Improved Elliptical Cryptography in FPGA Processor

Dr.K.Kalpana[1], Dr.B.Paulchamy [2], Dr.C.Natarajan[3] J.B. Jebish Kumar [4]
[1]*Associate Professor/ECE, Hindusthan Institute of Technology, Coimbatore-32*
[2]*Professor/ECE, Hindusthan Institute of Technology, Coimbatore-323*
[3]*Professor/Mechanical, Hindusthan Institute of Technology, Coimbatore-32*
[4]*PG Scholar/M.E-VLSI, Hindusthan Institute of Technology, Coimbatore-32*

Abstract: Moore's law, which asserts that the amount of microprocessor technology (measured in terms of the number of transistors) doubles about every two years, explains the fast advancement of this technology in today's world. Memory that is attached to the technology has to be able to store a significant quantity of data as it continues to advance. If the memory is located off-chip from the CPU, the speed at which data can be accessed or stored is slower, and the latency is higher. If the memory is located on the same chip as the CPU, the speed is increased, making it quicker to retrieve the data and resulting in reduced latency. On-chip cache memory has to be constructed in such a manner that it can store a high quantity of data without increasing the amount of space it takes up on the chip. Compression and decompression of cache memory must be used for high-speed microprocessors in order to access vast amounts of data without reducing the performance of the microprocessor, without increasing its size, and without using more power. This paper proposes and designs a lossless method for high speed processors, focusing specifically on cache compression and decompression techniques in particular. A cache memory compression and decompression technique has been suggested and developed within the scope of this study. This method of compression enables parallel compression of several words while operating in dictionary mode. By using parallel compression, it is possible to cut the input word in half and then insert each half into the dictionary entry separately. In the beginning, the input word length was tested with 32 bits, and now that number has been expanded to 64 bits and tested again. When using dictionary mode, data may be accessed more quickly since it quickly finds matches with previously searched data. The suggested technique has been simplified down to a register transfer level design, which makes it possible to estimate performance, power consumption, and area. There is no decrease in compression ratio as a result of the performance. Comparisons are made between the compression ratio and that of other techniques currently in use. When compared with IBM's Memory Expansion Technology, the output results of the simulation of the suggested cache compression technique are examined (MXT). X- Match is a dictionary-based algorithm that uses move to front coding strategy, Lempel-Ziv (LZ) algorithm, and finally the proposed algorithm is compared with Frequent Pattern Compression (FPC), which compresses cache lines at the L2 level. MXT is a memory compression / decompression technique that improves performance by increasing the usable size of off-chip main memory. The algorithm that was suggested and is now being used produces superior results in terms of performance, area, and power consumption expenses when compared to other algorithms that are already in use.

Index Terms **– Cryptography, C-Pack Compression, Data Compression, FPGA, Frequent Pattern Compression**

I. INTRODUCTION

The performance gap that exists between processors and memory is the consequence of the fast growth of semiconductor technology and the continued expansion in the number of micro architectural advances. Moore's law asserts that the performance and speed of CPU technology will double about every two years. L1 and L2 are the two layers of cache memory that are used by modern processors to cut down on latency and increase bandwidth [1]. It has been suggested that compressing cache memory may enhance system performance. This is due to the fact that effective capacity can be raised by the compression of data that is stored in on-chip caches, which in turn decreases the number of cache misses. Because on-chip cache memory structures can store more data in megabyte size, performance grows more quickly whenever there is a technological advancement in the CPU. When compared to the speed of the CPU, the speed of the off-chip memory is much

slower. When a multiprocessor is used in the design of the system, it needs a greater number of access points to memory. The off-chip communication speed with the CPU may be slowed down with the help of cache compression.

II. RELATED WORKS

Latency is introduced by both compression and decompression; however, the latency introduced by compression will not be on the crucial route, whereas the latency introduced by decompression will be. [2]Compression algorithms that provide a high level of compressibility are going to have a cost in terms of both space and latency. Both compressing and uncompressing data may be quite helpful when working with main memory and the last level cache. Different sizes of compressed blocks may lead to fragmentation, which is also known as a break in linear mapping. The data that is entered may be compressed in a variety of ways, depending on the inserted data. It's easy for replacement policy to get complicated. If the memory footprint of the programme is quite tiny, compression will be of little value. If the footprint is too big, the additional capacity that is offered through compression will not be adequate.

The primary benefit of any compression approach is that it minimizes the demand for the amount of data storage space. It presents a strategy for lowering the costs associated with the transport of large amounts of data across extended connection distances by increasing the efficiency with which the available bandwidth is used [3]. As a result of the decrease in data rate, data compression significantly improves the quality of presentations made via communication channels with restricted bandwidth. This is because of the reduction in data rate. Because of the many different compression methods that are available, the use of the internet has shifted more toward being dependant on images and graphics rather than being centred on data and text. The development of high-performance compression [4] has paved the way for new methods of developing applications such as digital libraries, digital archives, video teleconferencing, telemedicine, and digital entertainment, amongst others.

There are other benefits that come with data compression as well. For instance, it has a significant influence on the access to the database. Encrypting and sending data is a great way to increase the level of data security [5]. By making both the compression and decompression procedures completely visible to potential imposters, an additional layer of protection may be added to the system. It is possible for the pace of input and output operations in a computer device to rise as a result of shortening the form of data representation used. By keeping backup copies of huge database files in a compressed format, data compression cuts down the costs associated with the majority of applications. Because of its benefits, data compression will make it possible to develop more multimedia and video apps at a lower cost.

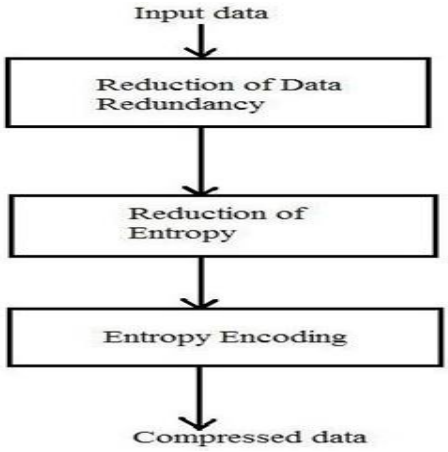

Figure 1.Data Compression Model

The basic block diagram of Data Compression Model is shown in Figure 1. The steps of basic data The three components that make up a compression model are data redundancy reduction, entropy reduction [7], and entropy encoding. The term "data redundancy" refers to the situation that arises when database management systems have a field that is duplicated in two or more tables. [8] A recognised cause of inconsistency is redundancy of data, which occurs whenever there is a duplicate of the input data. When establishing a relational database that will include several entities, it is important to prevent situations that might result in data deviation, such as data redundancy [9]. Database normalisation prevents redundancy and makes the best possible utilisation of storage. An entropy encoding is a method of data compression that does not result in the loss of any information and is not reliant on the properties of the media being used.

A. Contraindications of Utilizing Data Compression

- There are very few downsides associated with

data compression, however this does depend on the application. One of the negatives of data compression is the overhead that is incurred as a result of encoding and decoding, for instance.

- The dependability of the system suffers as a result of data compression. For example, a single bit mistake in compressed coding might lead the decoder to misread all succeeding bits which creates inaccurate data.
- The transmission of highly protected compressed data, such as that used in the medical field, over a noisy communication medium, such as a wireless channel, is fraught with danger because the burst errors that are introduced by the noisy channel have the potential to destroy the data that is being transmitted [10].
- Since the output of compressed data is different from the data that was originally provided, one of the most significant issues with data compression is that it disrupts the characteristics of the data.
- The additional complexity brought about by data compression may, in many hardware implementations, lead to an increase in the cost of the system, which in turn can lead to a reduction in the system's efficiency, particularly in the areas of applications that call for extremely low power. VLSI implementation.

B. Alterations to the data after compression

The data are automatically altered by the compression process. These shifts might sometimes result in an increase in unnecessary energy expenditure. The following are some of the reasons why this is the case:

• In any specific scenario, whether a 0 or 1 is communicated or stored has a significant impact. For instance, in the on-chip interconnects that have just transferred a bit with the value 0, transferring another bit with the value 0 over the same pin that has just transferred a bit with the value 0 requires almost no additional energy [11], whereas transferring a bit with the value 1 would require more energy. The energy efficiency of data transfer is significantly impacted when there is a greater number of switches on the connecting cable. These switches are referred to as bit toggles.

The vast majority of today's programming languages and compilers store data in a predictable manner, and as a result, data is often aligned at a granularity of four or eight bytes. This is also consistent with the manner in which the data is subsequently sent over communication channels (e.g., 16-byte alignment for many modern on- chip system network). This implies that numerous comparable bits continue to be passed across the same pins, which minimises the amount of energy required for the data transmission process. Unfortunately, data compression typically violates this assumption about the alignment of the data [12], which results in a large increase in the total number of bit toggles and, thus, an increase in the amount of energy required for on-chip data transfers.

III. PROPOSED METHODLOGY

C-Pack, which stands for "Cache Packer," is a technique for lossless compression [13] that was developed particularly for high-performance hardware-based on-chip cache compression. When used for the purpose of compressing data that is often located in low-level on-chip caches of microprocessors, such as L2 caches, it produces a decent compression ratio. The previous work that was done on pattern-based partial dictionary match compression was a significant inspiration for its design. On the other hand, this earlier study focused only on software-based main memory compression and did not take hardware implementation into consideration.

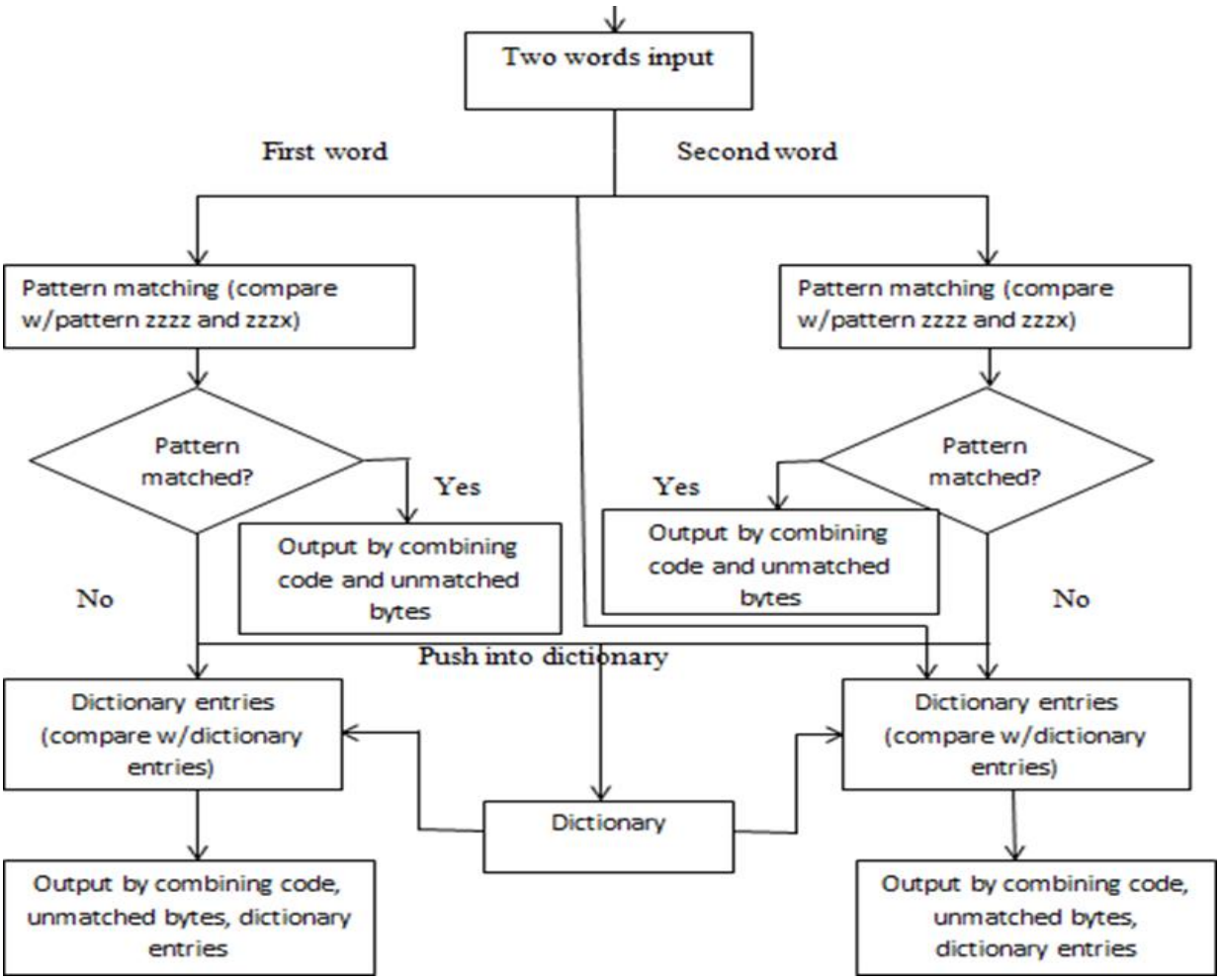

Figure 2. C-Pack Compression

Figure 2 shows the algorithm for C-Pack Compression. C-Pack is able to compress data through the use of two different methods: first, it uses compact encodings that are statically determined for data words that appear frequently, and second, it encodes using a dictionary that is dynamically updated so that it can adapt to other words that appear frequently. The dictionary enables users to do partial word matching in addition to complete word matching. As an example, we will utilize an input of two words for each cycle. The method, on the other hand, is readily adaptable to situations in which there are either one or more than two words every cycle. During the course of one cycle, a comparison is made between each word and the patterns "zzzz" and "zzzx." In the event that a match is found, the compression output is created by combining the code corresponding to the match with the bytes that were not found to be a match, as shown in Figure 2. In such case, the compressor examines the word in relation to each entry in the dictionary, looking for the one that has the greatest number of matched bytes. After that, the outcome of the compression is derived by combining the code, the index of the dictionary item, and any mismatched bytes that may exist. Words that are not recognized by the pattern matching system are added to the dictionary. Parentheses surround the code and the dictionary index, if there is one, in each output that is produced [14]. Despite the fact that a 4-word dictionary was used for demonstration purposes, the dictionary size in this implementation is set to 64B. Take into consideration that the dictionary receives an update upon the addition of each new term.

Each cycle requires the input of two words. During the first iteration, each word is first evaluated based on how closely it resembles certain patterns, such as "zzzz" and "zzzx." In the event that there is a match

between patterns, the output is generated by merging the code that is important to the match with the bytes that are not matched. A comparison of the data with each and every dictionary entry is possible if this is not the case. The output of the compression is then formed by combining the code, the index of the dictionary item, and any bytes that are mismatched. The data that does not correspond to the patterns is transferred into the dictionary that is shown in Figure 2.

A. The Proposed Block diagram for Cache Decompression Algorithm

During the decompression process, it first retrieves the compressed words and then extracts the codes in order to do the pattern analysis. In the event that the code identifies a pattern match, the first word is recovered by concatenating 0s and bytes that have not been matched. If this is not the case, the result of the decompression is created by mixing [15] bytes from the input word with bytes from the dictionary entries illustrated in Figure 3.3.

During the decompression process, the decompressor will first read compressed words and then extract the codes necessary for evaluating the patterns of each word. These codes will then be checked against the codes that are described in Table 3.1. In the event that the algorithm identifies a matching pattern, the first word is reconstructed by merging any zeroes and mismatched bytes, if there are any. If this is not the case, the output of the decompression is obtained by combining bytes from the input word with bytes from dictionary entries if the code indicates that there is a match in the dictionary.

The C-Pack method was developed with hardware implementation in mind from the very beginning. It does this by simultaneously comparing a word that is inputted with a number of probable patterns and dictionary entries. This enables quick execution while maintaining a decent compression ratio in a hardware implementation; however, it may not be appropriate for a software implementation. In general, software must perform actions in the order that they were given. When there are a huge number of patterns or dictionary entries, for instance, performing a matching operation against many patterns might become prohibitively costly for software implementations.

The naturally parallel architecture of C-Pack makes it possible for an effective hardware implementation, in which pattern matching, dictionary matching, and the processing of multiple words may all take place at the same time. In addition, numerous design parameters are selected, such as the dictionary replacement policy and the coding scheme, in order to reduce the complexity of the hardware. This is done despite the fact that the effective compression ratio of the entire system is slightly decreased as a result of our choices. In the implementation of C-Pack that has been suggested, two words will be processed in parallel for each cycle.

It may be difficult to achieve this goal while yet retaining the ability to make an accurate dictionary match for the second term. In the process of compression, two words that are very similar to one another but have not been seen by the compression algorithm in a while are taken into consideration. This is done on the assumption that the dictionary utilises the first-in-first-out (FIFO) replacement strategy. When processing the second word, the proper dictionary content is determined by determining whether or not the first word matched a static pattern. If this is the case, the first term will be omitted from the dictionary. In any other case, it will be included in the dictionary, and the fact that it is there may be utilised to encode the second word. As a result, the second word ought to be contrasted with the first word and all of the dictionary entries in parallel with the exception of the first entry. When compared to the more efficient strategy of not checking with the initial word, this results in an improvement in the compression ratio. Therefore, it is possible to compress two words in parallel without causing a reduction in their compression ratio. Figure.3 shows the algorithm for C-Pack decompression.

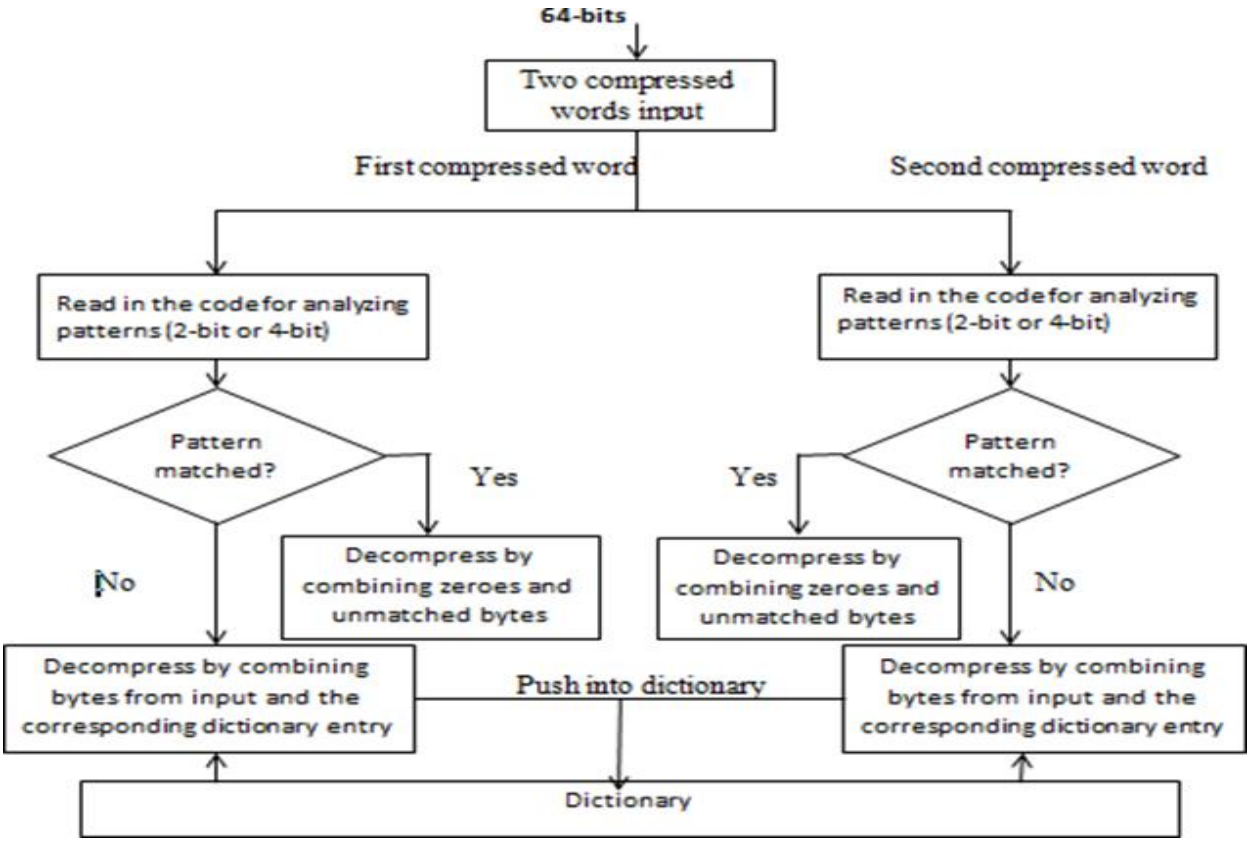

Figure 3. C-Pack decompression

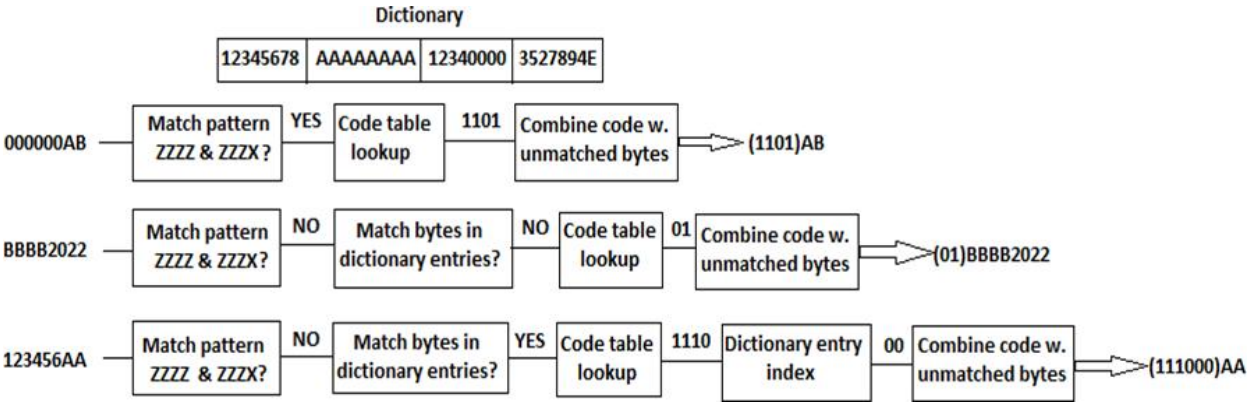

Figure 4 . Compression with different inputs

Figure.4 shows how the algorithm works for different input data to produce output. The advantage of this algorithm is an input word is compared with multiple patterns and with dictionary entries. This can be permitted for rapid execution with good compression ratio in hardware implementation. To reduce hardware complexity, various design parameters such as dictionary replacement policy and coding scheme can be chosen. In the proposed implementation, two words are processed in parallel per cycle.

IV. RESULTS AND ANALYSIS

During decompression the original input word is recovered. If the extracted code indicates a pattern match, then the original word is recovered by combining 0¨s and is shown in Figure 5.

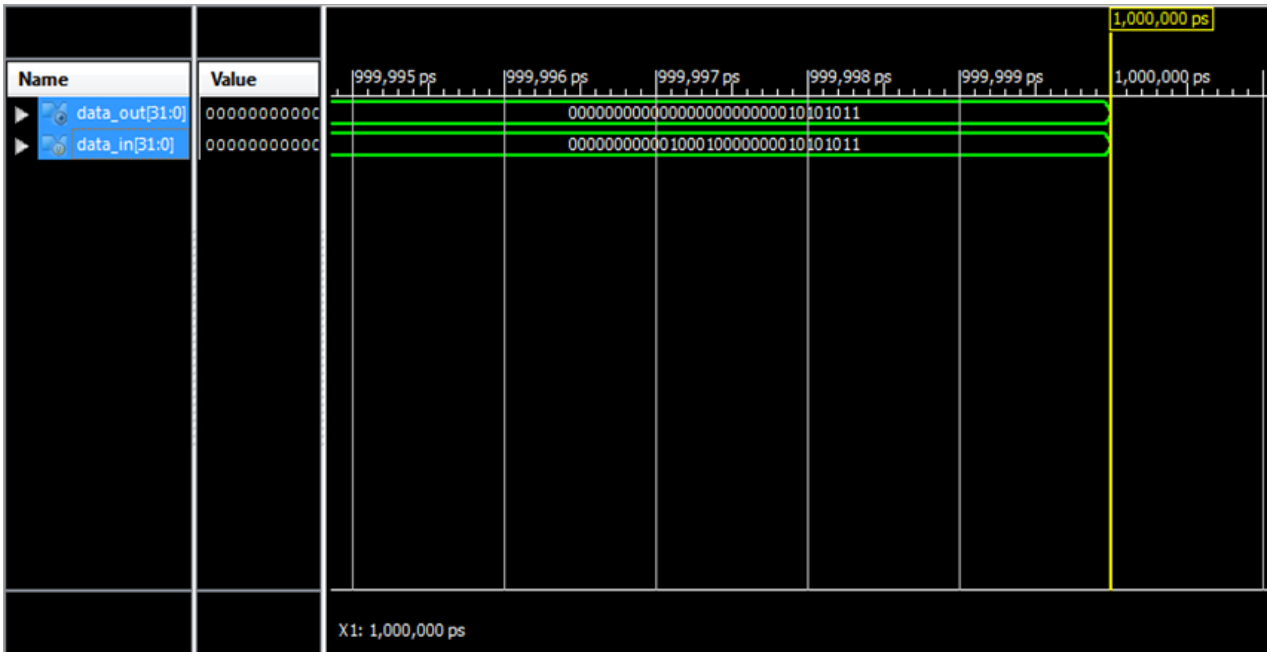

Figure 5. Decompression result for (1100) AB

If the code indicates that there is no match with the pattern but there is match with the dictionary entries then the original word is recovered by concatenating the zeroes and unmatched bytes, if any is shown in Figure 6.

```
Asynchronous Control Signals Information:
----------------------------------------
------------------------------------------------------------------+-------------------------+-
Control Signal                                                    | Buffer(FF name)         |
------------------------------------------------------------------+-------------------------+-
xbuffer/NLOAD(xbuffer/NLOAD:Q)                                    | NONE(Mram_nintop)       |
xbuffer/ichf_GND_53_o_OR_460_o_0(xbuffer/ichf_GND_53_o_OR_460_o11:O)| NONE(xbuffer/Mram_buf) |
------------------------------------------------------------------+-------------------------+-

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 5.135ns (Maximum Frequency: 194.744MHz)
   Minimum input arrival time before clock: 2.921ns
   Maximum output required time after clock: 1.618ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'CLK2'
  Clock period: 4.237ns (frequency: 236.032MHz)
  Total number of paths / destination ports: 161483 / 6134
-------------------------------------------------------------------------
```

Figure 6. Decompression result for (111000)AA

Timing analysis for the compressed output data is shown in Figure 7. The timing parameters are represented here. After the design implementation, design summary is verified and is shown in Figure 8.

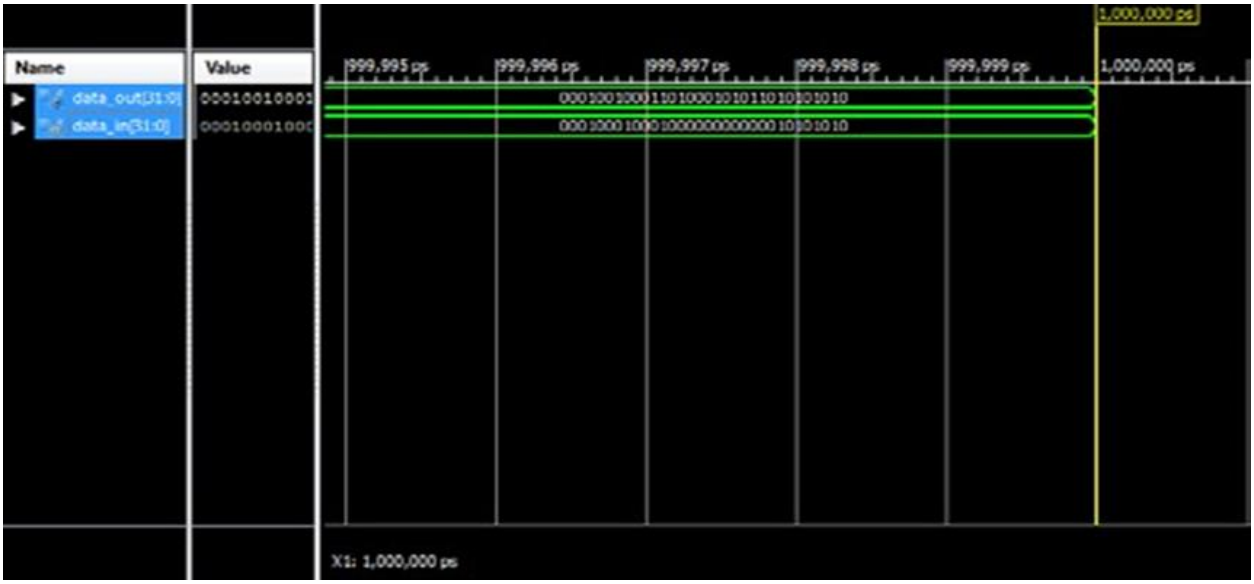

Figure 7. Timing analysis

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2664	408000	0%
Number of Slice LUTs	5034	204000	2%
Number of fully used LUT-FF pairs	2000	5698	35%
Number of bonded IOBs	89	600	14%
Number of Block RAM/FIFO	2	750	0%
Number of BUFG/BUFGCTRL/BUFHCEs	2	200	1%
Number of DSP48E1s	2	1120	0%

Figure 8. Device Utilization Summary

A. Comparison Of Compression Ratio

The proposed algorithm is compared with few algorithms namely X-Match, MXT and Frequent Pattern Compression (FPC). Other compression algorithms are excluded because they lack hardware design so they are notsuitable for cache compression. Compression ratio of different algorithms are checked and compared for the performance. The dictionary size and block size are checked and set to 64 B in test cases. LZSS Lempel-Ziv compression algorithm is being used instead of MXT to approximate compression ratio.

The raw compression ratio and system wide compression ratio in pair matching scheme are being summarized in Table 1. The raw compression ratio and system wide compression ratio are listed in each row. The Table 1 shows the results comparison of different cache compression algorithms. This result comparison is shown for the benchmark mpeg 2. The other benchmark files considered were mesa, art and wolf. Though the table shows the compression ratio value for one file type only, the average value had been calculated. In this analysis of average value calculation, the proposed algorithm and design had produced raw compression ratio of 58.30% and effective system wide compression ratio of 61.40%.

Table 1. Comparison of Compression ratio

Memory Compression technique	Raw compression ratio (%)	System wide compression ratio (%)
MXT	70.88	75.55
FPC	63.39	64.28
X-MATCH	49.50	57.97
Proposed algorithm	52.10	58.47

V. CONCLUSION

As a result of this research, a cache memory compression technique that is both effective and suitable for high-performance microprocessors has been presented. Pattern matching and partial dictionary coding serve as the foundation for the creation of the suggested method. Compression of several parallel words is allowed and will not reduce the chance of a dictionary match being found. The new method achieves an effective compression ratio of 59% over the whole of the system. When compared to those of alternative compression methods that may be used for this application, the results that were produced are comparable to being significantly superior. The data that has 64 bits or more may be compressed and decompressed with the help of the method that has been simulated and synthesised. The data are compressed into the cache in an effective manner while ensuring that the performance is not affected in any way. The compression ratio produced by this approach is satisfactory. Because less space is being used, memory latency will be reduced, which will result in an improvement in the speed of the system's memory performance. This approach may also be used for applications requiring lossless high performance data compression, either with or without the need for any changes. The compression and decompression techniques for cache memory were evaluated and contrasted with those of other compression algorithms, such as X-Match, Frequent Pattern Compression, and IBM's memory expansion technology. A comparison was made between the raw compression ratio and the effective system wide compression ratio. Using an interpolation method, the data compression technique was attempted for the picture data input. There is generation of both the technical and RTL schematic block diagrams. There is a listing of the timing analysis, as well as the area overhead and delay values. Only data compression via the use of the interpolation approach was identified in this investigation.

REFERENCES

[1] Acquaviva A & Ricc B, 2003, “Energy Characterization of Embedded Real-Time Operating Systems‟. In Proceedings of the Workshop on Compliers and Operating Systems for Low Power, pp.53-73.

[2] Alameldeen & Wood, D A, 2004, “Frequent pattern compression: A significance-based compression scheme for 12 caches”, Dept. Comp.Scie., Univ. Wisconsin-Madison, Tech.

[3] Alameldeen AR & Wood DA 2005, “Multifacet‟s general execution-driven multiprocessor rsimulator (gems) toolset”, In Computer Architecture News, pp.92-99.

[4] Alameldeen, R & Wood, DA 2004, “Adaptive cache compression for high-performance processors”, in Proc.Int.Symp. Computer Architecture, pp.212-223.

[5] Alghazo J, Akaaboune A & BotrosSf-lru, 2004, “Cache Replacement Algorithm‟, In Proceedings of the Records – International Workshop on Memory Technology, Design and Testing, pp. 19-24.

[6] Al-Zoubi H, Milenkovic A & Milenkovic M 2004, “Performance evaluation of cache

replacement policies for the SPECCPU 2000 benchmark suite", In Proceedings of the 42nd Annual Southeast Reg. Conference, ACM-SE 42, pp.267-272.

[7] Ardsher Ahmed, Pat Conway, Bill Hughes & Fred Weber 2002, "shared memory MP systems", In Proceeding of the 14th HotChips Symposium, pp.1-30.

[8] Bardine A, Foglia P, Gabrielli G &Prete CA 2007, "Analysis of staticand dynamic energy consumption in nuca caches: Initial results", In Proceedings of the Workshop on Memory Performance: Dealing with Applications, Systems and Architecture,pp.105-112.

[9] Benchmarks.Speccpu2006.In<http://www.spec.org/cpu2006,2006>.

[10] BeniniL & Micheli G De 2000, "System-level power optimization: techniques and tools", ACM Transactionon Design Automation Electronic System,Vol.5,Issue 2, pp.115–192.

[11] Benini L 2003, "Energy-Aware Design of Embedded Memories : ASurvey of Technologies, Architectures, and Optimization Techniques",Vol.2,Issue 1,pp.5–32.

[12] Bienia C, Kumar S, Singh JP & Li K 2008,"The parsec benchmark suite: Characterization and architectural implications", In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, pp.72-81.

[13] Borkar S & Chien A A 2011, "The Future of Microprocessors Communications of the ACM", Vol. 54, No.5, pp.67-77.

[14] Bradford M, Beckmann & David A.Wood 2004., "Managing Wire Delay in Large Chip-Multiprocessor Caches", In Proceeding of the 37th international Symposium on Microarchitecture (MICRO-37)", Portland, Oregon, pp.319-330.

[15] Burrows M & Wheeler D J 1994, "Ablock-sorting lossless data compression algorithm"

[16] Chang P P, Mahlke S A, Chen W Y, Warter N J & Hwu W 1991, "IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors", Proc. Of International Symposiumon Computer Architecture.

[17] Chen T & Baer J 1995, "Effective hardware-based data prefetching for high-performance processors. IEEE Transaction on Computing", Vol.44, issue 5, pp.609–623.

[18] Chetana N, Keltcher Kevin J. McGrath Ardsher Ahmed Pat Conway2003, "The AMD opteron processor for Multiprocessor Server", IEEE Micro, vol.23, issue 2, pp.66-76.

[19] Chishti Z, Powell M D & Vijaykumar T N,2003, "Distance associativity for high performance energy-efficient non-uniform cache architectures" In Proceeding of the 36th International Symposiumon Microarchitecture", pp.55-56.