Data Compression Device Based on Modified LZ4 Algorithm

Weiqiang Liu, *Senior Member, IEEE*, Faqiang Mei, Chenghua Wang, Maire O'Neill, *Senior Member, IEEE*, and Earl E. Swartzlander, Jr., *Life Fellow, IEEE*

***Abstract*—Data compression is commonly used in NAND flash-based solid state drives (SSDs) to increase their storage performance and lifetime as it can reduce the amount of data written to and read from NAND flash memory. Software-based data compression reduces SSD performance significantly and, as such, hardware-based data compression designs are required. This paper studies the latest lossless data compression algorithm, i.e., the Lempel-Ziv (LZ)4 algorithm which is one of the fastest compression algorithms reported to date. A data compression FPGA prototype based on the LZ4 lossless compression algorithm is studied. The original LZ4 compression algorithm is modified for real-time hardware implementation. Two hardware architectures of the modified LZ4 algorithm (MLZ4) are proposed with both compressors and decompressors, which are implemented on an FPGA evaluation kit. The implementation results show that the proposed compressor architecture can achieve a high throughput of up to 1.92 Gb/s with a compression ratio of up to 2.05, which is higher than all previous LZ algorithm designs implemented on FPGAs. The compression device can be used in high-end SSDs to further increase their storage performance and lifetime.**

***Index Terms*—FPGA, lossless compression, Lempel-Ziv (LZ) algorithms, LZ4, solid-state drives (SSDs).**

I. Introduction

SOLID-STATE drives (SSDs) based on NAND flash memory have become popular in consumer electronic devices such as smart phones, tablet, and desktop systems [1], [2]. It is highly desirable to reduce the amount of data in SSDs and the read/write data transmission time to/from SSDs as flash memory has a finite number of program-erase (P/E) cycles thus limited lifetime [3]. For example, older single-level cell (SLC) NAND-flash memory was able to withstand 150 000 P/E cycles, while multilevel cell (MLC) NAND-flash memory using 15–19 nm process technologies wears out after only 3000 P/E cycles [2], [4]. Furthermore, the performance of MLC flash memory is also much slower than that of its SLC counterpart. Also, more advanced triple-level cell NAND flash memory has an even lower number of P/E cycles [5]. This problem is expected to worsen with further scaling of the semiconductor process. Therefore, to increase the lifetime and also the performance of flash-based SSDs, the amount of data written to and read from the SSDs should be reduced, which can be achieved using data compression. Another benefit of using lossless data compression in SSDs is to reduce the I/O latency.

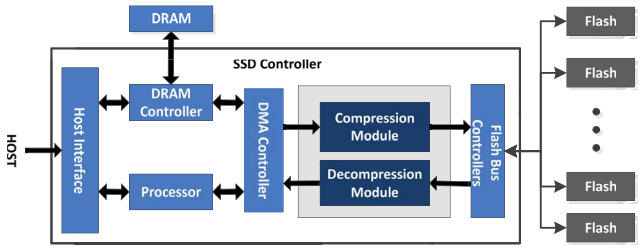

Fig. 1. Typical SSD architecture with data compression acceleration.

Data compression for SSDs has been widely adopted. Data compression can be implemented in three layers: 1) the application; 2) the file system; or 3) the firmware of the storage device. Most data compression algorithms are adopted in the application layer and the file system using software implementation. Software-based data compression can be useful in improving the lifetime of SSDs. However, the overall performance of SSDs is reduced significantly due to the slow compression and decompression speed. A recent study [6] based on realistic data and systems show that applying data compression in the firmware of the SSDs using a data compression hardware accelerator is the best approach. A typical SSD architecture with data compression acceleration is shown in Fig. 1.

Although hardware-based compression is required for NAND flash memory and SSDs, little research has been conducted on how to design a high performance hardware compression accelerator [7]–[13]. In [6], it was found that for high-end SSDs with transaction rates of up to 3K per second, compression/decompression rates of above 200 Mb/s (i.e., 1.6 Gb/s) are required. However, existing designs are limited in performance with compression speeds in the range of 0.567–1.6 Gb/s [7]–[13], which cannot meet the requirement of high-end SSDs.

In this paper, the design of a hardware accelerator based on the latest lossless data compression algorithm, i.e., Lempel-Ziv

Manuscript received December 4, 2017; revised February 10, 2018; accepted February 15, 2018. Date of publication March 2, 2018; date of current version March 29, 2018. This work was supported by the Fundamental Research Funds for the Central Universities China under Grant NS2017024. *(Corresponding author: Weiqiang Liu.)*

W. Liu, F. Mei, and C. Wang are with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China (e-mail: liuweiqiang@nuaa.edu.cn; meifaqiang@nuaa.edu.cn; chwang@nuaa.edu.cn).

M. O'Neill is with the Center for Secure Information Technologies, Queen's University Belfast, Belfast BT3 9DT, U.K. (e-mail: m.oneill@ecit.qub.ac.uk).

E. E. Swartzlander is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: eswartzla@aol.com).

Digital Object Identifier 10.1109/TCE.2018.2810480

(LZ)4 [14] for data compression in high-end SSDs is studied and demonstrated on an FPGA device. The original LZ4 algorithm is somewhat difficult to implement in hardware as it was proposed for software implementation. It is not possible to store all the text in calculating the hash. Its output delay is uncertain and the input data is limited by the address width of the hash table. As a result, the LZ4 algorithm has been modified for hardware implementation in this paper to solve these problems. By using the modified LZ4 algorithm (MLZ4), the hash computation is improved for the compression ratio and low output latency is achieved. The implementation results on an FPGA platform show the proposed MLZ4 architecture and provides the highest throughput performance compared with previous FPGA implementations of LZ algorithms, which makes it suitable for high-end SSDs.

This paper is organized as follows. Section II reviews lossless data compression algorithms and their hardware implementations. The original LZ4 algorithm is also reviewed in this section. Section III presents the modified LZ4 algorithm. Two hardware architectures of both the MLZ4 compressors and decompressors are proposed in Section IV. A comparison with other FPGA hardware designs of LZ algorithms is provided in Section V. Section VI concludes this paper.

II. Review

A. Data Compression Algorithms and Implementations

There are two main categories of data compression, namely, lossy and lossless compression [15]. As lossy compression allows loss of accuracy to an acceptable level, it is usually used for multimedia applications where errors can be tolerated [16]. Lossless compression can compress and then recover the data from compressed data without loss of information; and it is used for applications where even one single bit difference between the original and reconstructed data cannot be tolerated.

The applications of lossless data compression have been increasing significantly due to both the demand for increased bandwidth [17], [18] and the need to improve storage capacity [3]. Lossless data compression has been successfully deployed in storage systems including tapes, hard disk drives, SSDs, file servers, and storage area networks.

Lossless data compression can be achieved using two different approaches: 1) statistical model-based compression such as Huffman coding [19] and 2) dictionary-based compression including the LZ algorithms [20], [21]. The LZ algorithms belong to adaptive dictionary-based techniques, which are the most popular lossless compression algorithms when prior statistical characteristics of the data are unknown. The LZ algorithms have been adopted by many compression format standards such as Zip, GNU zip, and Zlib [22].

LZ algorithms were proposed by Ziv and Lempel in 1977 [20] and 1978 [21] in their two landmark papers. These papers presented two different approaches. The approach based on the 1977 paper is referred to as the LZ77 (or LZ1) family which includes LZ77, LZRW [23], LZSS [24], LZ-Markov chain algorithm (LZMA) [11], etc. LZ77 algorithms use a sliding window to examine the input sequence.

Token	Literal Length	Literals	Offset	Match Length
1 Byte	0-n Bytes	0-L Bytes	2 Bytes	0-n Bytes

Fig. 2. Data format of an LZ4 sequence.

Its principle is to find whether the sequence being compressed appears in the previously input data. If so, a pointer is used to point to the repeated strings. The dictionary refers to a portion of the previously encoded sequence. The approaches based on the 1978 paper are known as the LZ78 (or LZ2) family which includes LZ78, LZW [25], etc. LZ78 algorithms create a dictionary of phrases from the input data. When a match with the phrases that have appeared in the dictionary occurs, the encoder will output the phrase's index in the dictionary rather than the phrase itself.

Collet [14] proposed the LZ4 algorithm in 2011 [13], which is a variant of LZ77. The compression speed of a LZ4 software implementation is shown to be fastest among the LZ algorithms. However, there is little research conducted on the hardware implementation of LZ4 as it is much younger than other LZ algorithms. Hardware designs of lossless data compression algorithms are receiving increase attention due to the exponential expansion in network communication and data storage. FPGA implementations of LZ algorithms such as LZRW3 [12], LZW [8], [10], the LZMA [11], and LZ4 [13] have been proposed to meet real-time requirements. Thus, it is necessary to study hardware architectures of LZ4 in order to explore its performance for consumer electronic applications such as SSDs.

B. Review of the LZ4 Algorithm

This section reviews the LZ4 algorithm. Its data format and data flow are introduced. The shortcomings of the original LZ4 algorithm and data format are also discussed.

LZ4 was initially defined as a form of compressed data format. Compressed data files are composed of LZ4 sequences that include a token, literal length, offset, and match length as shown in Fig. 2. The token is used to indicate the length of unmatched and matched characters. The literal length indicates the length of uncompressed data and its value is equal to the value of the length of uncompressed data minus 15. The uncompressed data is stored as literals in the LZ4 sequence and it is copied from the original data. When the input data finds data that appeared before via searching, this data will be compressed. The value of the offset indicates the address of the current data minus the address of the prior data. Match length means the length of the matching data.

The operation of the LZ4 algorithm is mainly divided into the following five steps [14]: 1) hash computation; 2) matching; 3) backward matching; 4) parameter calculation; and 5) data output, which is shown in Fig. 3.

III. Modified LZ4 Algorithm

An improved data format is proposed in this section along with an improved algorithm to solve the defects in the original LZ4 algorithm.

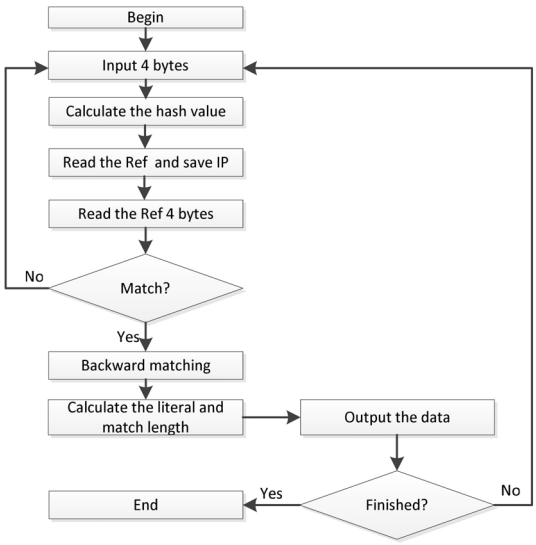

Fig. 3. Flow chart of original LZ4 algorithm.

Token	Literals	Literal Length	Literals	Literal Length	…	Offset	Match Length
1 Byte	0-15 Bytes	0-1 Byte	0-128 Bytes	0-1 Byte	…	2 Bytes	0-2 Bytes

Fig. 4. Data format of MLZ4.

The original LZ4 algorithm was proposed for software implementation in general processors. As such, there are some issues with the LZ4 algorithm for hardware implementation.

1) The hash calculation is only performed for unmatched characters. Hash calculation is not applied to the backward matching. Thus, part of the matching data's hash value will not be calculated.
2) For step 2 of the original LZ4 algorithm, when there is hash conflict (different data have the same hash value), more clock cycles are needed to recalculate the hash value, which reduces the compression speed.
3) The input data is limited by the address width in the hash table. The maximum number of memory addresses in the hash table is the maximum size of the input data. It cannot compress data constantly.
4) Output delay is uncertain. According to the original LZ4 data format, the length of matched characters and unmatched characters should be obtained before outputting the data. For example, if the unmatched character length is 40 kb, data can only be outputted after all 40 kb are searched.

In order to increase the compression speed in hardware, the LZ4 data format is changed as shown in Fig. 4. Note that the format of the token and offset is consistent with the original format. The main differences are as follows.

A. Literal Length

If the value of the first four bits of the token is less than 15, there is no literal length. If the value of the first four bits of token is 15, the length of unmatched literals is the sum of all literal lengths.

B. Match Length

If the value of the last four bits of the token is less than 15, there is no match length. The value of the actual match length is the value of the last four bits of token plus 4. If the value of the last four bits of the token is 15, the value of the match length is represented using 2 bytes after the offset. The actual value of the match length is the sum of them.

The MLZ4 algorithm (addressing the issues mentioned above) is detailed as follows.

1) To improve the compression ratio, the hash value of the data can be calculated during the backward matching in the modified algorithm to exploit the parallelism of hardware implementation.
2) To reduce the delay when a hash conflict occurs and to improve the compression speed, a hash dictionary that corresponds to the hash table is added. The difference between the hash table and the hash dictionary is that the hash table stores the address, while the hash dictionary stores the corresponding data based on the hash value. During match searching, the data stored in the hash dictionary can be read and compared when reading the address at the same time. As a result, the number of clock cycles can be reduced.
3) To allow continuous compression in hardware, a Valid Bit is added in the hash table. When the data is valid, the Valid Bit is set to "1." When the data is invalid in a hash table, the Valid Bit is reset to "0" and a data cleaning process is also added. In this way, when the data address reaches the maximum address and continues to search for backward matching, no matching error occurs, as there will be no overlapped address. Thus, continuous compression can be achieved.
4) To make sure the output delay is predictable, the LZ4 data format is changed as shown in Fig. 4. When the unmatched character length is longer than 300 bytes, the backward match is ignored. For the above mentioned example, if the length of the unmatched characters is 40 kb, the data can be output when the match length reaches 300 bytes according to the new data format. Thus, the waiting time for matching is reduced significantly.

A flow chart illustrating the operation of the MLZ4 algorithm is shown in Fig. 5.

IV. FPGA Architecture and Implementation of the MLZ4 Algorithm

Compared with software, a hardware implementation offers parallel processing that can allow multiple compressors to work at the same time to increase the throughput of compression. The FPGA implementation of the MLZ4 algorithm is presented in this section with two FPGA hardware architectures, i.e., MLZ4-1 and MLZ4-2, with both compressors (i.e., MLZ4C-1 and MLZ4C-2) and decompressors (i.e., MLZ4D-1 and MLZ4D-2).

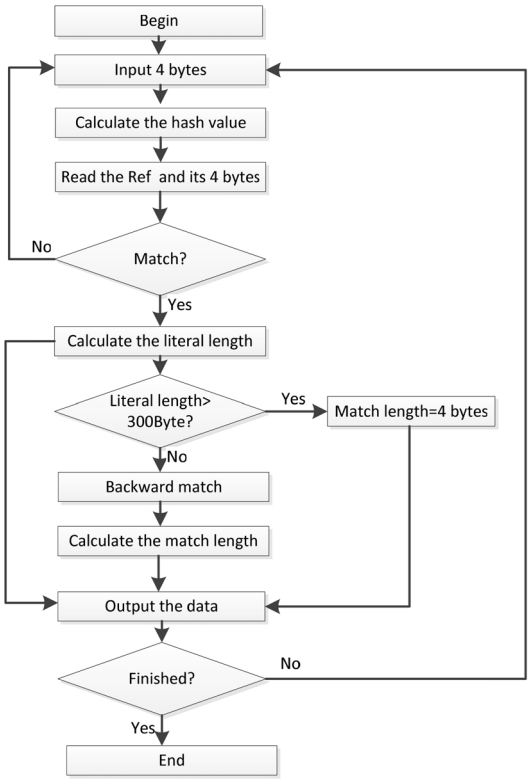

Fig. 5. Flow chart of MLZ4 algorithm.

A. 1st FPGA Architecture of MLZ4 Compressor

The 1st hardware architecture of the LZ4-1 compressor is shown in Fig. 6. It mainly consists of the input RAM, the output RAM, word shift register, reading back control module (i.e., ref. control), search module (including hash engine, word table, hash table, hash clear, match, and backward match), literal length calculation module (i.e., literal length), match length calculation module (i.e., match length), and output control module (including ports A, B, and C control).

The input and output RAM modules store the data before and after compression both in a 64k RAM. Data are read from the input RAM and then turned into 32-bit data through the word shift register. The 32-bit data is fed to the hash engine to compute the hash value. The ref. control module controls the reading pointer to read the data, and then uses them to find the backward matching data.

The search module performs the hash value calculation, reads the hash table. It also changes the ref. address, finds any matching conflicts, judges the match length, calculates the offset, and judges whether the input data address (denoted as IP) has reached the end.

Literal and match length calculation modules are used to calculate the length of the unmatchable characters and matching data. The ports A and B control modules write the compressed data to the output RAM according to the MLZ4 data format. The port C control module is used to control the compressed data from the output RAM to the PCI-E interface.

The designs in this paper are all implemented on an FPGA evaluation kit. The MLZ4C-1 runs at a frequency of 100 MHz and its throughput is 0.8 Gb/s. The hardware resources used in implementing the compressor are summarized in Table I. Compression results from testing the proposed designs with benchmark files from the Calgary corpus (paper1 and paper2) [26], the Canterbury corpus (asyoulik and cp) [26], and the Silesia corpus (dickens) [27] are shown in Table II. It can be seen that the compression ratio achieved is between 1.65 and 2.05.

TABLE I
RESOURCES USED FOR BOTH 1ST COMPRESSOR (MLZ4C-1) AND 1ST DECOMPRESSOR (MLZ4D-1)

Resource	MLZ4C-1	MLZ4D-1	Total
Slices	571	365	936
FFs	605	604	1,209
LUTs	1,302	767	2,069
BRAMs	76.5	32.5	109

TABLE II
TEST RESULTS OF COMPRESSION RATIO USING MLZ4 ALGORITHM

Test Files	Original Size (Bytes)	Compressed Size (Bytes)	Compression Ratio
paper1	53,161	28,686	1.85
paper2	82,199	46,750	1.76
asyoulik	125,179	75,467	1.66
cp	24,603	11,993	2.05
dickens	10,192,446	6,161,433	1.65

B. 1st FPGA Architecture of the MLZ4 Decompressor

The decompressor is simpler than the compressor. The information contained in token and literal length show the length of unmatched characters. The unmatched character can be output directly and the positions of offset and match length can be calculated according to the length of unmatched characters. Matched strings can be copied from decompressed data based on the values of offset and match length. All data can be decompressed after repeating the above operations. The decompressor of the MLZ4D-1 is also designed and implemented on an FPGA chip. Its hardware architecture is shown in Fig. 7. The decompressor mainly includes three modules: 1) the input control module; 2) the IP control module; and 3) the output control module. The Input RAM is used to store the data. The IP control module changes IP according to literal length, offset, and match length. The output control module is used to control the storage of decompressed data and output the final data.

The MLZ4D-1 decompressor can run much faster than the MLZ4C-1 compressor as LZ4 is an asymmetric compression algorithm. In this paper, the MLZ4D-1 operates at 120 MHz and its throughput is 0.96 Gb/s. The hardware resources used by the MLZ4D-1 design are also summarized in Table I.

C. 2nd FPGA Architecture of the MLZ4 Compressor

The 2nd hardware architecture of the LZ4 compressor is shown in Fig. 8. The main difference between MLZ4C-1 and MLZ4C-2 are the search module and the output module. Additionally, MLZ4C-2 includes a new IP Shift block. The detailed differences are as follows.

1) Search Module: The match block in MLZ4C-2 is divided into word compare, match compare, and IP compare blocks. Furthermore, the backward match block is divided into

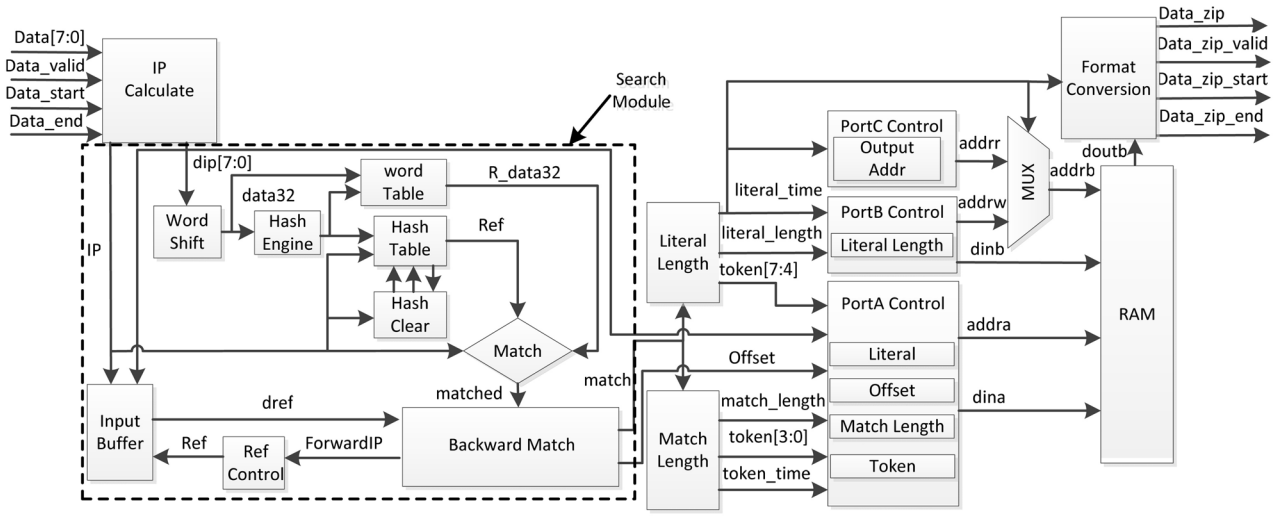

Fig. 6. Hardware architecture of the MLZ4C-1 compressor.

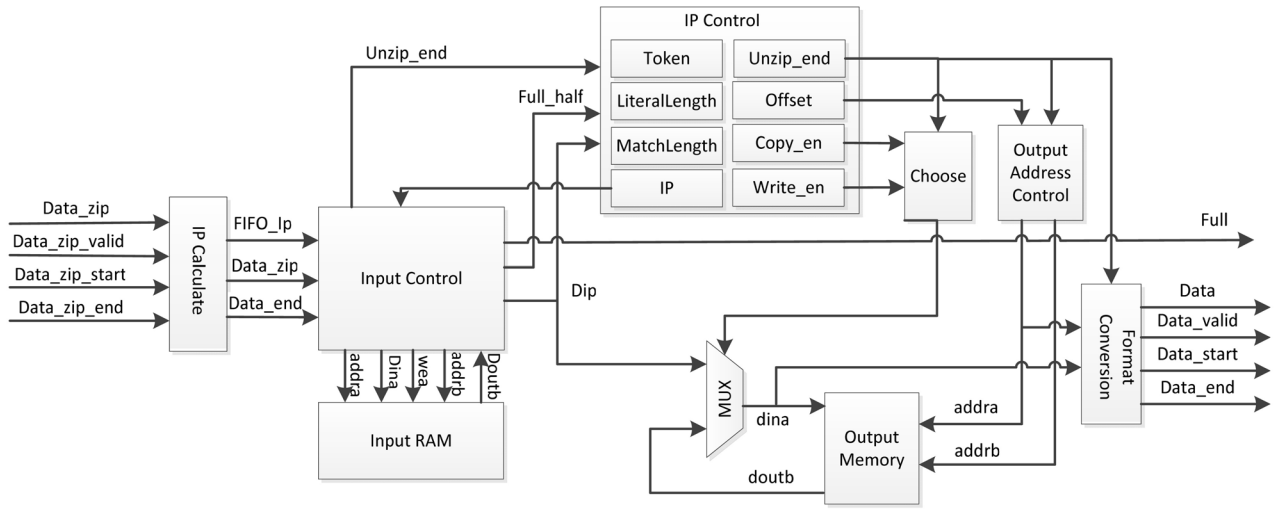

Fig. 7. Hardware architecture of the MLZ4D-1 decompressor.

backward compare and match flag blocks. Therefore, the critical path of both the match and backward match logic is further reduced by inserting pipeline registers.

2) Output Module: The even and odd output bits are written into RAM-A and RAM-B, respectively, by using two RAM blocks. Both RAM blocks can output the data at the same time. The even bits are in the eight most significant bits of the output data, i.e., dzip[15:8], and the odd bits are in the eight least significant bits, i.e., dzip[7:0], which is in the revised format as shown in Fig. 4.

The MLZ4C-2 design runs at a frequency of 240 MHz and its throughput is 1.92 Gb/s. The hardware resources used in implementing the compressor are summarized in Table III. Test results show that the compression ratios achieved by the MLZ4C-2 design are the same as that listed in Table II.

TABLE III
RESOURCES USED FOR 2ND COMPRESSOR (MLZ4C-2) AND 2ND DECOMPRESSOR (MLZ4D-2)

Resource	MLZ4C-2	MLZ4D-2	Total
Slices	345	155	500
FFs	937	377	1314
LUTs	573	342	915
BRAMs	69	20	89
DSPs	4	0	4

D. 2nd FPGA Architecture of the MLZ4 Decompressor

The MLZ4D-2 is similar to MLZ4D-1, as shown in Fig. 9. However, the input module that includes IP calculation, input control, and input RAM blocks is now replaced with an FIFO. The IP control block in MLZ4D-1 is changed to a read control block in MLZ4D-2, where the combinational logic has been further divided and pipeline registers have been added to increase the performance.

The MLZ4D-2 operates at 260 MHz and its throughput is up to 2080 Mb/s. The hardware resources used by the decompressor are also summarized in Table III.

E. Comparison Between MLZ4-1 and MLZ4-2

Due to the optimized pipelined architecture, the 2nd design is much faster than the 1st design. The throughput of

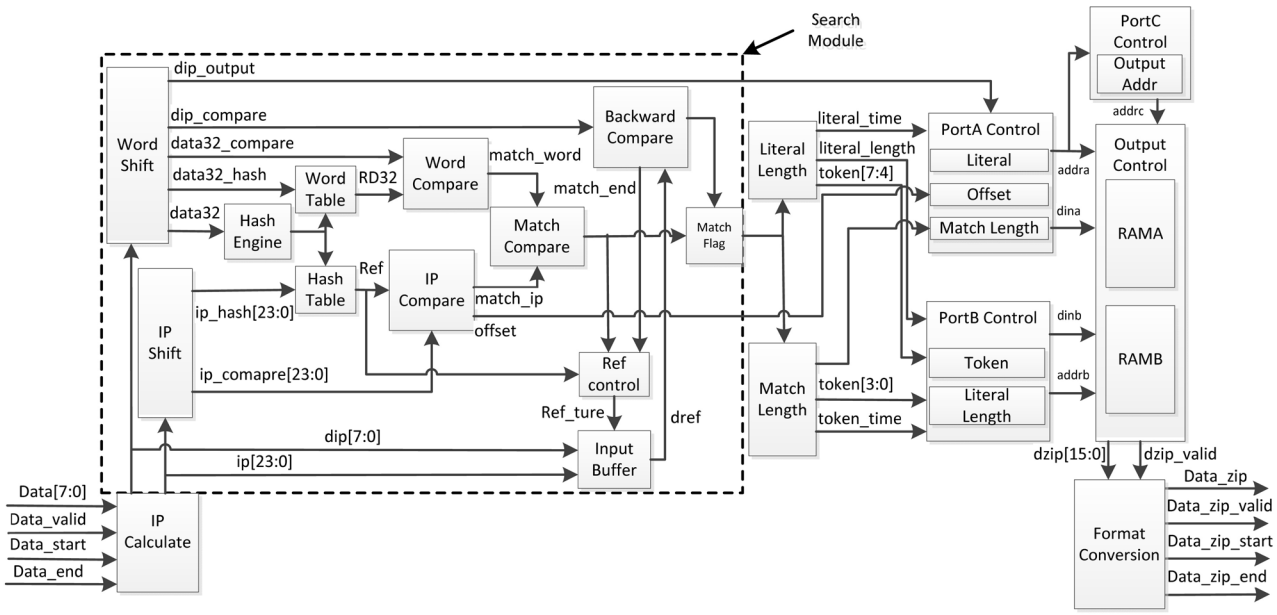

Fig. 8. Hardware architecture of the MLZ4C-2 compressor.

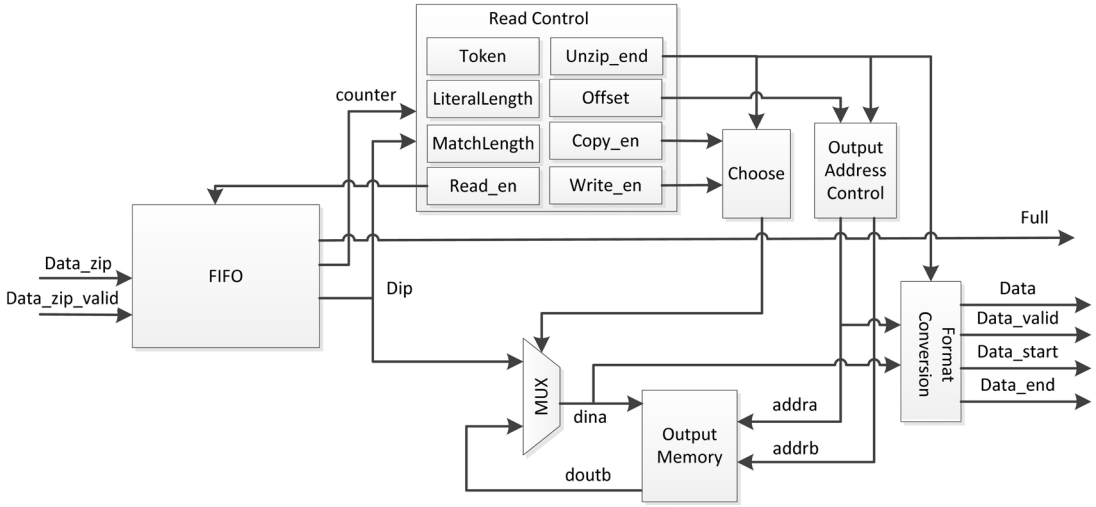

Fig. 9. Hardware architecture of the MLZ4D-2 decompressor.

MLZ4C-2 and MLZ4D-2 are 2.4 and 2.16 times that of MLZ4C-1 and MLZ4D-1, respectively. At the same time, the hardware required is also reduced significantly. MLZ4C-2 only uses 60% of the slices and 90% of the BRAMs used in MLZ4C-1. However, MLZ4C-2 uses four additional DSPs. MLZ4D-2 uses much fewer slices compared with MLZ4D-1, where more than half of slices are saved. The number of BRAMs used is also reduced by over 38%. The comparison results show that the second architecture is a much better design. Both MLZ4-1 and MLZ4-2 are further compared with previous work in the following section.

V. Comparison With Other FPGA Implementations of LZ Algorithms

In this section, the proposed designs are compared with other LZ algorithm FPGA implementations. The designs compared include X-MatchPROv4 [7] using the XMatchPRO algorithm, the conventional LZW [8], the ELDC-3 core [9] that implements four image compression algorithms, an improved LZW VLSI processor [10] which implements the new LZW algorithm, LZMA [11], the LZWR3 core [12] that implements the LZRW3 algorithm and an LZ4 FPGA device [13].

The comparison is summarized in Table IV. The MLZ4C-1 is a baseline design. Its performance is not so attractive compared with the previous best design. However, the revised and pipelined design, i.e., MLZ4C-2, has improved the compression performance significantly. From the table, it is clear that the proposed MLZ4C-2 offers the highest performance. Although it consumes slightly more slices, MLZ4C-2 increases the compression throughput by 20% compared with the best previous design in [13] which is also an LZ4 FPGA design, as shown in Fig. 10. The main difference between MLZ4C-2 and the design in [13] is that a word table added in MLZ4C-2 is

TABLE IV
COMPARISON OF LZ COMPRESSION AND DECOMPRESSION IMPLEMENTATIONS

Compression Device	Algorithms	FPGA Technology	Complexity	Clock Speed (MHz)	Troughput (Gbps)
X-MatchProv4 [7]	X-MatchPRO	180nm	5367 LUTs	50	0.567 (Compression)
LZW [8]	LZW	120nm/150nm	332 Slices 631 LUTs 247 Slices 474 LUTs	50	0.700 (Compression) 1.120~1.282 (Decompression)
ELDC-3 Core [9]	CGF, GZIP, ELIC, PNG	90nm	5900 Slices	75	0.400~0.528 (Compression)
Improved LZW Processor [10]	New LZW	90nm	3218 Slices 272 Kb RAMs	124	1.587 (Compression)
LZMA [11]	LZMA	40nm	NA	125	0.604 (Compression)
LZRW3 Core [12]	LZRW3	28nm	227 Slices 789 FFs 4~36 BRAMs	210	1.300 (Compression)
LZ4 [13]	LZ4	28nm	266 Slices 17 BRAMs 3 DSPs	200	1.600 (Compression)
MLZ4C-1	Modified LZ4	28nm	571 Slices 76.5 BRAMs	100 (Compression)	0.800 (Compression)
MLZ4D-1			365 Slices 32.5 BRAMs	120 (Decompression)	0.960 (Decompression)
MLZ4C-2	Modified LZ4	28nm	345 Slices 69 BRAMs 4 DSPs	**240 (Compression)**	**1.920 (Compression)**
MLZ4D-2			155 Slices 20 BRAMs	**260 (Decompression)**	**2.080 (Decompression)**

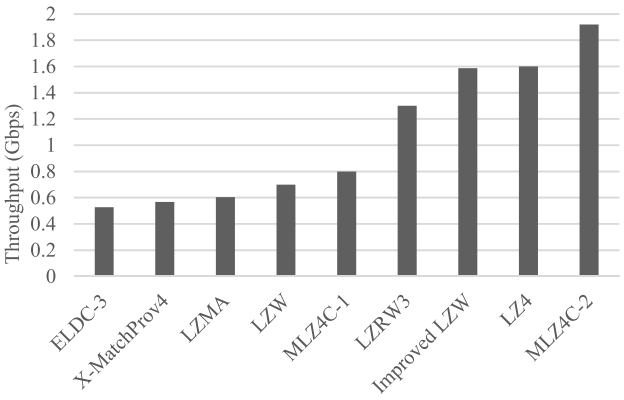

Fig. 10. Performance comparison with state-of-the-art LZ compressors.

used to find both the matched address and the matched data at the same time, which reduces the delay; the output module uses two RAM blocks; all combinational modules are further divided and registers are inserted to pipeline the design.

MLZ4C-2 also outperforms (over 47% faster) the leading commercial compression device, i.e., LZRW3 [12]. MLZ4D-2 is also the fastest decompressor compared with other state-of-the-art designs. This confirms that the proposed MLZ4-2 design is the fastest compression device, and hence, is suitable for high-end SSDs.

VI. CONCLUSION

This paper presents a modified LZ4 algorithm and its FPGA implementations. Several aspects of the original LZ4 algorithm are modified for efficient hardware implementation. These changes improve both the compression and decompression speeds. The implementation on an FPGA chip shows that the proposed designs can achieve compression and decompression throughputs of up to 1.92 Gb/s and 2.08 Gb/s, which is 20% and 47% faster than the previous best compressor and decompressor designs, respectively. The proposed MLZ4 and its hardware architectures can therefore be used to increase the storage performance and lifetime of high-end SSDs.

REFERENCES

[1] J. Luo, L. Fan, Z. Chen, and Z. Li, "A solid state drive architecture with memory card modules," *IEEE Trans. Consum. Electron.*, vol. 62, no. 1, pp. 17–22, Feb. 2016.
[2] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.
[3] Y. Park and J.-S. Kim, "zFTL: Power-efficient data compression support for NAND flash-based consumer electronics devices," *IEEE Trans. Consum. Electron.*, vol. 57, no. 3, pp. 1148–1156, Aug. 2011.
[4] J. H. Yoon and G. A. Tressler, "Advanced flash technology status, scaling trends & implications to enterprise SSD technology enablement," in *Proc. Flash Memory Summit*, Santa Clara, CA, USA, 2012, pp. 1–16.
[5] S. Lee, J. Park, K. Fleming, Arvind, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *IEEE Trans. Consum. Electron.*, vol. 57, no. 4, pp. 1732–1739, Nov. 2011.
[6] A. Zuck, S. Toledo, D. Sotnikow, and D. Harnik, "Compression and SSDs: Where and how?" in *Proc. 2nd Workshop Interactions NVM/Flash Oper. Syst. Workload (INFLOW)*, 2014, pp. 1–10.
[7] *Enhanced Lossless Data Compression (ELDC-3) IP-Core*, GEMAC mbH, Chemnitz, Germany, 2007.
[8] S. Naqvi, R. Naqvi, R. Riaz, and F. Siddiqui, "Optimized RTL design and implementation of LZW algorithm for high bandwidth applications," *Elect. Rev.*, vol. 87, no. 4, pp. 279–285, 2011.
[9] W. Cui, "New LZW data compression algorithm and its FPGA implementation," in *Proc. Picture Coding Symp.*, 2007, pp. 1145–1148.
[10] J. L. Nunez and S. Jones, "Gbit/s lossless data compression hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 3, pp. 499–510, Jun. 2003.

[11] B. Li, L. Zhang, Z. Shang, and Q. Dong, "Implementation of LZMA compression algorithm on FPGA," *Electron. Lett.*, vol. 50, no. 21, pp. 1522–1524, Oct. 2014.
[12] *LZRW3 Data Compression Core for Xilinx FPGA*, Helion Technol., Cambridge, U.K., Oct. 2008.
[13] M. Bartik, S. Ubik, and P. Kubalik, "LZ4 compression algorithm on FPGA," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.,* Cairo, Egypt, 2015, pp. 179–182.
[14] Y. Collet. (2011). *Real Time Data Compression: LZ4 Explained*. [Online]. Available: http://fastcompression.blogspot.ru/2011/05/lz4-explained.html
[15] M. Nelson and J.-L. Gailly, *The Data Compression Book*, 2nd ed. New York, NY, USA: M&T Books, 1995.
[16] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Trans. Consum. Electron.*, vol. 38, no. 1, pp. 18–34, Feb. 1992.
[17] Y. M. Siu, C. K. Chan, and K. L. Ho, "Teletext data change detection and noiseless data compression," *IEEE Trans. Consum. Electron.*, vol. 41, no. 4, pp. 1061–1068, Nov. 1995.
[18] R. Mehboob, S. A. Khan, Z. Ahmed, H. Jamal, and M. Shahbaz, "Multigig lossless data compression device," *IEEE Trans. Consum. Electron.*, vol. 56, no. 3, pp. 1927–1932, Aug. 2010.
[19] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[20] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.
[21] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, Sep. 1978.
[22] D. Harnik, E. Khaitzin, D. Sotnikov, and S. Tharlev, "A fast implementation of deflate," in *Proc. Data Compression Conf.*, 2014, pp. 223–232.
[23] R. N. Williams, "An extremely fast Ziv–Lempel data compression algorithm," in *Proc. Data Compression Conf.*, 1991, pp. 362–371.
[24] J. A. Storer and T. G. Syzmanski, "Data compression via textual substitution," *J. ACM*, vol. 29, no. 4, pp. 928–951, 1982.
[25] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, Jun. 1984.
[26] D. Salomon, *Data Compression: The Complete Reference*, 4th ed. London, U.K.: Springer, 2007.
[27] S. Deorowicz, "Universal lossless data compression algorithms," Ph.D. dissertation, Faculty Autom. Control, Electron. Comput. Sci., Silesian Univ. Technol., Gliwice, Poland, 2003.

Weiqiang Liu (M'12–SM'15) received the B.Sc. degree in information engineering from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 2006 and the Ph.D. degree in electronic engineering from the Queen's University Belfast (QUB), Belfast, U.K., in 2012, respectively.

In 2013, he joined the College of Electronic and Information Engineering, NUAA, where he is currently an Associate Professor. He was a Research Fellow with the Institute of Electronics, Communications and Information Technology, QUB from 2012 to 2013. He has published one research book by Artech House and over 50 leading journal and conference papers. His current research interests include very large scale integration design for digital signal processing and cryptography and emerging technologies in computing systems.

Dr. Liu's paper was a finalist in the Best Paper Contest of IEEE International Symposium on Circuits and Systems (ISCAS) 2011 and he is the co-author of a Best Paper Candidate of ACM GLSVLSI 2015. He serves as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS (TC), the Leader of The Multimedia Team at TC Editorial Board, and the Guest Editor of two special issues of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING. He has been a Technical Program Committee Member for several international conferences, including IEEE Symposium on Computer Arithmetic, the Annual IEEE International Conference on Application-specific Systems, Architectures and Processors, ISCAS, and International Conference on Neural Information Processing. He is a member of the IEEE CASCOM Technical Committee.

Faqiang Mei received the B.Sc. degree in information engineering from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China, in 2016, where he is currently pursuing the master's degree in circuit and systems with the College of Electronic and Information Engineering.

His current research interest includes FPGA design for lossless data compression algorithms and cryptographic hardware.

Chenghua Wang received the B.Sc. and M.Sc. degrees from Southeast University, Nanjing, China, in 1984 and 1987, respectively.

In 1987, he joined the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, where he became a Full Professor in 2001. He has published six books and over 100 technical papers in journals and conference proceedings. His current research interests include testing of integrated circuits and circuits and systems for communications.

Mr. Wang was a recipient of over ten teaching and research awards at the provincial and ministerial level.

Maire O'Neill (M'03–SM'11) received the M.Eng. degree (with Distinction) and the Ph.D. degree in electrical and electronic engineering from Queen's University Belfast (QUB), Belfast, U.K., in 1999 and 2002, respectively.

She is currently the Research Director of the Centre for Secure Information Technologies, QUB. She has authored two research books and has over 125 international peer-reviewed conference and journal publications.

Dr. O'Neill was a recipient of the Prestigious U.K. Engineering and Physical Sciences Research Council Leadership Fellowship from 2008 to 2015, and numerous awards for her research to date, which include the 2014 U.K. Royal Academy of Engineering Silver Medal, and the Women's Engineering Society Prize at the 2006 IET Young Woman Engineer of the Year Awards. She was a former holder of the U.K. Royal Academy of Engineering Research Fellowship from 2003 to 2008 and she was named British Female Inventor of the Year in 2007. She is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS and the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING and has acted as a Guest Editor for a number of journals, including the *IET Information Security* in 2005 launch issue and a special issue on cryptography in the coming decade of the *ACM Transaction on Embedded Computing* in 2015. She has been a Technical Program Committee Member for many international conferences, including Design Automation Conference, Cryptographic Hardware and Embedded Systems, Design, Automation and Test in Europe, IEEE SoC (System-on-Chip) Conference, ISCAS, IET Irish Signals and Systems Conference and Workshop on RFID Security and Privacy. She is an IEEE Circuits and Systems for Communications Technical Committee Member and was a Treasurer of the Executive Committee of the IEEE United Kingdom and Ireland Section from 2008 to 2009. She is a member of the Royal Irish Academy and a fellow of the Irish Academy of Engineering.

Earl E. Swartzlander, Jr. (SM'79–F'88–LF'11) received the B.S. degree from Purdue University, West Lafayette, IN, USA, in 1967, the M.S. degree from the University of Colorado, Boulder, CO, USA, in 1969, and the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 1972, all in electrical engineering.

He is a Professor of electrical and computer engineering with the University of Texas at Austin, Austin, TX, USA. In this position, he and his students conduct research in computer engineering with emphasis on application-specific processor design, including high-speed computer arithmetic, embedded processor architecture, very large scale integration technology, and nanotechnology. In 2016, he supervised 46 Ph.D. students. He has authored two books, authored or co-authored 86 refereed journal papers, 41 book chapters, and 310 conference papers, and edited 11 books.

Dr. Swartzlander was a recipient of the IEEE Third Millennium Medal, the Distinguished Engineering Alumnus Award from the University of Colorado, the Outstanding Electrical Engineer and Distinguished Engineering Alumnus Awards from Purdue University, and the IEEE Computer Society Golden Core Award. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS from 1990 to 1994 and was the Founding Editor-in-Chief of the *Journal of VLSI Signal Processing*. In addition, he has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and the IEEE JOURNAL OF SOLID-STATE CIRCUITS. He has been a member of the Board of Governors of the IEEE Computer Society from 1987 to 1991, the IEEE Signal Processing Society from 1992 to 1994, and the IEEE Solid-State Circuits Council/Society from 1986 to 1991. He has been a member of the IEEE History Committee from 1996 to 2004, the IEEE Fellows Committee from 2000 to 2003, the IEEE James H. Mulligan, Jr., Education Medal Committee from 2007 to 2011, the IEEE Awards Planning and Policy Committee from 2011 to 2013, the IEEE Awards Board Awards Review Committee from 2014 to 2016, the IEEE Awards Board from 2015 to 2016, and the IEEE Awards Policy and Portfolio Review Committee in 2017. He has chaired a number of conferences.