International Journal of Research (IJR) Vol-1, Issue-10 November 2014 **ISSN 2348-6848**

Lossless Implementation of NAND Flash Memory Architecture Using MERGE Scheme

Kota N V Tulasi Kumari[1], Y. Phani Kumar[2]

[1](PG Scholar, Department of ECE, ASIST, Vijayawada, AP, India)

Email: tulasikumari@rocketmail.com

[2] (Assistant Professor, Department of ECE, ASIST, Vijayawada, AP, India)

ABSTRACT

Now a days, FLASH memories plays leading role in non-volatile memories. In general, these are used in storage devices such as memory cards, USB flash drives, and solid-state drives etc. NAND Flash memories have lower erase times, less chip area per cell which allows greater storage density. Multi-level cell (MLC) Flash memories store two or more bits per cell by supporting four or more voltage states and provide greater storage density. For loss less transmission, error detection and correction scheme is used which enables reliable delivery of digital data over unreliable communication channels. Generally errors are caused by noise or other impairments during transmission. Here, error correction scheme is implemented by completely removing of redundancy bits by encoding technique and retrieving those at the other end using decoding method. For this error correction, merge scheme is used which often an advantageous of both column and row action in the same algorithm. Hence the power consumption required will reduces drastically. For this implementation, VHDL Code is used.

Key words- *error correction codes (ECCs), multi-level cell, , flash memories, merge scheme.*

1. INTRODUCTION

In present days, FLASH memories have become the dominant technology for non-volatile memories. Basic applications may include in memory cards, USB flash drives, and solid-state drives in application platforms such as personal digital assistants, laptop computers, digital audio players, digital cameras and mobile phones. In this paper, NAND Flash memories are taken in to consideration due to lower erase times, less chip area per cell which allows greater storage density, and lower cost per bit than NOR Flash memories [2]. Specifically, focus towards on multi-level cell (MLC) Flash memories which store two or more bits per cell by supporting four or more voltage states.

Electronic space provided by silicon chips (semiconductor memory chips) or magnetic/optical media as temporary or permanent storage for data and/or instructions to control a computer or execute one or more programs. Two main types of computer memory are: (1) Read only memory (ROM), which is smaller part of a computer's silicon (solid state) memory that is fixed in size and permanently stores manufacturer's instructions to run the computer when it is switched on. (2) Random access memory (RAM), which is larger part of a computer's memory comprising of hard disk, CD, DVD, floppies etc., (together called secondary storage) and employed in running programs and in archiving of data. Memory chips provide access to stored data or instructions that is hundreds of times faster than that provided by secondary storage. Memory errors are of two types, namely hard and soft.

- Hard errors are caused due to fabrication defects in the memory chip and cannot be corrected once they start appearing.
- Soft errors on the other hand are caused predominantly by electrical disturbances

2. ERROR CONTROL CODING

Error detection and correction or error control coding are techniques that enable reliable delivery of digital data over unreliable communication channels (or storage medium).

Error detection is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver. Error correction is the detection of errors and reconstruction of the original, error-free data. The goal of error control coding is to encode information in such a way that even if the channel (or storage medium) introduces errors, the receiver can correct the errors and recover the original transmitted information.

ECC stands for "Error Correction Codes" is a method used to detect and correct errors introduced during storage or transmission of data. Certain kinds of RAM chips inside a computer implement this technique to correct data errors and are known as ECC Memory.

An error-correcting code (ECC) or forward error correction (FEC) code is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors (up to the capability of the code being used) were introduced, either during the process of transmission, or on storage. Since the receiver does not have to ask the sender for retransmission of the data, a back-channel is not required in forward error correction, and it is therefore suitable for simplex communication such as broadcasting

Error-correcting codes are usually distinguished between convolutional codes and block codes:

Convolutional codes: These are processed on a bit-by-bit basis. They are particularly suitable for implementation in hardware, and the Viterbi decoder allows optimal decoding.

Block codes: These are processed on a block-by-block basis. Examples of block codes are repetition codes, Hamming codes and multidimensional parity-check codes. They were followed by a number of efficient codes, Reed-Solomon codes being the most notable due to their current widespread use. Turbo codes and low-density parity-check codes (LDPC) are relatively new constructions that can provide almost optimal efficiency.

Shannon's theorem is an important theorem in forward error correction, describes the maximum information rate at which reliable communication is possible over a channel that has a certain error probability or signal-to-noise ratio (SNR). This strict upper limit is expressed in terms of the channel capacity. More specifically, the theorem says that there exist codes such that with increasing encoding length the probability of error on a discrete memory less channel can be made arbitrarily small, provided that the code rate is smaller than the channel capacity. The code rate is defined as the fraction k/n of k source symbols and n encoded symbols. Some simple codes can detect but not correct errors; others can detect and correct one or more errors.

2.1 Parity Checking

Parity checking is the simple way to detect errors, but this simple check does have two limitations: it only detects errors, without being able to correct them; and it can't detect errors that invert an even number of bits.

2.2 Hamming Codes

Hamming codes are an extension of parity checking method that can be used to detect and correct a larger set of errors. Hamming's development is a very direct construction of a code that permits correcting single-bit errors.

Let us assume that the data to be transmitted consists of a certain number of information bits u and added them to a number of check bits *p* such that if a block is received that has at most one bit in error, then *p* identifies the bit that is in error (which may be one of the check bits). Specifically, in Hamming's code *p* is interpreted as an integer which is 0 if no error occurred, and otherwise is the 1-origined index of the bit that is in error. Let *k* be the number of information bits, and *m* the number of check bits used. Because the *m* check bits must check themselves as well as the information bits. Because *m* bits can distinguish cases, we must have

$$2^m \geq m + k + 1 \qquad (1)$$

Equation (1) applies to any single error correcting (SEC) binary FEC block code in which all of the transmitted bits must be checked. The check bits will be interspersed among the information bits in a manner described below. Because p indexes the bit (if any) that is in error, the least significant bit of p must be 1 if the erroneous bit is in an odd position, and 0 if it is in an even position or if there is no error. A simple way to achieve this is to let the least significant bit of p, be an even parity checks on the odd positions of the block, and to put in an odd position. The receiver then checks the parity of the odd positions. If the result is 1, an error has occurred in an odd position, and if the result is 0, either no error occurred or an error occurred in an even position. This satisfies the condition that p should be the index of the erroneous bit, or be 0 if no error occurred.

Similarly, let the next from least significant bit of p, be an even parity check of positions 2, 3, 6, 7, 10, 11, … (in binary, 10, 11, 110, 111, 1010, 1011, …), and put in one of these positions. Those positions have a 1 in their second from least significant binary position number. The receiver checks the parity of these positions. If the result is 1, an error occurred in one of those positions, and if the result is 0, either no error occurred or an error occurred in some other position. Continuing, the third from least significant check bit, , is made an even parity check on those positions that have a 1 in their third from least significant position number, namely positions 4, 5, 6, 7, 12, 13, 14, 15, 20, …, and is put in one of those positions.

Putting the check bits in power-of-two positions (1, 2, 4, 8 …) has the advantage that they are independent. That is, the sender can compute independently and more generally, it can compute each check bit independently of the others.

This code is called the (7, 4) Hamming code, which signifies that the code length is 7 and the number of information bits is 4. The positions of the check bits and the information bits are shown below.

1 2 3 4 5 6 7

2.3 Reed-Solomon Codes

Reed-Solomon codes were developed in 1960 by Irving S. Reed and Gustavo Solomon, who were then members of MIT Lincoln Laboratory. Their seminal article was entitled "Polynomial Codes over Certain Finite Fields." (Reed & Solomon 1960) When the article was written, an efficient decoding algorithm was not known Reedsolomn code [17, 18]. A solution for the latter was found in 1969 by Elwy Berlekamp and James Massey, and is since known as the Berlekamp-Massey decoding algorithm. In 1977, RS codes were notably implemented in the Voyager program in the form of concatenated codes. The first commercial application in mass-produced consumer products appeared in 1982 with the compact disc, where two interleaved RS codes are used. The parameters of a Reed-Solomon code are:

m = the number of bits per symbol
n = the block length in symbols
k = the uncoded message length in symbols
(n-k) = the parity check symbols (check bytes)
t = the number of correctable symbol errors
(n-k)=2t (for n-k even)
(n-k)-1 = 2t (for n-k odd)
Therefore, an RS code may be described as an (n, k) code for any RS code.

Consider the RS (255,235) code, the encoder groups the message into 235 8-bit symbols and adds 20 8-bit symbols of redundancy to give a total block length of 255 8-bit symbols. In this case, 8% of the transmitted message is redundant data. In general, due to decoder constraints, the block length cannot be arbitrarily large.

The Hamming and Hsiao codes, for example have low encoding and decoding complexity, but also have relatively low error- correcting capacity (e.g., Hamming is single error-correcting, double error-detecting). To achieve higher error- correcting capability, codes like Reed Solomon or BCH require more sophisticated decoding algorithms.

2.4 LDPC Codes

Low-density parity-check (LDPC) codes were first discovered by Gallagher in the early 1960s and have recently been rediscovered and generalized .It has been shown that these codes achieve a remarkable performance with iterative decoding that is very close to the Shannon limit. Consequently, these codes have become strong competitors to turbo codes for error control in many communication systems and digital systems where high reliability is required.

LDPC codes can be constructed using random or deterministic approaches. In this report, we focus on a class of LDPC codes known as Euclidean Geometric (EG) LDPC codes, which are constructed deterministically using the points and lines of a Euclidean geometry [1, 16]. The EG LDPC codes that we consider are cyclic and consequently their encoding can be efficiently implemented with linear shift registers. Minimum distances for EG codes are also reasonably good and can be derived analytically. Iteratively decoded EG LDPC codes also seem to not have the serious error- floors that plague randomly-constructed LDPC codes; this fact can be explained by the observation made in that EG LDPC codes do not have pseudo-code words of weight smaller than their minimum distance. For these reasons, EG LDPC codes are good candidates for use in applications like optical communications that require very fast encoders and decoders and very low bit error-rates.

However combinational logic has already started showing susceptibility to soft errors, and therefore the encoder and decoder (corrector) units will no longer be immune from the transient faults. Therefore, protecting the memory system support logic implementation is more important. Here we proposed a lossless system which detect the errors and correct the errors on 32bit input at a time to maintain the 32 bit processor with error free. Generally the errors occur in the memory is due to redundancy information .So if we remove the redundancy obviously the error prone will be reduced. In this project based on the proposed technique error free data oriented architecture is developed to encode and to decode the information.

With the increase in silicon densities, it is becoming feasible for multiple compression systems to be implemented in parallel onto a single chip. A 32-BITsystem with distributed memory architecture is based on having multiple 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 parallel data compression system which operates in high-speed to achieve high compression rate. By using Parallel architecture of compressors, the data compression rates are significantly improved. Also inherent scalability of parallel architecture is possible. The main parts of the system are the two Xmatchpro based data compressors in parallel and the control blocks providing control signals for the Data compressors, allowing appropriate control of the routing of data into the system.

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

3. METHODOLOGY

A second Lempel-Ziv method used a content addressable memory (CAM) capable of performing a complete dictionary search in one clock cycle. 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 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. Its 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. 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 parallel hardware would greatly increase the complexity of the design and it is likely that these aspects would need to be implemented sequentially. 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, 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 and Simpson and Sabharwal. 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 Store.

3.1 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.

3.2 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] . It uses a tag attached to each dictionary location to identify which node should be eliminated once the dictionary becomes full.

3.3. XMATCHPRO Based System

The Lossless data compression system is derivative of the XMatchPro Algorithm 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. 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.

3.4 Proposed Method

In it discusses about the Parallel Algorithm that can be implemented for High Speed Data Compression. The authors gives the basic idea about how the Data Compression is carried out using the Lempel-Ziv Algorithm and how it could be altered for Parallelism of the algorithm. It describes the Lempel-Ziv algorithm as a very efficient universal data compression technique, based upon an incremental parsing technique, which maintains codebooks of parsed phrases at the transmitter and at the receiver. An important feature of the algorithm is that it is not necessary to determine a model of the source.

4. ALGORITHM

The Lossless Parallel 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.

With the increase in silicon densities, it is becoming feasible for multiple XMatchPros to be implemented in parallel onto a single chip. A parallel system with distributed memory architecture is based on having multiple data compression and decompression engines working independently on different data at the same time.

A description of the XMatchPro algorithm in pseudo-code is given in the below.

```
Clear the dictionary;
Set the next free location (NFL) to 0;
Do
{
read in a tuple T from the data stream;
search the dictionary for tuple T;
IF (full or partial hit)
{
determine the best match location ML and match type MT;
output ‘0’;
output any required literal characters of T;
}
ELSE
{ output ‘1’;
 output tuple T;
}
IF (full hit)
{
move dictionary entries 0 to ML -1 down by one location;
}
ELSE
{
move all dictionary entries down by one location;
increment NFL (if dictionary is not full);
}
copy tuple T to dictionary location 0;
}
WHILE (more data is to be compressed);
```

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 parallel 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.

4.1. XMATCHPRO Based Compression System

The Research on the lossless XMatchPro data compressor has been on optimizing and implementing the XMatchPro algorithm for speed, complexity and compression in hardware. 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.

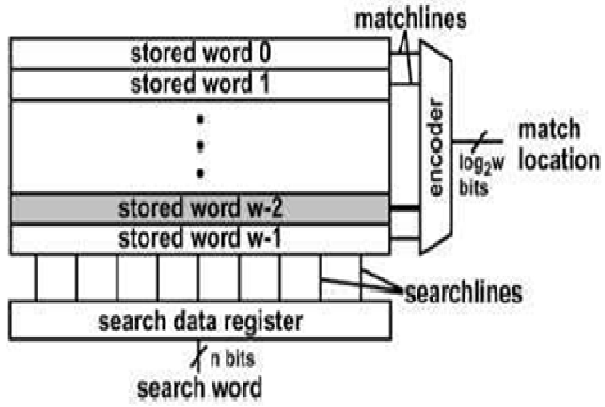

Fig.1 Conceptual view of CAM

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 parallel XMatchPro system, the data stream to be compressed enters the compression system, which is then partitioned and routed to the compressors.

For parallel compression systems, it is important to ensure all compressors are supplied with sufficient data by managing the supply so that neither stall conditions nor data overflow occurs.

4.2. 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 throughput.

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 search lines to the table of stored data. Each stored word has a match line that indicates whether the search word and stored word are identical (the match case) or are different (a mismatch case, or match).

The match lines are fed to an encoder that generates a binary match location corresponding to the match line 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.

4.3. 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, Content Addressable Memory COMPARATOR. 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 given as output.

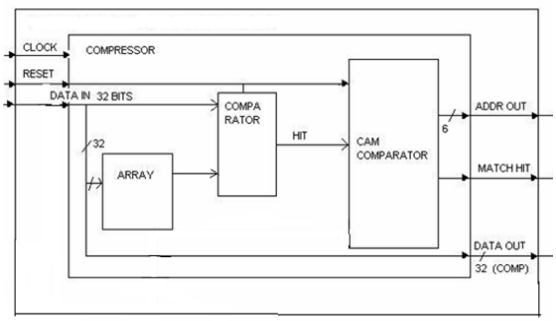

Fig.2 Diagram of 32-bit Compression

4.4. Description of Language used in source code

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. The hardware descriptions of these modules are done using VHDL Language. VHDL is an acronym for Very high-speed integrated circuits Hardware Description Language. It can be used to model a digital system at many levels of the abstraction, ranging from the algorithmic level to gate level.

The VHDL language can be regarded as an integrated amalgamation of the following languages:

Sequential language

Concurrent language

Net-list language

Timing specifications

Waveform generation language.

So the language has constructs that enable you to express the concurrent or sequential behavior of a digital system with or without timing. It also allows modeling the system as an inter-connection of components.

5. DESIGN OF COMPRESSOR / DECOMPRESSOR:

The block diagram gives the details about the components of a single 32- bit compressor / decompressor. The Same design approach is used for designing a 64-bit Compression/Decompression system which is essentially used for comparison of increased compression rates given by the 64-bit Lossless Parallel High-Speed Data Compression System. There are three components namely:

compressor

decompressor

control.

The compressor has the following components

comparator

array

cam comparator.

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.

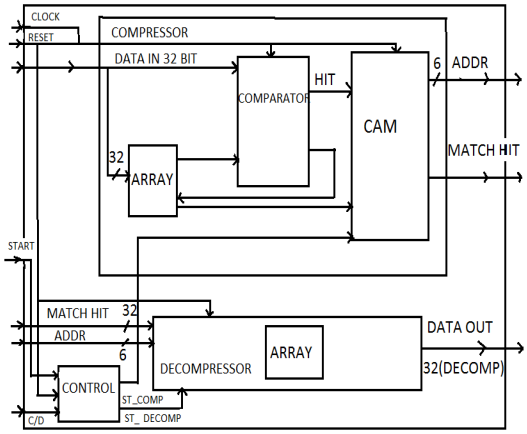

Fig.3 Block Diagram of 32 bit Compressor/ Decompressor

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 based compressor).

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 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 location is retrieved and sent as output along with the three components of the XMatchPro.

The decompressor has the following components – Array and Processing Unit. Array has the same function as that of the array unit 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 one bit to indicate the match is found. 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 compressor is stared when the value is 1 decompression is done.

6. SIMULATION RESULTS

The design coded in VHDL is simulated using Xilinx

The obtained waveforms are as follows

6.1. Comparator

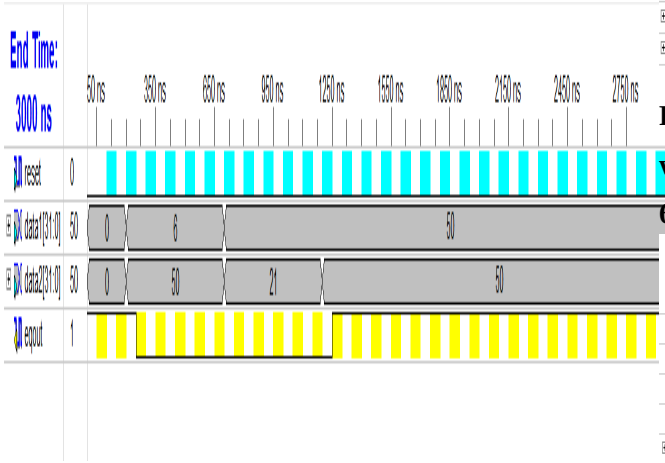

Fig.4 Comparator output waveform

6.2. CAM Comparator

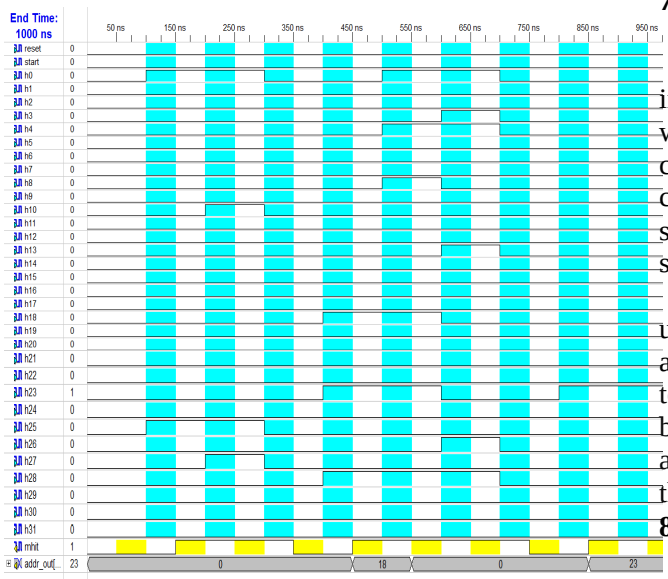

Fig.5 CAM Comparator output waveform

6.3. Data Compression

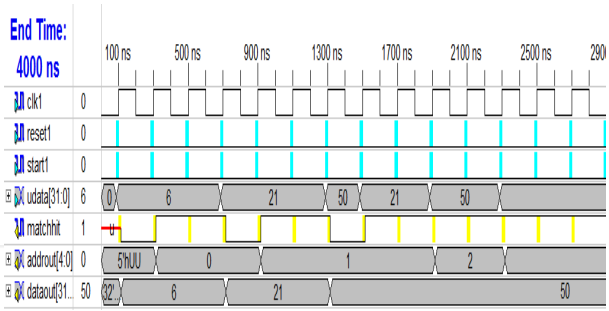

Fig.6 32-bit Single Compression output waveform

6.4. De-Compression

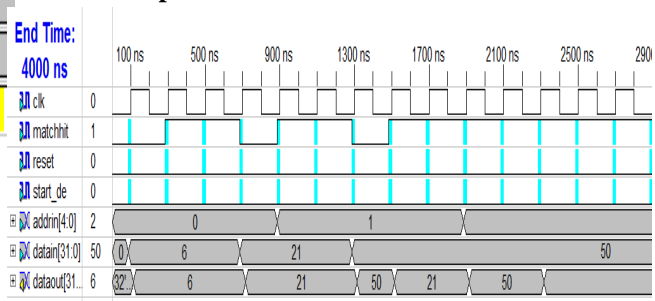

Fig.7 32 Bit Data decompression output waveform

7. CONCLUSION

In this paper we presented a method to implement lossless data compression system which operates at high-speed to achieve high compression rate. By using architecture of compressors, the data compression rates are significantly improved and also inherent scalability of parallel architecture is possible.

The algorithm "XMATCHPRO merge" used in this project is efficient at compressing and 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.So by this the error prone in the memory is reduced drastically.

8. REFERENCES

- IEEE transactions on very large scale integration (vlsi) systems, vol. 20, no. 12, december 2012

- Navabi, Z., VHDL Analysis and Modeling of Digital Systems,McGraw Hill, 1993
- B. Yuan, Z.Wang, L. Li,M. Gao, J. Sha, and C. Zhang, "Area-efficient Reed-Solomon decoder design for optical communications," IEEE Trans. Circuits Syst. II, Expr. Briefs, vol. 56, no. 6, pp. 469–474, Jun. 2009.
- Bhasker.J., VHDL PRIMER, Addison Wesely Longman Singapore Pte.Ltd. LPE,2000.
- H. Lee, "High-speed VLSI architecture for parallel Reed-Solomon decoder,"IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 11, no.2, pp. 288–295, Apr. 2003.
- S. Gregori, A. Cabrini, O. Khouri, and G. Torelli, "On-chip error correcting techniques for new-generation flash memories," Proc. IEEE, vol. 91, no. 4, pp. 602–616, Apr. 2003.
- C. Yang, Y. Emre, and C. Chaitali, "Flexible product code-based ECC schemes for MLC NAND flash memories," in Proc. IEEE Workshop Signal Process. Syst. (SiPS), 2011.