HARDWARE MEMORY BUFFER MODULE FOR MULTIPROCESSOR SYSTEM

[a]ALEXEY I. MARTYSHKIN, [b]IGOR I. SALNIKOV

[a,b]Penza State Technological University, Russia, Baidukova passage / Gagarina street, 1a/11, Penza, Penza Region, 440039, Russia
e-Mail: [a]alexey314@yandex.ru, [b] info@ores.su

Abstract: The article deals with the problem of so-called bottlenecks in high-performance multiprocessor systems, namely the conflicts for the access to a common system bus shared by all processors. They showed and described the possibility of implementation between the processors and the memory of the hardware-implemented memory buffer module necessary for quick access to memory (the associative memory on fast registers is used in a buffer device) of the multiprocessor system with a widely used "common bus" interface. The buffer is implemented on the register memory and consists of two parts, one of which is responsible for data record, the other one is used for reading. In the course of the research, the functional organization of the module has been determined, the algorithms for its operation have been developed and implemented, the VHDL file describing the operation of the device has been created and debugged, and simulation was performed in ISE Web Pack software. Due to the capabilities of the applied modern element base (programmable logic integrated circuits (PLIC)), the buffer device described in the article is reconfigurable and cross-platform. Due to the application of the described module, it is possible to solve partially the problem of the multiprocessor system "bottleneck" with the "common bus" interface. After the practical use of the described device, the throughput of the subsystem "processor-memory" and, accordingly, the performance of the entire multiprocessor system as a whole, will be increased.

Keywords: multiprocessor system, hardware memory buffer, structural organization, functional organization, operation algorithm, transaction splitting mode, PLIC, VHDL.

1 Introduction

The whole life of a modern man is literally imbued with computer technology: computers and the computer systems (CS) created on their basis. They penetrated everywhere: in household appliances, devices, communication devices, etc. The list can be continued for a long time. Among all the CS multiprocessor systems (MPS) stand apart, which are actively used for laborious calculations, for example, to model complex processes and other

scientific calculations, which require a huge performance and well-coordinated work of the subsystem "processor-memory."

The device developed and described in the article is intended for fast access to memory and processor load reduction. The result of this module use is a significant reduction of memory loading, the bandwidth of the "processor-memory" subsystem is increased and the speed of the MPS in general is also increased.

2 Problem formulation

The presented article is of a research nature in general. A number of literature sources (Biktashev & Knyazkov, 2004; Haemacher et al, 2003; Tsilker & Orlov, 2011) was analyzed to find unaffected issues and unresolved problems during the subject area study. A number of problematic issues related to the possibility of the memory buffer hardware implementation for multiprocessor systems to unload the processor-memory subsystem has not been adequately reflected in existing publications, the problematic issues were analyzed partially in (Martyshkin, 2014; Martyshkin, 2015; Timofeeva et al, 2017).

The purpose of the paper is to describe the consideration of possible algorithms for buffer memory device (BMD) operation of MPS memory with a common bus (CB) interface, which includes 4 processors (Figure 1). This issue is topical today due to global informatization and almost universal operation of huge amounts of data. In order achieve this goal, the article solves the problems of the device structure determination and the principles and the algorithms of its functioning. In existing MPS, several devices may apply for CB loading simultaneously, however, only one of them is possible to do it at any moment. In order to avoid possible conflicts, the CB must choose the mechanisms for request arbitration and the rules for a bus granting to a particular device among all those which requested it (Suvorova & Sheinin, 2003; Villalobos Antúnez, 2001).

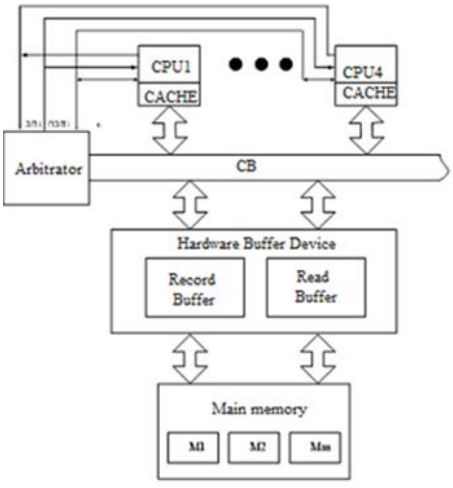

Fig 1. Block diagram of a four-processor system with a hardware memory buffer

The CB is presented in accordance with AMBA (Advanced Microcontroller Bus Architecture) specification (13), developed as a communication standard for high-performance systems-on-chip (Martyshkin & Yasarevskaya, 2015; Martyshkin & 2016; Salnikov et al, 2016).

The AMBA standard, the bus protocol and organization are in a good agreement with the design of synthesizable, parametrizable modules and the systems-on-chip based on them. The AMBA bus standard includes three bus specifications:

- AHB - Advanced High-performance Bus.
- ASB - Advanced System Bus.
- APB - Advanced Peripheral Bus.

Currently, AMBA-based communication systems are widely used in aerospace systems-on-a-chip. For example, AMBA buses are used to organize the system of communications in systems-on-chip on LEON processor core, organized in accordance with the SPARC V8 architecture .The AMVA AHB bus is also used in the developed domestic systems-on-the-chip, for example, within the framework of the "Multicore" project .

The AMBA standard is designed to develop high-performance systems. In accordance with this standard the data exchange is carried out in synchronous mode. The standard provides the

support for packet transfers and split transactions. The system must have no more than 16 master devices, the number of slaves is unlimited. The organization of communications by bus is carried out under the management of an arbitrator.

In order to implement the HBD, the AMBA AHB bus is used, acting as an intermediary between the processor and the memory. When the operation (transaction) of memory record or reading is performed continuously, the CB handles one of the system processors exclusively until the operation is completed. Thus, the bus and the processor are in the standby mode until the memory performs a physical reading or writing procedure. Thus, bus cycles are lost that could be used by other processors. In order to reduce the time losses and increase the bandwidth of the CB, it is necessary that it supports the modes of reading transaction splitting and record transaction buffering.

The memory read operation is subjected to splitting, and it is divided into an address transaction and a data transaction. When memory is required the processor sets an address to a CB, which is stored in HBD, after which the CB is released and the processor goes into the standby mode. The physical reading procedure takes place in the memory itself under HBD control, which at the end of the physical read procedure should signal the requesting processor about data readiness. In response, the processor requests CB again and reads the data word from HBD.

The buffering of record transactions is that the processor puts the address of the memory cell and the data to be written to the bus. They are stored in HBD registers, after which the processor releases the CB, since there is no reverse memory reaction in this case. The procedure of physical record in memory is performed under HBD control.

It follows from the stated above that a developed unit should be equipped with two buffer devices to store read and record transactions (Figure 2). In its turn, the read buffer has two parts. The first contains the registers to store the memory cell address into which a request is made, the second contains the registers to store the data selected from the memory data. The record buffer also consists of two parts. The first stores the memory cell addresses, which is required, the second stores the recorded data.

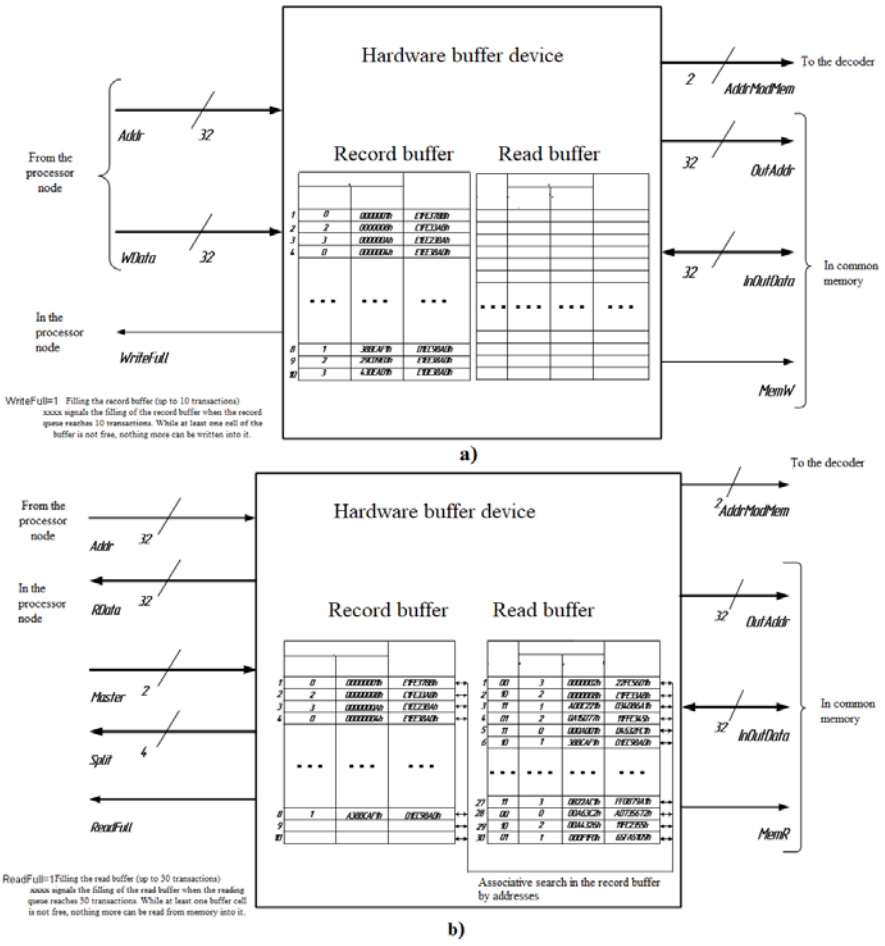

Fig 2. Block diagram of the hardware memory buffer in the record (a) and reading (b) mode

The work uses MPC architecture with the Unified Memory Access (UMA). In order to increase the memory bandwidth, it is divided into a number of independent modules, each of which has its own addressing and data buffering schemes. If a bus with a transaction splitting is used, it is possible to access the memory of several processors simultaneously. The access time to the data from memory does not depend on a processor accessing the memory, and on a memory chip containing the necessary data. At that each processor unit can use its own cache.

The principle of the device operation is the following one. Suppose that one or more processors generated a record transaction simultaneously. In order to implement it successfully, you need to access the CB, for which the processors send the request signals to the arbitrator, which in its turn checks whether the CB is currently available and, according to some rule, selects one of the processors to perform the operation. If the CB is free, then the processor captures it. Further, a check is made for record buffer filling, and if it is full, the processor is put into the standby mode. If there is at least one free register in the record buffer, the processor places the data word there. The further work of the processors does not depend on the record result, i.e. it makes no sense to wait until the end of the record, so it releases the bus.

A lot of requests can be accumulated in the record buffer, and it

is possible that the read request will refer to data already in HBD, and not in memory, so they can be read directly from HBD, not from the memory, which is much faster, than a request to memory. The addressable record buffer is performed in the form of associative memory for a fast implementation of this function (Martyshkin, 2017).

The reading procedure with transaction splitting allows a simultaneous execution of several transactions generated by different processors. At the beginning of the read operation, the requesting processor occupies the bus, places the address and read signal on it, which are fixed in the reading buffer. This transaction is executed quickly because buffers are implemented on hardware registers. After this procedure, the processor disconnects from the bus. The buffer device carries out the process of data physical reading itself from the memory module and the process of result storing in one of the read buffer registers. At an appropriate moment, when the bus is idle, the data is returned to the processor.

In the systems with shared memory, all processors have equal capabilities for a single address space access. A single memory can be built as single-block one or by modular principle, but usually there is a second option in practice. In order to improve performance, it makes sense to apply a memory split to addresses into 4 modules.

A possible structure and the principles of HBD functioning are shown in Here we will dwell in detail on the functional organization and the algorithms of HBD operation.

The circuit implementation does not work in VHDL code if you do not adhere to the description of a specific element for VHDL (for example, the description of a register, a counter, a decoder operation, etc.). But still, let's highlight some functional units (Figure 3):

- the unit of a transaction adding to the read queue;
- the unit of a transaction adding to the record queue;
- the unit of a pointer increase by the head of the read buffer and read queue increase;
- the unit of a pointer increase by the head of the record buffer and record queue increase;
- the unit for data search in the record buffer when the addresses coincide with the read buffer;
- the reading unit from memory according to the specified address;
- the unit of pointer increase to the tail of processed message queue;
- the unit of record to memory according to the specified address;
- the unit of pointer increase by record buffer tail;
- the unit of data output to the processor initiating the read request.

On Figure 3, all units are shown in a general view.

The unit of a transaction adding to the read queue works as follows. The internal register RGA1 receives a 32-bit address from the processor, by which data should be found. The RGMID register is supplied with a 2-bit processor identifier (takes the value from 0 to 3 according to the number of processors in a system). The enabling signal for the operation of these registers is the following combination of signals $\overline{TypeTrans \& W \& \overline{R} \& Sel}$.

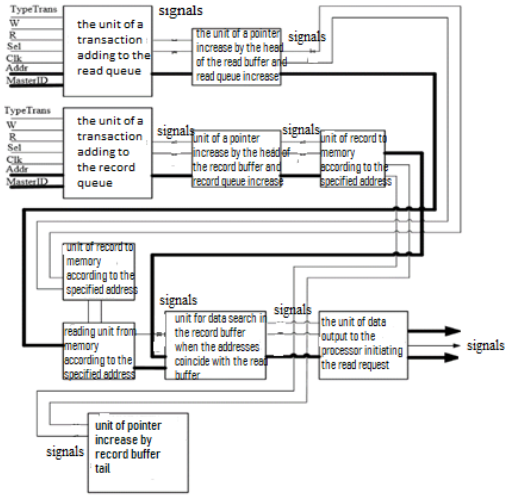

Fig 3. Functional units of a hardware memory buffer

The unit of a transaction adding to the record queue works as follows. The internal register RGA2 receives a 32-bit address from the processor, on which data should be written. The RGD register is supplied with 32-bit data from the processor. The enabling signal for the operation of these registers is the combination of signals $TypeTrans \& W \& \overline{R} \& Sel$.

- The unit of the pointer increase by the head of the read buffer and read queue increase operates as follows. The entire queue of requests is represented as consisting of the "queue head", the "queue tail" and the cell pointer at this moment. When you add the following request to the read queue, the "head" is incremented by one, and at the same time, the read queue increases. This unit can be represented in the form of counters.

- The unit of data search in the record buffer when addresses coincide with the read buffer can be implemented on a 32-bit comparator. One input of which is supplied by the address set by the processor in the read buffer, the other is supplied with addresses from the record buffer sequentially. When the addresses match, the data from the record buffer are returned to the processor. If the addresses do not match, i.e. the associative search has not provided results, it is necessary to access the memory according to the correct address. This action is produced by the following unit.

The unit of reading from memory according to the specified address works as follows. From the 32-bit register in which the address is stored this address is sent to HBD output, to address memory inputs. Two most significant bits of the address are used to select the memory module from which you need to read the data according to the specified address. A read signal is sent from MemR memory, which will be held in the unit for 50 ns - the time of data search in memory according to the specified address and the reading into the buffer. The data is stored in the data register of the read buffer, and when the processor that set the address for reading is reconnected to the CB to receive the

data, it will read the necessary data from the read buffer. A Split signal (processor number) is provided.

The unit of the pointer to the queue tail of processed messages works as follows. When a read operation from memory occurred, the data is not provided to the processor immediately. First, a so-called "queue of processed messages" is created. The message queue increases with each new processed transaction. The unit can be implemented on the counter.

The record unit into memory according to the specified address operates as follows. Among 32-bit registers in which the address and the data are stored in the record buffer, the record to memory is performed. Here, two most significant bits of the address are used to select a memory module where the data will be recorded. A record signal is sent to the MemW memory, which will be held for 10 ns - the time of data record into memory.

The unit of the pointer increase by the tail of the record buffer works as follows. During the record to memory, the pointer to the buffer tail is incremented by one with each processed transaction. After that, one cell is freed in the record buffer.

The data issue unit to the processor initiating the read request operates as follows. As soon as the queue of ready-made orders is formed, i.e. all requests of the queue are processed, the data from the memory is placed in the read buffer, then the processors can pick up the applications intended for them. The data from the registers is fed to the DataRead output and the processor reads the data.

Now we will show the variant of HBD algorithm functioning. Let's imagine the subsystem "processor-memory" as two subsystems: "processor-HBD" and "HBD-memory". Let's describe the algorithm of the subsystem "processor-HBD" operation. First, the processor checks the blocking line and determines whether the CB is free or busy at this moment. Suppose a high potential on the block line corresponds to a state in which the CB is free. If the processors interrogating the blocking line detect a high potential there, they send requests to the bus arbitrator. The processor with the highest priority will receive a signal confirming the request. After the processor captures the CB, an operation type is selected: read or record. In the case of a record operation, the record buffer is checked for free space and, if it is full, the processor is put into the standby mode until a free cell appears. If there is some place, an address and data are recorded, after which the processor releases the CB. If you select a read operation, the read buffer is first checked for free space and if it is absent, the processor is put into standby mode until a free cell appears. If there is an available cell, an address is recorded, according to which the data must be provided for reading. After the procedure of physical reading from memory, or the search in the associative memory of the record buffer the data is read into the read buffer according to a set address. The HBD notifies the processor, which has prepared the read data, about the readiness and it takes them from the HBD.

Let's describe the possible algorithm of the subsystem "HBD-memory". First, you select an operation type: read or write. When the record operation is selected, the request for the j-th memory module is made, where the data will be recorded sent to HBD from the processor. Next, the j-th memory module is checked for loading and, if it is full, the subsystem goes into the standby mode, and if it is free, the data is recorded to a desired address. Then the memory module, like one cell of the write buffer, is released: СчБЗ: = СчБЗ-1, where СчБЗ is the record buffer (semaphore) (in this paper it is assumed that СчБЗ=10, that is, the capacity of the record buffer makes 10 cells). When you perform a read operation, the k-th memory module is requested, from where the data will be read into HBD, for their further transmission to a corresponding processor. After that, the k-th memory module is checked for loading and, if it is full, the subsystem goes into the standby mode, if it is free, the data is read according the desired address: СчБЧт: = СчБЧт+1, where СчБЧт is the read buffer counter (semaphore). It is accepted in the article that СчБЧт = 30, i.e. the capacity of the read buffer makes 30 cells). After that, the read data is written to the cell of the read buffer. Then the memory module is released, and the HBD notifies the requested processor about the operation.

3 Experiment results

Based on these algorithms, the VHDL file describing HBD operation was created in ISE WebPack program, and the element was synthesized, and its debugging and modeling was performed, and operation time-series diagrams were obtained. The results of HBD operation modeling are shown on Figure 4. According to the obtained time diagrams, we can judge the correctness of a device according to the developed algorithms, by which it is possible to speak about the correct functioning of the device according to the algorithms described above.

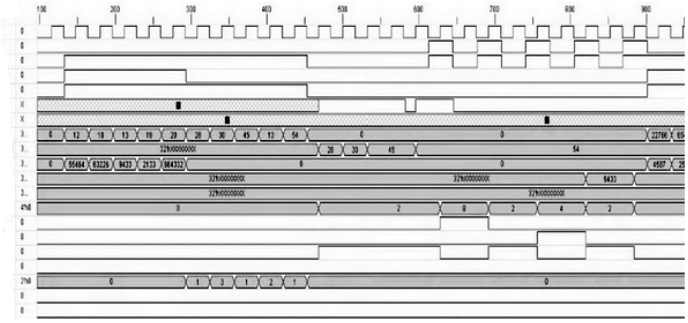

Fig 4. Time diagrams for HBD memory operation

The diagram has the following signals: clk - clock signal, R - reading signal, TypeTrans - operation selection signal (1 - record, 0 - read), W - record signal, MemR - memory read signal, MemW - memory record signal, Addr (31: 0) - 32-bit address, OutAddr (31: 0) - the address at the device output, DataRead (31: 0) - 32-bit data for recording, DataWrite (31: 0) - 32-bit data for reading, OutData (31: 0) - 32-bit record data to memory (reading from memory), Split (3: 0) - the signal "1" indicates the readiness of buffering device to the split transaction completion from a corresponding processor, MID - processor node identifier, WriteFull - recording buffer overflow signal, ReadFull - read buffer overflow signal, queuesizeread - the size of the read queue, headread - the «head» of the read queue, tailread - the «tail» of the read queue, queuesizewrite - the size of record queue, headwrite is the "head" of the write queue, tailwrite is the "tail" of the write queue, buftail is the "tail" of the buffer queue, tailcpl is the "tail" of the processed read request queue. Suppose, the signal "1" appears on the lines Sel, TypeTrans, W at the moment of 127 ns. It holds for 5 cycles on these lines, i.e. there are 5 requests according to the

corresponding addresses in the decimal system (12, 10, 13, 19, 20). Data is also set on the data line for the record to memory according to the specified addresses. At this time, the length of the record queue and the pointer to the "head" of the record buffer increases. Then the TypeTrans signal takes the value "0", which corresponds to the read operation. Now the signal "Sel" and "W" are only in "1". The reading queue is developed for 5 cycles. Addresses are set on the address line, and processor node identifiers are set on the MID line. Then the signals Sel, TypeTrans, W take the values of "0". The processing of the transaction queue begins. The priority for the reading process is chosen as the highest one. First, the read queue is processed, followed by the record queue. If the address in the read buffer matches the address in the record buffer, the data is taken directly from the record buffer without accessing the memory. This process is monitored on the time diagram. Address 13 is present both the record buffer and the read buffer. If you trace along the diagram lines, you can notice that the data is immediately selected from the record buffer at this address. If the addresses do not match, the buffer device accesses the memory. At the time of access to the memory, the signal "1" is set on the MemR line. When the queue of processed messages is generated, the buffer device signals the requesting processor node that it is ready by setting the Split signal (processor number) to "1", at this point the processor takes its data. After the processing the read queue, the write queue is processed. Then the actions are repeated upon the receipt of requests for reading or writing.

4 Conclusions

The paper touches upon the issues of the hardware buffer device functional organization and its work algorithms. The device in question differs from earlier ones by the following: previously the task in MPS was solved using the memory with NUMA or UMA architecture with alternating addresses, which made it impossible to use the mode of transaction splitting on a CB.

The result of this development use, implemented on the modern element base - PLIC, is memory loading reduction, the bandwidth of the subsystem "processor-memory" and the performance of the entire MPS are increased.

Literature:

1. Biktashev R.A., Knyazkov V.S. (2004). Multiprocessor systems. Architecture, topology, performance analysis: Textbook. - Penza: Publishing house of Penza State University, p.107.
2. Haemacher K., Vraneshic Z., Zaky S. (2003). Organization of COMPUTER. 5th edition. Translated from English by O. Zdir. - St. Petersburg: Peter; Kiev: Publishing Group BHV, p.848.
3. Tsilker B.Ya., Orlov S.A. (2011). Organization of computers and systems: Textbook for high schools. - 2nd edition. - St. Petersburg: Peter, p.688.
4. Martyshkin A.I. (2014). Mathematical modeling of the multiprocessor system memory buffer. Collection of materials of the XIIth International Scientific and Technical Conference Optoelectronic Devices and Devices in Image Recognition, Image Processing and Symbolic Information Systems. Recognition-2015. Kursk: South-Western state university, pp. 247-249.
5. Martyshkin A.I. (2015). Implementation of the hardware buffer for multiprocessor system memory // Proceedings of the XIIth International Scientific and Technical Conference "New Information Technologies and Systems." Penza: PSU, pp. 96-99.
6. Martyshkin A.I. (2015). The development of a hardware buffer for a multiprocessor system memory. Fundamental research, 1(3), pp. 485-489.
7. Martyshkin A.I., Yasarevskaya O.N. (2015). Mathematical modeling of the Task Managers for Multiprocessor systems on the basis of open-loop queuing networks. ARPN Journal of Engineering and Applied Sciences, 10(16), pp. 6744-6749.
8. Martyshkin. A.I. (2016). Functional organization and operation algorithms of a hardware buffer for a multiprocessor computer system memory. Fundamental Research, 12(3), pp. 518-522.
9. Salnikov I.I., Babich M.Yu., Butaev M.M., Martyshkin A.I. (2016). Investigation of the memory subsystem of information systems.The International Journal of Applied Engineering Research, 11(19), pp. 9846-9849.
10. Martyshkin A.I. (2016). Development and research of open loop models the subsystem processor-memory of Multiprocessor systems architectures UMA, NUMA and SUMA. ARPN Journal of Engineering and Applied Sciences, 11(23), pp. 13526-13535.
11. Suvorova E.A., Sheinin Yu.E. (2003). Design of digital systems on VHDL. St. Petersburg: BHV-Petersburg, p.576.
12. Martyshkin A.I. (2017). Mathematical modeling and the possibility of algorithm hardware support to control the interacting processes in high-performance computing systems XXIst century: results of the past and the problems of the present, 4 (38), pp. 132-139.
13. Timofeeva L.S., Kadyrova M.I., Akhmetova A.R. (2017). The historic city as an object of cultural tourism (on the example of Yelabuga), Astra Salvensis, Supplement No. 2, p. 177-183.
14. Villalobos Antúnez J.V. (2001). La ética y el derecho ante la filosofía intercultural y la globalización, Unica: Revista de Artes y Humanidades, No. 4, pp. 71-76.