decompression efficiency compared to other techniques without introducing any additional performance or area overhead.

Vii. Future Scope

There is a potential of doubling the performance of storage / communication system by increasing the available transmission bandwidth and data capacity with minimum investment. It can be applied in Computer systems, High performance storage devices. This can be applied to 64 bit also in order to increase the data transfer rate. There is a chance to easy migration to ASIC technology which enables 3-5 times increase in performance rate. There is a chance to develop an engine which does not require any external components and supports operation on blocked data. Full-duplex operation enables simultaneous compression /decompression for a combined performance of high data rates, which also enables self-checking test mode using CRC (Cyclic Redundancy Check). High-performance coprocessor-style interface should be possible if synchronization is achieved. Chance to improve the Compression ratio compare with proposed work.

VIII. Acknowledgements

The authors would like to thank everyone, whoever remained a great source of help and inspirations in this humble presentation. The authors would like to thank Pragathi Engineering College Management for providing necessary facilities to carry out this work.

References

[1]. Li, K. Chakrabarty, and N. Touba, “Test data compression using dictionaries with selective entries and fixed-length indices,” ACM Trans.Des. Autom. Electron. Syst., vol. 8, no. 4, pp. 470–490, 2003.

[2]. Wunderlich and G. Kiefer, “Bit-flipping BIST,” in Proc. Int. Conf. Comput.Aided Des., 1996, pp. 337–343.

[3]. N. Touba and E. McCluskey, “Altering a pseudo-random bit sequence for scan based bist,” in Proc. Int. Test Conf., 1996, pp. 167–175.

[4]. F. Hsu, K. Butler, and J. Patel, “A case study on the implementation of Illinois scan architecture,” in Proc. Int. Test Conf., 2001, pp. 538–547.

[5]. M. Ros and P. Sutton, “A hamming distance based VLIW/EPIC code compression technique,” in Proc. Compilers, Arch., Synth. Embed. Syst., 2004, pp. 132–139.

[6]. Seong and P. Mishra, “Bitmask-based code compression for embed systems,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 27, no. 4, pp. 673–685, Apr. 2008.

[7]. M.-E. N. A. Jas, J. Ghosh-Dastidar, and N. Touba, “An efficient test vector compression scheme using selective Huffman coding,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 22, no. 6, pp.797–806, Jun. 2003.

[8]. A. Jas and N. Touba, “Test vector decompression using cyclical scan chains and its application to testing core based design,” in Proc. Int.Test Conf., 1998, pp. 458–464.

[9]. A. Chandra and K. Chakrabarty, “System on a chip test data compression and decompression architectures based on Golomb codes,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 20, no. 3, pp.355–368, Mar. 2001.

[10]. X. Kavousianos, E. Kalligeros, and D. Nikolos, “Optimal selective Huffman coding for test-data compression,” IEEE Trans. Computers, vol. 56, no. 8, pp. 1146–1152, Aug. 2007.

[11]. M. Nourani andM. Tehranipour, “RL-Huffman encoding for test compression and power reduction in scan applications,” ACM Trans. Des.Autom. Electron. Syst., vol. 10, no. 1, pp. 91–115, 2005.

[12]. H. Hashempour, L. Schiano, and F. Lombardi, “Error-resilient test data compression using Tunstall codes,” in Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst., 2004, pp. 316–323.

[13]. M. Knieser, F.Wolff, C. Papachristou, D.Weyer, and D. McIntyre, “A technique for high ratio LZWcompression,” in Proc. Des., Autom., Test Eur., 2003, p. 10116.

[14]. M. Tehranipour, M. Nourani, and K. Chakrabarty, “Nine-coded compression technique for testing embedded cores in SOCs,” IEEE Trans.Very Large Scale Integr. (VLSI) Syst., vol. 13, pp. 719–731, Jun. 2005.

[15]. L. Lingappan, S. Ravi, A. Raghunathan, N. K. Jha, and S. T.Chakradhar, “Test volume reduction in systems-on-a-chip using heterogeneous and multilevel compression techniques,” IEEE Trans.Comput.-Aided Des. Integr. Circuits Syst., vol. 25, no. 10, pp.2193–2206, Oct. 2006.

[16]. 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.

[17]. X. Kavousianos, E. Kalligeros, and D. Nikolos, “Multilevel-Huffman test-data compression for IP cores with multiple scan chains,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 7, pp. 926–931,Jul. 2008.

[18]. X. Kavousianos, E. Kalligeros, and D. Nikolos, “Multilevel Huffman coding: An efficient test-data compression method for IP cores,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 26, no. 6, pp.1070–1083, Jun. 2007.

[19]. X. Kavousianos, E. Kalligeros, and D. Nikolos, “Test data compression based on variable-to-variable Huffman encoding with codeword reusability,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,vol. 27, no. 7, pp. 1333–1338, Jul. 2008.

[20]. S. Reda and A. Orailoglu, “Reducing test application time through test data mutation encoding,” in Proc. Des. Autom. Test Eur., 2002, pp.387–393.

[21]. E. Volkerink, A. Khoche, and S. Mitra, “Packet-based input test data compression techniques,” in Proc. Int. Test Conf., 2002, pp. 154–163.

[22]. S. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, “On test data volume reduction for multiple scan chain design,” in Proc. VLSI Test Symp., 2002, pp. 103–108.

[23]. F. Wolff and C. Papachristou, “Multiscan-based test compression and hardware decompression using LZ77,” in Proc. Int. Test Conf., 2002,pp. 331–339.

[24]. 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.

[25]. K. Basu and 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.

[26]. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms. boston, MA: MIT Press, 2001.

[27]. M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: Freeman, 1979.

[28]. Hamzaoglu and J. Patel, “Test set compaction algorithm for combinational circuits,” in Proc. Int. Conf. Comput.-Aided Des., 1998, pp.283–289.