Low-Latency Optimizations and Architectures for Compression Algorithms implemented in (Programmable) Hardware

by

Ing. Matěj Bartík

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics

Department of Digital Design

Prague, April 2021

Supervisor:
Dr. Ing. Sven Ubik
Department of Technology for Network Applications
CESNET z.s.p.o.
Zikova 4
160 00 Prague 6
Czech Republic

Co-Supervisor:
Ing. Pavel Kubalík, Ph.D.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and Contributions

This dissertation thesis focuses on researching which compression algorithms are suitable for a very specific use case of real-time multimedia transmission systems such as CESNET's MVTP, providing excellent image quality and low-latency operation at the same time. However, the MVTP's throughput requirements exceeding the capabilities of used communication interfaces slightly in certain situations. Because no hardware implementations of universal lossless compression algorithms met the MVTP's requirements, some new hardware-based optimizations and architectures had to be discovered. Some of these optimizations were inspired by software implementations of so-called "fast" lossless compression algorithms, which trade a worse compression ratio for a better compression speed.

In particular, the main contributions of the dissertation thesis are as follows:

1. Parallel (8-way) & Low-Latency Architecture for "Match Search Unit" capable of delivering the throughput of 16 Gbps with the latency of only 6 clock cycles.
2. Memory-optimized data flow, which allows the "Match Search Unit" to generate less stalls in data processing. The principle is to store a combined entry of data and data's original address instead of the original address only.
3. Technique for masking "Match Length Finding" initial latency to reduce the number of "stalls". I propose using the available memory data width to double the actual performance in certain phases of the compression process.
4. Novel architecture for implementing a status register, which is commonly used to store an information, of which memory entry is (in-)valid. The architecture is portable to any modern FPGAs.
5. Benchmarking methodology for digital designs using Xilinx synthesis tools, which helped me with the evaluation of the novel status register architecture.

Keywords: lossless, compression, dictionary, LZ4, LZ77, high-throughput, low-latency, digital design, architecture, optimization, hardware, FPGA, MVTP.

As a collaborator of Ing. Matěj Bartík and a co-author of his papers, I agree with Ing. Matěj Bartík's authorship of the research results, as stated in this dissertation thesis.

.....

Ing. Tomáš Beneš

Acknowledgements

First of all, I would like to express my gratitude to myself because of the strong will, endurance, and effort needed to complete my dissertation thesis regardless of my supervisors and other circumstances. Fulfilling the requirements for obtaining a Ph.D. degree has not been easy for any doctoral student, even with the aid of a competent supervisor(s) [1].

Therefore, I would like to thank my supervisor Dr. Ing. Sven Ubik for the initial research topic, which later evolved into a more viable scientific topic summarized by this dissertation thesis. I am also glad you allowed me a significant flexibility to work on the research and my assignments in CESNET.

I would like to express my deepest gratitude to my dear colleagues: doc. Ing. Petr Fišer, Ph.D. and doc. Ing. Jan Schmidt, Ph.D. for their guidance, consultations, valuable comments and feedback, and proofreading. In fact, they treated me the same way as their own doctoral students without having any benefits. I also would thank the head of the department of digital design, doc. Ing. Hana Kubátová, CSc. for her infinite patience.

I need to mention Ing. Tomáš Beneš and Ing. Karel Hynek, my former masters' students. I had the honor of being their supervisor and had the option to see their progress towards their academic careers.

Finally, my greatest thanks goes to my family members (especially my dear mother) for their infinite patience and support.

I would like to thank CESNET z.s.p.o. and Czech Technical University for partial material and financial support. Besides that, my research has been partially supported by the Technology Agency of the Czech Republic, grants:

- LM2010005 "Large Infrastructure CESNET"
- EF16_013/0001797 "E-infrastructure CESNET – modernization"

and by the Grant Agency of the Czech Technical University in Prague, grants:

- SGS15/119/OHK3/1T/18 "Attack-Resistant and Fault-Tolerant Architectures Based on Reconfigurable Devices"

- SGS16/121/OHK3/1T/18 “Dependable architectures suitable for FPGAs”
- SGS17/017/OHK3/3T/18 “Dependable and attack-resistant architectures for programmable devices”
- SGS20/211/OHK3/3T/18 “Design, programming and verification of embedded systems”

Life Motto

Obsessed is just a word the lazy use to describe the dedicated.

Contents

List of Figures

List of Tables

Abbreviations

Common Mathematical Functions and Operators

10_2	Numbers' radices are designated with a subscript
$\mathbf{b}$	Vector $\mathbf{b}$
b_i	the i^{th} element of vector $\mathbf{b}$
$\Omega(x)$	The big Ω notation
$O(x)$	The big O notation
$\Theta(x)$	The big Θ notation

Miscellaneous Abbreviations

ASIC	Application Specific Integrated Circuit
BRAM	Block Random Access Memory
CAM	Content Addressable Memory
CESNET	Czech Educational and Scientific NETwork
CODEC	COder-DECoder or COmpression-DECompression
CPU	Central Processing Unit
DDR3	Double–Data–Rate 3 SDRAM
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EAV	End of Active Video
FF	Flip–Flop
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
FPS	Frames per second
GIF	Graphics Interchange Format
Gbps	Gigabit per second
HD	High Definition
IP	Internet Protocol or Intellectual Property

Miscellaneous Abbreviations – Continued

JPEG	Joint Photographic Experts Group
JPEG–XS	JPEG eXtra Small or JPEG eXtra Speed
L1	Level 1
LRU	Least Recently Used
LSIC	Linear Small Integer Code
LUT	Look–Up Table
LZ	Lempel–Ziv
LZMA	Lempel–Ziv–Markov–Chain Algorithm
LZO	Lempel–Ziv–Oberhumer
LZW	Lempel–Ziv–Welch
Mbps	Megabit per second
MLF	Match Length Finder
MTPP	Modular Traffic Processing Platform
MSU	Match Search Unit
MVTP	Modular Video Transmission Platform
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PID	Proportional–Integral–Derivative
PNG	Portable Network Graphics
RAM	Random Access Memory
RGB	Red, Green, and Blue
RJ45	Registered Jack–45
RLE	Run–Length Encoding
RTP	Real–time Transport Protocol
SAV	Start of Active Video
SDI	Serial Digital Interface
SDRAM	Synchronous Dynamic Random Access Memory
SFP	Small Form–factor Pluggable
SMPTE	Society of Motion Picture and Television Engineers
SRAM	Static Random Access Memory
SSD	Solid–State Drive
TICO	TIny COdec
UDP	User Datagram Protocol
UHD	Ultra High Definition
VoIP	Voice over Internet Protocol
XFP	10 Gigabit Small Form–factor Pluggable

Chapter 1

Introduction

Communication services are more and more critical for today's world and society. This digital era has begun with the invention of the Internet and related services. The development has been accelerated by multimedia (motion images and sound) transmission systems and services, which have allowed participants from opposite sides of the world to collaborate in real-time[1]. The majority of such systems and services have been using asynchronous networks.

The requirements and needs of such users have become even harder to satisfy every year. The demands such as better image resolution, sharper images without compression artifacts, and clear audio became essential for effective collaboration.

Thus, the communication network bandwidth had to be increased. New (complex) compression algorithms were invented to satisfy these requirements, although the commonly used compression algorithms are usually lossy. The amount of processed data and the complexity of algorithms consequently have increased latency.

The latency itself does not need to be an issue if it is not excessive above human senses and abilities. For a (physically) shorter interconnection, several milliseconds' latency cannot be observed by a human, and the impact is negligible (VoIP services, for example). On the other hand, some use case scenarios (such as telesurgery or musical performances) can be heavily affected by the increased latency.

	London	**New York**	**Prague**	**San Francisco**	**Santiago**	**Tokyo**
London	—	71.4	29.1	133.5	196.1	245.3
New York	71.6	—	99.5	72.7	133.8	176.2
Prague	28.7	99.5	—	168.7	240.1	259.3
San Francisco	133.2	72.7	168.7	—	191.4	109.2
Santiago	196.1	134.0	240.0	191.4	—	323.8
Tokyo	245.4	176.3	259.1	109.1	323.9	—

Table 1.1: Typical network latency (in milliseconds) between several cities. [7]

[1]Even more important during the on-going global pandemic of SARS-CoV-2

For such image quality-critical or latency-sensitive systems, the latency should be decreased at all costs. In general, some trade-offs must be made, for example:

- An optimized (dedicated) hardware solution is preferred, which tends to be more expensive than a software counterpart.
- Dedicated network routes (GÉANT [9] for example) with no other traffic are often used, reducing actual network latency by 50% usually [10] compared to the typical latency of commercially operated networks (see Table 1.1 for examples of various intercontinental and transcontinental Internet routes).
- Less complex and fast compression algorithms are used, therefore trading a compression ratio for the required network bandwidth. Also, the used compression algorithm should be (visually-)lossless.

It is complicated and expensive to decrease a network path's latency between two (or more) endpoints running a latency-sensitive application. A crucial question arises: is reducing the latency worth the effort and money? The answer is, it is worthy under some circumstances, as demonstrated on the new optical fiber route laid out between New York and Chicago [11], which decreased the latency by three milliseconds.

Therefore, this dissertation thesis aims to invent, implement, and evaluate new optimizations to lower the endpoints latency, which will be suitable for hardware-implemented lossless compression algorithms.

1.1 Motivation

As stated above, some latency-sensitive and image quality-critical use case scenarios exist. I would like to present an example of CESNET's MVTP (Modular Video Transmission Platform) endpoint system for high-quality/low-latency motion image transmissions. The MVTP has been an experimental broadcast system that allows the transmission and reception of (multiple) video streams in high-quality 4K/UHD resolution.

Besides video streams, transmission and reception of ancillary data as defined in SMPTE 291M [12] has been supported. The system's capabilities are constrained by a physical networking interface (10G Ethernet) because a full 4K/UHD stream requires 12 Gbps of total throughput usually. There are two modes of operation:

- **Uncompressed mode:** MVTP expects an incoming stream with reduced image sub-sampling (4:2:2 instead of full 4:4:4), and auxiliary data like blanking periods are omitted. Due to the absence of a complex compression, the required bandwidth can exceed the Ethernet interface capabilities for certain video formats [13].
- **JPEG2000 compression mode**: the JPEG2000 compression is lossy (but it does not harm the image quality significantly). However, it supports all 4K/UHD data features and can compress the stream while requiring less than 1 Gbps of bandwidth.

On the other hand, the JPEG2000 CODEC requires three times larger FPGA than MVTP with uncompressed mode only.

1.2 Problem Definition

The MVTP supports two basic modes of operations: uncompressed or JPEG2000 compression. There is a significant difference in the data flow demonstrated on a "Full HD" image, which consists of 1080 pixel lines. In uncompressed mode, the image data are processed as a line of image pixels. The amount of time (the latency) represented by the line can be expressed as 1 $sec/FPS/Lines \approx 15.4\ \mu s$ for a 60 FPS stream.

However, the used JPEG2000 CODEC implementation requires a complete image (one frame) to be buffered first prior to the processing. The CODEC can process only one frame at the same time. Thus the minimal time (latency) of one frame can be expressed as 1 $sec/FPS \approx 16.6\ ms$. In the case of MVTP, a commercial implementation of JPEG2000 provided by intoPix [14] is used. IntoPix states an average JPEG2000 CODEC implementation has latency of 1.5 frame per operation [14], resulting into added latency of 3 frames (equivalent to 49.8 ms) for 60 FPS stream. The latency can be further increased, if the used FPS is lower.

To allow remote collaboration in real-time, overall latency of 100 milliseconds is considered as a firm limit. For some advanced use cases, such as telesurgery or musical performances, the maximum latency should be less than 50 ms. The latency of a typical setup for bidirectional transmission (see Fig. 1.1) consists of:

- Network latency [7],
- MVTP endpoint latency, depending on used mode,
- Low-latency camera (about 5 ms [15]),
- Low-latency display (about 5 ms; 1-2 ms at best).

Therefore, it is clear the JPEG2000 CODEC latency prevents the requirements of the latency-sensitive MVTP from being satisfied. The estimated JPEG2000 CODECs' latency is almost the same as the requirement for the critical applications.

It is obvious we have no influence on a network's latency (it is beyond our control) and little influence on peripheral's latency. The substantial portion of latency is introduced by the JPEG2000 CODEC used by the MVTP. The conclusion is rather simple: MVTP needs a (de-)compression engine with these requirements and properties:

- The compression algorithm does not need to be as powerful as the JPEG2000, thus saving about 10% of the required bandwidth only is considered to be sufficient [13] (see Table 3.1) for certain use cases (1080p at 24, 25, and 30 FPS or 4K at 60 FPS).
- Latency should be kept as low as possible.

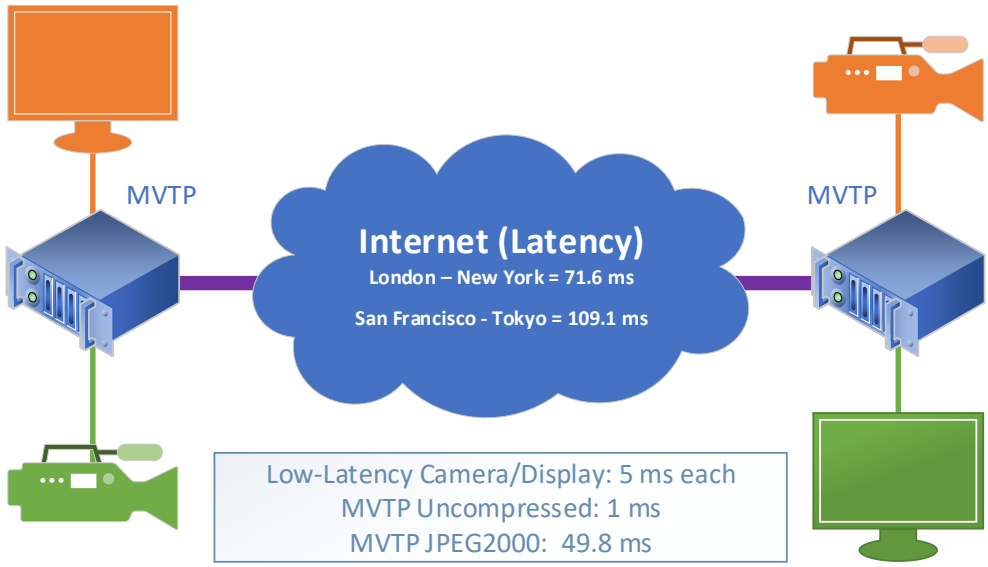


Figure 1.1: Latency of an example MVTP setup for real-time collaboration.

- The compression algorithm should be universal to support various data types, including video, audio, subtitles, and other ancillary data as defined in SMPTE 291M [12].
- Low utilization of FPGA resource utilization is preferred, but optional.

1.3 Goals of the Dissertation Thesis

Due to the fact no hardware implementation of a lossless compression algorithm reached the required throughput of 10 Gbps in 2014, a research needs to be conducted in the following areas:

1. Analysis of available (lossless) compression algorithms and their (theoretical) properties and features. A study of the latest trends and innovations in the field of compression algorithms.
2. Determine which algorithms are viable for the expected payloads types.
3. Survey on a compression algorithm hardware implementations.
4. Invent some optimizations to increase throughput towards the 10 Gbps requirement (and towards 100 Gbps in the near future) and to decrease the necessary latency to the minimum at the same time.

1.4 Structure of the Dissertation Thesis

The thesis is organized into seven chapters, as follows:

1. *Introduction*: Describes the motivation behind our effort together with our goals.
2. *Background and State-of-the-Art*: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.
3. *LZ4 Introduction and Analysis from a Hardware Designer Point of View*: Presents an initial survey of the LZ4 algorithm (and it's reference software implementation) as a representative example of so-called "fast" lossless compression algorithms.
4. *Highly Parallel Match Search Unit Architecture*: Explores the hash table based architecture and possible optimizations towards increased parallelism of a compression engine's "Match Search Unit."
5. *High Throughput and Low Latency LZ4 Compressor on FPGA*: Describes low-latency and high-throughput FPGA implementation of the LZ4 compression algorithm. The implementation uses our "Match Search Unit" architecture while delivering a throughput of 6 Gbps.
6. *Novel Status Register Architecture*: Presents a new architecture suitable for implementing a status register. Viable use cases are compression dictionaries and histogram calculations. Also presents a benchmarking methodology for digital designs using Xilinx synthesis tools, which helped me with the fair evaluation of the novel status register architecture.
7. *Conclusions*: Summarizes the research results, suggests a possible topic for further research (the literal length and match length limit concept) and concludes the thesis.

Chapter 2

Background and State-of-the-Art

This chapter presents a summary of the previous related work comprising the state-of-the-art associated with this dissertation thesis. The chapter includes the technical background describing the SDI, the MVTP platform, and describing the fundamental principles and properties of various compression algorithms and techniques.

2.1 Serial Digital Interface

Serial Digital Interface (SDI) is a digital video interface designed by the "Society of Motion Picture and Television Engineers" (SMTPE) for professional usage, especially in broadcasting domain. There are plenty of standards dealing with various aspect of SDI to support high bitrates (starting at 270 Mbps to 12 Gbps), new video formats and so on. Despite the number of standards, the fundamental frame format (see Fig. 2.1) has not changed since the introduction of the first standard in 1989.

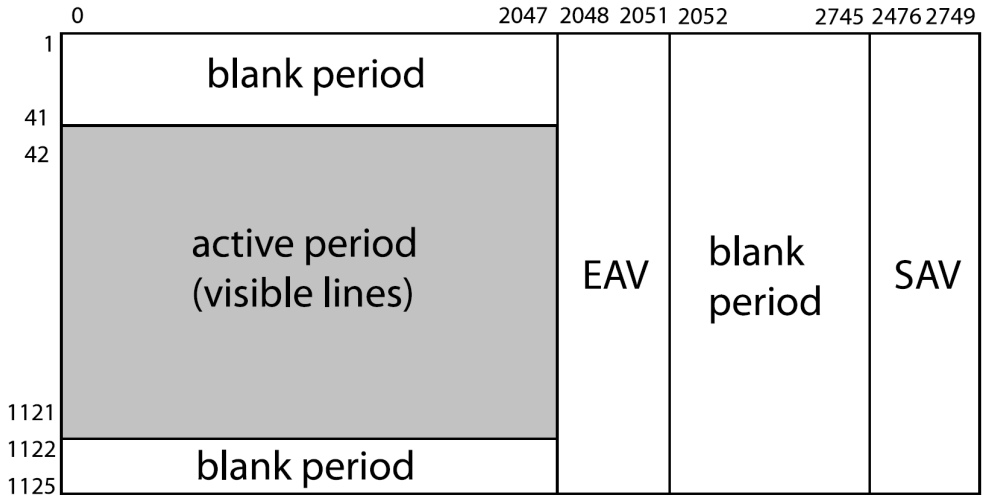


Figure 2.1: Simplified structure of the SDI frame format (2K example). [2]

The frame consists of defined numbers of lines (depending on used resolution), where each line (see Fig. 2.2) consists of several regions such as "start of active video (SAV)", "end of active video (EAV)", line number, checksum, synchronization, and "blanking area" which is used to embed ancillary data such as embedded audio [3].

Figure 2.2: SDI line format. [3]

2.2 Modular Video Transmission Platform (MVTP)

The MVTP is a scalable and modular platform for receiving and transmitting multimedia streams over an asynchronous network in real–time [2]. The designed system emphasized low-latency/high-throughput communication to satisfy real-time requirements. The MVTP architecture is based on MTPP (Modular Traffic Processing Platform) [4], a predecessor of the MVTP.

MVTP devices operate in pairs usually, but other network configurations are possible. All MVTP devices are equipped with multiple SDI interfaces (up to 8 input and 8 output interfaces) and Ethernet interfaces (10 Gbps XFP/SFP+ and/or 1 Gbps RJ45 connectors). The key features of MVTP are:

- 4K/UHD resolution (up to a resolution of 4096×2160 pixels),
- up to 60 FPS, interlaced/progressive format,
- up to 3G-SDI [16] interface support (a new generation with 12G-SDI [17] interface is currently under development),
- each SDI channel is transported independently, but synchronization and variable SDI channel grouping is optional,
- uncompressed or JPEG2000 compression operation modes (compression engine process data at full link speed, e.g., 3 Gbps in case of currently used 3G-SDI standards),
- very low latency of less than 1 ms in the uncompressed mode [18].

The MVTP receiver side is synchronized to a transmitting device by PID regulators measuring the time to pass through the receiver's buffer [18]. When the receiver and

transmitter clocks are synchronized to each other, there is theoretically no need for buffering data in the uncompressed mode of operation. However, some small buffers are still present, allowing the PID regulator to operate properly. Besides, the buffer masks a network jitter.

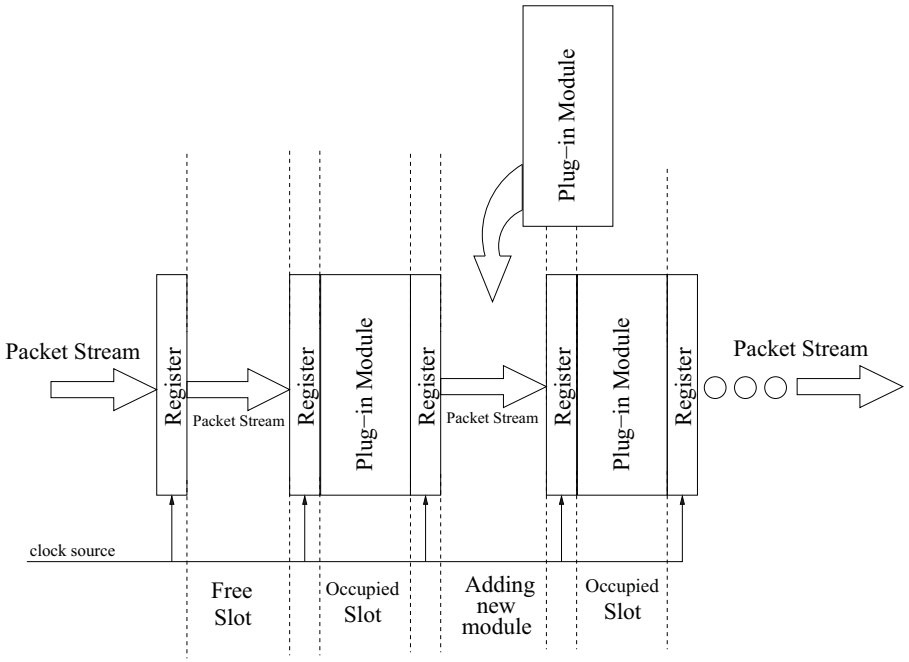


Figure 2.3: MTPP pipeline architecture. [4]

2.2.1 Conversion Process of an SDI Stream into IP Packets

An incoming SDI data stream contains multiple channels of multimedia data: the video payload is represented by luminance and chroma samples and an ancillary data channel carrying embedded audio, subtitles, and other service information. The incoming SDI data stream is being processed by an MTPP [4] inspired processing pipeline. The pipeline consists of several modules (see Fig. 2.3), and therefore acts like a systolic array. Such modules perform:

- extracting video samples (encoded using a 20-bit word coding scheme [19]) and various control signals,
- omitting the inactive area, EAV, SAV, and other non-video data from the SDI frame (Fig. 2.1) to save some of the required bandwidth; with the exception of an embedded audio,
- performing (optional) JPEG2000 compression,
- aligning the data for the network processing (a conversion from 20-bit to 64-bit),
- adding protocols headers (RTP, UDP, and IP).

In case the ancillary data (embedded audio) are required to be transmitted, there is a parallel datapath with the same pipeline stages applied (except compression). Due to the fact the video and the audio data are in the same line, two consequent IP packets will be generated.

Lastly, the stream is forwarded to the MVTP network processing part. A Jumbo IP packets [20] (each packet represents a one-pixel line) are used to minimize a transmission overhead.

An asynchronous computer network such as the Internet does not guarantee the packets will be received in the same order as they were transmitted. However, using a dedicated network route with (almost) no other traffic minimizes the risk of "swapping" packets on the way (and also helps to keep the network jitter low). Due to the fact the video and ancillary data from the same pixel line are transmitted in two consecutive packets, the required size of the receiving buffer could be low. The smaller buffer (and all packets delivered mostly in-order) lowers the latency on the receiving side.

Therefore, usage of multiple compression algorithms (suitable for a different data type) will likely lead to:

- IP packets can be generated in a wrong (non-deterministic) order because of the different (computational) complexity of such compression algorithms,
- the hardware architecture will require multiple compression algorithms to be implemented resulting in a more complex datapath including more complex output packet multiplexing,
- same applies also to the receiving side, where multiple decompression algorithms must be implemented as well,
- a larger buffer will be required to properly re-order all incoming packets prior processing, which results in increased latency.

However, it is impossible to support and implement a data type specific compression for all existing data types which can be embedded in the ancillary data region nor to apply a lossy compression for general binary data. Therefore, only the usage of a universal lossless compression, which also guarantees the same computational complexity. This behavior allows IP packets to be generated in order. It is also possible to keep a single datapath with a single implementation (engine) of such compression algorithm.

2.2.2 IntoPIX JPEG2000 CODEC

The MVTP uses the IntoPIX JPEG 2000 CODEC [14]. The JPEG2000 compression algorithm is based on a discrete wavelet transformation [21]. Every SDI frame is compressed when an entire frame is buffered completely, therefore affecting the latency. IntoPix states an average JPEG2000 CODEC implementation has a latency of 1.5 frame per operation [14], resulting in an added latency of 3 frames (equivalent to 49.8 ms) for 60

FPS stream. Some early JPEG2000 implementations (limited to 30 FPS) were capable of reaching a latency of 6 frames which is approximately 200 ms [22].

The CODEC processing speed is equivalent to the link speed of the source interface (3G-SDI actually), therefore 4 or 8 CODECs (for 3G-SDI or HD-SDI respectively) are required to transmit and receive a single image in 4K/UHD resolution. It is not possible to utilize combining two incoming streams to feed the compressor engine with data.

The IntoPIX JPEG 2000 CODEC allows decreasing the required network bandwidth, but it also increases significantly the amount of used FPGA resources. It requires more than twice the FPGA resources than the design with implemented uncompressed mode only. The IntoPIX JPEG 2000 CODEC also requires some additional DDR3 memory chips to implement frame buffers.

2.2.3 MVTP Summary

I described the principles of the MVTP platform, including the IntoPIX JPEG2000 CODEC, including aspects that made the CODEC not viable for the defined use case: the excessive latency, the need for a larger FPGA, and the additional DDR3 memory chips for the MVTP design. The positive aspect of the JPEG2000 is the bandwidth reduction by approximately 90%. The lossy compression is also not ideal for certain use cases such as the transmission of medical images [23] [24], where any color shifts could harm a patient.

Due to these facts, I should be looking for a universal compression algorithm that is not as powerful as the JPEG2000; however, it will be viable for various data types and will keep IP packet generated in order. In certain use cases (1080p at 24, 25, and 30 FPS or 4K at 60 FPS), reducing the required bandwidth by 10% is considered to be sufficient [13] (see Table 3.1) to "squeeze" the network traffic into 1G or 10G Ethernet.

2.3 Fundamentals of Compression Algorithms

Compression is a process of finding and removing redundant information from input and transforming the input to an output using fewer bits. The basic types of compression schemes are described in this section [25]. This analysis emphasized finding a compression algorithm candidate suitable for the presented use case (MVTP). The requirements are:

- Universal compression for every data type (video, audio, general data).
- High throughput at gigabits per second.
- Low-latency for real-time interaction.
- Little overhead for incompressible data.
- Good compression ratio is the least important parameter in our case.

A summary of elementary compression techniques, classification, and properties follows.

2.3.1 Compression Ratio and Compression Dictionary

The compression ratio is the most common metric for the comparison of compression algorithms. It is defined as the relation between the sizes of the original and the compressed data. There is a directly proportional relationship between the compression ratio and the size of a compression dictionary, where a larger dictionary increases the probability of finding a piece of duplicate information, therefore improving the compression ratio. The compression ratio is usually evaluated on the "corpora" (such as Calgary [26], Canterbury [27], and Silesia [28]), which are a set of sample files designed for testing of compression algorithm properties.

2.3.2 Lossless or Lossy?

The second most common classification of compression algorithms is their ability to reconstruct original data, e.g., lossless or lossy compression. The lossless compression enables the restoration of the original content from the compressed data. The lossy compression scheme uses the imperfection of human senses (vision, hearing) to omit unnecessary information. The lossy compression algorithms typically reach better compression ratios than lossless counterparts. Examples of lossless schemes are LZ77 [29], LZ78 [30], LZW [31], Huffman encoding [32], etc. and examples of lossy schemes (suitable for image compression) are JPEG [33], H.264 [34], H.265 [35], etc.

2.3.3 Symmetry

The symmetry is one of the elementary properties of compression algorithms. The compression scheme is symmetric when the compression and decompression processes have the same complexity (algorithmic or time). In the case of different complexity (of encoding and decoding), the compression scheme is called asymmetric. Asymmetric algorithms are more common, and the compression phase usually has higher complexity than decompression.

2.3.4 Number of Input Data Passes Through a Compression Algorithm

Another important property is the number of passes required for the input data through the compression algorithm. As an example, file-based compression tools usually use multiple compression schemes and each compression scheme can have multiple passes [25] [36]. A higher number of passes allows to achieve a better compression ratio, but it also increases the latency. Therefore high number of passes may reduce throughput and makes the scheme unsuitable for real-time applications (for example, LZMA). Representative examples of single pass algorithms are LZ77, LZ78, and LZW.

2.3.5 Suitability for Certain Data Types

The first kind of compression scheme is universal algorithms that can compress any data (text, video, audio, binary files) with usually worse compression ratio than the compression

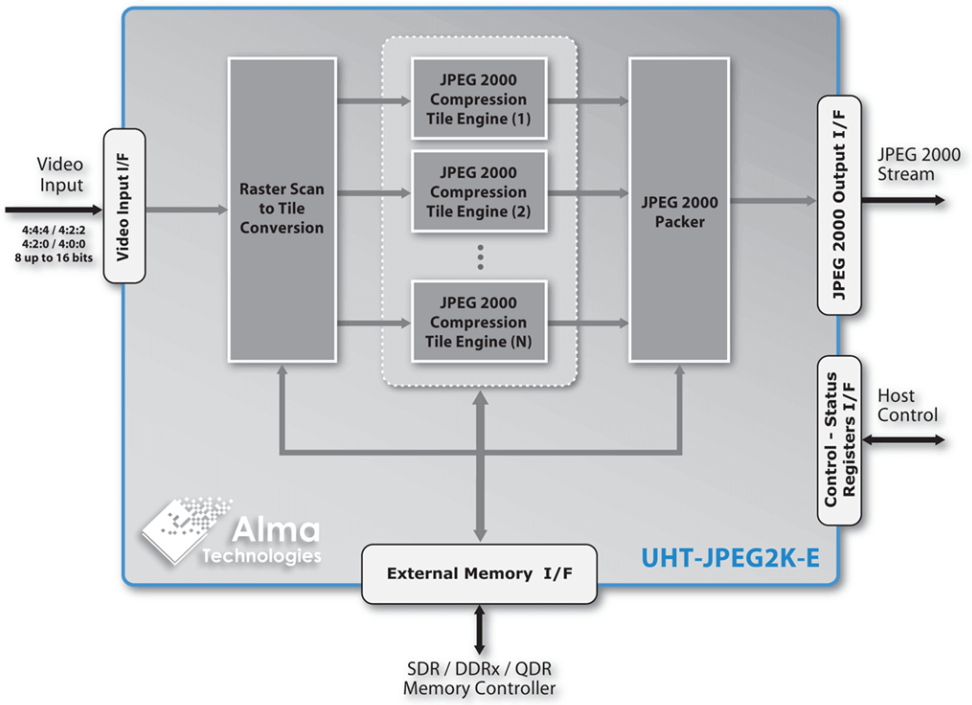


Figure 2.4: The architecture of a JPEG2000 hardware based compressor. [5]

algorithms designed for specific data types. Data specific algorithms can achieve better a compression ratio, but these algorithms are typically very complex [37]. The higher complexity typically means a higher number of passes or more compression algorithms are used at the same time. Compression algorithms can be further divided into three groups:

- **Universal algorithm**: typically merges input data into blocks where each block is compressed separately. A special case of block-based algorithms is the stream compression algorithm, which uses very small blocks that can be stored in small buffers closely associated with algorithm processing (e.g., CPU L1 cache). Examples of such compression algorithms are LZ4 [6] and LZO [38].

- **Frame-based algorithm**: designed for the compression of still images. These methods compress an individual frame/image at once. The most known algorithms are GIF, JPEG(2000), PNG, etc. The time and algorithmic complexity are usually higher than for universal algorithms. To lower the latency requirements, some implementations (such as IntoPix TICO [39] or very recent JPEG-XS [40]) were introduced in recent years. Such implementations are designed to split a frame into smaller segments (several pixel lines) and to process them independently (see Fig. 2.4). Typical combined latency for compression and decompression is tens of pixel lines [40]. However, such implementations perform worse in term of compression ratio [41]. Additionally, it still requires processing the entire frame to prevent the data from being corrupted.

- **Inter-frame algorithm**: representative examples are H.264 [34] or H.265 [35]. They provide the best compression ratio, high compression rates. The basic principle is to encode the differences between two or more frames. The main issue of inter-frame compression is an increase in latency caused by the inter-frame principle (holding multiple frames buffered in memory) and computational complexity.

Therefore, it is evident the compression latency is in the best case directly proportional to the size of the processed data and the compression algorithm complexity.

2.4 A Brief Comparison of Hardware-Implemented Lossless Compression Algorithms

This section presents a brief analysis of various compression algorithms' most relevant hardware implementations, describing their common properties, advantages, and disadvantages. The analysis summarizes a selection of scientific papers [42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69].

2.4.1 Summary of Hardware Implementations

Articles published before the year of 2010 were mostly focused on a (re-)implementing software-implemented compression algorithm in hardware (as an accelerator). Such hardware implementations showed no significant speed improvements because no further (hardware) optimizations were used. Therefore, the "better" speed was an effect of "slow computers" against "hardware with no software overhead" used in that era. The majority of such articles lack many important results and properties like resource utilization, frequency, compression speed/throughput/latency, used FPGA chip, etc. Therefore, it was impossible to do a thorough survey of such fragmented information.

- The majority of designs are based on the LZ77 algorithm [29] or derived algorithms such as LZ78 [30], or LZW [31] implemented in FPGA. ASIC implementations are rare.
- LZ78 and LZW implementations are significantly slower than LZ77 implementations due to higher algorithm complexity.
- Only a few implementations are capable of reaching speeds above 2 Gbps.
- The latest designs experiment with new (LZ77-derived) algorithms focused on better compression ratio (LZMA [47, 48, 49]) or speed (LZ4 [6, 67, 68, 69]).
- The compression speed is improved by massive pipelining or parallelization (systolic arrays) [59] of the match searching mechanism [70].

- The majority of these FPGA implementations use embedded memory blocks (kilobytes in size) rather than external memory [43] such as DRAM or SRAM.
- Compression dictionaries use three fundamental approaches: CAM (Content Addressed Memory) [71], hash table [72], and small (shift) register array for stream operating implementations [44, 52, 67], where the dictionary stores just a few processed data words.

However, several exceptions with throughput above 2 Gbps exist [44, 64] in 2014. The respective throughput of the most advanced designs is still below the 10 Gbps requirement; however, they are very close. Some new accelerators have appeared [65, 66] in the following decade, thus confirming the supremacy of LZ77 and dictionary-based algorithms for high-throughput designs. All of these implementations have some common properties:

- Highly optimized and pipelined design,
- Multiple compression engines (many-core principle) or highly parallel match search algorithm,
- Emphasis on processing more than one bite per clock cycle,
- Throughput is measured "per device" usually, not per engine/core.

There is one additional conclusion: hardware implementations were not evolving fast enough to keep track of technological progress in communications, such as introduction of 100 Gbps and faster networks.

2.5 Modern and "Fast" Software Compression Algorithms

On the other hand, the world of software-implemented compression algorithms has been evolving significantly. Therefore, adapting some optimizations and concepts from the software world may increase the overall performance of hardware implementations. Thus, it is essential to analyze the latest trends used in modern compression algorithms and determine which optimizations can be adapted to improve hardware compression algorithm implementations' performance. I selected two candidates of such algorithms for a detailed examination.

2.5.1 LZ4

LZ4 compression algorithm [6] is one of the pioneers of fast algorithms. LZ4 is an LZ77 [29] derivate, and it is intended for "real-time" compression. Such an example of LZ4 usage is compression of Linux kernel [73]. Multiple experiments and measurements evaluating LZ4 for the real-time image and lossless video compression have been described in [74]. In

this paper, the authors assessed the suitability of some "fast" compression algorithms for real-time transmission of a 4K/UHD video stream.

It was demonstrated that LZ4 could be a way to improve the predecessor algorithms like DEFLATE by replacing the LZ77 front-end with LZ4 to improve the compression speed [75]. LZ4 can also be a high-performance alternative to ZLIB for scientific-data compression [76]. The influence of some CPU specific optimizations is described in [77].

2.5.2 LZO

LZO performs slightly worse than LZ4 in (de)compression speed, but compression speed is comparable to DEFLATE. An experiment [78] was conducted to prove the suitability of LZO for lossless image compression. The performance of LZO can be further improved by using a CPU specific optimization or running LZO with multiple threads [79].

2.5.3 Performance and Common Features

A representative example of a benchmark designed for testing and evaluating "fast" lossless compression algorithms has been published in [8]. The results are shown in Tab. A.1. The alternative benchmark focused on all kinds of lossless compression algorithms can be found on [80]. There are a few shared features among "fast" algorithms:

- They benefit from a massive use of a CPU specific optimization:
  - data are processed in the width equal to the word width of the particular CPU,
  - memory access is aligned.
- Dictionary is typically small to fit into CPU L1 cache.

2.6 The Research Question & Methods

There are two mutually exclusive approaches from which I had to choose: I can implement a compression algorithm for every desired data type supported by the MVTP architecture (and dealing with several drawbacks) or I can implement a single universal lossless compression algorithm (because lossy universal compression algorithms seems not to be feasible [81]).

The MVTP supports multiple data types (image, audio, text, and others) to comply with SMPTE 291M [12]. However, this standard is being updated every few years to support new data types and formats. Beside the SMPTE 291M, some other standards have been introduced such as SMPTE 334M [82] or SMPTE 2108-2 [83]. Therefore, I have decided an universal algorithm is a viable way to keep the MVTP architecture (forward) compatible despite introduction of new SMPTE standards in the close future.

On the other hand, it is evident the existing (hardware) implementations of universal lossless compression algorithms were not feasible for the presented use case (MVTP).

The reason was the hardware implementations (and their overall performances) were not evolving fast enough to match the progress of communication technologies. Therefore, is it possible to invent some new optimization and principles to improve both throughput and latency of such hardware implementations? Some feasible methods to reach this goal are:

- Determining lossless compression algorithms which have the potential to reach a throughput higher than 10 Gbps,
- Adopting ideas from the software world, especially from "fast" algorithms,
- Identifying bottlenecks of current hardware implementations,
- Implementing suggested optimizations to evaluate their impact.

Chapter 3

LZ4 Introduction and Analysis from a Hardware Designer Point of View

This chapter presents an initial survey of the LZ4 algorithm as a representative of so-called "fast" lossless compression algorithms. The LZ4 algorithm (and it's reference software implementation) was analyzed from a hardware designer's point of view to identify a data path, possible bottlenecks, and other (dis-)advantages. A simple hardware implementation of the LZ4 algorithm was designed on Xilinx Virtex-6 and 7-Series FPGAs to verify the analysis results and claims.

Main findings are:

- LZ4 is based on LZ77, therefore LZ4 is also asymmetrical. This means we can focus our effort on the compression phase only because the decompression is supposed to be less complex (by definition) in term of decompression time or decompression computational complexity.
- LZ4 (and LZ77 as well) requires the input data to be processed only once (a single pass algorithm), therefore the minimal latency can be halved by definition compared to two-pass (or multiple-pass) compression algorithms.
- I identified the used hash table design (implementing compression dictionary) has a significant impact on throughput while clearing the hash table, and data buffers negatively influence the overall latency.
- I estimated the used hash algorithm can be efficiently implemented in hardware.

The content of this chapter is based on the following paper (which has been cited 21 times): **Bartík, M. and Ubik, S. and Kubalík, P.**, *"LZ4 Compression Algorithm on FPGA"*, 21st IEEE International Conference on Electronics, Circuits, and Systems, ISBN 978-1-4799-2451-6, pp. 179–182, Cairo, Egypt, 2015 [A.1].

LZ4 Compression Algorithm on FPGA

Matěj Bartík
CTU FIT & CESNET
matej.bartik@fit.cvut.cz

Sven Ubik
CESNET
ubik@cesnet.cz

Pavel Kubalík
CTU FIT
pavel.kubalik@fit.cvut.cz

***Abstract*—This paper describes analysis and implementation of a LZ4 compression algorithm. LZ4 is derived from a standard LZ77 compression algorithm and is focused on the compression and decompression speed. The LZ4 lossless compression algorithm was analyzed regarding its suitability for hardware implementation. The first step of this research is based on software implementation of LZ4 with regard to the future hardware implementation. As a second step, a simple hardware implementation of LZ4 is evaluated for bottlenecks in the original LZ4 code. Xilinx Virtex–6 and 7–Series FPGAs are used to obtain experimental results. These results are compared to the industry competitor.**

I. Introduction & Motivation

Fast lossless compression algorithms become more important than before, even though they do not reach compression ratios of their predecessors. The main usage of these algorithms is in reducing bandwidth reqirements, typically in multimedia applications, where bandwidth of a multimedia interface is slighly higher than of a transmission line. For example [1], it allows transmition of 12G-SDI (4K60p uncompressed video) over 10 Gbit Ethernet or Full HD video can be fit into a standard metalic gigabit ethernet. The following parameters are important for such a kind of an application:

- Universal compression for every data type (video, audio, service informations),
- High throughput for video,
- Low–Latency for real–time interaction,
- Little overhead for uncompressible data,
- Compression ratio is the least important parameter.

There are two algorithms which satisfy these requirements. The LZ4 compression algorithm, which is mentioned above and the LZO [2]. The LZ4 better fits our requirements [3]. Our research is focused on the LZ4 compression algorithm. LZ4 is based on LZ77 (Lempel – Ziv) [4] like other fast compression algorithms, because LZ77 is one of the few one–pass compression algorithm [5].

It has been shown that LZ77 can be used in an application, where throughput up to 9.17 Gbps is required [6], [7]. The authors also said, that their design can operate on higher frequencies, when pipelining or loop unrolling will be used. This will enable to reach 10G+ throughput.

However, this architecture is using extremely small search and look–ahead buffers (only a few bits wide), that make the architecture very inappropriate for a practically useful compression application.

A. Other Examples of LZ4 Usage

- Fast (de)compression of GNU/Linux kernel [8],
- A new use can be (de)compressing data between Solid–State Drive (SSD) for increasing throughput. There is a high probability, that SSD's vendors are prefering high throuhput/low latency algorithms based on LZ77.
- The LZ77 is also used for (de)compressing FPGA bitstreams [9].
- And also for the IP packet compression [10].

II. Teoretical background of the LZ4

It the following sections we describe main features of the LZ4 lossless compression algorithm [11], [12] when compared to the LZ77. The biggest advantages of the LZ4 are a hash based match search and the support of match overlaping.

A. LZ4 Lossless (De)compression Algorithm

LZ4 itself is not an algorithm in the original meaning. LZ4 only defines an output data format (like LZ77 [13]). This allows to create various derivates of compression methods (with different speeds and compression ratios) and also allows to decompress the LZ4 file format by a single tool, no matter what compression algorithm was used.

However, the author of LZ4 created a reference code in the C programming language and this code has been ported to many other programming languages. The LZ4 code contain various optimizations for different processor architectures to achieve maximum performance.

LZ4 as well as LZ77 is an asymmetric compression method, where decompression is much simplier (and faster) than compression. The decompression process is very similar to LZ77. It is based on copying literals from the decoded part.

B. Pseudocode of LZ4

A very simplified reference code expect the following parameters (byte oriented):

- I : An input data buffer
- O : An output data buffer
- Isize : Size of input buffer

```
pointer ip = 0; // address to I
pointer op = 0; // address to O
hash_table HT; // Zeroed
```

```
while (ip < Isize-5) {
  h_adr = read U32 *ip, calculate hash;
  read possible match address HT(h_adr);
  store current address HT(h_adr)=ip;

  if !(match found) ||
     !(distance < offset_limit) ip++;
  else {
    if (ip > Isize-12) break;

    // writing to O buffer
    encode Token;
    encode Literals length;
    copy literals;
    encode Offset;
    encode Match length;

    increase input and output pointers;
  }
}

encode last literals;
return output pointer (data size);
```

III. LZ4 Analysis from the Hardware Designer View

In this section we discuss difficulties and advantages when implementing LZ4 compression algorithm in FPGA. The LZ4 implementation on the standard processor architecture benefits from high frequencies of CPU (Central Processing Unit) or RAM (Random Access Memory), instruction and data caches and software based optimization (software pipelining, loop unrolling, usage of compiler/processor specific instructions) [14]. For this evaluation, revision r127 is used for analysis.

A. Hash Table and Hashing Algorithm

The first improvement of LZ4 againts LZ77 is the use of a hash table for storing reference addresses. LZ77 uses a searching algorithm for match detection with linear complexity. Due to performance reasons, LZ4 uses the least complex method for storing addresses, overwriting the previous address in the hash table. The hash table size is designed to fit the L1 Data Cache in the standard processor architecture.

One of possible further improvements is to implement the hash table as a regular cache with the limited degree of associativity extended with the LRU (Least Recently Used) algorithm. This may improve the compression ratio without significant performance loss.

The hash calculation algorithm is based on the Fibonacci hashing principle [15]. Read value is multiplacated with a constant 2654435761. This number is the closest Prime to $\frac{2^{32}}{\phi} = 2654435769$, where ϕ is the value of the Golden ratio. This 32-bit constant can be easily multiplicated by four DSP48 slices (and three only after place & route phase) available since Xilinx Virtex–5 chip generation in 6 clock periods. Generated IP core is pipelined. The result of multiplication is trimmed to the highest bits used for hash table address.

The biggest weakness of the LZ4 hashing mechanism is zeroing of the hash table. A larger RAM implemented by the on-chip BlockRAM or a distributed RAM does not have a feature for clearing the entire RAM content by reset. For the first run, we can benefit from the bitstream loading process, because the RAM content is part of the bitstream. For next runs it is necessary to clear the RAM content, for example by linear passage and clearing memory cell, one by one.

B. Match Search and Memory Access for Reference Addresses

The process of match search starts from reading a reference address (read from hash table). A 32-bit data are read from the reference address and from the input pointer address and then, these values are compared. When the difference between these two addresses is less than offset limit, a match is found. Otherwise, the input pointer is increased by one.

However, buffers are byte oriented and LZ4 also supports 64-bit computing. The memory subsystem of the LZ4 algorithm should support 8-bit, 32-bit and 64-bit (optional for maximum memory performance) access. This will result into a complex memory subsystem. Alternatively the memory subsystem can be 8-bit only and support of the two remaining modes can be solves by reading portions of 8-bit values. This also solves problems with unaligned memory access (caused by increment of input pointer by a one).

The biggest disadvantage of a simple 8-bit memory subsystem is a massive loss of performance. There is also no mechanism, that prevents match searching from reference addresses, that are equal to zeroes. Zeroed address means no write to a hash table cell and may be skipped.

C. LZ4 Sequence Encoding

The encoding of an LZ4 sequence requires linear passage through the input buffer byte after byte. However, the literals can be copied in up to 64-bit data blocks between the input and output buffer. There are same problems as we discussed in the previous section.

D. Size of Input and Output Buffers

The LZ4 algorithm adds only a tiny overhead for the uncompressible data. It is necessary to generate one block with all literals and with literal match. That is not a problem to allocate the output buffer slightly bigger (necessary size can be calculated with the formula $o_{size} = i_{size} + \frac{i_{size}}{255} + 16$) than the input one. For example the 16kB input buffer will require 273 byte long overhead (0.4% relative). But the FPGA has hard RAM blocks (BlockRAM) with the limited sizes and limited bus widths. The most important thing is that all BlockRAMs have the same size. There are same several ways to solve this:

- The input and the output buffer will have the same size and pretend that the problem does not exist (buffers are bigger than limits).
- Limit the size of input buffer.

- Combine BlockRAM and Distributed RAM to provide additional space with significant complexity increase of the address decoder.

E. Other Sources of Inefficiency

A standard computer architecture works with memories in a simple way with one read/write in one clock cycle. However, FPGA's BlockRAM has the possibility of using a Dual-Port BlockRAM to increase memory throughput by reducing the time to read longer data, for example. That means further analysis of the LZ4 reference code to create an optimized code for the Dual-Port BlockRAM and its memory controller.

IV. Implementation of LZ4 Compression Algorithm

A. Principles of Implementation Platform

CESNETs MVTP–4K (Modular Video Transmission Platform – 4K) is an FPGA based system, that provides up to 8 input–output pairs of SDI (Serial Digital Interface) interfaces for transmitting 4K video content with audio. MVTP systems are communicating over optical version of 10G Ethernet. Uncompressed 4K transmission takes approximately 5 Gbps and a Full HD channel takes approximately 1.1 Gbps. The 4K video content is split into four quadrants and transmitted over four HD-SDI interface separately with synchronization. Each quadrant frame consist lines with video, audio and service data. The lengths of a line in each quadrant is less than the maximum size of an Ethernet jumbo frame payload, so one packet is created from each SDI line. This way of transmission allows for easy compensation of lost lines by duplicating the previous line. This property can be maintained with line-by-line compression. On the contrary, with inter-frame compression, the multimedia stream may disintegrate to the next synchronization frame. Next reason for lossless compression is to allow raw multimedia stream distribution between servers without loss of quality.

B. Assumptions for Implementation

- SDI signal is transmitted line by line. The length of one SDI line is slightly less than the maximum size of an IP packet payload in a jumbo Ethernet frame.
- We are looking for a block compression for IP packets, that is up to 9216 bytes. The closest power of 2 greater then this limit is 16kB. Therefore the size of input and output buffer will be 16kB.
- We reduce the hash table size from 4096 records (16 kB) to 1024 records (N = 10) and we reduce the size of buffer pointers (from 32–bit to 13–bit).
- The current system for video transmission is based on Xilinx Virtex-6 (XC6VLX240T-2FF1156), therefore design will be optimized for Virtex–6, but it can be easily transfered to Xilinx 7–Series FPGAs chips.

C. Implementation and Results

First, we created a simple design without any advanced features. The design is written and tested in VHDL language. The goal was to evaluate how LZ4 code exactly works and to provide information about bottlenecks. The LZ4 code was divided into a datapath and its control. The control is realized as a Moore finite state machine, that represents lines of the original C code. The FPGA resource utilization includes buffering all input and output signals. The current resource utilization for multiple Xilinx FPGAs are in Table I.

The required frequency of MVTP system is 156.25 MHz, so the compression core meets requirements of the MVTP. We didn't measure the precise throughput/latency of the implemented LZ4 core yet, because it was designed for evaluation only (based on the reference LZ4 source code).

The implemented datapath (Fig. 1) contains all important blocks, but finite state machine used for control is very complex and slow. The datapath design is also reduced to support only 8–bit operations, therefore operations with wider operands has to be split. Even when the design was simplified, it was sufficient for evaluation and critical parts of LZ4 compression algorithm for hardware implementation were found. The critical path is a memory controller, where the address bus is created from cascaded multiplexers (multiple pointers to the input memory are required by the LZ4 algorithm) and combined with an adder (for offset support) in the last stage.

D. Comparison with Competitor

The Helion LZRW3 core [16] has been selected for a very simple comparison. Both algorithms (LZRW and LZ4) are derived from the LZ77. The Helion LZRW core is a highly optimized state of the art industry solution. But the current version of LZ4 (even unoptimized) is comparable to the LZRW core in resource utilization (usually slightly more LUTs are taken because of two additional bits in memory subsystem and we are using three DSP48 blocks). Even that, Virtex–6 implementation take less resources.

The maximum frequency of the LZ4 core is quite higher (with same speed grades and synthesis technology, with exception of Artix–7 might be caused by limited routing capabilities in low–cost FPGAs) then LZRW and there is still a huge space for optimizations (software based pipelining of the reference source code caused duplication of all memory pointer registers).

TABLE I
Comparison with Helion LZRW3 Core [17]

Virtex–6 (XC6VLX75T-2FF784) Resource Utilization						
Solution	**Slices**	**LUTs**	**FFs**	**BRAMs[1]**	**DSP48s**	**Frequency**
LZ4	216	729	404	16 + 1[2]	3	224 MHz
Helion	225	770	N/A	4	0	198 MHz
Kintex–7 (XC7K70T-2FBG676) Resource Utilization						
Solution	**Slices**	**LUTs**	**FFs**	**BRAMs[1]**	**DSP48s**	**Frequency**
LZ4	266	891	441	16 + 1[2]	3	241 MHz
Helion	227	789	N/A	4	0	210 MHz
Artix–7 (XC7A100T-2FGG676) Resource Utilization						
Solution	**Slices**	**LUTs**	**FFs**	**BRAMs[1]**	**DSP48s**	**Frequency**
LZ4	243	764	375	16 + 1[2]	3	146 MHz
Helion	226	789	N/A	4	0	148 MHz

Note 1: LZ4 is using two 16 kB buffers for I/O, Helion is using 2 kB size.
Note 2: One BRAM is dedicated for the hash table for LZ4 design.

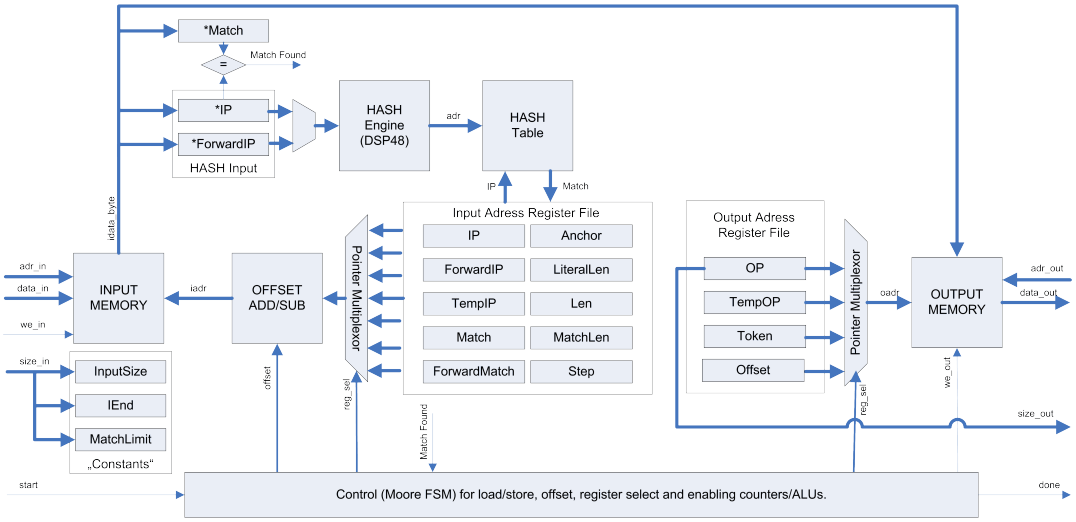


Fig. 1. Architecture of the LZ4 Lossless Compression Algorithm Design.

The LZRW core was synthesized with 25% buffers size, so we can be expected 20% deterioration of all results, as mentioned in the documentation of the LZRW core. Parameter comparison is summarized in Table I.

V. Future Work

We plan to improve the LZ4 lossless compression algorithm speed in hardware. Possible improvements may be pipelining or an advanced memory subsystem. The impact of these improvements will be measured. We expect a significant improvement by porting 64–bit CPU specific optimizations into an FPGA. This should exceed a theoretical (throughput) limit of current hardware implementations. This limit can be calculated by multiplying a frequency (200 MHz) and an operand–width (8–bits), that results into 1.6 Gbps peak performance [18] per compression core. We also plan to create and compare HLS (High Level Synthesis) version of the LZ4 reference C source code.

VI. Conclusion

We have implemented The LZ4 lossless compression algorithm on FPGA in VHDL (and compared to the industry solution) from a reference C code (r127). Limitations and bottlenecks hashing table zeroing, software pipelining overhead, differences between architectures, etc.) of the LZ4 lossless compression algorithm have been found during the process of analysis and implementation.

Acknowledgment

This research has been partially supported by the project SGS15/020/OHK3/1T/18.

References

[1] Hafi, L. ”Mapping SDI with a Light-Weight Compression for High Frame Rates and Ultra–HD 4K Transport over SMPTE 2022–5/6” [Online]. Available: tinyurl.com/o3lxgcl

[2] Oberhumer, M.:LZO real-time data compression library [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[3] Ruan Delgado Gomes, Yuri Gonzaga Gonalves da Costa, Lucenildo Lins Aquino Jnior, Manoel Gomes da Silva Neto, Alexandre Nbrega Duarte, and Guido Lemos de Souza Filho. 2013. A solution for transmitting and displaying UHD 3D raw videos using lossless compression. In Proceedings of the 19th Brazilian symposium on Multimedia and the web (WebMedia ’13). ACM, New York, NY, USA, 173-176.

[4] Rigler, S.; Bishop, W.; Kennings, A., ”FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms,” Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on , vol., no., pp.1235,1238, 22-26 April 2007

[5] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337343, 1977.

[6] Mehboob, R.; Khan, S.A.; Ahmed, Z.; Jamal, H.; Shahbaz, M., ”Multigig lossless data compression device,” Consumer Electronics, IEEE Transactions on , vol.56, no.3, pp.1927,1932, Aug. 2010

[7] El Ghany, M.A.A.; Salama, A.E.; Khalil, A.H., ”Design and Implementation of FPGA-based Systolic Array for LZ Data Compression,” Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on , vol., no., pp.3691,3695, 27-30 May 2007

[8] Lee, K. ”LZ4 Compression and Improving Boot Time” [Online]. Available: tinyurl.com/q3p4v2l

[9] Khu, A.: Xilinx FPGA Configuration Data Compression and Decompression. [Online]. Available: tinyurl.com/nhhgqbj

[10] Munteanu, D.; Williamson, C.; An FPGA-based Network Processor for IP Packet Compression. [Online]. Available: http://pages.cpsc.ucalgary.ca/ carey/papers/2005/FPGA.pdf

[11] Collet, Y.: RealTime Data Compression: Development blog on compression algorithms. [Online]. Available: tinyurl.com/qc9yve4

[12] Fiedler, O.: LZ-Family Data Compression Methods, Bachelor Thesis, 2014. [Online]. Available: http://tinyurl.com/opmd5sc

[13] Solomon, D.: Data Compression: The Complete Reference (Fourth ed.). 20007, Springer. ISBN 9781846286032.

[14] Kane, J.; Yang Q., ”Compression Speed Enhancements to LZO for Multi-core Systems,” Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on , vol., no., pp.108,115, 24-26 Oct. 2012

[15] Knuth, D. E.: The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998, ISBN 0-201-89685-0.

[16] Helion Technology Limited, LZRW Compression cores. [Online]. Available: http://www.heliontech.com/comp_lzrw.htm

[17] Xilinx Inc., LZRW1 & LZRW3 Lossless Data Compression (Helion) [Online]. Available: tinyurl.com/ngmtx25

[18] Naqvi, S.; Naqvi, R.; Riaz, R; Siddiqui, F. Optimized RTL design and implementation of LZW algorithm for high bandwidth applications [Online]. Available: http://pe.org.pl/articles/2011/4/68.pdf

Chapter 4

Highly Parallel Match Search Unit Architecture

This chapter further explores the hash table architecture (which is widely used for implementing a compression dictionary) and possible optimizations towards increased parallelism. The parallelism factor of the presented architecture is eight, which allows increasing throughput (up to 16 Gbps) and decreasing latency to the bare minimum (to just 6 clock cycles) at the same time. The presented architecture uses a shared dictionary, and it is suitable for any hardware-implemented LZ77 based compression engine.

The parallelization principle builds on the fact the reference LZ4 software implementation reads a 4-byte wide word from the byte oriented input buffer to calculate the hash table address (eg. address in compression dictionary) to find a match. If no match is found (via readback to the input buffer), the read address to the input buffer is incremented by one and the process repeats (see Fig. 1).

Therefore, the next word can be constructed as 3-bytes from the previous word plus a new byte read from the input buffer. Consequently, it is possible to construct multiple (overlapping) words at the same time by utilizing a wider memory read (16 bytes in this case instead of 4 byte originally). In the presented case, the total of eight words are read in a single memory transaction (see Fig. 2). The related hash calculation can be performed independently for each word. Thanks to the 8-way parallel compression dictionary, the hash addresses can be written in a single clock cycle.

Finally, I introduced a memory-optimized data flow (see Fig. 3), which allows the Match Search Unit to completely avoid stalls in the data processing. The fundamental principle is to store a combined entry of data and data's original address instead of the original address only, as it was introduced in the reference LZ4 software implementation. The reference software implementation requires a readback to the input buffer (this would cause stalls in hardware) to determine a match candidate is a true match or just a hash collision. Therefore, it is possible to determine a hash collision immediately in hardware without causing any stalls for all eight compression dictionary addresses (see Fig. 4).

The content of this chapter is based on the following paper (which has been cited 2 times): **Beneš T. and Bartík, M. and Kubalík, P.**, *"Design of a High-Throughput Match Search Unit for Lossless Compression Algorithms"*, The 9th IEEE Annual Computing and Communication Workshop and Conference (CCWC), ISBN 978-1-7281-0554-3, pp. 732-738, Las Vegas, USA, 2019 [A.4].

Contributions of Tomáš Beneš are the implementation, simulation flow, and obtained experimental results.

Design of a High-Throughput Match Search Unit for Lossless Compression Algorithms

Matěj Bartík
DDD & Department 706
CTU FIT & CESNET a.l.e.
Prague, Czech republic
matej.bartik@fit.cvut.cz

Tomáš Beneš
DDD
CTU FIT
Prague, Czech republic
benesto3@fit.cvut.cz

Pavel Kubalík
DDD
CTU FIT
Prague, Czech republic
pavel.kubalik@fit.cvut.cz

***Abstract*—This paper presents an attempt to combine recent research in fields of hardware- and software-based high-throughput universal lossless compression algorithms and their implementations, resulting into a case study focusing on one of the most critical parts of compression algorithms – a Match Search Unit (MSU) and its parallelization. The presented FPGA design combines ideas of the LZ4 algorithm (which is derived from the most common LZ77) with the state of the art hardware architectures for lossless compression also based on LZ77. This approach might lead to a smaller, better organized or more efficient "building block" for modern implementations of hardware driven lossless compression algorithms. The presented design focuses on optimization of the main problem of the LZ77 family, namely the construction of and searching in a compression dictionary. Particularly, we combine a Live Value Table (LVT) with multi-ported memory in order to improve the bandwidth of the dictionary and the Fibonacci hashing principle originating from LZ4 algorithm to decrease latency of the MSU and to achieve overall higher throughput rate. For the design synthesis an FPGA of the Xilinx Virtex-7 family was used.**

***Index Terms*—FPGA, high, bandwidth, fast, lossless, compression, algorithm, architecture, LZ4, LZ77, hash, table, LVT, multiport, memory, Xilinx, Virtex**

I. Introduction

In recent decades throughput of lossless compression systems has increased by an order of magnitude [1]. Further progress requires new and more complex architectures. To increase performance and decrease consumption of FPGA logic resources opens the possibility of accelerators with higher density and frequency on a single chip.

There are many applications of compression algorithms. The architectures used in those applications are influenced by the required

- throughput,
- latency and latency guaranties,
- compression ratio,
- assumed corpus, and
- implementation environment with its resources metrics.

All these requirements can be present as hard constraints, or optimization goals. Consequently, actual implementations targeted at different requirements are hard if not impossible to compare.

978-1-7281-0554-3/19/$31.00©2019 IEEE

Unlike previous efforts, our target is lossless compression as a way to increase network throughput, without any specific traffic in mind. Existing architectures [3]–[6] aim mainly for throughput in different contexts. In our case, the current emphasis on low network latency [2] puts strict limitations on compression latency, which differs our architecture from others.

Network traffic is inversely proportional to compression ratio without any overhead taken into account. Therefore, compression ratio is rarely a hard requirement.

Originally, compression algorithms were designed to compress human-readable text. This is, for a long time, not the only case. For network traffic, the actual mix of text, binary files, video etc. is unknown. The Silesia Corpus [30] seems to the closest to the assumed traffic.

The rest of the paper is structured as follows. In Section II we summarize relevant previous works and we analyze key features of lossless compression algorithms, their software implementations and current FPGA architectures, which may have influence on compression performance. In Section III we propose an optimized FPGA design of a Match Search Unit. Performance measurements of this design are presented in Section IV. Finally, we propose further possible improvements in Section V.

II. State of the Art & Analysis

The era of an universal lossless compression had begun in 1977 with the introduction of the first Lempel-Ziv algorithm known as the LZ77 [7]. Principally, the algorithm tries to match a current sequence with sequences in the past text kept in a sliding window. When a match is found, it is encoded in a standardized format. All LZ77 variants process data by blocks, so that compression of different blocks is independent. The abstract atomic parts of blocks, called literals in the standards, are normally bytes. A simple non-parallel architecture [15] served as the starting point of the analysis.

Several architectures of an LZ77-based compression scheme in FPGA have been recently presented [3]–[6] and focus on real-time lossless compression of IP (Internet Protocol) packets at 10-40 Gbps throughput.

Match search and encoding are two nearly independent parts of an algorithm of the LZ77 family. As they both

influence compression parameters, they offer opportunities to tune the algorithm to requirements. For example, the popular DEFLATE algorithm [9] combines LZ77 match search with Huffman Encoding [8]. Variants which aim at high compression and especially decompression speed, are LZO [10], [13] or LZ4 [11], [14]. Their encoding is adapted to efficient processing in software, at the cost of encoded sequence length.

LZ4 was chosen as a candidate for our design following a full analysis of LZ4 [14] from a hardware designer's perspective. The advantageous characteristics of LZ4 include the highest (de-)compression speed among other lossless algorithms, low latency, resource efficiency and easy implementation in an FPGA.

The specification of LZ4 deals mainly with the output format, which all decompressors must understand. It explicitly states that it is independent of match search algorithms [11]. Some searches require the entire blocks to be present and parsed, and perform iterative passes through the compressed sequences. Such searches are typical for the High Compression (LZ4-HC) category.

It is possible to search for matches in a single pass, with much less effort but lower compression ratio (for an informal explanation, see [12]). This is the Fast Compression (LZ4-FC) category. This category is interesting for hardware low-latency implementations because encoding latency can be lower and constant.

Single-pass match searches usually keeps sequences of given minimum length in a table, usually called the dictionary. In the case of LZ4, the minimum length is, due to the output format, 4 bytes. When a match is longer than the minimum length, the algorithm just accumulates the match length during input scan and outputs it at the end of the match.

In hardware, the two parts of the algorithm are implemented as Match Search Unit (MSU) and Encoding Unit (EU). MSU is usually the most complex part of compression device with highest impact on its speed. Therefore, we focus entirely on this unit in the paper.

A. Match Search Unit Architecture and Performance

The data flow of an LZ4 MSU based on hashed dictionary is in Fig. 1, also implemented in [15]. We see that the critical path goes through all blocks. It includes two reads from the input buffer, hash computation, a read and a write from the hash table, and a data comparison. The data flow therefore offers no opportunity for parallelization at the level of a single MSU.

Some authors (e.g., [4] but also [15]) parallelize compression at the block level. This architecture can accommodate multi-pass searches. Assuming n copies and block-level parallelism, however, we obtain latency of n block transfer times. Therefore, block-level parallelism is not ideal for low-latency applications.

We employed parallelism at the MSU level and single-pass match search. We let multiple MSUs to search for matches on data that are consecutively offset by 1 byte, and then let the encoding unit to compose final matches from partial results.

In such an architecture, the MSUs are independent with the exception of two points. First, the MSUs must be supplied by input data from the input buffer. Then, the table should be shared, because any MSU should find a sequence met by another MSU. The parallel access to the shared dictionary introduces possible conflicts between writes, which need to be resolved by implementing priority access.

B. Hash Table

For a single-pass compression algorithm, the theoretical table size is the algorithm's window size, with each item holding a sequence of the minimum length. Such a table would be implemented e.g. as a CAM. Then, each match present in the input sequence would be detected and the algorithm would achieve its best compression ratio.

Such a table is rarely practical. Most designs, e.g., [3], [4], [14], [15] try to avoid CAMs for cost and area reasons. A fast search scheme is required, which is solved using a hashing function. Hashing function leading to non-optimal use of the table space. Smaller or non-optimally used table causes loss of matches and worse compression ratio. Therefore, we can identify two sources of compression ratio loss in the case of hashed table implementation, for a particular algorithm:

1) the table size used, and
2) the hash function employed, i.e. more collisions than necessary in a table of the given size.

Their effect is hard to separate, which makes it difficult to learn from other sources. Therefore, we performed experiments with software model described in Section IV. We evaluated the algorithm actually employed, with actual hashing function on three corpuses (Calgary [28], Canterbury [29], Silesia [30]). From Tab. I we see that in order to improve the compression ratio by a constant factor, the table size has to grow almost exponentially. In the end, we choose the size of 256 (marked with an asterisk) as a compromise with respect to the intended application.

TABLE I
COMPRESSION RATIOS FOR DIFFERENT HT SIZES

HT Size	Calgary	Canterbury	Silesia
64	1.28	1.40	1.24
128	1.34	1.46	1.28
***256**	1.38	1.5	1.31
512	1.39	1.57	1.32
1024	1.47	1.68	1.4
2048	1.54	1.75	1.48
4096	1.61	1.81	1.56

Non-optimal table usage caused by hash function can be overcome by rehashing, which is common in software implementation. It can be used in hardware designs [15] as well. However, its latency is larger and, more importantly, not uniform. Therefore, rehashing is not used in low-latency and high-throughput compression designs. The penalty on compression ratio is usually acceptable.

As we cannot assume any particular composition of the input data, we have to use a universal hash function. From

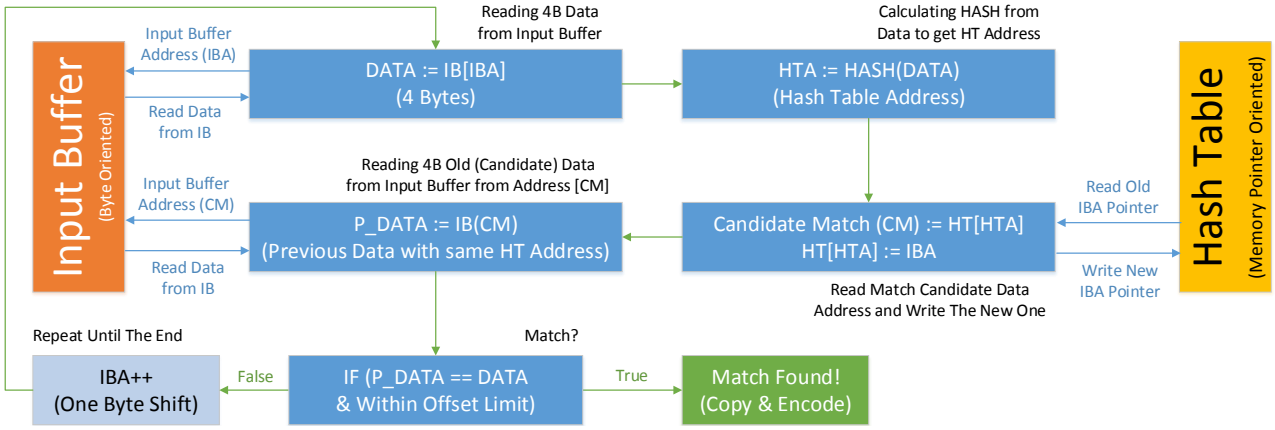


Fig. 1. Data flow of the original LZ4 MSU [14], [15].

the known functions, multiplicative hashes [16] have the simplest hardware implementation, especially when the table size is a power of two. Fibonacci hashing [16] uses simple multiplication with a constant, which is the closest possible prime number to the golden ratio.

As discussed in Section III-C, the hash table should be shared between all MSUs working in parallel. Their access to the table has no regular pattern, so that the table must be multiported. Concurrent writes to a single item cannot be avoided. Luckily, the MSUs are naturally ordered by the ordering of data they process. Hence, the MSU having the earliest match has the best priority, without any loss of compression quality.

C. Multiport Memory in FPGAs

As discussed above, multiport memory is essential for implementation of a shared dictionary. The Live Value Table (LVT) [18] is the most common approach for creating a multiport memory from an ordinary single port (or dual port) memory blocks, such as Xilinx BRAM or Altera M9K blocks.

There is also an alternative approach using an XOR technique [26], [27] for accessing the latest value. The main advantage of this method is reducing the number of logic elements and increasing speed for low-depth memories. However, for larger memories, this method results in lower design frequency and thus is not well-suited for high throughput designs.

Before the introduction of the LVT, multiport memory was implemented usually from registers and general reconfigurable logic. There are no hard limits in capacity and the number of ports. However, it does not scale well. The area and clock period grow rapidly especially with the number of ports.

The idea of the LVT is to divide such a memory design using general re-configurable logic into two parts:

1) Data memory using single or dual port embedded FPGA memory blocks (BRAMs) where the number of read-/write ports can be increased by replication, banking and multipumping [18].
2) Control memory (created from general logic) keeps for each written address the information in which part of the data memory is the latest value stored. With each read operation, this information is used to set output multiplexers to the location of the latest value written for the given address.

III. Our Approach

We used the architecture with independent MSUs and shared dictionary to build a low-latency high-throughput compression core in an FPGA. It is intended to cooperate with common 10G Ethernet IP cores that use a 64-bit data path clocked at 156.25 MHz [19]. The selected platform was a member of the Xilinx 7-Series family, however, the described architecture can be easily implemented in any modern FPGA. The selected block size is 9 kB, required by the application.

With 8-bit literals and 64 bit datapath width, we need 8 MSUs in the design. As discussed in Section III-A, the input buffer is a 16 kB dual port memory with a 64-bit write port (from the Ethernet core) and a 128 bit read port (to the MSUs). The shared dictionary is a LVT-based memory of 256 items, with 8 read ports and 8 write ports. Each item contains an input buffer pointer (up to 16 bits) and a 4-byte sequence (Section III-C). The control memory of the LVT scheme is also used for validity flags [14], so that the dictionary needs not to be cleared between input data blocks (Section III-B).

The optimized MSU follows the data flow in Fig. 1. One read access to the input buffer is saved as the data are read from the dictionary (Section III-C). Hashing is performed in DSP blocks. The optimized data path and latencies are in Fig. 4.

A. Input Buffer (IB)

For each 4-byte block of memory that is tested for a match, we need to perform two reads from the input buffer. One read provides the data to search in the dictionary, the other read verifies that the data at the pointer found in the dictionary agree and that a hash collision did not take place.

Usually, the data width of one read operation would be 256 bits for 8 MSUs, where each is capable of processing 32 bits. In our case, however, the actual data flow is much smaller – 88 bits, because of the data overlap (see Figure 2). There are eight 4-byte sequences with 1-byte overlap. The nearest

suitable port size of a double-port memory is 128 bit. Only one *Input Buffer Address (IBA)* for reading is required, however, the memory must be able to read at 64 bit boundaries.

B. Hash Table (HT)

The major issue is the parallel access to the *HT* (Hash Table) representing a dictionary in this particular case and the following access operations to *IB* (readback of candidate match positions). Data are stored in the *HT* at nearly random positions – they are no longer consecutive as the data before the hashing phase. Therefore all read/write operations in *HT* are accessing random addresses.

We decided to implement a shared dictionary (unlike 842B [4] technique or any multibank-based principle), where all MSUs have access to a single dictionary that behaves as a single memory. Without such a dictionary, the compression ratio would be significantly worse, because matching would occur only between blocks processed by the same *IBA* pointer. This decision requires to implement a multiport memory using the LVT [18] technique.

The MSU calculates *Hash Table Address (HTA)* from data at *IBA*, reads the old *IBA* at the address and stores a new one in its place. The FPGA embedded block memory can perform both operations as a single transaction in a single clock cycle. The data at the old *IBA* must be verified, because a hash collision can occur and a false match can result. This doubles the number of reads from the input buffer.

Our idea is to reduce the number of *IBA* accesses by merging the processed data and the *IBA* pointer into a single record inside the Hash Table (see Fig. 3). This approach removes the need for the following read accesses from *IB*. The *HT* thus becomes the only component where the number of ports scales with the number of MSUs. The resulting architecture is simplified because only one LVT based memory is required.

1) Hash Calculation & Pipelining: The last important part required for the MSU architecture is the hash calculation block. Fibonacci hashing is used, the input value is multiplied by a constant and higher bits are selected to create the *Hash Table Address (HTA)* [14].

The Xilinx DSP48 block can be used for the hash calculation. The optimal settings (recommended by the Xilinx CoreGen tool) for multiplication of two 32-bit numbers are: four DSP48 blocks and the calculation requires six clock cycles (estimation made the CoreGen) thus the block will be able to operate at approximately 700 MHz in case of a Virtex-7 chip [20]. Pipelining is required to mask the computation latency and to maximize the throughput. We can also use higher frequency for the DSP48 blocks than for the rest of the design, thus reducing the length of the hash calculation pipeline for to 3 or 2 cycles.

C. MSU Optimized Architecture

The FPGA design expects 8 pairs of *IBA* pointers and the related data (32-bit long sequences), both provided by the *IB*. These pairs are pipelined along a hash calculation block, where 8 *HTA*s are calculated. The length of the pipeline must be the same as the latency of the hash calculation blocks which are clocked at a higher frequency to decrease the latency. These pairs will be read and written to the locations specified by *HTA*s in the *HT* with the latency of one clock cycle. Each pair is also pipelined along the *HT* representing the compression algorithm dictionary.

The previous data (*P_DATA*) are extracted from a candidate match (*CM*) pair, which is read from the *HT*. If the previous data are equal to the current data which is read from the pipeline, a match occurs. Both previous and current IBAs will become a match. The last step is to resolve possible multiple matches in a single clock to a single match via multiplexers controlled by a priority encoder (the lower the value of *IBA*, the higher the priority).

IV. Results

In this section we discuss measurement setup, resources usage of our architecture, analysis of achievable compress ration and latency influence.

A. Measurement Setup & Synthesis Results

The Xilinx ISE 14.7 toolkit was used to synthesize the presented architecture for the Xilinx Virtex-7 XC7V330T-2FFG1157 chip. For the synthesis, the optimizations were adjusted to *Speed/High*. The FPGA resource utilization for the routed design is summarized in Table II. We used the random FPGA pin placement feature available in the Xilinx ISE toolkit to get a fully routed design. Thus, we were able to measure the design speed in a more realistic way than with a partially routed design.

TABLE II
LZ4 MSU resources utilization for Virtex-7 XC7V330T

Slice	LUT	Flip Flop	BRAM	DSP48	Frequency
4828	15014	8530	64	32	250 MHz

B. Resource Usage Comparison

We know that the Xilinx 7-Series logic *SLICE* block contains four LUT6 (6-input Look Up Table) blocks plus eight flip-flops [21]. We also know that the Altera Stratix V ALM contains two LUT6 and 4 flip-flops [23], thus a single Xilinx Slice can be considered equal to two Altera ALMs.

This simplification allows us to compare our architecture to the previous work [3]. The *PWS=8 w/ HT* (Parallelization Window Size) variant can process 8 bytes per a clock cycle, therefore we have selected this variant for a brief comparison (see Tab. III). Our architecture excludes the input and output buffers and the output sequence encoding part. However, the amount of FPGA logic resources are comparable (PWS=8 w/ HT variant has 16519 ALMs compared to 9656 ALMs) from perspective of the used FPGAs.

The overall latency of the [3] is 41 clock cycles. The parts of the architecture [3] which are implemented also in our approach include: a hash calculation, a hash table update, a string match and a match selection. The latency of these

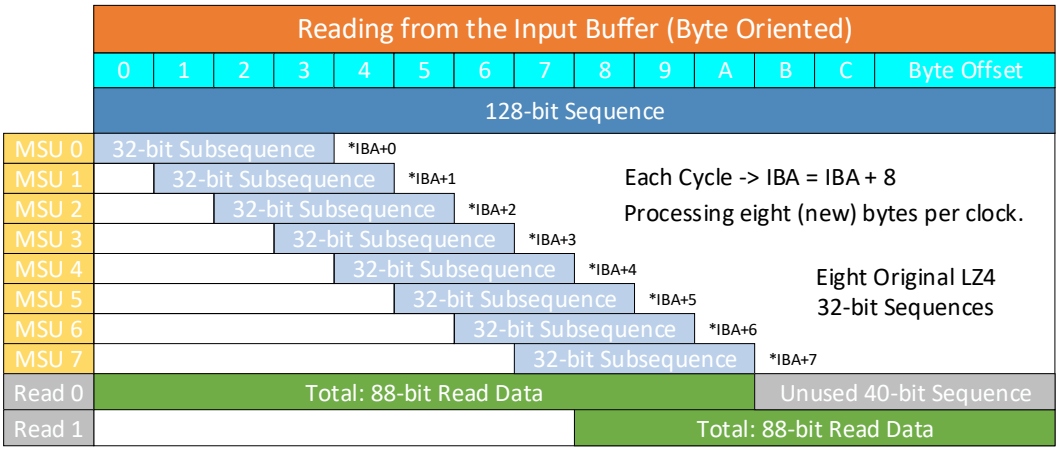


Fig. 2. Hardware optimized LZ4 memory read schema

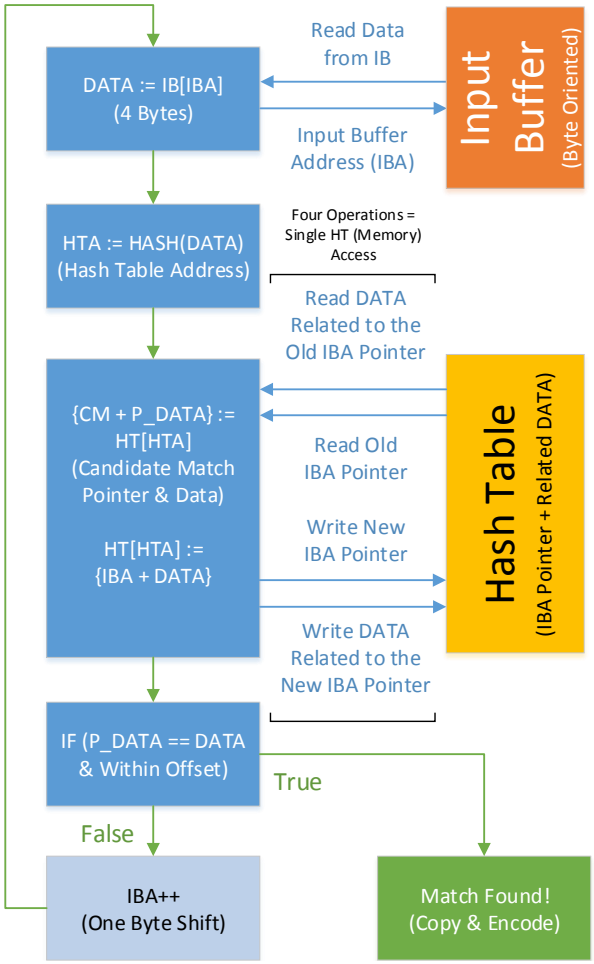


Fig. 3. Flow of the (memory access) optimized LZ4 MSU

selected parts was originally 29 clock cycles, whereas our approach has the latency of 7 clock cycles only, thus the latency has been decreased approximately four times.

C. Compression Ratio Analysis and Simulation

We developed a functionally equivalent software model of our hardware architecture. This software model was connected to an existing software LZ4 architecture re-using other parts of LZ4 (input buffer, output buffer and the LZ4 encoding algorithm). The model served two purposes.

First, we used it for functional verification of the design. On random data, the results of the model and traces from hardware simulation had to agree. Using software decompression, we excluded errors common in hardware and the software model.

Second, we assume that the compression ratio of the software model equals to the presented hardware architecture. The software model uses identical dictionary size and identical hash function. Also the process of output encoding is the same and cannot affect the compression ratio (see Fig. 5).

We processed three compression corpuses (Calgary [28], Canterbury [29], Silesia [30]) through our (hardware based) software model of LZ4 to obtain experimental results. The compression ratio is scaling up in an expected way in relation to the size of the hash table (see Tab. I).

The LZ4 8-bit [15] and Software LZ4 use advanced collision handling and have better compression ratio. This, however, comes at a cost. We assume that the LZ4 8-bit architecture uses 24-bit wide address of the input buffer, thus 32 or 64 BRAMs are used for implementation of the dictionary (the most feasible configurations). All remaining BRAMs realize the input and the output buffer. The worst case of 32 BRAMs is representing the dictionary size of 65536 entries [22] compared to our 256 entries. Furthermore, these designs cannot guarantee their worst case throughput. LZ4 8-bit [15] is only providing peak performance throughput, which is derived from processing 8 bytes per clock. We present an MSU with worst case performance, which is an essential characteristic of a stable high-throughput design.

TABLE III
BRIEF MSU PERFORMANCE COMPARISON

Solution	ALMs	Latency [Cycles]	Throughput [Gbps]
Our LZ4 MSU	9656	7	16,0
PWS=8 w/HT [3]	16519	29 (41)	11,2
LZ4 ASIC [17]	N/A	17	4,0
LZ4 8-Bit [15]	690	N/A	2,0

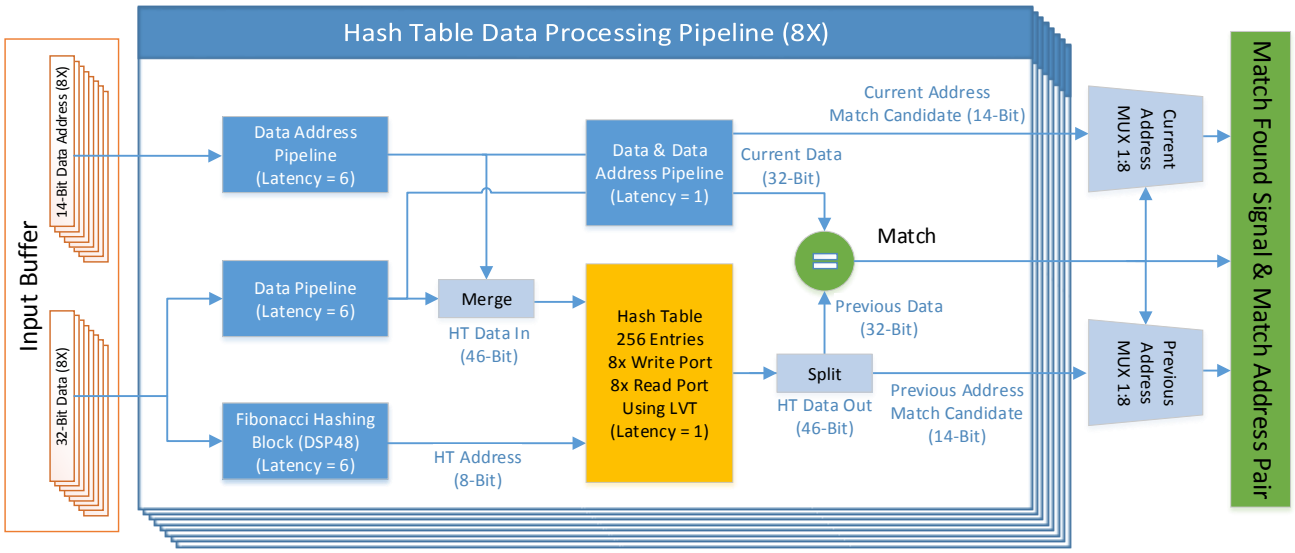


Fig. 4. Architecture of the new MSU inspired by the (hardware) optimized LZ4 flow

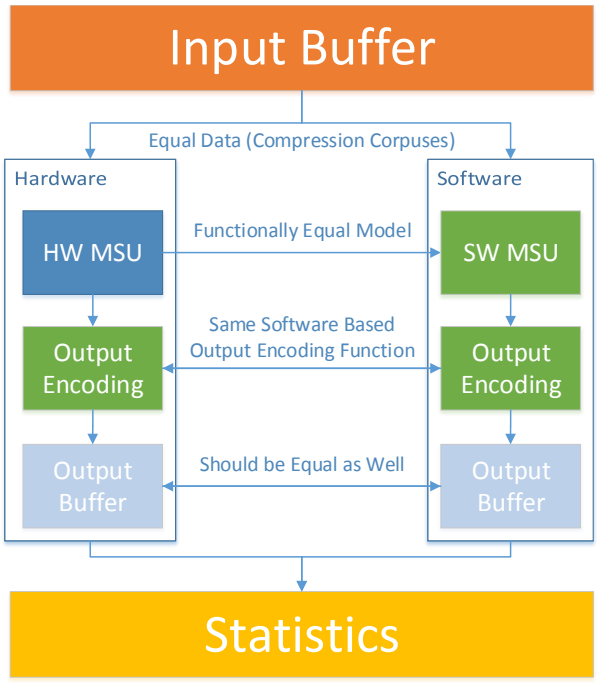


Fig. 5. Experimental simulation flow to obtain compression ratios.

D. Influence of the Latency

An MSU (and overall system) latency affects the system readiness to accept new data. Lower latency also allows us to "squeeze" more data into constant throughput media by lowering an inter-frame gap, in a packet oriented real-time systems (with packets processed one by one, not in a continuously streamed manner), in case that more parallel blocks are used at the same time.

All MSU architectures designed for high throughput applications use pipelining principle where data are processed alongside control signal or other data (for example hash calculations or matching) in several stages. The latency reduction might enable the number of pipeline stages to decrease resulting in lower usage of logic resources.

TABLE IV
BRIEF COMPRESSION RATIO COMPARISON

Solution	Calgary	Canterbury	Silesia
Our LZ4 MSU	1,38	1,5	1,31
PWS=8 w/HT [3]	1,82	N/A	N/A
LZ4 ASIC [17]	N/A	N/A	N/A
LZ4 8-Bit [15]	Incomplete (1,65 – 2,05)		
Ref SW LZ4	2,26	2,11	2,41

V. FUTURE WORK

Several optimizations of the presented architecture are still possible. The first idea is to design an optimized hash calculation unit which by itself is currently capable of running at approximately 700 MHz [24] with the latency of 6 clock cycles, whereas the current design can run up to 250 MHz (and cannot be further significantly increased). The architecture of a DSP48 block allows to bypass some pipelining stages to limit the latency, thus limiting the frequency. We can reduce the latency to just 3 clock cycles while matching the frequency of DSP48 blocks to the rest of the MSU design. This way, we can save additional resources in *Data* and *Data Address* pipelines (see Fig. 4). The reduction in this particular case will be 50% of flip-flops required for these pipelines.

We can also apply similar principle to an embedded memory block (also capable of running at approx. 600 MHz [24]) to implement the multipumping principle [18] in order to save resources (BRAMs) required for the LVT based hash table. However the multipumping principle will increase the amount

of other used FPGA resources. Further optimizations of the LVT principle will be also explored [25].

We started implementing a system realizing the LZ4 lossless compression algorithm with use of the presented MSU architecture with the aim of minimum 10 Gbps throughput.

VI. Conclusion

We presented an architecture of a Match Search Unit (MSU) inspired by a modern fast LZ4 lossless compression algorithm suitable for hardware implementations. We proposed optimizations of the original LZ4 data flow for reducing the memory read/write accesses towards the implementation platform (Xilinx Virtex-7 FPGA logic). Match combination in the Encoding Unit made the MSUs independent, which eased their parallel operation.

The latency of the presented MSU solution has been reduced 4 times compared to the previous work [3], while the amount of FPGA logic resources is comparable. In contrast to [15] our MSU architecture guarantees minimal throughput, has substantially higher throughput rate with comparable amount of FPGA resources and achieves slightly lower compression ratio with the use of significantly smaller dictionary size (just 256 entries against 65536).

The design has the potential for further optimizations in order to significantly reduce the overall latency and resource consumption. For the current implementation, the theoretical throughput is 16 Gbps.

Acknowledgment

This research has been partially supported by the CTU project SGS17/017/OHK3/1T/18 "Dependable and attack-resistant architectures for programmable devices" and by the project "E-infrastructure CESNET – modernization" no. CZ.02.1.01/0.0/0.0/16 013/0001797.

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2008–2013," [Online] https://tinyurl.com/yb6wf4k3 [Accessed 15 November 2018]

[2] CESNET, "Special video transmission," 2017. [Online] Available: https://www.cesnet.cz/services/special-video-transmissions/?lang=en [Accessed 16 November 2018]

[3] J. Fowers, J. Y. Kim, D. Burger and S. Hauck, "A scalable high-bandwidth Architecture for lossless compression on FPGAs," 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, 2015, pp. 52-59. doi: 10.1109/FCCM.2015.46

[4] B. Sukhwani, B. Abali, B. Brezzo and S. Asaad, "High-throughput, lossless data compresion on FPGAs," 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, Salt Lake City, UT, 2011, pp. 113-116. doi: 10.1109/FCCM.2011.56

[5] R. Mehboob, S. A. Khan, Z. Ahmed, H. Jamal and M. Shahbaz, "Multigig lossless data compression device," in IEEE Transactions on Consumer Electronics, vol. 56, no. 3, pp. 1927-1932, Aug. 2010. doi: 10.1109/TCE.2010.5606348

[6] K. Papadopoulos and I. Papaefstathiou, "Titan-R: A reconfigurable hardware implementation of a high-speed compressor," 2008 16th International Symposium on Field-Programmable Custom Computing Machines, Palo Alto, CA, 2008, pp. 216-225. doi: 10.1109/FCCM.2008.14

[7] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," in IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, May 1977. doi: 10.1109/TIT.1977.1055714

[8] D. A. Huffman, "A method for the construction of minimum-redundancy Codes," in Proceedings of the IRE, vol. 40, no. 9, pp. 1098-1101, Sept. 1952. doi: 10.1109/JRPROC.1952.273898

[9] D. Harnik, E. Khaitzin, D. Sotnikov and S. Taharlev, "A fast implementation of Deflate," 2014 Data Compression Conference, Snowbird, UT, 2014, pp. 223-232. doi: 10.1109/DCC.2014.66

[10] M.F.X.J Oberhumer, "LZO real-time data compression library"", 2011. [Online]. Available: http://www.oberhumer.com/opensource/lzo/ [Accessed 1 Dec 2018]

[11] Y. Collet, "Real Time Data Compression: LZ4 Explained", 2011 [Online]. Available: http://fastcompression.blogspot.ru/2011/05/lz4-explained.html [Accessed 1 October 2018]

[12] Ticki, "How LZ4 works". [Online]. Available: http://ticki.github.io/blog/how-lz4-works/ [Accessed 1 Dec 2018]

[13] J. Kane and Q. Yang, "Compression speed enhancements to LZO for multi-core systems," 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing, New York, NY, 2012, pp. 108-115. doi: 10.1109/SBAC-PAD.2012.29

[14] M. Bartík, S. Ubik and P. Kubalík, "LZ4 compression algorithm on FPGA," 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, 2015, pp. 179-182. doi: 10.1109/ICECS.2015.7440278

[15] W. Liu, F. Mei, C. Wang, M. O'Neill and E. E. Swartzlander, "Data compression device based on modified LZ4 algorithm," in IEEE Transactions on Consumer Electronics, vol. 64, no. 1, pp. 110-117, Feb. 2018. doi: 10.1109/TCE.2018.2810480

[16] E. D. Knuth, "The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching," 1998, ISBN 0-201-89685-0. Addison Wesley Longman Publishing Co., Inc.

[17] S.M. Lee, J.H. Jang, J.H. Oh, J.K. Kim and S.E. Lee, "Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression," IEICE Electronics Express, ISSN 1349-2543 , doi: 10.1587/elex.14.20170399

[18] C. LaForest and S. Gregory, "Efficient Multi-ported Memories for FPGAs," Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, 2011, pp. 41-50. doi: 10.1145/1723112.1723122

[19] Z. Bradac and S. Valach, "10G bit ethernet phy implementation in FPGA based systems," In IFAC Proceedings Volumes, Volume 39, Issue 21, 2006, pp. 427-432, ISSN 1474-6670. doi: 10.1016/S1474-6670(17)30224-0.

[20] 7 Series DSP48E1 Slice User Guide (UG479), Xilinx [Online]. Available: https://tinyurl.com/ybhx4r93 [Accessed 29th October 2018]

[21] 7 Series FPGAs Configurable Logic Block User Guide (UG474), Xilinx [Online]. Available: https://tinyurl.com/xug474 [Accessed 29th October 2018]

[22] 7 Series FPGAs Memory Resources (UG473), Xilinx [Online]. Available: https://tinyurl.com/y75gctmw [Accessed 29th October 2018]

[23] Stratix V Device Handbook (SV-5V1 2017.12.15), Altera [Online]. Available: https://tinyurl.com/stx5-alm [Accessed 29th October 2018]

[24] Virtex-7 T and XT FPGAs Data Sheet – DC and AC Switching Characteristics (DS183), Xilinx [Online]. Available: https://tinyurl.com/v7-ds183 [Accessed 29th October 2018]

[25] M. Bartík, S. Ubik and P. Kubalík, "A novel and efficient method to initialize FPGA embedded memory content in asymptotically constant time," 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2016, pp. 1-6. doi: 10.1109/ReConFig.2016.7857146

[26] C.E. Laforest, Z. Li, T. O'rourke, M.G. Liu and J.G. Steffan, "Composing multi-ported memories on fpgas," ACM Trans. Reconfigurable Technol. Syst., Vol. 7 Issue 3, September 2014. doi: 10.1145/2629629

[27] A. Abdelhadi and G. G.F. Lemieux. "Modular multi-ported sram-based memories," In Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays , FPGA'14, pages 35-44, New York, NY, USA, 2014. ACM. doi: 10.1145/2554688.2554773

[28] T.C. Bell, I.H. Witten and J.G. Cleary, "Modeling for text compression," Computing Surveys 21(4): 557-591; December 1989; ISSN: 0360-0300. doi: 10.1145/76894.76896

[29] R. Arnold and T. Bell, "A corpus for the evaluation of lossless compression algorithms," Proceedings DCC '97. Data Compression Conference, Snowbird, UT, USA, 1997, pp. 201-210. doi: 10.1109/DCC.1997.582019

[30] S. Deorowicz, "Silesia Corpus." Silesian University of Technology, 2003 [Online] Available: http://sun.aei.polsl.pl/ sdeor/index.php?page=silesia [Accessed 12 November 2018]

Chapter 5

High Throughput and Low Latency LZ4 Compressor on FPGA

This chapter is based on the information and ideas provided by the two previous chapters, which resulted in a high-throughput and low-latency FPGA implementation of the LZ4 compression algorithm. The implementation uses the architecture of the "Match Search Unit" and is capable of reaching a throughput of 6 Gbps (evaluated on Calgary [26], Canterbury [27], and Silesia [28] corpora) in the worst case scenario (very short matches of several bytes). The actual peak throughput could vary between 10 Gbps (no match was found) and 20 Gbps (very long repetitive data sequences).

The throughput and latency performances are directly proportional to an average length of discovered matches. The implementation latency can be expressed as a function related to the input data size (see Table 5.1).

Latency [clock cycles]	
Start-Up	Overall
26	$26 + \frac{\text{Data Size}}{8 \text{ Bytes}} \times \frac{10 \text{ Gbps}}{6 \text{ Gbps}}$
to first output data word	*to the last output data word*

Table 5.1: The latency of the presented "High Throughput and Low Latency LZ4 Compressor" FPGA implementation.

In the case of MVTP uncompressed mode, each processed pixel line is transmitted as a single IP (jumbo) packet with a maximum payload size of 9000 bytes. Thus, the required (worst case) latency for each packet can be calculated (by using Formula 5.1) for the assumed frequency of 156.25 MHz (the respective period is 6.4 ns). Therefore, the resulting worst-case latency is just 12.17 µs.

$$Latency_{MAX} = \left(26 + \frac{9000}{8} \times \frac{10}{6}\right) \times 6.4 \text{ ns} = 12166,4 \text{ ns} \approx 12.17 \ \mu s \tag{5.1}$$

Presenting a worst-case performance is a major difference compared to other hardware implementations (and respective papers) where a peak (maximum) throughput is being presented instead. The situation with the latency is quite similar. The lower limit for the latency of an average LZ4 hardware implementation (8-bit datapath running at 200 MHz) is 45 µs (see Formula 5.2), which is approximately 3.7 times worse than the presented architecture (12.17 µs). However, the real difference will be much higher, because the comparison was made between the worst case and the best case latency for both 64-bit and 8-bit architectures.

$$Latency_{MIN} = 9000 \times 5 \text{ ns} = 45000 \text{ ns} \approx 45 \ \mu s \tag{5.2}$$

The primary invention used in the architecture of "High Throughput and Low Latency LZ4 Compressor" is to use the entire memory bandwidth, which wasn't fully utilized by the "Match Search Unit". The "Match Search Unit" processes only 64-bit of new data per clock cycle. To process 64-bit of new data, a memory read of 88-bits is required, thus leaving 40-bit unused of the 128-bit memory bus.

Therefore I propose to implement an input (and output) buffer with a wider memory bus than the minimum required memory bus width. The extra memory bandwidth can be used to mask "Stalls" which occur in a pipeline during data processing. In this particular case, the double width memory bus (potentially resulting in a 2x speed-up) is used to mask a "start-up" latency for "Match Length Finding" and "Output Encoding" phases.

The content of this chapter is based on the following paper:
Beneš T. and Bartík, M. and Kubalík, P., *"High Throughput and Low Latency LZ4 Compressor on FPGA"*, The 9th International Conference on ReConFigurable Computing and FPGAs, ISBN 978-1-7281-1957-1, pp. 1–5, Cancún, Mexico, 2019 [A.5].

The respective paper is based on the diploma thesis (which I supervised) of Tomáš Beneš, *"High throughput FPGA implementation of LZ4 algorithm"*, Czech Technical University, Prague, 2019. The diploma thesis has been cited once.

Contributions of Tomáš Beneš are: the implementation, the concept of storing matches (and their respective lengths) in FIFOs, and the obtained experimental results. The authorship of the "Match Length Finder" architecture is split in half between Tomáš Beneš and me.

High Throughput and Low Latency LZ4 Compressor on FPGA

Tomáš Beneš
CTU FIT & CESNET a.l.e.
benesto3@fit.cvut.cz

Matěj Bartík
CTU FIT & CESNET a.l.e.
bartik@cesnet.cz

Pavel Kubalík
CTU FIT
pavel.kubalik@fit.cvut.cz

***Abstract*—This paper presents an FPGA design implementing a single LZ4 lossless compression IP block, providing a throughput of 6 Gbps combined with extremely low latency, while still retaining full binary compatibility with the original LZ4 format. The best-known competitor is capable of processing up to 2 Gbps per block/engine with unknown latency. The presented design uses two key features: a low-latency 8-way match search unit and consequently a match buffer which allows encoding LZ4 sequences independently to reduce stalls in the data processing pipeline. The design was evaluated on several compression corpora with an average compression ratio of 1.7.**

***Index Terms*—LZ4, LZ77, lossless, compression, FPGA, pipeline, multi-port, memory, match, buffer**

I. Introduction

Universal lossless compression algorithms have been used for decades for saving space on storage or to save bandwidth in communication networks. Most of these currently used compression algorithms are based on LZ77 [1] (and its successors such as DEFLATE [2] or LZW) including some modern "fast" [3] compression algorithms such as LZ4 [4]–[6] or LZO [7] which trade the compression speed for the compression ratio. The mentioned algorithms were originally implemented in software, but as the CPU speed became a bottleneck, the more recent research focuses on accelerating them using (programmable) hardware such as FPGAs.

Hardware implementations of LZ77/DEFLATE can reach the compression speed of 100 Gbps [8]–[10], however representative examples of these implementations [11], [12] are using very small dictionaries to reduce the design complexity. The throughput of these LZ77 based implementations [9]–[12] cannot be compared directly to the LZ4 implementation regarding the fact that the LZ4 format is not suitable for implementing in hardware [5]. Several hardware implementations of LZ4 exist, with a throughput of 2 Gbps [5] per engine (0.5 Gbps respectively [6]).

In this paper, we would like to focus on designing an engine implementing the LZ4 compression suitable for relatively large data blocks, full binary compatibility [13] with LZ4 (unlike LZ4-Modified [5] variant), and low-latency.

II. Motivation

The latest papers describing a hardware implementation of a compression algorithm are using two common terms: the latency and the throughput. It might seem that these terms have a similar meaning, but it cannot be further from the truth. The throughput is usually defined as an amount of data processed over a specified time (seconds). However, the high (peak) throughput does not guarantee a low latency of the processed data. For example, high throughput is reached by implementing a many-core architecture where each core (compression engine) can be particularly slow.

On the other hand, the latency is usually defined as the time required to transform the input data by an algorithm (or a communication media) into the output data. The throughput of the implemented system could be considered to be independent of the system's latency. A particular example of high-throughput and high-latency system is a hardware implementation which uses an extreme case of the superpipelining principle. Therefore, each additional stage increases the system latency. The pipeline principle also increases the risk of stalls due to the need of "flushing" old data out of a datapath. The process of loading new data into a deep(er) pipeline increases the latency, thus lowering the actual throughput.

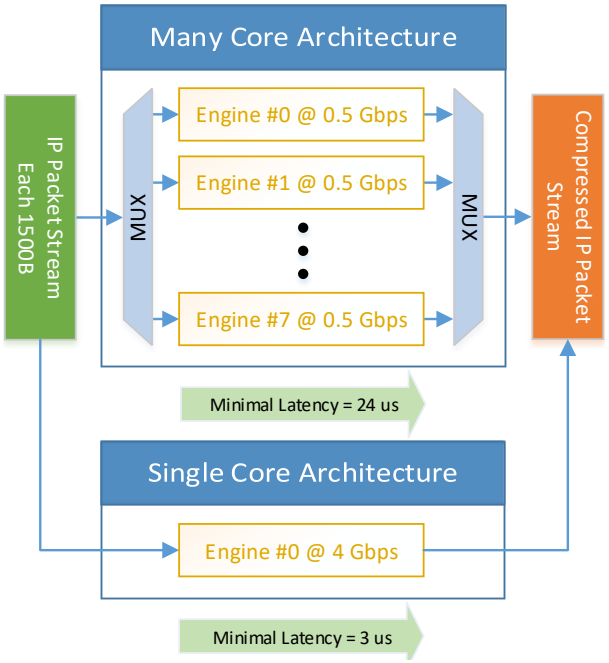


Fig. 1. Two possible attitudes for implementing a high-throughput architecture with different latency requirements.

In contrast to previous efforts, our target is a lossless compression as a way to increase the network throughput, without

978-1-7281-1957-1/19/$31.00 ©2019 IEEE

any specific traffic in mind. Existing architectures [9]–[12] mainly aim at the throughput in different contexts. In our case, the current emphasis on a low network latency [14] puts strict limitations on the compression latency (see Fig. 1), which differentiates our architecture from the others. Fig. 1 also explains the minimal latency required to process a standard IP packet. The network traffic is inversely proportional to the compression ratio, without any overhead taken into account. Therefore, the compression ratio is rarely a hard requirement.

III. State of the Art

There are several existing implementations [5], [6], [20], [21], which are considered the state of the art of LZ4 algorithm. None of them is suited for low-latency and high-throughput designs.

MLZ4C-1 and MLZ4C-2 [5] are single-core sequential architectures using a single dictionary with an 8-bit internal datapath only. These architectures are using the rehashing principle to deal with hash collisions; therefore their respective throughput heavily depends on the respective input data. Additionally, this implementation introduces a modified LZ4 format, which is not compatible with the reference LZ4 implementation/tools.

The HALZ4 [6] many-core architecture uses multiple independent dictionaries in each particular core to find multiple match candidates to select the best candidate. The architecture is mainly focused on ASICs and does not provide FPGA resource utilization for a direct comparison.

The Xilinx LZ4 [20] architecture is many-core architecture, which is using eight cores to achieve its 13.28 Gbps throughput overall. The architecture uses an 8-bit wide datapath. The design requires a lot of FPGA resources including memory blocks called UltraRAMs, which are available only in Xilinx's latest (and the most expensive) UltraScale+ FPGAs.

Comparing the compression ratio of given lossless compression implementations is done by comparing results from the same test data sets called *corpora*. These *corpora* sets usually contain a wide variety of input data, which should be relative to real-world usages because most of the compression implementations are data sensitive and their performance should not be evaluated in a narrow input data set. The most common compression *corpora* used by the community are Calgery [17], Canterbury [18] and Silesia [19].

A. LZ4 Analysis

The first step was to analyze the principles of LZ4 and how is LZ4 being implemented in hardware. We would like to summarize the fundamental properties and features of several existing LZ4 HW/FPGA implementations [5], [20], [21] which are affecting the overall performance the most.

LZ4 can be divided into four fundamental blocks [4]: an input buffer, a match search algorithm (unit) including a compression dictionary (usually implemented as a hash table), an output encoder, and an output buffer. The buffers are less complicated than the two remaining blocks; therefore, we will focus on the most performance affecting parts of LZ4.

1) Match Search Unit (MSU): The original LZ4 reference C code uses a hash table to implement a compression dictionary. The Fibonacci hashing principle is used due to its speed and low complexity. The main disadvantage of the method is that it cannot deal well with produced collisions. This problem can be solved by using a re-hashing principle (an input data will be hashed until a free entry in a hash table is found – increases latency).

The most common methods used in the recent implementations [5] are the original Fibonacci hashing principle combined with the rehashing principle. Thus the measured throughput cannot be guaranteed; therefore, re-hashing introduces additional stalls in the data processing.

Currently, there is state of the art MSU architecture [16] for lossless algorithms, which focuses on high throughput and low latency systems. The MSU uses parallelism with a shared dictionary to find matches. It is capable of reaching a 10G throughput and the latency of 8 clock cycles.

LZ4 MSU architecture [16] is reached by introducing parallelism (8x factor).

2) LZ4 Sequence Format and the Output Encoder: The LZ4 output format is called a sequence (see Fig. 2). The sequence does not seem to be complex; however a (hardware) problem arises due to the method chosen for encoding the match and the literal length. The LSIC (Linear Small Integer Code) [15] produces an output with variable word sizes, thus the encoder has to process the data first to estimate the length to generate the code word. This also introduces uncertainty in the number of clock cycles needed for encoding the sequence.

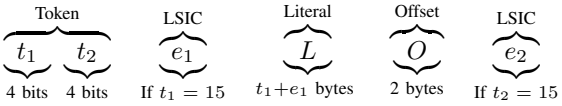


Fig. 2. LZ4 sequence structure. [15]

Another problem of the LZ4 output format is the number of items in the sequence. The format has three elements (Token, Literals, Match Offset), which are written usually in three different steps (e.g. clock cycles in hardware). This behavior causes an issue in the worst case for small length matches up to 12 bytes. These matches will be processed in three clock cycles, and they will process less than 24 bytes from the output buffer, which are required to guarantee the throughput of 10 Gbps. Fortunately, this is a very pessimistic worst-case scenario, which is very unlikely in real-world applications.

3) Binary (In-)Compatibility: Each block of the LZ4 format starts with the length of the block. This is the main latency limitation of the LZ4 format because the application has to wait until the compression of the whole block is finished before it can output the beginning of the block. Some implementations [5] are modifying the LZ4 format to avoid this inconvenience; however, they are no longer compatible with the official LZ4 tools available in modern PCs.

4) Usage of Trivial Parallelism: All current LZ4 hardware implementations are using the easiest way to increase the overall throughput – multiple independent engines (particularly

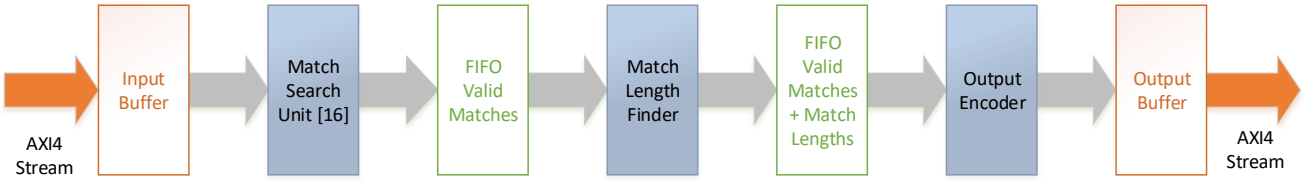


Fig. 3. The pipelined architecture of our LZ4 compression engine.

slow, typically with 8-bit datapath only) are used instead of a single engine (which has a significantly wider datapath). An active engine is being selected by the round-robin scheme usually.

The main advantage of the single-engine & high-throughput architecture is its low latency, which is crucial for use in low latency systems. It can also be used as a base of a multi-core architecture, which would maintain its low latency feature and also have the desired throughput.

B. Summary

We have analyzed several LZ4 HW/FPGA implementations, and we have found four fundamental properties which have a significant impact to overall to the system throughput and/or latency.

- High-throughput is reached by a trivial parallelism (many-core architecture). [20], [6]
- Implementations are not considering the low-latency feature as a factor (unlike us). [20], [5], [6]
- Re-hashing principle introduces stalls in data flow, thus it increases the latency. [5]
- Some of the implementations are not binary compatible with the official LZ4 format. [5]

IV. Our Approach

We decided to explore a non-trivial attitude to the parallelization, which is going to be the primary constraint and probably the best way to lower a system latency while the system throughput will remain the same or higher. We further explored ideas of the work [16], which describes a novel architecture suitable for high-throughput and low-latency LZ4 hardware implementation.

The high theoretical throughput and low latency (thanks to its short pipeline) of such LZ4 MSU architecture [16] is reached by introducing parallelism (8x factor) into the original LZ4 design [4]. Because the architecture has been already implemented, we decided to implement the remaining blocks like buffers and the output encoder.

The LZ4 format specifies the maximum amount of input data to 64 kB, which easily fits into the Level-1 cache of a modern processor. However, we decided to reduce the size of the buffers to 16 kB only; therefore, it is the nearest value related to the maximum payload size of an IP packet – 9000 bytes. The used MSU architecture [16] requires a buffer with a 128-bit read port interface. Reducing the buffer size also limits the theoretical maximum latency of the engine.

Our architecture consists of the *Input buffer, Match Search Unit, Match Length Finder, Encoder*, and *Output Buffer* blocks which are fully pipelined (see Fig. 3). Two FIFOs connect such blocks in the pipeline. These FIFOs allow the blocks to operate independently and continuously without any stalls or pipeline flushing. The system architecture uses the AXI4-Stream interface for the input and the output.

A. Match Searching Unit

The architecture of the used Match Searching Unit (MSU) was introduced in [16]. The main advantages of such MSU are the pipelined design, throughput of 10 Gbps, and latency of just 8 clock cycles. The MSU uses eight concurrent hashing units with a shared dictionary implemented in multiport memory.

The sequential principle used in [4] has to be modified to utilize the full potential of MSU. The main pitfall of the sequential principle is the processing of the found match, which modifies the match searching index accordingly. This process would require our approach to flush the pipeline of MSU, which would have a negative impact on the throughput and latency.

We have decided to process the entire input buffer by MSU and store the output into a pipelining FIFO. The full throughput and latency potential of MSU is utilized. MSU produces match candidate records which are guaranteed to have a minimum length of 4. MSU is not skipping bytes during the search to avoid pipeline flushing; therefore it produces match candidates which can overlap.

B. Match Candidate Filtering and Validation

The match candidate has to be tested for its actual match length, which can be larger than four bytes. The LZ4 format allows the worst-case scenario of a sequence with a match of a length of 4 bytes. This scenario requires each match to be validated in a single clock cycle to achieve the 10 Gbps throughput.

This process is handled by Match Length Finder (MLF) block by utilizing a large memory bus of the Input Buffer (128 bits - e.g., 16 bytes) and a predictive execution. The simplified MLF architecture is depicted in Fig. 4.

The MLF block can access the Input Buffer with a latency of 1 clock cycle, and the match processing is pipelined. MLF always predicts that the match length will be shorter then 20 bytes and the next read address will be for the next match. The throughput can be evaluated in the following two cases.

Solution	Binary Compatible	Slices	LUTs	LUTMEMs	Regs	BRAM	DSP	URAM	Throughput	Frequency
HALZ4 single-core [6]	Yes	-	-	-	-	-	-	-	0.5 Gbps	75 MHz
MLZ4C-1 [5]	No	571	1302	-	605	76.5	0	0	0.8 Gbps	120 MHz
Xilinx LZ4 single-core [20]	Yes	-	2041	16333	8 525	19.25	0	6	1.66 Gbps	274 MHz
MLZ4C-2 [5]	No	345	573	-	937	69	4	0	1.92 Gbps	240 MHz
HALZ4 many-core [6]	Yes	-	-	-	-	-	-	-	4.0 Gbps	75 MHz
Our single-core architecture	Yes	-	14076	433	2803	82	32	0	6.08 Gbps	156.25 MHz
Xilinx LZ4 many-core [20]	Yes	-	71934	16333	68201	154	1	48	13.28 Gbps	274 MHz

TABLE I
RESOURCE UTILIZATION AND PERFORMANCE COMPARISON

Corpus	64	128	256	512	1024	2048	4096	ref.
Cantebury	1.41	1.46	1.50	1.57	1.68	1.75	1.81	2.27
Silesia	1.24	1.28	1.31	1.32	1.40	1.48	1.56	2.42
Calgery	1.29	1.34	1.38	1.39	1.47	1.54	1.61	2.08

TABLE II
RELATION BETWEEN THE NUMBER OF ENTRIES IN THE COMPRESSION DICTIONARY AND THE COMPRESSION RATIO

1) Prediction successful - the current match length is inside of the current bus width and the next clock will process the next match. The throughput is in the worst-case equal to 10 Gbps. Therefore, the MLF is able to process matches with lengths of 4-20 bytes in a single clock cycle.
2) Prediction unsuccessful - the current match is outside of the current bus width, flush memory pipeline, and issue the next address. The next match will be validated in 3 clock cycles in total, which will have the throughput greater than 10 Gbps (20 Gbps actually due to the doubled memory bus width). This respective speed-up allows to compensate the initial latency.

This process can be repeated until the full match length is determined. The issue with overlapping matches is then solved by skipping match candidates based upon the last validated match.

C. Format Encoding

The last step in generating the LZ4 format is to encode the validated matches and unmatched literals. This step is not optimized to match the 10 Gbps performance in the worst-case, which does not occur in the real-life application. However, the throughput is being increased by using a 20 Gbps memory bus for the literal transfer. The memory bus allows to temporally increase the performance of the most used operation during encoding. It also allows us to overcome the performance loss during some other operations.

V. RESULTS

In this section, our architecture is compared with other FPGA implementations of the LZ4 algorithm. The designs were compared by their frequency, resource utilization, and throughput in Table I. The respective implementation platforms (FPGAs) are found comparable in terms of resources and/or frequency due to its similarity. The comparison includes MLZ4C-1 [5], MLZ4C-2 [5], HALZ4 [6] and Xilinx LZ4 [20] official implementation from Xilinx.

The implementation of our single-core architecture is the most compact one among others, and it has the best throughput out of the existing solutions (MLZ4C-1, MLZ4C-2, HALZ4, Xilinx LZ4). In case the provided results of many-core architectures are not showing the resource utilization for a single core, a simple division estimated the resource utilization of such core. This attitude has no impact on a single core throughput.

Our single-core architecture can be used to create a trivial many-core architecture thanks to its high throughput and low latency, which can be compared to the many-core architecture Xilinx LZ4. It would have a comparable throughput, lower latency factor of 4x, and its resource usage would be just a fraction of the resources used by the Xilinx LZ4 architecture. Therefore, it would also outperform HALZ4.

The compression ratios of our architecture for all of the commonly used corpora Calgery [17], Canterbury [18] and Silesia [19] are shown in Table II. The compression ratio depends on the size (number of entries) of the dictionary used inside the architecture. The official implementation of the LZ4 algorithm is used as reference [22].

VI. FUTURE WORK

We would like to improve our single-core architecture to reach the throughput of 10 Gbps, which could be used to create trivial many-core architecture capable of 100 Gbps compression speed while maintaining the same latency. Further optimizations that can be applied to the compression dictionary will be also explored [23].

VII. CONCLUSION

We have presented a single-core architecture implementing the LZ4 compression algorithm. Our architecture is optimized for low-latency and high-throughput systems. It outperforms existing single-core architectures by 300% thanks to its internal parallelism with an 8x factor. Our architecture also requires the least amount of FPGA resources while it keeps the full binary compatibility with the LZ4 format. The minimum width of the datapath is 64-bits compared to other implementation using an 8-bit datapath only. The architecture can be used for the implementation of a many-core architecture by using trivial parallelism. The resulting architecture would have a comparable throughput, lower latency at least by a factor of 4, and a lower resource usage than existing many-core

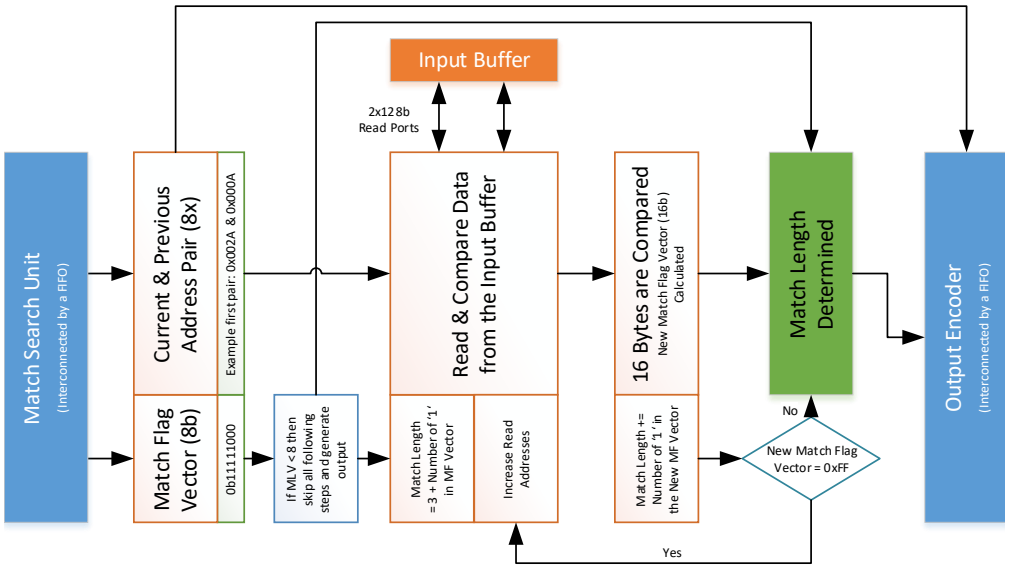


Fig. 4. Simplified architecture of Match Length Finder block

architectures. We have also proposed the idea of reducing the latency of a compression system by only increasing the throughput of a used elementary engine.

ACKNOWLEDGMENT

This research has been partially supported by the CTU project SGS17/017/OHK3/1T/18 “Dependable and attack-resistant architectures for programmable devices” and by the project “E-infrastructure CESNET – modernization” no. CZ.02.1.01/0.0/0.0/16 013/0001797.

REFERENCES

[1] J. Ziv and A. Lempel, ”A universal algorithm for sequential data compression,” in IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, May 1977. doi: 10.1109/TIT.1977.1055714

[2] D. Harnik, E. Khaitzin, D. Sotnikov and S. Taharlev, ”A fast implementation of Deflate,” 2014 Data Compression Conference, Snowbird, UT, 2014, pp. 223-232. doi: 10.1109/DCC.2014.66

[3] M.F.X.J Oberhumer, ”LZO real-time data compression library””, 2011. [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[4] M. Bartík, S. Ubik and P. Kubalík, ”LZ4 compression algorithm on FPGA,” 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, 2015, pp. 179-182. doi: 10.1109/ICECS.2015.7440278

[5] W. Liu, F. Mei, C. Wang, M. O’Neill and E. E. Swartzlander, ”Data compression device based on modified LZ4 algorithm,” in IEEE Transactions on Consumer Electronics, vol. 64, no. 1, pp. 110-117, Feb. 2018. doi: 10.1109/TCE.2018.2810480

[6] S.M. Lee, J.H. Jang, J.H. Oh, J.K. Kim and S.E. Lee, ”Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression,” IEICE Electronics Express, ISSN 1349-2543 , doi: 10.1587/elex.14.20170399

[7] J. Kane and Q. Yang, ”Compression speed enhancements to LZO for multi-core systems,” 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing, New York, NY, 2012, pp. 108-115. doi: 10.1109/SBAC-PAD.2012.29

[8] ZipAccel-C, GZIP/ZLIB/Deflate Data Compression Core [Online] Available: http://www.cast-inc.com/ip-cores/data/zipaccel-c/cast-zipaccel-c-x.pdf

[9] J. Fowers, J. Y. Kim, D. Burger and S. Hauck, ”A scalable high-bandwidth Architecture for lossless compression on FPGAs,” 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, 2015, pp. 52-59. doi: 10.1109/FCCM.2015.46

[10] B. Sukhwani, B. Abali, B. Brezzo and S. Asaad, ”High-throughput, lossless data compresion on FPGAs,” 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, Salt Lake City, UT, 2011, pp. 113-116. doi: 10.1109/FCCM.2011.56

[11] R. Mehboob, S. A. Khan, Z. Ahmed, H. Jamal and M. Shahbaz, ”Multigig lossless data compression device,” in IEEE Transactions on Consumer Electronics, vol. 56, no. 3, pp. 1927-1932, Aug. 2010. doi: 10.1109/TCE.2010.5606348

[12] K. Papadopoulos and I. Papaefstathiou, ”Titan-R: A reconfigurable hardware implementation of a high-speed compressor,” 2008 16th International Symposium on Field-Programmable Custom Computing Machines, Palo Alto, CA, 2008, pp. 216-225. doi: 10.1109/FCCM.2008.14

[13] Y. Collet, ”Real Time Data Compression: LZ4 Explained”, 2011 [Online]. Available: http://fastcompression.blogspot.ru/2011/05/lz4-explained.html

[14] CESNET, ”Special video transmission,” 2017. [Online] Available: https://www.cesnet.cz/services/special-video-transmissions/?lang=en

[15] Ticki, ”How LZ4 works”. [Online]. Available: http://ticki.github.io/blog/how-lz4-works/

[16] M. Bartík, T. Beneš and P. Kubalík, ”Design of a High-Throughput Match Search Unit for Lossless Compression Algorithms,” 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2019, pp. 0732-0738. doi: 10.1109/CCWC.2019.8666521

[17] T.C. Bell, I.H. Witten and J.G. Cleary, ”Modeling for text compression,” Computing Surveys 21(4): 557-591; December 1989; ISSN: 0360-0300. doi: 10.1145/76894.76896

[18] R. Arnold and T. Bell, ”A corpus for the evaluation of lossless compression algorithms,” Proceedings DCC ’97. Data Compression Conference, Snowbird, UT, USA, 1997, pp. 201-210. doi: 10.1109/DCC.1997.582019

[19] S. Deorowicz, ”Silesia Corpus.” Silesian University of Technology, 2003 [Online] Available: http://sun.aei.polsl.pl/ sdeor/index.php?page=silesia

[20] LZ4 compression/decompression is FPGA based implementation of standard LZ4, Xilinx [Online]. Available: https://tinyurl.com/y58dwo25

[21] Kim, Jeehong & Cho, Jundong. (2019). Hardware-accelerated Fast Lossless Compression Based on LZ4 Algorithm. 65-68. doi: 10.1145/3316551.3316564.

[22] LZ4 - Extremely fast compression [Online]. Available: https://github.com/lz4/lz4

[23] M. Bartík, S. Ubik and P. Kubalík, ”A novel and efficient method to initialize FPGA embedded memory content in asymptotically constant time,” 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2016, pp. 1-6. doi: 10.1109/ReConFig.2016.7857146

Chapter 6

Novel Status Register Architecture

This chapter describes and summarizes a novel architecture suitable for implementing a status register, which is commonly used to store information, of which memory entry is valid or invalid. Such memory-based data structures (a compression dictionary, for example) are often cleared or initialized to a default value between individual runs of an implemented algorithm (not particularly limited to compression algorithms only). With the aid of a status register, we can determine a value occupying a specific address is the default value (no write occurred) or the written value is the same as the default one. For example, it is difficult to differentiate between a memory address value *0x0000*, because it can represent a valid address (the first cell of an array) or an uninitialized value at the same time.

Two state-of-the-art techniques (sequential zeroing and assigning a flip-flop logic gate to each memory cell) are commonly used; however, they were found unsuitable from the perspective of latency or resource utilization. A medium-sized memory has been defined as a memory consisting of several BlockRAMs, e.g., several kilobytes of RAM. This range (and memory type) is often used to implement a compression dictionary (among other suitable data structures). The range is viable for the presented use-case, the IP packet lossless compression in real-time.

The novel architecture supersedes the flip-flop based principle. The entire address range (array of flip-flops) is split into smaller chunks segments of 64-bits. Each segment is replaced by "Look-Up Table" (a fundamental block in FPGAs realizing logic gates), which is configured as "Distributed Memory", and can hold the same individually addressable 64 bits. Because of FPGA organization, the amount of resources is lowered 64 times. Therefore, the design is more dense with a significantly reduced number of interconnections. Consequently, this results in an easier design placement and routing which allows running the respective design on a higher frequency. However, some additional logic is required to perform reset. The time required for the reset is 64 clock cycles, which is the only minor disadvantage.

Overall, the presented status register architecture combines the advantages of both predecessors to improve overall performance without any significant disadvantage. The architecture is portable to any modern FPGAs.

At last, a benchmark with multiple phases was developed to evaluate the architecture in a fair way. The benchmark uses randomized pin placement to decrease the influence of respective tools and actual hardware configuration of computers used for the synthesis process.

6.1 Alternative Use Case - Histogram Calculation

The use case of a compression dictionary is not the only one, which can benefit from the "Distributed" technique. The technique is suitable for any (larger) temporary memory structures, which has to be often (re-)initialized. A histogram calculation (with several thousands of bins) [84] [85] [86] can be the alternative use case.

For example, the presented technique could be great in a combination of this particular paper [87], which could further reduce the number of activated memory banks for read accesses only (the banks that haven't been written into doesn't need to be activated; therefore, they are not consuming power).

6.2 Analysis of LZ4 Suitability for Image Data

To justify the required size of a compression dictionary (and the need to introduce a new status register architecture), I performed an experiment to show the ability of LZ4 to compress image data with respect to different color depths and encodings. The used "Corpus" consisted of eight uncompressed images representing typical scenes with a different complexity taken by a digital camera. This dataset was provided by the CineGrid [88, 89]. The last dataset is a captured traffic from an MVTP device (several real-world recordings in a loop).

The results (see Table A.2) indicate LZ4 could save approximately 13% of the required network bandwidth in our primary use case of broadcasting SDI data streams while using a decently sized compression dictionary. Therefore, the LZ4 compression algorithm can be considered viable to fulfill the initial requirement of the "Light" compression to save approximately 10%. The suitability of LZ4 for image compression was independently confirmed by another research [74].

The content of this chapter is based on the following papers:
Bartík, M. and Ubik, S. and Kubalík, P., *"A Novel and Efficient Method to Maintain FPGA Embedded Memory Content with an Asymptotically Constant Time (Re)Initialization Designed for an IP Packet Lossless Compression"*, International Conference on ReConFigurable Computing and FPGAs (ReConFig 2016), ISBN 978-1-5090-3707-0, pp. 1–6, Cancún, Mexico, 2016 [A.2].

Bartík, M. and Beneš T. and Kubalík, P., *"An In-sight into How Compression Dictionary Architecture can Affect the Overall Performance in FPGAs"*, IEEE Access,

2020 (8), pp. 183101–183116, ISSN 2169-3536, 2020 [A.6].

Contributions of Tomáš Beneš are bash scripts to automate experiments and to extract presented results.

A Novel and Efficient Method to Initialize FPGA Embedded Memory Content in Asymptotically Constant Time

Matěj Bartík
CTU FIT & CESNET a.l.e.
matej.bartik@fit.cvut.cz

Sven Ubik
CESNET a.l.e.
ubik@cesnet.cz

Pavel Kubalík
CTU FIT
pavel.kubalik@fit.cvut.cz

***Abstract*—This paper describes analysis and implementation of a new method for maintaining valid content of FPGA memory blocks with an asymptotically constant time synchronous clear ability, that can be useful for (re)initialization to one default value. A particular application can be for high-speed real-time LZ77 [1] lossless compression algorithms, where a dictionary has to be (re)initialized before each run of the implemented compression algorithm.**

The method is based on two most widely used techniques for clearing the memory content: a linear passage of the memory and clearing each cell by writing a default value and creating a register field providing an (in)valid bit for each memory cell. Our solution combines these two techniques together with the use of FPGA distributed memory blocks implemented in LUTs (Look-Up Tables) to overcome negative features of each previous method without losing the most of positive features. Our solution provides a balance between the two previous techniques and exceeds them in speed, resources utilization and latency of (re)initialization.

I. Introduction & Motivation

Fast lossless compression algorithms become very popular, even though they do not reach compression ratios of their predecessors like LZ77 [1]. An important application of these algorithms is for reducing bandwidth requirements of fast real-time network transmissions. A typical target use are IP packet compression devices like [2]–[5].

A. Fast Lossless Compression Algorithms

These algorithms have been designed to favor the compression speed (compression time) over a compression ratio, because in some cases it is more important to meet throughput requirements rather than the compression ratio. These algorithms have to be lossless because of the fundamental principles of packet network communication.

The most widely known representative examples of modern fast lossless compression algorithms are LZ4 [6], LZO [7], Snappy [8], QuickLZ [9], etc.. All of these algorithms are based on the Lempel-Ziv (LZ77) lossless compression schema, which means that all algorithms are using a dictionary for searching a match of processed data. The speed improvements are achieved by the use of many CPU (Central Processing Unit) optimizations or general computer architecture optimizations (an efficient memory access) respectively [10], [11].

B. LZ77 Lossless Compression Scheme

LZ77 [1] is the fundamental lossless compression scheme used in many further algorithms like GIF or DEFLATE [12]. LZ77 introduced the sliding window principle, where the sliding window is usually divided into a search buffer (a dictionary) and a look-ahead buffer. The found longest prefix of the look-ahead buffer starting in the search buffer is encoded as a triplet **(i, j, X)**, where **i** is the distance of the beginning of the found prefix from the end of the search buffer, **j** is the length of the found prefix and **X** is the first character after the prefix in the look-ahead buffer.

C. LZ77 Based IP Packet Compression

The LZ77 algorithm and LZ77 derivatives are widely used in IP packets compression devices like [2]–[5]. The research [13], [14] in the last 20 years has shown LZ77 abilities to reach high throughput of tens of gigabits per second when implemented in hardware (FPGA). Some of the hardware implementations have adopted the SIMD (Single Instruction Multiple Data) principle and introduced an implementation with wider processed words like 64 or 128 bits per one clock cycle. Other hardware implementations rely on multiple independent compression blocks.

D. LZ4 Fast Lossless Compress Algorithm

LZ4 [6] is a representative example [11] of a modern and fast lossless compression algorithm. The main difference to LZ77 is the use of hash-based search algorithm [16]. Reference addresses of 32-bit data words are stored in a hash table, which realizes a dictionary. This dictionary has to be (re)initialized (to all zeros) before each run of the LZ4 algorithm and it is designed to fit in a L1 CPU cache. LZ4 is designed to process data in an efficient way, most of data flow operations are processed in 32-bit (hash calculation) or 64-bit (memory access) words to exploit the maximum memory bandwidth. However, memory buffers are byte oriented and the memory subsystem of the LZ4 algorithm has to support 8-bit unaligned memory access. This results into a complex memory subsystem [16].

978-1-5090-3707-0/16/$31.00

E. Evaluation of LZ4 for Multimedia Use

There were some attempts to use a modern lossless compression algorithm for multimedia streams (4K/UHD in this particular example [17]) under real-time conditions with average reduction of 23% of the required bandwidth. This particular example was measured using a software version of LZ4 running on a high-performance PC equipped with Dual Xeon E5620 (8 cores in total) and 32 GB RAM.

Based on previous results, we did our experiments [18] with the LZ4 algorithm, setting LZ4 parameters (and thus its behavior) to be more suitable for a hardware implementation in FPGA. We assumed the following properties:

- Hardware implementation will be focused on IP packet lossless compression, where packets are carrying multimedia data split into jumbo packets of the maximum size 9216 bytes [19].
- The hash table (dictionary) will be implemented in the FPGA embedded memory (such as BlockRAM in Xilinx FPGAs), thus limiting its size to several kilobytes. Larger hash tables will be unsuitable for this hardware implementation.
- We will measure LZ4 compression ratios on multimedia testing datasets with different color depths and color encodings. Testing data sets were from the CineGrid database [20] with the exception of the SDI format, where the real transmission data were used.

The measured results are shown in Table I and visualized in Figure 1. **N** parameter represents the number of hash table records, where each record is a 4 byte wide (32-bit memory pointer) resulting in LZ4 hash table size formula 2^{N+2}. The compression ratio is changing according to the change of the **N** parameter. A sweet spot has been found for the **N** parameter in the range of 12–14 (4096–16384 records respectively).

TABLE I
AVERAGE LZ4 COMPRESSION RATIO VS. HASH TABLE SIZE VS. IMAGE COLOR DEPTH AND COLOR ENCODING [18].

Encoding \ N	8	10	11	12	13	14
8-bit RGB	0,686	0,669	0,655	0,643	0,636	0,632
24-bit RGB	0,914	0,905	0,902	0,899	0,898	0,897
32-bit RGB	0,863	0,850	0,845	0,841	0,839	0,838
48-bit RGB	0,991	0,989	0,988	0,988	0,988	0,987
SDI 20-bit YCbCr	0,883	0,878	0,874	0,871	0,868	0,867

F. LZ4 for IP Packet Compression

LZ77 (and the successors like LZ4) can achieve [2]–[4], [13], [14] the throughput of 10 Gbps or more. The minimal requirement for 10 Gbps throughput in an FPGA design is a 64-bit data bus clocked at 156.25 MHz. We have to ensure that the compression design will be able to process 64-bit data in each clock cycle.

The maximum time for data loading into an input buffer will be 1150 clock cycles (9000 byte is the maximum payload size [15] of a jumbo packet further divided by 64-bit datapath size). The hash table sweet spot begins at 1024 records and

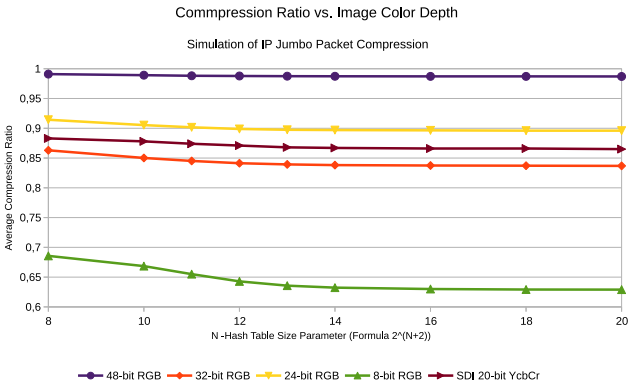


Fig. 1. The dependency of the compression ratio and the dictionary size during the simulated LZ4 compression of IP jumbo packets [18].

ends at 4096 records. Its unreasonable to create a hash table larger than the input buffer. The size of 1024 records is the largest hash table (dictionary) suitable for (re)initializing by the linear memory passage (a simple counter can be used) with the minimum impact to FPGA resources. For larger hash table sizes, status flags indicating that a given table item has been written can be used, but the impact on FPGA resources is much bigger, causing the overall design size to be increased by 300% (for $N = 10$). For a higher **N** parameter, the overhead will be even much higher.

G. Problem Definition

Our research is focused on low-latency and high-throughput real-time multimedia compression applications. Our assumptions and experiments with the hardware based version of the LZ4 compression algorithm resulted in to the data flow and dependency analysis. This analysis has shown the dictionary initialization process a weak spot that increases the latency significantly.

II. METHODS FOR MAINTAINING VALID DICTIONARY CONTENT

There are two fundamental methods for maintaining valid dictionary records [21], the linear passage and the status register field. Both methods are widely used in FPGA designs. They differ to each other in speed (design frequency), (re)initialization time (latency), required resources and suitability for particular use cases. We assume that the evaluated dictionary (memory) is initialized with one constant value.

A. Linear Passage

The linear memory passage is a fundamental method for (re)initializing memory to a default (constant) value [22]. The fundamental part is a counter with the same width as the memory address vector, thus all addresses are generated (including other control signals like write enable) for the (re)initialization purpose. The data input port (vector) is multiplexed by the default value during the (re)initialization process.

Advantages of the linear passage method are: a simple and straightforward design and low resources requirement.

Disadvantages are: that the (re)initialization process requires a lot of clock cycles to pass through all addresses and that the adder used by a counter has a long carry chain that limits the maximum design frequency. These two disadvantages increase latency of the design.

B. Flip-Flop Based Status Register File

The second approach [21] is flip-flop based status register file, where each flip-flop preserves a bit wide flag indicating the status of the related memory cell (written or not written). When a memory read occurs before a write operation, the memory output will be multiplexed to the default read value.

Advantages of the flip-flop based approach are higher operating frequency than with the linear passage and the latency of one clock cycle, because all flip-flops can be effectively cleared in parallel. Disadvantages are the exponential growth of required resources with respect to the memory address vector width. When a large memory is used, the operating frequency will drop significantly due to FPGA routing and synthesis issues.

III. Our Approach

In this section, we present an alternative way for storing informations inside FPGA without using flip-flops and we describe our approach in detail .

A. Alternative Ways for Storing Information Inside FPGA

Flip-flops are not the only way for storing information inside FPGA. We designed our solution with Xilinx FPGA resources [23], but there are no restrictions for the use with Altera FPGAs. A Configuration Logic Block (CLB) is an elementary FPGA block, consisting of two slices. Each slice consists of four Look-Up Tables (LUTs) and eight flip-flops.

30% of LUTs in Xilinx FPGAs have three modes of operation [23]. The first is the default look-up table designed for realizing a combinatorial logic with up to 6 inputs and one output. The second mode is a SRL mode where the LUT is transformed into a shift register with up to 32-bits without set or reset abilities. This can be considered as a disadvantage. The last mode of operation is distributed memory, where the LUT is turned into a small RAM consisting up to 64x1-bit single port memory. The memory capacity and the access port count can be scaled up to 256x1-bit single port or 64x1-bit quad port distributed memory within one slice [23].

B. Distributed Memory Based Solution

Distributed memory features a higher density (256 bits) per slice than flip-flops (8 bits). The amount of resources is doubled when a CLB is considered. The high dense design will make a routing process easier.

Distributed memory can not be cleared in one clock cycle like a flip-flop based approach and requires a linear passage for clearing all bits. The latency of the distributed memory based design is limited to 64 clock cycles in our particular example. The limit can be computed as $Latency = 2^{LUTInputs}$. Due to FPGA parallelism, all distributed memory blocks can be cleared in parallel.

The asymptotic computational complexity is the same as in the flip-flop based approach because the latency is independent from the problem size (memory address vector width). The same idea can be applied to the consumed resources, when both methods require an exponential amount of resources related to the memory address vector width. Both methods are equivalent from the asymptotic complexity point of view [24].

C. Elementary Block Design

The elementary block (EB) design (depicted in Figure 2) consists from a distributed memory block of 64-bit size, where each bit represents the status flag (written or not written). The second part is an address multiplexer, where one address input is used to the standard operation mode and the second input is dedicated to the (re)initialization mode. The reset signal is used for switching the multiplexer, the write enable and the input data signal during the process of (re)initialization.

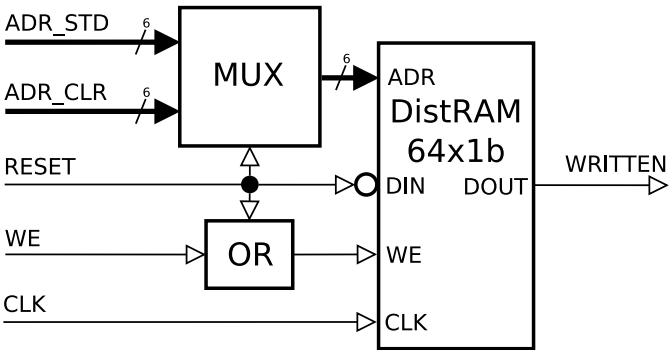


Fig. 2. The inner architecture of the elementary block.

D. Architecture of the Status System

The architecture of the status system consists from five main blocks.The block diagram of the architecture is depicted in Figure 3. The first part is a set of multiple elementary blocks corresponding to the required main memory (dictionary) size. The second part is the output selection logic used for picking the right elementary block output (by masking outputs).

The third part is the memory address splitter for splitting the memory address vector into two parts. The lower part (6 bits according to the number of LUT input ports) is forwarded directly to the elementary blocks as the standard operation mode address. The upper part is forwarded to the fourth block denoted as EB SEL. The EB SEL is an acronym for the elementary block selector, thus EB SEL is the address decoder.

The fifth and the last block is the modulo 64 counter generating the address range for all elementary blocks during the (re)initialization process.

E. Testing Setup

We developed the design implementing all the three mentioned approaches (linear passage, flip-flop array and distributed memory based approach) with a unified interface (see Program 1).

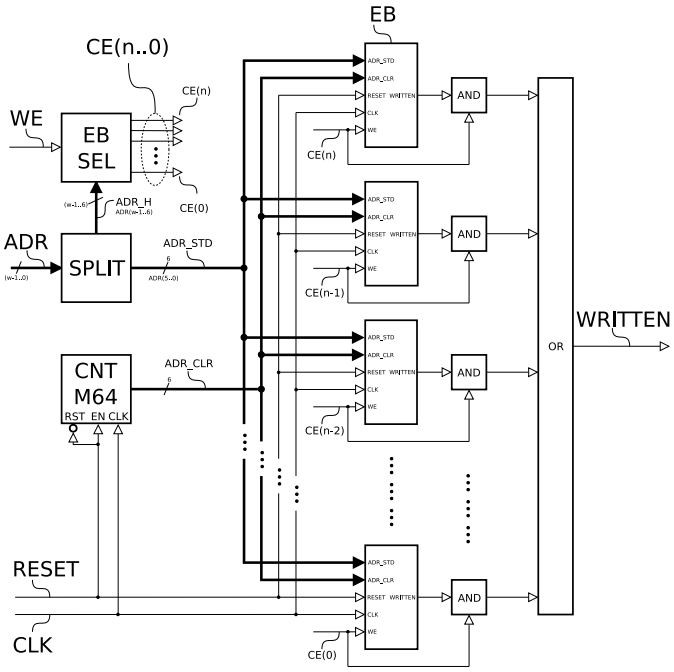


Fig. 3. Architecture of the LUT based status flag system.

Program 1 A unified VHDL interface for all implementation types.

```
entity top is port (
  clk, reset, we : in std_logic;
  adri : in std_logic_vector(W-1 downto 0);
  din  : in std_logic_vector(35 downto 0);
  dout : out std_logic_vector(35 downto 0);
  written : out std_logic);
end top;
```

The implementation type and some other parameters (such as LUT size) are set by generic constants. The design properties and assumptions are the following:

- LUT has 6 inputs,
- A memory cell width is 36 bits (one of the native BlockRAM widths [25], intended for simulating a 32-bit memory pointer like software version of LZ4 does, this dictionary cell width has been used elsewhere [26]),
- **W** parameter stands for the intended memory address vector width and defines the memory capacity ($2^W * 36bits$),
- The output multiplexer is not implemented in the flip-flop based approach and distributed memory based approach (we prefer the status flag to the default value for handling by a control finite state machine). The default value detector is not implemented in the linear passage approach as a compensation,
- The design input and output signals are buffered with flip-flops,
- The synthesis strategy favors the design speed,

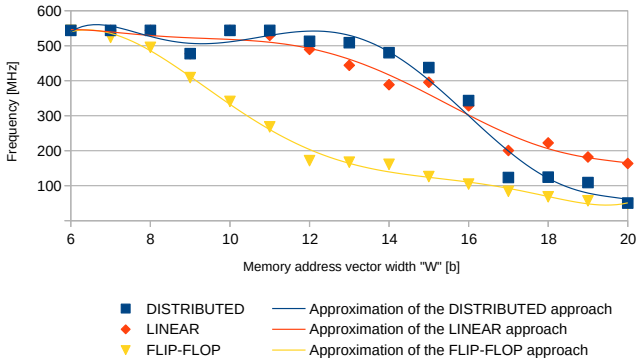


Fig. 4. The relation between maximum design frequency and the **W** parameter across all approaches.

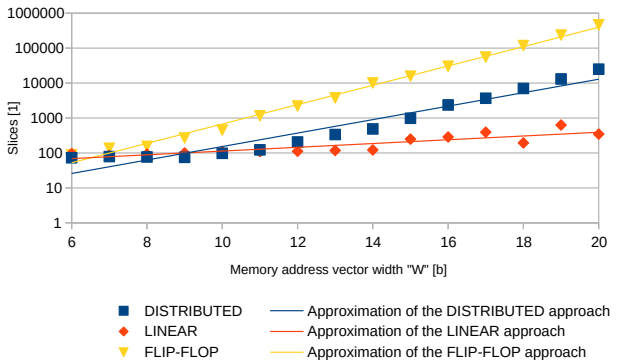


Fig. 5. A relationship between the design FPGA resources consumption and the **W** parameter across all approaches.

- The frequency is measured on a fully routed design.

The design has been synthesized by the Xilinx ISE toolset (14.7 version) on a 6-core Intel Xeon E5-1650 v3 (15 MB Cache, 3.50 GHz), 32 GB DDR4 and SSD drive. The selected FPGA chip has been Virtex-7 690T (XC7V690T-2FFG1158). This chip is the highest density FPGA included in our Xilinx ISE license.

F. Results

The resource consumption, achieved frequency and latency are summarized in Table II and visualized in charts Figure 4 (speed comparison) and Figure 5 (FPGA resources consumption).

The evaluation can be divided into three parts based on value of the **W** parameter. For **W** in the range 6–10 we see an advantage of the distributed memory based solution, beating other approaches in the maximum achieved frequency and it consumes the same or less amount of resources. For **W** in the range of 11–15 the distributed memory based solution has an advantage to the flip-flop based approach in the maximum frequency and lower FPGA resources utilization.

The flip-flop based solution can not even be synthesized if the **W** parameter was equal or greater than 16. This behavior was probably caused by the inefficiency of the synthesis tool (Xilinx XST) to synthesize such a large design. When the **W** parameter was equal or greater than 17, a significant drop of

Linear passage approach															
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Slices	96	85	93	100	111	110	112	118	122	249	287	393	194	631	347
LUTs	179	185	191	196	197	203	206	211	215	386	457	533	299	805	568
Registers	50	52	54	56	59	61	63	65	67	75	76	95	79	90	91
Frequency [MHz]	543,8	543,8	543,8	477,3	543,8	529,7	489,5	444,0	388,7	395,6	328,3	200,5	222,3	182,0	163,5
Latency [Cycles]	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576
Flip–flop based approach															
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Slices	89	137	155	268	453	1157	2205	3776	10108	15774	*30335*	*55482*	*118295*	*236468*	*463594*
LUTs	221	309	478	837	1542	3417	6562	13850	31750	57582	*153323*	*242202*	*525419*	*1119292*	*2349075*
Registers	111	176	303	594	1148	2223	4157	8270	16607	32928	*75852*	*128087*	*258144*	*512953*	*1023707*
Frequency [MHz]	543,8	523,8	495,3	408,8	340,8	268,2	171,5	167,1	161,1	126,0	*104,4*	*83,8*	*68,1*	*56,7*	*47,5*
Latency [Cycles]	1														
Distributed memory based approach															
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Slices	73	79	77	75	98	123	206	337	488	984	2364	3686	7033	13104	24987
LUTs	146	151	159	171	209	235	377	471	912	1889	3657	6737	13341	25227	55607
Registers	54	57	63	73	91	129	195	239	619	1118	2176	71	72	190	74
Frequency [MHz]	543,8	543,8	543,8	477,3	543,8	543,8	512,6	508,6	480,1	437,4	343,2	123,4	124,3	108,9	50,2
Latency [Cycles]	64														

*Grey colored results has been extrapolated due to inability of Xilinx XST to synthesize such a large design.

TABLE II
PROPERTIES OF MEASURED DESIGN LIKE LOGIC GATES COUNT, FREQUENCY, ECT. DEPENDING ON THE **W** PARAMETER.

frequency and register count occurred. The synthesis tool (the Xilinx XST from Xilinx ISE 14.7 toolset) probably broke the locality of the references principle (counter modulo 64 was not replicated as for designs with the **W** parameter smaller than 16).

The linear passage approach shows the best results of resource consumption and decent frequency for higher values of the **W** parameter, but the latency (2^W clock cycles) is enormous across the whole range. The latency is 1 clock cycle for the flip-flop based approach and 64 clock cycles for the distributed memory approach.

Although the latency of our approach 64 clock cycles is higher than in the flip-flop based approach is still smaller than the latency of the compression block. The compression time (loading data + compression itself + unloading data) will usually take more time than the dictionary (re)initialization in our case where jumbo packets have been used. The latency of the linear passage approach will not satisfy the requirements for higher values of the **W** parameter, which are typical for desired dictionary capacities (the latency should be less than 1150 clock cycles, but the less, the better).

IV. CONCLUSION

We presented a novel approach for maintaining valid content of an FPGA embedded memories, which is suitable for implementing a dictionary for lossless compression algorithms like LZ77 or LZ4, which is suitable for real-time IP packet compression applications. We analyzed the most widely used approaches (flip-flop array and linear passage) and their advantages and disadvantages. The novel approach is based on a distributed memory mode of a Xilinx LUT blocks and it combines the advantages of both previously known approaches with only one minor disadvantage.

Our approach has the same asymptotic time complexity as the flip-flop approach. However, our approach has better performance, resource utilization and synthesis abilities

Our approach outperforms the linear passage based approach in the achieved frequency (with the exception of synthesis issues for address width over 16) and the latency across the measured range. The linear passage approach is outperformed even in resource utilization, when the **W** parameter is equal or less than 10.

The only minor disadvantage of our approach is a slightly increased latency of 64 clock cycles when compared to the 1 clock cycle of the flip-flop based approach, but this not an issue issue for the target application in lossless compression algorithms.

Our approach can be used for the effective implementation of a dictionary for compression algorithms in a mid-density memory like Xilinx UltraRAM feature [27], which has increased the FPGA memory density by 600%. This particular use case can be covered by the range 15–20 of the **W** parameter. Our approach can be used for FPGAs of other vendors like Altera. We plan to continue with our measurements on different platforms (Altera based) or synthesis tools such as Xilinx Vivado or Altera Quartus.

ACKNOWLEDGMENT

This work has been partially supported by the CESNET Large Infrastructure project (LM2010005) financed by the Ministry of Education, Youth and Sport of the Czech Republic and by the grant SGS16/121/OHK3/1T/18.

REFERENCES

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," in IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, May 1977. doi: 10.1109/TIT.1977.1055714

[2] Mehboob, R.; Khan, S.A.; Ahmed, Z.; Jamal, H.; Shahbaz, M., "Multigig lossless data compression device," Consumer Electronics, IEEE Transactions on , vol.56, no.3, pp.1927,1932, Aug. 2010, doi: 10.1109/TCE.2010.5606348

[3] El Ghany, M.A.A.; Salama, A.E.; Khalil, A.H., "Design and Implementation of FPGA-based Systolic Array for LZ Data Compression," Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on , vol., no., pp.3691,3695, 27-30 May 2007, doi: 10.1109/ISCAS.2007.378644

[4] Papadopoulos, K.; Papaefstathiou, I., "Titan-R: A Reconfigurable Hardware Implementation of a High-Speed Compressor," Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on , vol., no., pp.216,225, 14-15 April 2008 doi: 10.1109/FCCM.2008.14 [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4724904

[5] M. Stohanzl, Z. Fedra and M. Bobula 1 Gbps Ethernet TCP/IP and UDP/IP Header Compression in FPGA, in Proceedings of The Seventh International Conference on Systems and Networks Communications, ICSNC 2012, Lisbon, 2012, pp. 136142.

[6] Collet, Y.: RealTime Data Compression: Development blog on compression algorithms. [Online]. Available: tinyurl.com/qc9yve4

[7] Oberhumer, M.:LZO real-time data compression library [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[8] Google Inc.: Snappy - A fast compressor/decompressor. [Online]. Available: https://google.github.io/snappy/

[9] Seipp, A.: QuickLZ 1.5.x compression library [Online]. Available: http://www.quicklz.com/

[10] J. Kane and Q. Yang, "Compression Speed Enhancements to LZO for Multi-core Systems," Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on, New York, NY, 2012, pp. 108-115. doi: 10.1109/SBAC-PAD.2012.29

[11] Fiedler, O.: LZ-Family Data Compression Methods, Bachelor Thesis, 2014. Available: https://dspace.cvut.cz/handle/10467/24453

[12] Solomon, D.: Data Compression: The Complete Reference (Fourth ed.). 20007, Springer. ISBN 978-1-84628-602-5.

[13] B. Sukhwani, B. Abali, B. Brezzo and S. Asaad, "High-Throughput, Lossless Data Compresion on FPGAs," Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on, Salt Lake City, UT, 2011, pp. 113-116. doi: 10.1109/FCCM.2011.56

[14] J. Fowers, J. Y. Kim, D. Burger and S. Hauck, "A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs," Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on, Vancouver, BC, 2015, pp. 52-59. doi: 10.1109/FCCM.2015.46

[15] M. Bencivenni et al., "Performance of 10 Gigabit Ethernet Using Commodity Hardware," in IEEE Transactions on Nuclear Science, vol. 57, no. 2, pp. 630-641, April 2010. doi: 10.1109/TNS.2009.2032264

[16] M. Bartík, S. Ubik and P. Kubalík, "LZ4 compression algorithm on FPGA," 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, 2015, pp. 179-182. doi: 10.1109/ICECS.2015.7440278

[17] Ruan Delgado Gomes, Yuri Gonzaga Gonalves da Costa, Lucenildo Lins Aquino Jnior, Manoel Gomes da Silva Neto, Alexandre Nbrega Duarte, and Guido Lemos de Souza Filho. 2013. A solution for transmitting and displaying UHD 3D raw videos using lossless compression. In Proceedings of the 19th Brazilian symposium on Multimedia and the web (WebMedia '13). ACM, New York, NY, USA, 173-176. doi: 10.1145/2526188.2526228

[18] Bartík, M.; Ubik, S.; Kubalík, P., "Rychlé bezztrátové kompresní algoritmy," Sborník příspěvků PAD 2015, ISBN 978-80-7454-522-1, pp. 31–36, 2–4. September 2015

[19] Jiri Halak, Michal Krsek, Sven Ubik, Petr Zejdl, and Felix Nevrela. Real-time long-distance transfer of uncompressed 4k video for remote collaboration. Future Generation Computer Systems, 27(7), pp. 886–892, 2011. CineGrid: Super high definition media over optical networks. doi: 10.1016/j.future.2010.11.014

[20] CineGrid Exchange. [Online]. Available: http://cinegrid.org/

[21] M. Stohanzl and Z. Fedra, "The FPGA implementation of dictionary; HW consumption versus latency," Telecommunications and Signal Processing (TSP), 2013 36th International Conference on, Rome, 2013, pp. 82–85. doi: 10.1109/TSP.2013.6613896

[22] Rigler, S.; Bishop, W.; Kennings, A., "FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms," Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on, vol., no., pp.1235,1238, 22-26 April 2007, doi: 10.1109/CCECE.2007.315

[23] Xilinx Inc.: UG474 - 7 Series FPGAs Configurable Logic Block. [Online]. Available: http://tinyurl.com/jzt3dlc

[24] Tvrdík, P.: Parallel algorithms and computing (Second ed.). Prague: CTU, 2009. ISBN 978-80-01-04333-2.

[25] Xilinx Inc.: UG473 - 7 Series FPGAs Memory Resources. [Online]. Available: http://tinyurl.com/zufqh9w

[26] NAQVI S. Optimized RTL design and implementation of LZW algorithm for high bandwidth applications. [Online]. Available: pe.org.pl/articles/2011/4/68.pdf

[27] Xilinx Inc.: WP447 - UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 12, 2020, accepted September 27, 2020, date of publication October 8, 2020, date of current version October 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3029691

An In-Sight Into How Compression Dictionary Architecture Can Affect the Overall Performance in FPGAs

MATĚJ BARTÍK[1,2], (Member, IEEE), TOMÁŠ BENEŠ[1], (Student Member, IEEE), AND PAVEL KUBALÍK[1]

[1]Department of Digital Design, Faculty of Information Technology, Czech Technical University in Prague, 160 00 Prague, Czech Republic
[2]Department of Technology for Network Applications, CESNET, 160 00 Prague, Czech Republic

Corresponding author: Matěj Bartík (bartimat@fit.cvut.cz)

This work was supported in part by the Czech Technical University Project Design, Programming and Verification of Embedded Systems under Grant SGS20/211/OHK3/3T/18, and in part by the project E-infrastructure CESNET – Modernization under Grant CZ.02.1.01/0.0/0.0/16013/0001797.

ABSTRACT This paper presents a detailed analysis of various approaches to hardware implemented compression algorithm dictionaries, including our optimized method. To obtain comprehensive and detailed results, we introduced a method for the fair comparison of programmable hardware architectures to show the benefits of our approach from the perspective of logic resources, frequency, and latency. We compared two generally used methods with our optimized method, which was found to be more suitable for maintaining the memory content via (in)valid bits in any mid-density memory structures, which are implemented in programmable hardware such as FPGAs (Field Programmable Gate Array). The benefits of our new method based on a ''Distributed Memory'' technique are shown on a particular example of compression dictionary but the method is also suitable for another use cases requiring a fast (re-)initialization of the used memory structures before each run of an algorithm with minimum time and logic resources consumption. The performance evaluation of the respective approaches has been made in Xilinx ISE and Xilinx Vivado toolkits for the Virtex-7 FPGA family. However the proposed approach is compatible with 99% of modern FPGAs.

INDEX TERMS Compression algorithm, compression dictionary, FPGA, hash table, LZ4, LZ77, memory architecture, performance comparison, status register.

I. INTRODUCTION

Lossy or lossless high-speed and low-latency compression is important for many applications in real-time networking, video transmissions or disk storage. The research in lossless compression has led to the development of new types of devices that perform compression in real-time. Over the last decade, the throughput of these devices has increased up to 44.8 Gbps (Gigabit per second) [1] from a gigabit speed. The progress has been made by improving designs step by step with new techniques or tweaks that have made these designs more efficient in terms of speed, logic resources utilization or compression ratio. We focus on the improvement of the specific area of compression algorithms – compression dictionaries and how to increase their overall performance.

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

II. BRIEF MOTIVATION AND CONTRIBUTIONS

In this paper, we focused on exploring new ways of improving the overall performance of hardware-implemented compression algorithms. We put emphasis on these design properties:

- maximum throughput,
- maximum frequency,
- resources utilization,
- computation latency,
- predictability.

The mentioned design properties have no direct impact on the compression ratio; however, they may have an indirect effect, such as using an original amount of logic resources to implement a larger dictionary when a resource-efficient architecture was selected over the original one (due to saved resources).

In some specific use cases, the compression ratio may be less important than the design throughput, latency, and predictability. These requirements are considered to be

fundamental for Real-Time systems [2]. To achieve a great compression ratio, some sophisticated techniques for a match search are usually used (re-hashing principle, for example). On the other hand, these sophisticated techniques introduce variability in data processing, which results in stalls in a compression engine datapath; thus, computation latency is variable and unpredictable.

Several hardware architectures focused on maximizing the design throughput have been introduced in the past few years [1], [3], [4]. On the other hand, no current architecture has emphasized lowering an architecture overhead to reduce the respective architecture computation latency or resource utilization. It has been identified [5] that the overhead (a compression dictionary initialization) necessary for the compression process (computation phase) could require more time than the compression itself.

Our goal is to reduce the computation latency and resource utilization while increasing the operating frequency by improving the compression dictionary architecture.

The contributions of the paper are:

- We analyzed three existing techniques suitable for implementing a compression algorithm dictionary, including our new technique [5].
- We created a new methodology for evaluating the analyzed techniques.
- We performed an experimental test to obtain results.
- A conclusion has been made that our method (''Distributed Memory'') shows better results than the other techniques in terms of maximum frequency, computation latency, and amount of required logic gates for our specific use case. This statement has been supported by quantitative analysis and experimental results.

III. THEORETICAL BACKGROUND

Implementations of lossless compression algorithms in hardware (in both FPGAs and ASICs – Application Specific Integrated Circuit) appeared right after the moment when software implementations were unable to satisfy the desired performance requirements such as throughput or latency. In the last two decades, a device realizing real-time compression of network communication using IP (Internet Protocol) principles became a widespread use case. The authors would like to summarize the properties of such implementations [1], [3], [7]–[32] as follows:

- The majority of designs are based on the LZ77 algorithm [33] or derived algorithms such as LZ78 [34] or LZW [35].
- The latest designs experiment with new derived algorithms focused on better compression ratio (LZMA [12]–[14], [36]) or speed (LZ4 [4], [31], [32], [37]).
- The compression speed is improved by massive pipelining or parallelization (systolic arrays) [24] of the match searching mechanism [38].
- There is a direct proportion between the compression ratio and the size of a compression dictionary. However, most of the mentioned implementations use (FPGA) embedded memory blocks (kilobytes in size) rather than external memory [8] such as DRAM (Dynamic Random Access Memory) or SRAM (Static Random Access Memory) chips.
- Compression dictionaries use three fundamental approaches: CAM (Content Addressed Memory) [39], hash table [40], and small (shift) register array for stream operating implementations [9], [17], [32], where the dictionary stores a few processed data words.
- Many implementations have small (size of kilobytes on average) input/output buffers optimized towards a block-oriented compression that makes them suitable for IP packet oriented compression [31].

A representative example of a hardware implementation of a lossless compression algorithm has the following features: It is based on LZ77 with massive parallelization of a match search mechanism with particularly small data/compression dictionary buffers.

A. LZ77 PRINCIPLES AND THE IMPACT OF THE DICTIONARY

LZ77 is a universal compression algorithm that is asymmetrical (the compression requires more time or resources than decompression) and single pass (data to be compressed are processed only once). LZ77 is a fundamental lossless compression scheme used in many further algorithms such as DEFLATE [41] or GIF (Graphics Interchange Format). The technique of the ''Sliding Window'' [6] for searching match candidates is used by the LZ77 algorithm (see Fig. 1).

The sliding window is usually divided into a search buffer (a dictionary) and a look-ahead buffer. The longest found prefix of the look-ahead buffer starting in the search buffer is encoded as a triplet **(i, j, X)**, where **i** is the distance of the beginning of the found prefix from the end of the search buffer; **j** is the length of the found prefix; and **X** is the first character after the prefix in the look-ahead buffer. The size (and the architecture) of the dictionary has a great influence on the compression ratio. A larger or better organized dictionary improves the compression ratio of the implemented compression algorithm because of the increased probability of finding a match over the larger sliding window [5].

There are three most common architectures of a dictionary. We would like to summarize their advantages and disadvantages from the perspective of their suitability for IP packet compression.

1) SHIFT REGISTERS FOR STREAM OPERATING IMPLEMENTATIONS

This type of a dictionary focuses on maximum performance in terms of operating frequency and the implementation architecture is carefully designed to process data in a (deep) pipeline to achieve maximum throughput. This seems to be an optimal solution for IP packets aware compression (a continuous stream of IP packets) with minimal latency [9], [32], but the compression ratio is quite low compared to

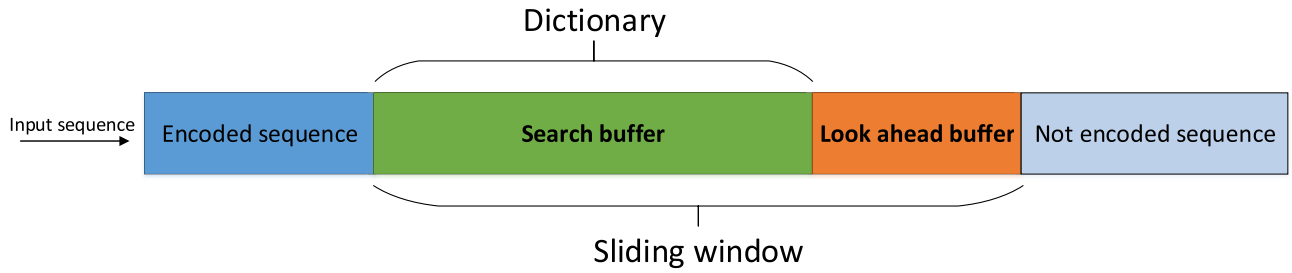


FIGURE 1. LZ77 sliding window technique. [6].

other approaches. The depth of the pipeline limits the sliding window of the dictionary to several words of processed data. The dictionary content does not have to be initialized because the match is only being searched in the pipelined data.

2) CONTENT ADDRESSED MEMORY (CAM)

The CAM based approach utilizes data spacial locality where the dictionary can accommodate more entries if the processed data are highly repetitive. The disadvantage is that each CAM memory cell is usually implemented by flip-flops and requires its own comparator for searching a match, requiring many logic resources. Therefore the enormous logic consumption consequently slows down the entire design by reducing the maximal operating frequency of such design [42].

Several techniques [42], [43] were introduced to improve some design properties such as reducing the amount of (some) required logic resources via the usage of other design primitives like embedded memory block. Another disadvantage is a latency of five clock cycles for a search operation.

On the contrary, a CAM based dictionary could easily be initialized via a dedicated reset/clear input of these flip-flops. Overall, we found the usage of CAM based dictionary not viable for our specific case requiring low-latency operations.

3) HASH TABLE

The hash table principle became popular when fast, modern compression algorithms (LZ4 [37], LZO [44]) appeared. The common idea of these algorithms is improving the throughput by increasing the width of the processed data word (the word width is 32 or 64 bits to match the ALU (Arithmetic Logic Unit) register width in modern processors [45]). These word widths are too large to be used as a direct address to the dictionary (the dictionary will have 4 gigabytes for the 32-bit word width). The CAM technique will make these algorithms slower [46] but fairly large dictionaries became required for a decent compression ratio. This led to implementing the dictionary as a hash table [1], [3]. The important features of a hash table implementation are the following:

- The hash algorithm can be extremely fast (just a constant multiplication in LZ4 [31], [47], the result is trimmed to an appropriate number of address bits to match the dictionary size).
- Produced hashes can collide with each other reducing the compression ratio a little (but saving memory required for the dictionary).
- Dictionaries are usually implemented in embedded memory blocks. In our particular example, the used Xilinx BlockRAMs are RAM based blocks with densities of 36 kilobits [48]. The content (a dictionary) in embedded/DRAM memory cannot be cleared in a single clock cycle [48], [49] like flip-flop (SRAM) based memory. This embedded memory block design is a trade-off between the memory capacity and the number of transistors required for the memory cell matrix [48].
- IP Packet optimized designs require clearing the entire dictionary before each run of the implemented compression algorithm (each IP packet is considered as one block).

B. REQUIREMENTS FOR THE DICTIONARY DESIGN

The requirements are set with emphasis to the particular use case: the IP protocol packet compression accelerators implementing LZ77 algorithm.

- The dictionary design should be suitable for IP packet compression (block compression oriented).
- The maximum payload will be 9 kB (the maximum size of a jumbo packet) [50].
- 10 Gbps throughput requirement leads to a 64-bit datapath clocked at 156.25 MHz at least because a design with 8-bit datapath will require a 1.25 GHz system clock which is significantly above the FPGA limits.
- The time required for loading the processed data from the buffer is 1150 clock cycles in the worst case.
- The dictionary size should be in the range of 1k–16k of entries (larger dictionary makes no sense compared to input/output buffer size).
- The dictionary will be implemented as a hash table. Therefore, we have to deal with the problem of potentially slow (re-)initialization.

The problem to be solved: the dictionary will be implemented as embedded/DRAM based memory. We have to find an efficient method (in terms of time) for (re-)initialization of the dictionary content. The efficiency in terms of logic resources can lead to a trade-off with a time sub-optimal solution.

FIGURE 2. Common phases of a hardware implemented compression algorithm.

C. COMPUTATION TIME OF A COMPRESSION HARDWARE BLOCK

A general (hardware) implementation of a compression algorithm has several phases where most of them are not originally related to the compression itself. However, they are needed for proper operation of the compression block. Some of these phases (see Fig. 2) can overlap each other because they were implemented [1], [3], [17], [26] in a smart way. Further details about each phase follow:

1) LOADING DATA INTO AN INPUT BUFFER

The time required for storing data into the input buffer is dependent on the type of application, and on the size and throughput of the respective buffer. The (maximum) required time [51] can easily be determined as a ratio between the throughput and the size. This phase can run in parallel with the next phase (initialization); however this phase is essential for proper operation of the compression block.

2) COMPRESSION BLOCK INITIALIZATION

The initialization of the compression block is intended to set-up default values of design registers or any other data structures like a compression dictionary, acquiring the size of the data, etc.

Despite the fact that the initialization phase can run in parallel with the data loading phase, the computational time of this phase is heavily affected by the overall architecture of the particular design. Initialization of complex data structures (used by a compression dictionary, for example) could easily be more time consuming than the data loading phase.

3) COMPRESSION

The most important phase is the compression itself, which is supposed to search for matches in a compression dictionary and encode respective output with particular examples of LZ based algorithms [38]. The maximum computation time can be estimated as the ratio between the input buffer size and the respective throughput of the compression phase. The computational time could be lower than the time of the initialization phase in our particular case (see Section III-B).

4) SAVING COMPRESSED DATA TO AN OUTPUT BUFFER

This phase is intended to store the compressed data into an output buffer. This functionality is usually implemented in the compression phase, therefore, these phases can overlap. In certain situations, the compression ended, but some data are still not copied to the output buffer. It is obvious that the computational time will be low.

D. SUMMARY

The conclusion is quite simple – calculating the overall latency is not simple, because it involves latencies of some other phases besides the compression phase as the primary function. The time (latency) required for transferring the processed data to/from input/output buffers has the same lower bound asymptotic complexity as the compression itself, the $\Omega(n)$. The question is, which compression dictionary architecture can match or decrease such lower bound asymptotic complexity, especially when the required dictionary can be larger than the buffers [5]?

IV. STATE OF THE ART – EVALUATED METHODS FOR INITIALIZING A DRAM BASED MEMORY STRUCTURES

We have selected a hash table for implementing a dictionary for a lossless compression algorithm. The choice has been made based on the analysis in the previous chapter. We are looking for a design for IP packet compression based on LZ77 with a minimum throughput of 10 Gbps per implemented block for applications in 10 gigabit ethernet networks. We put emphasis on the latency of the dictionary (re-)initialization phase.

We assume optimizations and techniques introduced by modern fast lossless compression, such as LZ4, can improve the ratio between logic gates count and throughput. This might allow implementing multiple compression blocks in a single FPGA. In the following sections, we will discuss three alternative techniques suitable for the hash table (implemented using BlockRAMs) based compression dictionary architecture. These techniques can be used in other architectures that are also BlockRAM based.

A. LINEAR PASSAGE APPROACH

The linear memory passage is a fundamental method for initializing memory to a default (constant) value [7].

The fundamental part is a counter with the same width as the memory address vector, thus all addresses are generated (including other control signals like write enable) for the (re-)initialization purpose. The data input port (vector) is multiplexed by the default value during the (re-)initialization process.

Advantages of the linear passage method are a simple and straightforward design and low resource requirements.

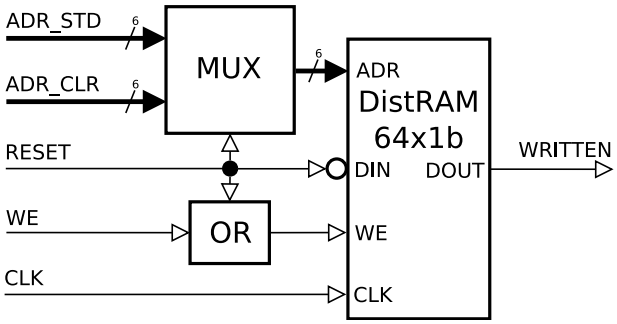


FIGURE 3. Architecture of the "Elementary Block". [5].

Disadvantages are that the initialization process requires a lot of clock cycles to pass through all addresses and that the adder used by the counter has a long carry chain that limits the maximum design frequency. These two disadvantages increase the latency of the design. This method is the only suitable approach for clearing a high density memory (with millions of entries) such as entire (external) DRAM [52] chips.

B. FLIP FLOP BASED APPROACH FOR A STATUS REGISTER

The second approach [53] uses a flip-flop based status register file, where each flip-flop preserves a single-bit wide flag indicating the status of the related memory cell (written or not written). When a memory read occurs before a write operation, the memory output will be multiplexed to the default read value.

Advantages of the flip-flop based approach are a higher operating frequency than with the linear passage and the latency of one clock cycle, because all flip-flops can be effectively cleared in parallel. Disadvantages are the exponential growth of required resources with respect to the memory address vector width. When a large memory is used, the operating frequency will drop significantly due to FPGA routing and synthesis issues.

C. DISTRIBUTED MEMORY BASED APPROACH FOR A STATUS REGISTER

We proposed an approach [5], [54] that combines previous approaches to get as many advantages (like the same asymptotic complexity [51] requiring less FPGA resources than the flip flop based approach) and to mitigate as many disadvantages from both techniques. The idea is to use an alternative way of storing data in an FPGA-based design instead of ordinary flip-flops, using the distributed memory block [55] in our particular case. This approach is comparable to the LVT (Live Value Table) [56], [57], where the idea is to split the design into two parts. Each part uses a different type of memory (flip-flop & BlockRAM) instead of a single type (flip-flop).

The flip flop based array of single-bit wide flag registers is split into small segments with the same size as a single distributed memory block (64-bits for the Xilinx 7-series architecture). The distributed memory block based design is divided into two parts: the "Elementary Block" (EB) and the "Address Control Logic" (ACL).

1) ELEMENTARY BLOCK

The EB (see Fig. 3) is composed of one distributed memory block with the size of 64 individual bits (the maximum size for a single 6-input LUT (Look-Up Table) [55] implementation). The distributed memory block has to be cleared (initialized) by the linear passage approach requiring 64 clock cycles. The linear passage approach also requires a multiplexer for switching address vectors between the standard and initialization mode (selected via the reset signal). The standard address vector input of the EB is the last (lowest) six bits of the address range for the status register. The second (initialization mode) address vector input is dedicated to the logic of the linear passage of the ACL block. The "Written" signal represents the information indicating whether a particular memory cell had a write request and the related record in a dictionary contains valid data. The default value for the initialization of the "Written" signal is logic zero (therefore, all bits in the distributed memory block).

2) ADDRESS CONTROL LOGIC

The ACL architecture (see Fig. 4) shows four individual parts of the status register:

- "CNT M64"– The 6-bits wide counter (counting as modulo 64) for generating the address vector for the elementary blocks while the initialization mode is active.
- "SPLIT"& "EB SEL" – The "SPLIT" block splits the address vector input into the upper and lower part. The lower part has six bits to match the address range of the EB. The upper part is forwarded to "EB SEL" implementing an address decoder. The address decoder is generating the "Chip Enable" (CE) encoded as one-hot value for each EB in the design. Consequently, only one EB is selected at each clock cycle.
- Output Masking – Only the output of the chosen EB is passed to the "Written" signal via AND/OR logic gates. Outputs of rest EBs are masked.

V. A QUANTITATIVE ANALYSIS

This section presents a discussion about which design parameters have an impact on such compression dictionary design. The general observed properties for a hardware accelerator implementing a compression algorithm are:

- compression ratio,
- throughput,
- latency,
- operating frequency,
- amount of logic resources.

As stated earlier in Section III-A, the compression dictionary size significantly affects the respective compression ratio. In the case of a hardware accelerator, input/output buffers and compression dictionary are often implemented using embedded memory blocks (called BlockRAM/M9K

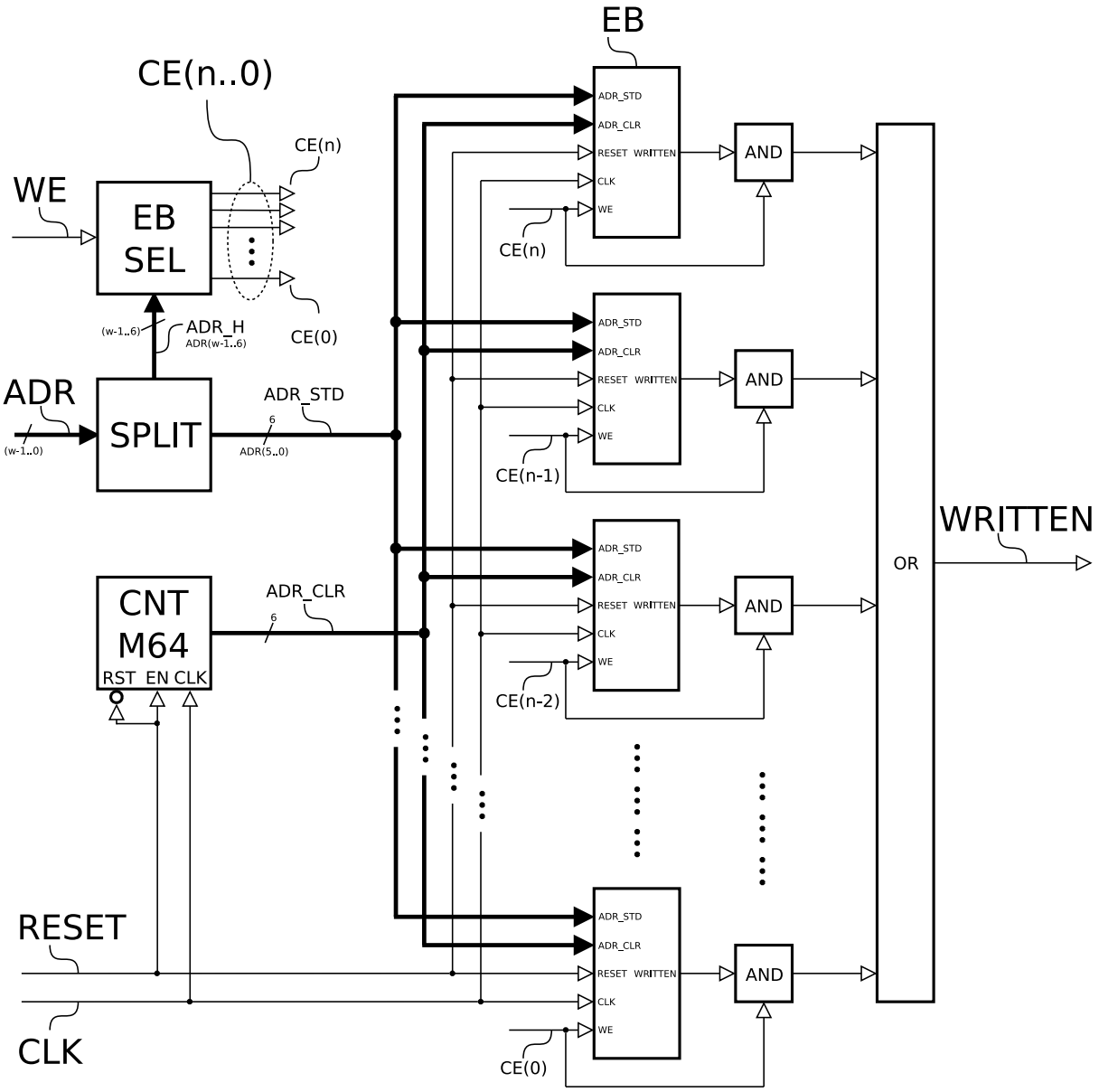


FIGURE 4. Architecture of the "Status Register" based on EBs and the ACL. [5].

in the case of Xilinx and Intel/Altera FPGAs, respectively). Therefore the sizes of (the example) dictionaries in our case are simulated by modifying the **"W"** parameter [5], which stands for a memory address bus width. The relation between memory capacity (which accommodates a compression dictionary, for example) and the address width can be expressed as formula (1). The formula assumes a digital system. Therefore, the address is a binary number, and the number of entries is also a power of two (cases where an address range does not match a number of entries are not considered because they are rare in digital design).

$$W = log_2\left(\frac{Memory\ capacity}{Memory\ entry\ width}\right) = log_2(Number\ of\ memory\ entries) \quad (1)$$

The operating frequency and the amount of used logic resources are also affected by the used FPGA type (technological parameters).

A. ARCHITECTURE INFLUENCE

The particular architecture of such a dictionary affects the remaining observed properties, which usually depend on the used FPGA. In general, the architecture complexity affects the number of logic resources needed for an implementation in hardware and properties as such the latency required for clearing them off. It is assumed that more complex architectures will require a higher amount of logic resources. There is no such assumption on (theoretical) architecture latency.

These logic resources have to be placed ("floorplanned") [58] in the 2D space of the integrated circuit and

interconnected by wires. It is clear that a higher number of logic resources must occupy a larger area in a silicon. Therefore the higher the area occupied, the longer the respective wires will be, and they will contain more junctions.

Consequently, the increased wire lengths and increased logic gate output load will increase the signal propagation delay, and thus the maximum operating frequency of such accelerator will be limited by the ''slowest'' signal [59]. The reduced frequency will also reduce the respective throughput of a compression accelerator.

In the particular case of a hardware compression accelerator, it is a common attitude to design the accelerator to minimize the number of (system) clocks; only one clock signal is being used in most cases. Such a compression accelerator usually consists of several smaller ''building blocks'', for example: input and output buffers, compression dictionary, match search unit [38], encoding unit, etc. It is obvious that the lowest frequency of these ''building blocks'' will be the resulting operating frequency of the particular accelerator. Therefore, the motivation is to design an accelerator where all blocks are close to each other in terms of frequency to improve the overall accelerator frequency, thus improving the performance.

From this perspective, the architecture used for implementing a compression dictionary has an impact on the amount of required logic resources, thus frequency, and thus the overall accelerator performance. In case the respective architecture saves a lot of logic resources (against previously used compression dictionary architecture), it will increase the overall accelerator frequency. Despite the fact that there is no direct influence on the compression ratio, a more resource-efficient architecture could allow hardware designers to implement a bigger dictionary with the same amount of resources as it was originally, using to achieve a better compression ratio [5], [38]. On the other hand, the extra logic resources can also be used for implementing multiple accelerators with a higher overall throughput while keeping the same (constrained) area of an integrated circuit.

B. ESTIMATIONS

The amount of logic resources needed is affected by the **"W"** parameter and the capabilities of the used FPGA for this estimation. The frequency parameter is usually indirectly proportional to the amount of logic resources. The latency of initialization of a compression dictionary is architecture specific. The compression ratio parameter cannot be estimated in this particular case.

1) XILINX CONFIGURABLE LOGIC BLOCK (CLB) ARCHITECTURE

As an abbreviation, FPGA is quite self-explanatory. It is a giant array of fundamental blocks (CLBs [55] in the Xilinx case) interconnected by a matrix of wires (FPGA fabric) which can realize a desired logic function. This principle has been shared among all major FPGA vendors. CLB can be divided further into two slices. Each slice consists of four LUTs and eight flip-flops (registers) plus an interconnection fabric.

The most common LUT width is 6 bits in most FPGAs (LUT6). However, some older FPGAs had 4-bit LUTs only. Some LUTs have available alternative use cases such as distributed memory blocks or wide shift registers. The 6-bit LUT can usually be split further into two 5-bit LUTs, which are more suitable for implementing less complex logic functions. It seems a wider LUT is not going to be introduced by FPGA vendors in the near future. Not all elements in a CLB have to be utilized.

2) LINEAR PASSAGE APPROACH

As stated in the above text, the approach uses a counter (counter width is equal to the **"W"** parameter) generating all addresses (4), which is connected to the BlockRAMs address input via a multiplexer. The multiplexer switches the normal and reset operation addresses. The amount of logic resources can be estimated [59] in the following way: the counter will require at least **"W"** LUT5s and **"W"** registers. The multiplexer will require **"W"** LUT5s only for the implementation. Therefore the linear passage approach will likely require several LUTs (2) and registers (3) in total.

$$LUT_{Linear} = 2 * W \tag{2}$$

$$REG_{Linear} = W \tag{3}$$

$$Latency_{Linear} = 2^{W}. \tag{4}$$

3) FLIP-FLOP BASED APPROACH

This approach requires generating an array of registers equal to the number of entries in a dictionary (2^W in our case). An address decoder is also required to select the individual register during the operation. Therefore at least one LUT will be required for each register resulting in estimations (5) and (6)

$$LUT_{Flip-Flop} = \lceil log_6(W) \rceil * 2^{W} \tag{5}$$

$$REG_{Flip-Flop} = 2^{W} \tag{6}$$

$$Latency_{Flip-Flop} = 1. \tag{7}$$

4) DISTRIBUTED MEMORY BASED APPROACH

The numbers of required LUTs and registers are expressed in formulas (8) and (9); thus they can be described as a difference between the Distributed and the Flip-Flop based approach:

- Each EB replaces 64 flip-flops; therefore, the total number of EBs is 2^{W-6}.
- Individual address decoder for each EB is less complex because it decodes 6 fewer address bits, which are omitted by the SPLIT function.
- Each EB consists of 7 LUTs.
- The output masking function uses the 2-input AND logic gates, which can be packed into the OR logic gate ($log_6(W-6)$ originally).

TABLE 1. Estimated properties for all techniques depending on the "W" parameter.

LUTs										
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Linear	12	14	16	18	20	22	24	26	28	30
Flip-Flop	64	256	512	1024	2048	4096	8192	16384	32768	65536
Distributed	15	22	38	71	135	263	519	1159	2311	4615

Registers										
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Linear	6	7	8	9	10	11	12	13	14	15
Flip-Flop	64	128	256	512	1024	2048	4096	8192	16384	32768
Distributed	6									

Latency [Cycles]										
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Linear	64	128	256	512	1024	2048	4096	8192	16384	32768
Flip-Flop	1									
Distributed	64									

- CNT64 requires only 6 LUTs and 6 registers in total. However, the counter itself might be replicated several times by a synthesis tool (principle of locality [60]).

Therefore, the number of logic resources for the Distributed approach can be expressed in the following way:

$$\begin{aligned} LUT_{Dist} &= \lceil log_6(W-6) \rceil * 2^{(W-6)} + 7 * 2^{(W-6)} \\ &\quad + 2 * \lceil log_6(W-6) \rceil + 6 \\ &\approx \lceil log_6(W-6) \rceil * 2^{(W-6)} \qquad (8) \\ REG_{Dist} &= 6 \qquad (9) \\ Latency_{Dist} &= 2^6 = 64. \qquad (10) \end{aligned}$$

5) ESTIMATION DISCUSSION AND SUMMARY

We estimated several properties (latency, resource utilization) of respective techniques (see Table 1) by using formulas mentioned in the previous section. Therefore we discussed our expectations (design complexity and frequency) for individual techniques.

The Linear Passage technique requires the least amount of logic resources and also has the potential to reach high frequencies. However, the latency will grow exponentially, and this prevents this technique from being suitable for compression dictionaries unless a large external memory is used (memory is initialized only once during a power-up phase, for example), and the latency of initialization is not an issue.

Therefore, the Flip-Flop and Distributed based techniques were found suitable for implementations requiring dictionaries to be (re-)initialized before each run of a compression accelerator where the low latency is one of the requirements.

The Flip-Flop based approach having the best latency of one clock cycle is redeemed by enormous logic consumption (both LUTs and registers), which grows exponentially. The Distributed memory based approach seems to have the same advantage (constant latency) and disadvantage (the number of logic resources growing exponentially), however, the amount of required logic resources is decreased by a factor of 64.

TABLE 2. Brief estimations and expectations.

Technique	Linear	Flip-Flop	Distributed
Resources	Minimal	Enormous	Moderate
Design complexity	Low	Moderate	High
Frequency	High	Moderate	High
Latency	Enormous	Minimal	Fair
Suitability[1]	Low	Fair	Great

Note 1: For hash table based compression dictionary architecture.

On the other hand, the latency is increased by the same factor to 64 clock cycles.

Thus the Distributed memory technique consumes less logic resources than the Flip-Flop technique, and it is assumed the frequencies will be higher in favor of the Distributed memory technique. The respectively increased latency will not be an issue in our case because the compression dictionary initialization could run in parallel with the loading data phase (see section III-C1). We assume this phase will take more clock cycles than the compression dictionary initialization phase for both Distributed and Flip-Flop techniques.

The general expectations for all approaches are summarized in Table 2. We assume the combination of latency, logic resources consumption, and frequency in the "sweet spot" range [5] (**"W"** between 6 to 15) will favor the Distributed technique. It is assumed that the final properties and results of individual techniques will change after implementation due to the various optimization used by synthesis tools [58], [59], such as resource sharing [61], [62], logic duplication, or register balancing [63] may be applied.

VI. THE DISADVANTAGE OF THE PREVIOUSLY USED METHODOLOGY

The initial set of measurements [5] was performed on a single computer with the Xilinx ISE toolset using the same initial conditions as those discussed in the following section VII-B. We used the "Random PAR" (Place & Route) mode in ISE, which allows to synthesize and PAR the design without setting-up physical constraints such as FPGA pins assignment. The creation of timing constraints such as a clock period is not affected by this mode. The disadvantage of this procedure is that designs (representing different approaches) with the same value of the **"W"** parameter have different pin placements. The observed randomness of the pin placements may affect the process of the synthesis, the PAR, and the STA in the final consequence. This might make an advantage (pins can be placed closer to an evaluated design) for one approach and penalize other approaches. This led us to prepare a new workflow to prevent this issue and to be supported by both ISE and Vivado.

VII. OUR APPROACH

We decided to choose a universal FPGA (in term of support in the Xilinx tools) to perform objective measurements. Thus we have selected the Xilinx Virtex-7 690T (XC7V690T-2FFG1158) due to its size and being a representative

TABLE 3. Computer systems used for the evaluation.

System/Platform	CPU	C/T[1]	RAM
Sun SunFire X4150	2x Intel Xeon L5420	8/8	64 GB
Supermicro H8DME-2	2x AMD Opteron 2382	8/8	64 GB
HP DL360 G6	2x Intel Xeon E5530	8/16	64 GB
Dell PowerEdge T620	2x Intel Xeon E5-2640	12/24	48 GB
ASRock B450M PRO4	1x AMD Ryzen 7 1700X	8/16	64 GB

Note 1: Cores/Threads

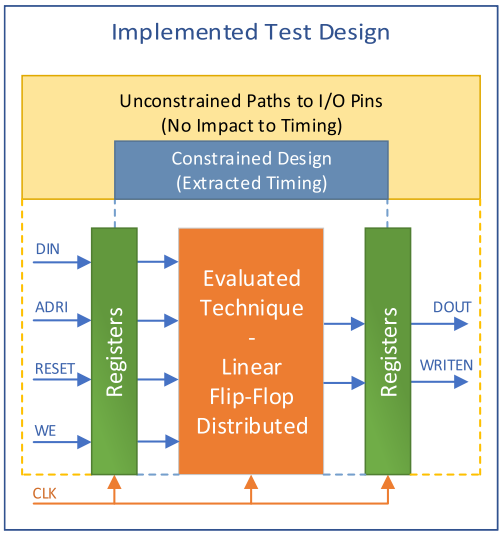


FIGURE 5. Architecture of the design wrapper.

example of the Xilinx 7-Series FPGAs. The 7-Series architecture is the basis for the latest Xilinx FPGAs, such as the UltraScale(+) [64] platform. An advantage of Virtex-7 is that it is supported by both Xilinx development toolsets: the Xilinx ISE (Integrated System Environment) and the Xilinx Vivado in latest versions (version 14.7 and 2017.2 respectively). The reason to chose both tools is that the research started in 2015 (on Virtex-6 platform) when Vivado wasn't recognized as a "mature" product. Usage of the ISE also allows us to evaluate and compare the synthesis process against Vivado without violating Xilinx ISE license [65]. All tests were performed on several computer systems (see Table 3) representing trends in the last decade.

The synthesis process uses randomized algorithms; however, each system has it's own "seed". Therefore, we included the information that the used computer systems were different (and their respective configuration).

All systems have been following the requirements for Xilinx ISE [66] and Xilinx Vivado [67].

A. EXPERIMENTAL SETUP

We developed the design implementing all three mentioned approaches (linear passage, flip-flop array and distributed memory based approach) with a unified interface (see Program 1) in VHDL language.

The implemented approach and some other parameters (such as LUT size and the address vector width **"W"**) are set by generic constants. We assume the following design properties:

- Xilinx 7-Series architecture with 6-input LUTs.
- A memory cell width of 36 bits (one of the native BlockRAM widths [48], intended for simulating a 32-bit memory pointer like software version of LZ4 does. This dictionary cell width has also been used elsewhere [16]),
- **"W"** parameter stands for the intended memory address vector width and defines the memory capacity ($36*2^W$ bits).

As a precaution, we designed a test "Wrapper", which embeds an evaluated technique into a register array. The architecture (see Fig. 5) of the "Wrapper" will prevent the paths between physical FPGA I/O pins and the respective logical signals to have any impact on timing analysis. Therefore any path length could be virtually unlimited. Thus the technique designs can be substantially dense and floorplanned almost anywhere in an FPGA.

Program 1 A Unified VHDL Interface for All Implementation Types. [5]

```
entity top is port (
clk, reset, we: in std_logic;
adri: in  std_logic_vector(W-1 downto 0);
din: in  std_logic_vector(35 downto 0);
dout: out std_logic_vector(35 downto 0);
written: out std_logic);
end top;
```

B. ADDITIONAL SETTINGS FOR SYNTHESIS TOOLS

We changed some of the parameters from the defaults for Xilinx ISE and Xilinx Vivado to force the tools to favor the design speed instead of area. Some additional parameters were set to overcome some of the synthesis issues, such as a memory overflow.

1) Xilinx ISE [63]
   - Synthesis – Optimization Effort = Fast (Synthesis consumes less memory allowing synthesis of larger designs without a crash of Xilinx XST).
   - Synthesis – Register Balancing = Yes
   - Map – Register Duplication = On
   - Map – Allow Logic Optimization Across Hierarchy = Yes
2) Xilinx Vivado (Strategies) [68]
   - Synthesis – PerfOptimized_High
   - Implementation – Performance _ExplorePostRoutePhysOpt

C. OUR TEST METHODOLOGY WITH THE LINEAR PASSAGE APPROACH AS AN EXAMPLE

The new workflow improves the original workflow [5] by removing of the observed randomness of physical constraints via fixing the used constraints across all implemented approaches. The new workflow is depicted in Fig. 6, and some of these phases will be described in an example (the Linear Passage technique in our case) in a more detailed way.

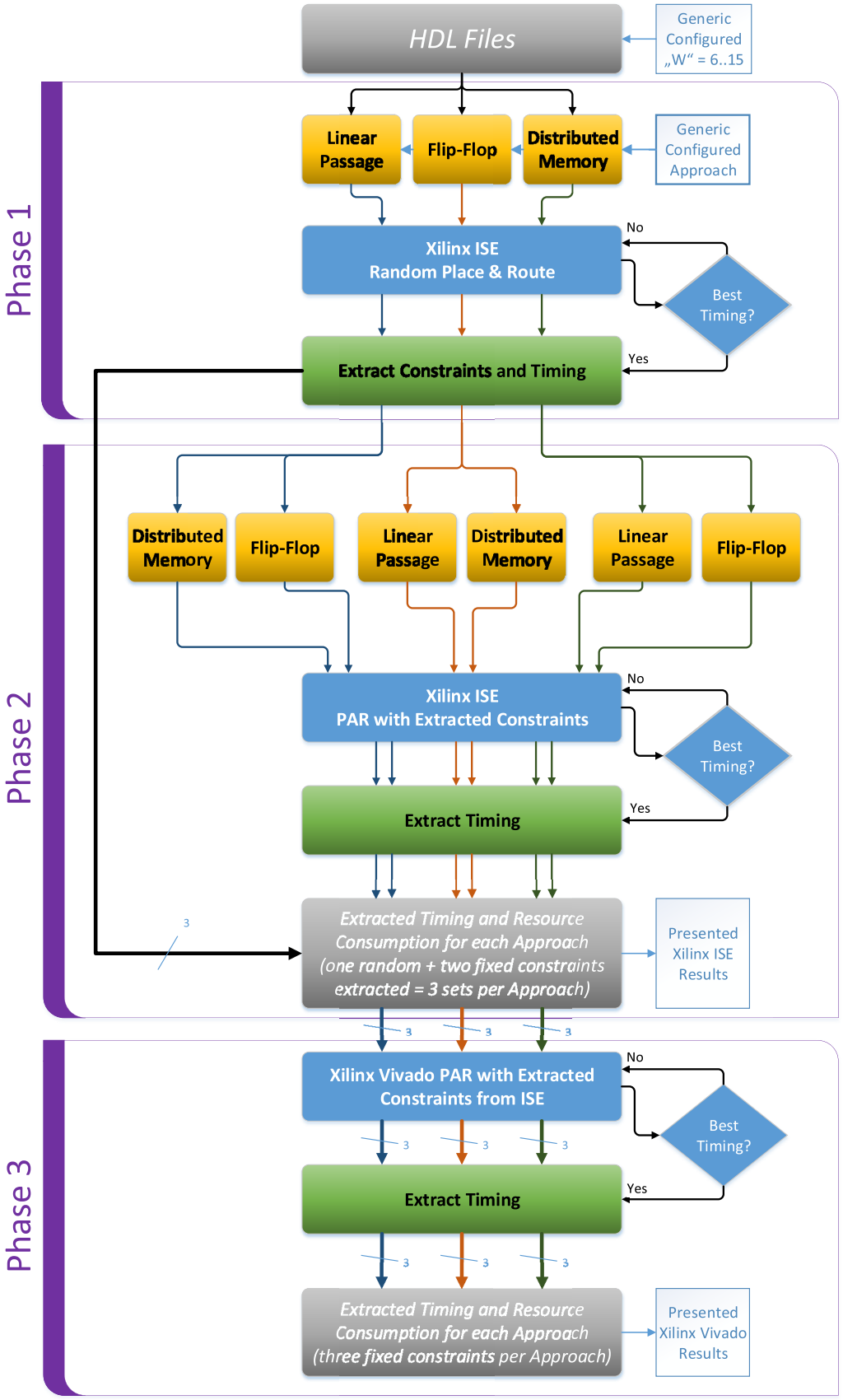


FIGURE 6. The test flow used for experimental measurements.

The respective workflow paths for Linear Passage, Flip-Flop, and Distributed methods are highlighted with colors (blue, orange, and green).

1) PHASE ONE

The source files (HDL and initial timing constraints) are generated for implementation in Xilinx ISE. The best-reached clock period is extracted from the STA (Static Timing Analysis) report of the fully (randomly) routed design. The clock period is decreased by 0.05 ns (the smallest available step recognized by the Xilinx ISE) for the next iteration. The previous step is repeated until timing errors occur. The physical constraints (pin placement) and other reports (such as STA and PAR resources utilization report) are extracted at the end of phase one.

2) PHASE TWO

Phase two uses the extracted physical constraints (of the Linear Passage) to perform the implementation of the two remaining approaches (Flip-Flop and Distributed Memory based Status Register). The search for the best timing is the same as in phase one. The result of phase two is a set of three designs representing all approaches with their timing & physical constraints and resources utilization. These best frequencies (of the measured designs) are averaged over all approaches to reduce the influence of the random pin placement.

3) PHASE THREE

The collected ISE constraints are converted to a constraints format suitable for Xilinx Vivado. The designs are evaluated in the same manner as in phase one and phase two in Xilinx ISE.

D. COLLECTED DATA SETS

Nine data sets were collected after all three phases in the Xilinx ISE. Each approach had its own subset of three measurements:

- Linear Passage
  - Native constraints set (randomly generated)
  - Flip-Flop constraints set (fixed)
  - Distributed memory constraints (fixed)
- Flip–Flop
  - Native constraints set (randomly generated)
  - Linear Passage constraints set (fixed)
  - Distributed memory constraints (fixed)
- Distributed Memory
  - Native constraints set (randomly generated)
  - Linear Passage constraints (fixed)
  - Flip-Flop constraints set (fixed)

Each measurement had 10 fully routed designs with the STA report ranging the addresses with the **"W"** parameter from 6 to 15 bits. The measured datasets from the Xilinx Vivado had the same structure as the Xilinx ISE datasets.

TABLE 4. Properties of measured designs like logic gates count, frequency, etc. depending on the "W" parameter.

Linear passage approach										
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Latency [Cycles]	64	128	256	512	1024	2048	4096	8192	16384	32768
Xilinx ISE										
Slices	70	92	98	90	103	84	111	112	114	109
LUTs	159	164	170	174	197	201	206	210	215	219
Registers	86	88	90	92	58	60	62	64	66	68
Frequency [MHz]	402	402	402	402	593	590	588	585	582	578
Xilinx Vivado										
Slices	18	25	26	27	26	36	36	49	52	60
LUTs	50	52	54	56	58	60	62	64	66	69
Registers	38	47	50	45	52	61	51	51	53	54
Frequency [MHz]	571	534	483	557	481	441	298	229	238	258

Flip–flop based approach										
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Latency [Cycles]	**1**									
Xilinx ISE										
Slices	92	134	157	304	577	932	2275	4506	10026	17503
LUTs	221	309	478	840	1544	3250	6495	13912	31747	57586
Registers	145	210	339	615	1073	2167	4147	8244	16461	32955
Frequency [MHz]	402	373	395	297	260	245	218	204	176	161
Xilinx Vivado										
Slices	42	69	121	234	418	792	1563	2978	5888	12342
LUTs	109	174	317	596	1095	2145	4237	8312	16584	33573
Registers	89	178	340	713	1361	2690	5316	10577	21025	42303
Frequency [MHz]	472	292	295	232	230	234	224	194	207	207

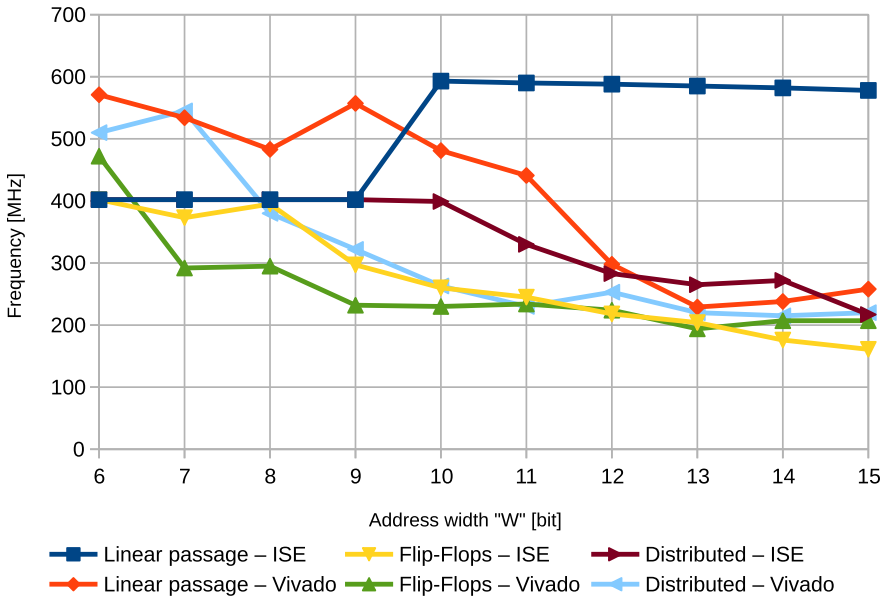
Distributed memory based approach										
Address width "W"	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Latency [Cycles]	**64**									
Xilinx ISE										
Slices	61	82	96	118	144	255	416	716	1382	2608
LUTs	51	58	71	96	145	245	439	839	1635	3235
Registers	87	94	107	132	145	242	435	820	1589	3126
Frequency [MHz]	402	402	402	402	399	330	283	265	272	217
Xilinx Vivado										
Slices	18	23	36	55	93	179	333	631	1206	2422
LUTs	142	158	189	255	372	605	1037	1991	3696	7491
Registers	16	32	62	123	245	494	986	1971	3941	7866
Frequency [MHz]	510	545	380	322	263	230	253	220	215	220

VIII. EXPERIMENTAL RESULTS

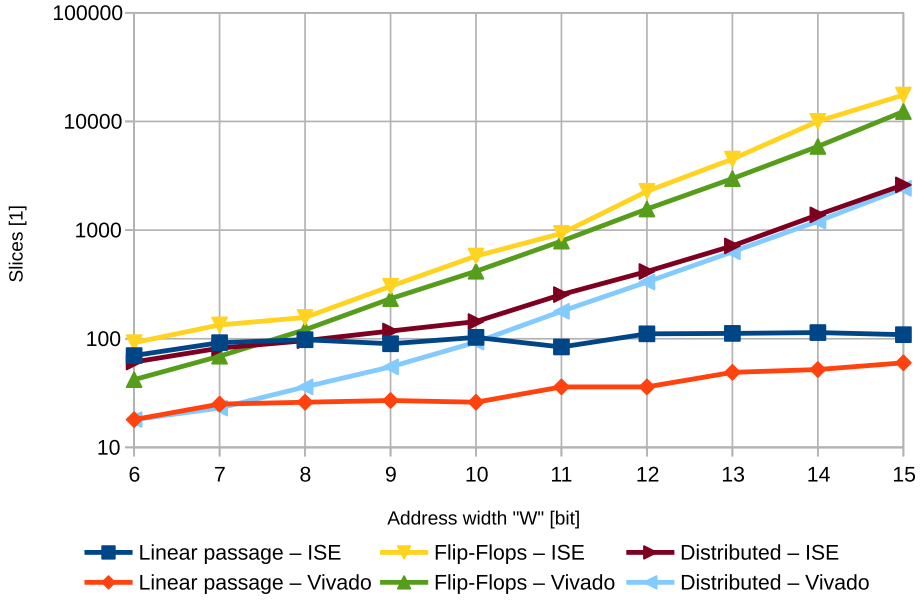
We evaluate and comment on the measured data (see Table 4) in terms of design speed, FPGA resources utilization, and suitability for compression dictionary design. These results are also visualised as graphs (Figures 7a, 7b, 8a, 8b).

A. LINEAR PASSAGE APPROACH

The Xilinx Vivado seems to be a better tool for implementing the linear passage approach in terms of chip area, thus consuming less FPGA resources in all cases. A synthesis or implementation issue appears to be in the Xilinx ISE. There is an unexpected "step" in the ISE frequencies, and results do not scale down in relation to the **"W"** parameter. The use of fast carry logic between neighboring slices might be the reason when no DSP48 [69] blocks are used



(a) Reached maximum frequency for all approaches (both ISE and Vivado).



(b) Slice consumption for all approaches (both ISE and Vivado).

FIGURE 7. Overall results, part I.

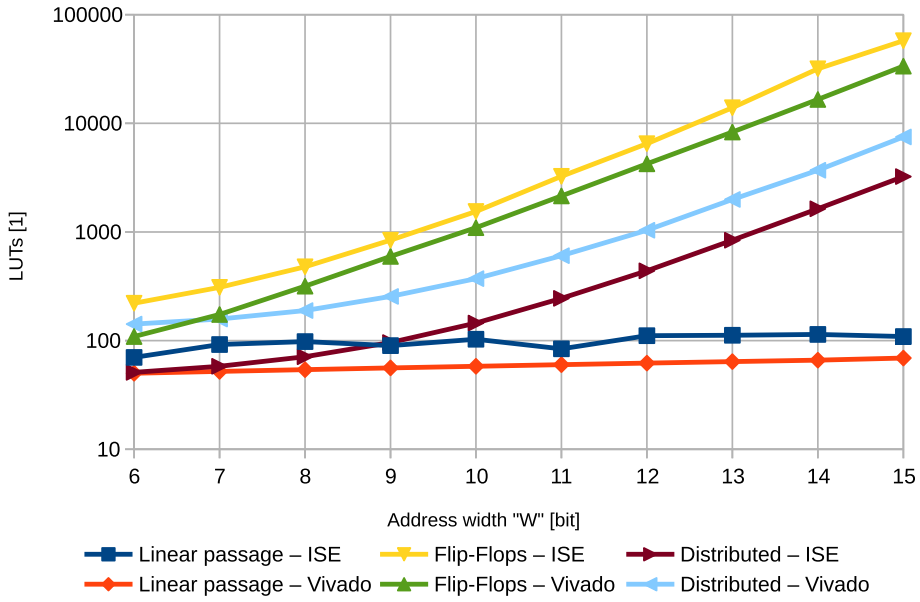
(the number of DSP48 blocks used is zero after PAR). However we were unable to determine the actual reason for the "step" even with a detailed analysis of respective floor-planned designs. It seems the implementation of such small designs in the such a large FPGA can lead to unpredictable results.

The initial results [5] showed that higher (better) frequencies were reached for the entire range while the graph curve is converging down for the Xilinx Vivado frequencies. This behavior needs to be further analyzed by performing the measurements on multiple computers.

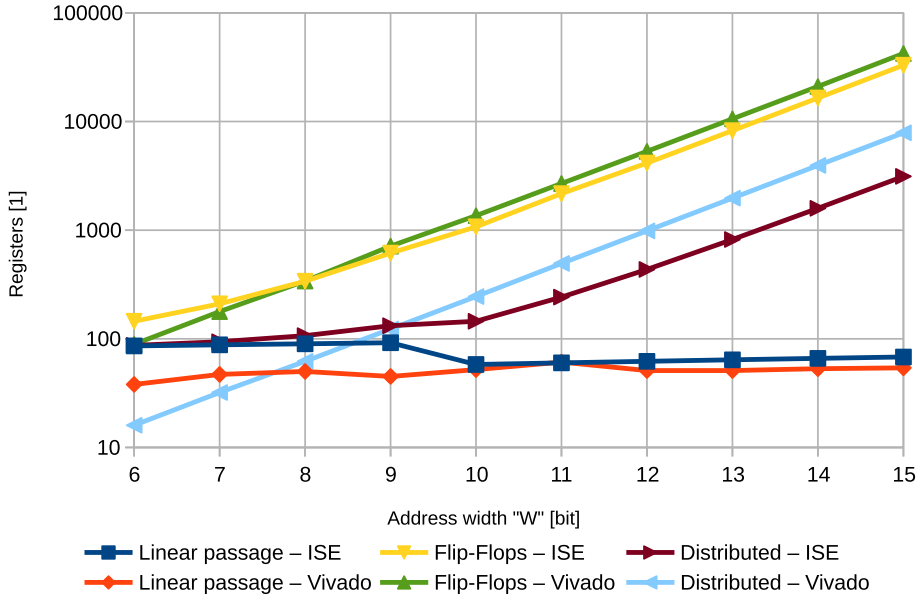
This approach is not suitable for larger dictionaries due to the enormous latency (growing exponentially) required for memory initialization. The approach is optimal in terms of used chip area.

B. FLIP-FLOP APPROACH

Implemented designs based on the flip-flop approach are comparable in terms of frequency for both Xilinx toolsets. The Xilinx ISE frequencies start on a lower point than Vivado, but convergence is slower compared to the Vivado results. The Xilinx Vivado is a better tool in terms of used FPGA resources saving 15%–55% of used slices with 32.5% on average. The solution of the flip-flop based status register is optimal in terms of latency (a single clock cycle, and it is constant for the entire range of the **"W"** parameter).



(a) LUT consumption for all approaches (both ISE and Vivado).



(b) Register consumption for all approaches (both ISE and Vivado).

FIGURE 8. **Overall results, part II.**

C. DISTRIBUTED MEMORY APPROACH

The last evaluation deals with the distributed memory approach of a status register. We prepared an additional table (see Table 5) to show the differences between the Xilinx ISE and the Xilinx Vivado results (see Fig. 9). The formula (11) used for calculating the differences follows

$$Difference = \left(1 - \frac{Vivado}{ISE}\right) * 100\ [\%]. \qquad (11)$$

The Xilinx ISE results show that the frequency start lower than Vivado for the first two smallest designs, but for the rest of the measured range, ISE is better than Vivado in terms of speed. On the other hand, the Xilinx Vivado has better results in terms of resource consumption over the entire range, but the advantage of Vivado is gradually diminishing (we assume the Xilinx ISE will be better for **"W"** parameter above the value of 15). There is no obvious winner in the end, because any single advantage of ISE or Vivado converges to zero while the **"W"** parameter rises (see Fig. 9).

D. SUITABILITY FOR A DICTIONARY DESIGN

We evaluated all approaches with respect to the requirements discussed in Section III-B. All approaches satisfy the requirement of the minimum design speed of 156.25 MHz. We have to exclude the linear passage approach due to its latency (it didn't meet the requirements, and it was too high compared

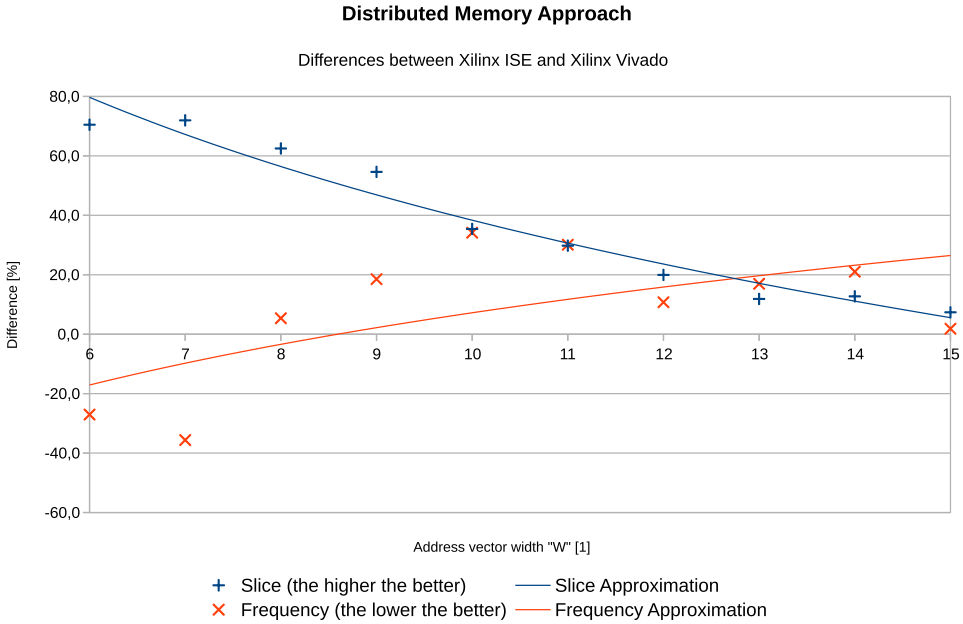


FIGURE 9. Differences for distributed memory approach.

TABLE 5. Differences between Vivado and ISE for distributed memory approach related to the "W" parameter – positive values represents the Vivado advantage.

Distributed Memory Approach										
"W"	6	7	8	9	10	11	12	13	14	15
Slices [%]	70.5	72.0	62.5	54.6	35.4	29.8	20.0	11.9	12.7	7.4
Frequency [%]	-27.0	-35.6	5.4	18.5	34.2	30.1	10.8	17.0	21.0	1.8

TABLE 6. Advantage of distributed memory approach compared to flip-flop for the Xilinx Vivado results.

FPGA Resources – Distributed Memory vs. Flip-Flop										
"W"	6	7	8	9	10	11	12	13	14	15
Area [%]	57.1	66.7	70.2	76.9	77.8	77.4	78.7	78.8	79.5	80.3

to the other two approaches). The flip-flop approach and the distributed memory approaches are comparable in terms of latency. The asymptotic complexity of both approaches is constant (1 respectively 64 clock cycles). The minor disadvantage of the distributed memory approach (the latency is slightly higher, but meets the requirements) is balanced by its better speed (higher frequencies are reached over the entire range), and less FPGA resources are consumed (see Table 6).

The average saving was 75% in terms of FPGA resources for the distributed memory approach. The issue of the flip-flop approach is the FPGA resources required for implementations. For example, the flip-flop approach will consume almost all resources in the Xilinx Virtex-7 690T FPGA for the **"W"** parameter equal to 20 (such design will support 1 million entries in a dictionary). The flip-flop approach cannot be used effectively with larger embedded memory such as the Xilinx UltraRAM feature [70], which increases the amount of FPGA embedded memory available by 600% from previous FPGA generations.

E. THE "DISTRIBUTED MEMORY" METHOD COMPATIBILITY WITH OTHER FPGA VENDORS

We considered four different FPGA vendors (Xilinx, Intel/Altera, Lattice, Microsemi/Actel) for compatibility with our method. Xilinx seems to have the most comprehensive support of LUT based "Distributed Memory" feature since the introduction of the very first Virtex/Spartan FPGAs [71].

Intel/Altera's FPGA support for the "Distributed Memory" feature varies across families. In general, modern and expensive FPGAs [72] do support the feature, while low-cost oriented and older families lack the support for the feature [73], [74]. Some FPGAs of Lattice and Microsemi/Actel also support the feature in certain more recent families [75], [76]. Due to the fact that these four vendors have 99% market share [77], our technique can be ported to nearly every modern FPGA.

IX. CONCLUSION

We presented a comprehensive analysis of three methods (linear passage, flip-flop, and distributed memory) suitable for initializing memory oriented data structures, including our distributed memory based approach. A performance comparison was performed in terms of the maximum reached operating frequency, FPGA resources consumption, and the requirements of the lossless compression dictionary design.

All approaches were measured over the range of hash table sizes suitable for IP compression devices. We presented the new test flow to support the Xilinx ISE and the Xilinx Vivado toolkits in the measurement process.

According to our methodology, the distributed memory approach shows the best combined performance against remaining techniques. This method is probably the only technique to satisfy the needs of future FPGA based

implementations of various compression algorithms where a larger embedded memory such as the Xilinx UltraRAM feature is used. The presented technique is compatible and can be ported to any modern SRAM based FPGA architecture.

REFERENCES

[1] J. Fowers, J. Y. Kim, D. Burger, and S. Hauck, "A scalable high-bandwidth architecture for lossless compression on FPGAs," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2015, pp. 52–59.

[2] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. New York, NY, USA: Springer, 2011.

[3] B. Sukhwani, B. Abali, B. Brezzo, and S. Asaad, "High-throughput, lossless data compresion on FPGAs," in *Proc. IEEE 19th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2011, pp. 113–116.

[4] T. Beneš, M. Bartík, and P. Kubalák, "High throughput and low latency LZ4 compressor on FPGA," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2019, pp. 1–5.

[5] M. Bartík, S. Ubik, and P. Kubalík, "A novel and efficient method to initialize FPGA embedded memory content in asymptotically constant time," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–6.

[6] P. E. Bender and J. K. Wolf, "An improved sliding window data compression algorithm based on the Lempel-Ziv data compression algorithm [magnetic recording]," in *Proc. Global Telecommun. Conf. Exhib. Commun. Connecting Future (GLOBECOM)*, Dec. 1990, pp. 1773–1777 vol. 3.

[7] S. Rigler, W. Bishop, and A. Kennings, "FPGA-based lossless data compression using Huffman and LZ77 algorithms," in *Proc. Can. Conf. Electr. Comput. Eng.*, Apr. 2007, pp. 1235–1238.

[8] S. Rigler, "FPGA-based lossless data compression using GNU Zip," M.S. thesis, Dept. Elect. Comput. Eng., Univ. Waterloo, Waterloo, ON, Canada, 2007. [Online]. Available: http://hdl.handle.net/10012/2692

[9] R. Mehboob, S. A. Khan, Z. Ahmed, H. Jamal, and M. Shahbaz, "Multi-gig lossless data compression device," *IEEE Trans. Consum. Electron.*, vol. 56, no. 3, pp. 1927–1932, Aug. 2010.

[10] R. Mehboob, S. A. Khan, and Z. Ahmed, "High speed lossless data compression architecture," in *Proc. IEEE Int. Multitopic Conf.*, Dec. 2006, pp. 84–88.

[11] I. Shcherbakov, C. Weis, and N. Wehn, "A high-performance FPGA-based implementation of the LZSS compression algorithm," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum (IPDPSW)*, May 2012, pp. 449–453.

[12] E. J. Leavline and D. A. A. G. Singh, "Hardware implementation of LZMA data compression algorithm," *Int. J. Appl. Inf. Syst.*, vol. 5, no. 4, pp. 51–56, 2013.

[13] 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.

[14] P. M. Parekar and S. S. Thakare, "Hardware implementation of lossless LZMA data compression algorithm," *Prog. In Sci. Eng. Res. J.*, vol. 2, no. 3, pp. 201–205, May-Jun. 2014.

[15] M. Morales-Sandoval and C. Feregrino-Uribe, "On the design and implementation of an FPGA-based lossless data compressor," in *Proc. Sociedad Mexicana Ciencias Computación (ReConFig)*, 2004, pp. 29–38. [Online]. Available: https://www.tamps.cinvestav.mx/~mmorales/research.html

[16] S. Naqvi, R. Naqvi, R. A. Riaz, and F. Siddiqui, "Optimized RTL design and implementation of LZW algorithm for high bandwidth applications," *Przegląd Elektrotechniczny Elect. Rev.*, vol. 87, no. 4, pp. 279–285, 2011.

[17] J. L. Nunez, S. Jones, and S. Bateman, "X-MatchPRO: A high performance full-duplex lossless data compressor on a ProASIC FPGA," in *Proc. Int. Workshop Intell. Data Acquisition Adv. Comput. Syst. Technol. Appl.*, 2001, pp. 56–60.

[18] 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.

[19] J. L. Nunez, C. Feregrino, S. Bateman, and S. Jones, "The X-MatchLITE FPGA-based data compressor," in *Proc. 25th EUROMICRO Conf.*, vol. 1, 1999, pp. 126–132.

[20] J. L. Nunez and S. Jones, "Lossless data compression programmable hardware for high-speed data networks," in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2002, pp. 290–293.

[21] M. Milward, J. L. Nunez, and D. Mulvaney, "Design and implementation of a lossless parallel high-speed data compression system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 6, pp. 481–490, Jun. 2004.

[22] J. L. Nunez-Yanez and V. A. Chouliaras, "Gigabyte per second streaming lossless data compression hardware based on a configurable variable-geometry CAM dictionary," *IEE Proc.-Comput. Digit. Techn.*, vol. 153, no. 1, pp. 47–58, Jan. 2006.

[23] Helion Technology. *LZRW Compression Cores*. Accessed: Aug. 21, 2020. [Online]. Available: http://www.heliontech.com/comp_lzrw.htm

[24] M. A. A. El ghany, A. E. Salama, and A. H. Khalil, "Design and implementation of FPGA-based systolic array for LZ data compression," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 3691–3695.

[25] M. A. A. El ghany, A. E. Magdy, and A. E. Salama, *Design and Implementation of FPGA-Based Systolic Array for LZ Data Compression*. Rijeka, Croatia: InTech, 2010, pp. 75–92.

[26] K. Papadopoulos and I. Papaefstathiou, "Titan-R: A multigigabit reconfigurable combined compression/decompression unit," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 2, pp. 7:1–7:25, May 2010.

[27] W. J. Huang, N. Saxena, and E. J. McCluskey, "A reliable LZ data compressor on reconfigurable coprocessors," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2000, pp. 249–258.

[28] M. Morales-Sandoval and C. Feregrino-Uribe, "A hardware architecture for elliptic curve cryptography and lossless data compression," in *Proc. 15th Int. Conf. Electron., Commun. Comput. (CONIELECOMP)*, Feb. 2005, pp. 113–118.

[29] K. Papadopoulos and I. Papaefstathiou, "Titan-R: A reconfigurable hardware implementation of a high-speed compressor," in *Proc. 16th Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2008, pp. 216–225.

[30] A. Khu. (Sep. 25, 2001). *Xilinx FPGA Configuration Data Compression and Decompression*. Accessed: Aug. 21, 2020. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp152.pdf

[31] M. Bartík, S. Ubik, and P. Kubalík, "LZ4 compression algorithm on FPGA," in *Proc. IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2015, pp. 179–182.

[32] S. M. Lee, J. H. Jang, J. H. Oh, J. K. Kim, and S. E. Lee, "Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression," *IEICE Electron. Exp.*, vol. 14, no. 11, p. 6, 2017.

[33] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.

[34] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 5, pp. 530–536, Sep. 1978.

[35] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, Jun. 1984.

[36] I. Pavlov. (2016). *LZMA SDK*. Accessed: Aug. 21, 2020. [Online]. Available: http://www.7-zip.org/sdk.html

[37] Y. Collet. (May 26, 2011). *LZ4 Explained*. Accessed: Aug. 21, 2020. [Online]. Available: http://fastcompression.blogspot.cz/2011/05/lz4-explained.html

[38] T. Beneš, M. Bartík, and P. Kubalák, "Design of a high-throughput match search unit for lossless compression algorithms," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2019, pp. 0732–0738.

[39] W. A. Crofut and M. R. Sottile, "Design techniques of a delay-line content-addressed memory," *IEEE Trans. Electron. Comput.*, vol. EC-15, no. 4, pp. 529–534, Aug. 1966.

[40] F. J. Burkowski, "A hardware hashing scheme in the design of a multiterm string comparator," *IEEE Trans. Comput.*, vol. C-31, no. 9, pp. 825–834, Sep. 1982.

[41] D. Salomon, *Data Compression: The Complete Reference*. Berlin, Germany: Springer-Verlag, 2007.

[42] Z. Ullah, M. K. Jaiswal, and R. C. C. Cheung, "Z-TCAM: An SRAM-based architecture for TCAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 2, pp. 402–406, Feb. 2015.

[43] Z. Ullah, K. Ilgon, and S. Baeg, "Hybrid partitioned SRAM-based ternary content addressable memory," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 12, pp. 2969–2979, Dec. 2012.

[44] M. F. Oberhumer. *Lempel-Ziv-Oberhumer*. Accessed: Aug. 21, 2020. [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[45] J. Kane and Q. Yang, "Compression speed enhancements to LZO for multi-core systems," in *Proc. IEEE 24th Int. Symp. Comput. Archit. High Perform. Comput.*, Oct. 2012, pp. 108–115.

[46] O. Fiedler, "LZ-family data compression methods," M.S. thesis, Dept. Theor. Comput. Sci., CTU FIT, Prague, Czechia, 2014.

[47] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, vol. 3, 2nd ed. Redwood City, CA, USA: Addison Wesley, 1998.
[48] Xilinx Inc. (2019). *UG473 (v1.14)—7-Series FPGAs Memory Resources*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf
[49] F. E. Goetting and T. J. Bauer, "Block RAM with reset," U.S. Patent 6 101 132 A, Feb. 3, 1999. [Online]. Available: https://patents.google.com/patent/US6101132
[50] M. Bencivenni et al, "Performance of 10 gigabit Ethernet using commodity hardware," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, pp. 630–641, Apr. 2010.
[51] J. Hartmanis and R. E. Stearns, "On the computational complexity of algorithms," *Trans. Amer. Math. Soc.*, vol. 117, pp. 285–306, May 1965.
[52] C. Yoo, "High-speed DRAM interface," *IEEE Potentials*, vol. 20, no. 5, pp. 33–34, Dec. 2002.
[53] M. Stohanzl and Z. Fedra, "The FPGA implementation of dictionary; HW consumption versus latency," in *Proc. 36th Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2013, pp. 82–85.
[54] M. Bartík and S. Ubik, "System for implementation of a hash table," U.S. Patent 10 262 702, May 3, 2019. [Online]. Available: https://patents.google.com/patent/US10262702B2/en
[55] Xilinx Inc. (2016). *UG474—7-Series FPGAs Configurable Logic Block*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf
[56] C. E. LaForest and J. G. Steffan, "Efficient multi-ported memories for FPGAs," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, New York, NY, USA, 2010, pp. 41–50.
[57] C. E. LaForest, Z. Li, T. O'Rourke, M. G. Liu, and J. G. Steffan, "Composing multi-ported memories on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 3, pp. 16:1–16:23, Sep. 2014.
[58] V. Sklyarov, I. Skliarova, A. Barkalov, and L. Titarenko, *Synthesis and Optimization of FPGA-Based Systems*. Cham, Switzerland: Springer, 2014.
[59] C. Woods and B. Holdsworth, *Digital Logic Design*, 4th ed. Burlington, MA, USA: Newnes, 2002.
[60] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann, 2017.
[61] B. Ronak and S. A. Fahmy, "Improved resource sharing for FPGA DSP blocks," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–4.
[62] S. Hadjis, A. Canis, J. H. Anderson, J. Choi, K. Nam, S. Brown, and T. Czajkowski, "Impact of FPGA architecture on resource sharing in high-level synthesis," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, New York, NY, USA, 2012, pp. 111–114.
[63] Xilinx Inc. (2013). *UG627-XST User Guide for Virtex-4, Virtex-5, Spartan-3 and Newer CPLD Devices*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst.pdf
[64] Xilinx Inc. (2017). *DS890—UltraScale Architecture and Product Data Sheet: Overview*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf
[65] *Xilinx ISE 14.7 EULA*, Xilinx, San Jose, CA, USA, 2013.
[66] Xilinx Inc. *FPGA Memory Recommendations Using the ISE Design Suite 14*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/ise-design-suite/memory.html
[67] Xilinx Inc. *FPGA Memory Recommendations Using the Vivado Design Suite*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/memory.html
[68] Xilinx Inc. (2017). *UG904—Vivado Design Suite User Guide—Implementation*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug904-vivado-implementation.pdf
[69] Xilinx Inc. (2018). *UG497—7 Ser. DSP48E1 Slice User Guide*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
[70] Xilinx Inc. (2016). *WP447—UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf
[71] Xilinx Inc. (2005). *XAPP464 (v2.0)—Using Look-Up Tables as Distrib. RAM in Spartan-3 Generation FPGAs*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp464.pdf
[72] Altera Corporation. (2011). *Logic Array Blocks and Adaptive Logic Modules in Stratix V Devices*. Accessed: Aug. 21, 2020. [Online]. Available: http://www2.engr.arizona.edu/~ece506/readings/project-reading/6-cad/stx5_51002.pdf
[73] N. Pramstaller and J. Wolkerstorfer, "A universal and efficient AES co-processor for field programmable logic arrays," in *Field Programmable Logic and Application*, J. Becker, M. Platzner, and S. Vernalde, Eds. Berlin, Germany: Springer, 2004, pp. 565–574.
[74] Altera Corporation. (2011). *Memory Blocks Cyclone IV Devices*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-iv/cyiv-51003.pdf
[75] Lattice Semiconductor Corporation. (2013). *Technical Note TN1201—Memory Usage Guide for MachXO2 Devices*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.latticesemi.com/-/media/LatticeSemi/Documents/ApplicationNotes/MO/MemoryUsageGuideforMachXO2Devices.ashx?document_id=39082
[76] Lattice Semiconductor Corporation. (2018). *AC476 Application Note—Design Migration Guidelines From Xilinx 7-Series to PolarFire*. Accessed: Aug. 21, 2020. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/1243552-ac476-design-migration-guidelines-from-xilinx-7-series-to-polarfire
[77] J. Johnson. (Jul. 15, 2011). *List and Comparison of FPGA Companies*. Accessed: Aug. 21, 2020. [Online]. Available: http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html

MATĚJ BARTÍK (Member, IEEE) was born in Prague, Czech Republic. He received the B.Sc. degree in computer science from the Faculty of Electrical Engineering, Czech Technical University in Prague, in 2012, and the M.Sc. degree in informatics from the Faculty of Information Technology, Czech Technical University in Prague, in 2014, where he is currently pursuing the Ph.D. degree.

He joined CESNET, as an FPGA and an Electronics Designer, in 2014. His research interests include electronic and PCB design, embedded system design, including embedded safety and security topics, and low-level FPGA optimizations.

TOMÁŠ BENEŠ (Student Member, IEEE) was born in Prague, Czech Republic. He received the B.Sc. and M.Sc. degrees in informatics from the Faculty of Information Technology, Czech Technical University in Prague, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree.

Since 2016, he has been an FPGA Hardware Designer with the Research and Development Department, CESNET. His research interests include hardware design, low-level optimization, and hardware network monitoring.

PAVEL KUBALÍK was born in Hořice, Czech Republic. He received the M.Sc. and Ph.D. degrees in informatics from the Faculty of Electrical Engineering, Czech Technical University in Prague, in 2002 and 2007, respectively.

From 2004 to 2009, he was an Assistant Professor with the Faculty of Electrical Engineering, Czech Technical University in Prague. Since 2009, he has been an Assistant Professor with the Faculty of Information Technology, Czech Technical University in Prague. His research interests include fault-tolerant design in FPGA, digital design in FPGA, self-testing circuits-based on FPGA, HW design of networks, high-speed wireless networks, arithmetic in FPGA, error control, and self-repair codes in FPGA.

Chapter 7

Conclusions

This chapter summarizes the contributions of this dissertation thesis and briefly suggests possible future work.

7.1 Summary

In this dissertation thesis, I discuss the relationship between communication latency and throughput. It is a common misinterpretation that (overall) latency can be lowered by simply increasing the throughput. At a certain point (physical limitations such as the system's maximum operating frequency), the latency can not be further reduced without a complex optimization (non-trivial parallelization) of such algorithms and systems.

This problem was demonstrated on the example of the MVTP system and its demand to reduce the latency of the "JPEG2000 compression mode" by replacing the JPEG2000 IP core with a different compression technique. The JPEG2000 compression engine was limited to 3 Gbps of throughput by the chosen standards (3G-SDI) as physical interfaces. One instance of JPEG2000 engine is required per SDI interface.

To reduce the overall number of implemented data-specific compression algorithms (which can be expressed as number of SDI interfaces multiplied by a number of supported data types), I chose to implement a single lossless universal compression algorithm with the aim towards throughput 10 Gbps or above. There also was a strong emphasis on low-latency operation. Such compression engine was supposed to be placed in the MVTPs' networking part, acting as real-time IP packet compression.

Because there was no suitable (hardware) implementation of a universal lossless compression algorithm fulfilling the initial requirements, LZ4 (among others) was evaluated and found suitable for desired use case(s) and later on, it served as source of inspiration to develop new hardware specific optimization techniques and architectures which I propose.

An FPGA implementation of LZ4 was developed to prove the impact of such optimizations and architectures. The implementation features the best single-engine compression throughput (with worst case performance of 6 Gbps), while the latency was decreased to the 12.17 µs for a IP jumbo packet (9000 bytes of payload) which in case of the MVTP

contains an entire SDI pixel line. The latency has been reduced by a factor of 3.7 compared to other hardware implementations using 8-bit wide datapath. Despite the fact the actual implementation did not reach the required 10 Gbps throughput, its datapath width (64-bit originally) can be easily scaled up to 128, or 256, or even more bits to reach higher throughput.

7.2 Contributions of the Dissertation Thesis

7.2.1 Analysis of "Fast" Lossless Compression Algorithms from Hardware Designer's Perspective

The detailed survey, how the software implementations of LZ4 works, what are common features and properties of LZ4 and other "fast" lossless compression algorithms (and their relation to LZ77). Each phase of LZ4 was analyzed from the perspective of the hardware designer, pointing out possible problems and bottlenecks using simple hardware architecture implemented in FPGA.

7.2.2 Demonstration of LZ4 Suitability for 'Light" Compression of Image Data

The results (see Table A.2) indicate LZ4 could save approximately 13% of the required network bandwidth in our primary use case of broadcasting SDI data streams while using a decently sized compression dictionary. Therefore, the LZ4 compression algorithm can be considered viable to fulfill the initial requirement of the "Light" compression to save approximately 10%.

7.2.3 Parallel & Low-Latency Architecture for Match Search Unit

Non-trivial parallelism has been found to be the only way for increasing throughput to overcome the physical limits of various hardware implementations. The proposed 8-way architecture for the "Match Search Unit" phase of compression algorithms is capable of processing 64-bit wide words. The architecture can process data at a throughput of 16 Gbps with a latency of only 6 clock cycles. The architecture uses a hash table as a compression dictionary, which is shared among all MSUs.

7.2.4 Memory Access Optimized Scheme

I introduced memory access optimized data flow, which allows the "Match Search Unit" to generate fewer stalls in the data processing. The principle is to store a combined entry of data and data's original address instead of the original address only, as introduced in the reference LZ4 software implementation. Therefore, it is possible to determine a hash collision immediately preventing "Stalls" from occurring.

7.2.5 Masking "Match Length Finding" Initial Latency

To reduce the number of "stalls" even further, I propose using available memory data bus width to double the actual performance in certain phases ("Match Length Finding" and "Copying Literals") to mask start-up latency of such phases. This idea was demonstrated on our high-throughput/low-latency LZ4 hardware implementation, which is designed to process (new) 64 bits per clock cycle (the required memory read width is 88 bits, rounded up to the 128 bits, the value of the nearest power of two); however, the mentioned phases are processing 128 bits per clock cycle.

7.2.6 Novel Status Register Architecture

A novel architecture for implementing a status register was introduced. The status register is commonly used to store information of which memory entry is (in-)valid. Such memory-based data structures are often cleared or initialized to a default value between individual runs of an implemented algorithm (not particularly limited to compression algorithms only). The architecture is portable to any modern FPGAs.

7.2.7 Benchmarking Methodology for Digital Designs using Xilinx Synthesis Tools

To evaluate our status register architecture in a fair way, I created the benchmark with multiple phases, including randomized pin placement to decrease the influence of respective tools and actual hardware configuration of computers used for the synthesis process.

7.3 Future Work

The LZ4 output format (sequences) was found ineffective for hardware implementations. Therefore, I propose a concept of "Literal Length and Match Length Limit", which reveals the fact the main (software) advantage of the LZ4 is also it's biggest (hardware) disadvantage. The sequence format was designed with an emphasis on its decompression speed (in software). Therefore I suggest exploring in-depth the field of output formats used by compression algorithms implementations and to propose a new compression output format suitable and effective for both hardware and software implementations at the same time. LZ77-based algorithms are asymmetrical; thus, I propose emphasizing coding at high speed. I assume the research conducted in this are is highly probably worth of another dissertation thesis.

7.3.1 Literal Length and Match Length Limit Concept Proposal

This section presents an unpublished proposal of limiting a match length to achieve better predictability and overall performance. The concept does not provide an extensive quant-

itative survey; however, this wasn't the goal. The data format of LZ4 was selected for a brief evaluation and to demonstrate the ideas.

7.3.1.1 LZ4 Sequence Format and the Output Encoder

The LZ4 output format is called a sequence (see Fig. 7.1). The sequence does not seem to be complex; however, a (hardware) problem arises due to the method chosen for encoding the match and the literal length. The LSIC (Linear Small Integer Code) [6] produces an output with variable word sizes; thus, the encoder has to process the data first to estimate the length to generate the codeword. This also introduces uncertainty in the number of clock cycles needed for encoding the sequence.

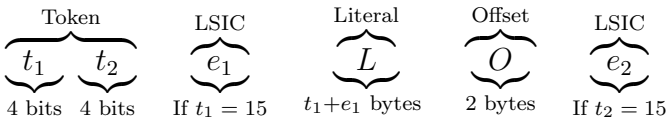


Figure 7.1: LZ4 sequence structure. [6]

7.3.1.2 Modified LZ4 Sequence Format

Some of the LZ4 hardware implementations [90, 91] introduced a modified LZ4 format to make it more suitable for hardware implementation to overcome the issue. The modification [90] is intended to "swap" fields "Literals Length" (t_1 plus e_1) and "Literals". Fields t_2 and e_2 represent a "Match Length" while "Offset" represents "Match" location within "Literals" including recently encoded ones. In addition, the "Literals Length" no longer uses a code with variable codeword length. On the other hand, these fields can repeat (virtually unlimited) within one LZ4 sequence. The modified LZ4 sequence format is no longer compatible with the original format; therefore, this can be considered a major disadvantage.

The modified format's main advantage is that the output encode function latency can be predicted in a more precise way (see Fig. 7.2) because the encode function can copy literals on the fly. When a variable coding is used for "Literals Length", the number of bytes necessary for encoded LSIC code word of each LZ4 sequence could vary between 1 and 256 bytes [68] for the minimal (respectively, for the maximum of 64 kiB) size of an LZ4 input block.

7.3.1.3 Introducing Literals Length Limit

I propose adjusting a maximal length of "Literals" to a fixed value. There are two complementary cases of LZ4 sequences: with "Literals" and without them (only a "Match" related fields are present). Therefore a found "Match" is located within actual "Literals" and the "Literals Length" has to be determined prior encoding an LZ4 sequence. In the second case (only "Match" is present), it is clear the "Literals Length" will be zero.

I assume the limit of fewer than 15 bytes (which can be encoded in the "Token" field only) can be considered as too short to be practical due to increased overhead. On the

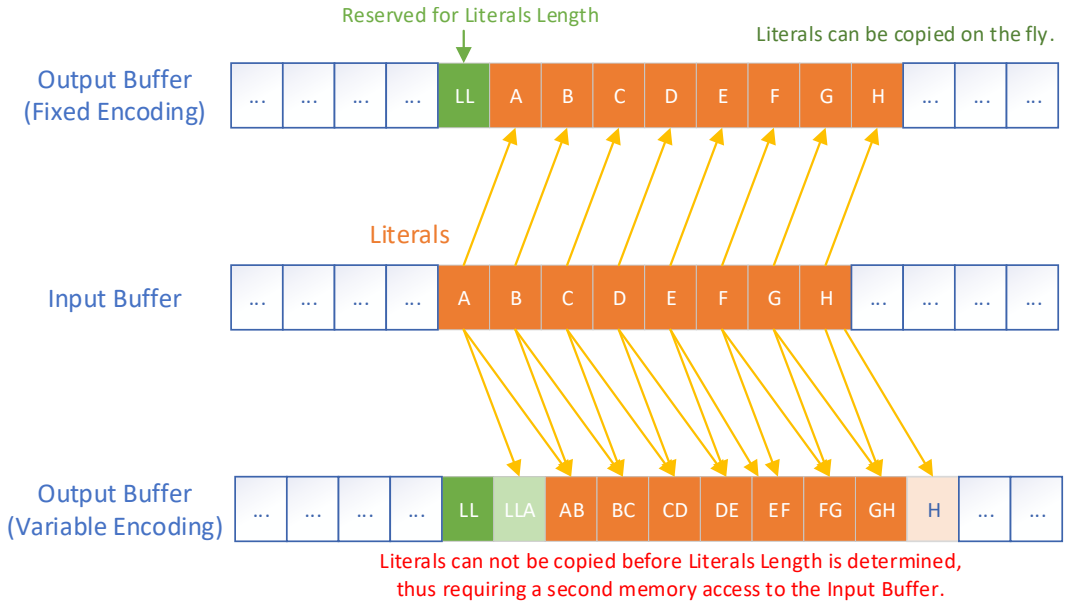


Figure 7.2: Literals output buffer placement for fixed and variable encoding.

other hand, increasing limit to 15+254=269 bytes (15 encoded by the token plus one extra byte which represents t_1 field, thus 2 bytes in total). For the case of the real-time IP packet compression with a maximum payload of 9000 bytes (Jumbo packet), a single encoded LZ4 sequence requires $(9000 - 15) : 255 \approx 36$ bytes. In the case where "limited" sequences are used, $9000 : (15 + 254) \times 2 \approx 66$ bytes are required. The difference of 30 bytes results in the extra 0,33% worst-case overhead. For the maximum payload size of 1500 bytes (a standard packet), the extra overhead is 0,25%. Therefore, for shorter blocks and for compressible data, the extra overhead will become negligible (due to the fact the "Matches" will generate more sequences to the output).

The proposed "Semi-Variable" (requiring one or two bytes only) "Literals Length" encoding retains full compatibility with the original LZ4 sequence format, which can be considered as an advantage.

In the case an LZ4 hardware implementation is capable of processing 8 bytes per clock cycle [70, 69], the number of the required bytes (in the output buffer) can be determined in two clock cycles; thus, these bytes can be allocated in advance and respective "Literals" can be copied right after them. This solution has virtually no disadvantages. The extra encoding overhead is considered negligible and slightly increased latency (just two clock cycles that can be masked). This seems to be worthy of increased predictability (no stalls) resulting in a (further) significant saving in terms of latency (see Fig. 7.3).

7.3.1.4 Introducing Match Length Limit

Another possible way to make the computation more predictable (therefore reducing the latency) might be splitting a "Match" into several smaller ones (see Fig. 7.3). The "Match" encoding process uses LSIC encoding for a "Match Length" in the original LZ4 sequence

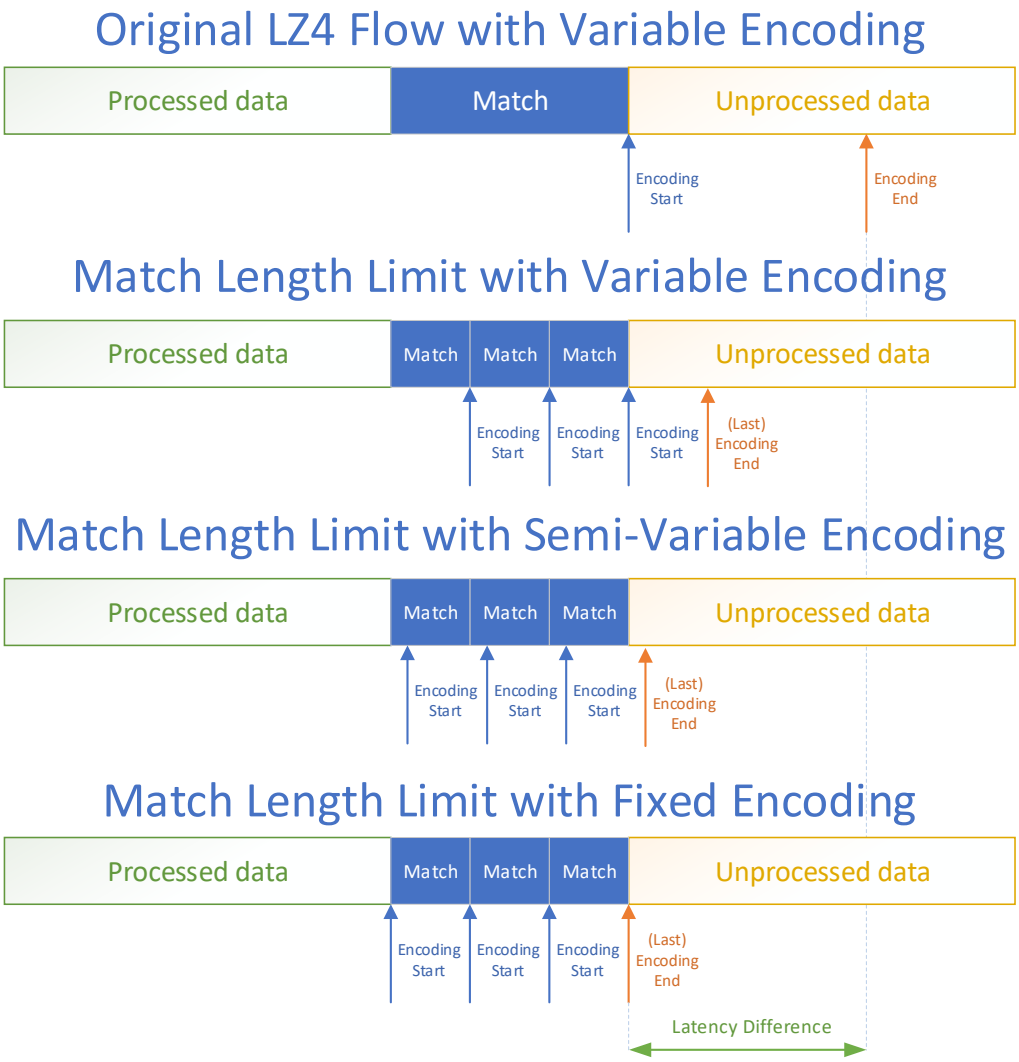
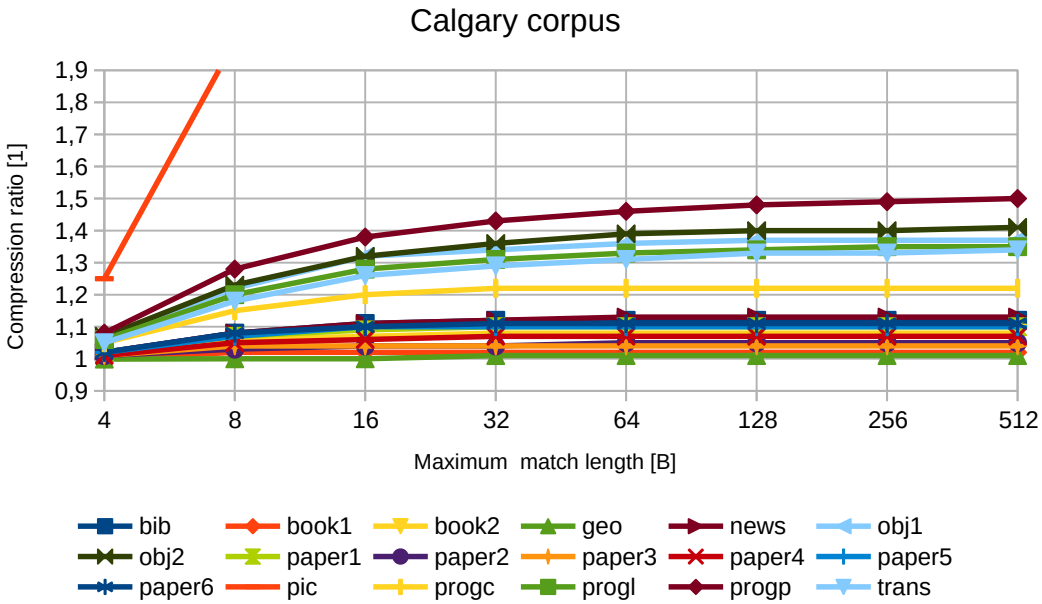


Figure 7.3: Visualised influence of the proposed concepts.

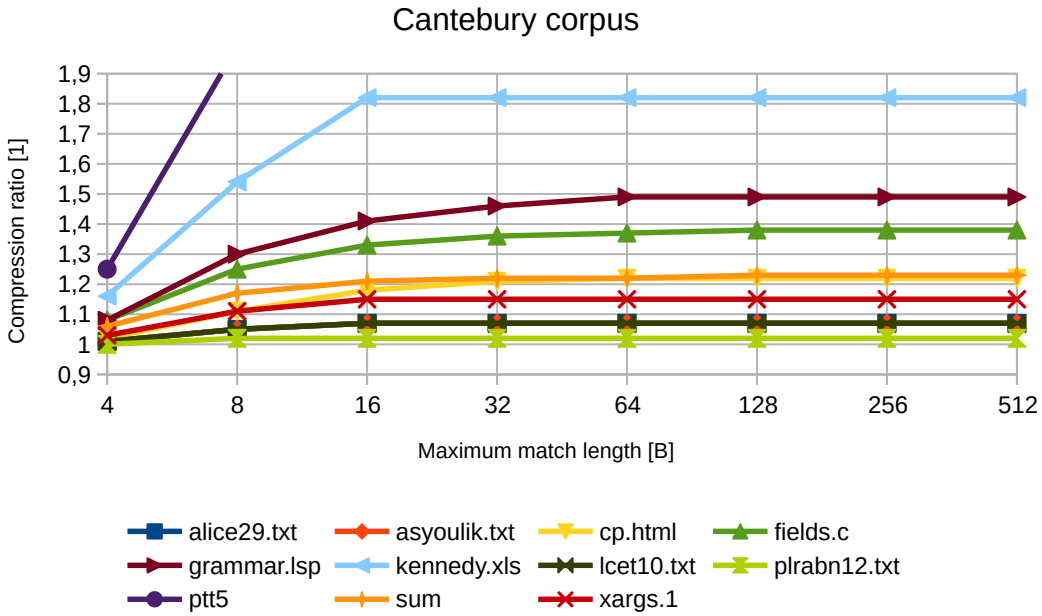
format. Therefore, the field "Offset" can not be written to the output buffer before the "Match Length" is determined. The problem (and a possible solution) is the same as in the case of "Literals Length".

Therefore, I propose limiting the "Match Length" in the same way as is proposed in the case of "Literals Length". This solution uses the "Semi-Variable" principle, which allows allocating space for a sequence in advance. Therefore, a compression algorithm implementation could encode multiple "Matches" concurrently (some kind of a "Matches" buffer might be necessary [69]). The usage of the "Semi-Variable" principle also retains the compatibility with the original LZ4 sequence format.

The results (see Table A.3 and Fig. 7.5b) indicate the overall compression ratio is very slightly reduced for the "Match Length" limit of 256 bytes (which requires only one additional byte to be reserved while "Semi-Variable" encoding is used).

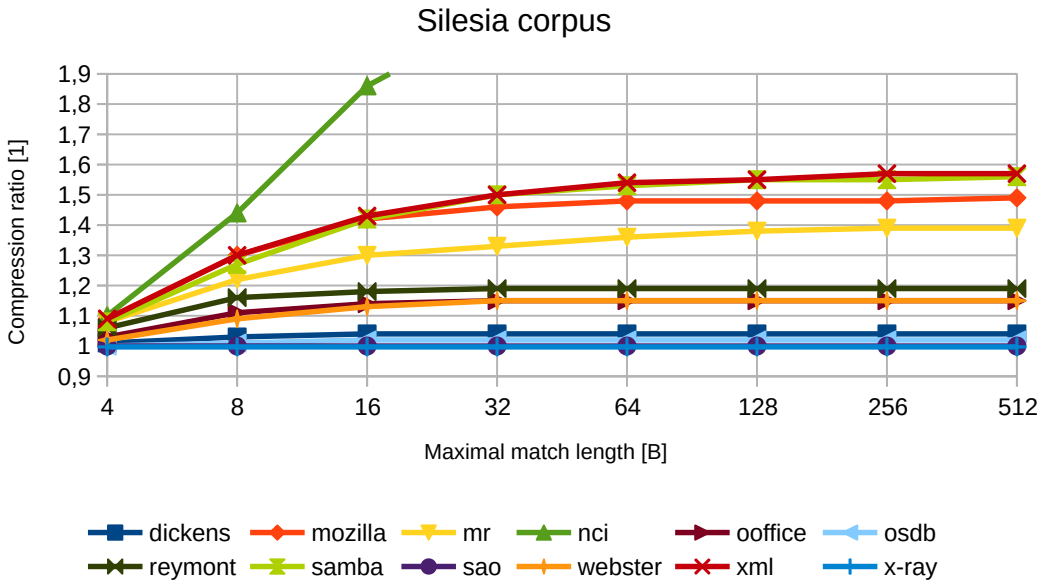


(a) Compression ratio of Calgary corpus.

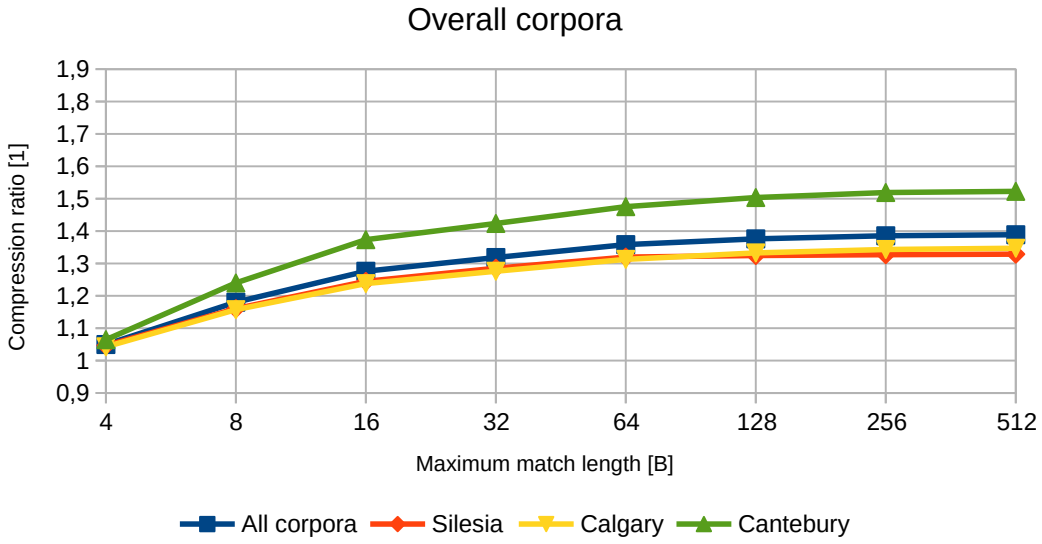


(b) Compression ratio of Cantebury corpus.

Figure 7.4: Relation between compression ratio and maximum match length — Part I.



(a) Compression ratio of Silesia corpus.



(b) Overall compression ratio of all corpora.

Figure 7.5: Relation between compression ratio and maximum match length — Part II.

Bibliography

[1] Fang, J. *Database Acceleration on FPGAs*. Dissertation thesis, Delft University of Technology, ISBN 978-94-028-1868-0, 2019, doi: 10.4233/uuid:84dfc577-ca6f-43ea-9b24-4dc160c103f5.

[2] Halák, J.; Krsek, M.; Ubik, S.; et al. Real-time long-distance transfer of uncompressed 4K video for remote collaboration. *Future Generation Computer Systems*, volume 27, no. 7, 2011: pp. 886 – 892, ISSN 0167-739X, doi: 10.1016/j.future.2010.11.014.

[3] Tektronix. *A Guide to Standard and High-Definition Digital Video Measurements*. [Online; accessed 8-Apr-2021]. Available from: `https://www.appliedelectronics.com/documents/GuidetoStandardHDDigitalVideoMeasurements.pdf`

[4] Halák, J.; Ubik, S. MTPP - Modular Traffic Processing Platform. In *Design and Diagnostics of Electronic Circuits Systems, 2009. DDECS '09. 12th International Symposium on*, April 2009, pp. 170–173.

[5] Alma Technologies s.a. *Scalable Ultra-High Throughput Lossy and Lossless JPEG 2000 Encoder*. [Online; accessed 8-Apr-2021]. Available from: `https://www.alma-technologies.com/ip-core.UHT-JPEG2K-E`

[6] Collet, Y. LZ4 explained. `http://fastcompression.blogspot.cz/2011/05/lz4-explained.html`, May 26, 2011, [Online; accessed 8-Apr-2021].

[7] WonderNetwork. *Global Ping Statistics*. [Online; accessed 8-Apr-2021]. Available from: `https://wondernetwork.com/pings`

[8] In-memory benchmark with fastest LZSS (QuickLZ, Snappy) compressors. `http://encode.ru/threads/1266-In-memory-benchmark-with-fastest-LZSS-(QuickLZ-Snappy)-compressors/page3?s=b2da7f50c8181fb016eb919d33133ede`, Nov 6, 2015, [Online; accessed 15-May-2016].

[9] Farina, F.; Szegedi, P.; Sobieski, J. GÉANT world testbed facility: Federated and distributed testbeds as a service facility of GÉANT. In *2014 26th International Teletraffic Congress (ITC)*, 2014, pp. 1–6, doi: 10.1109/ITC.2014.6932972.

[10] Ubik, S.; Halák, J.; Melnikov, J.; et al. Ultra-Low-Latency Video Transmissions for Delay Sensitive Collaboration. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 2020, pp. 1–4, doi: 10.1109/MECO49872.2020.9134361.

[11] Krugman, P. Three Expensive Milliseconds. `https://www.nytimes.com/2014/04/14/opinion/krugman-three-expensive-milliseconds.html`, April 13, 2014, [Online; accessed 8-Apr-2021].

[12] ST 291-1:2011 - SMPTE Standard - Ancillary Data Packet and Space Formatting. *ST 291-1:2011*, 2011: pp. 1–17, doi: 10.5594/SMPTE.ST291-1.2011.

[13] Halák, J. *Extendable and Scalable FPGA-based High-speed Packet Processing*. Dissertation thesis, Faculty of Information Technology, Czech Technical University in Prague, 2013.

[14] intoPIX s.a. *JPEG 2000 - Reliable Video Quality from Production to Archiving* . 2018, [Online; accessed 8-Apr-2021]. Available from: `https://www.intopix.com/blogs/post/JPEG-2000-Reliable-Video-Quality-from-Production-to-Archiving`

[15] Ubik, S.; Pospíšilík, J. Video Camera Latency Analysis and Measurement. *IEEE Transactions on Circuits and Systems for Video Technology*, 2020: pp. 1–1, doi: 10.1109/TCSVT.2020.2978057.

[16] ST 424:2012 - SMPTE Standard - 3 Gb/s Signal/Data Serial Interface. *SMPTE ST 424:2012*, Oct 2012: pp. 1–10, doi: 10.5594/SMPTE.ST424.2012.

[17] ST 2082-10:2015 - SMPTE Standard - 2160-line Source Image and Ancillary Data Mapping for 12G-SDI. *SMPTE ST 2082-10:2015*, March 2015: pp. 1–21, doi: 10.5594/SMPTE.ST2082-10.2015.

[18] Halák, J.; Ubik, S.; Žejdl, P. Receiver synchronization in video streaming with short latency over asynchronous networks. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, April 2010, pp. 403–405, doi: 10.1109/DDECS.2010.5491745.

[19] ST 292-0:2011 - SMPTE Overview Document - SMPTE Bit-Serial Interfaces at 1.5 Gb/s x2014; Roadmap for the 292 Document Suite. *SMPTE ST 292-0:2011*, March 2011: pp. 1–2, doi: 10.5594/SMPTE.ST292-0.2011.

[20] Stohanzl, M.; Fedra, Z. The FPGA implementation of dictionary; HW consumption versus latency. In *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, July 2013, pp. 82–85.

[21] Charrier, M.; Cruz, D. S.; Larsson, M. JPEG2000, the next millennium compression standard for still images. In *Multimedia Computing and Systems, 1999. IEEE International Conference on*, volume 1, Jul 1999, pp. 131–132 vol.1, doi: 10.1109/MMCS.1999.779134.

[22] Žejdl, P. *Low-Latency Video Transmissions for Real-Time Collaboration with a Scalable Hardware Acceleration*. Dissertation thesis, Faculty of Information Technology, Czech Technical University in Prague, 2014.

[23] Pinto, S. J.; Gawande, J. P. Performance analysis of medical image compression techniques. In *2012 Third Asian Himalayas International Conference on Internet*, Nov 2012, ISSN 1089-7801, pp. 1–4, doi: 10.1109/AHICI.2012.6408455.

[24] Oh, H.; Bilgin, A.; Marcellin, M. W. Visually Lossless Encoding for JPEG2000. *IEEE Transactions on Image Processing*, volume 22, no. 1, Jan 2013: pp. 189–201, ISSN 1057-7149, doi: 10.1109/TIP.2012.2215616.

[25] Salomon, D. *Data Compression: The Complete Reference*. Berlin, Germany / Heidelberg, Germany / London, UK / etc.: Springer-Verlag, 2007, ISBN 1-84628-602-6, xxv + 1092 pp., doi: 10.1007/978-1-84628-603-2.

[26] Bell, T.; Witten, I.; Cleary, J. Modeling for Text Compression. *ACM Computing Surveys*, volume 21, no. 4, Dec. 1989: pp. 557–591, ISSN 0360-0300, doi: 10.1145/76894.76896.

[27] Arnold, R.; Bell, T. A corpus for the evaluation of lossless compression algorithms. In *Proceedings DCC '97. Data Compression Conference*, 1997, pp. 201–210.

[28] Deorowicz, S. *Universal lossless data compression algorithms*. Dissertation thesis, Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, Institute of Computer Science, 2003. Available from: `http://sun.aei.polsl.pl/~sdeor/pub/deo03.pdf`

[29] Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, volume 23, no. 3, May 1977: pp. 337–343, ISSN 0018-9448, doi: 10.1109/TIT.1977.1055714.

[30] J. Ziv; A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, volume 24, no. 5, Sep 1978: pp. 530–536, ISSN 0018-9448, doi: 10.1109/TIT.1978.1055934.

[31] Welch, T. A. A Technique for High-Performance Data Compression. *Computer*, volume 17, no. 6, June 1984: pp. 8–19, ISSN 0018-9162, doi: 10.1109/MC.1984.1659158.

[32] Huffman, D. A. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, volume 40, no. 9, Sept 1952: pp. 1098–1101, ISSN 0096-8390, doi: 10.1109/JRPROC.1952.273898.

[33] ITU T.81 : Information technology - Digital compression and coding of continuous-tone still images (ITU-T SG16). `https://www.itu.int/rec/T-REC-T.81-199209-I/en`, 1992-09-18, [Online; accessed 8-Apr-2021].

[34] ISO/IEC 14496-10. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=66069`, 2004-10-01, information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding.

[35] ITU-T H.265 : High efficiency video coding. `https://www.itu.int/rec/T-REC-H.265`, 4/2014, [Online; accessed 8-Apr-2021].

[36] Feldspar, A. An Explanation of the DEFLATE Algorithm. `https://www.cs.ucdavis.edu/~martel/122a/deflate.html`, 23 August 1997, [Online; accessed 8-Apr-2021].

[37] Jahaya, B. A.; Rehman, A. U.; Defilippis, I. Hardware implementation of JPEG2000 encoder for video compression. In *Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on*, Nov 2007, pp. 1296–1299, doi: 10.1109/ICIAS.2007.4658594.

[38] Oberhumer, M. F. Lempel - Ziv - Oberhumer. `http://www.oberhumer.com/opensource/lzo/`, Mar 01, 2017, [Online; accessed 8-Apr-2021].

[39] intoPIX s.a. *TICO-RAW is the new RAW !* [Online; accessed 8-Apr-2021]. Available from: `https://www.intopix.com/tico-raw-fpga-asic-ip-cores`

[40] Descampe, A.; Keinert, J.; Richter, T.; et al. JPEG XS, a new standard for visually lossless low-latency lightweight image compression. In *Applications of Digital Image Processing XL*, volume 10396, edited by A. G. Tescher, International Society for Optics and Photonics, SPIE, 2017, pp. 68 – 79, doi: 10.1117/12.2273625.

[41] KOBAYASHI, H.; KIYA, H. Extension of JPEG XS for Two-Layer Lossless Coding. In *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, 2020, pp. 94–97, doi: 10.1109/GCCE50665.2020.9291824.

[42] Rigler, S.; Bishop, W.; Kennings, A. FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms. In *2007 Canadian Conference on Electrical and Computer Engineering*, April 2007, ISSN 0840-7789, pp. 1235–1238, doi: 10.1109/CCECE.2007.315.

[43] S. Rigler. *FPGA-Based Lossless Data Compression Using GNU Zip*. Master's thesis, University of Waterloo, 2007. Available from: `http://hdl.handle.net/10012/2692`

[44] Mehboob, R.; Khan, S. A.; Ahmed, Z.; et al. Multigig lossless data compression device. *IEEE Transactions on Consumer Electronics*, volume 56, no. 3, Aug 2010: pp. 1927–1932, ISSN 0098-3063, doi: 10.1109/TCE.2010.5606348.

[45] Mehboob, R.; Khan, S. A.; Ahmed, Z. High Speed Lossless Data Compression Architecture. In *2006 IEEE International Multitopic Conference*, Dec 2006, pp. 84–88, doi: 10.1109/INMIC.2006.358141.

[46] Shcherbakov, I.; Weis, C.; Wehn, N. A High-Performance FPGA-Based Implementation of the LZSS Compression Algorithm. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 449–453, doi: 10.1109/IPDPSW.2012.58.

[47] Leavline, E. J.; Singh, D. A. A. G. Hardware Implementation of LZMA Data Compression Algorithm. *International Journal of Applied Information Systems (IJAIS)*, volume 5, 2013: pp. 51–56.

[48] Li, B.; Zhang, L.; Shang, Z.; et al. Implementation of LZMA compression algorithm on FPGA. *Electronics Letters*, volume 50, no. 21, October 2014: pp. 1522–1524, ISSN 0013-5194, doi: 10.1049/el.2014.1734.

[49] Parekar, P. M.; Thakare, S. S. Hardware Implementation Of Lossless LZMA Data Compression Algorithm. *Progress In Science and Engineering Research Journal*, volume 2, no. 3, May-June 2014: pp. 201–205, ISSN 2347-6680.

[50] Morales-Sandoval, M.; Feregrino-Uribe, C. On the Design and Implementation of an FPGA-based Lossless Data Compressor. *Sociedad Mexicana de Ciencias de la Computación*, 2004: pp. 29–38, doi: 10.1.1.211.9098.

[51] Naqvi, S.; Naqvi, R.; Riaz, R. A.; et al. Optimized RTL design and implementation of LZW algorithm for high bandwidth applications. *PRZEGLAD ELEKTROTECHNICZNY (Electrical Review)*, volume 87, no. 4, 2011: pp. 279–285, ISSN 0033-2097.

[52] Nunez, J. L.; Jones, S.; Bateman, S. X-MatchPRO: a high performance full-duplex lossless data compressor on a ProASIC FPGA. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, International Workshop on, 2001.*, 2001, pp. 56–60, doi: 10.1109/IDAACS.2001.941979.

[53] Nunez, J. L.; Jones, S. Gbit/s lossless data compression hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 11, no. 3, June 2003: pp. 499–510, ISSN 1063-8210, doi: 10.1109/TVLSI.2003.812288.

[54] Nunez, J. L.; Feregrino, C.; Bateman, S.; et al. The X-MatchLITE FPGA-based data compressor. In *EUROMICRO Conference, 1999. Proceedings. 25th*, volume 1, 1999, ISSN 1089-6503, pp. 126–132 vol.1, doi: 10.1109/EURMIC.1999.794458.

[55] Nunez, J. L.; Jones, S. Lossless data compression programmable hardware for high-speed data networks. In *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on*, Dec 2002, pp. 290–293, doi: 10.1109/FPT.2002.1188694.

[56] Milward, M.; Nunez, J. L.; Mulvaney, D. Design and implementation of a lossless parallel high-speed data compression system. *IEEE Transactions on Parallel and Distributed Systems*, volume 15, no. 6, June 2004: pp. 481–490, ISSN 1045-9219, doi: 10.1109/TPDS.2004.7.

[57] Nunez-Yanez, J. L.; Chouliaras, V. A. Gigabyte per second streaming lossless data compression hardware based on a configurable variable-geometry CAM dictionary. *IEE Proceedings - Computers and Digital Techniques*, volume 153, no. 1, Jan 2006: pp. 47–58, ISSN 1350-2387.

[58] Helion Technology ltd. *LZRW Compression cores*. [Online; accessed 8-Apr-2021]. Available from: `http://www.heliontech.com/comp_lzrw.htm`

[59] El ghany, M. A. A.; Salama, A. E.; Khalil, A. H. Design and Implementation of FPGA-based Systolic Array for LZ Data Compression. In *2007 IEEE International Symposium on Circuits and Systems*, May 2007, ISSN 0271-4302, pp. 3691–3695, doi: 10.1109/ISCAS.2007.378644.

[60] El ghany, M. A. A.; Magdy, A. E.; Salama, A. E. *Design and Implementation of FPGA-based Systolic Array for LZ Data Compression*. InTech, 2010, ISBN 978-953-307-063-6, pp. 75–92, doi: 10.5772/8872.

[61] Papadopoulos, K.; Papaefstathiou, I. Titan-R: A Multigigabit Reconfigurable Combined Compression/Decompression Unit. *ACM Transactions on Reconfigurable Technology and Systems*, volume 3, no. 2, May 2010: pp. 7:1–7:25, ISSN 1936-7406, doi: 10.1145/1754386.1754388.

[62] Huang, W. J.; Saxena, N.; McCluskey, E. J. A reliable LZ data compressor on reconfigurable coprocessors. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, 2000, pp. 249–258, doi: 10.1109/FPGA.2000.903412.

[63] Sandoval, M. M.; Feregrino-Uribe, C. A Hardware Architecture for Elliptic Curve Cryptography and Lossless Data Compression. In *15th International Conference on Electronics, Communications and Computers (CONIELECOMP'05)*, Feb 2005, pp. 113–118, doi: 10.1109/CONIEL.2005.8.

[64] Papadopoulos, K.; Papaefstathiou, I. Titan-R: A Reconfigurable Hardware Implementation of a High-Speed Compressor. In *Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, April 2008, pp. 216–225, doi: 10.1109/FCCM.2008.14.

[65] Sukhwani, B.; Abali, B.; Brezzo, B.; et al. High-Throughput, Lossless Data Compresion on FPGAs. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, May 2011, pp. 113–116, doi: 10.1109/FCCM.2011.56.

[66] Fowers, J.; Kim, J. Y.; Burger, D.; et al. A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, May 2015, pp. 52–59, doi: 10.1109/FCCM.2015.46.

[67] Lee, S. M.; Jang, J. H.; Oh, J. H.; et al. Design of Hardware Accelerator for Lempel-Ziv 4 (LZ4) Compression. *IEICE Electronics Express*, volume 14, no. 11, 2017: p. 6, doi: 10.1587/elex.14.20170399.

[68] Bartík, M.; Ubik, S.; Kubalík, P. LZ4 compression algorithm on FPGA. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Dec 2015, pp. 179–182, doi: 10.1109/ICECS.2015.7440278.

[69] Beneš, T.; Bartík, M.; Kubalík, P. High Throughput and Low Latency LZ4 Compressor on FPGA. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig'19)*, Dec 2019, ISBN 978-1-7281-1957-1, pp. 1–5, doi: 10.1109/ReConFig48160.2019.8994794.

[70] Bartík, M.; Beneš, T.; Kubalík, P. Design of a High-Throughput Match Search Unit for Lossless Compression Algorithms. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2019, ISBN 9781728105543, pp. 0732–0738, doi: 10.1109/CCWC.2019.8666521.

[71] Crofut, W. A.; Sottile, M. R. Design Techniques of a Delay-Line Content-Addressed Memory. *IEEE Transactions on Electronic Computers*, volume EC-15, no. 4, Aug 1966: pp. 529–534, ISSN 0367-7508, doi: 10.1109/PGEC.1966.264360.

[72] Burkowski, F. J. A Hardware Hashing Scheme in the Design of a Multiterm String Comparator. *IEEE Transactions on Computers*, volume C-31, no. 9, Sept 1982: pp. 825–834, ISSN 0018-9340, doi: 10.1109/TC.1982.1676098.

[73] Lee, K. LZ4 Compression and Improving Boot Time. `https://events.static.linuxfound.org/sites/events/files/lcjpcojp13_klee.pdf`, [Online; accessed 8-Apr-2021].

[74] Gomes, R. D.; Costa, Y. G. G. d.; Aquino Júnior, L. L.; et al. A Solution for Transmitting and Displaying UHD 3D Raw Videos Using Lossless Compression. In *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*, WebMedia '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-2559-2, pp. 173–176, doi: 10.1145/2526188.2526228.

[75] Harnik, D.; Khaitzin, E.; Sotnikov, D.; et al. A Fast Implementation of DEFLATE. In *2014 Data Compression Conference*, March 2014, ISSN 1068-0314, pp. 223–232, doi: 10.1109/DCC.2014.66.

[76] Almeida, S.; Oliveira, V.; Pina, A.; et al. *Computational Science and Its Applications – ICCSA 2014: 14th International Conference, Guimarães, Portugal, June 30 – July 3, 2014, Proceedings, Part IV*, chapter Two High-Performance Alternatives to ZLIB Scientific-Data Compression. Cham: Springer International Publishing, 2014, ISBN 978-3-319-09147-1, pp. 623–638, doi: 10.1007/978-3-319-09147-1_45.

[77] Fiedler, O. *LZ-family data compression methods*. Master's thesis, CTU FIT, 2014.

[78] Siedelmann, H.; Wender, A.; Fuchs, M. *Pattern Recognition: 37th German Conference, GCPR 2015, Aachen, Germany, October 7-10, 2015, Proceedings*, chapter High Speed Lossless Image Compression. Cham: Springer International Publishing, 2015, ISBN 978-3-319-24947-6, pp. 343–355, doi: 10.1007/978-3-319-24947-6_28.

[79] Kane, J.; Yang, Q. Compression Speed Enhancements to LZO for Multi-core Systems. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, Oct 2012, ISSN 1550-6533, pp. 108–115, doi: 10.1109/SBAC-PAD.2012.29.

[80] CompressionRatings.Com – Summary (brief results). `http://compressionratings.com/sort.cgi?rating_sum.brief+6n`, [Online; accessed 21-Aug-2020].

[81] Shkel, Y.; Raginsky, M.; Verdú, S. Universal lossy compression under logarithmic loss. In *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 1157–1161, doi: 10.1109/ISIT.2017.8006710.

[82] ST 334-1:2015 - SMPTE Standard - Vertical Ancillary Data Mapping of Caption Data and Other Related Data. *ST 334-1:2015*, 2015: pp. 1–8, doi: 10.5594/SMPTE.ST334-1.2015.

[83] ST 2108-2:2019 - SMPTE Standard - Vertical Ancillary Data Mapping of KLV Formatted HDR/WCG Metadata. *ST 2108-2:2019*, 2019: pp. 1–13, doi: 10.5594/SMPTE.ST2108-2.2019.

[84] Khan, A.; Khan, M. U. K.; Bilal, M.; et al. Hardware architecture and optimization of sliding window based pedestrian detection on FPGA for high resolution images by varying local features. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2015, ISSN 2324-8432, pp. 142–148, doi: 10.1109/VLSI-SoC.2015.7314406.

[85] Touil, L.; Ben Abdelali, A.; Abdelatif, M. A hardware acceleration of a real time video processing. In *2012 16th IEEE Mediterranean Electrotechnical Conference*, March 2012, ISSN 2158-8473, pp. 862–865, doi: 10.1109/MELCON.2012.6196565.

[86] Mondal, P.; Banerjee, S. A Reconfigurable Memory-Based Fast VLSI Architecture for Computation of the Histogram. *IEEE Transactions on Consumer Electronics*, volume 65, no. 2, May 2019: pp. 128–133, ISSN 1558-4127, doi: 10.1109/TCE.2019.2900541.

[87] Sanny, A.; Yang, Y. E.; Prasanna, V. K. Energy-efficient histogram on FPGA. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, Dec 2014, ISSN 2325-6532, pp. 1–6, doi: 10.1109/ReConFig.2014.7032517.

[88] Liu, S.; Schulze, J. P.; Herr, L.; et al. CineGrid Exchange: A workflow-based peta-scale distributed storage platform on a high-speed network. *Future Generation Computer Systems*, volume 27, no. 7, 2011: pp. 966–976, ISSN 0167-739X, doi: https://doi.org/10.1016/j.future.2010.11.017, cineGrid: Super high definition media over optical networks. Available from: `https://www.sciencedirect.com/science/article/pii/S0167739X10002372`

[89] CineGrid Exchange. `cinegrid.org`, [Online; accessed 15-May-2016].

[90] Liu, W.; Mei, F.; Wang, C.; et al. Data Compression Device Based on Modified LZ4 Algorithm. *IEEE Transactions on Consumer Electronics*, volume 64, no. 1, 2018: pp. 110–117, doi: 10.1109/TCE.2018.2810480.

[91] Kim, J.; Cho, J. Hardware-Accelerated Fast Lossless Compression Based on LZ4 Algorithm. In *Proceedings of the 2019 3rd International Conference on Digital Signal Processing*, ICDSP 2019, New York, NY, USA: Association for Computing Machinery, 2019, ISBN 9781450362047, p. 65–68, doi: 10.1145/3316551.3316564.

Reviewed Publications of the Author Relevant to the Thesis

[A.1] Bartík, M. and Ubik, S. and Kubalík, P. *LZ4 Compression Algorithm on FPGA*; 21st IEEE International Conference on Electronics, Circuits, and Systems, 978-1-4799-2451-6, pp. 179 - 182, Cairo, Egypt, 2015.

The paper has been cited (21[1]) in:

- Cheng, L.; Guo, S.; Wang, Y.; Yang, Y. *Lifting Wavelet Compression Based Data Aggregation in Big Data Wireless Sensor Networks*; IEEE 22nd International Conference On Parallel And Distributed Systems (ICPADS), pp. 561 - 568, 2016. ISSN 1521-9097.
- Lin Tao, X.; Cai Wenting, X.; Chen Xianyi, X.; Zhou Kailun, X.; Wang Shuhui, X. *Lossless Compression Algorithm Based on String Matching with High Performance and Low Complexity for Screen Content Coding*; Journal of Electronics & Information Technology, vol. 39, no. 2, pp. 351 - 359, 2017. ISSN 1009-5896.
- Reeks, Christian *Sensing and Human Pose Estimation in Extreme Industrial Environments for Physical Human Robot Interaction*; http://hdl.handle.net/10453/123281, 2017.
- Anu, V.R; Sherly, E *Optimized delta compression in live migration of virtual machines*; International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, 2017, pp. 2866-2872.
- Liu, W.; Mei, F.; Wang, C.; O'Neill, M.; Swartzlander, Earl E. *Data Compression Device Based on Modified LZ4 Algorithm*; IEEE Transactions On Consumer Electronics, vol. 64, no. 1, pp. 110 - 117, 2018. ISSN 0098-3063.

[1] Auto-citations are excluded for all listed publications.

- Rawal, Bharat S. *Cloud-based Distributed Nth Order Binary Encoding redefines the limit of lossless compression*; 2018 IEEE International Conference On Smart Cloud (SMARTCLOUD), pp. 99 - 104, 2018.
- Moreira, A.; Ivson, P.; Celes, W. *Hybrid Cloud Rendering System for Massive CAD Models*; Proceedings 2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), pp. 234 - 241, 2018. ISSN 1530-1834.
- Asamoah Owusu, D. *Modeling Outputs of Efficient Compressibility Estimators*; 2018 Retrieved from the University of Minnesota Digital Conservancy, http://hdl.handle.net/11299/200159.
- Fuzong, W.; Helin, G.; Jian, Z. *Dynamic Data Compression Algorithm Selection for Big Data Processing on Local File System*; ACM CSAI '18: Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence, ISBN 978-1-4503-6606-9/18/12
- Kim, J.; Cho, J. *Hardware-accelerated Fast Lossless Compression Based on LZ4 Algorithm*; 2019 3rd International Conference on Digital Signal Processing (ICDSP 2019), pp. 65 - 68, 2019.
- Santos, B.; Cruz, N.; Fernandes, A.; Carvalho, P. F.; Sousa, J.; Goncalves, B.; Riva, M.; Pollastrone, F. et al. *Real-Time Data Compression for Data Acquisition Systems Applied to the ITER Radial Neutron Camera*; IEEE Transactions on Nuclear Science, vol. 66, no. 7, pp. 1324 - 1329, 2019. ISSN 0018-9499.
- Fang, J.; Mulder, Yvo T. B.; Hidders, J.; Lee, J.; Hofstee, H. *In-memory database acceleration on FPGAs: a survey*; VLDB Journal, 2019. ISSN 1066-8888.
- Fang J.; Chen J.; Lee J.; Al-Ars Z.; Hofstee H. P. *Refine and Recycle: A Method to Increase Decompression Parallelism*; 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 2019, pp. 272-280.
- Fang, J. *Database Acceleration on FPGAs*; Delft University of Technology, 2019, pp. 1-110, ISBN 978-94-028-1868-0.
- Jiang, H.; Lin, S.-J. *A Rolling Hash Algorithm and the Implementation to LZ4 Data Compression*; IEEE Access, vol. 8, pp. 35529-35534, 2020. ISSN 2169-3536.
- Song, Y.; Zhu, Y.; Hou, J.; Du, S.; Song, S. *Astronomical Data Preprocessing Implementation Based on FPGA and Data Transformation Strategy for the FAST Telescope as a Giant CPS*; IEEE Access, vol. 8, pp. 56837 - 56846, 2020. ISSN 2169-3536.
- Szabarin, B.; Kiss, A. *Word pattern prediction using Big Data frameworks*; Acta Universitatis Sapientiae, Informatica, vol 12., no. 1, pp. 51-69, 2020, ISSN 2066-7760.

- Fang J.; Chen J.; Lee J.; Al-Ars Z.; Hofstee H. P. *An Efficient High-Throughput LZ77-Based Decompressor in Reconfigurable Logic*; Journal of Signal Processing Systems, vol. 92, pp. 931–947, 2020, ISSN 1939-8115.
- Sriraman, A.; Dhanotia, A. *Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale*; ACM ASPLOS '20: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 733-750, 2020, ISBN 978-1-4503-7102-5/20/03.
- Sánchez-Gallegos, D.D., Carrizales-Espinoza, D., Reyes-Anastacio, H.G., Gonzalez-Compean, J.L., Carretero, J., Morales-Sandoval, M., Galaviz-Mosqueda, A. *From the edge to the cloud: A continuous delivery and preparation model for processing big IoT data* Simulation Modelling Practice and Theory (2020), 105, art. no. 102136, ISSN: 1569-190X.
- Choi, D. Y.; Oh, J. H. ; Kim, J. K.; Lee, S. E. *Energy Efficient and Low-Cost Server Architecture for Hadoop Storage Appliance* KSII Transactions on Internet and Information Systems, Volume 14, Issue 12, 31 December 2020, Pages 4648-4663 ISSN: 1976-7277.

[A.2] Bartík, M. and Ubik, S. and Kubalík, P. *A Novel and Efficient Method to Maintain FPGA Embedded Memory Content with an Asymptotically Constant Time (Re)Initialization Designed for an IP Packet Lossless Compression*; International Conference on ReConFigurable Computing and FPGAs (ReConFig 2016), 978-1-5090-3707-0, pp. 1 - 6, Cancún, Mexico. 2016.

[A.3] Bartík, M. and Ubik, S. and Kubalík, P. and Beneš, T. *Performance Comparison of Multiple Approaches of Status Register for Medium Density Memory Suitable for Implementation of a Lossless Compression Dictionary* (Abstract); Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2018), 978-1-4503-5614-5, p. 290, Monterey, USA. 2018.

[A.4] Bartík, M. and Beneš T. and Kubalík, P. *Design of a High-Throughput Match Search Unit for Lossless Compression Algorithms*; The 9th IEEE Annual Computing and Communication Workshop and Conference (CCWC), ISBN 978-1-7281-0554-3, pp. 732 - 738, Las Vegas, USA. 2019.

The paper has been cited (2) in:

- Loukil, H.; Abbas, M.; Algahtani, A.; Kessentini, A.; Muneer, P.; Ijyas, T.; Wase, M. *Role of Nanotechnology in Lossless Video Compression Coding*; Nanoscience And Nanotechnology Letters, vol. 11, no. 12, pp. 1617 - 1632, 2019. ISSN 1941-4900.
- Zhou, Q.; Cai, S.; Zhang, Y. *Parallel Heuristic Community Detection Method Based on Node Similarity*; IEEE Access, vol. 7, pp. 184145 - 184159, 2019. ISSN 2169-3536.

[A.5] Beneš T. and Bartík, M. and Kubalík, P. *High Throughput and Low Latency LZ4 Compressor on FPGA*; 2019 International Conference on ReConFigurable Computing and FPGAs, ISSN 2640-0472. ISBN 978-1-7281-1957-1, pp. 1 - 5, Cancún, Mexico 2019.

[A.6] Bartík, M. (50%) and Beneš T. (25%) and Kubalík, P. (25%) *An In-sight into How Compression Dictionary Architecture can Affect the Overall Performance in FPGAs*; IEEE Access, 2020 (8), pp. 183101 - 183116. ISSN 2169-3536. 2020.

Granted Patents of the Author Relevant to the Thesis

[A.7] Bartík, M. (75%) and Ubik, S. (25%) *Systém pro realizaci rozptylovací tabulky*; CZ306787, 24th May 2017,

[A.8] Bartík, M. (75%) and Ubik, S. (25%) *System for implementation of a hash table*; EP3244324, 12th September 2018,

[A.9] Bartík, M. (75%) and Ubik, S. (25%) *System for Implementation of a Hash Table*; US10262702B2, 26th April 2019,

Remaining Reviewed Publications of the Author not Relevant to the Thesis

[A.10] Bartík, M. (75%) and and Novotný, M. (25%) *Advanced control unit for linear motor for precise measurements in biomechanics*; 4th Embedded Computing. Mediterranean Conference (MECO 2015), pp. 129-133. ISBN 978-1-4799-8999-7. 2015.

The paper has been cited (1) in:

- Soares, M.; Almeida Junior, A.; Canuto Alves, T.; Neto, L. *LIM Control Strategy Supported by Genetic Algorithm with Unbalanced AC Source*; International Journal Of Emerging Electric Power Systems, vol. 19, no. 4, 2018. ISSN 1553-779X.

[A.11] Bartík, M. and Pichlová, D. and Kubátová, H. *Hardware-software co-design: A practical course for future embedded engineers*; Proceedings of the 5th Mediterranean Conference on Embedded Computing (MECO 2016), p. 347-350, ISSN 2377-5475, ISBN 978-1-5090-2221-2. 2016.

The paper has been cited (3) in:

- Hung, P.D.; Nam, L.H.; Van Thang, H. *Flexible development for embedded system software*; 4th International Conference on Research in Intelligent and Computing in Engineering, RICE 2019, 2019, ISBN 978-981152779-1.
- Pfundt, B.; Reichenbach, M.; Fey, D. *Comprehensive curriculum for reconfigurable heterogeneous computer architecture education*; IET Circuits Devices & Systems, vol. 11, no. 4, pp. 292 - 298, 2017. ISSN 1751-858X.
- Mite-Baidal, K; Delgado-Vera, C; Aguirre-Munizaga, M; Calle-Romero, K *Prototype of an Embedded System for Irrigation and Fertilization in Greenhouses*; Technologies and Innovation. CITI 2019. Communications in Computer and Information Science, vol 1124, 2019, ISBN 978-3-030-34988-2.

[A.12] Bartík, M. and Buček, J. *A Low-Cost Multi-Purpose Experimental FPGA Board for Cryptography Applications*; IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), pp. 1-6, ISBN 978-1-5090-4473-3. 2016.

The paper has been cited (7) in:

- Socha, P.; Novotný., M; *Towards High-Level Synthesis of Polymorphic Side-Channel Countermeasures*; 2020 23rd Euromicro Conference on Digital System Design (DSD), Kranj, Slovenia, 2020, pp. 193-199, ISBN 978-1-7281-9535-3.
- Socha, P.; Miškovský, V.; Kubátová, H.; Novotný., M; *Efficient algorithmic evaluation of correlation power analysis: Key distinguisher based on the correlation trace derivative*; Microprocessors and Microsystems, Volume 71, November 2019, ISSN 0141-9331.
- Miškovský, V. *Side-Channel Analysis: Efficient Attacks and Fault-Tolerant Countermeasures*; dissertation thesis, 2020, `https://ddd.fit.cvut.cz/PhD/PhDThesis_Miskovsky.pdf`
- Tybura, M. *Evolutionary algorithms with and without adaptive mutation in AI based cryptography*; ITM Web Conf. Volume 21, 2018, Computing in Science and Technology (CST 2018), ISSN 2271-2097.
- Socha, P.; Miškovský, V.; Novotný., M; Kubátová, H. *Correlation power analysis distinguisher based on the correlation trace derivative*; 21st Euromicro Conference on Digital System Design (DSD), Prague, 2018, pp. 565-568, ISBN 978-1-5386-7377-5.
- Socha, P.; Miškovský, V.; Novotný., M; Kubátová, H. *Speeding up differential power analysis using integrated power traces*; 7th Mediterranean Conference on Embedded Computing (MECO), Budva, 2018, pp. 1-5, ISBN 978-1-5386-5683-9.
- Miškovský, V. *Fault tolerance and resistance against side channel attacks in FPGA*; 2017, `https://hwlab.fit.cvut.cz/_media/gacr/publikace/report-miskovsky.pdf`

[A.13] Bartík, M. *Clock Domain Crossing – An Advanced Course for Future Digital Design Engineers*; 7th Mediterranean Conference on Embedded Computing (MECO), pp. 76-80, ISBN 978-1-5386-5683-9. 2018.

[A.14] Socha, P.; Brejník, J.; Bartík, M. *Attacking AES Implementations Using Correlation Power Analysis on ZYBO Zynq-7000 SoC Board*; 7th Mediterranean Conference on Embedded Computing (MECO), pp. 29-32, ISBN 978-1-5386-5683-9. 2018.

The paper has been cited (3) in:

- De Los Reyes, E.M.; Sison, A.M.; Medina, R.P. *Modified AES cipher round and key schedule*; Indonesian Journal of Electrical Engineering and Informatics (IJEEI), vol. 7, pp. 29 - 36, 2019. ISSN 2089-3272

- Gui, Y.; Tamore, S. M.; Siddiqui, A.S.; Saqib, F. *A Key Update Scheme for Side-Channel Attack Mitigation*; 16th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT and AI (HONET-ICT), 2019. ISBN 978-1-7281-3971-5. ISSN 1949-4092.
- Prinetto, P; Roascio, G *Hardware Security, Vulnerabilities, and Attacks: A Comprehensive Taxonomy*; Proceedings of the Fourth Italian Conference on Cyber Security (ITASEC 2020), vol. 2597, pp. 177 - 189, 2020. ISSN 1613-0073.

[A.15] Hynek, K. and Beneš, T. and Bartík, M. and Kubalík, P. *Ultra High Resolution Jitter Measurement Method for Ethernet Based Networks*; 9th IEEE Annual Computing and Communication Workshop and Conference (CCWC), pp. 847-851, ISBN 978-1-7281-0554-3. 2019.

[A.16] Bartík, M. and Beneš, T. and Hynek, K. *An Example of PCB Reverse Engineering - Reconstruction of Digilent JTAG SMT3 Schematic*; 7th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering, pp. 1-6, ISBN 978-1-7281-6730-5. 2019.

[A.17] Bartík, M. *Reverse Engineering of Arrow USB Programmer2 JTAG Adapter for Intel/Altera FPGAs*; 9th Mediterranean Conference on Embedded Computing (MECO), pp. 1-4, ISSN 2637-9511, ISBN 978-1-7281-6949-1. 2020.

[A.18] Bartík, M. *External Power Gating Technique – An Inappropriate Solution for Low Power Devices*; 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 241-245, ISSN 2644-3163, ISBN 978-1-7281-8416-6. 2020.

Research Projects of the Author

[A.19] Large Infrastructure CESNET *LM2010005, Ministry of Education, Youth and Sports*; member of a research team; 2011–2015;
Available: `https://starfos.tacr.cz/en/project/LM2010005`

[A.20] The Optimalization of Physical Characteristics of Vascular Substitutes for Low Flow *NT13302, Ministry of Health*; technical support; 2012–2015;
Available: `https://starfos.tacr.cz/en/project/NT13302`

[A.21] Attack-Resistant and Fault-Tolerant Architectures Based on Reconfigurable Devices *SGS15/119/OHK3/1T/18, Czech Technical University*; member of a research team; 2015;

[A.22] Dependable architectures suitable for FPGAs *SGS16/121/OHK3/1T/18, Czech Technical University*; member of a research team; 2016;

[A.23] Fault-Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features *GA16-05179S, Czech Science Foundation*; member of a research team; 2016–2018;
Available: `https://starfos.tacr.cz/en/project/GA16-05179S`

[A.24] 8K Studio over IP bridge *7D16005, Ministry of Education, Youth and Sports and Eurostars project (as part of Horizon 2020)*; member of a research team; 2016–2018;
Available: `https://www.era-learn.eu/network-information/networks/eurostars-2/eurostars-cut-off-17-09.2015/8k-studio-over-ip-bridge`

[A.25] Dependable and attack-resistant architectures for programmable devices *SGS17/213/OHK3/3T/18, Czech Technical University*; member of a research team; 2017–2019;

[A.26] CESNET E-Infrastructure - Modernisation *EF16_013/0001797, Ministry of Education, Youth and Sports*; member of a research team; 2017–2020;
Available: `https://starfos.tacr.cz/en/project/EF16_013%2F0001797`

[A.27] Monitoring of sensitive objects using the Internet of Things *FV30192, Ministry of Industry and Trade*; member of a research team; 2018–2020;
Available: `https://starfos.tacr.cz/en/project/FV30192`

[A.28] Distance collaboration in education in performing arts with modern telecommunication technologies *TL01000106, Technology Agency of the Czech Republic*; member of a research team; 2018–2021;
Available: `https://starfos.tacr.cz/en/project/TL01000106`

[A.29] Device for low-latency video transmissions in JPEG XS format *FW01010230, Technology Agency of the Czech Republic*; member of a research team; 2020–2023;
Available: `https://starfos.tacr.cz/en/project/FW01010230`

Evaluation Activities

[A.30] Organizing committee of the *Prague Embedded Systems Workshop (PESW)*, 2015, available: `https://pesw.fit.cvut.cz/2015/`

[A.31] Conference paper reviews (6x) for the *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015

[A.32] Conference paper reviews (5x) for the *Mediterranean Conference on Embedded Computing (MECO)*, 2015–2021, Available: `http://embeddedcomputing.meconet.me/`

[A.33] Journal article review for the *Integration, the VLSI Journal*, 2020, ISSN: 0167-9260, Available: `https://www.journals.elsevier.com/integration`

[A.34] Journal article reviews (4x) for the *Microprocessors and Microsystems*, 2020–2021, ISSN: 0141-9331, Available: `https://www.journals.elsevier.com/microprocessors-and-microsystems`

Doctoral Workshop Publications of the Author

[A.35] Bartík, M. *Practical use of FPGA Chips for Implementation*; Proceedings of the 2nd Prague Embedded Systems Workshop (PESW 2014), pp. 15. 2014.

[A.36] Bartík, M. and Ubik, S. and Kubalík, P. *Rychlé bezztrátové kompresní algoritmy*; Počítačové architektury a diagnostika – Sborník příspěvků (PAD 2015), pp. 31-36. ISBN 978-80-7454-522-1. 2015.

[A.37] Bartík, M. and Ubik, S. and Kubalík, P. *Nová a efektivní metoda pro zajištení platnosti dat ve vestavných pamětech FPGA se zaměřením na kompresi IP packetů v reálném čase*; Počítačové architektury a diagnostika – Sborník příspěvků (PAD 2016), p. 89-92. ISBN 978-80-214-5376-0. 2016.

[A.38] Bartík, M. and Pichlová, D. *Development of a sound recording system for audio cassette duplication on an industrial scale*; Proceedings of the 4th Prague Embedded Systems Workshop (PESW 2016), pp. 2-3. ISBN 978-80-01-05984-5. 2016.

[A.39] Bartík, M. and Buček, J. *A Low-Cost Unified Experimental FPGA Board for Cryptography Applications*; TRUDEVICE 2016 Final Conference, pp. 75-80. 2016.

[A.40] Bartík, M. *Xilinx 7-Series FPGA Based Evaluation Platform for Physically Unclonable Function*; TRUDEVICE 2018, p. 2. 2018.

Appendix A

Thesis Results and Related Data

Compressor name	Compression	Decompression	Compr. size	Ratio
memcpy	4786 MB/s	4764 MB/s	1073450940	100.00
brotli 2015-10-29 level 1	144 MB/s	529 MB/s	246271300	22.94
brotli 2015-10-29 level 2	122 MB/s	546 MB/s	242682971	22.61
fastlz 0.1 level 1	370 MB/s	697 MB/s	424460100	39.54
fastlz 0.1 level 2	356 MB/s	702 MB/s	396926168	36.98
lz4 r131	**602 MB/s**	**3003 MB/s**	**410592728**	**38.25**
lz4fast r131 acc=3	677 MB/s	2936 MB/s	449111230	41.84
lz4fast r131 acc=17	1075 MB/s	3541 MB/s	632585278	58.93
lz5 r131b	403 MB/s	1239 MB/s	319290041	29.74
lzf level 0	373 MB/s	669 MB/s	436283836	40.64
lzf level 1	379 MB/s	685 MB/s	417005148	38.85
lzjb 2010	330 MB/s	609 MB/s	558864581	52.06
lzo1b 2.09 -1	**297 MB/s**	**742 MB/s**	**396101254**	**36.90**
lzo1b 2.09 -9	177 MB/s	771 MB/s	357517128	33.31
lzo1b 2.09 -99	145 MB/s	791 MB/s	341665725	31.83
lzrw1	318 MB/s	583 MB/s	492971960	45.92
lzrw1a	338 MB/s	620 MB/s	484531017	45.14
lzrw2	337 MB/s	670 MB/s	437177676	40.73
lzrw3	350 MB/s	572 MB/s	410412696	38.23
lzrw3a	178 MB/s	643 MB/s	376266574	35.05
pithy 2011-12-24 level 0	565 MB/s	1819 MB/s	400800683	37.34
pithy 2011-12-24 level 9	421 MB/s	2039 MB/s	333923449	31.11
quicklz 1.5.0 -1	496 MB/s	698 MB/s	353272969	32.91
quicklz 1.5.1 b7 -1	544 MB/s	670 MB/s	353272969	32.91
snappy 1.1.3	441 MB/s	1523 MB/s	421117004	39.23
tornado 0.6a -1	391 MB/s	537 MB/s	404475734	37.68
tornado 0.6a -2	326 MB/s	482 MB/s	328041309	30.56
tornado 0.6a -3	218 MB/s	347 MB/s	267889476	24.96
tornado 0.6a -4	197 MB/s	373 MB/s	251555844	23.43
wflz 2015-09-16	358 MB/s	914 MB/s	459671278	42.82
yappy 1	154 MB/s	2393 MB/s	425838645	39.67
yappy 10	121 MB/s	2490 MB/s	414283056	38.59
yappy 100	108 MB/s	2501 MB/s	412713445	38.45
zlib 1.2.8 -1	114 MB/s	387 MB/s	316582284	29.49
zstd v0.3	371 MB/s	865 MB/s	271782671	25.32
zstd_HC v0.3 -1	370 MB/s	859 MB/s	271782671	25.32
zstd_HC v0.3 -5	121 MB/s	ERROR	228623205	21.30

Table A.1: Performance of "fast" compression algorithms [8].

8-bit (RGB)							
File	**N**	**8**	**10**	**11**	**12**	**13**	**14**
00000050.tif		0,113	0,109	0,108	0,108	0,108	0,107
00000200.tif		0,628	0,614	0,601	0,591	0,585	0,583
00001000.tif		0,572	0,556	0,545	0,536	0,531	0,529
00005000.tif		0,743	0,720	0,698	0,679	0,670	0,666
00010000.tif		0,819	0,802	0,787	0,772	0,763	0,758
00015000.tif		0,623	0,607	0,598	0,590	0,585	0,582
00020000.tif		0,729	0,712	0,701	0,689	0,680	0,676
Average		0,686	0,669	0,655	0,643	0,636	0,632
24-bit (RGB)							
File	**N**	**8**	**10**	**11**	**12**	**13**	**14**
00000050.tif		0,181	0,176	0,174	0,173	0,172	0,172
00000200.tif		0,854	0,844	0,840	0,837	0,835	0,834
00001000.tif		0,850	0,836	0,831	0,827	0,826	0,825
00005000.tif		0,972	0,961	0,955	0,951	0,948	0,947
00010000.tif		0,978	0,973	0,970	0,968	0,966	0,966
00015000.tif		0,887	0,875	0,871	0,869	0,869	0,869
00020000.tif		0,945	0,943	0,943	0,941	0,941	0,941
Average		0,914	0,905	0,902	0,899	0,898	0,897
32-bit (RGB)							
File	**N**	**8**	**10**	**11**	**12**	**13**	**14**
00000050.tif		0,168	0,165	0,165	0,165	0,165	0,165
00000200.tif		0,792	0,778	0,772	0,768	0,766	0,764
00001000.tif		0,768	0,750	0,747	0,743	0,741	0,740
00005000.tif		0,930	0,915	0,907	0,901	0,898	0,897
00010000.tif		0,951	0,941	0,934	0,929	0,926	0,924
00015000.tif		0,823	0,808	0,804	0,802	0,801	0,800
00020000.tif		0,914	0,909	0,907	0,905	0,904	0,904
Average		0,863	0,850	0,845	0,841	0,839	0,838
48-bit (RGB)							
File	**N**	**8**	**10**	**11**	**12**	**13**	**14**
00000050.tif		0,249	0,244	0,243	0,241	0,241	0,241
00000200.tif		0,958	0,955	0,954	0,954	0,953	0,953
00001000.tif		0,992	0,989	0,987	0,986	0,986	0,985
00005000.tif		1,003	1,003	1,002	1,002	1,002	1,002
00010000.tif		1,001	1,000	0,999	0,998	0,998	0,998
00015000.tif		0,990	0,989	0,988	0,988	0,987	0,987
00020000.tif		1,002	0,999	0,999	0,999	0,999	0,999
Average		0,991	0,989	0,988	0,988	0,988	0,987
SDI 20-bit (YCbCr)							
File	**N**	**8**	**10**	**11**	**12**	**13**	**14**
Average		0,883	0,878	0,874	0,871	0,868	0,867

Table A.2: LZ4 compression ratio vs. hash table size vs. color depth and color encoding.

Silesia								
File / MML	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
dickens	1.01	1.03	1.04	1.04	1.04	1.04	1.04	1.04
mozilla	1.09	1.3	1.42	1.46	1.48	1.48	1.48	1.49
mr	1.08	1.22	1.3	1.33	1.36	1.38	1.39	1.39
nci	1.1	1.44	1.86	2.09	2.38	2.39	2.39	2.39
ooffice	1.03	1.11	1.14	1.15	1.15	1.15	1.15	1.15
osdb	1	1.01	1.02	1.02	1.02	1.02	1.02	1.02
reymont	1.06	1.16	1.18	1.19	1.19	1.19	1.19	1.19
samba	1.08	1.27	1.42	1.5	1.53	1.55	1.55	1.56
sao	1	1	1	1	1	1	1	1
webster	1.02	1.09	1.13	1.15	1.15	1.15	1.15	1.15
xml	1.09	1.3	1.43	1.5	1.54	1.55	1.57	1.57
x-ray	0.997	0.997	0.997	0.997	0.997	0.997	0.997	0.997
Calgary								
File / MML	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
bib	1.02	1.08	1.11	1.12	1.12	1.12	1.12	1.12
book1	1	1.02	1.02	1.02	1.02	1.02	1.02	1.02
book2	1.01	1.06	1.07	1.08	1.08	1.08	1.08	1.08
geo	0.999	1	1	1.01	1.01	1.01	1.01	1.01
news	1.02	1.08	1.11	1.12	1.13	1.13	1.13	1.13
obj1	1.07	1.22	1.32	1.34	1.36	1.37	1.37	1.37
obj2	1.07	1.23	1.32	1.36	1.39	1.4	1.4	1.41
paper1	1.02	1.07	1.09	1.1	1.1	1.1	1.1	1.1
paper2	1.01	1.03	1.04	1.04	1.05	1.05	1.05	1.05
paper3	1.01	1.04	1.04	1.04	1.04	1.04	1.04	1.04
paper4	1.01	1.05	1.06	1.07	1.07	1.07	1.07	1.07
paper5	1.02	1.07	1.1	1.1	1.1	1.1	1.1	1.1
paper6	1.02	1.08	1.1	1.11	1.11	1.11	1.11	1.11
pic	1.25	1.99	2.78	3.21	3.73	4.02	4.19	4.23
progc	1.05	1.15	1.2	1.22	1.22	1.22	1.22	1.22
progl	1.06	1.2	1.28	1.31	1.33	1.34	1.35	1.35
progp	1.08	1.28	1.38	1.43	1.46	1.48	1.49	1.5
trans	1.05	1.18	1.26	1.29	1.31	1.33	1.33	1.34
Cantebury								
File / MML	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
alice29.txt	1.01	1.05	1.07	1.07	1.07	1.07	1.07	1.07
asyoulik.txt	1.01	1.05	1.07	1.07	1.07	1.07	1.07	1.07
cp.html	1.02	1.11	1.18	1.21	1.22	1.22	1.22	1.22
fields.c	1.08	1.25	1.33	1.36	1.37	1.38	1.38	1.38
grammar.lsp	1.08	1.3	1.41	1.46	1.49	1.49	1.49	1.49
kennedy.xls	1.16	1.54	1.82	1.82	1.82	1.82	1.82	1.82
lcet10.txt	1.01	1.05	1.07	1.07	1.07	1.07	1.07	1.07
plrabn12.txt	1	1.02	1.02	1.02	1.02	1.02	1.02	1.02
ptt5	1.25	1.99	2.78	3.21	3.73	4.02	4.19	4.23
sum	1.06	1.17	1.21	1.22	1.22	1.23	1.23	1.23
xargs.1	1.03	1.11	1.15	1.15	1.15	1.15	1.15	1.15

Table A.3: LZ4 compression ratio vs. match length limit.