iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** I **Month of publication:** January 2020

DOI: http://doi.org/10.22214/ijraset.2020.1063

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.177
Volume 8 Issue I, Jan 2020- Available at www.ijraset.com

Design and Implementation of Efficient LZW Compression and Decompression Technique

J V S R Sri Harsha Bolem[1], Lavanya Pavuluri[2], Nithisha Basineni[3], Nikhil G[4]
[1]*B.Tech, Member, IEEE,*
[2, 3, 4]*B.Tech, Dhanekula Institute of Engineering and Technology*

Abstract: ***Data compression and decompression are essential in order to reduce the resources that required during transmitting and receiving data and some applications where the deviations from could be deleterious. LZW is an universal lossless technique that provides high efficiency for text as well as image data and is one of the most famous dictionary-based algorithm. Here, the designed dictionary is based on content addressable memory (CAM) array. The code for each character is available in the dictionary which utilizes less number of bits (5 bits) than its ASCII code. However, LZW technique is chosen to implement from the set of lossless methods such that it can reduce the irrelevant and redundancy of data in order to be able to transmit in efficient form and also to increase the speed of transmission. The main contribution of this work is to present a hardware LZW data compression and decompression in which LZW compression& decompression algorithms are implemented, thus the text data can be effectively compressed and recovered more efficiently which is same as that of input data.***

Keywords: Compression Rate, LZW codes, Binary Data, FPGA, Finite State machines, Compression & Decompression and RAM logic cells.

I. INTRODUCTION

Data compression is often referred to as coding, where coding is a very general term encompassing any special representation of data which satisfies a given need. Information theory is defined to be the study of efficient coding and its consequences, in the form of speed of transmission and probability of error. Data compression may be viewed as a branch of information theory in which the primary objective is to minimize the amount of data to be transmitted. The purpose of this paper is to present and analyze an efficient lossless technique among different data compression algorithms.

A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Data compression has important application in the areas of data transmission and data storage. Many data processing applications require storage of large volumes of data, and the number of such applications is constantly increasing as the use of computers extends to new disciplines. At the same time, the proliferation of computer communication networks is resulting in a massive transfer of data over communication links. Compressing data to be stored or transmitted reduces storage and/or communication costs. When the amount of data to be transmitted is reduced, the effect is that of increasing the capacity of the communication channel. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. It may then become feasible to store the data in a higher, thus faster, level of the storage hierarchy and reduce the load on the input/output channels of the computer system.

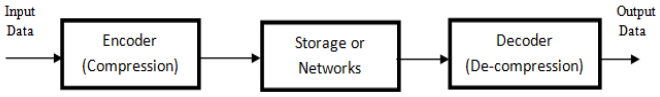


Fig 1: Basic block diagram

Compression means reducing the number of bits by eliminating redundant data from the original data and the process of decompression involves in recovering the original data without any deviation.

Differential Compression and decompression techniques are evolved mainly due to two reasons:

1) Technology needs to transfer the data more securely throughout the communication and to transmit the data through the limited bandwidth more efficiently.

2) People like to accumulate data and hate to throw anything away. No matter, however large a storage device may be, sooner or later it is going to overflow. Data compression seems useful because it delays this inevitable.

The system works on an input stream of data and builds up a list of dictionary entries (a translation table) of 'phrases' which appear in the stream. When a sub-string from the stream is already identified as being in the dictionary (that is, it has already occurred previously) it is replaced in the compressed output by a reference to the dictionary entry. If it is not present, a new entry is placed on the table, and the reference sent to the compressed stream. These references are generally smaller in size than the uncompressed phrases and thus compression is attained.

To decompress, a compressed stream is read and references are added to a dictionary if not present. The phrases can then be restored building up the dictionary as it progresses. The advantage of this procedure is that it is not necessary to store the table within the compressed output as it is built up as required when decompression takes place. When initializing the dictionary before compression, the first 256 entries are set to values 0016 through FF16 (all possible byte values) from which all sub-strings can be built. As both the encoder and decoder are aware of this, there is no need to keep the dictionary stored with the data.

II. LOSSLESS TECHNIQUE

The main aim of the field of data compression is of course to develop methods for better and better compression. Experience shows that fine tuning an algorithm to squeeze out the last remaining bits of redundancy of the data gives diminishing returns. Data compression has become so important that some researchers have proposed the "simplicity and power theory". Specifically it says, data compression may be interpreted as a process of removing unnecessary complexity in information and thus maximizing the simplicity while preserving as much as possible of its non-redundant descriptive power.

Lossless technique is used to reduce the amount of source information to be transmitted in such a way that when compressed information is decompressed, there is not any loss of information. It achieves only about a 2:1 compression ratio. This type of compression technique looks for patterns in strings of bits and then expresses them more concisely.

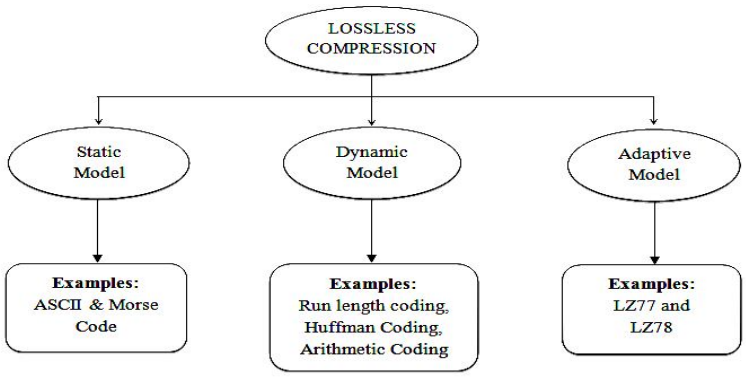


Fig 2: Classification of Lossless Technique

III. LZW TECHNIQUE

LZW is a lossless method for compressing data can be found in several of the popular image file formats, including GIF, TIFF and PostScript Level 2. It is based around substitution, or dictionary-based LZ77 and LZ78 algorithms devised by Abraham Lempel and JakobZiv in 1977-78. This was extended in 1984 by Terry Welch, thus Lempel-Ziv Welch was born.

It provides a fast symmetric way of compressing and decompressing any type of data without the need for floating point operations. Another reason for its popularity is that it works as well on little- and big-endian machines because information is written as bytes and not words (although the bit-order and fill-order problems could still be encountered).

A. Dictionary Based Technique

As the process of compression and decompression takes place with the help of different methods, which includes the efficient technique but that provides the lower compression rate. As we need both efficiency and compression rate, we need to go for a new technique that is dictionary based techniques. Among the different dictionary based techniques LZW technique is most suitable in various circumstances like speed, compression ratio and efficiency. Here, the dictionary was created based on the characters present in a given input string that might be non-repeated characters and providing the index values for each and every character. Let us consider an example to describe how assigning of index values takes place. Let input string be "abbabcababc". Here non-repeated characters are only "ABC", we are providing value for only these 3 characters

Character	ASCII value	Providing Index
A	97 (1100001)	1 (01)
B	98 (1100010)	2 (10)
C	99 (1100011)	3 (11)

Table 1

Here 7 bit ASCII value of each character is replaced with 2 bit INDEX value. The index values of the dictionary are varied in accordance with the different combinations developed in string due to these character sets.
LZW compression replaces strings of characters with single codes which were given in above table. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.
The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to the strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0-255 refer to individual bytes, while codes 256-4095 refers to substrings.

B. Elimination Of Repeated Characters

Initially a new dynamic string is to be declaring to perform an elimination process of repeated characters and input string must remain unchanged to perform decompression task. Removing repeated characters of a given input string will go to perform here as shown in the algorithm.

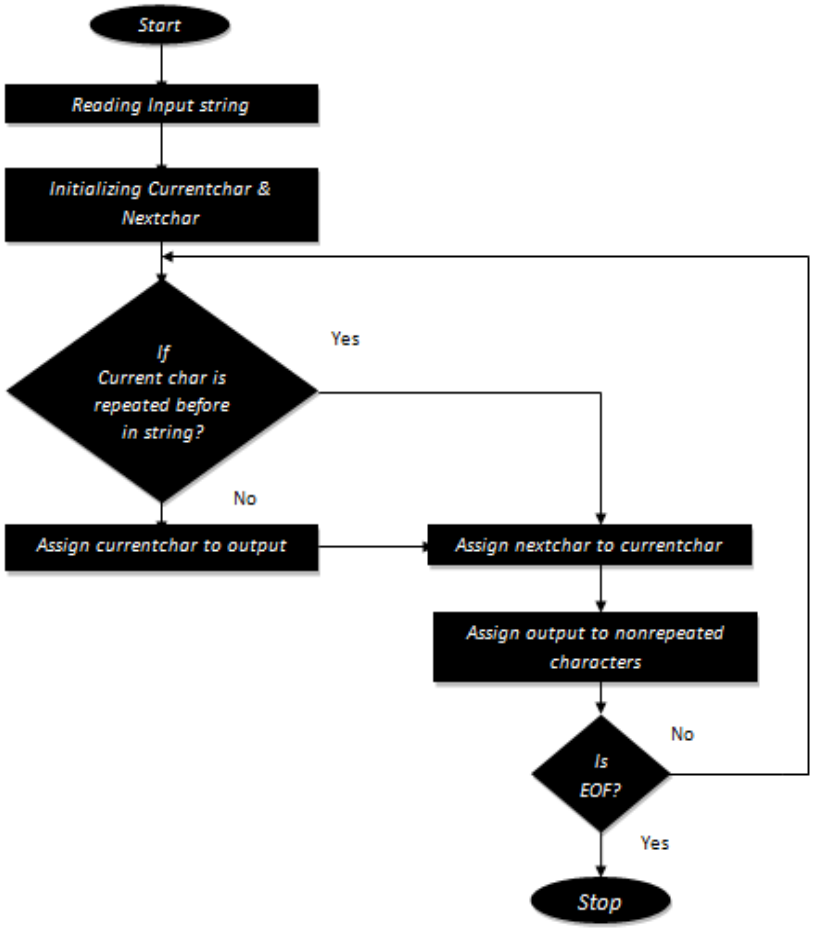


Fig 3: Flowchart for elimination of repeated characters

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.177
Volume 8 Issue I, Jan 2020- Available at www.ijraset.com

C. Creating Dictionary

Several variations of the LZW algorithm increase its efficiency in some applications. One common variation uses index pointers that vary in length, usually starting at 9 bits and growing upward to 12 or 13 bits. When an index pointer of a particular length has been used up, another bit is tacked on to increase precision.

Before creating a dictionary for providing index values we need to arrange the non-repeated character set in order to provide better understanding. Here we need to create a multiple dictionaries to store multiple values which lead to better efficiency. Index values to be provided based on the set of characters that obtained under non-repeated characters of the string. Based on provided indices the code word requires less number of bits when compared with ASCII values of characters. The pictorial representation is given below

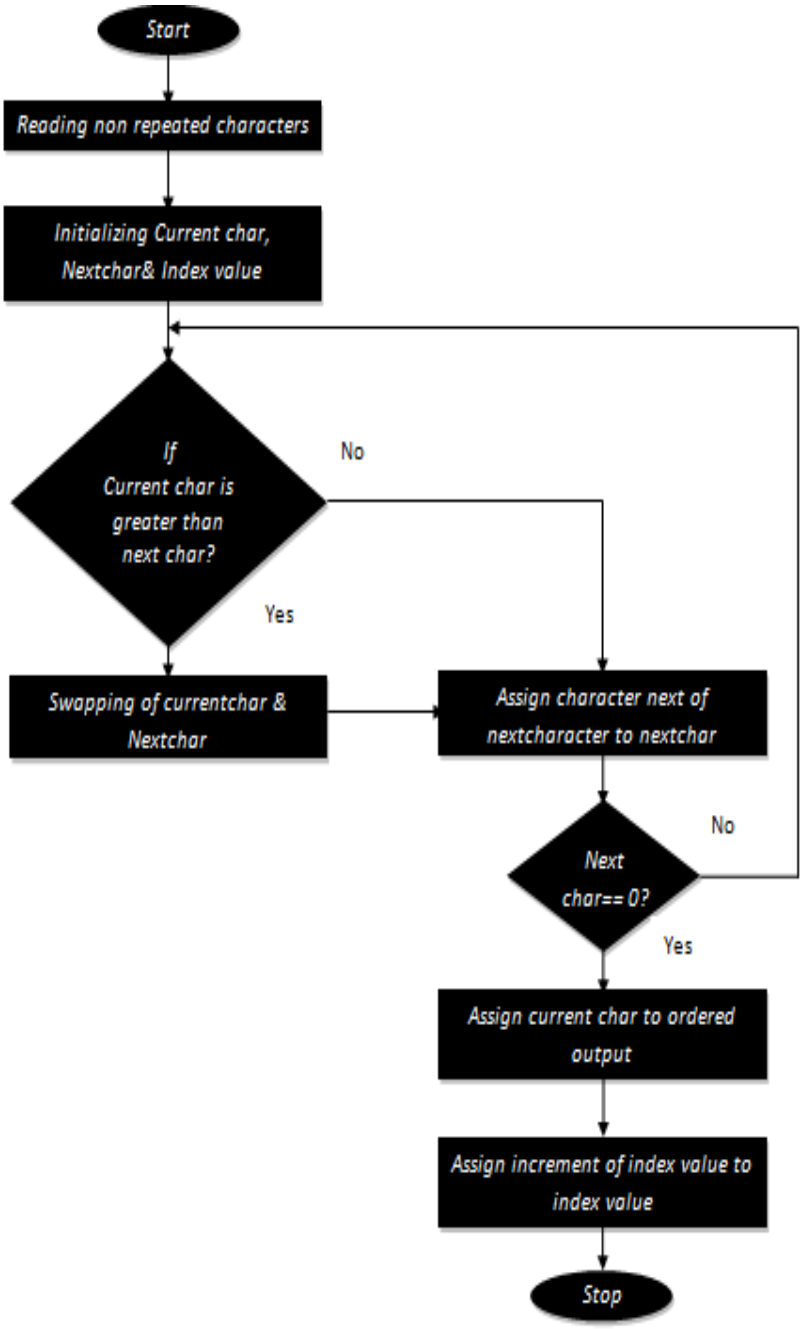


Fig 4: Flowchart for assigning indices using dictionary

D. Compression

While data can be covered up inside writings in such a path, to the point that the vicinity of the message must be recognized with information of the secret key, for instance, when utilizing the prior specified system utilizing a freely accessible book and a mix of character positions to conceal the message, the vast majority of the systems include alterations to the cover source. These alterations can be caught by searching for examples in the writings or disturbing thereof. To perform this we need to compress the given text data, thus in this thesis we are taking a string which contains the set of characters and is compressing each and every character and taken algorithm mainly concentrate on repeated data.

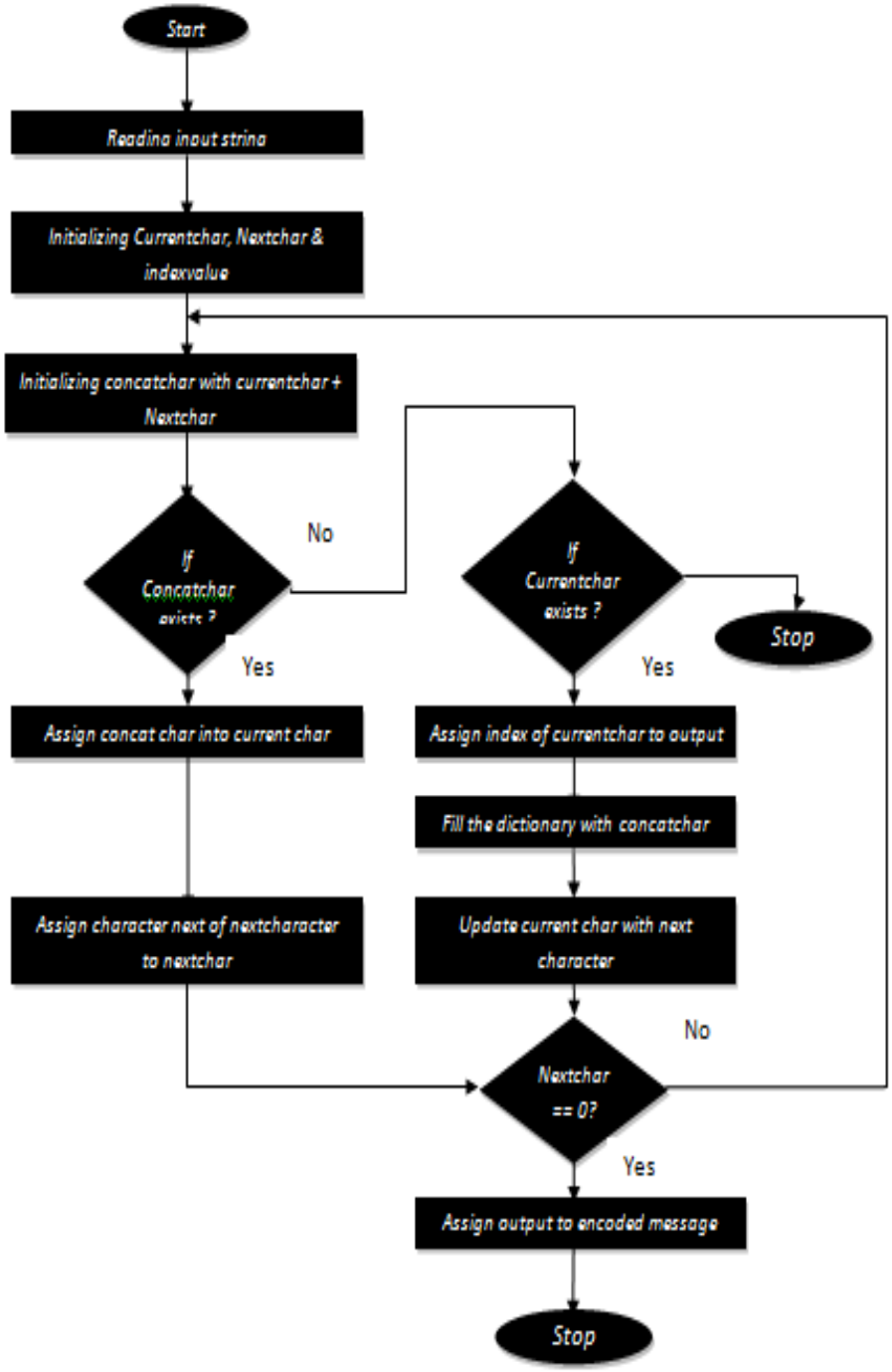


Fig 5: Flowchart for compression phase

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.177
Volume 8 Issue I, Jan 2020- Available at www.ijraset.com

E. Decompression

In communication channel after transmitting data, the data to be recovered at receiving end. For many applications we need the decompressed data which is an exact replica of input data without any deviation. Here the code word obtained as the output of compressor is used as input and the process involves in this phase is that each value of code word is reading and searching in the dictionary where we store the index values of single characters. If not found then we are moving to another dictionary which contains the index values of different combinations of multiple characters. If found, then the character of corresponding position where the value found is provided as decoded output. The decoded output must be concatenated to provide the entire string as output. To provide all this we need to store the output in dynamic string and for index values we need to initialize dynamic arrays and different dictionaries to store single characters in one and multiple combinations in another dictionary. The entire process of decompression is represented in diagrammatic form as shown below:

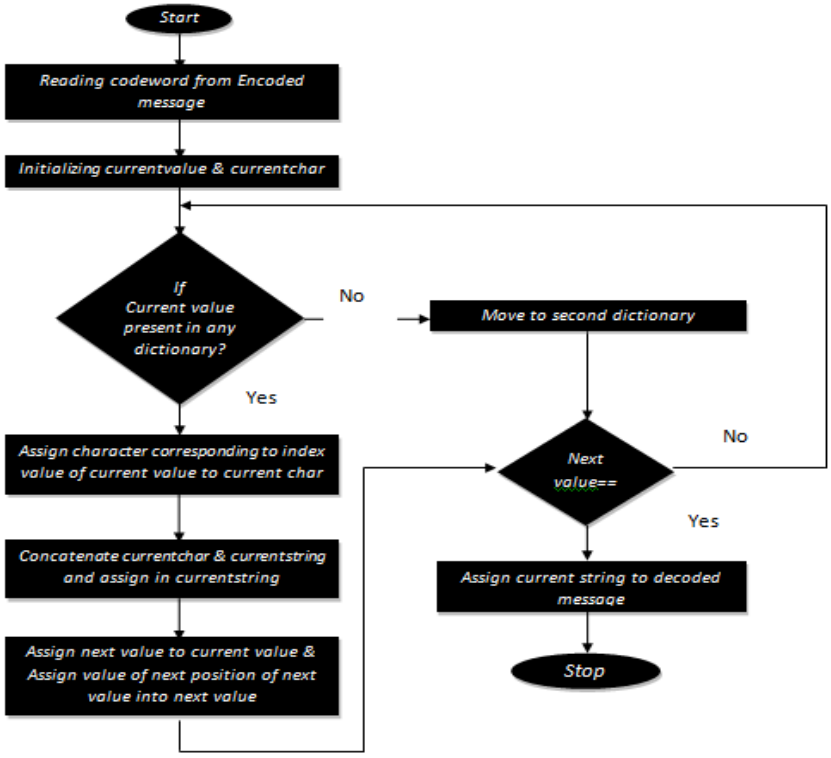


Fig 6: Flowchart for decompreesion phase

IV. EXPERIMENTAL EVALUATION

In the proposed method of text data compression, the non-repeated characters and their indices stored using dictionary are the main considerations. Result analysis of the proposed scheme is mainly based on the set of characters which forms as string data.

Input – ACBACCACCCBACACCCD$

Compressed output –2 4 3 6 4 9 7 6 11 5 1

Decompressed Output –ACBACCACCCBACACCCD$

The input string is shown in below figure:

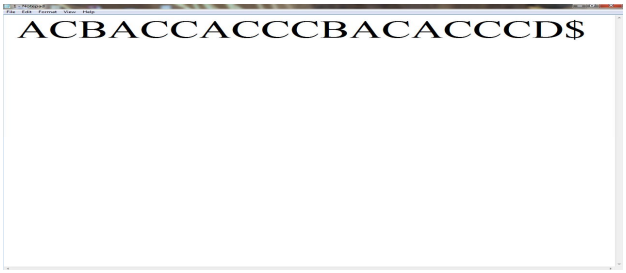


Fig 5

A. Elimination of Repeated Characters

In this phase, each and every character compare with remaining to eliminate the repeated set for providing non-repeated characters as output.

Fig 6

B. Providing Indices For Non-Repeated Characters

The output of non repeated characters is the input for phase-2 that provides the indices for characters arranged in order.

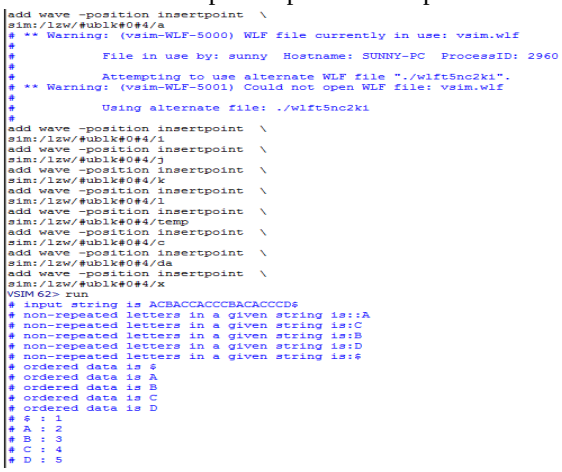


Fig 7

C. Compression

Based on indices provided, the compression operation is going to perform which mainly depends on various dictionaries of different sizes.

Fig 8

D. Decompression

As the codeword obtained as the output of compression phase the decompression is also completely based on dictionaries created for performing compression phase.

```
#
# Top level modules:
#       lzw
ModelSim> vsim lzw
# vsim lzw
# Loading sv_std.std
# Loading work.lzw
# ** Warning: (vsim-PLI-3003) new1.sv(222): [TOFD] - System task or function '$diplay' is not defined.
#
#         Region: /lzw/#ublk#0#4
VSIM 98> run -all
# input string is ACBACCACCCBACACCCD$
# $ : 1
# A : 2
# B : 3
# C : 4
# D : 5
# AC : 6
# CB : 7
# BA : 8
# CA : 10
# D$ : 14
# $  : 15
#  ACC : 9
#  CBA : 12
#  ACA : 13
# ACCC : 11
# Compressed output :
#  '{2, 4, 3, 6, 4, 9, 7, 6, 11, 5, 1}
# Decompressed output :
# A
# C
# B
# A
# C
# C
# A
# C
# C
# C
# B
# A
# C
# A
# C
# C
# C
# D
# $

VSIM 99>
```

Fig 9

V. PERFORMANCE ANALYSIS

A. Compression Ratio

It is a very logical way of measuring how well a compression algorithm compresses a given set of data is to look at the ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression. This ratio is called 'Compression ratio' (Or) Compression Ratio is the ratio between the size of the compressed file and the size of the source file. Ex. Suppose storing an image requires 65536 bytes, this image is compressed and the compressed version requires 16384 bytes. So the compression ratio is 4:1. It can be also represented in terms of reduction in the amount of data required as a percentage i.e. 75%

$$compressionratio = \frac{sizebeforecompression}{sizeaftercompression}$$

Here Size before compression = 19 (Characters)* 7 (ASCII)

= 133 Bits

Size after compression = 11 (Code word) * 4 (Max No of bits)

= 44 Bits

Compression Ratio = 133/44

= **3.023: 1**

B. Compression Factor

It is the inverse of the compression ratio. That is the ratio between the size of the source file and the size of the compressed file.

$$compressionFactor = \frac{sizeaftercompression}{sizebeforecompression}$$

Compression Factor = 44/133

= **0.3308**

C. Reduction Percentage

It is the value that how much amount of input data is compressed when compared with original input data.

$$Reduction\ \% = \left(1 - \frac{Size\ after\ compression}{Size\ before\ compression}\right) * 100$$

(Or)

$$Reduction\ \% = (1 - compression\ factor) * 100$$

Compression factor from above = 0.3308

Reduction % = (1- 0.3308) * 100

= 0.6692 * 100

= **66.92 %**

VI. ADVANTAGES

The benefits of proposed LZW dictionary based technique for text data:

A. LZW compression works best for files containing lots of repetitive data. This is often the case with text & monochrome images.
B. Files that are compressed but that do not contain any repetitive information at all can even grow bigger.
C. LZW technique is fast
D. High compression ratio
E. It can be used in TIFF files, GIF files and PDF files too.

VII. CONCLUSION

Text data Compression is a topic of much importance and has many applications in communications, we have designed lossless LZW compression and decompression algorithms for text data using System verilog to ease the description, verification and simulation based on the DUT. We get the same output text which was provided at the input without any loss which was the major advantage of LZW algorithm that data remains unchanged. The algorithm is adaptive because it will add new strings to the dictionary. Implementation of this dictionary based algorithm produced better results in compression ratio, reduction of data and the better efficiency. The proposed LZW data compression and decompression technique was designed which provides the compression factor of 0.33 and reduction % of 66.9 %.

As a future work more focus on improvement of compression ratio, which obtained by reducing memory wastage, using Finite State Machine and Hardware implementation.

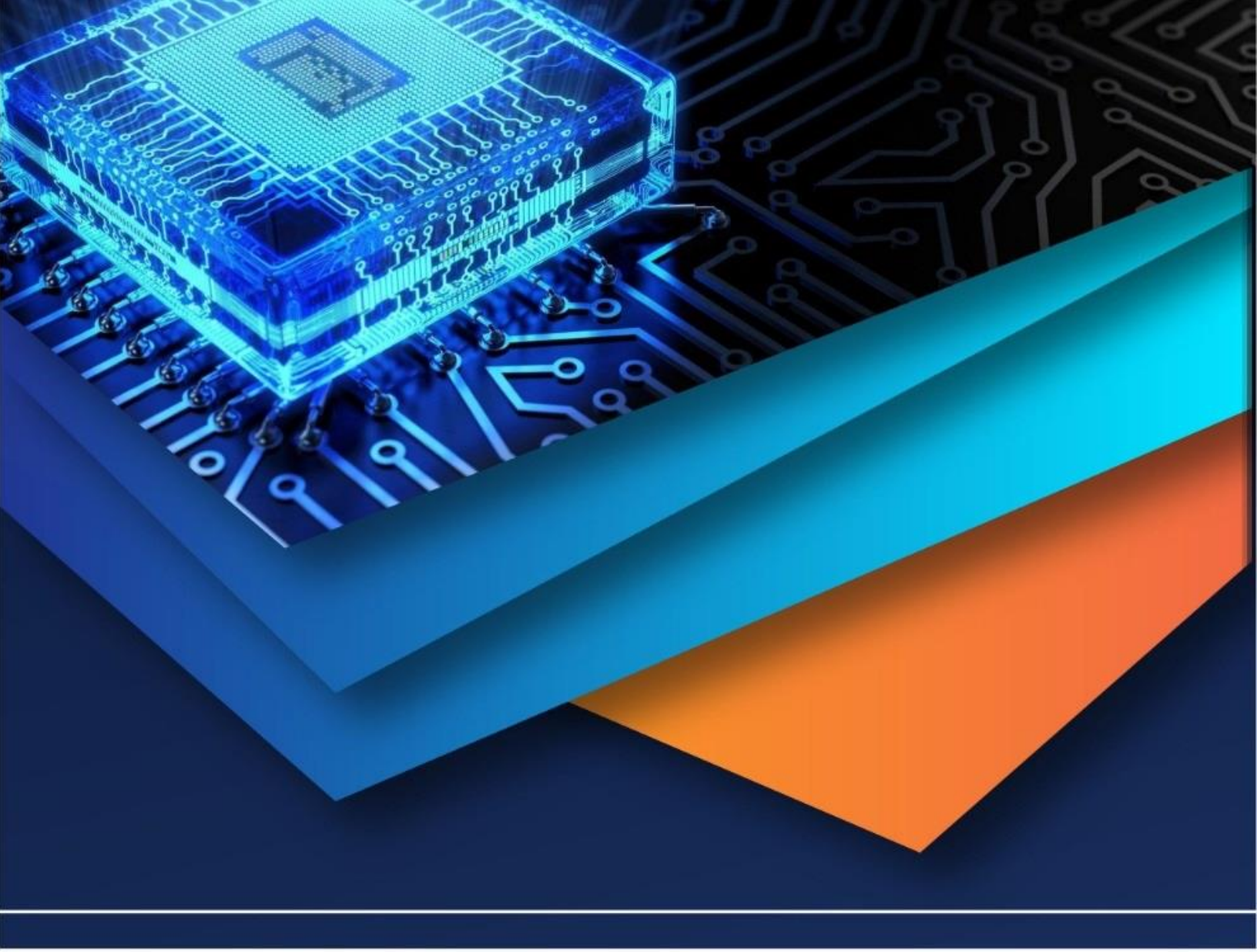
VIII. ACKNOWLEDGMENT

This research was supported by Dr. G. L. Madhumathi, HOD of Electronics and communication department and Mrs. J Sushma, Assistant Professor, Department of Electronics & Communication engineering, Dhanekula Intitute of Engineering and Technology. We are very thankful to Nagarjuna.Gattu employe of ECIL, Hyderabad who provided expertise that greatly assisted the research, although they may not agree with all of the interpretations provided in this paper. We have to express out appreciation to the Nikhil G and Nithisha B for sharing their pearls of wisdom with us during the course of this research.

CATALOGUE

[1] J. Ziv and A. Lempel, “A Universal Algorithm for Sequential Data Compression,” IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337–343, 1977.
[2] ——, “Compression of Individual Sequences via Variable- Rate Coding,” IEEE Transactions on Information Theory,vol. 24, no. 5, pp. 530–536, 1978.
[3] T. A. Welch, “A technique for high-performance data compression,”IEEE Computer, vol. 17, no. 6, pp. 8–19, June 1984.
[4] Adobe Developers Association, TIFF Revision 6.0, June 1992. [Online]. Available: http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf
[5] K. Nakano and Y. Ya78//magishi, “Hardware n Choose k Counters with Applications to the Partial Exhaustive Search,” IEICE Transactions on Information & Systems, 2005.
[6] Xilinx Inc., 7 Series FPGAs Memory Resources User Guide, Nov. 2014.
[7] K. Nakano and E. Takamichi, “An Image Retrieval System using FPGAs,” IEICE Transactions on Information and Systems, vol. 86, no. 5, pp. 811–818, 2003.
[8] X. Zhou, Y. Ito, and K. Nakano, “An Efficient Implementation of the Hough Transform using DSP slices and block RAMs on the FPGA,” in Proceedings of IEEE 7th International Symposium on Embedded Multicore Socs (MCSoC), 2013, pp. 85–90.
[9] Y. Ago, K. Nakano, and Y. Ito, “A Classification Processor for a Support Vector Machine with embedded DSP slices and block RAMs in the FPGA,” in Proceedings of IEEE 7th International Symposium on Embedded Multicore Socs(MCSoC), 2013, pp. 91–96.

[10] X. Zhou, Y. Ito, and K. Nakano, "An Efficient Implementation of the Gradient-based Hough Transform using DSP slices and block RAMs on the FPGA," in Proceedings of International Parallel and Distributed Processing Symposium Workshops, 2014, pp. 762–770.
[11] K. Hashimoto, Y. Ito, and K. Nakano, "Template Matching using DSP slices on the FPGA," in Proceedings of International Symposium on Computing and Networking (CANDAR), 2013, pp. 338–344.
[12] X. Zhou, Y. Ito, and K. Nakano, "An Efficient Implementation of the One-Dimensional Hough Transform Algorithm for Circle Detection on the FPGA," in Proceedings of International Symposium on Computing and Networking (CANDAR), 2014, pp. 447–452.
[13] Helion Technology, LZRW3 Data Compression Core for Xilinx FPGA, October 2008.
[14] S. Navqi, R. Naqvi, R. A. Riaz, and F. Siddiqui, "Optimized RTL design and implementation of LZW algorithm for high bandwidth applications," PRZEGLAD ELEKTROTECHNICZNY (Electrical Review), vol. 4, pp. 279–285, 2011.
[15] M. Lin, "A hardware architecture for the LZW compression and decompression algorithms based on parallel dictionaries," Journal of VLSI signal processing systems for signal, image and video technology, vol. 26, no. 3, pp. 369–381, 2000.
[16] M. Lin, J. Lee, and G. E. Jan, "A Lossless Data Compression and Decompression Algorithm and Its Hardware Architecture," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14, no. 9, pp. 925–936, 2006.
[17] S. Prakash, M. Purohit, and A. Raizada, "A novel approach of speedy-highly secured data transmission using cascading of PDLZW and arithmetic coding with cryptography," International Journal of Computer Applications, vol. 57, no. 19, 2012.
[18] S. Funasaka, K. Nakano, and Y. Ito, "A parallel algorithm for LZW decompression, with GPU implementation," in to appear in Proc. of International Conference on Parallel Processing and Applied Mathematics, 2015.
[19] K. Shyni and K. V. M. Kumar, "Lossless LZW data compression algorithm on CUDA," IOSR Journal of Computer Engineering, pp. 122–127, 2013.
[20] S. T. Klein and Y. Wiseman, "Parallel Lempel Ziv coding," Discrete Applied Mathematics, vol. 146, no. 2, pp. 180–191, 2005.
[21] M. K. Mishra, T. K. Mishra, and A. K. Pani, "Parallel Lempel-Ziv-Welch (PLZW) technique for data compression," International Journal of Computer Science and Information Technologies, vol. 3, no. 3, pp. 4038–4040, 2012.
[22] Xilinx Inc., 7 Series FPGAs Configuration User Guide, 2013.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY