International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 5 (2018) pp. 2872-2878

Cascading of Compression Algorithm to Reduce Redundancy in FPGAs Configuration Bitstream -A Novel Technique

Pravin N. Matte
G. H. Raisoni College of Engineering and Management, Savitribai Phule Pune University, Pune, India - 412207, pravin.matte@raisoni.net

Dr. D. D. Shah
JSPM's Imperial College of Engineering and Research, Savitribai Phule Pune University, Pune, India - 412207.

Abstract

In recent times with increasing in capacity of FPGA, bitstream size of configuration file is increasing. This leads to need of more configuration memory. Also, reconfiguration time increases with increased in bitstream size. Redundant bitstream increases overheads in reconfiguration. Bitstream compression is technique to reduce redundancy in configuration bitstream. Various researchers have used existing and modified traditional lossless bitstream compression techniques. Existing studies in this field have focused on (i) small compression ratio with less decompression speed and (ii) more decompression speed compromising the compression ratio. We proposed an approach of cascading of bitstream compression algorithm to further reduce the bitstream. It is found that cascading of RLE with Arithmetic, LZW, Huffman and LZSS algorithm gives better compression ratio. Our approach of cascading of compression algorithm improves compression ration by an average of ≈ 23 %.

Keywords: bitstream, field programmable gate array, compression, decompression, compression ratio.

INTRODUCTION

The computing systems build using field programmable gate arrays, aims to achieve high performance close to application specific processor and flexibility compared to general purpose processor. FPGAs are mostly used in the reconfigurable systems. Reconfigurable computing intend to bridge gaps between hardware and software. Hardware description languages like VHDL or Verilog are used to describe logic circuits. A configuration bitstream file is generated which carries the logic circuit's information and is used to configure FPGA. Configuration bitstream is stored in configuration memory. Memory may be internal or external to FPGA device. Size of configuration bitstream limits memory size and access bandwidth. Limited memory size puts restriction on required functionalities that can be included in configuration bitstream. Reconfiguration time is also affected by size of the configuration bitstream. One of the way by which the size of configuration bitstream file is reduce is compression.

Data compression is the technique used to remove the redundant data reducing the size of the file. Lossy and lossless compressions are the two types of the compression methods. In lossy compression technique data is retrieved but loss of information occurs. Human eyes have limitations. These limitations are used by lossy compression algorithms to compress data by discarding unseen information by human eye. Contrast to lossy compression, lossless compression algorithms retrieved original data upon decompression. In FPGA configuration lossless bitstream compression algorithms are used as loss of data during compression or decompression and cannot be afforded.

Compression ratio (C.R.) is defined by following formula,

$$C.R. = \frac{\text{Compressed Bitstream size}}{\text{Original Bitstream size}} \quad (1)$$

Lower the compression ratio better is the compression rate.

As shown in Figure 1 configuration bitstream in FPGA are compressed by applying configuration algorithms and then stored in configuration memory. Compressed bitstream is decompressed by decompression hardware and FPGA is then reconfigured.

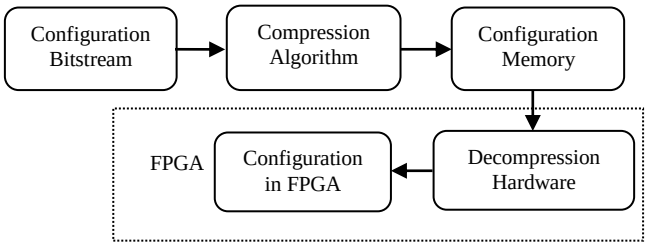

Figure 1: Compression and Decompression of Configuration Bitstream in FPGA.

In this paper we have discussed related work done in section II. The compression algorithms selected for configuration bitstream compression is briefly discussed in section III. Our approach of cascading of compression algorithms is explained in section IV. Section V describes experimental set up. Finally, results are presented in section VI and the paper is concluded by discussing conclusion and future scope in section VII.

RELATED WORK

Configuration bitstream in FPGA can be compressed by numerous compression algorithms. Basically, generic and format specific compression are the two categories of bitstream compression techniques based on how the redundancies in FPGA are utilized.

Wolfe and Chanin [1] applied proposed compression technique for the first time for embedded processor. Huffman coding is used to compress programs and memory to store programs. Latter researcher applied compression techniques to FPGA configuration bitstreams.

Format specific bitstream compression techniques require specially hardware features to access configuration memory like partial reconfiguration, wildcard registers or frame read back. XC6200 series FPGAs in [2] supports wildcard based compression scheme. Frame reordering and active frame read back used in [3] to achieve better redundancy.

Other Compression technique uses only the redundancy in the configuration file and does not require any special hardware support; therefore this compression technique is used virtually by all kinds of FPGAs. First, the compression divides the entire configuration file into several short words, and then compress them using the usual algorithms, such as Huffman compression [4], Arithmetic Compression [5] , or Dictionary based compression [6]. In [7] decode aware compression technique is proposed where bitstream is compressed, considering bottleneck of decompression hardware. Bitmask compression and Run Length Encoding (RLE) is used to compress configuration bitstream. A bitstream compression algorithm based on LZ77 is introduced in [8]. LZSS based technique for XCV2000E FPGA is proposed in [9]. It is observed in [10] that simple algorithm like LZSS maintain decompression overheads within range. This leads to compromise on compression ratio. A dictionary based approach is proposed in [11] for configuration bitstream configuration of FPGA based embedded system. The input configuration bitstream file is read in reverse order sequentially. LZW compression algorithm is used to frame dictionary and the index. In [12] various modifications to standard algorithms are discussed to fulfill the objectives of the compression ratio, throughput, and resource overheads. Further [12] has developed a benchmark including several dense configuration bitstream. These bitstream are hard to compress and are kept for free access at [13].

Our configuration bitstream compression algorithm is based on cascading of compression algorithms to compress it further by applying another algorithm. We have used benchmarks from [13] to study impact of cascading.

LOSSELESS COMPRESSION TECHNIQUES

Lossy and lossless are the two data compression techniques. In lossy compression techniques data are compressed with loss of data but in lossless data compression no loss of data occurs. By using lossless compression technique we can achieve compression ratio but is less than lossy compression algorithms. Some of the traditionally used lossless compression techniques like Run Length encoding, Huffman encoding, Arithmetic encoding, and LZW is described briefly.

A. Run Length Encoding

The most simple lossless data compression algorithm is Run Length Encoding (RLE). Long runs of symbol are replaced by symbol and symbol length. RLE is useful in case of more repetitions. But if there are no repetitions in input data, then instead of compression file expands. Following table illustrate run length encoding concept.

Table 1: Run Length Encoding

Input File	Size (bytes)	Compressed File	Size (bytes)
aaa	3	a3	2
aaab	4	a3b	3
aaabbbbb	7	a3b5	4
abababab	8	a1b1a1b1a1b1a 1b1	16

B. Huffman Encoding

Huffman encoding is a lossless compression technique developed by Devid A. Huffman in 1952. In this method original data is retrieved after decompression. Redundant data is removed in compression and reconstructed during decompression. Such lossless methods are mostly used by data communication system, transmission- reception and data storage systems. Huffman encoding is used because of its variable length coding (VLC) features. Variable length code table has been derived by estimating probability of occurrence of each symbol. Higher probability of occurrence has shorter code word length and less probability of occurrence have longer code word. The Huffman algorithm is briefly summarized in Table 2.

Table 2: Huffman Encoding

Symbol	Count	log(1/p)	Code	Subtotal (bits)
a	16	1.38	0	16
b	8	2.48	100	24
c	7	2.70	101	21
d	7	2.70	110	21
E	6	2.96	111	18
Total bits				87

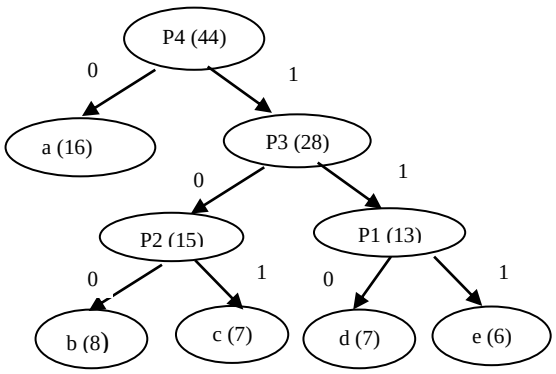

Figure 2: Huffman Encoding

C. Arithmetic Encoding

Arithmetic encoding is generation of variable length codes. It is very useful in dealing with binary sources and alphabets with highly distorted probabilities. In Arithmetic coding, stream of input symbols are taken and is replaced with a single floating point number. As the size and complexity increases the bits needed to represent the output increases. There is prefix 0, and the stream represents a fractional binary numbers between 0 & 1.

Consider a string "bccb" taken from a set of alphabet {a, b, c}. When first "b" is coded, all symbols have a 33% probability. The range between low and high is divided into the symbols according to their probabilities. Initially the range is between 0 and 1; we restrict ourselves each time to the subintervals that codify the given symbol. At the end, the whole sequence can be codified by any of the number in the final range (0.639, 0.650).

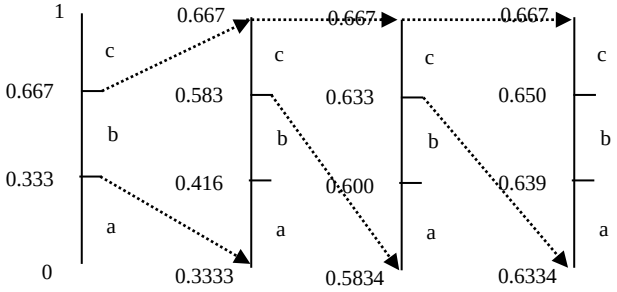

Figure 3: Arithmetic Encoding

D. The Lempel Ziv Welch Algorithm

Lempel-Ziv-Welch (LZW) compression algorithm was published in 1984. It is a universal algorithm for lossless data compression created by Abraham Lempel, Jacob Ziv, and Terry Welch. In LZW dictionary, variable length string is constructed from the input data and each occurrence of string in input data is replaced by the index to the appropriate entry in dictionary. Input data is compressed by adopting following steps.

- The source sequence is sequentially passed into strings that have not appeared so far.
- After every separation, input sequence is scanned until the shortest string that has not been marked off before, is found.
- This phrase is coded by giving the location of the prefix and value of last bit.

Assume input data as: "aaaabaaaa"

The above input data is compressed by applying LZW.

Table 3 illustrate LZW encoding.

Table 3: *LZW Encoding*

Positions	1	2	3	4	5
Input Sequence	a	aa	ab	aaa	a
Numerical representation	Φ a	1 a	1 b	2 a	1
Code (a = 0 and b = 1)	0	1,0	1,1	10,0	0

CASCEDING OF COMPRESSION ALGORITHM

Figure 4 depicts concept of cascading of compression algorithms. Configuration bitstream files are obtained for designs under consideration with the help of VLSI CAD tool. Bitstream is generated at the last step of FPGA based design flow. Compression algorithms: Arithmetic, Huffman, LZW, RLE and LZSS are chosen as either compression algorithm 1 or 2 during stage I or II of cascading of compression algorithm.

.

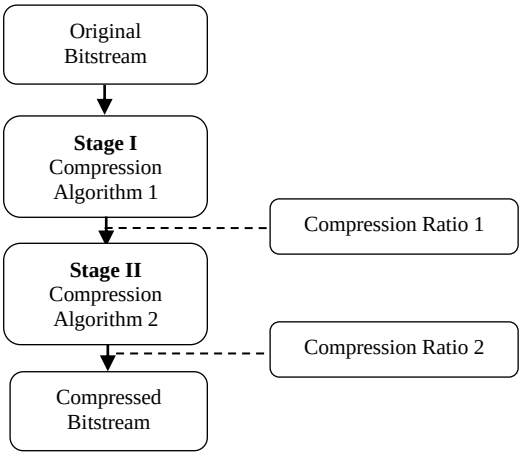

Figure 4: Cascading of Compression Algorithms.

In the stage I of cascading compression algorithm, the original configuration bitstream is given as input to the compression algorithm. Any compression algorithm among five is selected as compression algorithm 1. In stage II compressed configuration bitstream obtained in stage I is given as input to stage II. Also in stage II, any compression algorithm is selected as compression algorithm 2. In Figure 5.a Arithmetic algorithm is chosen as compression algorithm 1 and reset of algorithms including arithmetic algorithm are chosen as algorithm 2. In Figure 5.b to Figure 5.e compression algorithms 1 are changed to LZW, RLE and Huffman. Compression ratios are calculated in stage I and stage II. The overall compression ratio is calculated as ratio of compressed bitstream to the original bitstream. Bitstream are generated for each of the benchmark design by selecting different FPGA targets. For each of the bitstream obtained, steps mentioned in Figures 5.a to 5.e are followed and overall compression ratio is calculated.

Algorithm: Cascading of Compression Algorithms

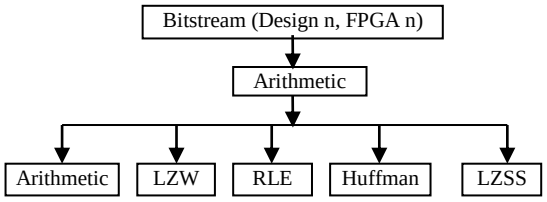

Figure 5.a: Arithmetic as Algorithm 1 and rest as Algorithm 2

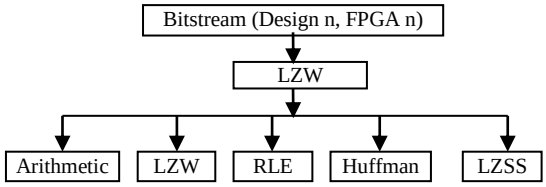

Figure 5.b: LZW as Algorithm 1 and rest as Algorithm 2

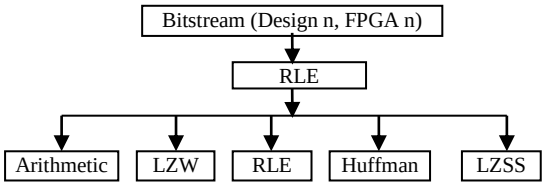

Figure 5.c: RLE as Algorithm 1 and rest as Algorithm 2

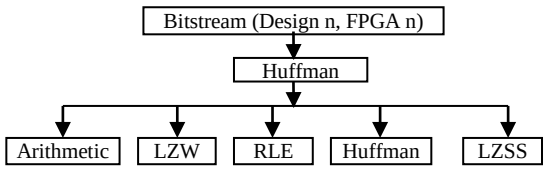

Figure 5.d: Huffman as Algorithm 1 and rest as Algorithm 2

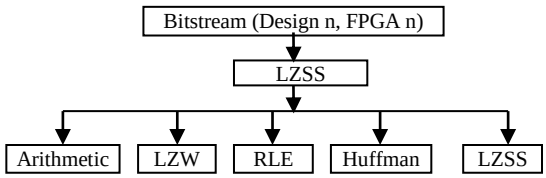

Figure 5.e: Huffman as Algorithm 1 and rest as Algorithm 2

EXPERIMENTAL SETUP

Bitstream compression & decompression algorithms are implemented in C and Verilog programming language. Since the algorithms used under study are lossless, decompression yields original bitstream files. Decompressors were developed using Xilinx ISE Design Suite 14.7. Also, we have used wxHexEditor tool to compare compressed and decompressed binary files after each stage of cascading. This confirms the compression and decompression of bitstream obtained in both the stages of cascading is correct or not. Experimental setup in Figure 6 and Figure 7 is used for compression and decompression of bitstream file.

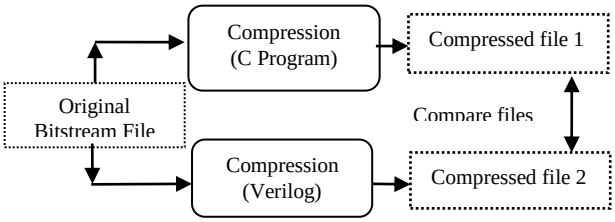

Figure 6: Compression using C and Verilog.

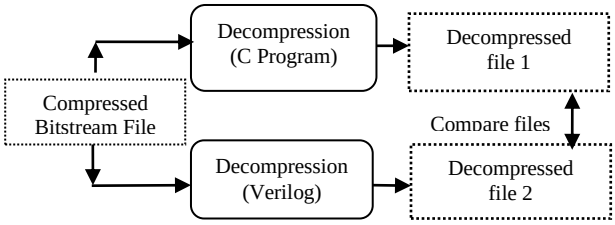

Figure 7: Decompression using C and Verilog.

Benchmark for compression is obtained from [13] which covered different category of application areas like cryptography, image processing, networks, and switches occupying most of the resources in FPGA.

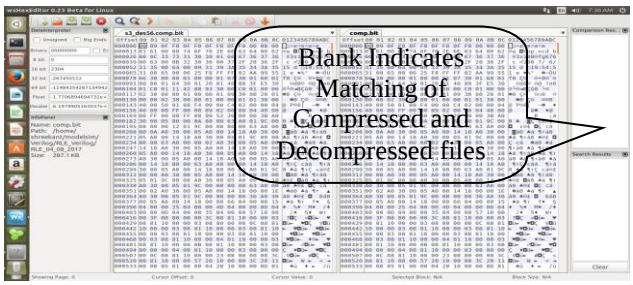

Figure 8: Comparisons of Bitstream file Obtained after Compression/Decompression using wxHexEditor.

RESULTS AND DISCUSSION

Arithmetic, LZW, RLE, Huffman and LZSS compression techniques studied on benchmark obtained from [13]. Figure 9, Figure 10 and Figure 11 shows compression ratio obtained by applying RLE, Huffman, Arithmetic, LZW and LZSS compression on bitstream of DSA, RC5, FFT, FIR, Xbar, RoCoNode, RoCoLink benchmark design implemented on Spartan-3, Vertex-II and Cyclone-II FPGA.

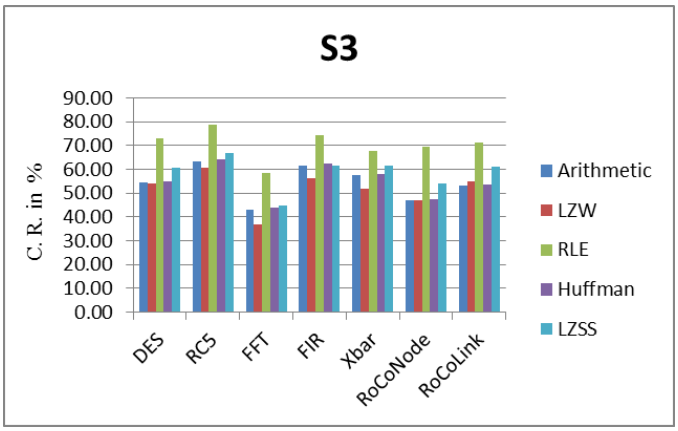

Figure 9: Compression Ratio Obtained in Stage One for Spartan-3

Table 3: Compression Ratio after Stage II

	A																							
Appl >	1	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2
Algo >	Arith	Arith	LZW	RLE	Huff	LZSS	LZW	Arith	LZW	RLE	Huff	LZSS	RLE	Arith	LZW	RLE	Huff	LZSS	Huff	Arith	LZW	RLE	Huff	LZSS
RoCoLink Cyc2	46.93	47.01	62.71	46.97	47.34	46.93	52.16	50.87	66.24	52.41	51.37	57.72	75.37	**45.76**	52.82	80.27	46.51	61.84	47.34	46.59	60.13	47.76	46.97	51.16
RoCoLink v2	31.66	31.06	39.17	30.06	31.66	33.48	29.83	**29.42**	38.18	29.96	29.65	33.13	54.08	29.61	30.55	60.38	30.08	37.00	33.39	30.63	35.46	29.77	30.9	31.74
RoCoLink S3	52.99	52.24	68.00	50.54	52.66	56.24	54.87	52.71	68.75	55.11	53.22	60.28	71.20	**49.69**	55.48	75.62	50.16	59.86	53.46	52.28	64.52	50.96	52.71	55.44
RoCoNode Cyc2	37.08	36.87	47.13	36.87	37.08	40.66	38.51	37.65	37.65	38.70	37.92	42.61	61.09	**35.78**	39.08	66.72	36.41	45.15	38.20	36.22	43.18	37.15	36.49	38.24
RoCoNode v2	31.89	31.39	38.36	30.55	31.59	33.43	31.00	30.57	30.57	31.14	30.78	34.44	59.29	30.91	31.77	66.30	31.27	38.87	33.44	30.93	35.33	**30.02**	31.14	32.21
RoCoNode S3	47.02	46.50	57.99	45.43	46.77	47.02	47.10	45.78	45.78	47.32	46.13	52.01	69.43	**44.67**	47.82	75.19	45.16	53.45	47.47	46.23	54.34	45.48	46.5	48.41
Xbar Cyc2	47.22	47.18	62.96	47.22	47.55	52.33	50.71	49.63	64.87	50.96	50.04	56.23	73.38	**45.43**	51.37	77.87	45.97	61.71	47.55	46.72	59.88	48.05	47.09	51.20
Xbar v2	38.27	37.24	45.79	35.83	37.49	39.92	31.70	**31.29**	40.39	31.84	31.53	35.22	58.91	35.03	32.41	66.44	35.42	39.62	39.19	36.59	38.92	96.30	36.83	36.57
Xbar S3	57.47	56.15	67.97	54.47	56.48	60.25	51.76	50.00	63.28	51.99	**11.94**	57.00	67.99	51.91	52.11	71.34	52.36	56.75	57.87	55.56	59.53	55.26	55.91	56.10
FIR Cyc2	54.80	54.61	68.07	54.82	54.87	60.65	58.78	57.58	71.96	59.05	**12.80**	14.21	83.77	53.52	59.30	87.56	53.98	70.76	55.22	54.28	65.34	55.69	54.61	60.14
FIR v2	40.04	39.00	47.04	37.57	39.25	41.42	32.25	31.84	41.11	32.41	**30.12**	33.56	61.65	36.67	32.86	67.57	37.00	38.88	40.66	38.24	38.98	37.04	38.57	35.69
FIR S3	61.74	60.47	72.58	58.59	60.79	64.57	56.10	54.24	68.34	56.35	54.62	61.79	74.19	56.82	56.58	78.19	57.30	59.78	62.33	60.35	63.97	60.22	60.69	60.30
FFT Cyc2	37.13	37.46	52.48	37.13	37.92	41.16	40.30	39.77	54.40	40.50	40.30	44.74	64.20	**36.40**	41.03	69.69	37.13	48.58	38.39	37.00	48.97	38.78	37.52	40.11
FFT v2	38.02	37.10	45.24	35.65	37.34	39.62	30.18	**29.88**	38.69	30.31	30.12	33.56	59.05	34.66	30.67	65.54	35.01	37.55	38.78	36.34	36.91	35.13	36.59	34.54
FFT S3	43.13	42.44	56.11	40.46	43.05	44.27	37.02	**36.11**	49.24	37.18	36.72	40.84	58.32	38.02	37.40	62.60	38.70	41.53	44.12	42.14	47.25	40.31	42.67	39.69
RC-5 Cyc2	37.02	37.00	47.38	37.02	37.17	40.80	38.74	48.69	48.69	38.93	38.36	43.03	67.63	36.94	39.63	74.08	37.40	47.95	38.07	**36.58**	44.27	37.71	36.87	39.23
RC-5 v2	42.98	42.23	50.55	40.98	42.46	45.05	39.85	49.19	49.19	40.02	39.55	44.27	69.90	40.98	40.54	75.93	41.55	49.18	43.36	41.84	46.53	40.02	42.08	43.13
RC-5 S3	63.28	62.23	75.68	60.50	62.58	67.34	60.82	73.97	73.97	61.12	**59.23**	67.00	78.98	59.31	61.41	82.41	59.90	65.91	64.04	62.48	70.10	62.18	62.85	65.53
DES Cyc2	41.36	41.63	57.97	41.43	42.09	46.00	45.27	44.41	60.09	45.53	44.94	50.17	68.30	**40.17**	120.65	45.93	41.10	53.74	42.22	41.10	54.14	42.82	41.63	44.54
DES v2	38.16	37.51	46.18	36.16	37.75	40.43	35.11	**34.52**	44.09	35.26	34.76	38.94	57.10	35.69	43.23	35.68	36.13	42.52	39.39	37.03	42.09	35.75	37.28	38.67
DES S3	54.67	53.95	66.43	52.48	54.24	58.56	54.27	52.56	66.28	54.52	52.88	59.83	72.95	**52.01**	54.81	77.49	52.38	59.38	54.99	53.95	62.01	52.95	54.22	57.25

	E						
Appl >	1	2	2	2	2	2	
Algo >	LZSS	Arith	LZW	RLE	Huff	LZSS	
RoCoLink Cyc2	52.16	50.87	66.24	52.41	51.37	57.72	
RoCoLink v2	29.83	**29.42**	38.18	29.96	29.65	33.13	
RoCoLink S3	54.87	52.71	68.75	55.11	53.22	60.28	
RoCoNode Cyc2	38.51	37.65	37.65	38.70	37.92	42.61	
RoCoNode v2	31.00	30.57	30.57	31.14	30.78	34.44	
RoCoNode S3	47.10	45.78	45.78	47.32	46.13	52.01	
Xbar Cyc2	50.71	49.63	64.87	50.96	50.04	56.23	
Xbar v2	31.70	**31.29**	40.39	31.84	31.53	35.22	
Xbar S3	51.76	50.00	63.28	51.99	**11.94**	57.00	
FIR Cyc2	58.78	57.58	71.96	59.05	**12.80**	14.21	
FIR v2	32.25	31.84	41.11	32.41	**30.12**	33.56	
FIR S3	56.10	54.24	68.34	56.35	54.62	61.79	
FFT Cyc2	40.30	39.77	54.40	40.50	40.30	44.74	
FFT v2	30.18	**29.88**	38.69	30.31	30.12	33.56	
FFT S3	37.02	**36.11**	49.24	37.18	36.72	40.84	
RC-5 Cyc2	38.74	48.69	48.69	38.93	38.36	43.03	
RC-5 v2	39.85	49.19	49.19	40.02	39.55	44.27	
RC-5 S3	60.82	73.97	73.97	61.12	**59.23**	67.00	
DES Cyc2	45.27	44.41	60.09	45.53	44.94	50.17	
DES v2	35.11	**34.52**	44.09	35.26	34.76	38.94	
DES S3	54.27	52.56	66.28	54.52	52.88	59.83	

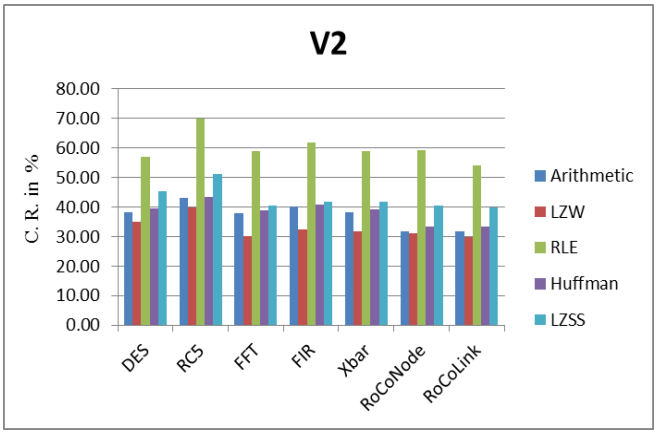

Figure 10: Compression Ratio Obtained in Stage One for Vertex-II

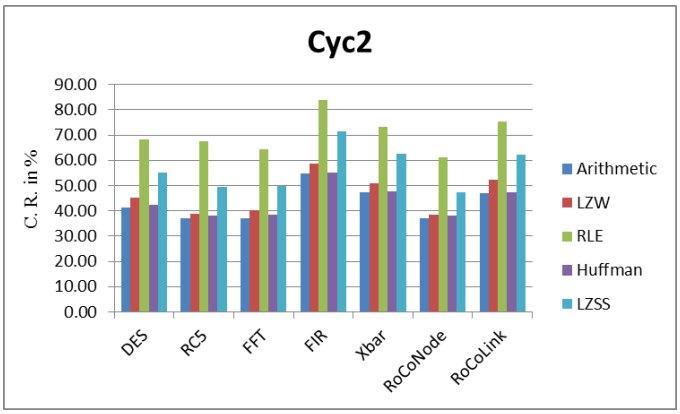

Figure 11: Compression Ratio Obtained in Stage One for Cyclone-II

In stage I for Spartan-3 better compression ratio achieved with LZW algorithm for most of the benchmark designs. Also in case of Vertex –II, LZW algorithm shows better compression ratio compared with others. But with Cyclone-II, though compression ratio is not better with LZW algorithm the difference of compression ratio is not more as compared with arithmetic algorithm.

In stage II compression algorithms are applied to compressed bitstream obtained in stage I. The overall compression ratio in each case is tabulated in Table 3. Minimum compression ratio in each benchmark design is highlighted in each row of Table 3. Columns A, B, C, D, and E gives results of five cascading combinations applied on different types of FPGAs and benchmark designs. In column A, "algo" row indicates first application of Arithmetic compression algorithm in stage I and then second time application of Arithmetic, LZW, RLE, Huffman, LZSS and LZW in sage II. All columns in second row marked as 1 indicates application of compression algorithm in stage I and columns with 2 indicates second time applied compression algorithm in stage II.

Results of average percentage improvement in compression ratio obtained after stage II is shown in Table 4. Whenever RLE is applied in stage I and Arithmetic, LZW, Huffman and LZSS algorithms in stage II the highest average percentage improvement in compression ratio is achieved. An average percentage improvement in these cases are observed as 24.61 %, 18.82 %, 24.09 %, and 24.61 % respectively as mentioned in Table 4.

Table 4: Average Percentage Improvement in Compression Ratio obtained after Stage II

Cascading Combinations of compression algorithm		Average percentage improvement in C.R.
Stage I	Stage II	
Arithmetic	Arithmetic	0.55
Arithmetic	LZW	-11.09
Arithmetic	RLE	1.53
Arithmetic	Huffman	0.22
Arithmetic	LZSS	-2.71
LZW	Arithmetic	-0.68
LZW	LZW	-9.78
LZW	RLE	-0.2
LZW	Huffman	4.73
LZW	LZSS	-2.11
RLE	Arithmetic	24.61
RLE	LZW	18.82
RLE	RLE	-2.61
RLE	Huffman	24.09
RLE	LZSS	24.61
Huffman	Arithmetic	1.74
Huffman	LZW	-5.35
Huffman	RLE	-0.95
Huffman	Huffman	1.4
Huffman	LZSS	-0.02
LZSS	Arithmetic	1.52
LZSS	LZW	-8.34
LZSS	RLE	-0.15
LZSS	Huffman	2.95
LZSS	LZSS	-5.63

CONCLUSIONS

Our approach of cascading of compression algorithm improves the compression ratio by an average of ≈ 23 % when RLE cascaded with Arithmetic, LZW, Huffman and LZSS algorithm. Further maximum times minimum compression is obtained for a combination of RLE (stage I) & Arithmetic algorithm (stage II). In some cases whenever minimum compression is found with other cascading combinations, the compression ratio with combination of RLE and Arithmetic algorithm is nearer to the minimum value. In first stage, comparatively minimum compression ratio is obtained with LZW algorithm. Improved compression ratio is obtained when LZW in stage I cascaded with Arithmetic and Huffman algorithms in stage II. In most of the cases with Arithmetic compression algorithm in stage II, minimum compression ratio is achieved irrespective of compression algorithm in stage I.

REFERENCES

[1] A. Wolfe and A. Chanin, "Executing compressed programs on an embedded RISC architecture," *MICRO* 1992.

[2] S. Hauck, Z. Li, and E. Schwabe, " Configuration compression for the Xilinx XC6200 FPGA," *IEEE Trans. Comput-Aided Des. Integr. Circuits Syst.*, Aug. 1999, vol. 18, no. 8: pp. 1107-1113.

[3] J.H. Pan, T. Mitra, and W.F. Wong, "Configuration bitstream compression for dynamically reconfigurable FPGAs," in *Proc. Int. Conf. Computer-Aided Des*., 2004, pp. 766-733.

[4] D.A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, 1952, col. 40, no. 9: pp. 1098-1101.

[5] A. Moffat, R. Neal, and I. H. Witten, "Arithmetic coding revisited," *in Proc. Data Compression Conf*., 1995, pp. 202-211.

[6] Jacob Ziv and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Tnformation Theory*., May 1977, vol. 23, no. 3.

[7] Xiaoke Qin, Chetan Muthry, and Prabhat Mishra, "Decoding-Aware Compression of FPGA Bitstreams," *in IEEE Trans. On Very Large Scale Integratation (VLSI) Syst.*, March 2011, vol.19, no. 3

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

[9] M. Huebner, M. Ullmann, F. Weissel, and J.Becker, "Real-time configuration code decompression for dynamic FPGA self-reconfiguration," *in Proc. Int. Parallel Distrib. Process.Symp*., 2004, pp 138-143.

[10] R. Stefan and S. Cotofana, "Bitstream compression techniques for Virtex 4 FPGAs," *in Proc. Int. Conf. Field Program Logic Appl*, 2008, pp. 323-328.

[11] A. Dandalis and V. K. Prasanna, " Configuration compression for FPGA-based embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*., Dec. 2005, vol. 13, no. 12, pp. 1394-1398.

[12] D. Koch, C. Beckhoff, and J. Teich, "Bitstream decompression for high speed FPGA configuration and slow memories," *in Proc. Int. Conf. Field-Program. Technol.*, 2007, pp. pp. 161-168.

Dept. of Computer Science 12, Bitstream Compression Benchmark, http://www.reconets.de /bitstreamcompression/.