THE FIRST HARDWARE MSC ALGORITHM IMPLEMENTATION

V. Fabera[*], *T. Musil*[*], *J. Rada*[*]

Abstract: This paper describes the first attempt of hardware implementation of Multistream Compression (MSC) algorithm. The algorithm is transformed to series of Finite State Machines with Datapath using Register-Transfer methodology. Those state machines are then implemented in VHDL to selected FPGA platform. The algorithm utilizes a special tree data structure, called MSC tree. For storage purpose of the MSC tree a Left Tree Representation is introduced. Due to parallelism, the algorithm uses multiple port access to SDRAM memory.

Key words: *multistream compression, MSC, FPGA, compression, parallel compression, left tree representation*

Received: January 16, 2017 **DOI:** 10.14311/NNW.2017.27.029
Revised and accepted: December 3, 2017

1. Introduction

The MSC (MultiStream Compression) is a new lossless compression method, invented by Czech scientist Jiří Kochánek. The method is based on the idea that data can be split into different parts [1]. For each part of the data, coding method that gives the best compression result is chosen. Different methods can be used that best suit the targeted objective. The methods Elias alpha and ZEBC used in this implementation aim to get the best compression rate.

Each of the parts contains own data for compression, which are arranged in streams. This method differs from other models based on splitting data into streams by the fact that in this case the streams do not contain symbols, but counters. The process of compression is rather complex and it presents the possibility of parallel processing.

Soon after the MSC algorithm invention the research was almost exclusively focused on the algorithm application, improvement of its compression rate (and less execution time) using various transformations of input data and comparison with different compression methods.

The diploma thesis [2] which tests the performance of MSC on Silesia corpus and text files concludes that MSC provides the best results without a parser in conjunction with BWT (Burrows-Wheeler Transform) and MTF (Move-to-Front). And this combination gives better compression rate than RLE (Run Length Encoding) and algorithms invented by Lempel and Ziv, namely LZC and LZSS. The

[*]Vit Fabera – Corresponding author; Tomas Musil; Jakub Rada; Czech Technical University in Prague, Faculty of Transportation Sciences, Konviktska 20, 110 00 Praha 1, E-mail: `fabera@fd.cvut.cz`, `musil@fd.cvut.cz`, `radajak3@fd.cvut.cz`

source [3] presents the advantages of LPT (Length-Preserving Transform) and SCLPT (Shortened-Context LPT) transformations with BWT when compressing large text files. It is also noted that some transformations have positive impact on execution times and the source also points out the significance of transformations that can lead to reduction of the length of compressed data to half compared to simple MSC.

In [4] it is concluded that small text files are better coded using MTF or loading word by word. In case of bigger English texts they are better coded using syllable dictionary and subsequent application of BWT compared to simple MSC.

In paper [5] the authors tested the algorithm on Silesia Corpus that contains various types of data and compared its performance with Huffman and arithmetic coding. They found out that in comparison with those methods the Multistream compression algorithm gave better results. All of the tested algorithms were used also in conjunction with BWT and MTF. They also tested the algorithm on pictures as part of the JPEG algorithm again with success. It is supposed the algorithm can be widely used for different kinds of data in telecommunications [12].

All experiments so far have been done in software. However, due to its properties the MSC algorithm seems suitable for hardware implementation due to its inherent parallel structure. This paper focuses on the implementation details of the algorithm in hardware.

2. Implementation overview

The described hardware implementation was built using RTL methodology as describe in [6]. This means that the algorithm is elaborated in the form of Finite State Machines with Datapath (FSMD). These FSMDs are formulated in VHDL [7] (Very High Speed Integrated Circuits Hardware Description Language), one of the most used languages for hardware design, and are mapped into a chosen FPGA platform.

The Fig. 1 shows simplified designed structure of the MSC algorithm HW implementation, including memories. The FSMDs representing steps of the algorithm are pictured as rectangles with sharp edges whereas memories are pictured as rectangles with rounded edges. The arrows between FSMDs show the flow of the algorithm and arrows between FSMDs and memories determine the communication with memories in each FSMD. For easier orientation FSMDs are highlighted by grey background.

The parallel segments of the algorithm are also illustrated in the scheme in the Fig. 1. Some of the memories are parallelized as well. Besides, there are nonparallel memories that are accessed from parallel blocks. To prevent collisions caused by simultaneous access to memory, an arbiter needs to be present in the design, which decides which parallel blocks are prioritized in the access over others. As different FSMDs share the memories, memory inputs need to be multiplexed, otherwise they would be driven by more sources.

Due to the high memory demand, an external SDRAM is used. Therefore a memory controller has to be part of the design serving as an interface between the memory and the compression algorithm.

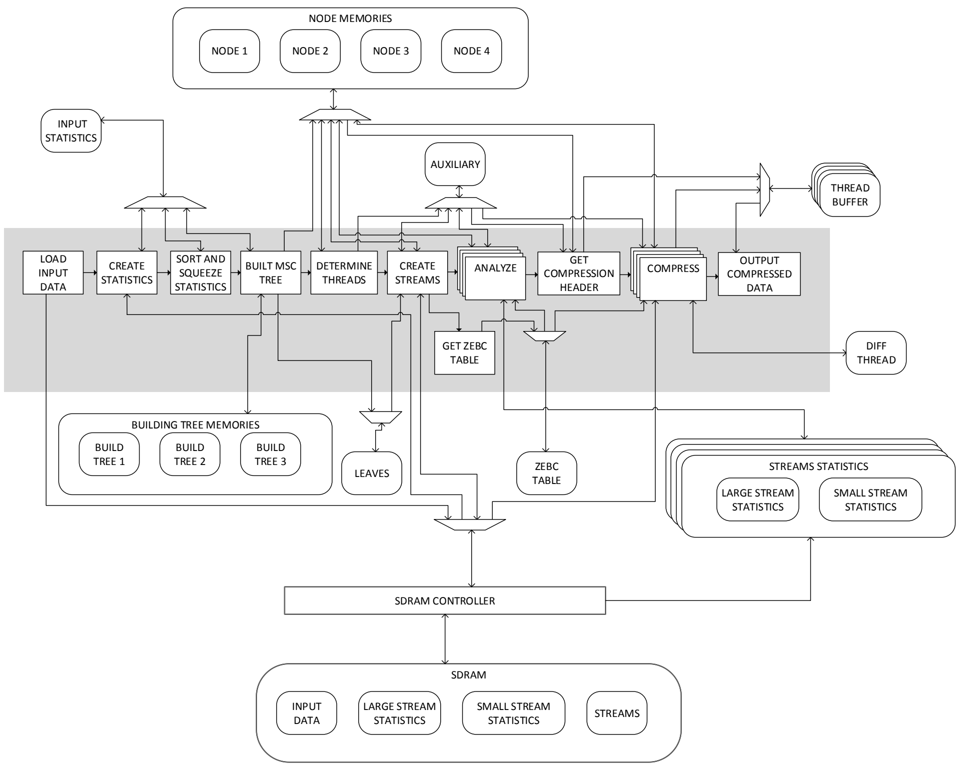

Fig. 1 *Scheme of the MSC algorithm implementation.*

3. The steps of the algorithm

The steps of the MSC algorithm are described in [5] (or detailed in Czech [1]). Thus, the paper describes only implementation details, not the working principles. The algorithm is specific of using a binary tree based upon the statistics of input data like a Huffman tree.

3.1 Creation of binary tree

Before the process of tree building is described, it is necessary to make introduction into representation of binary tree in memory. There are two widely used models of binary tree representation in memory described in Section 3.1.1a) and 3.1.1b) [8], which are found to be not suitable for MSC tree representation, as the tree has some special properties:

- each parent knows the position of left and right child
- each node is capable of determining the direction to a particular leaf
- the option to traverse only specified subtree of the MSC tree (for parallel processing)

For this purpose a novel Left tree representation is introduced. All of the considered tree representations will be demonstrated on the example of coding a word “abracadabra”. This tree can be seen in the Fig. 2.

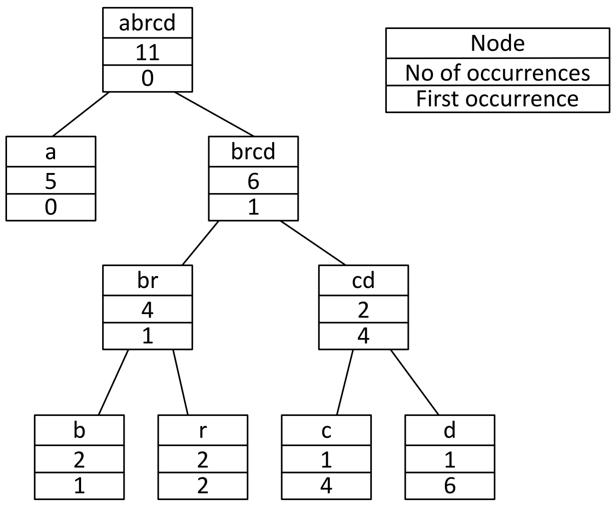

Fig. 2 *Example of MSC tree for “abracadabra”.*

3.1.1 MSC tree representation

a) Sequential representation In this representation, the position of all nodes in memory is fixed and so the root is always stored at index 0, the left child of any node with index n is stored on position $2n + 1$ and the index of right child of the same node is calculated as $2n + 2$. In case of uneven tree (tree where the depth of the leaves varies considerably), a lot of memory is needed for the representation and also big percentage ends up unused. Another problem is that when the tree is to be traversed from the root to a particular leaf, additional information is required in order to determine the correct path. For the example shown in Fig. 2, fifteen cells are needed for storage of nine nodes, leaving 40 % of memory wasted (Tab. I).

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
node	abrcd	a	brcd	–	–	br	cd	–	–	–	–	b	r	c	d

Tab. I *Scheme of the MSC algorithm implementation.*

b) Linked representation In linked representation (LR), all nodes can be stored in cells one after another, but it does not come for free – some additional information is needed. Each node must store the index of left and right child in the array. The obstacle with this representation is that only with this information, the path from root to particular node cannot be found due to the random position of each node. Each leaf of the tree would require some string of bits that would hold the information about which way to go from every node on the path.

The Tab. II shows one of the possible layouts of tree nodes in memory. It can also be seen that there is a lot of unused memory space as leaves do not have children and the path is needed for leaves only.

Index	0	1	2	3	4	5	6	7	8
Node	d	c	r	b	a	cd	br	brcd	abrcd
left_ch	–	–	–	–	–	1	3	6	4
right_ch	–	–	–	–	–	0	2	5	7
Path	111	110	101	100	0	–	–	–	–

Tab. II *Example of MSC tree for "abracadabra".*

c) Left tree representation As the previous representations are barely usable for the MSC tree, a new representation was invented for this purpose – Left Tree Representation (LTR).

It does not waste memory but each item has the deterministic position in the memory, which however depends on structure of the tree. Furthermore, it allows the tree traversing from root to leaves without any added information.

Due to this representation, each node (if it is not a leaf) knows where to find its descendants in memory and it can find a path from root to leaf due to smart organization of nodes, which is governed by following rules.

– left child of node is stored on a subsequent position:

$$\text{left\_ch\_index} = \text{parent\_index} + 1$$

– right child is stored on a position that is computed in a following way:

$$\text{right\_ch\_index} = \text{parent\_index} + \text{no\_of\_nodes\_in\_left\_tree},$$

where no_of_nodes_in_left_tree stands for number of nodes in left tree of the active node. As Fig. 3 shows, left tree of "brcd" node contains 4 nodes.

The tree in the example above would be represented as node's array in following way (Tab. III).

Index	0	1	2	3	4	5	6	7	8
Node	abrcd	a	brcd	br	b	r	cd	c	d
Number of nodes in left tree	2	1	4	2	1	1	2	1	1

Tab. III *Sequential representation of nodes in memory.*

The final test for the Left tree representation is to find the path from root to a particular leaf. The only thing that is necessary for this task is the index of the

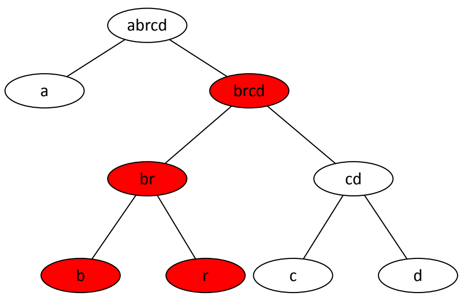

Fig. 3 *Left tree of "brcd" node.*

leaf in node's array. Then, the task comes down to comparing the leaf's index with active node's right child index. If leaf's index is smaller than node's right child index, we proceed to left child, else proceed to right child.

3.1.2 Process of tree building

The construction of a MSC tree is governed by two restrictive requirements, which causes that the process has to take place in two steps to achieve the desired representation.

As a first step, the Linked Representation of tree is created. The leaves are stored at lowest indexes from index 0 and newly created nodes are stored behind them one after another.

In the second step, the linked representation is transformed into the Left Tree Representation according to Fig. 4.

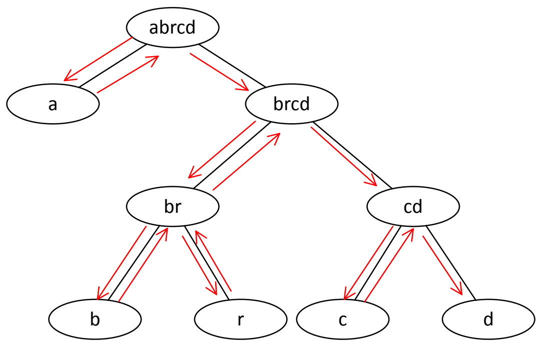

Fig. 4 *Traversing of tree during transformation from LR to LTR.*

3.2 Determination of subtrees

As some steps of the algorithm are carried out in parallel, each node needs to be assigned to a particular block. To get those blocks, a tree needs to be divided into subtrees or layers. In this implementation, the first option was chosen.

Selection of the subtrees influences the processing time of stages of the algorithm that run in parallel. The execution time of the analysis phase depends mainly on total number of counters of all nodes in particular block. Duration of the compression phase then depends considerably on number of occurrences of root node of subtree.

We select blocks from the top of the tree as the nodes have higher number of occurrences and thus higher number of counters in a stream that will be processed. However, the root of the tree has only one counter in stream, so we generally do not want it to have both children in different blocks, because during the analysis stage there would be no nodes to analyze. This step copies the SW implementation with a little simplification.

3.3 Creation of counter streams

This step was also inspired by the software implementation [1], but was adjusted for implementation in hardware. The streams are stored into two different memory structures, one for process of analysis and the other for the compression itself.

In the first case, each node of MSC tree has its own stream of counters. For memory saving purposes, this stream is divided into two structures. Higher values are stored in the structure called LARGE STATISTICS and smaller values to SMALL STATISTICS. Those are divided by a delimiter.

In the described implementation, the value of delimiter was computed with objective to minimize the memory demand of the STATISTICS. If memory is not a concern and the goal is to maximize speed of the algorithm, the delimiter value should be determined differently. The best way seems to be empiric determination by observation for various input data. The lower counters are generally more tightly packed. The values of counters from the lowest values to some set threshold after which the values become dispersed shall be stored to SMALL STATISTICS and the values above the threshold shall be stored to LARGE STATISTICS.

The counters for compression are stored into streams, whose number depends on number of parallel blocks. Each of those streams contains values of counters of nodes that belong to particular block and also counters of direct descendant nodes of particular block. Those counters are ordered according to traversing MSC tree. Their order is important.

3.4 Statistical analysis of streams

During analysis, for each parallel block each node of subtree is entered once and analyzed. Also, the total length of coded data is calculated for particular parallel block. The implementation chooses the method giving the best compression ratio. The method is picked from Elias Alpha [9] and ZEBC coding.

The subtree of a parallel block is traversed from left top to right bottom. To obtain attributes of a particular node, which are stored according to LTR, the index of the node is just incremented by one in order to proceed into the next node. Sometimes, it happens that the node on the next index is from a different parallel block. If this node is left child of its parent, then right child is entered instead. If the node is right child, the analysis is terminated (right child is at the subsequent index in LTR if the node at the current index is leaf node).

3.5 Compression

The compression is carried out as specified in the patent [1], with preservation of all items in the overhead and headers. The binary values that appear on the output from the algorithm can be either values coded by inverted Elias Alpha coding that have defined structure and variable length or it can be arbitrary binary coded value of length between 1 and 32 bits.

As the coded values appear on the output in random time and with random lengths, an element that would buffer those values is needed. A combination of 8-bit register and BRAM (Block RAM in FPGA used as medium data storage usually of the size of kBs) was selected for this task.

The register stacks and splits the variable length values so that they form 8-bit blocks. When the register is full, its content is copied into first unoccupied position in BRAM, thus buffering the values.

Generally, the data are compressed in parallel, so each parallel block has its own buffer and the last byte in the each buffer is marked by the parity bit. Also the overhead has its own buffer, but due to its known length, there is no need to denote the last byte.

4. Memory aspects of implementation

One of the principal differences between coding in SW and HW is that in the latter case the developer accesses directly memory. Thus, he must decide how to structure the data in memory, how many bits should be used or which width of the memory data bus should be used. Also, because the memory resources are very limited, the developer sometimes has to trade some saved space for performance.

4.1 MSC tree in memory

Each node of the MSC tree contains many attributes. The single value attributes of one node take up 138 bits in total. They are mapped into memories with 36-bit data bus width (maximum data bus width supported by Spartan-6 BRAMs with 18kbits of capacity). All the attributes are stored on four addresses and take up 144 bits, leaving 6 bits unused for one node.

As the maximum number of nodes in a tree in this implementation is 511 $(2 \cdot \mathit{alphabet\_size} - 1)$, it is obvious that the single value attributes will need $511 \cdot 144 = 73,584$ bits, which is nearly the maximum capacity of four 18kbit block RAMs. As the attributes of one node are stored in 4 addresses, it is advantageous to store them to separate memories. Due to this separation, all of the attributes are advantageously stored on the same address, only in different memory. Another advantage is that there is a possibility to retrieve all the attributes of a node in one clock cycle. The structure of the NODE memories can be seen in Tab. IV, V, VI and VII. Among the node attributes there are values of the statistics, number of left tree nodes, values needed during traversing of the tree or needed for compression and so on.

0	no_of_occ (35–20)	first_occ (19–4)	par_thr (3–2)	unused bits (1–0)
⋮				
510	no_of_occ (35–20)	first_occ (19–4)	par_thr (3–2)	unused bits (1–0)

Tab. IV *First block of memory containing node attributes (NODE1).*

0	counter (35–20)	symbol (19–12)	type (11–10)	left_tree (9–1)	dir (0)
⋮					
510	counter (35–20)	symbol (19–12)	type (11–10)	left_tree (9–1)	dir (0)

Tab. V *Second block of memory containing node attributes (NODE2).*

0	point_str (35–20)	point_par_thr (19–4)	thr_ind (3–2)	flg1 (1)	unused bit (0)
⋮					
510	point_str (35–20)	point_par_thr (19–4)	thr_ind (3–2)	flg1 (1)	unused bit (0)

Tab. VI *Third block of memory containing node attributes (NODE3).*

0	ls_items (35–28)	base (27–22)	flg2 (21)	unused bit (20)	sum_counters (19–4)	best_m (3–2)	unused bits (1–0)
⋮							
510	ls_items (35–28)	base (27–22)	flg2 (21)	unused bit (20)	sum_counters (19–4)	best_m (3–2)	unused bits (1–0)

Tab. VII *Fourth block of memory containing node attributes (NODE4).*

4.2 Counter statistics

Besides the attributes stored in the NODE memories, the counter STATISTICS is another attribute of node. However, it needs to be stored in external memory due to high memory requirements. It is divided into two parts by delimiter. Delimiter is a defined number that separates values of counters to statistics of low value counters SMALL STATISTICS and statistics of large value counters LARGE STATISTICS. The value of the delimiter is computed with respect to minimization of memory demand for statistics storage. The delimiter is computed in a following way, which applies for the worst case scenario.

$$\min\left((\text{delimiter}-1)\cdot 16+\left\lfloor\frac{\text{size of input data}}{\text{delimiter}+1}\right\rfloor\cdot 32\right)$$

For this particular configuration, the delimiter is equal to 361. To understand the formula above, the structure of both of the statistics have to be understood first. One item in the SMALL STATISTICS occupies 16 bits and stores only the number of occurrences of counter with particular value as the counter value is specified by the position in the array.

For large statistics this approach would not be very convenient. The occurrence of high values of counters is very sparse, which would leave us with a lot of unused memory. Instead, the value of counter as well as its number of occurrences is stored taking up 32 bits of memory and the items of large statistics are stored one after another.

To get the maximum number of the items in LARGE STATISTICS, delimiter+1 must divide the maximum number of items in the input data. For this implementation:

$$\left\lfloor\frac{65535}{361+1}\right\rfloor = 181$$

The statistics for one node will therefore take up

$$(360\cdot 16+181\cdot 32) = 5760+5792 = 11,552 \text{ bits}$$

As the statistics needs to be stored for each node separately, it consumes more than 5 Mbits of memory.

In the described implementation, the value of delimiter was selected with objective to minimize the memory demand of the streams STATISTICS. If memory is not a concern and the goal is to maximize speed of the algorithm, the delimiter value should be determined differently. It could be determined empirically by observation for various input data. The lower counters are generally more tightly packed. The values of counters from the lowest values to some set threshold after which the values become dispersed shall be stored to SMALL STATISTICS and the values above the threshold shall be stored to LARGE STATISTICS.

4.3 SDRAM interface

The SDRAM memory is high capacity memory that is needed for the capacity-demanding data structures in the hardware implementation of the MSC algorithm.

Namely, those demanding structures are INPUT DATA, STREAMS, SMALL STATISTICS AND LARGE STATISTICS.

The interface between the algorithm Finite State Machines and the SDRAM controller contains 4 equivalent ports. Each port is formed by 3 buses. The first bus is used for control of the interface as well as for manipulation of a single item of data. Each of the two remaining buses control a BRAM. Fig. 5 shows only one port of the interface.

The complexity of the interface is given by the variety of data that are transferred via this interface and also due to parallel processing. More blocks can run concurrently doing the same thing for different data.

Parallel processing is used for analysis and compression. During analysis, both SMALL and LARGE STATISTICS are needed. In this case, it is not only one value but the whole block for one particular node. For this purpose, both statistics are copied into dedicated BRAMs. The LARGE STATISTICS is modified in the process but need not be copied back to SDRAM. When the compression is being carried out, values from STREAMS are read one by one.

4.3.1 Block diagram

The block diagram of SDRAM interface captures only 1 of 4 ports. How many ports are needed (1 to 4) is determined by the number of parallel blocks used. The arrows show the direction of the communication. The n in the names of the data buses ranges from 0 to 3 and so the interface can be 4 times larger than shown on Fig. 5.

Description of signals on the interface:

- **CMD** specifies the mode of action (READ/WRITE) or idling of SDRAM.
- **SIZE** specifies the amount of transferred data. If SIZE = ‘0’, only one item is read from/written to an address specified by INDEX and POSITION. The size of an item depends on the TYPE. Else if SIZE = ‘1’, items on all POSITIONs from particular INDEX are transferred between SDRAM and particular BRAM. This option applies only for SMALL STATISTICS and LARGE STATISTICS.
- **TYPE** determines which data are manipulated. The list of used four types is in Tab. VIII.
- **INDEX** specifies exact position for block of data and rough position for specific item.
- **POSITION** specifies exact position of specific item.
- **DATA_IN** bus specifies data to be written into SDRAM in case of writing one specific item.
- **READY** is output signal to announce that DATA_OUT is ready or that writing is finished.
- **DATA_OUT** outputs the read data in case of reading one specific item.

– **CLK_MSC/CLK_SDRAM** are two different clock domains used by different interface sides

– other signals are standard signals for using BRAM memories.

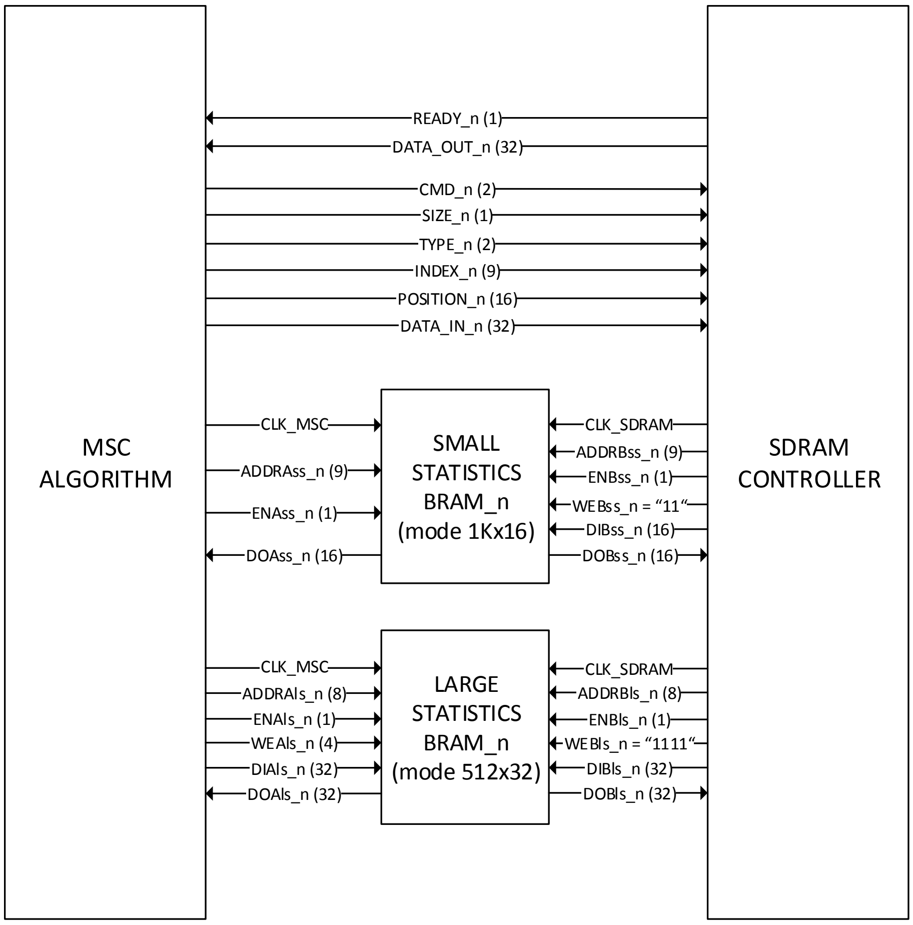

Fig. 5 *The SDRAM interface.*

type	Bit representation	Index	Position	Data_size
Input data	00	0	0-65534	8
Stream	01	0-3	0-65535	16
Small statistics	10	0-510	0-360	16
Large statistics	11	0-510	0-180	32

Tab. VIII *The SDRAM interface.*

5. Results

Due to a high range of FPGA products and variety of manufacturers a special care was dedicated to selection of the right platform. The main criteria reflected during the selection of platform were price and available resources.

Choosing an oversized platform results in unnecessary expenses, which would eventually make the implementation of design unfeasible, but the platform must dispose with enough resources for the design. As the amount of all resources grows more or less gradually with price, the decision to use external SDRAM due to high memory demand of the algorithm presented the way how to prevent wasting resources.

As the design was not optimized on any of the FPGA architectures which are offered by the manufacturers it was decided to use Xilinx platform due to wider data bus in built-in FPGA memories that brought the possibility to retrieve more data in one clock cycle compared to its competitor Altera.

As it is advised to utilize FPGA resources not more than of 80 % of its full capacity [10] a platform with the XC6SLX45 FPGA type from Spartan-6 family [11] connected to external SDRAM memory was selected. Tab. IX shows the utilization of FPGA resources. The values of utilization are obtained from the development tools after FPGA mapping and routing process. The values are taken for the algorithm utilizing SDRAM memory.

Resource	Absolute Usage	Relative Usage
Number of Slice Registers	9,265	16 %
Number of Slice[1] LUTs[2]	14,346	52 %
Number of occupied Slices	4,786	70 %
Number of used BRAMs	25	21 %

Tab. IX *FPGA resource utilization.*

Due to optimization after the synthesis process (extensive pipelining) the system frequency reaches 80 MHz. The SDRAM memory interface uses auxiliary clock domain of 320 MHz synchronous with the system frequency.

The blocks that perform operations with long strings of bits, namely comparators and adders in the analysis and build the tree modules are in our case the main aspect limiting the maximum working frequency and consequently data throughput of the algorithm. These operations are translated to 5 levels of logic (levels of combinational logic between two registers) considering Spartan-6 6-input LUTs, causing time delay that prevents us from increasing the working frequency. The working frequency could be increased in several ways using a different platform:

a) Utilization of an FPGA with generally faster logic elements, e.g. from the Virtex-6 family

[1]Repetitive structure grouping of LUTs, flip-flops and connections in Xilinx FPGAs LUTs.
[2]LUT (Look-Up Table) – n-inputs combinational function generator.

b) Utilization of an FPGA with multi-input LUTs (compared to 6-input LUT of Spartan-6), e.g. platform from Altera Stratix II series

c) Implementation of more mathematical operations using DSP blocks and logic functions transformed into block memories thus requiring a larger circuit from the Spartan-6 series due to the limited amount of DSP blocks and BRAMs

All of the options above would mean the use of more expensive FPGAs than originally chosen Xilinx Spartan-6 family.

6. Conclusion

The paper describes the very first hardware implementation of MSC algorithm. The software implementation served as a source of inspiration, however, new utilities needed to be introduced. The new way of storing nodes of the MSC tree called Left Tree Representation presented in the paper is viewed as a main contribution. It can be used not only for MSC tree but basically any binary tree.

Due to high memory demand external SDRAM is used in the design. The parallel access to the external memory generates a need of a complex SDRAM interface which consumes additional logic in the FPGA.

This initial implementation that supports the maximum length 8 bit of input data is programmed into an FPGA belonging to a low-end family. The implementation uses 2 clock domains with frequencies 80 MHz and 320 MHz. With the planned optimization of design it is expected that the clock period as well as amount of utilized resources decrease.

The MSC hardware implementation is particularly advantageous for designing FPGA compression coprocessor both in mobile applications or end devices and in high performance computers - here specifically in the form of PCI-Express card. Nevertheless, it is difficult to carry out a fair performance comparison between software and hardware implementation of the algorithm due to lack of a suitable methodology for comparison of both platforms itself. The methodology sometimes used in signal processing i.e. comparison of number of floating point operations per second is not applicable for this purpose. In the case of PC and dataflow comparison, the full data path must be taken into account including data transmission via PCI-Express bus. Rigorous comparison will be provided after designing complete PCI-Express MSC card in further experiment.

References

[1] KOCHÁNEK J. Způsob transformace a bezeztrátové komprimace dat v elektronické podobě. Czech Republic: Patent Application, 2007. Appl. no. 2007-114.

[2] UZEL P. *Entropic coders.* Prague: UK 2009, Diploma thesis, UK, Faculty of Mathematics and Physics, Department of Software Engineering.

[3] UNGER L. *Improvements of Multistream compression.* Prague: UK 2010, Diploma thesis, UK, Faculty of Mathematics and Physics, Department of Software Engineering.

[4] JELÍNEK J. *Suitable methods of data prepration for mutistream compression.* Prague: UK 2011, Bachelor thesis, UK, Faculty of Mathematics and Physics, Department of Software Engineering.

[5] KOCHÁNEK J., LÁNSKÝ J., UZEL P., ŽEMLIČKA M. *The New Statistical Compression Method: Multistream Compression.* In: First International Conference on the Applications of Digital Information and Web Technologies [online], Ostrava: IEEE, 2008 [cit: 07/17/16]. Available on: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4664366.

[6] CHU, P. P. *RTL Hardware Design Using VHDL.* Cleveland: John Wiley & Sons, Inc, 2006. ISBN: 978-0-471-72092-8.

[7] IEEE Standard VHDL Language Reference Manual," in IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002), pp. c1–626, [online][cit: 07/15/16]. Available on: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp={&}arnumber=4772740.

[8] ROHINI S. Representation of binary tree in memory, 2013 [online] [cit: 06/27/16]. Available on: http://www.slideshare.net/rrohinishinde/representation-of-binary-tree-in-memory.

[9] TRIVIÁLNÍ ALGORITMUS PRO VYHLEDÁVÁNÍ VZORU. *Dokumentografické informační systémy – Komprese.* [Online] [cit: 07/17/16]. Available on: http://www.ms.mff.cuni.cz/~kopecky/vyuka/dis/11/dis11_v1.html.

[10] FPGA CENTRAL. *FPGA Device Selection [Presentation].* Published on: 2009 [online] [cit: 08/15/16]. Available on: http://www.slideshare.net/vkr101/fpga-device-selection.

[11] XILINX. [online]. *Spartan-6 Family Overview.* 2011 [cit: 07/19/16]. Available on: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf.

[12] JANESOVA V., DOUDA V.: Predictive Model and Methodology for Optical Telecommunications Infrastructure. Neural Network World, 26(4), pp. 351–362, 2016, doi: 10.14311/NNW.2016.26.020.