COMPRESSION OF FPGA BITSTREAMS USING A NOVEL BITMASK AND RLE ALGORITHM

[1]RUI YU, [2]YIBIN LI

[1,2]Shandong University
E-mail: jiou0962@gmail.com

Abstract- The configuration file size of a field-programmable gate array (FPGA) has increased rapidly. Thus, compression techniques are used to decrease the size of bitstreams. Two parameters are employed to measure compression, namely, compression ratio (CR) and decompression speed. Existing studies in this field have focused on two directions: small CR with slow decompression speed, and fast decompression speed at the cost of CR. This paper proposes a novel decode-aware compression technique with small compression ratio and minimal decompression time. The paper contributes to the literature by 1) proposing a novel dictionary selection algorithm that can avoid dictionary repetition, and 2) introducing an improved parameter selection that can provide a better CR. The experimental results showed that our approach outperforms the existing compression maximum of 15% in terms of compression ratio. Moreover, the decompression hardware can operate at 330 MHZ on a Virtex-5 FPGA.

Keyword- FPGA, Bitstream, Compression, Dictionary Compression

I. INTRODUCTION

Field-programmable gate array (FPGA) is an integrated circuit that can be configured by a customer or a designer after manufacturing. The function of FPGA can be customized by loading the design (a formatted bitstream file) into the configuration memory. With the development of semiconductor technology, a growing number of resources are being integrated into FPGA, causing the configuration file to increase rapidly. Given that the configuration file for FPGA has to be stored in an external or internal memory drive (flash memory) as bitstreams, limited memory capacity and access bandwidth become the key factors for determining whether or not a design can be configured and how fast the FPGA can recover to work after the device is switched on. In many applications, such as wireless sensor networks, adopting a memory with more capacity or improved bandwidth is expensive. However, the compression technique can lessen the memory constraint by reducing the size of the bitstreams and improving the throughput capacity without increasing expensive storage overhead. Compression ratio (CR) is generally used to evaluate the performance of compression algorithms, which is defined as follows:

$$CR = \frac{\text{Compressed program size}}{\text{Original program size}}$$

Thus, a smaller CR indicates a better compression technique when the decoding speed is not considered. However, the efficiency of decompression (i.e., speed of decoding and re source consumption of the decompression engine) is more important than a smaller CR in many instances. For example, the decoding speed is more important than a better CR for applications with high request for real-time performance. For certain low-power applications or applications designed for low-end FPGAs, a complex decompression engine with high resource consumption is unacceptable.

II. RELATED WORK

The existing techniques in bitstream compression can be classified into two categories based on whether or not special hardware support is needed during decompression. Several approaches require special hardware features when accessing the configuration memory, such as dynamic reconfiguration or frame read back, which are provided only by certain advanced FPGAs. For example, frame reordering and frequent frame read back are used to obtain better redundancy. The difference among consecutive frames is encoded by either Huffman-based run length encoding (RLE) or Lempel–Ziv–Storer–Szymanski (LZSS) compression. Such a sophisticated encoding method can produce excellent CR. However, the decompression overhead is not mentioned in, which is important for applications with high request for real-time performance or applications designed for low-end FPGAs.

Meanwhile, another kind of compression technique uses only the redundancy in the configuration file and does not require any special hardware support; therefore, this compression technique can be used by virtually all kinds of FPGAs. First, the compression divides the entire configuration file (configuration bitstreams) into several short words, and then compresses them using the usual algorithms, such as the Huffman compression, arithmetic compression, or dictionary-based compression. Among these techniques, LZSS-based compression has received significant attention because the compressed stream can be decompressed efficiently without complex logic. For example, Xilinx employed a bitstream compression

technique based on LZ77. Huebner et al. presented an LZSS-based compression for Xilinx Virtex-2 FPGA. Furthermore, a novel method was proposed in paper to reduce the configuration bits of the lookup tables by removing redundancies.

Among the various compression algorithms that do not need special hardware, bitmask-based compression is suitable for bitstream compression because of small CR and a simple decoding scheme. However, the original algorithm (bitmask-based compression) is presented for instruction compression; thus, it cannot be adapted for bitstream compression on FPGAs. Using variable-length coding (VLC) poses challenges to the design of decompression circuit because VLC requires an expensive buffering circuit. The compression technique proposed in studies combines bitmask compression with RLE, which provides better results (small CR with fast decoding speed). However, the compression algorithm of these techniques has the same deficiency, that is, dictionary repeat, which is disadvantageous in obtaining a smaller CR.

Our method absorbs the advantages of the previous compression techniques, combining BMC with RLE compression. To compress bitstreams more effectively,we propose a novel dictionary selection algorithm which can avoid dictionary repetition. Our proposed bitstream compression technique does not require special hardware support.

Also, the resource consumption of the decompression logic is simple. Thus, it can be used by virtually all kinds of FPGAs.

The rest of this paper is organized as follows. Section III provides the basic compression algorithms involved in this study. The proposed novel bitmask and RLE compression technique (BMR) and the decompression engine are described in Section IV. Experimental results are presented in Section V. The conclusion is provided in Section VI.

III. BASIS COMPRESSION ALGORITHM

This section introduces three compression algorithms, namely, dictionary compression, bitmask compression, and RLE compression. Among these algorithms, dictionary compression is the foundation of the latter two compressions. The latter two are relatively independent compression algorithms. In our BMR compression, bitmask compression is combined with RLE compression to obtain a better CR.

A. Dictionary Compression

The basic idea for the dictionary-based code compression is to encode commonly occurring symbol sequences using a dictionary, where the repeating words are replaced by the dictionary index. The compressed file consists of both compressed and uncompressed symbols. Figure 1 shows an example of dictionary compression. The original program consists of 10 8-bit patterns, i.e., a total of 80 bits. The dictionary has two entries (16-bit). The compressed bitstreams require 62 bits, and the dictionary needs 16 bits. Thus, in this example, the CR is 97.5%.

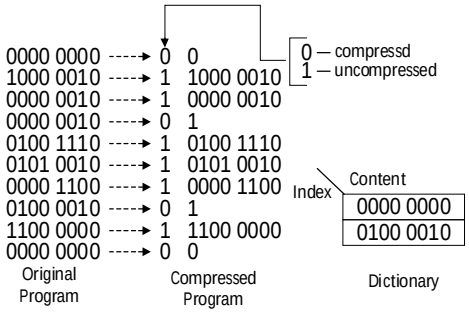


Fig. 1. An example of dictionary-based code compression.

B. Bitmask Compression

In many cases, the dictionary compression is combined with other compression algorithms to obtain a better compression ratio. In this study, vast similar symbols have been found in the configuration bitstreams, such as “00000000”, “100000001”, which has a difference of only 2 bits. Therefore, bitmask compression based on dictionary compression can be used to save more bits than when only the dictionary compression is used. Fig. 2 shows an example of bitmask compression. The original bitstream is 80 bits, which is reduced to 70 bits, including cryptograph and dictionary entries, after bitmark decompression. Thus, in this example, CR was reduced to 87.5%.

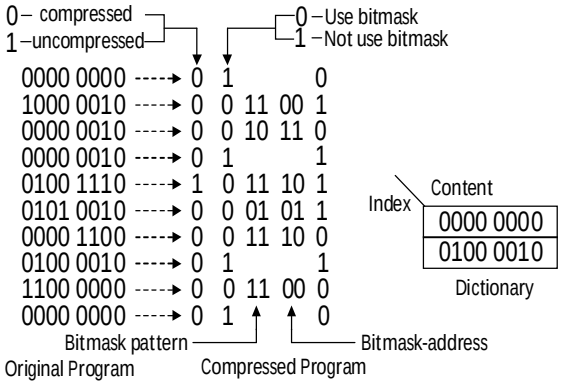


Fig. 2. Bitstream compression using the bitmask selection approach.

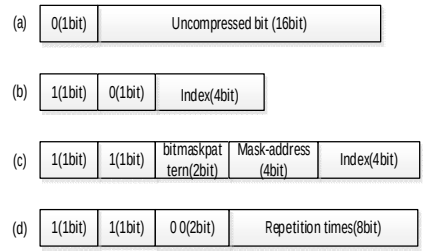


Fig. 3. Encoding format for the proposed compression technique.

Fig. 3 shows an example of encoding format for the proposed compression technique: (a) Uncompressed symbol; (b) Symbol compressed with dictionary compression; (c) Symbol compressed with bitmask compression; and (d) Symbol compressed with Bitmask and RLE Compression. According to paper, only consider six kinds of bitmask patterns (1s, 2s, 2f, 3s, 3s, 3f) have been considered in our study of bitstream compression.

C. RLE Compression

The configuration bitstream typically contains consecutive repeating bit sequences. Although the bitmask compression encodes such patterns using the same repeated compressed words, as suggested in and, the RLE of these consecutive repeating bit sequences may obtain a better CR. Fortunately, the encoding scheme does not require extra bits. The bitmask pattern value '00' is never employed because it is a complete match; therefore, RLE compression is encoded into bitmask compression using the zero bitmask pattern. Repetitions can be encoded without changing the code format of bit mask compression using the bitmask pattern “00” as a marker of RLE compression. Fig. 4(d) shows the encoding format for the RLE compression. Our results show that Fig. 4(d) is only a special situation of Fig. 4(d).

Figure 5 shows the bitmask and RLE compression. The original program contains symbol “00000000”, which is consecutively repeated six times. According to common bitmask compression, these consecutively repeating symbols are compressed with consecutively repeated compressed words. For this instance, the first symbol “00000000” in this program is encoded normally; however, the remaining five repeated symbols are encoded by RLE. Our method marks such a situation using a special bitmask pattern value “00”. The times of repetition is encoded as mask address and dictionary index combined together, which is “0000 0100” in this example (i.e., the bitmask address is “0000” and the dictionary index is “0100”).

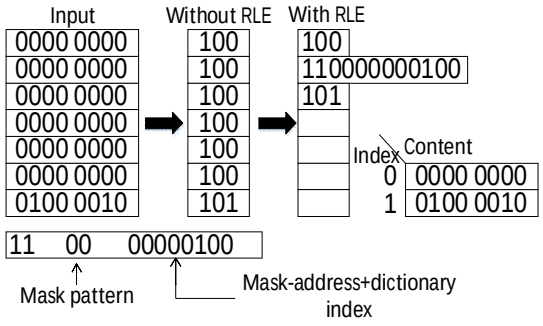


Fig. 4. RLE Compression

IV. BMR COMPRESSION AND DECOMPRESSION ENGINE

Figure 5 shows our bitstream compression framework, which contains three parts: compression, decode-aware placement algorithm, and decompression. The former two parts are executed on a PC, whereas decompression is conducted during run time. On the compression side, the original FPGA configuration bitstream is analyzed for bitmask pattern selection and dictionary selection. The compressed file is then generated using bitmask compression with RLE. Thereafter, decode-aware placement algorithm is used to store the compressed file into the flash memory, so that the compressed file can be decompressed efficiently at run time. During run time, the compressed file is delivered from the flash memory into the decompression logic. The original configuration bitstream reconstructed by the decompression engine is then loaded into the configuration memory. Our decode-aware placement algorithm (Section B) and decompression engine (Section C) are similar to those presented in. For simplicity, the two parts (Sections B and C) are introduced in a straightforward manner.

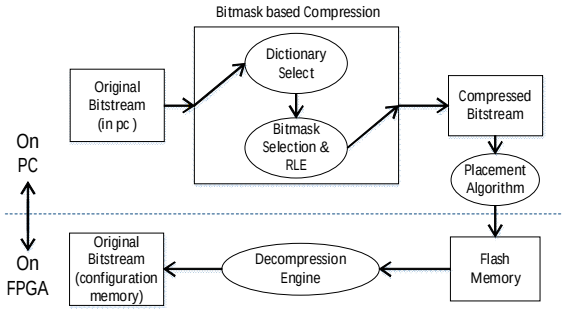


Fig. 5. Our bitstream compression framework

A. BMR compression

Four main steps are included in our compression algorithm (shown in Algorithm 1). First, the original configuration file (i.e., the input bitstream) is divided into a sequence of symbols with fixed length of w (16 or 32). The suitable parameter and dictionary used for bitmask-based and RLE compression are then selected in step 2. The symbol sequence is compressed by bitmask and RLE compression in step 3. Finally, the compressed bitstream is stored into a flash memory device using our decode-aware placement algorithm.

Algorithm 1: Novel Bitstream & REL Compression

//Input : Input bit stream
//Output : Compressed bitstream placed in memory
Step 1 : Divide input bit stream into fixed size symbol sequence (SQ).
Step2 : do parameter and dictionary selection()
Step 3 : Compress symbol into code sequence using Bitmask and RLE Compression.
Step 4 : Perform decode aware placement of code().

Fig. 6. Algorithm 1: Novel Bitstream and REL Compression.

The main difference in our compression methods is step 2 (i.e., function parameter and dictionary selection). The function parameter and dictionary selection (shown in Fig. 1) is used to select the parameters and the dictionary, including dictionary, mask pattern, and dictionary capacity. In this function, both dictionary capacity and mask pattern are considered. For each candidate dictionary capacity (16, 32, 64, 128, 256, or 512 entries in our methods) and each mask pattern (1s, 2s, 2f, 3s, 3s, 3f), the novel dictionary selection algorithm is executed to find the dictionary with maximum bit savings.

To select the dictionary entries, we developed a novel dictionary selection technique that considers bit savings as a metric. Fig. 9 shows the novel dictionary selection algorithm. Step 1: A directed graph is constructed. The nodes are the unique w bit symbols from the input symbol sequence SL. Edges are constructed for all the nodes in this directed graph based on the following rules: for any two nodes (e.g., nodes A and B) that can be matched with each other through

```
parameter and dictionary selection()
//input: fixed size symbol sequences
//output: Dictionary, mask pattern, dictionary
          capacity
{
  maximum total bit saves=0;
  for( each Candidate dictionary capacity )
   {
     for (each Candidate mask pattern )
     {
       Dictionary = Dictionary Selection();
       calculate current total bit saves;
       if(current total bit saves> max total
         bit saves)
        max total bit saves= calculate current
        total bit saves;
     }
   }
 return  Dictionary , mask pattern, dictionary
          capacity;
}
```

Fig. 7. Parameter and dictionary selection

the bitmask pattern, two edges are added to the graph (one edge points node A to node B; the other edge points node B to node A). Values are then added to each edge and node. The value of the node indicates the bit savings of the node when the node is selected as dictionary entry. The value of the edges represents the number of bits the edges saved. Step 2: The total bit savings of each the node (total bit savings of a node is the value of the node pulsing all the value of its output edges) is calculated. Step 3: Node I (the node with the maximum total bit savings) is selected as a new dictionary entry. Step 4: The present node I is removed from the graph and all the linked edges are deleted as well. Step 5: All the input edges of the nodes that can be matched with node I are deleted. Step 6: Steps 2 to 5 are repeated until dictionary D is full or the graph is empty. Finally, dictionary D is obtained. This dictionary selection technique is different from the one introduced in paper. According to, when nodes A and node B match each other through the bitmask pattern, the dictionary constructs only one edge without direction. The node (e.g., node I) is deleted using the following rule: For each node v that is adjacent to node I, the edge is removed when the edge between the adjacent nodes and I is duplicated. This method leads to an extremely bad situation that leads to a larger CR. More than one dictionary entry matches the same symbol. Fig. 8 shows an illustration of the difference between our dictionary selection technique and the method proposed by paper.

Fig. 8(a) illustrates the method proposed by paper. In this graph, node A has a total savings of 52 (40+12), node B has 34 (20+12+12), C has 42 (30+12), and node D has 36. Therefore, A is chosen as the best candidate and inserted into dictionary D. After being inserted into the dictionary, node A is removed from the graph. The edge between nodes A and B are then removed as well. In the next loop, node C and the edge between nodes B and C are deleted. In this case, more than one

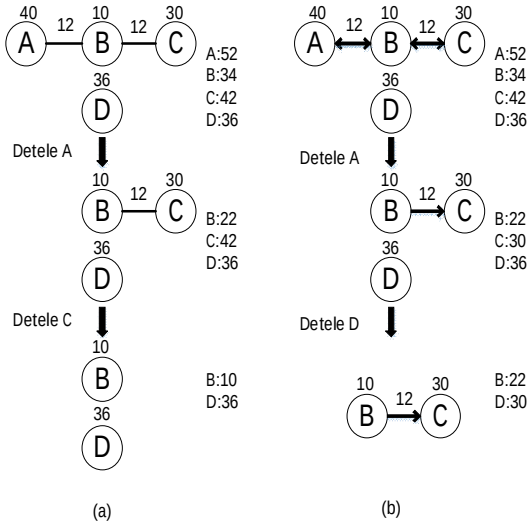


Fig. 8(a). Dictionary selection method proposed by paper ; (b) Our improved dictionary selection method.

dictionary entry (nodes A and C) match the same symbol (symbol B), leading to a larger CR because the total bit savings of node C is inaccurate. During compression, dictionary entry A is used to match with node B. Thus, the edge between nodes B and C becomes meaningless and repetitive. In our approach, a directed graph is used to represent the matching relationship between symbols. In this case, when one node that matches node N (N is a common node) is selected as a dictionary entry and removed from the directed graph, all the edges ending with node N should be deleted. For example, in Fig. 8(b), edge C, which points to B must also be deleted when node A is deleted (this edge is repeated with edge A pointing to B). The experimental results in Section V show that our

method produces a smaller CR because dictionary repetition has been effectively avoided.

B. Decode Aware Placement of Compressed Bitstreams

The basic idea of this program is to separate the original single VLC bitstream into multiple FLC (fixed-length coding) bitstreams for storage. During decompression, these FLC bitstreams are buffered separately, and then used to reconstruct the original bitstream using bitmask decoding. Given that the buffering circuitry for FLC bitstreams is simpler than that of the VLC bitstreams, the overall decoding performance is promoted even when multiple FLC buffers are used. The "power-two n-bit stream," is used in the remaining sections. We describe how to split the original compressed bitstream into multiple FLC bitstreams and how to place these FLC bitstreams into the memory such that the original bitstream can be reconstructed during decompression.

Algorithm:Novel Dictionary Selection

//Input: symbol sequence SL, symbol length, dictionary capacity N, mask patterns.

//Output: Dictionary entries D

Step 1: Construct graph G=(V,E) from SL.(A,B are Nodes in SL that can match each other by bitmask, then build two edge , one from A to B ,One form B to A)

Step 2: Calculate bit savings of all nodes in V.

Step 3: Select the node I which bit savings is maximal among all the node.

Step 4: Remove node I and all its edges from V ,then insert node I into dictionary D.

Step 5: For each node J which comes from node I , remove those edges which ending node is J.

Step 6: Repeat Steps 2 to 5 until dictionary is full or G is empty.

Return D

Fig. 9. Novel dictionary selection algorithm.

C. Decompression Engine

The schematic of our decompression engine (8-bit memory) is shown in Fig. 10. A barrel shifter (BBS) is displaced by an assemble buffer with a left shifter array (ABLSA). The basic operating principle of ABLSA is the use of an array of left shift registers to separately buffer the power-two bitstreams produced by the placement algorithm. Given that the code length of the compressed symbol is unique, as determined by the first two bits of a code, the length of a decompressed symbol can be easily obtained by checking the front bits (i.e., CS and BS). The shift register that holds the bits of the code is identified by the value of CS and BS. Finally, the original bitstream is assembled in the assemble buffer, and then loaded to the RLE decoders. When another shifter becomes empty, the next content of the compressed bitstream is loaded. This process continues until the last compressed symbol is decompressed.

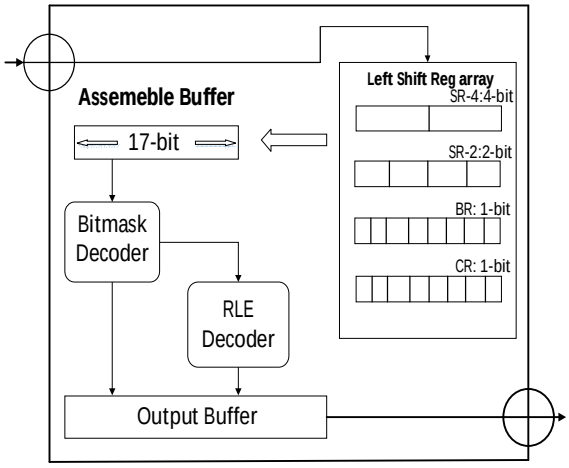


Fig. 10. Decompression engine

V. EXPERIMENTS

A. Compression Ratio

The bitmask-based compression technique proposed in this study is compared with the other main techniques, as shown in Fig. 11. Figure 11 (a) use the benchmarks of paper. The benchmarks of paper are used in Fig 11(b).The four different types of compression techniques that are evaluated are as follows: i) BMC - bit mask compression technique only; ii) LZSS - best known parameterized LZSS-based decompression accelerator proposed by Koch et al.; iii) pBMC+RLE - the BMC & RLE compression mentioned in; iv) BMR compression - the novel bitmask and RLE compression.

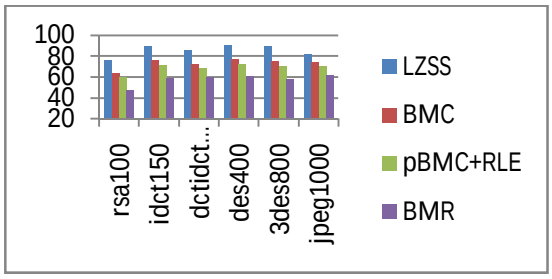
From fig.11(a), it can be seen that, the CR of BMR has a observable improvement (from 11.7% to 21.6%). However, the CR in Fig.11(b) only decrease slightly (from 2.08% to 5.26%). The reason is that the symbols of benchmarks in Fig11.(a) are relatively dispersive. Therefore, a large dictionary is needed for better compression. In paper, dictionary capacity of those benchmarks is a fixed value (512 entries). In this situation , dictionary repetition and the selection of dictionary capacity (too large or too small) affect CR enormously. while dictionary produced by our method avoid repetition, and dictionary capacity is selected from several candidate values(16, 32, 64, 128, 256, 512). Therefore, CR in fig.11(a) improve observably.

On the contrary, the symbols of benchmarks in Fig11.(b) are relatively centralized. As a result, a few dictionary entries can play a key role. In this situation, only missing of those key dictionary entries or too large size of dictionary can bring a higher CR. However, when missing of those key entries by using the

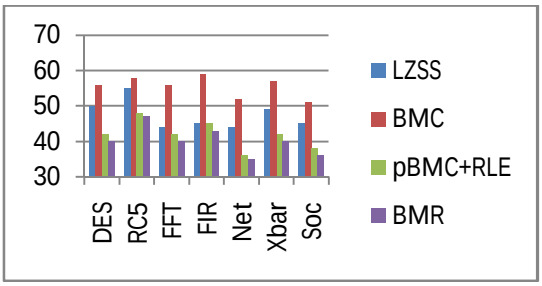
method proposed in paper rarely happen, as those decisive dictionary entries can be selected effectively. In this situation, dictionary repetition became the secondary cause which only affect CR slightly. Therefore, in this case, the improvement of CR become indistinctive.

B. Decompression Efficiency

Four indicators are used to measure the efficiency of the decompression engine: maximum frequency, slice usage, and power. The decompression units are synthesized for variable-length bitmask-based compression, difference Vector-based compression, and our proposed approach on Xilinx Vertex 5 family XUPV5-LX110T device FG1136 package using ISE14.2. The results are reported in Table I.



(a)



(b)

Fig. 11. Comparison of compression ratios

TABLE I
Operating frequency, slice usage, and power

Type	Frequency (MHz)	Slice Usage	Power(w)
Bitmask decoder	254	260	6.69
LZSS	336	64	1.75
Our Approach	329	146	2.46

CONCLUSION

The existing compression algorithm of bitstream compression focused on two directions: small CR with slow decompression speed, and fast decompression speed at the cost of CR. This paper proposed a novel bitmask and RLE compression algorithm that can achieve both excellent compression efficiency and fast decompression speed. A smarter dictionary selection algorithm was designed to avoid dictionary repetition. More compressed parameters (capacity of dictionary entries) were considered to better fit different applications. The combinations of bitmask-based compression and run length encoding provide an efficiently compressed bitstream; thus, more configuration information can be stored using the same memory. The experimental results of this study demonstrated that our technique improved the compression ratio by 5% to 15% and the decompression engine could operate at 330 MHz in Virtex v5 FPGAs.

REFERENCES:

[1] X. Qin, C. Muthry, and P. Mishra, "Decoding-aware compression of FPGA bitstreams," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 19, p. 411-419, 2011.

[2] J. H. Pan, T. Mitra, and W. Wong, "Configuration bitstream compression for dynamically reconfigurable FPGAs," in Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on, 2004, p. 766-773.

[3] D. A. Huffman, "A method for the construction of minimum redundancy codes," Proceedings of the IRE, vol. 40, p. 1098-1101, 1952.

[4] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," ACM Transactions on Information Systems (TOIS), vol. 16, p. 256-294, 1998.

[5] A. Khu, "Xilinx FPGA configuration data compression and decompression," WP152 ed. Xilinx, San Jose, CA, 2001.

[6] M. Huebner, M. Ullmann, F. Weissel, and J. Becker, "Real-time configuration code decompression for dynamic FPGA self-reconfiguration," in Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004, p. 138.

[7] A. Abdelhadi and G. G. Lemieux, "Configuration Bitstream Reduction for SRAM-based FPGAs by Enumerating LUT Input Permutations," in Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on, 2011, p. 20-26.

[8] S. Seong and P. Mishra, "Bitmask-based code compression for embedded systems," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 27, p. 673-685, 2008.

[9] P. M. Sandeep and C. S. Manikandababu, "Compression and decompression of FPGA bitstreams," in Computer Communication and Informatics (ICCCI), 2013 International Conference on, 2013, p. 1-4.

[10] P. Hemnath and V. Prabhu, "Compression of FPGA bitstreams using improved RLE algorithm," in Information Communication and Embedded Systems (ICICES), 2013 International Conference on, 2013, p. 834-839.

[11] S. Hauck and W. D. Wilson, "Runlength compression techniques for FPGA configurations," in Field-Programmable Custom Computing Machines, 1999. FCCM'99. Proceedings. Seventh Annual IEEE Symposium on, 1999, p. 286-287.

[12] D. Koch, C. Beckhoff, and J. Teich, "Bitstream Decompression for High Speed FPGA Configuration from Slow Memories.," in FPT, 2007, p. 161-168.

★★★