International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 13, Number 12 (2020), pp. 4635-4645


# Pilot Model of the Embedded Reconfigurable Real Time Computing System

**Alexey I. Martyshkin**

*Candidate of Engineering Sciences, Associate Professor, Sub-Department of Computers and systems, Penza State Technological University, Russia. 440039, Penza, Baydukova Passage / Gagarin Street, 1a/11,*

**Abstract**

The paper discusses the possibility of simulation modelling the control nodes using the example of a task manager included in a reconfigurable computing system using modern hardware on a modern element base. The purpose of the paper is to conduct experiments to study the control nodes of a reconfigurable computing system for digital signal processing by means of hardware equipment. The object of development and research in this paper is a device consisting of 4 reconfigurable processors implemented on the basis of a FPGA. Reconfigurable computing systems today are promising developments in the field of high performance computing. The paper proposes a block diagram of the device, and a printed circuit board, according to which a prototype of the reconfigurable system was created. To the end, conclusions are drawn from the work. The research of the operation principles of the reconfigurable system was carried out in CAD ModelSim-Altera 10.0c Starter Edition, which allows assessing the correct operation of the device without building physical prototypes. To check the results obtained on the models, a full-scale experiment was carried out on a laboratory bench, which includes an AKIP-9101 logic analyser and a prototype of the reconfigurable system. The effective implementation of the system under consideration is ensured due to the fact that the dispatching and scheduling subsystem is implemented as an independent coprocessor device. This solution unloads the CPU from task scheduling operations and increases overall system performance. The results obtained in the paper can be used in medicine, geographic information systems, and also specialized systems.

**Keywords:** reconfigurable computing system, high performance system, simulation modelling, scheduler, task manager, hardware implementation, FPGA.

## I. INTRODUCTION

The development and enhancement of logic in modern FPLDs of FPGA-type allows for more complex algorithms that can be programmed in a microcircuit. Interfacing this FPGA type with a modern processor (CPU) via a high-speed bus such as PCI Express allows the configurable logic to act as a coprocessor rather than a peripheral. This has led to reconfigurability in high performance computing. Reconfigurable computing systems (RCS) are promising developments in the field of high performance computing [1, 2, 3]. To obtain the necessary functional dependences of the characteristics in the developed and investigated systems on certain parameters, it is necessary to apply abstract mathematical models based on the mathematical relation language [4, 5]. Basically, the choice of the structure of a computing system is based on the study of the task flows entering and leaving the system for processing, the lengths of queues in front of the servicing devices, and the durations of the waiting times for service. The main research methods in this area are imitation, analytical, and experimental. The paper discusses an experimental method based on measuring the indicators of computational processes performed in real systems and devices, and processing the measurement results in order to determine the values required for research. Experimental studies provide more accurate data, but the results are usually particular.

### I.I Theory

This paper is by and large an exploratory one. When studying the subject area, the literature sources [6 - 10] were analysed in connection with the search for little-studied and unsolved problems. Of course, certain issues related to the simulation modelling of control nodes in a reconfigurable computing system using hardware means are rather poorly reflected in published works, but nevertheless, these issues are partially considered in the publications [11-13], [14-17] and [18-21].

The main goal of this paper is to conduct experiments to investigate the control nodes of a reconfigurable computing system for digital signal processing using hardware means. The designated issue is relevant today due to the widespread informatisation of society and the almost widespread processing of colossal volumes of data of a diverse level. To achieve the above goal, the work solves a number of problems to study the previously created [22, 23] prototype of the device and the principles of its functioning.

In this paper, the reconfigurable computing system is a device that consists of 4 reconfigurable CPUs implemented on the basis of a FPGA. The creation of such a system on the basis of the FPGA provides an opportunity for reconfiguring (resetting) the device for various classes of digital signal processing tasks and processing large amounts of data. Figure 1 shows a block diagram of the described reconfigurable computing system.

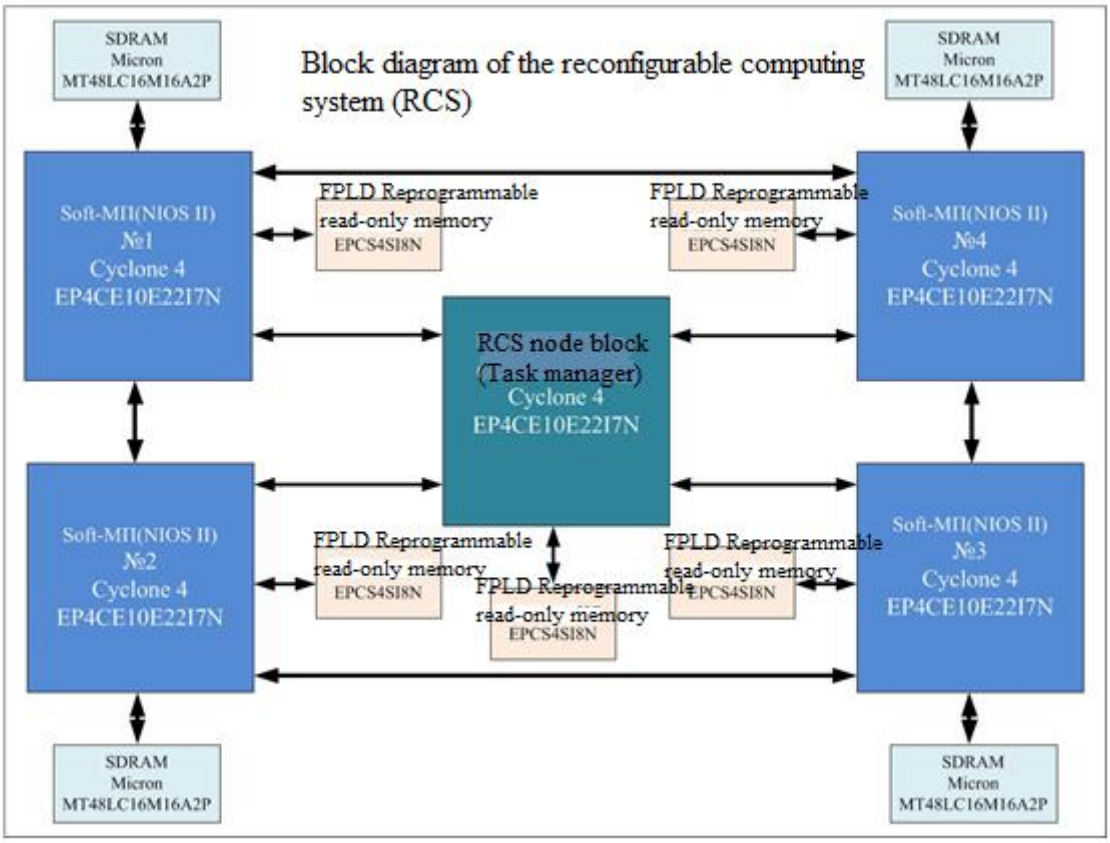



**Figure 1.** Block diagram of the reconfigurable computing system

A schematic diagram, a printed circuit board diagram (Figure 2), the printed circuit board itself (Figure 3, a) and a prototype of the device (Figure 3, b) were developed on the basis of the given block diagram.

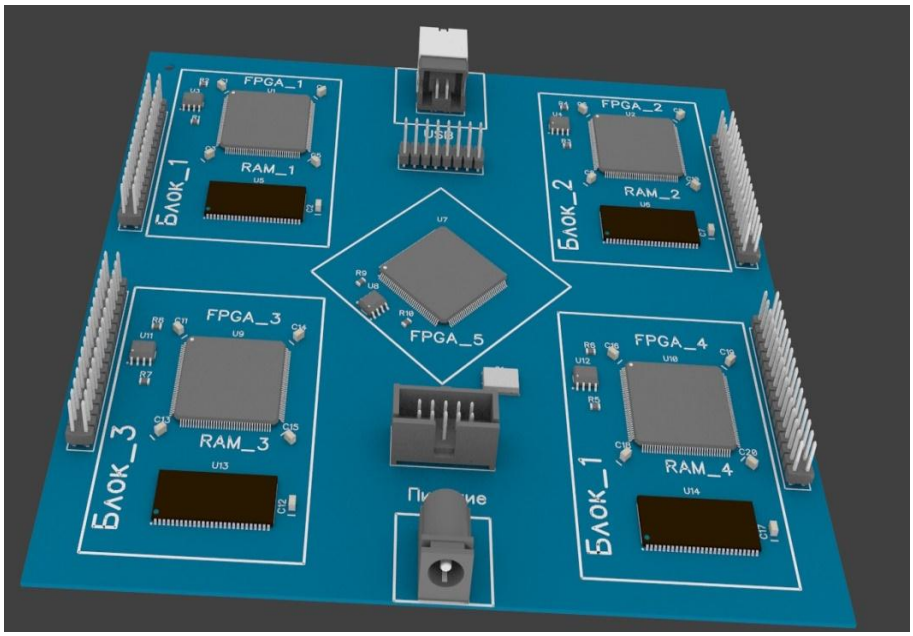
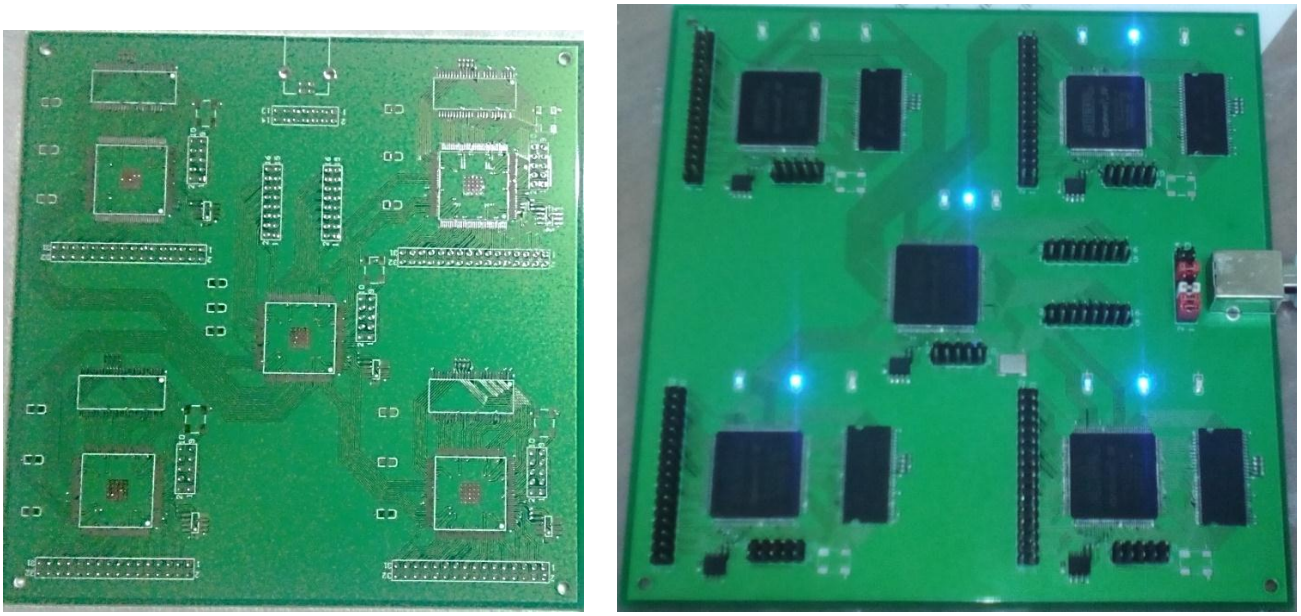



**Figure 2.** Schematic diagram of the printed circuit board for a reconfigurable computing system prototype



a b

**Figure 3.** Printed circuit board of the reconfigurable computing system prototype (a), the reconfigurable computing system prototype (b)

The embedded real-time reconfigurable computing system consists of the following components:

- 5 FPGAs Cyclone 4 EP4CE10E22I7N from Intel FPGA: 4 FPGAs are used to implement soft-microprocessor cores Nios II; one FPGA is dedicated to be a unit comprising control nodes of the system;

- 5 configuration devices for FPGA EPCS4SI8N from Intel FPGA to store the configuration data of each FPGA;

- 4 SDRAM MT48LC16M16A2 from Micron, 64 MB RAM blocks.

When creating a reconfigurable computing system, development engineers face the main problem associated with reducing the time losses when planning processes and flows within the system. At the operating system (OS) level, the assignment of processes (flows) to processor nodes is performed by the task dispatching function.

While fast enough, the software implementation in user space is complex, since 3 semaphores (counters) are required for the synchronization procedure. The first counter counts the number of places occupied by processes ready for processing, the second counts the number of active processors, and the third counter counts the mutex for the mutual exclusion function, which prevents some free processors from simultaneously accessing one queue, which is a shared system resource. To resolve this issue, the method of synchronization in the kernel space is used, but the increased time consumption entails a rather strong decrease in the performance of the reconfigurable computing system [24].

The optimal solution to the above problems lies in the hardware implementation of the process synchronization function (including the planning and task scheduling subsystem), since this removes the responsibility for performing these functions from the processors, increases the OS performance and reliability. The proposed method is based on the fact that the task manager (TM) function will be performed by an independent specialized CPU as part of the reconfigurable computing system [25].

Let's move on to considering a possible hardware implementation of the task manager, the block scheme of which is shown in Figure 4. Based on it, an algorithm for functioning in the Quartus II CAD system in VHDL language was developed. According to the presented block scheme of the task manager, it includes the following units: a task queue control unit, a FIFO unit, a free processor control unit, and a synchronizing unit.

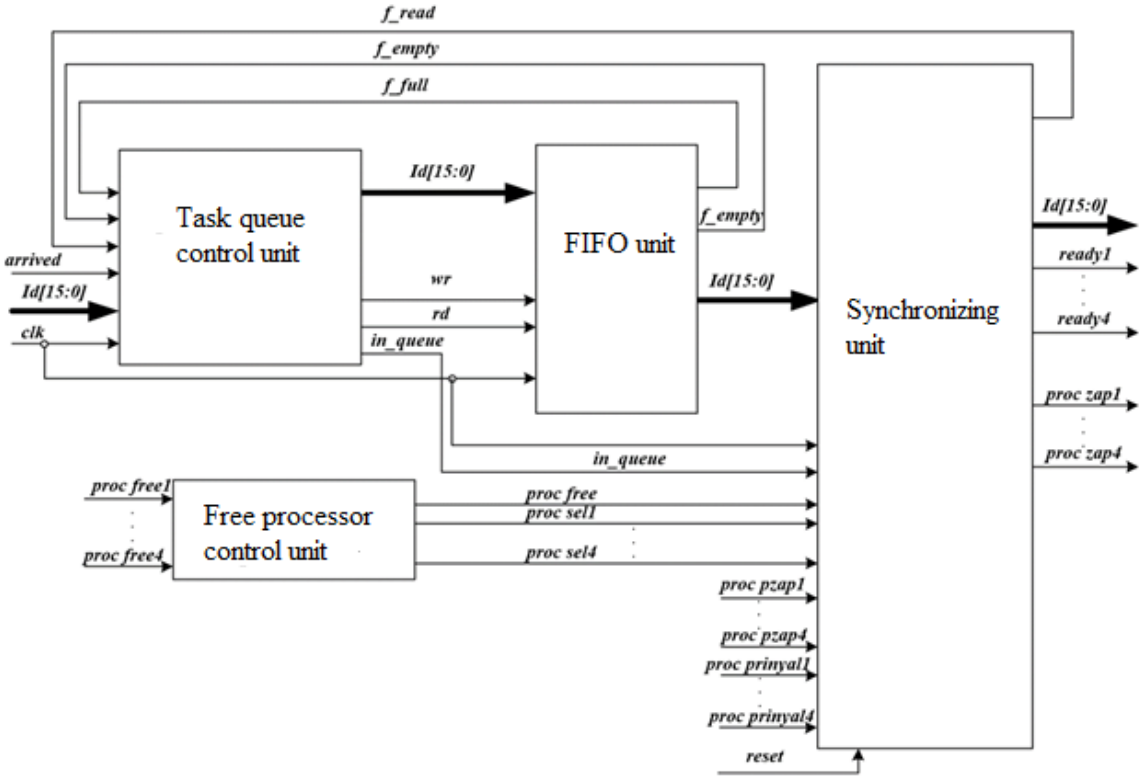



**Figure 4.** The task manager block scheme

The task queue control unit receives the identifiers of the incoming tasks. Then it determines whether there is a vacancy in the task queue and if there is free space, it sends the identifier of the new task to the FIFO queue unit. It also fetches the task identifier from the queue for processing in the free CPU.
In accordance with the request signal from the queue control unit, the FIFO unit places the identifier of the new task at the end of the list or fetches the identifier of the task from the head of the list for forwarding it to the CPU.

The synchronizing unit is the main unit in the device. Its functioning consists in the analysis of information about whether there are pending requests in the reconfigurable computing system, and whether there are free CPUs that can be assigned to service these requests. The synchronizing unit facilitates interaction with any CPU in the reconfigurable computing system under consideration and is responsible for broadcasting the task identifier to the free CPU assigned for processing [4].
Below there is a listing of the VHDL synchronizing unit code.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity block_dispetcher is
        port
        (
                clk                     : in std_logic;
                rst                     : in std_logic;
                s_task_in_queue: in std_logic;
                s_proc_free             : in std_logic;
                v_q_fifo_in             : in std_logic_vector(15 downto 0);
                s_proc_sel_1            : in std_logic;
                s_proc_sel_2            : in std_logic;
```

```
                s_proc_sel_3            : in std_logic;
                s_proc_sel_4            : in std_logic;
                s_proc_pzapr_1 : in std_logic;
                s_proc_pzapr_2 : in std_logic;
                s_proc_pzapr_3 : in std_logic;
                s_proc_pzapr_4 : in std_logic;
                s_proc_prinyal_1        : in std_logic;
                s_proc_prinyal_2        : in std_logic;
                s_proc_prinyal_3        : in std_logic;
                s_proc_prinyal_4        : in std_logic;
                s_fifo_read_out  : out std_logic;
                v_task_id_for_proc      : out    std_logic_vector(15 downto 0);
                s_task_ready_1  : out std_logic;
                s_task_ready_2  : out std_logic;
                s_task_ready_3  : out std_logic;
                s_task_ready_4  : out std_logic;
                s_proc_zapr_1_o         : out    std_logic;
                s_proc_zapr_2_o         : out    std_logic;
                s_proc_zapr_3_o         : out    std_logic;
                s_proc_zapr_4_o         : out    std_logic
        );
end block_dispetcher;
architecture block_dispetcher_arch of block_dispetcher is
signal reg_task_id : std_logic_vector(15 downto 0);
        type state_upr is( init, zapros_proc, p_zapros_proc, read_task,task_in_proc, wait_proc_read_task );
signal state                    : state_upr;
signal state_next               : state_upr;
signal  s_proc_zapr_1           : std_logic;
signal  s_proc_zapr_2           : std_logic;
signal  s_proc_zapr_3           : std_logic;
signal  s_proc_zapr_4           : std_logic;
signal  s_proc_g_1              : std_logic;
signal  s_proc_g_2              : std_logic;
signal  s_proc_g_3              : std_logic;
signal  s_proc_g_4              : std_logic;
signal cnt: std_logic_vector(3 downto 0):="0000";
begin
                s_proc_zapr_1_o<=s_proc_zapr_1;
                s_proc_zapr_2_o<=s_proc_zapr_2;
                s_proc_zapr_3_o<=s_proc_zapr_3;
                s_proc_zapr_4_o<=s_proc_zapr_4;
```

```
process(rst)
begin
        if rst='1' then
                state<=init;
        else
                state<=state_next;
        end if;
end process;
process (clk)
begin
        if (clk'event and clk='1') then
                case state_next is
                        when init =>
                                s_proc_zapr_1<='0';
                                s_proc_zapr_2<='0';
                                s_proc_zapr_3<='0';
                                s_proc_zapr_4<='0';
                                s_task_ready_1<='0';
                                s_task_ready_2<='0';
                                s_task_ready_3<='0';
                                s_task_ready_4<='0';
                                s_proc_g_1<='0';
                                s_proc_g_2<='0';
                                s_proc_g_3<='0';
                                s_proc_g_4<='0';
                                v_task_id_for_proc<=x"0000";
                                reg_task_id<=x"0000";
                                state_next      <=zapros_proc;
                                when zapros_proc =>
                if s_task_in_queue='1' and s_proc_free='1' and s_proc_sel_1='1' then
                        s_proc_zapr_1<='1';
                state_next      <=p_zapros_proc;
                elsif s_task_in_queue='1' and s_proc_free='1' and s_proc_sel_2='1' then
                        s_proc_zapr_2<='1';
                        state_next      <=p_zapros_proc;
                elsif s_task_in_queue='1' and s_proc_free='1' and s_proc_sel_3='1' then
                        s_proc_zapr_3<='1';
                        state_next      <=p_zapros_proc;
                elsif s_task_in_queue='1' and s_proc_free='1' and s_proc_sel_4='1' then
                        s_proc_zapr_4<='1';
                        state_next      <=p_zapros_proc;
```

```
            end if;
                        when p_zapros_proc =>
            if (s_proc_zapr_1 and s_proc_pzapr_1)='1' then s_proc_g_1<='1'; s_fifo_read_out<='1'; state_next
        <=read_task;
            elsif (s_proc_zapr_2 and s_proc_pzapr_2)='1' then s_proc_g_2<='1'; s_fifo_read_out<='1'; state_next
        <=read_task;
            elsif (s_proc_zapr_3 and s_proc_pzapr_3)='1' then s_proc_g_3<='1'; s_fifo_read_out<='1';state_next
        <=read_task;
            elsif (s_proc_zapr_4 and s_proc_pzapr_4)='1' then s_proc_g_4<='1'; s_fifo_read_out<='1'; state_next
        <=read_task;
            end if;
                        when read_task =>
                            s_fifo_read_out<='0';
            if cnt="0010"then
                cnt<="0000";
                reg_task_id<=v_q_fifo_in;
                state_next      <=task_in_proc;
            else
                cnt<=cnt+'1';
            end if;
                        when task_in_proc =>
                            v_task_id_for_proc<=reg_task_id;
            if s_proc_g_1='1'       then s_task_ready_1<='1';state_next     <=wait_proc_read_task;
            elsif s_proc_g_2='1' then s_task_ready_2<='1'; state_next <=wait_proc_read_task;
            elsif s_proc_g_3='1' then s_task_ready_3<='1';state_next  <=wait_proc_read_task;
            elsif s_proc_g_4='1' then s_task_ready_4<='1'; state_next <=wait_proc_read_task;
            end if;
                        when wait_proc_read_task =>
            if (s_proc_g_1 and s_proc_prinyal_1)='1' then v_task_id_for_proc<=x"0000"; s_task_ready_1<='0';
state_next      <=init;
            elsif (s_proc_g_2 and s_proc_prinyal_2)='1' then v_task_id_for_proc<=x"0000"; s_task_ready_2<='0';
state_next      <=init;
            elsif (s_proc_g_3 and s_proc_prinyal_3)='1' then v_task_id_for_proc<=x"0000"; s_task_ready_3<='0';
state_next      <=init;
            elsif (s_proc_g_4 and s_proc_prinyal_4)='1' then v_task_id_for_proc<=x"0000"; s_task_ready_4<='0';
state_next      <=init;
            end if;
            end case;
        end if;
end process;
end block_dispetcher_arch;
```

### I.II Experimental research

Simulation units were created for modelling the proposed task manager model in an experimental study with the use of 4 CPUs and a task generating unit. The simulation modelling was carried out using the ModelSim-Altera 10.0c Starter Edition software. After compilation, the project was simulated. Based on the simulation modelling results, time diagrams (Figure 5) for the operation of the reconfigurable computing system with the task manager were obtained.

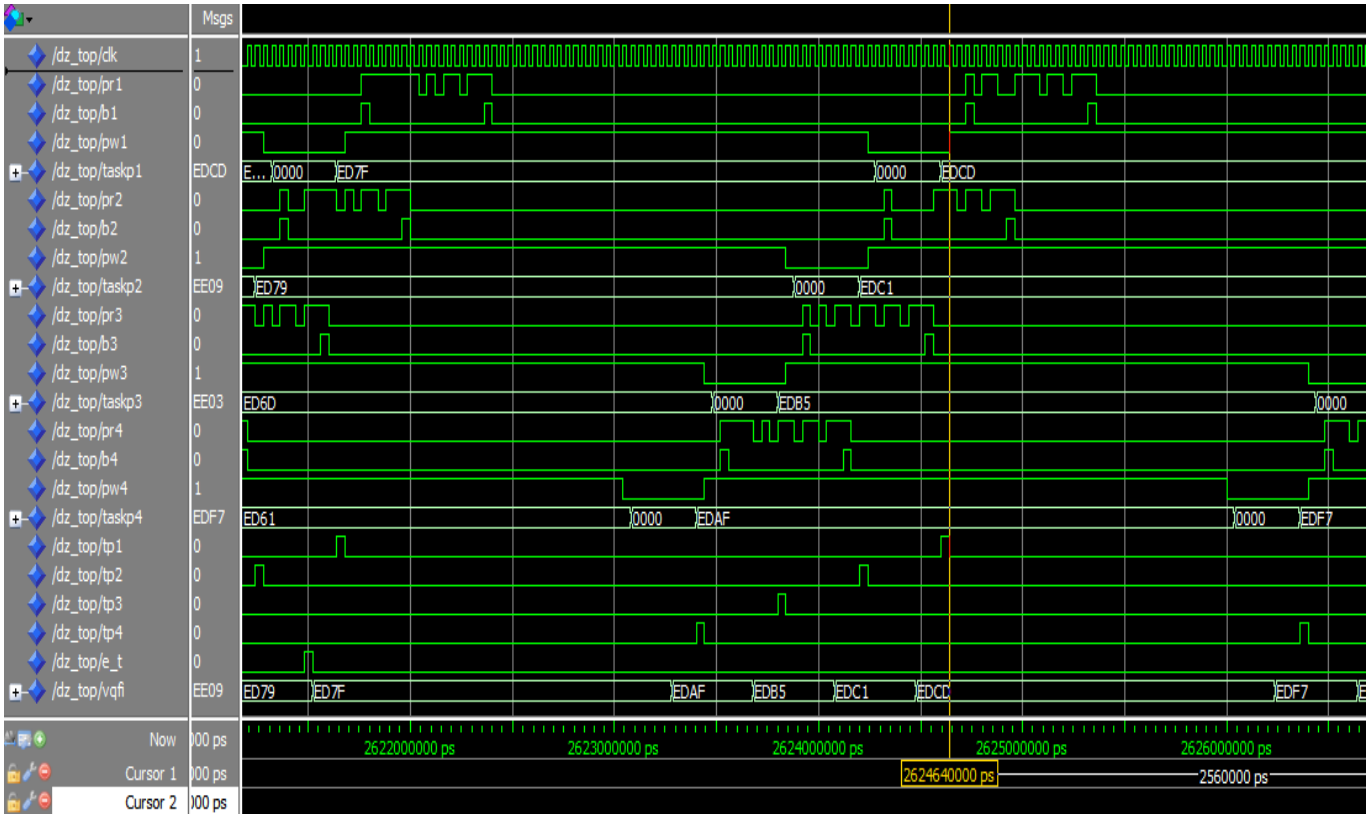



**Figure 5.** Time diagrams for the reconfigurable computing system in the ModelSim-Altera 10.0c Starter Edition software

From the shown time diagrams, it is clear that the fourth CPU is assigned to be first to serve the current task, followed by the third, etc. For example, the task identifier EDAF was accepted for servicing by the fourth CPU (taskp4), after which the CPU sent a signal that the task was being processed (tp4). The CPU number 4 was busy, so the next task ID EDB5 was accepted for service by the third CPU. Like the fourth CPU, it sets a signal (tp3) that it has started serving the task. This principle of assigning tasks is due to the scheme of priorities of the task execution in the system, i.e. from the fourth CPU to the first one [19, 20].

Typically, simulation modelling results do not always correspond to the results obtained with their hardware implementation. To verify the task manager operation algorithm presented in the work and timing diagrams obtained in ModelSim-Altera 10.0c Starter Edition, a full-scale experiment was carried out on the basis of a laboratory stand, which includes the AKIP-9101 logic analyser and a prototype of the proposed reconfigurable computing system (Figure 6).

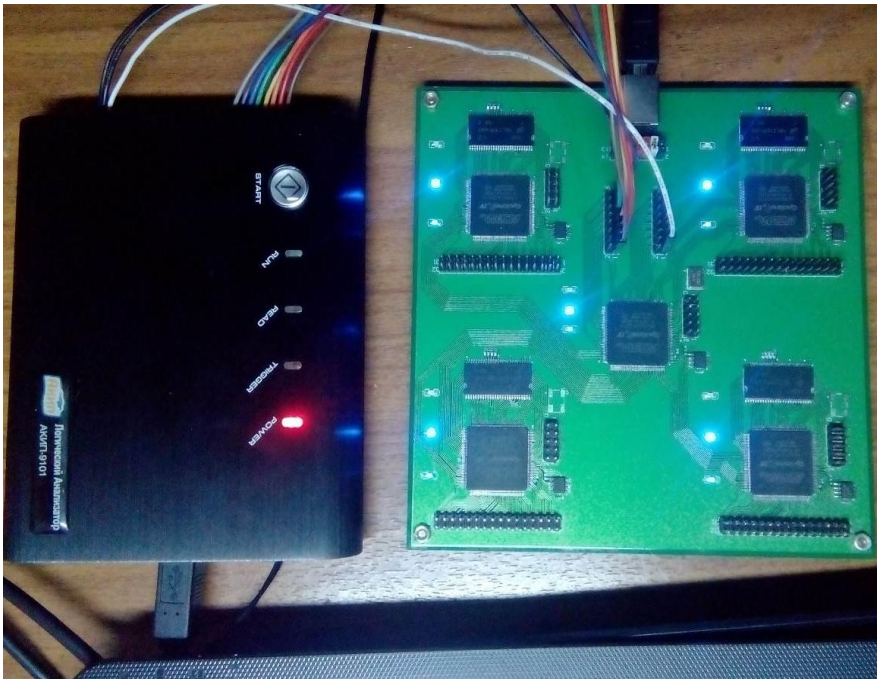



**Figure 6.** The stand for the full-scale experiment

Figure 7 shows the timing diagrams of the reconfigurable computing system operation, which were obtained in the real mode of operation.

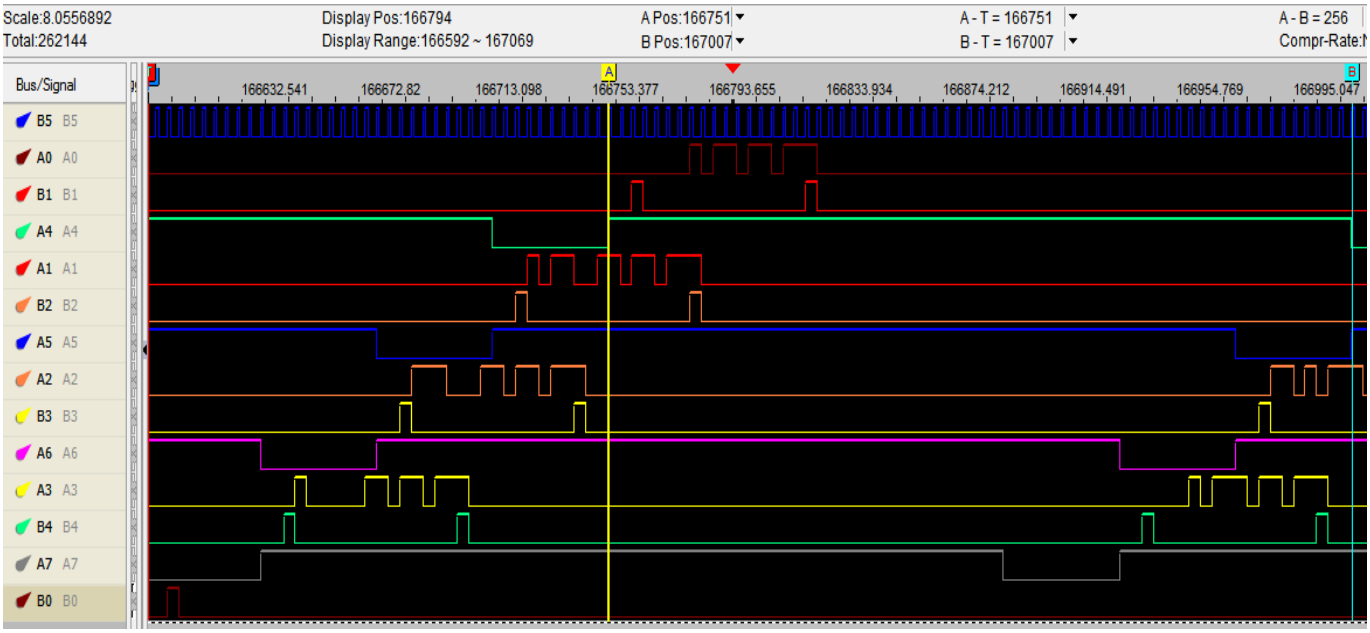



**Figure 7.** Timing diagrams of the reconfigurable computing system, which were obtained in the real mode of operation

Table 1 shows comparisons and descriptions of signals in the ModelSim-Altera 10.0c Starter Edition software and during operation of the AKIP-9101 logic analyser. Due to the fact that the logic analyser has a 16-bit bus, only the main signals are displayed for analysis: the task identifier presented in sequential form, the designation of the first and last identifier bits, the CPU task processing signal, the beginning of a new task flow, and the clock signal.

**Table 1.** Comparisons and descriptions of signals in the ModelSim-Altera 10.0c Starter Edition software and during operation of the AKIP-9101 logic analyser

| ModelSim-Altera 10.0c signals | AKIP-9101 signals | Signal assignment |
|---|---|---|
| clk | B5 | 50 MHz clock signal |
| pr1 | A0 | Task ID 1 in sequential form |
| b1 | B1 | Designation of the first and last bit of the CPU1 identifier |
| pw1 | A4 | CPU1 task processing signal |
| taskp1 | - | Parallel CPU1 Task ID (Hexadecimal code) |
| pr2 | A1 | CPU2 task identifier in sequential form |
| b2 | B2 | Designation of the first and last bit of the CPU2 identifier |
| pw2 | A5 | CPU2 task processing signal |
| taskp2 | - | Parallel CPU2 Task ID (Hexadecimal code) |
| pr3 | A2 | CPU3 task identifier in sequential form |
| b3 | B3 | Designation of the first and last bit of the CPU3 identifier |
| pw3 | A6 | CPU3 task processing signal |
| taskp3 | - | Parallel CPU3 Task ID (Hexadecimal code) |
| pr4 | A3 | CPU4 task identifier in sequential form |
| b4 | B4 | Designation of the first and last bit of the CPU4 identifier |
| pw4 | A7 | CPU4 task processing signal |
| taskp4 | - | Parallel CPU 4 Task ID (Hexadecimal code) |
| e_t | B0 | Starting a new task flow |
| vqfi | - | The task flow going to the task manager for its allocation to the CPUs |

According to the results of the experiment carried out using the stand, we can say that the algorithm of the task manager fully corresponds to its model in the ModelSim-Altera 10.0c Starter Edition environment, this can be seen from the timing diagrams of the CPU (pw1 signal, A4 signal in Figures 5 and 7, respectively) which is 256 clock counts or 2560 ns.

## II. CONCLUSIONS

The hardware implementation of the task manager completely removes the problem of time losses taking place during the distribution and synchronization of processes in the reconfigurable computing system under consideration. Based on the experiment carried out, it was found that the task manager is quite capable of accepting and assigning to service all tasks entering the system. It is not overloaded in any way and is able to handle a higher flow of incoming requests to be processed.

It is also necessary to note such an advantage of this development, which is associated with the implementation of a reconfigurable computing system with a modern element base (FPGA).

The application area for the results obtained in the paper is computing systems, where it is important to increase productivity and efficiency (processing of graphic information in medicine (X-ray, ultrasound, MRI, etc.), digitization and processing of maps in geographic information systems, specialized intelligent security systems).

## ACKNOWLEDGEMENTS

**The paper is published with the support of the scholarship of the President of the Russian Federation for young scientists and graduate students for 2018-2020 (SP-68.2018.5).**

## REFERENCES

[1] Kalyaev IA, Levin II, Semernikov EA, Dordopulo AI. Reconfigurable computing systems based on FPLDs of the VIRTEX-6 family. Bulletin of the Ufa State Aviation Technical University. 2011;15(5(45)):148-154.

[2] Levin II, Dordopulo AI, Kalyaev IA, Gudkov VA. High-performance reconfigurable computing systems based on FPLD VIRTEX-7. Parallel computing technologies (PAVT'2014): Proceedings of the International Scientific Conference. 2014:131-139.

[3] Martyshkin AI, Martens-Atyushev DS. Review of modern domestic reconfigurable computing systems. Modern methods and means of processing space-time signals:

Collection of papers of the XV All-Russian scientific and technical conference. 2017:65-69.

[4] Mayorov SA, Novikov GI, Aliev TI, Makharev EI, Timchenko BD. Fundamentals of the computing systems theory: textbook. Edited by S.A. Mayorov. - M.: Higher school. 1978:- 408 p.

[5] Aliev TI. Basics of the simulation modelling of discrete systems. SPb.: SPbGU ITMO. 2009:- 363 p.

[6] Tanenbaum E, Bos H. Modern operating systems. - SPb.: Peter. 2015:- 1120 p.

[7] Rumyantsev AS. Organization and tools of reconfigurable computing systems. Scientific and technical bulletin of information technologies, mechanics and optics. 2012;4:79-84.

[8] Platunov AE. Embedded control systems. Control Engineering. Russia. 2013;43(1):16-24.

[9] Jozwiak L, Nedjah N. Modern architectures for embedded reconfigurable systems–A survey. J. of Circuits, Systems, and Computers. 2009;18(2):209-254.

[10] Jozwiak L, Nedjah N, Figueroa M. Modern development methods and tools for embedded reconfigurable systems: A survey. Integration, the VLSI J. 2010;43(1):1-33.

[11] Martens-Atyushev DS, Martyshkin AI. Development and research of a reconfigurable system for digital signal processing. International student scientific bulletin. 2016;3-1:86-88.

[12] Martyshkin AI, Martens-Atyushev DS. Development of a subsystem for planning and assigning tasks of a reconfigurable computing system for digital signal processing. Modern methods and means of processing space-time signals: collection of papers of the XIV All-Russian Scientific and Technical Conference. Edited by I.I. Salnikov. 2016: 115-119.

[13] Martyshkin AI, Martens-Atyushev DS. Mathematical modelling and calculation of the probabilistic and temporal characteristics of a planning subsystem and assignment of tasks for a reconfigurable computing system intended for digital signal processing. Modern technologies in science and education - STNO-2017: collection of works of the II International scientific-technical and scientific-methodological conference: in 8 volumes. Ryazan State Radio Engineering University. 2017:210-215.

[14] Martyshkin AI. Mathematical simulation modelling of Tasks Managers with the strategy of separation in space with a homogeneous and heterogeneous input flow and finite queue. ARPN Journal of Engineering and Applied Sciences. 2016;11(19):11325-11332.

[15] Martyshkin Alexey I, Martens-Atyushev Dmitry S. Experimental study of a reconfigurable system with hardware task manager and a distributed queue. Journal of Computational and Theoretical Nanoscience. 2019;16(7):3040-3045.

[16] Martyshkin AI, Martens-Atyushev DS. Mathematical modelling and evaluation of the characteristics of specialized reconfigurable systems based on a common bus at the stage of synthesis of the system configuration. Journal of Advanced Research in Dynamical and Control Systems. 2019;11(8):2852-2860.

[17] Martyshkin Alexey I. Study of Distributed Task Manager Mathematical Models for Multiprocessor Systems Based on Open Networks of Mass Servicing. AD Alta-Journal of Interdisciplinary Research. 2018;8(1):309-314.

[18] Martens-Atyushev DS, Martyshkin AI. Development and research of the module for the dispatching subsystem of tasks of a reconfigurable computing system for digital signal processing. International student scientific bulletin. 2017;4-9:1411-1414.

[19] Martyshkin AI, Martens-Atyushev DS, Markin EI. On the issue of building a reconfigurable computing system based on FPLD for digital signal processing. Modern innovative technologies for training engineering personnel for the mining industry and transport. 2017;4:433-439.

[20] Martens-Atyushev DS. Development and research of a subsystem for dispatching tasks of a reconfigurable computing system for digital signal processing. Information technologies in economic and technical problems: Collection of scientific papers of the International Scientific and Practical Conference. 2016:247-250.

[21] Martyshkin AI. Investigation of the characteristics of the subsystem for assigning tasks of a reconfigurable computing system for digital signal processing. Models, systems, networks in economics, technology, nature and society. 2017;1(21):142-149.

[22] Martyshkin AI. Implementation of a prototype of a reconfigurable computing system for digital signal processing based on programmable logic integrated circuits. New information technologies and systems: collection of scientific papers of the XIV International Scientific and Technical Conference dedicated to the 70th anniversary of the Department of Computing Machinery and the 30th anniversary of the Department of Systems computer-aided design". Penza. 2017:243-246.

[23] Martens-Atyushev DS, Martyshkin AI. Experimental study of a planning subsystem of a reconfigurable computing system for digital signal processing. International student scientific bulletin. 2017;4-9:1408-1410.

[24] Biktashev RA, Knyazkov VS. Multiprocessor systems. Architecture, topology, performance analysis. Tutorial. - Penza: Publishing House of the Penza State University. 2004: - 107 p.

[25] Martyshkin AI. Mathematical modelling of task dispatchers in multiprocessor computing systems on the basis of stochastic queuing networks: abstract of the thesis for a degree of a candidate of engineering sciences: 05.13.18. Penza State Technological University. Penza. 2013:- 23 p.