ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 2, No. 2, 2017

ORDERED ACCESS MEMORY AND ITS APPLICATION IN PARALLEL PROCESSORS

Anatoliy Melnyk

Lviv Polytechnic National University, 12, S. Bandera str., Lviv, 79013, Ukraine
Authors e-mail: *aomelnyk@lp.edu.ua*

Submitted on 01.12.2017

Abstract: **In this paper, after analyzing the known memory access methods, conventional memory organization and its challenging problems, we propose new ordered memory access method and a new type of memory – the ordered access memory. This method is aimed at working with data arrays and provides memory access in the prescribed manner. Proposed method unlike widely used method of sequential memory access allows extending the functionality of the memory as it provides not only sequential, but also any other ordered memory access. Unlike another widely used method of address memory access, the implementation of the proposed method provides parallel conflict-free memory access. It also allows eliminating data binding to a specific memory location that makes it possible to disintegrate the apparatus for data ordering and eliminates the need to store addresses of locations the data are placed in, and the need to submit the address to the address inputs during data writing and reading.**

The new method distinctive features compared to the known memory access methods are considered. Input data, their indices and output data of the ordered access memory are described as well as the approaches to this type of memory design and use. The interface of the ordered access memory is considered as well as its advances compared to the random, associative, and sequential access memories. An example of the ordered access memory usage in application-specific processors with parallel and pipeline structures is demonstrated and the results of the ordered access memory implementation in FPGA are considered.

Index Terms: **Computer memory, Memory wall, Memory types, Parallel memory, Memory access method, Ordered access memory, Memory organization, Parallel processors.**

I. INTRODUCTION

Computers have become a mandatory component of our life. They are used for solving science and engineering problems including the design of automobiles, buildings, electronic devices, aircrafts, medications, and robots. Their using makes automobiles safer, more aerodynamic, more comfortable, and more energy-efficient. The more powerful computer is used the more considerable effect of it. For example, exascale computing is used to understand economics, national security, and setting public policy. Billions of processor hours are applied to understanding and predicting climate change. With faster computer systems scientists and engineers could simulate critical details – such as clouds in a climate model or mechanics, chemistry, and fluid dynamics in the human body [1,2].

Main direction of computer systems performance increasing is their parallelization. Parallel multiprocessor systems on a chip became the mainstream products of the microprocessor industry. Although software developers have found the ways to use multicore systems by running independent tasks on each core, they have not, for the most part, parallelized individual tasks in such a way as to make full use of the available computational capacity [3]. Moreover, if industry continues to follow the same trends, they will soon be delivering chips with thousands of cores. Harnessing these cores will require new techniques for parallel computing, including breakthroughs in software models, languages, and tools. From the other side, most software developers today think and program by using a sequential programming model to create software for single general purpose microprocessors, for single dedicated microprocessors, including digital signal and image processors, and for single application-specific processors as FFT, FCT, AES etc. The problem to increase performance of these single processors, which will be the base for the future multiprocessor systems, is very important.

To solve this problem the following improvements must be done: reduced memory wall; provided parallel conflict-free memory access; created new model of computing which has no limitations of John von Neumann one. We see the way to realize it by creation of the new parallel memory architecture and propose new type of computer memory with parallel ordered memory access.

The primary systematic feature of computer memory is the memory access method that is implemented in it [4]. This feature designates an ability of memory in relation to the conflict-free parallel access. Such access is not fully provided by any modern type of memory: random, associative, and sequential access memories that significantly complicates the organization of the computer and results in its low performance. The task arises to find new memory access methods.

In works [5–8] the new memory access method and the memory device that implements this method were proposed. During last years this method and the memory

device characteristics have been improved [9, 10]. This new device we will further call the ordered access memory (OAM).

The ordered access memory is dedicated to a work with arrays of data. This memory provides access in the prescribed manner, i.e. index, which enters the memory together with data, or during its reading, indicates its place in the output array. To clearly allocate features of the new computer memory type the ordered memory access method will be explored in this article as well as its distinctive features compared to the known memory access methods. Input data, indices, and output data of the ordered access memory will be described as well as approaches to its design and use. The interface of these types of the OAM will be further considered as well as advances of the OAM comparing to random, sequential and associative access memories. The results of the OAM implementation in FPGA are considered. At the end of this article the OAM usage in application-specific processors with parallel and pipeline structures is demonstrated.

II. MOTIVATION

Four major issues motivate this work: (1) the challenge to find the approaches of overcome or to reduce the memory wall; (2) the need for parallelizing processor work; (3) the need for performing data ordering inside memory; (4) the need for creating the base for the new model of computing which has no limitations of John von Neumann one.

A. *Memory wall*

The challenging problem in high-performance computing is the memory wall. The memory wall, the growing mismatch between memory bandwidth and processor cycle time, is a major factor limiting performance [1, 11–13]. On many applications modern processors are limited by memory system performance to a small fraction of peak performance. Despite of the tremendous changes that have occurred in the computer memory design technology, which have led to dramatically improving of its characteristics [14], this problem appears to be growing worse over time. The memory wall is a special case of non-uniform scaling. As technology improves, different aspects of technology scale at different rates, leading to disparities in system parameters. When these disparities become large enough, or when a new technology is introduced, there is a discontinuity in system design that calls for innovative architecture.

To address the memory wall problem, innovative architectures are needed that increase memory bandwidth. If make a look inside the memory we will find that the main contribution to memory bandwidth slowing accounts for the tools of address access to its locations. These tools introduce the main delay to the time of memory access, moreover with the grown of the memory size this delay increases. With design technology improvement the characteristics of the memory components become better but at the same time its size grows that lids to its relative slowing. To improve memory bandwidth modern DRAM devices store data in banks which can be indexed by row and column addresses [15]. A memory access to such a DRAM device requires besides the data transfer the following operations: bank precharge, row activate and column access [16]. Finding a schedule of these operations with the minimum schedule length is the focus of some works [17, 18], proposed techniques of which group the accesses of the same row together and interleave the execution of memory accesses from different banks. It allows improving DRAM's speed and throughput. But the number of banks is limited by the complexity of memory controller that does not allow to solve the memory wall problem dramatically.

In our opinion the growing mismatch between memory bandwidth and processor cycle time can be overcome by creating of the computer memory, organization of which does not require defining the place where data is stored.

B. *Processor work parallelizing*

The problem to increase performance of the single processor can be solved using beside increasing its clock frequency and overcoming the memory wall by parallelizing its work.

Algorithms of most problems solving have natural parallelism and suppose parallel data processing. The level of parallelism is limited by the access methods of the modern types of the computer memory. It does not allow solving these problems in appropriate time. Parallelizing of the processors work required by an application can be achieved by creation of the new memory architecture with parallel conflict-free access.

C. *Data ordering inside memory*

The need to order data arises in process of many problems solving and, as it is pointed in [19], over 25 % of machine time is spent on this procedure. In many areas, such as matrix computations, digital signal and image processing, graph theory and combinatorial optimization, there are algorithms with data dependencies, flow graphs of which have irregular structure and are characterized by the low efficiency in the case of their implementation in the multiprocessor systems as it requires intensive data exchange between processors. Their sequential execution requires performing of intermediate data ordering. For their parallel execution a lot of multiprocessor architectures based on the communication networks [20–22] have been proposed. In some of them the sorting networks are used for processors and memory modules connection [23, 24]. It is the reason of large activity in the direction of sorting networks VLSI implementation [25, 26].

Above pointed forms need for creation parallel memory with performing data ordering inside it.

Moreover, there is a need for memory with new properties, which are not available for modern memory types. Specifically, while performing intensive calculations on data arrays which are necessary, for example, in solving multimedia and telecommunication problems, the memory must provide storing of data arrays coming from many channels simultaneously with ordering data in the arrays and reading previously accepted data arrays for their processing in multi-ALU units, i.e. in parallel [27, 28]. Performing these functions using the existing types of memory is a complex and often impossible task with acceptable performance, due to their potential limitations [29–31].

Therefore, there is a need for extension the memory functionality to enable its use for solving the above-mentioned problems of working with arrays of data by including to its functions the procedure of data ordering in arrays during their writing, saving and reading.

D. *THE BASE FOR THE NEW MODEL OF COMPUTING*

Memory is one of the main elements of computer. Its organization defines used in computer model of computing. Traditionally, John von Neumann model of computing is based on the address access memory or RAM. Known associative model of computing is also based on corresponding memory organization. Thus, new memory access method may serve as a base for the new model of computing.

III. BACKGROUND

A. *KNOWN MEMORY ACCESS METHODS*

As we already know, the memory consists of locations storing data, where data are written in through the input ports and are read out through the output ports. Let us define the memory access method as a sequence of actions that are performed under content of the memory locations directed to write and read data. There are several memory access methods.

The most used in modern computers is the address memory access method. The memory MEM includes tools AALi of address access to each i-th location where data are being stored, in addition to M locations Li, where i is a location number (i = 0,1, ... M-1), i.e.

$$MEM = \{Li, AALi, i = 0,1, ... M-1\}.$$

According to this method, the address of the memory location (its number) is submitted to the memory together with each data while it is writing in to the memory and while it is reading out from the memory. This address identifies the location in a way that the tools AALi write data in to this location, or read it out from this location. That the content of the memory location is determined by the expression

$$Li\ [0 – (n-1)] = \{ID\ [0 – (n-1)], A = i\},$$

where ID is n-bit input data; A is its address.

Output data OD is determined by the expression

$$OD\ [0 – (n-1)] = \{Li\ [0 – (n-1)], i = A\}.$$

The main feature of the sequential memory access method is that data are placed in the memory locations sequentially. Data are read out from the memory at the same order as they are written in to the memory, or in reverse order. Direct order is provided by the FIFO, “First In – First Out”, memory. Reverse order is provided by the LIFO, "Last In – First Out”, memory. The memory MEM includes M locations Li, where i is a location number (i = 0,1, ... M-1), where data are being stored, i.e.

$$MEM = \{Li, i = 0,1, ... M-1\}.$$

The content of the memory location is determined by the expression

$$Li\ [0 – (n-1)] = \{ID_i\ [0 – (n-1)]\},$$

where ID is n-bit input data.

Output data OD is determined by the expression

$$ODi\ [0 – (n-1)] = \{Li\ [0 – (n-1)]\}.$$

According to the associative memory access method data is stored in the memory location together with its key and is loaded from the location when its key coincides with the given one. The memory MEM includes tools AALi of associative access to each i-th location where data are being stored, in addition to M locations Li, where i is a location number (i = 0,1, ... M-1), i.e.:

$$MEM = \{Ki, AALi, i = 0,1, ... M-1\}.$$

Tools AALi compare the keys of all data placed in the memory with the given one and find data to be read. The contents of the i-th location is determined by the expression

$$Li\ [0 – (k-1), 0 – (n-1)] = \{DK[0 – (k-1)], ID[0 – (n-1)]\},$$

where DK is k-bit data key, and ID is n-bit input data.

Output data OD is determined by the expression

$$OD\ [0 – (n-1)] = \{Li\ [[0 – (n-1)], DK = GK\},$$

where GK is the given key.

B. *CONVENTIONAL MEMORY ORGANIZATION*

Depending on the access method that is implemented in computer memory, it can be classified as follows [4,11-13]:

- The Random Access Memory, where address memory access method is implemented. This memory allows to write in or to read out data in each cycle by an arbitrary address.
- The Sequential Access Memory, where sequential memory access method is implemented. This memory allows to write in or to read out data sequentially in each cycle one after another.
- The Associative Access Memory, where associative memory access method is implemented. In this memory data are searched by their keys.

The Random Access Memory (RAM) is used mostly in modern computers. It consists of locations, each of which stores a unit of information called a word.

Word has n bits, where n is a word length. Memory locations are numbered so that each of them has its own number, or address. The same address has the word that is stored in this location. That is this address points the place of the word in the memory. If memory can hold M words, the numbers from 0 to M-1 are used as addresses. If binary coding is used, m bits are necessary to represent all addresses, where m = Ceiling ($\log_2 M$).

The RAM performs two operations: to write and read. In write mode data and its address are sent to the RAM inputs and by the clock signal data is written to the memory location specified by the address. In read mode address of data is sent to the RAM input and by the clock signal data is read out to the RAM output from the memory location specified by the address. The RAM organization is shown in Fig. 1.

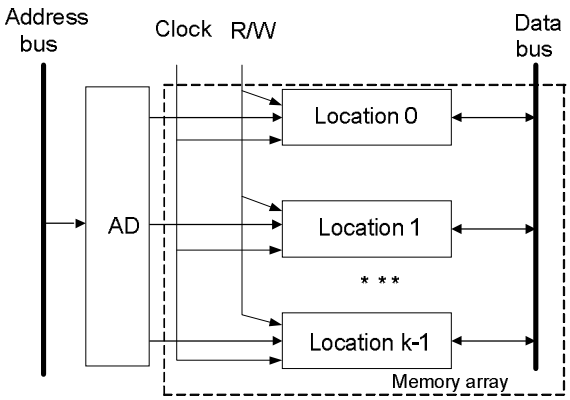

Fig. 1. The RAM organization

The RAM interface consists of m-bit address bus, n-bit data bus, and R/W and clock inputs. In write mode data is written only to the register pointed by according output of address decoder AD. In read mode data comes to the data bus from register pointed by according output of address decoder AD. The outputs of the other registers at this moment are in a high impedance state.

The Sequential Access Memory (SAM) is also widely used in computers. Data are read out from the SAM in the order they were written in or in reverse order. An example of the SAM organization is shown in Fig. 2, where in write mode data come from data bus and are stored in the sequentially connected registers R_0, R_1, ... R_{K-1}, and in read mode come to data bus from these registers.

This memory is the fastest one and is simple for implementation.

The Associative Access Memory (AAM) stores data together with their keys. The AAM organization is shown in Fig. 3. The registers are used here as the memory locations. In write mode data are written in to arbitrary available registers through entering-fetching device EFD. In read mode data are read out from the AAM through EFD when their keys are coinciding with the key K in the key register KR. The comparator CP compares keys in the registers with the key in KR.

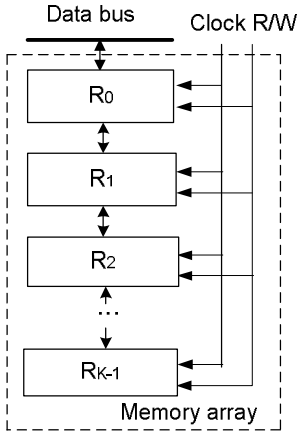

Fig. 2. The FILO SAM organization

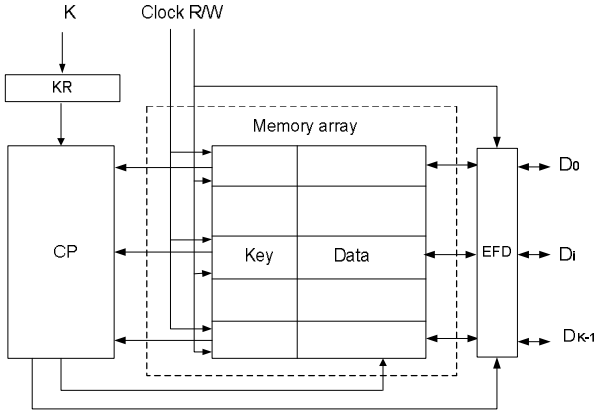

Fig. 3. AAM organization

The possibility of parallel search is the main advantage of the AAM. This memory allows simultaneous reading of all data with the same key, what provides its widespread use in computers.

C. *Challenging problems of the conventional memory usage in high-performance computers*

Each memory type shown above brings the problems when it is considered for using in high-performance computers. The challenging problem of the RAM is its inability for parallel access from many ports. The reason lies in the conflict in case of addresses convergence on multiple memory ports in write mode. This is exactly the reason of inability to implement parallel RAM-based processors. Two other problems: the need to store addresses of memory locations where data were written in so that if necessary they can be found and read out, and the need to put the addresses at the address input of the memory in both write and read modes.

The main disadvantage of the SAM is that it takes a while for particular data searching. In the worst case it may require reading of all previously written data. In addition, this memory has low functionality as it does not provide more than sequential memory access. And

the last thing – this method is designed to work with vector data and does not provide parallel memory access.

The challenging problem of the AAM is the need to provide access to each location from its ports and the need to compare simultaneously the keys in all its registers with the given one that requires large equipment volume and slows down access time. In addition, this method allows parallel memory access only to data with the same key that limits its application.

IV. Method of ordered memory access

The goal of developing the above mentioned method is to expand the functions of memory by allowing the parallel conflict-free memory access and ordering data according to their indices during their writing, storing or (and) reading.

This memory consists of locations where data array may be written in, stored, and read out. This method supposes to perform the following steps:

- an index is assigned to each input data whose numeric value determines the place of this data in the output array;
- the indices are processed, for example, they are sorted in the ascending or descending order of their numerical values;
- data of the input array are ordered according to the values of their indices and the output data array is formed.

As it follows from the above described, the ordered memory access method, unlike the sequential memory access method, allows extending the functionality of the memory, as it provides not only the sequential, but also any other ordered access.

Unlike the address memory access method, the implementation of the ordered memory access method allows eliminating binding data to a specific memory location through eliminating the need to use the addresses during data writing and reading. This eliminates the need to store the addresses of locations where data are placed, and the need to submit the address on the address input of memory within both data write and read modes because, according to the proposed method, there is an only requirement to enter an index with each data during its writing into the memory, which indicates the place of data in the output data array, and to organize memory in such a way that provides output data reading in the order specified by their indices.

V. Ordered access memory organization

The ordered access memory includes tools OALi of ordered access to each i-th location in addition to M locations Li, where i is a location number (i = 0,1, ... M-1), where data are being stored, i.e.:

$$OAM = \{Li, OALi, i = 0,1, ... M-1\}.$$

Tools OALi compare the indices of all data placed in the memory and order data according to their indices. Output data OD is determined by the expression

$$OD_{SID}\,[0 – (n-1)] = \{ID\,[[0 – (n-1)]\},$$

where ID is n-bit input data; SID is the index of input data ID.

Organization of the memory that implements proposed ordered access method is shown in Fig. 4.

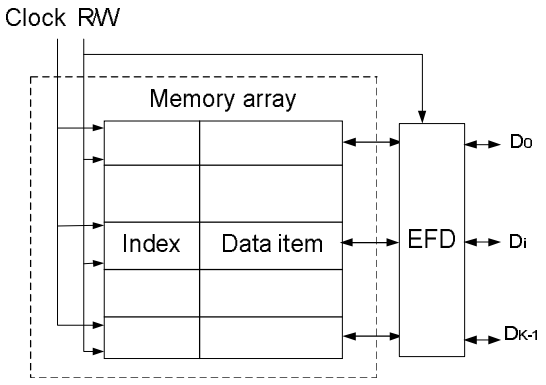

Fig. 4. The OAM organization

The ordered access memory consists of the memory array with P locations to store the data items and their indices, where P=kl, k is the number of the data items in the column of the input data matrix, l is the number of the data items in the row of the input data matrix. It also consists of an entering-fetching device (EFD) which enters the input data items into the memory locations, fetches the output data items from the memory locations, and forms the output data matrix with P data items, where P=mn, m is the number of the data items in the column of the output data matrix, n is the number of the data items in the row of the output data matrix. The input data items and their indices are written into the memory array by the rows of the input data matrix and the output data items are read out from the memory array by the rows of the output data matrix using R/W signal.

The input data array that is written in to the memory is served as a matrix. The input data are written in to the OAM from *l* ports by rows of the matrix

$$\begin{vmatrix} ID_{0,0} & ID_{0,1} \ldots & ID_{0,l-1} \\ ID_{1,0} & ID_{1,1} \ldots & ID_{1,l-1} \\ \ldots & & \\ ID_{k-1,0} & ID_{k-1,1} \ldots & ID_{k-1,l-1} \end{vmatrix},$$

where $ID_{i,j}$ is the input data which is placed in the i-th row (i = 0,1,…k-1) and j-th column (j = 0,1,…*l*-1) of the input data matrix.

Output data are read out from the OAM to the m ports by the rows of the matrix

$$\begin{vmatrix} OD_{0,0} & OD_{0,1} \ldots & OD_{0,n-1} \\ OD_{1,0} & OD_{1,1} \ldots & OD_{1,n-1} \\ & \ldots & \\ OD_{m-1,0} & OD_{m-1,1} \ldots & OD_{m-1,n-1} \end{vmatrix},$$

where $OD_{s,t}$ is the output data which is placed in the s-th row (s = 0,1,…m-1) and t-th column (t = 0,1,…n-1) of the output data matrix.

The matrix of the indices that are assigned to each item of the input data array appears as follows

$$\begin{vmatrix} SID_{0,0} & SID_{0,1} \ldots & SID_{0,l-1} \\ SID_{1,0} & SID_{1,1} \ldots & SID_{1,l-1} \\ \multicolumn{3}{c}{\ldots} \\ SID_{k-1,0} & SID_{k-1,1} \ldots & SID_{k-1,l-1} \end{vmatrix},$$

where $SID_{i,i}$ is the index of input data $ID_{i,i}$ which is placed in the i-th row (i = 0,1,…k-1) and j-th column (j = 0,1,…*l*-1) of the input data matrix.

The matrix of indices can come into the OAM together with input data or serve as the base for forming the ordering code that is sent to the OAM.

VI. OAM COMPARISON WITH THE OTHER TYPES OF MEMORY

The OAM has some important advantages compared to the sequential access memory, the RAM, and the associative memory.

Supposing there is a need to order data in input data matrix IDM with k=3, *l*=4, according to data indices from index matrix IM with the purpose to obtain output data matrix ODM with m=2, n=6, as it is shown in Fig. 5.

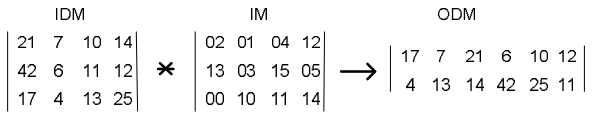

Fig. 5. Example of data ordering

The SAM is not suitable to perform data ordering as it provides only direct (FIFO) or reverse (LIFO) order of data in ODM as compared to IDM.

If the RAM will be used for data ordering then it will need to perform 12 operations of data writing in to the 12 locations of the RAM by specifying the address of every location, and 12 operations of data reading out from the 12 locations of the RAM, also by specifying the address of every location. Moreover, all mentioned operations will be performed sequentially.

In contrast, when the OAM with 4 inputs and 6 outputs will be used, there will be a need of only 3 operations of data writing in to the OAM together with their indices and 2 operations of data reading out from the OAM. In total 5 operations will be performed instead of 24 operations when ordering is performed in the RAM.

The use of the associative memory for data ordering will discard advantages of the associative search as there is a need to perform separate search for every data according to its index that requires a lot of time. That makes the associative memory even less suitable in this case then the RAM is.

In common, the main advantages of the OAM comparing to the RAM are the following:

The OAM is multiport and allows simultaneous parallel conflict-free access to data in matrix from all its ports when the RAM is single-port. Then the OAM allows data ordering in the matrixes simultaneously with data storing. This operation is frequently used and is usually time-consuming.

There is no need to save information concerning the data place in the OAM. Here it is enough to point index of data in the write mode. And opposite, the RAM usage requires data address to be stored, that complicates the computer and increases hardware volume.

As data are not tailed to the memory locations, the OAM has not got complex and slow address decoders as the RAM has got. The functions of data ordering can be disintegrated, which makes it possible to decrease delay and improve the memory speed.

Thus, the OAM has a new architecture which allows parallel conflict-free access to data and includes disintegrated apparatus for it that has less delay comparing to accordance apparatuses used in the existing memory types and higher clock frequency.

In order to compare the OAM with the RAM, we have implemented these two memory types in the same FPGA from Altera (RAM was implemented without embedded RAM blocks usage to make the comparison most adequate).

The OAM IP Core pinout is shown on Fig. 6.

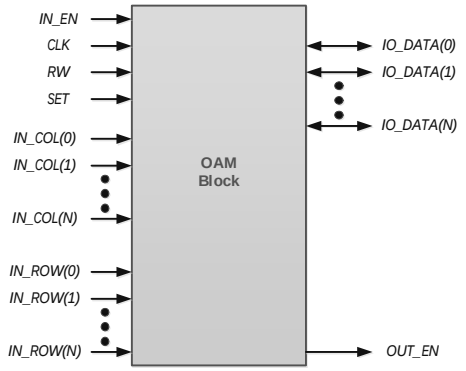

Fig. 6. OAM IP Core pinout

The OAM IP Core interface is described in the Table 1.

The OAM IP Core has the following features:

- Configurable Data width;
- Configurable number of data channels;
- Memory configurable capacity;
- Simultaneous access to data from ports without
- conflicts;
- Implementation of data ordering in matrixes;
- Need not to save information concerning data place.

The OAM parameters are depicted in the Table 2.

The bandwidths of the 8-port 32-bit OAM and the 32-bit RAM of the same capacities are shown in Fig. 7.

As it follows from this chart, the bandwidth of the 8-port 32-bit OAM is approximately of one order of magnitude higher than the bandwidth of the 32-bit RAM, and as the RAM capacity grows, its bandwidth decreases, while in case of the OAM the bandwidth does not depend on the capacity.

Table 1

The OAM IP Core interface

Pin	Activity	Description
CLK	Rising Edge	Clock
IN_EN	HIGH	Memory enable flag
RW	Read – LOW Write – HIGH	Read/Write signal
SET	Rising Edge	Reset
IN_COL(0), IN_COL(1), …, IN_COL(N)	–	Output data column index (M bits)
IN_ROW(0),IN_ROW (1), …, IN_ROW(N)	–	Output data row index (K bits)
IO_DATA(0), IO_DATA(1),…, IO_DATA(N)	–	In/Out Data (L bits)
OUT_EN	HIGH	Output data flag

Table 2

The OAM parameters

Memory capacity	1024x8x32
Number of data channels	8
Data width	32
Device Family	Xilinx xc7v2000t -2
System Clock f max	444.6MHz
Bandwidth	14.9 GBytes/s
Slice registers:	738324
IOs:	365
Synthesis and Implementation tool	Xilinx ISE 14.5

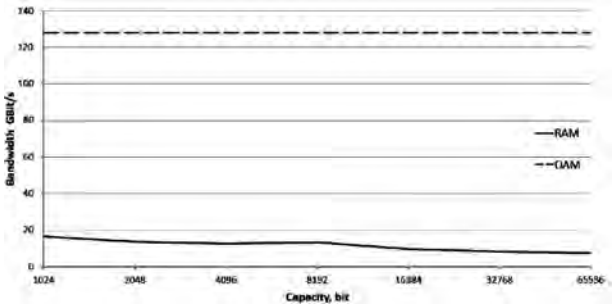

Fig. 7. Comparison of the bandwidths of the 8-port 32-bit OAM and the 32-bit RAM of the same capacities

VII. OAM-BASED PARALLEL PROCESSORS

The OAM can be used in computer systems as the buffer multiport memory, multiport memory of the processor, and multiport memory of a multiprocessor system. Many schematic solutions for computer systems, where the parallel OAM with separate data input and output ports or with joint data input and output ports could be used, can be proposed.

For example, among typical procedures those are often used for image processing and are perfectly suitable for the OAM conception: image rotation on a given angle, separation of an image part, image scaling; buffering and reordering data of an image, image processing. Some of these procedures can be fully implemented on the OAM base. In some of them the OAM could be used as an intermediate device between the image processing stages.

As an example, two structures of application-specific processor (ASP) based on the parallel OAM are shown below.

Fig. 8 depicts the OAM-based ASP of a parallel structure. In addition to the OAM it includes parallel ALU and control unit. ALU implements the operation of the algorithm that the ASP is designed to perform for. The control unit forms the control signals and data indices for the OAM, operation codes for parallel ALU, organizes cycles and performs ASP common synchronization.

As it can be seen, the ASP memory, which is the OAM, stores input data from n ports, stores intermediate data from *l* outputs of the parallel ALU, submits the intermediate data through m ports for processing in parallel ALU, and submits output data through k ports to the output. Also in this memory data are ordered according to the requirements of the algorithms implemented in the ASP.

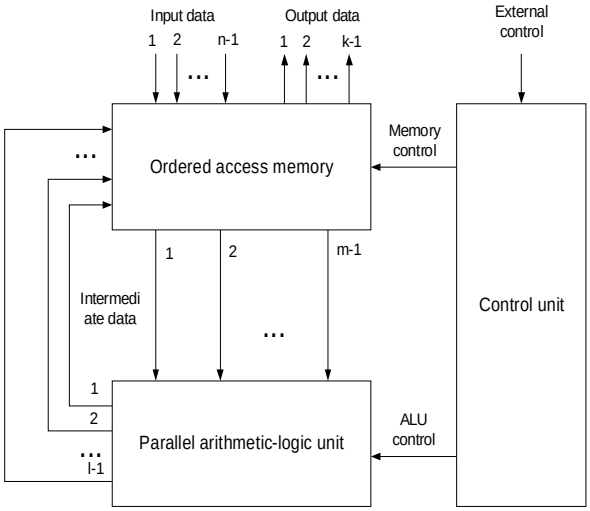

Fig. 7. Application-specific processor of a parallel structure

Fig. 8 depicts the ASP of a pipeline structure where the parallel OAM is used as the memory of its stages.

In this structure each OAM performs the following functions: receiving input data through the input ports, storing input data, ordering input data according to the digital values of indices, received from the control device, submission of ordered data to the output ports.

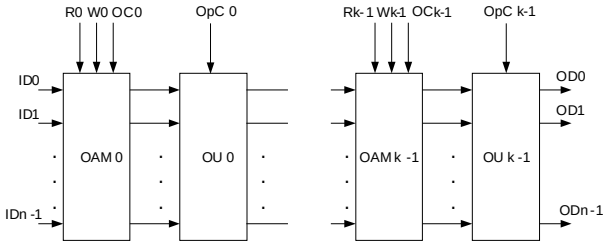

Fig. 8. Application-specific processor of a pipeline structure

Here OU is Operating Unit, R denotes Read, W denotes Write, OpC is Operation Code, OC is Ordering Code. The number of input and output ports in each OAM can vary as well as different can be the number of input and output ports in the OAM of each stage. Noting that the above described functions can't be provided by any of the existing types of memory, except of the parallel OAM.

Thus, as it follows from above considered, the OAM usage in processors allows to reduce the memory wall; to parallelize processor work by the way of multichannel data processing and to perform data ordering inside memory. In addition, it allows simplifying the control unit of the application-specific processor and its design flow respectively.

Creation of the OAM can be the base for the new model of computing which has no limitations of John von Neumann one.

VIII. CONCLUSIONS

There are several memory access methods: address, sequential and associative. The challenging problem of the memory types, which realize these methods, is their inability for parallel conflict-free memory access from many ports. The major issue, that has motivated this work, is the need to find an approach to create the high bandwidth computer memory with parallel conflict-free memory access. As the decision, an ordered memory access method and the ordered access memory are proposed in this article. The proposed OAM is targeted to work with data arrays and provides memory access in the prescribed manner, i.e. index, which enters in to the memory together with data, indicates its place in the output array. The proposed method unlike the method of sequential memory access allows extending the functionality of the memory as it provides not only sequential, but also any other ordered memory access. Unlike the method of address memory access the implementation of the proposed method provides parallel conflict-free memory access. It also allows eliminating data binding to a specific memory location that makes it possible to disintegrate the apparatus for data ordering and eliminates the need to store addresses of locations the data are placed in, and the need to submit the address to the address inputs within data writing and reading.

Implementation in FPGA has shown that the bandwidth of the 8-port 32-bit OAM is approximately of one order of magnitude higher than the bandwidth of the 32-bit RAM. It opens the opportunities for the OAM to be widely used as the multiport memory in computer systems, computer and telecommunication networks, application-specific processors, high-performance computers, multiprocessor systems, computers of a new architecture. To show the fields of application, ordered access memory usage in application-specific processors of a parallel and a pipeline structures was demonstrated in the article.

REFERENCES

[1] National Research Council. The Future of Computing Performance: Game Over or Next. Level? The National Academies Press. 2011.

[2] 21st Century Computer Architecture A community white paper May 25, 2012 http://www.cra.org/ccc/files/docs/init/21stcentury architecturewhitepaper.pdf

[3] Richard C. Murphy. On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance. In IEEE International Symposium on Workload Characterization 2007 (IISWC2007), September 27–29, 2007

[4] Melnyk, A., Computer architecture, Lutsk regional printing. Lutsk, 2008.

[5] A. Melnyk. Buffer memory device. USSR patent No. 1479954, issued at 1989.

[6] A. Melnyk. Sorting memory devices for digital signal processing systems. 1-th Ukrainian conference "Signal processing and image recognition". Kyiv, 17–21 of November 1992. – pp. 187–188.

[7] A. Melnyk. Design principles of buffer sorting memory // Proceedings "Computer engineering and information technologies". Lviv Polytechnic State University, 1996. – No. 307. – pp. 65–71.

[8] A. Melnyk. Real-time application-specific computer systems. Lviv Polytechnic State University, 1996. – 60 P.

[9] A. Melnyk. Structure organization of ordered access memory based on the tunable sorting networks. // Informatics and computing technique. University "Ukraine", 2011, pp. 34–46.

[10] A. Melnyk. Ordered access memory. – Lviv Polytechnic Publishing House, 2014. – 330 p.

[11] Patterson, D. and Hennessy, J., Computer Architecture. A Quantitative Approach, Morgan Kaufmann Publishers Inc., 1996.

[12] Stallings, W., Computer Organization and Architecture, Pearson, 10th ed., 2016.

[13] Tanenbaum, A., Structured Computer Organization, 6th ed., Pearson, 2013.

[14] Bruce Jacob. Memory Systems: Cache, DRAM, Disk / Bruce Jacob, Spencer Ng, David Wang, Morgan Kaufmann Series in Computer Architecture and Design, 2007.

[15] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. High-performance drams in workstation environments. IEEE Transaction on Computer, 50(11):1133–1153, 2001.

[16] J. Shao and B. T. Davis. A burst scheduling access reordering mechanism. In HPCA '07: 13 th International Symposium on High-Performance Computer Architecture, Phoenix, AZ, USA, Februaru 10–14, 2007.

[17] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Meikang Qiu, Yingchao Zhao, Edwin H.-M. Sha. Minimizing Memory Access Schedule for Memories. The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09), 2009.

[18] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Qingfeng Zhuge, Yingchao Zhao, Edwin H.-M. Sha. Memory Access Schedule Minimization for Embedded Systems. Journal of Systems Architecture: Embedded Software Design (JSA), Oct. 2011.

[19] Knuth D.E. The Art of Computer Programming. Volume 3: Sorting and Searching. 2nd edn. Addison-Wesley. 1998.

[20] H. S. Stone. Parallel Processing with the Perfect Shuffle, IEEE Transactions on Computers, Vol. 20, pp. 153–161, 1971.

[21] G. M. Masson, G. C. Gingher and S. Nakamura. A Sampler of Circuit Switching Networks. EEE Computer, Vol. 12, No. 6, pp. 32–47, June 1979.

[22] D. Nassimi and S. Sahni. A Self-Routing Benes Network and Parallel Permutation Algorithms. IEEE Transactions on Computers, Vol. 30, pp. 332–340, 1981.
[23] K. E. Batcher. Sorting Networks and Their Applications. Proc. AFIPS Spring Joint Computer Conf. 32, pp. 307–314, 1968.
[24] C. D. Thompson and H. T. Kung. Sorting on a Mesh-Connected Parallel Computer. Comm. ACM, Vol. 20, pp. 263–271, 1977.
[25] Rene Mueller, Jens Teubner, Gustavo Alonso. Sorting Networks on FPGAs. The VLDB Journal, Vol. 21, No. 1, p. 1–23, February 2012.
[26] Y.Jun, Li Na, D. Jun, Guo Y., Tang Z. A research of high-speed Batcher's odd-even merging network. E-Health Networking, Digital Ecosystems and Technologies (EDT), Vol. 1, pp. 77–80, April 2010.
[27] Jean-Philippe Thiran, Herve Bourlard, Ferran Marques, Multi-Modal signal processing: methods and techniques to build multimodal interactive systems. Academic Press Inc. 23 November 2009. – 448 p.
[28] F. Camastra and A. Vinciarelli. Machine Learning for Audio, Image and Video Analysis: Theory and Applications. Springer, 2008.
[29] Handbook of Signal Processing Systems. Editors: Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, Jarmo Takala. – Springer, 2010. – 1117 p.
[30] Melnyk A., Melnyk V. "Personal Supercomputers: Architecture, Design, Application". – Lviv Polytechnic Publishing House, 2013. – 516 p.
[31] J.Leverich Comparative Evaluation of Memory Models for Chip Multiprocessors/ J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, C. Kozyrakis // ACM Transactions on Architecture and Code Optimization. – November 2008.

Anatoliy O. Melnyk is a Head of Computer Engineering Department at Lviv Polytechnic National University since 1994. He graduated from Lviv Polytechnic Institute with the Engineer Degree in Computer Engineering in 1978. In 1985 he obtained his Ph.D. in Computer Systems from Moscow Power Engineering Institute. In 1992 he received his D.Sc. degree from the Institute of Modeling Problems in Power Engineering of the National Academy of Science of Ukraine. He was recognized for his outstanding contributions to high-performance computer systems design as a Fellow Scientific Researcher in 1988. He became a Professor of Computer Engineering in 1996. Since 1982 to 1994 he has been a Head of Department of Signal Processing Systems at Lviv Radio Engineering Research Institute. Since 1994 to 2008 he has been Scientific Director of the Institute of Measurement and Computer Technique at Lviv Polytechnic National University. Since 1999 to 2009 he has been Dean of the Department of Computer and Information Technologies at the Institute of Business and Perspective Technologies, Lviv, Ukraine. He has served since 2000 as President and CEO of Intron ltd. He has also been a visiting professor at Kielce University of Technology, University of Information Technology and Management, Rzeszow, University of Bielsko-Biala. Currently he is a visiting professor at the Department of Artificial Intelligence of John Paul II Catholic University of Lublin.