ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

An ISO 3297: 2007 Certified Organization Volume 6, Special Issue 5, March 2017

National Conference on Advanced Computing, Communication and Electrical Systems - (NCACCES'17)

24th - 25th March 2017

Organized by

C. H. Mohammed Koya
KMEA Engineering College, Kerala- 683561, India

Dictionary based Code Compression Scheme using BitMasks Approach

Chithra M [1], Riyas K S [2]

P.G. Student, Department of Electronics and Communication Engineering, Rajiv Gandhi Institute of Technology, Kottayam, Kerala, India[1]

Associate Professor, Department of Electronics and Communication Engineering, Rajiv Gandhi Institute of Technology, Kottayam, Kerala, India [2]

ABSTRACT*:* Embedded systems have become an essential part and are widely used worldwide. Memory plays a crucial role in designing embedded systems. A larger memory can accommodate more and large applications but increases cost, area, as well as energy requirements. Code compression is a technique used to address this issue by reducing the code size of application programs. In this paper, a bitmask-based code compression which is a modified version of dictionary-based code compression is presented. The basic purpose of bitmask is to record the mismatched values and their positions to compress a greater number of instructions. A dictionary selection technique to minimize the code size based on the selected bit-masks is proposed. Inorder to achieve the potential performance gain (compression ratio) without introducing any decompression penalty, the decompression mechanism is used. The proposed architecture is simulated using Xilinx ISE design suite.

KEYWORDS*:* Embedded systems, bitmask-based code compression, dictionary-based code compression (DCC), compression ratio, decompression mechanism

I.INTRODUCTION

Embedded systems are everywhere from household appliances to biomedical, military, geological and space equipments. The complexity of such systems is increasing at an exponential rate due to both advancements in technology and demands for sophisticated applications, in the areas of communication, multimedia, networking and entertainment. Memory is one of the key driving factors in embedded system design since a larger memory indicates an increased chip area and a higher cost. As a result, memory imposes constraints on the size of the application programs. Code compression techniques address the problem by reducing the program size.

The concept of code compression was first introduced by Wolfe and Chanin [1] inorder to conserve the memory usage. Their research in code compression further led to the reduction of the code size and power consumption, as well as to improve the performance. For all the existing code compression techniques, all binary instructions are compressed offline and decompressed as required during execution. Thus, reducing the code size and providing a simple decompression engine are both challenges when applying code compression to embedded systems. In [2], the authors proposed a dictionary-based code compression (DCC) which was able to achieve an efficient CR (compression ratio), possess a relatively simple decoding hardware, and provided a higher decompression bandwidth than the code compression by applying lossless data compression methods. Although several existing code compression algorithms have exhibited favourable compression performance, no single

compression algorithm has efficiently worked for all kinds of benchm. In this paper, various steps in the code compression process were combined into a new algorithm to improve the compression performance (including the CR) with a smaller hardware overhead.
An efficient code compression technique to improve the compression ratio further by creating more matching sequences using bit-mask patterns is proposed here. This technique introduces many challenges to make it viable in practice. First, how to select a set of mask-patterns that will maximize the matching sequences while minimizing the cost of using bit-masks and second, how to perform efficient dictionary selection using the profitable bit-masks.
Paper is organized as follows. Section II presents a review of related studies on code compression. Section III describes the existing technique of dictionary-based code compression. Section IV describes the proposed bitmask-based compression approaches [3]–[5]. Section V presents the simulation results of compression and decompression mechanism. Finally, the conclusion is drawn in Section VI.

II. RELATED WORK

Numerous lossless data compression algorithms have been applied to code compression for embedded systems. The first code compression technique for embedded processors was proposed by Wolfe and Chanin [1]. Their technique uses Huffman coding and the compressed program is stored in the main memory. The decompression unit is placed between main memory and the instruction cache. They used a Line Address Table (LAT) to map original code addresses to compressed block addresses. The idea of using dictionary to store the frequently occurring instruction sequences has been explored by various researchers [2], [6]. Lekatsas and Wolf [7] proposed a statistical method for code compression using arithmetic coding and Markov model.

The techniques discussed so far target RISC processors. There has been a significant amount of research in the area of code compression for VLIW and EPIC processors. The technique proposed by Ishiura and Yamaguchi. [8] Splits a VLIW instruction into multiple fields and each field is compressed using a dictionary based scheme. Nam et al. [9] also uses dictionary based scheme to compress fixed format VLIW instructions. Various researchers have developed code compression techniques for VLIW architectures with flexible instruction formats [10], [11].

Larin and Conte applied Huffman coding for code compression. Xie et al. [11] used Tunstall coding to perform variable-to-fixed compression. Lin et al. proposed a LZW-based code compression for VLIW processors using a variable-sized-block method. Ros and Sutton have used a post-compilation register reassignment technique to generate compression friendly code. Das et al. applied code compression on variable length instruction set processors.

Several techniques [12], [13] have been proposed to improve the standard dictionary based code compression by considering mismatches. However, the efficiency of these techniques are limited by the number of bit changes (hamming distance) used during compression. The cost of storing the information for more bit positions cancels out the advantage gained by generating more repeating instruction sequences. Studies [13] have shown that it is not profitable to consider more than three bit changes when 32-bit vectors are used for compression. Prakash et al. [12] have considered mismatch in only one bit position.

Recent research in code compression has focused on two directions: 1) applying existing compression methods to various architectures for optimization and 2) combining several approaches to improve the performance, including CR. Seong and Mishra [3], [4] and Wang and Lin [14] observed that no single compression algorithm operated efficiently for all the benchmarks. Thus, this paper integrates several approaches to form a new algorithm with smaller hardware overhead. New dictionary architecture is used to improve the decompression engine performance.

III. DICTIONARY-BASED CODE COMPRESSION

Dictionary-based code compression techniques provide compression efficiency as well as fast decompression mechanism. The basic idea is to take the advantage of commonly occurring instruction sequences by using a dictionary. The repeating occurrences are replaced by a code word that points to the index of the dictionary that contains the pattern. The compressed program consists of both code words and uncompressed instructions.

Fig. 1 shows an example of dictionary based code compression using a simple program binary. The binary consists of ten 8-bit patterns i.e., total 80 bits. The dictionary has two 8-bit entries. The compressed program requires 62 bits and the dictionary requires 16 bits. In this case, the compression ratio is 97.5% using the equation,

Compression Ratio = Compressed Program Size / Original Program Size

Recently proposed techniques [12], [13] improve the dictionary based compression technique by considering mismatches. The basic idea is to determine the instruction sequences that are different in few bit positions (hamming distance) and store that information in the compressed program and update the dictionary (if necessary). The compression ratio will depend on how many bit changes are considered during compression. It is obvious that if more

bit changes are allowed, more matching sequences will be generated. However, the size of the compressed program will increase depending on the number of bit positions.

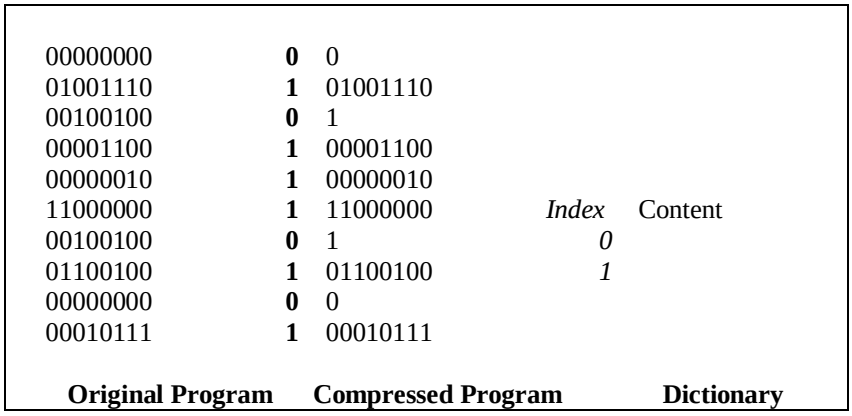

Fig. 1: Dictionary-based Code Compression: An Example

Fig. 2 shows the improved dictionary based scheme using the same example (shown in Fig. 1). This example considers only 1-bit change. An extra field is necessary to indicate whether mismatches are considered or not. In case a mismatch is considered, another field is necessary to indicate the bit position that is different from an entry in the dictionary. For example, the third pattern (from top) in the original program is different from the first dictionary entry (index 0) on the sixth bit position (from left). The compression ratio for this example is 95%.

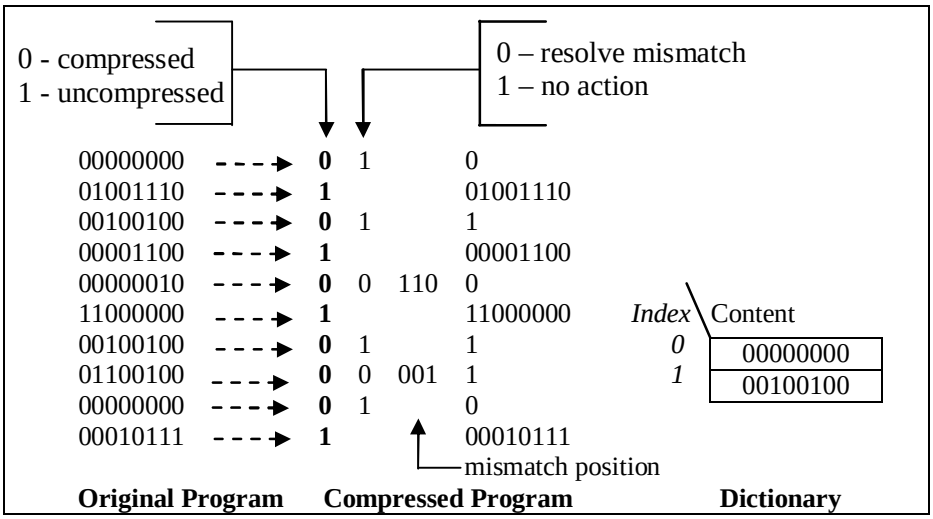

Fig. 2: Improved Dictionary-based Code Compression

IV. CODE COMPRESSION USING BIT-MASKS

The proposed approach tries to incorporate maximum bit changes using mask patterns without adding significant cost (extra bits) such that the compression ratio is improved. Our compression technique also ensures that the decompression efficiency is improved or remains the same compared to the existing techniques. Fig. 3 shows the generic encoding scheme used by our compression technique.

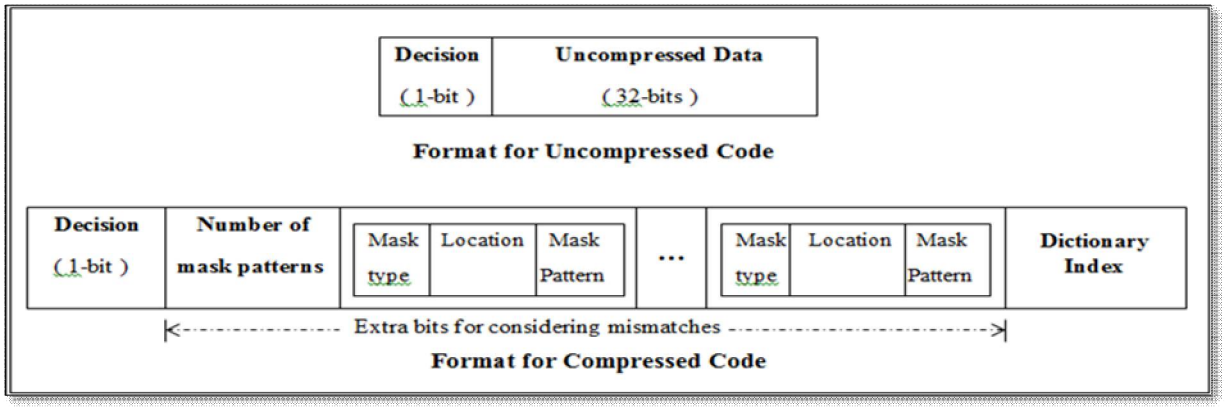

Fig. 3: Encoding Format for Our Compression Technique

This encoding format (Fig. 3) can store information regarding multiple mask patterns. For each pattern, it stores the mask type, the location where mask needs to be applied, and the mask pattern. The generic encoding scheme can be optimized once the types and sizes of the bitmask combinations are determined.
Inorder to further improve the compression ratio, we utilized the code compression using bitmask approach (Fig. 4). A 2-bit mask (only on quarter byte boundaries) is sufficient to create 100% matching patterns and thereby improves the compression ratio (87.5%) as shown in Fig. 4.

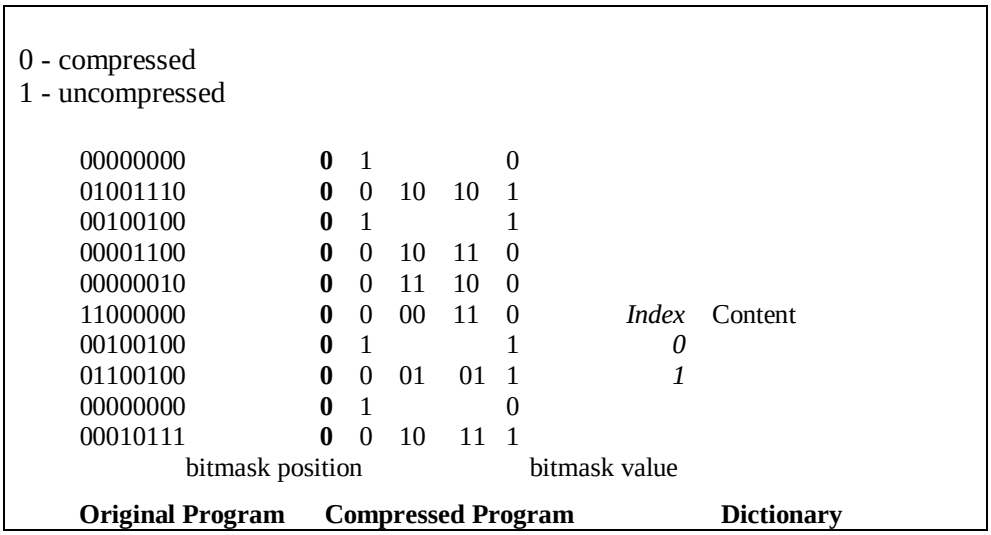

Fig. 4: Code Compression Using Bit-Mask Approach

Code compression using bit-masks is a promising approach. However, it introduces two major challenges. First, how to select a set of mask-patterns that will maximize the matching sequences while minimizing the cost of using bit-masks and second, how to perform efficient dictionary selection using the selected bit-masks. Inorder to overcome these challenges, an efficient mask selection and dictionary selection technique to improve compression ratio without introducing any decompression penalty is proposed.

Fig. 5 shows the traditional code compression and decompression flow where the compression is done off-line (prior to execution) and the compressed program is loaded into the memory. The decompression is done during the program execution (online).

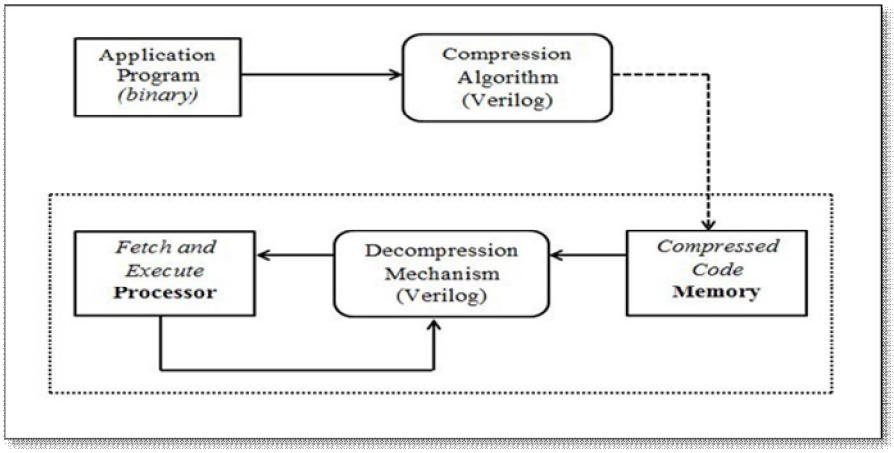

Fig. 5: Traditional Code Compression Methodology

A. COMPRESSION ALGORITHM

Fig. 6 shows the flowchart of Code Compression using Bitmasks. The algorithm accepts the original code consisting of 32-bit vectors. The first step is to determine which mask set to use for compression. The second step is to select the optimized dictionary. The third and final step of the algorithm converts each 32-bit vector into compressed code (when possible) using the format shown in Fig. 3. The compressed code along with any uncompressed ones is composed serially to generate the final compressed program code.

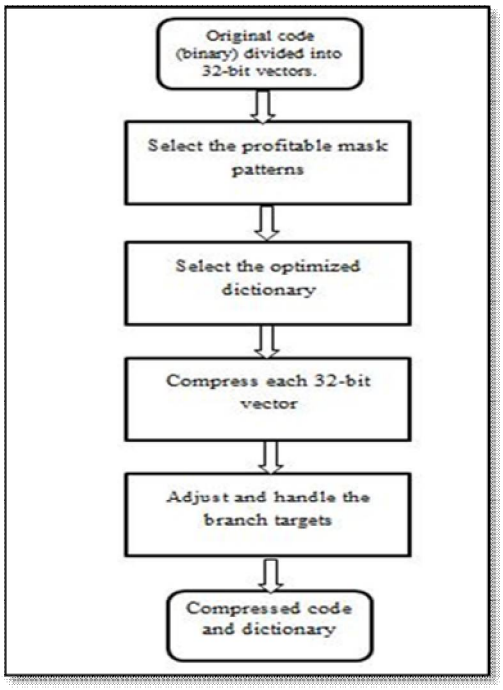

Fig. 6: Flowchart of Code Compression using Bitmasks

B. DECOMPRESSION MECHANISM

The decompression scheme for our compression technique is similar to the existing dictionary based code compression techniques. Fig. 7 shows one possible implementation of the decompression unit in hardware when up to two mask patterns is considered. The compressed code is accessed serially and checked if it is compressed or not. If compressed, the corresponding dictionary entry is retrieved and XOR-ed with a 32-bit mask pattern. The 32-bit mask pattern is generated based on the mask patterns and their location information in the compressed code. Finally the original uncompressed code is obtained at the output.

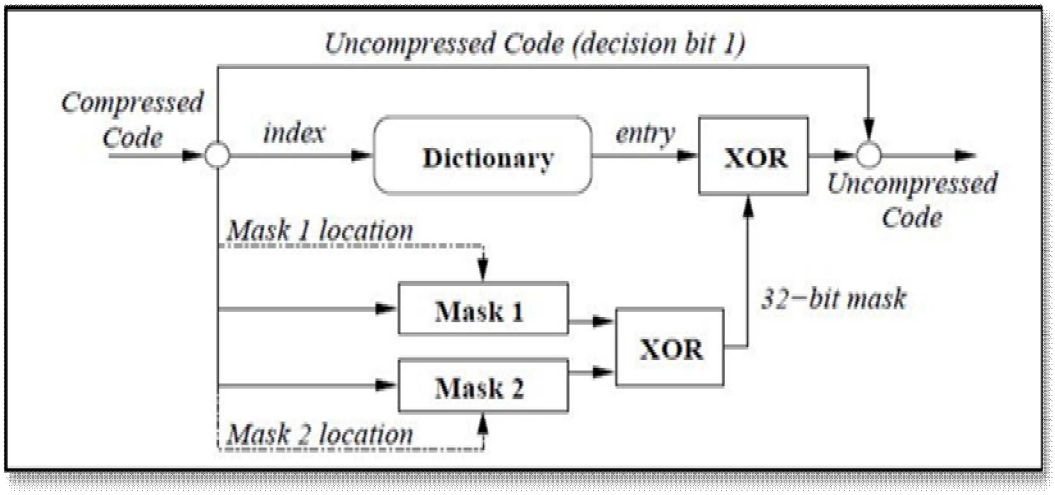

Fig. 7: Implementation of a Decompression Unit

V. SIMULATION RESULTS

A. *COMPRESSION*

This module is used to integrate all the modules. This is the top module for compression. Test vectors and bit masking type are given to this module and we will get dictionary table and compressed data as outputs. Here as an example, a 4-bit TestAddr and a 10-bit DictEntr whose VectLen as 10-bit is given as input to get DictCalOver as output of compression.

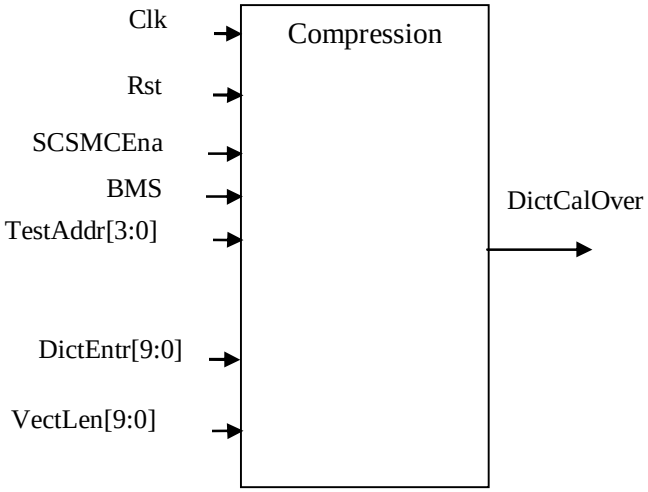

Fig. 8: Block Diagram of Compression

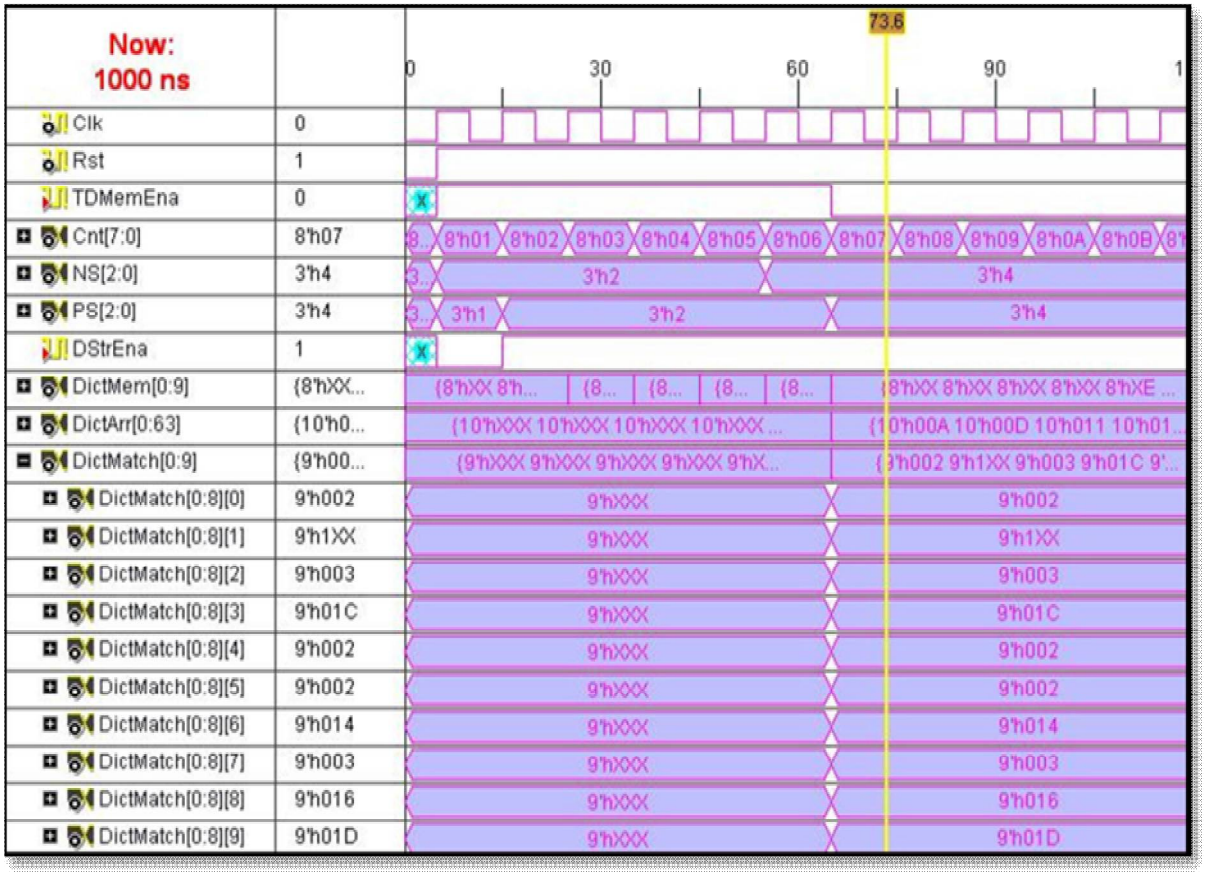

Fig. 9: Waveform of Compression

B. DECOMPRESSION

This module is used to decompress the data. Compressed data is decompressed using dictionary table and we will get original uncompressed data. Here as an example, the 9-bit Comp Data along with 8-bit DicData is decompressed to get 8-bit DOut as our uncompressed output.

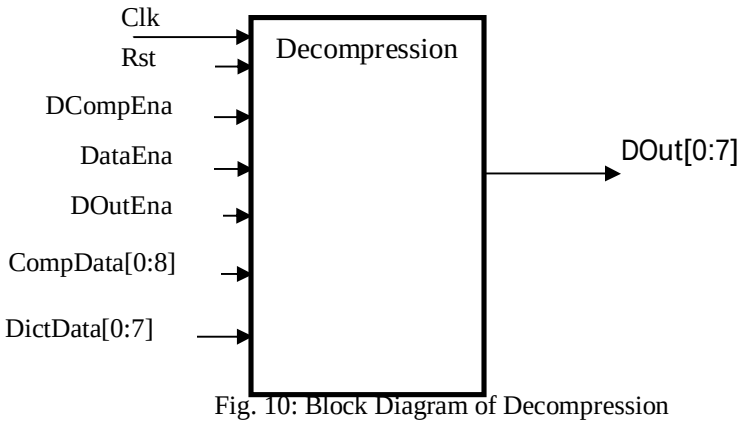

Fig. 10: Block Diagram of Decompression

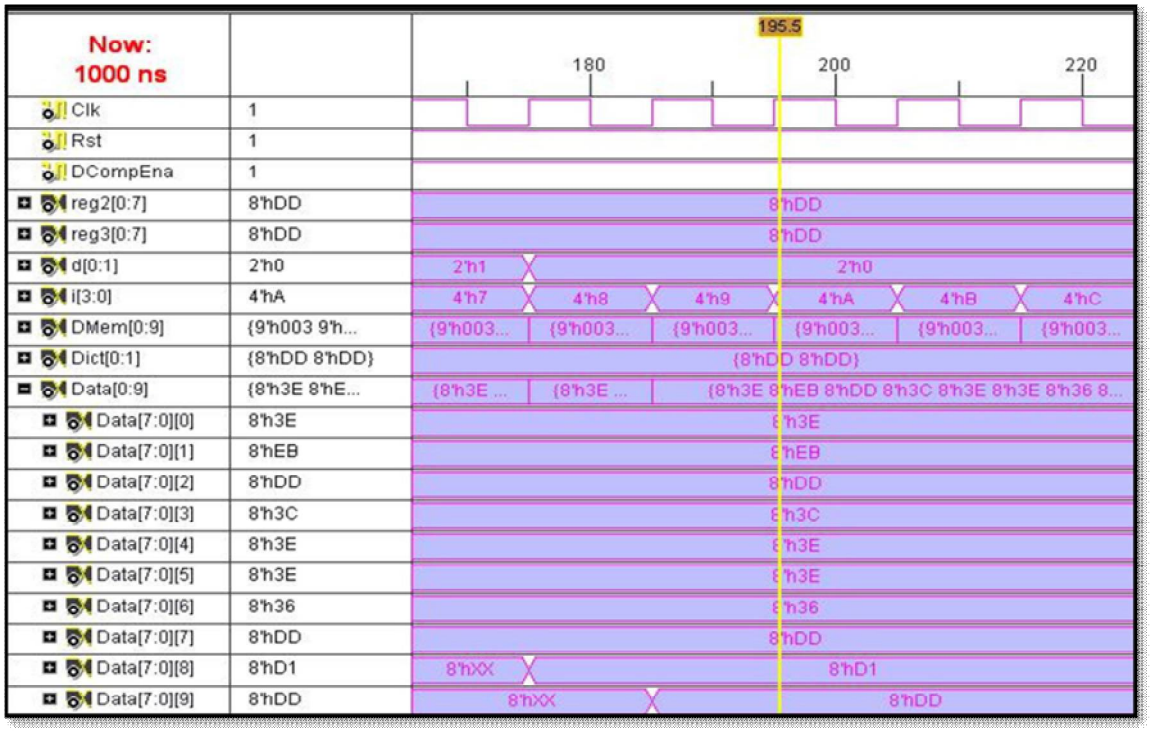

Fig. 11: Waveform of Decompression

Compression Ratio Comparison

Fig. 12 compares the proposed approach with the existing code compression techniques. The dictionary-based code compression techniques provide compression ratio of 97.5%. Later some techniques were proposed to improve the dictionary based compression technique by considering mismatches. Thus the improved dictionary based code compression scheme could attain a compression ratio of 95%. Our proposed code compression algorithm using the selected masks (2-bit mask) is sufficient to create 100% matching patterns and thereby improves the compression ratio to 87.5%.

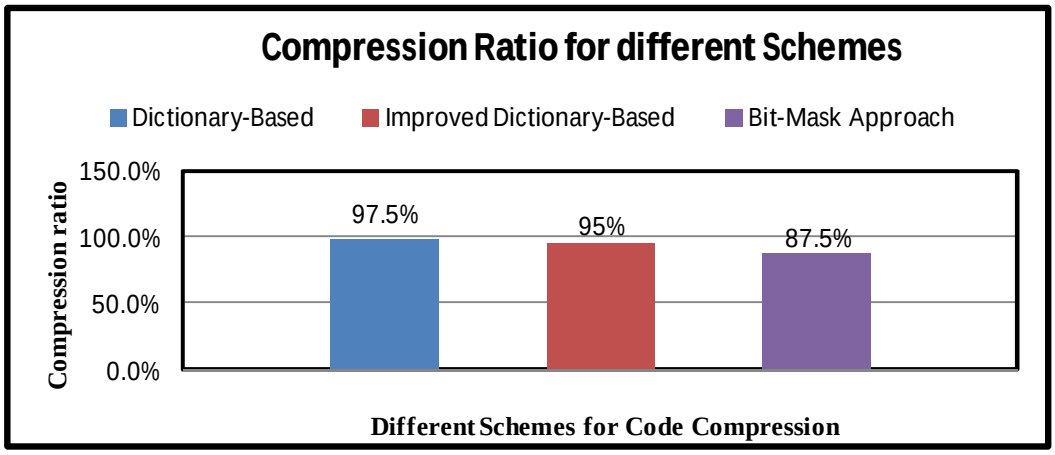

Fig. 12: Compression Ratio for different Schemes

VI. CONCLUSION

We have simulated the dictionary based code compression technique using bitmasks in Xilinx ISE Design Suite. In this paper, we presented an efficient code compression scheme using bit-masks that can significantly increase the number of matching patterns. This approach can handle multiple bit mismatches without incurring any compression or decompression penalty. Our approach outperforms the existing dictionary-based techniques giving a compression ratio of 87.5%.

REFERENCES

[1] Wolfe and A. Chanin, "Executing compressed programs on an embedded RISC architecture," in Proc. 25th Annu. Int. Symp.Microarchitecture, Dec. 1992, pp. 81–91.

[2] C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge, "Improving code density using compression techniques," in Proc. 30th Annu. ACM/IEEE Int. Symp. MICRO, Dec. 1997, pp. 194–203.

[3] S.-W. Seong and P. Mishra, "A bitmask-based code compression technique for embedded systems," in Proc IEEE/ACM ICCAD, Nov.2006, pp. 251–254.

[4] S.-W. Seong and P. Mishra, "An efficient code compression technique using application-aware bitmask and dictionary selectionmethods," in Proc. DATE, 2007, pp. 1–6.

[5] M. Thuresson and P. Stenstrom, "Evaluation of extended dictionary based static code compression schemes," in Proc. 2nd Conf. Comput. Frontiers, 2005, pp. 77–86

[6] S. Liao, S. Devadas, and K. Keutzer. Code density optimization for embedded dsp processors using data compression techniques. InProceedings of Advanced Research in VLSI, pages 393–399, 1995.

[7] H. Lekatsas and W. Wolf. Samc: A code compression algorithm for embedded processors. IEEE Transactions on Computer-AideDesign of Integrated Circuits and Systems, 18(12):1689–1701, Dec.1999.

[8] N. Ishiura and M. Yamaguchi. Instruction code compression for application specific vliw processors based of automatic fieldpartitioning. In Proceedings of Synthesis and System Integration of Mixed Technologies (SASIMI), pages 105–109, 1997.

[9] S. Nam, I. Park, and C. Kyung. Improving dictionary-based code compression in VLIW architectures. IEICE Trans. Fundamentals, A(11):2318–2324, November 1999.

[10] S. Larin and T. Conte. Compiler-driven cached code compression schemes for embedded ilp processors. In Proceedings of International Symposium on Microarchitecture (MICRO), pages 82–91, 1999.

[11] Y. Xie, W. Wolf, and H. Lekatsas. Code compression for vliw processors using variable-to-fixed coding. In Proceedings of International Symposium on System Synthesis (ISSS), pages 138–143, 2002.

[12] J. Prakash, C. Sandeep, P. Shankar, and Y. Srikant. A simple and fast scheme for code compression for vliw processors. In Proceedings of Data Compression Conference (DCC), page 444, 2003.

[13] M. Ros and P. Sutton. A hamming distance based vliw/epic code compression technique. In Proceedings of Compilers, Architectures,Synthesis for Embedded Systems (CASES), pages 132–139, 2004.

[14] W. J. Wang and C. H. Lin, "An improved BitMask based code compression algorithm for embedded ystems," in Proc. IEEE Int Symp. Electron. Syst. Design, Dec. 2011, pp. 152–157.