VOL. 13, NO. 23, DECEMBER 2018 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

HARDWARE BUFFER MEMORY OF THE MULTIPROCESSOR SYSTEM

Martyshkin A. I.
Baydukov Proyezd / Gagarin Street, 1a/11, Penza, Penza region, Penza State Technological University, Russia
E-Mail: alexey314@yandex.ru

ABSTRACT

The article is devoted to solving issues related to the problem of the "bottlenecks" in multiprocessor computing systems, namely, conflicts for access of processors to the shared system bus. It is described the possibility of placing between the processor and the memory of the hardware-implemented module of the buffer device, which is necessary for quick access to memory (in the structure of the buffer module uses associative memory) of a multiprocessor computer system with a widely used "common bus" interface. The buffer is implemented in register memory and consists of two parts, one of which is responsible for writing data, the other for reading data. In the course of the operation, the functional organization of the hardware buffer unit was defined, the algorithms for its operation were developed and implemented, a VHDL file describing the operation of the device was created and debugged, simulation of the correctness of work in the ISE Web Pack program. Using modern element base, namely, field-programmable gate array (FPGAs), the described buffer device is reconfigurable (you can adjust the VHDL file to change the parameters of the work, and the structure and functionality at any time) and cross-platform, because of universality of VHDL-code the device can be implemented on FPGAs of different manufacturers. Thanks to the application of the described block, it is partially possible to solve the problem of the "bottleneck" of the multiprocessor system with the "common bus" interface. As the result of 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 increase.

Keywords: multiprocessor system, hardware buffer memory, structural organization, functional organization, algorithm, transaction splitting mode, distributed memory, associative memory, write buffer, read buffer, write mode, read mode.

1. INTRODUCTION

The whole life of modern people is literally saturated with computer technology: computers and computer systems (CS) created on their basis. It penetrated everywhere: in household appliances, appliances, communication devices, etc. The list can be continued for a long time. Among all the CS stand apart multiprocessor computing systems (MCS), which are actively used for time-consuming calculations, for example, in modeling complex processes and other scientific calculations that require colossal performance and well-coordinated work of the subsystem "processor-memory".

The main purpose of the article is to develop a hardware memory buffer device (HMBD) for memory of the MCS with a common bus, consisting of 4 processors (CPU). The developed device is designed for fast access to memory and unloading of CPUs. As a result of using this module, memory loading is significantly reduced, the bandwidth of the processor-memory subsystem is increased, and the speed of the MCS as a whole is increasing.

2. GOAL SETTING

Represented article as a whole is exploratory in nature. Literature sources [1-3] were analyzed in the course of studying the subject area in order to find unaffected issues and unresolved problems. Some problematic issues related to the possibility of hardware implementation of the memory buffer for multiprocessor systems for offloading the processor-memory subsystem have not been adequately reflected in existing publications, some problematic issues were analyzed in [4-7] and [8-11].

The purpose of the paper is to describe the possible algorithms for the operation of a HMBD of the memory of the MCS with a common bus (CB) interface including 4 CPUs (Figure-1). This problem is relevant today due to global informatization and the widespread operation of colossal amounts of data. To achieve this goal, the article solves the problems of determining the structure of the device and the principles and algorithms of its functioning. In existing MCS, several devices may claim for the employment of the CB at any one time, however, only one of them is possible at any one time. In order to avoid possible conflicts, the CB must select mechanisms for arbitrating requests and the rules for granting a common bus to one particular device from the requesters [12].

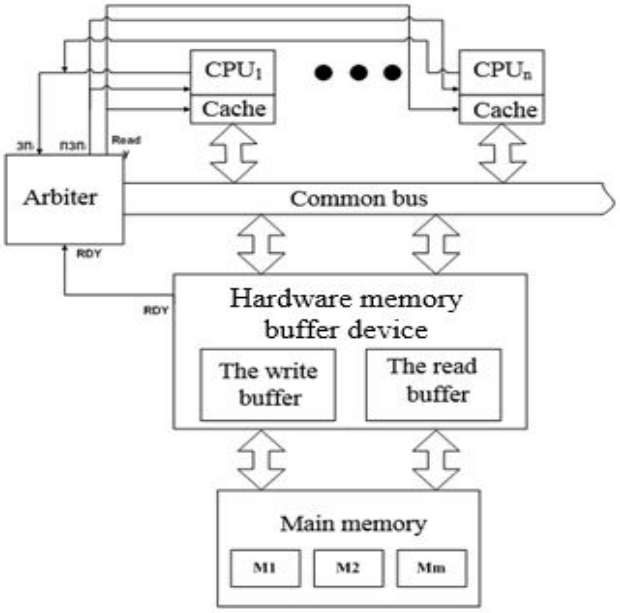

Figure-1. Block diagram of a multiprocessor system with a hardware memory buffer device.

The CB is presented in accordance with the AMBA specification (Advanced Microcontroller Bus Architecture) [13], developed as a communication standard for high-performance systems-on-chip.

The AMBA standard, protocol and bus organization are in good agreement with the design of synthesizable, parameterizable modules and systems-on-chip based on them. The AMBA bus standard includes three bus specifications [13]:

- 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 system-on-chip on the LEON processor core, organized in accordance with the SPARC V8 architecture [12]. The AMBA bus AHB is also used in the developed native systems-on-a-chip, for example in the "Multicore" project [12].

AMBA standard is designed for building high-performance systems. Data exchange in accordance with this standard is performed in the synchronous mode. The standard provides support for packet transfers and split transactions. The system must have no more than 16 leading devices; the number of slaves is unlimited. The organization of communications on the bus interface is carried out under the management of an arbitrator.

To implement the HMBD, the AMBA AHB bus [13] was used, acting as a mediator between the CPUs and the memory. When continuously performing a write transaction or read transaction of the memory the CB monopoly owns one of the CPUs of the system until the operation is completed. Thus, the bus and the CPU are in standby mode until the memory performs a physical reading or writing procedure. As a result, the bus cycles that could be used by other CPUs are lost. To reduce the loss of time and increase the capacity of CB requires that it supports splitting modes of reading transactions and buffering the writing transactions.

A memory read operation is subjected to the splitting, and it is divided into an address transaction and a data transaction. When accessing the memory, the CPU sets the address to the CB, which is stored in the HMBD, after which the CB is released and the CPU goes into the standby mode. The physical reading procedure takes place in the memory itself under the control of the HMBD, which, at the end of the physical reading procedure, should signalized the requesting CPU about the readiness of the data. In response, the CPU again requests the CB and reads the data word from the HMBD [6, 8].

Buffering of the write transaction is that the CPU puts to the bus the address of the memory cell and the data that will be written. This data is stored in registers of the HMBD, after which the CPU releases the CB, since there is no reverse memory reaction in this case. The procedure of physical recording in memory is performed under the control of the HMBD [6, 8].

From the above, it follows that the developed unit should be equipped with two buffer devices for storing read and write transactions (Figure-2). Besides the read buffer has two parts. The first part contains registers for storing the address of the memory cell into which the reference is made, the second part is the registers for storing the data selected from the memory. The write buffer also consists of two parts. The first part stores the addresses of the memory cell, which is accessed, the second stores the data being written.

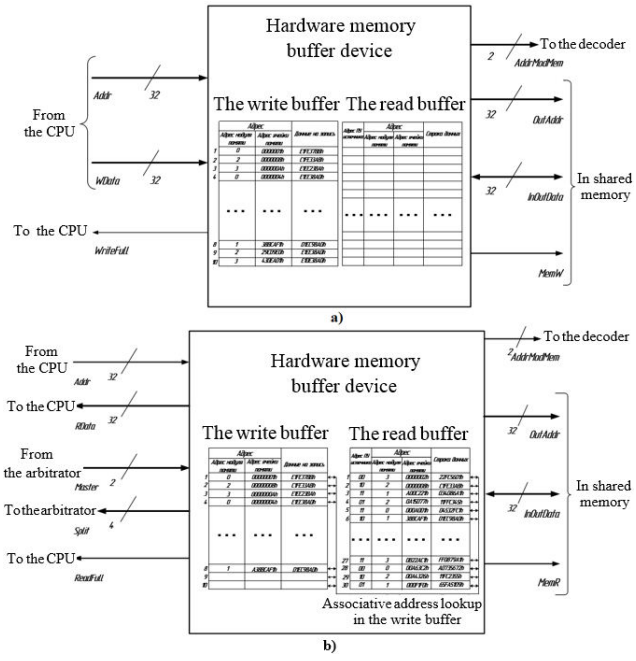

Figure-2. Block diagram of a hardware memory buffer device in the writing (a) and reading mode (b).

The architecture of the Unified Memory Access (UMA) is used in the article. To increase the memory bandwidth, it is divided into several 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 CPUs simultaneously. The time of access to the data from memory does not depend on which CPU is accessing the memory, or on what memory chip contains the necessary data. In this case, each CPU can use its own cache.

The principle of operation of the device is as follows. Suppose that one or more CPUs simultaneously generated a write transaction. To successfully implement it, CPU needs to access the CB, for which the CPU sends the request signals to the arbitrator, which checks whether the CB is currently available and selects one of the CPUs for the operation according to some rule. If the CB is free, then CPU grabs it. Further, a check is made to fill the write buffer, and if it is full, the CPU is put into the standby mode. If there is at least one free register in the write buffer, the CPU puts the data word there. The further operation of the CPU does not depend on the result of the write, i.e. it makes no sense to wait until the end of the write, so it releases the CB.

Many requests can accumulate in the write buffer, and it is possible that a read request will refer to data already in the

HMBD, and not in memory, so it can be read directly from the HMBD, not from shared memory, which is faster, than access to shared memory. For fast implementation of this function, the addressable write buffer is performed in the form of associative memory.

The procedure for reading with the splitting of transactions allows the simultaneous execution of several transactions generated by different CPUs. At the beginning of the read operation, the requesting CPU occupies the CB, places the address and read signal on it, which are fixed in the read buffer. This transaction is performed quickly because buffers are implemented on hardware registers. After this procedure, the CPU is disconnected from the CB. The buffer device itself carries out the process of physically reading data from the shared memory module and storing the result in one of the read buffer registers. At the appropriate time, when the CB is idle, the data is returned to the CPU.

In systems with shared memory, all CPUs have equal capabilities for accessing a single address space. A single memory can be built as single-block or modularly, but usually in practice there is a second variant [3]. In order to improve performance, it makes sense to apply a shared memory bundle to addresses of 4 modules.

The possible structure and principles of operating of the HMBD functioning are shown in [4, 5] and [6, 8]. Here we will dwell in detail on the functional organization and algorithms of the HMBD operation.

When we make a VHDL code, the circuit implementation does not work if we do not adhere to the description of a specific element for VHDL (for example, the description of the operation of the register, counter, decoder, etc.). Let's highlight some functional modules (Figure-3):

Module of adding transaction to the read queue;

Module of adding transaction to the write queue;

Module of increasing the pointer to the head of the read buffer and increasing the read queue;

Module of increasing the pointer to the head of the write buffer and increasing the write queue;

Module of searching data in the write buffer when addresses match the read buffer;

Module of memory reader to the specified address;

Module of increasing the pointer to the tail of the queue of processed messages;

Module of memory write to the specified address;

Module of increasing the pointer to the tail of the write buffer;

Module of data issuing to CPU that initiated the read request.

In Figure-3, all the blocks are shown in a general view.

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

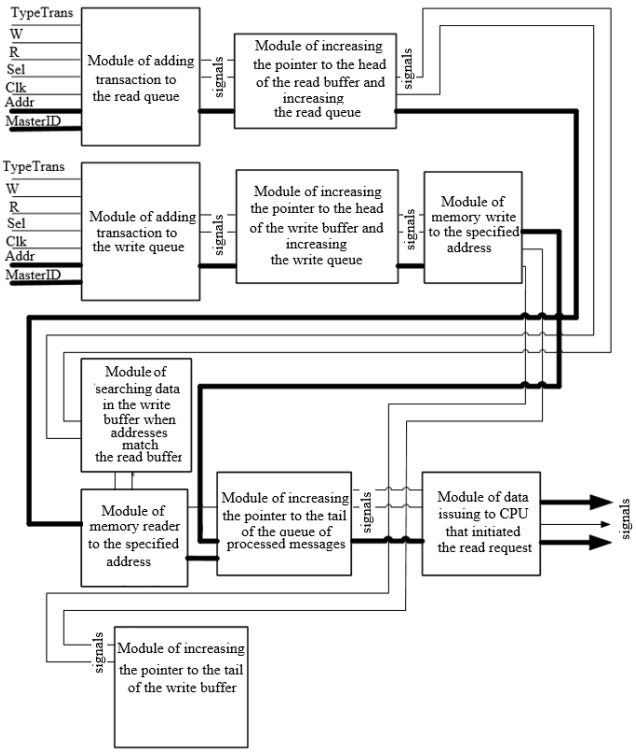

Figure-3. Functional units of a hardware memory buffer device.

Module of adding transaction to the write queue works as follows. The internal register RGA2 receives a 32-bit address from the CPU, on which data should be written. The RGD register is supplied with 32-bit data

from the CPU. The enabling signal for the operation of these registers is a combination of signals $TypeTrans \& W \& \overline{R} \& Sel$.

Module of increasing the pointer to the head of the read buffer and increasing the read queue operates as follows. The entire queue of applications is represented as consisting of the "queue head", "queue tail" and the cell pointer at a given time. When we add the following application to the read queue, the "head" is incremented by one, and at the same time, the read queue itself increases. This module can be represented form the counters.

Module of searching data in the write buffer when addresses match the read buffer can be implemented on a 32-bit comparator. One input of which is supplied by the address set by the CPU in the read buffer, another address is sequentially supplied from the write buffer. When the CPU addresses match, the data from the write buffer is returned. If the addresses do not match, i.e. associative search has not yielded results, it is necessary to refer to memory at the right address. This action produces the following block.

Module of memory reader to the specified address works as follows. From the 32-bit register in which the address set by the CPU is stored, this address is sent to the output of the HMBD, to address memory inputs. The two most significant bits of the address are used to select the memory module, from where we need to read the data to the specified address. A read signal (MemR) is sent from memory, which will be kept in the unit for 50 ns - the time of data search in memory at the specified address and reading into the buffer. The data is stored in the data register in the read buffer, and when the CPU that has set the read address is reconnected to the CB to receive the data, it will read the necessary data from the read buffer to it. A Split signal (CPU number) is output.

Module of increasing the pointer to the tail of the queue of processed messages works as follows. When there was a read operation from memory, the data is not immediately issued by the CPU. First, a so-called "queue of processed messages" is created. With each new processed transaction, the message queue increases. The block can be implemented on the counter.

Module of memory write to the specified address operates as follows. At the 32-bit registers in which the address and data are stored in the write buffer, a write to memory occurs. Here, also the two most significant bits of the address are used to select the memory module where the data will be written. A write signal (MemW) is sent to memory, which will be kept in the unit for 10 ns – the time of writing data to memory.

Module of increasing the pointer to the tail of the write buffer works as follows. When writing to memory, the pointer to the tail of the buffer is incremented by one with each processed transaction. After that, one cell is freed in the write buffer.

Module of data issuing to CPU that initiated the read request operates as follows. Once the queue of ready-made orders is formed, i.e. all requests for the queue are processed, data from the memory is placed in the read buffer, then CPUs can take orders that are intended for them. Data from the registers is fed to the Data Read output and the CPU reads the data.

Now we will demonstrate the variant of the algorithm of functioning of the HMBD. For example, the subsystem "processor-memory" consists of two subsystems: "processor-HMBD" and "HMBD-memory". We describe the algorithm of the subsystem "processor-HMBD". First, the CPU checks the lock line and determines whether the CB is free or busy at present moment. Suppose that a high potential on the lock line corresponds to a state where the CB is free. If the interrogators of the CPUs lock line detect high potential there, CPUs send requests to the bus arbitrator. The CPU with the highest priority will receive a signal confirming the request. After the CPU captures the CB, the type of operation is selected: read or write. In the case of a write operation, the write buffer is checked for free space and, if it is full, the CPU putting into the standby mode, where it is before the free cell appears. In case there is a place, the address and data are recorded, after which the CPU releases the CB. If a read operation was selected, the read buffer is first checked for available space and, in the absence of a read operation, the CPU is put into standby mode until a free cell appears. If there is an available cell, an address is write, according to which data must be provided for reading. After the procedure of physical reading from memory, or search in the associative memory of the write buffer at the exposed address, the data is read into the read buffer. The HMBD notifies the CPU, to which the read data is prepared, of the readiness and it takes them from the HMBD.

Let's describe the possible algorithm of the subsystem "HMBD-memory". First, we select the type of operation: read or write. When the writing operation is selected, a request is made for the j-th memory module, where the data sent to the HMBD from the CPU will be written. Next, the j-th memory module is checked for employment, and if it is busy, the subsystem goes into standby mode, if it is free, the data is written to the desired address. Then, the memory module as one cell write buffer exempt: CountWB: = CountWB-1, where CountWB - counter (semaphore) of write buffer (in this article it is assumed that CountWB= 10, i.e., the capacity of the write buffer is 10 cells). When the read operation is performed, the k-th memory module is requested, from where the data will be read into the HMBD, for further transmission to the corresponding CPU. After that, the k-th memory module is checked for employment, and if it is busy, the subsystem goes into the standby mode, if it is free, the data is read to the desired address: CountRB: = CountRB + 1, where CountRB is the semaphore of read buffer. In the article it is accepted that CountRB = 30, i.e. the capacity of the read buffer is 30 cells). After that, the read data is written to the cell of the read buffer. Then the memory module is released, and the HMBD notifies the requested CPU about the operation.

3. THE RESULTS OF THE EXPERIMENTS

Based on the above algorithms, a VHDL file describing the operation of the HMBD was created in the ISE

WebPack environment and the element was synthesized. It's debugging and modeling was performed, and time diagrams were obtained. The results of modeling the operation of the HMBD are shown in Figure 4. According to the obtained time diagrams, we can judge the correctness of the 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.

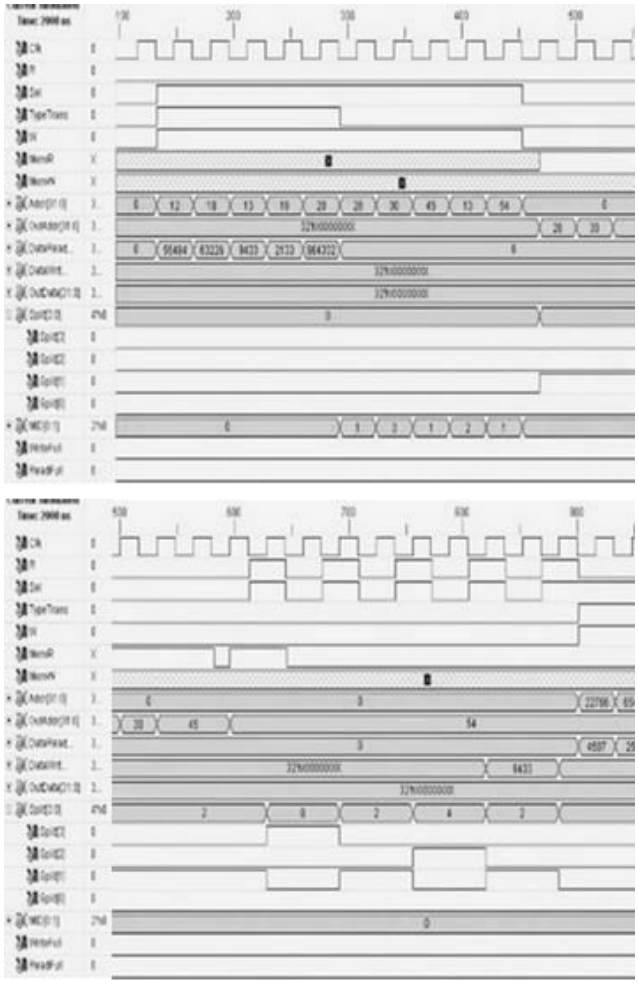

Figure-4. Timing diagrams of a hardware memory buffer device.

4. CONCLUSIONS

The paper discusses the functional organization of the hardware memory buffer device and its algorithms. The device described in the article differs from existing ones by the fact that earlier in the MCS the problem was solved using memory with the NUMA architecture or the memory of the UMA architecture with alternating addresses, which made it impossible to use the transaction-splitting mode on the CB.

As a result of using a hardware memory buffer device implemented on the modern element base - FPGAs, memory loading is reduced, the throughput of the "processor-memory" subsystem is increased, and the performance of the entire MCS as a whole is increased [14].

The work has been done with the financial support of RFBR (Grant No. 16-07-00012 A).

REFERENCES

[1] Biktashev R. A. Knyazkov V. S. 2004. Multiprocessor systems. Architecture, topology, performance analysis: Textbook. Penza: Publishing PSU. p. 107.

[2] Hamaher K., Vraneshich Z., Zaki S. 2003. Computer organization. 5th edition. Translated from English by O. Zdir. SPb. Peter; Kiev. Publishing group BHV. p. 848.

[3] Tsilker B. Y., Orlov S. A. 2011. Organization of computers and systems: Textbook for universities. 2nd edition. SPb. Peter. p. 688.

[4] Martyshkin A.I. 2015. Mathematical modeling of the hardware memory buffer of multiprocessor systems // Collection of materials of the XII International Scientific and Technical Conference "Optoelectronic Devices and Devices in Image Recognition, Image Processing and Symbolic Information Systems. Recognition-2015». Kursk: SWSU. pp. 247-249.

[5] Martyshkin A.I. 2015. Implementation of the hardware memory buffer for Multiprocessor system. // Proceedings of the 12th International Scientific and Technical Conference "New Information Technologies and Systems. Penza: PSU. pp. 96-99.

[6] Martyshkin A.I. 2015. The development of the hardware buffer memory of Multiprocessor system. Fundamental Research. (12-3): 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): 6744-6749.

[8] Martyshkin A.I. 2016. Functional organization and algorithms of operation of a hardware buffer memory of a Multiprocessor computer system // Fundamental Research. (2-3): 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): 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): 13526-13535.

[11] Martyshkin A.I. 2016. Mathematical modeling of Tasks Managers with the strategy in space with a homogeneous and heterogeneous input flow and finite queue. ARPN Journal of Engineering and Applied Sciences. 11(19): 11325-11332.

[12] Suvorova E. A., Sheynin Yu. E. 2003. Design of digital systems in VHDL. SPb. BHV-Petersburg. p. 576.

[13] 1999. AMBA Specification. Rev 2.0. ARM Limited.

[14] Martyshkin A.I. 2016. The studying of distributed Task Managers for Multiprocessor systems based on open queuing networks. Engineering Studies. 8(3-2): 349-356.