ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 4, April 2014

FPGA Based Implementation of Data Compression using Dictionary based "LZW"Algorithm

Ms. Agrawal Arohi K[1], Prof. V. S. Kulkarni[2],

PG Student, E & TC Department, SKNCOE, Vadgaon Bk, Pune, India

Professor, E & TC Department, SKNCOE, Vadgaon Bk, Pune, India

Abstract: Field Programmable Gate Array (FPGA) technology has become a viable target for the implementation of algorithms in different compression methods applications. In a distributed environment, large data files remain a major bottleneck. Compression is an important component of the solutions available for creating file sizes of manageable and transmittable dimensions. When high-speed media or channels are used, high-speed data compression is desired. Software implementations are often not fast enough. In this paper, we present the very high speed hardware description language (VHDL) modeling environment of Lempel-Ziv-Welch (LZW) algorithm for binary data compression to ease the description, verification, simulation and hardware realization. The LZW algorithm for binary data compression comprises of two modules compressor and decompressor. The input of compressor is 1-bit bit stream read in according to the clock cycle. The output is an 8-bit integer stream fed into the decompressor, which is an index that represents the memory location of the bit string stored in the dictionary. In this paper, data compression technique is described using Lempel-Ziv-Welch algorithm. Software reference model for data compression using LZW has been modelled in MATLAB/ Simulink.

Keywords: Binary Data Compression, LZW, Lossless data compression, VHDL Simulation.

I. INTRODUCTION

Compression is the art of representing information in a compact form rather than its original or uncompressed form [1]. The main objective of data compression is to find out the redundancy and eliminate them through different efficient methodology; so that the reduced data can save, space: to store the data, time: to transmit the data and cost: to maintain the data. To eliminate the redundancy, the original file is represented with some coded notation and this coded file is known as 'encrypted file'. For any efficient compression algorithm this file size must be less than the original file. To get back the original file we need to 'decrypt' the encoded file. Data compression has an undeserved reputation for being difficult to master, hard to implement, and tough to maintain. In fact, the techniques used in the previously mentioned programs are relatively simple, and can be implemented with standard utilities taking only a few lines of code. This article discusses a good all-purpose data compression technique: Lempel-Ziv-Welch, or LZW compression. The original Lempel Ziv approach to data compression was first published in 1977, followed by an alternate approach in 1978. Terry Welch's refinements to the 1978 algorithm were published in 1984. The algorithm is surprisingly simple. In a nutshell, LZW compression replaces strings of characters with single codes.

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 routines shown here belong in any programmer's toolbox. For example, a program that has a few dozen help screens could easily chop 50K bytes off by compressing the screens. Or 500K bytes of software could be distributed to end users on a single 360K byte floppy disk. Highly redundant database files can be compressed down to 10% of their original size. Once the tools are available, the applications for compression will show up on a regular based LZW 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 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 refer to substrings.

II. LITERTAURE SURVEY

Compression is of two type Lossless compression and Lossy compression

A. Types of compression methods

Compression is of two type Lossless compression and Lossy compression. These two types of compression technique is explain below

1) Lossless Compression

In the process compression if no data is lost and the exact replica of the original file can be retrieved by decrypting the encrypted file then the compression is of lossless compression type. Text compression is generally of lossless type. In this type of compression generally the input file is encrypted first which is used for storing or

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 4, April 2014

transmitting data, and then at the output the file is decrypted to get the original file.

2) Lossy compression

Data compression is contrasted with lossless data compression. In this technique some data is loss at the output which is acceptable. Generally, lossy data compression schemes are guided by research on how people perceive the data in question. For example, the human eye is more sensitive to subtle variations in luminance than it is to variations in colour. There is a corresponding trade-off between information lost and the size reduction

B. Lossless Compression methods

The lossless compression algorithm is divided into following two coding techniques

1) Entropy Based Encoding

In this compression process the algorithm first counts the frequency of occurrence of each unique symbol in the given text. And then it is replaced by the unique symbol generated by the algorithm. The frequency of the symbols varies with respect to the length in the given input file.

a) Huffman coding

Is a technique which is entropy based and which reduces the average code length of symbols in an alphabet such as a human language alphabet or the data-coded alphabet ASCII. Huffman coding has a prefix property which uniquely encodes symbols to prevent false interpretations when deciphering the encoded symbols, such that an encoded symbol cannot be a combination of other encoded symbols. The binary tree that is used to store the encoded symbols is built bottom up, compared to the Shannon-Fanon tree which is built top down [6, 44, 48, and 57]. When the probability of each symbol appearing in the data stream has been determined, they are sorted in order of probability. The two symbols with the lowest probability are then grouped together to form a branch in the binary tree. The branch node is then assigned the combined probability value of the pair. This is iterated with the two nodes with the lowest probability until all nodes are connected and form the binary tree. After the tree has been completed each edge is either assigned the value one or zero depending on if it is the right or left edge. It does not matter if the left edge is assigned one or zero as long as it is consistent for all nodes. The code for each symbol is then assigned by tracing the route, starting from the root of the tree to the leaf node representing the symbol. In this way we have explain the Huffman coding technique in which coding is done with respect to probability of occurrence of the character.

b) Shannon fanon encoding

It works in the same way as Huffman coding except that the tree is built top down. When the probability of each symbol has been determined and sorted the symbols are split in two subsets which each form a child node to the root. The combined probability of the two subsets should be as equal as possible. This is iterated until each subset only contains one symbol. The code is created in the same way as with Huffman codes. Due to the way the Shannon-Fanon tree is built it does not always produce the optimal result when considering the average code length

c) Arithmetic Coding

IBM is the one who has developed the arithmetic coding technique and then it uses this compression technique for their mainframes. Arithmetic coding is the best entropy coding method where high compression ratio is needed. Arithmetic encoding technique gives better results than Huffman encoding. But the disadvantage is that it is complicated then other compression technique. In this the input data is a single rational number between 0and 1 rather then splitting the probabilities of symbols into a tree. Then, it is further transformed into a fixed-point binary number which is the encoded result. The value can be decoded into the original output by changing the base from binary back to the original base and replacing the values with the symbols they correspond to.

A design flow for arithmetic coding is as follows:

1. Calculate the number of unique symbols in the input. This number represents the base b (e.g. base 2 is binary) of the arithmetic code.
2. Assign values from 0 to b to each unique symbol in the order they appear.
3. Take the value from the upper step and assign their code to it.
4. Convert the result from step 3 from base b to a sufficiently long fixed-point binary number to preserve precision.
5. Record the length of the input string somewhere in the result as it is needed for decoding

C. The Dictionary based technique is given below

Substitution encoding is another name for dictionary based encoding process. A data structure known as Dictionary is maintained throughout the process [3]. This data structure consists of no of string. The encoder matches the substrings chosen from the original text and finds it in the dictionary; if a successful match is found then the substring is replaced by a reference to the dictionary in the encoded file .Lempel-Ziv string encoding creates a dictionary of encountered strings in a data stream. At first the dictionary only contains each symbol in the ASCII alphabet where the code is the actual ASCII-code. Whenever a new string appears in the data stream, the string is added to the dictionary and given a code. When the same word is encountered again the word is replaced with the code in the outgoing data stream. If a compound word is encountered the longest matching dictionary entry is used and over time the dictionary builds up strings and their respective codes. In some of the Lempel Ziv algorithms both the compressor and decompressor needs construct a dictionary using the exact same rules to ensure that the two dictionaries match.

1) Suitable LZ algorithms

In this section some suitable algorithms for our implementation will be described. These are the algorithms that were most often used in the studies we

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 4, April 2014

analyzed. Figure 1 shows how some of the algorithms described in this section are related to each other and when they were developed.

a) LZ77

LZ77 is the dictionary-based algorithm developed by Lempel and J. Ziv in 1977 [3]. This algorithm uses dictionary based on a sliding window of the previously encode characters. The output of the algorithm is a sequence of triples containing a length l; an o set o and the next symbol c after the match. If the algorithm cannot match, l and o will both be set to 0 and c will be the next symbol in the stream. Depending on the size of l and o, the compression ratio and memory usage will be affected. If o is small, the number of characters will be small, reducing the size of the sliding window and in turn the memory usage.

b) LZ78

The LZ78 algorithm was presented by A. Lempel and J. Ziv in 1978 [8]. Like LZ77, it is a dictionary but with LZ78 the dictionary may contain strings from anywhere in the data. The output is a sequence of pairs containing an index i and the next non-matching symbol c. The memory usage of LZ78 might be more unpredictable than that of LZ77, since the size of the dictionary has no upper limit in LZ78, though there are ways to limit it. A benefit of LZ78 compared to LZ77 is the compression speed. The below fig gives the following no of LZ77 based algorithm.

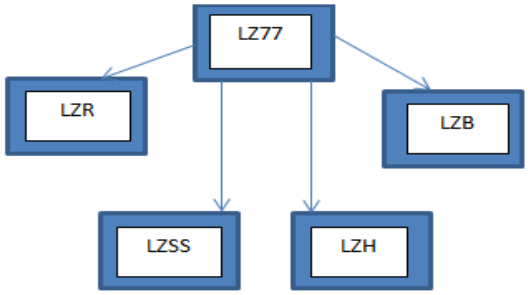


Fig. 1. The Lempel Ziv LZ77 algorithm family

i) LZR

The LZR is an algorithm based on LZ77 and its modification allows pointers to reference anything that has already been encoded without being limited by the length of the search buffer (window size exceeds size of expected input).

ii) *LZSS*

LZSS is an algorithm based on LZ77 that was developed by J. Storer and T. Szymanski in 1982. Like LZ77, LZSS also uses a dictionary based on a sliding window. When using LZSS, the encoded consists of a sequence of characters and pointers. Each pointer consists of an o set and a length l. The position of the string is determined by o and l determines the length of the string.

iii) *LZB*

LZB compression technique uses lengths which vary with the size of the given file.

iv) *LZH*

The LZH implementation employs Huffman coding to compress the pointers.

The below diagram gives detail explanation of different algorithm based on LZ78 compression technique.

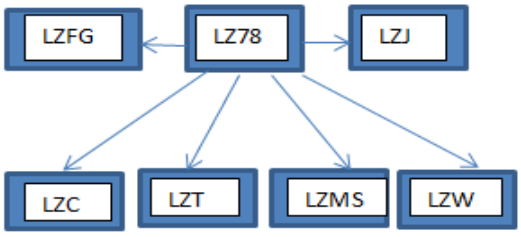


Fig. 2. The LZ78 algorithm family

i) *LZT*

This is another algorithm for data compression little bit different form the LZW and slightly similar to LZC the only variation is that it provides place for new phrases that can be added to the dictionary.

ii) LZMS

LZMS creates new dictionary entries not by appending the first non-matching character, but by concatenating the last two. The LZR is an algorithm based on LZ77 and its modification allows pointers to reference anything that has already been encoded without being limited by the length of the search buffer (window for hardware solution is not constant.

iii) LZJ

The dictionary used by LZJ contains a no of unique strings which is coded by fixed length. Once the dictionary is full, all strings that have only been used once are removed. This is continued until the dictionary becomes static

iv) LZFG

LZFG uses the dictionary building technique from the original LZ78 algorithm, but stores the elements in a tie data structure

v) LZC

LZC was developed from LZW and is used in the UNIX utility compress. Its code begins at 9 bits, and doubles the size of the dictionary when all 9-bit codes have been used by increasing the code size to 10 bits. This is repeated until all 16-bit codes have been used at which point the dictionary becomes static .If a decrease in compression ratio is detected, the dictionary is discarded. In the study by K. Barr, they found that compress was relatively balanced compared to the other algorithms, not being the best or worst in any of the categories. The data used for compression in this study was 1 MB of the Calgary corpus and 1MB of common web data.

vi) LZW

This improved version of the original LZ78 algorithm is perhaps the most famous modification and is sometimes even mistakenly referred to as Lempel Ziv algorithm. The

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 4, April 2014

LZ-Algorithm in 1984 was published by Terry Welch ; it basically applies the LZSS principle of not explicitly transmitting the next non matching symbol to the LZ78 algorithm. The only remaining outputs of this improved algorithm are fixed-length references to the dictionary (indexes). Of course, we can't just remove all symbols from the output. Therefore the dictionary has to be initialized with all the symbols of the input alphabet and this initial dictionary needs to be made known to the decoder.

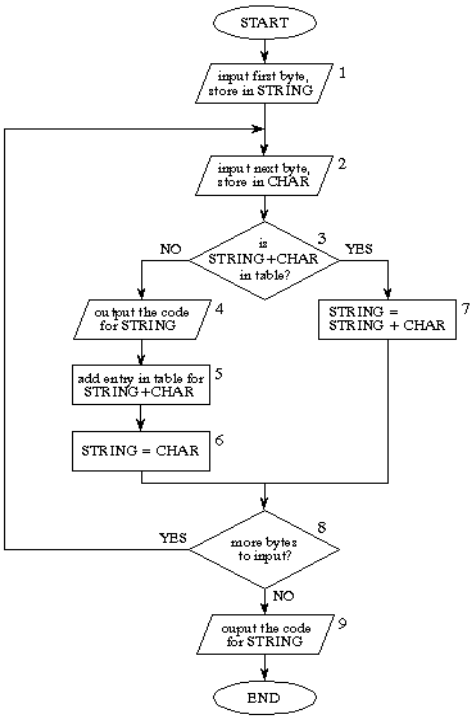
III. PROPOSED LZW METHOD

In this paper, data compression is described using Lempel Ziv Welch based technique called as LZW. Data compression is basically a process in which the data is provided in compact form rather then original form or uncompressed form combines [1]. This chapter discussed the LZW algorithm. Lempel-Ziv-Welch proposed a variant of LZ78 algorithms; in which compressor always give a code as an output instead of a character. To do this, a major change in LZW is to preload the dictionary with all possible symbols that can occur. LZW compression replaces string of characters with codes. LZW is a dictionary based lossless data compression algorithm and its detail implementation is shown. There are many algorithms which have been used for data compression like Huffman and Lempel-Ziv-Welch (LZW), arithmetic coding. LZW algorithm is the most popular algorithm. LZW algorithm is just like a greedy approach and divides text into substrings. Like the LZW algorithm proposed in [2]. LZW algorithm has both compression and decompression techniques which is explained as below.

A. LZW algorithm for compression and decompression

LZW algorithm is a lossless dictionary based algorithm. In which an index number is provided for each symbol, character, and special characters. Firstly the input text file which we has to compressed is read from the stored file .The stored file is stored as "message.text"and every time we want to read the input we have to call this file. Then a new dictionary is formed and stored which consist of each characters and symbols from the keyboard starting from 0-255.Intially the data is enter in the storage place to see where it is present in the dictionary or not and if yes then to generate its code. And if the same character is not found in the dictionary it will add it as a new character and then assign to it a code. And if the character if already present in the dictionary its index number is provide by the dictionary. This technique is generally used for text compression as at the end of process no data is lost. The compression algorithm consists of two variables which are CHAR and STRING. The variable CHAR is generally used to define a single character having single byte from 0-255 while the variable STRING is used to define a group of more then one character. The algorithm starts as flows .Firstly we have to take the first byte from the input text file and stored it in the variable string as shown in the below given example where the first character is "A". This same procedure is followed every time when a new character appears. Each time new character appears is stored in the variable CHAR. Then it is seen that whether the combination of STRING+CHAR is present in the dictionary or not. If it is present then it is given index number and if it is not present then it is added in the dictionary in variable STRING. And the dictionary obtains a new CHAR but with a single byte. Example of this combination is as below in the example it will take "AB" AS A STRING+CHAR and then stored in a dictionary as a new variable and it will then get a index number or code as "AB=256".Then from the given below table its is seen that the 6th character is "C". so know the knew obtain variable is "ACC" which then added to the dictionary having index code as 261.And in this way the longest sequence is consider. This flow of algorithm is continued until all the characters in the input file are added in the dictionary.

The decompressor algorithm consists of four variables which are NCODE, OCODE, and STRING AND CHAR. In decompression algorithm the first byte of the compressed text file is first placed in the OCODE. Then combination is consider of two variable if the combination is found in the dictionary then it is stored as STRING=OCODE and if the match is not found then STRING =NCODE. How it works and explains the flow required for compression.



B. LZW Decompression Algorithm

In LZW decompression algorithm, the index number which is consider as a code is taken at the output and then use this code to get the original input file by assign the characters which are placed at that index value. Decompression algorithm is shown as: For example if NCODE and OLDCODE+CHAR is „ACCCD" then the absence of the pattern is returned after comparing all the elements in the dictionary (shown in table 1 line number - 16 for encoding and table-2 line number 10 for decoding)..

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN ELECTRICAL, ELECTRONICS, INSTRUMENTATION AND CONTROL ENGINEERING
Vol. 2, Issue 4, April 2014

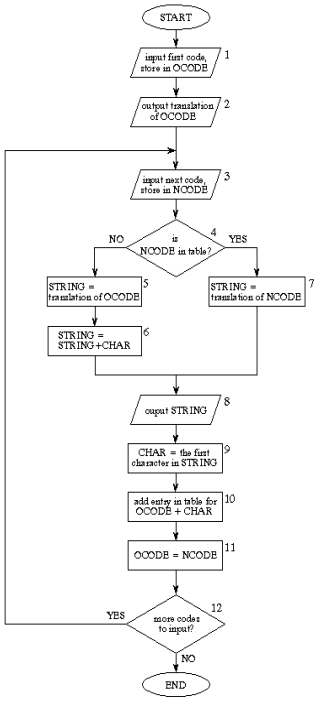


Fig .4.Flow design for decompression algorithm

VI. RESULTS & DISCUSSIONS

LZW algorithm is simulated in MATLAB 7.14 and an input text is used as shown below .Firstly a dictionary is initialised in which all the symbols and characters present on our keyboard is stored first and then the characters from the input file is compared with the characters stored in the dictionary this is repeated for each character .Then a string is taken which is again compare with the existing dictionary and if the string is not present it is added and new word are formed.

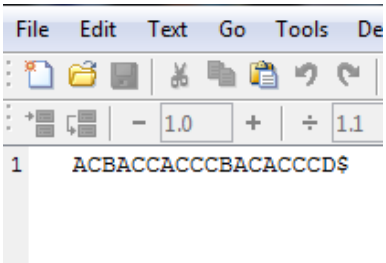


Fig. 3. Input text

The given input text message is as shown above.
After the input text is read .A new dictionary is formed, first by storing every symbol and character from the keyboard from 0-255.The input text after compression is as shown below. In this firstly the character is consider.

```
  '>'    '?'    '@'    'A'    'B'    'C'    'D'    'E'    'F'

Columns 72 through 80

  'G'    'H'    'I'    'J'    'K'    'L'    'M'    'N'    'O'

Columns 81 through 89

  'P'    'Q'    'R'    'S'    'T'    'U'    'V'    'W'    'X'

Columns 90 through 98

  'Y'    'Z'    '['    '\'    ']'    '^'    '_'    '`'    'a'

Columns 99 through 107
```

Fig. 5. Input text when compressed

The output is as shown below fig.5 after compression which shows us that no data is lost .We obtain the output as it is given in the input only the bytes as been reduced.

```
    'Þ'
    'ÿ'
    'AC'
    'CB'
    'BA'
    'ACC'
    'CA'
    'ACCC'
    'CBA'
    'ACA'
    'ACCCD'

received original string=
ACBACCACCCBACACCCD
>>
```

Fig. 5. Output obtain

In this way we shows how LZW algorithm is better for text compression as no data is lost .And we obtained the original input file.

V. CONCLUSIONS

An comparison of a number of different dictionary based lossless compression algorithms for text data is given. Several existing lossless compression methods are compared for their effectiveness. Although they are tested on bases of compression speed because lossless algorithm have better compression speed. By considering the compression ratio, the LZW algorithm may be considered as the most efficient algorithm among the selected ones. Those values of this algorithm are in an acceptable range and it shows better results. And we get the exactly same output text which was provide as an input without any loss this an advantage of LZW algorithm that after compression still the original file is obtained

REFERENCES

[1] Pu, I.M., Elsevier, Britain, “Fundamental Data Compression”, 2006.
[2] Blelloch E. “Introduction to Data Compression”, Computer Science Department, Carnegie Mellon University.2002.
[3] Kesheng, W., J. Otoo and S. Arie, “Optimizing bitmap indices with efficient compression”, ACM Trans Database Systems, 2006, pg no 31: 1-38.
[4] Kaufman, K. and T. Shmuel, “Semi-lossless text compression”, Intl. J. Foundations of Computer Science,2005,pg no1167-1178.
[5] Vo Ngoc and M. Alistair, “Improved word aligned binary compression for text indexing”, IEEE Trans. Knowledge & Data Engineering,2006, pg no 857-861.
[6] Cormak, V. and S. Horspool,“Data compression using dynamic Markov modeling,” Comput. J,1987, pg no 541–550.
[7] Capocelli, M., R. Giancarlo and J. Taneja, “Bounds on the redundancy of Huffman codes”, IEEE Trans. Inf. Theory,1986
[8] Gawthrop, J. and W. Liuping “Data compression for estimation of the physical parameters of stable and unstable linear systems”, Automatica,2005, pg no1313-1321.
[9] Bell,T.C.,Clearly, J. G., AND Witten, I. H. Text Compression. Prentice Hall, Upper Sadle River, NJ .