An Approach of nMPRA Architecture using Hardware Implemented Support for Event Prioritization and Treating

Ionel ZAGAN[1,2], Nicoleta Cristina GAITAN[1,2] and Vasile Gheorghita GAITAN[1,2]
[1]Stefan cel Mare University of Suceava, 720229, Romania
[2]Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Stefan cel Mare University, Suceava, Romania

***Abstract*—One of the fundamental requirements of real time operating systems is the determinism of executing critical tasks and treating multiple periodic or aperiodic events. The present paper presents the hardware support of the nMPRA processor (Multi Pipeline Register Architecture) dedicated to treating time events, interrupt events and events associated with synchronization and inter-task communication mechanisms. Because in real time systems the treatment of events is a very important aspect, this paper describes both the mechanism implemented in hardware for prioritizing and treating multiple events, and the experimental results obtained using Virtex-7 FPGA circuit. The article's element of originality is the very short response time required in treating and prioritizing events.**

Keywords*—*nMPRA; event treating; mutex; inter-task communication; hardware scheduler

I. Introduction

Context switching and treating periodic and aperiodic events represent key factors in implementing real time schedulers, because they enable the operating system to allocate immediately higher priority events to the processor. In full-preemptive systems, the execution of the current task can be interrupted at any time by a task with a higher priority. In some implementations used for embedded systems, context switching can be completely prohibited in order to avoid unpredictable interferences between tasks, but also for enhancing the system's predictability. For some real time systems (RTS), the preemptive scheduler can be disabled only for certain periods of time during the execution of critical sections, such as the ISR (Interrupt Service Routines).

In these days, most commercial operating systems for embedded systems do not allow a task to synchronize with more events used for resource sharing, time management or asynchronous interrupts treatment.

A parameter with a negative influence on the performance on a real-time system is the over-control due to the operating system [1]. The scheduling algorithm and task context switching operations may significantly influence the scheduling limit for those systems with a high frequency of task switching. In many practical situations, such as I/O scheduling, or communication using shared environments, an interrupt is hard, or even impossible, to accept. This is because suspending the current task would cause an increase of the cache miss effect and negatively influence the pre-fetch mechanism, by involving an unpredictable worst-case execution time (WCET).

Sectoral Operational Program for Increase of the Economic Competitiveness co-funded from the European Regional Development Fund.

For identifying the peripheral device that generated the interrupt, four types of techniques can be employed [2]. The existence of multiple interrupts lines between the CPU and the I/O modules is the simplest one. Therefore, even if multiple lines are used, each line is likely to have attached more than one I/O module, so one of the following three techniques could be used for each line. The software pool technique lies in the fact that when the CPU detects an interrupt, it performs a branch to the ISR that tests each I/O module in order to determine which module triggered the event. The major disadvantage of this method is that it is inefficient and time consuming. Daisy chain, also called vectored interrupt, is a more efficient technique implemented in hardware and based on the recurrent checking of interrupt signals (hardware pool). In case of treating interrupts by using this method, all I/O modules partake the same line interrupt request. Bus arbitration is another technique for treating interrupts that uses the vectored interrupts concept. In this case, a priority scheme must be used, so that the process of assigning priorities to multiple devices deals with situations in which multiple I/O modules aim simultaneously at taking control.

In order to address questions and problems related to current RTS, this paper validates the treatment of multiple events by the nMPRA processor [3], thus demonstrating the functionality and the real time performances of the integrated scheduler and the flexibility of the nMPRA processor.

This paper is structured as follows: the first section contains a brief introduction, and section two describes the nMPRA processor architecture; section III addresses the implementation in hardware of the mechanism of treating multiple events; the experimental results thus obtained will be analyzed and discussed in section IV; section V presents related work and the paper ends with the final conclusions in section VI.

II. nMPRA Architecture and Hardware Support

The nMPRA processor implemented for n threads is based on a hardware implemented scheduler as an integral part of the processor entitled Hardware Scheduler Engine (nHSE). The nMPRA processor based on the five stage pipeline assembly line is designed to execute the MIPS instruction set [4], implementing new instructions for task scheduling operations. The nMPRA concept replaces the stack saving classical method with a remapping technique, which uses the replication of program counter, register file and pipeline registers for n threads, as shown in Figure 1. So, an instance of the CPU will be called semi CPU (sCPUi for task i). In order to implement the nMPRA project, the authors use nHSE with static scheduling algorithms for tasks, interrupts, and events. nHSE is directly responsible for the remapping operation of the pipeline registers set and of the registers file [5].

In order to ensure a rapid context switching, the nMPRA architecture is based on multiplying the general purpose registers. Thus, each semiprocessor implements 32 registers on 32 bits representing a bank of registers, and all banks make the register file of the nMPRA architecture. The control unit generates the control signal for the register file, according to the decoded instruction and the active sCPUi, so that, at a given time, three simultaneous operations are allowed for the same bank of registers, one for writing and two for reading. The register bank is switched through the signal generated by the nHSE module and each sCPUi has a corresponding register bank. The function contexts saving and the parameters transmission are performed in a similar way to that in the case of classic MIPS processors. The bank selection is performed in hardware, the operation being independent of instructions executed at the level of each sCPUi [6].

The ID/EX pipeline register stores the data signals obtained from decoding the instruction and extracting the operands from the register file and the control lines needed in the following stages. Therefore, in the ID stage from the assembly line, the decoding of the instruction read from memory in the IF stage and the reading of data from the register file are performed. The operands read from the register file will be either stored in the next pipeline stage, if the MIPS instruction is type R or I, or ignored as in the case of jump instructions [7]. The shift registers for memory alignment in a Word-wide memory of 32 bits is designed in the ID pipeline stage, and, in order to ensure the width of the word data, the hardware support for sign extended operation is also designed. For accessing the data memory during the reading or writing operations, the memory controller has been implemented in the MEM pipeline stage.

The control units dictate the operations of reading and writing in the data memory through the M_MemRead and M_MemWrite control signals; the control signals are transmitted and stored at every clock cycle once with the instruction context, throughout all stages of the assembly line. In designing these processor architectures, the operations of reading and writing data in memory are performed during a clock cycle, the on-chip implemented memory being dual-port, with multiple access that runs at a superior frequency to that of the processor. Because the current implementation places special emphasis on the development and validation of the nMPRA processor, ensuring the predictable execution of tasks in a hard real-time system with mixed-criticality, the memory controller and on-chip memory were designed only to meet the resource requirements for the validation of the nMPRA project.

The WB stage performs the writing of the result in the register or in the subsequent stage when the hazard detection signals the emergence of a hazard situation. Being previously memorized in the MEM/WB pipeline register, the multiplexer from the WB stage, controlled by the WB_MemtoReg control signal provided by the control unit, performs the selection of the registers resulting from the ALU unit and of the data read from memory. According to the arithmetic or logical operation, or the access to memory performed by the instruction executed, this stage will provide at output the necessary data.

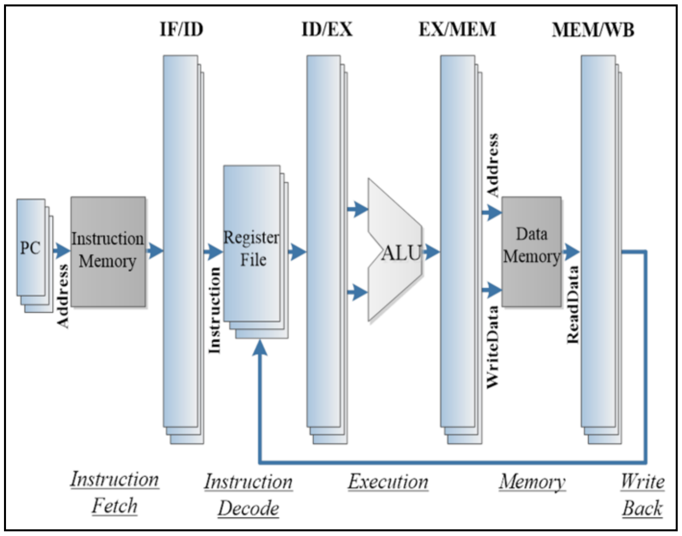

Fig. 1. Replication of resources of the nMPRA architecture. PC-program counter, IF/ID-Instruction Fetch/Instruction Decode, ID/EX-Instruction Decode/EXecute, EX/MEM-Execute/MEMory, MEM/WB-MEMory/Write Back pipeline register [8]

In the case of classical processor architectures, the saving and restoring of contexts is achieved through operations of accessing the external memory; the time needed to perform these operations depends directly on the number and the dimensions of the saved registers and the width of the data bus between the processor and the RAM memory.

The nMPRA processor uses a Harward memory architecture and the access to data and to instructions is performed in a separate address space. For the module to access the dual-port memory, the interface for both data and addresses is on 32 bits, using the big endian or little endian format, depending on the value of the Big_Endian parameter, set in the MIPS_Parameters.v file. The nMPRA supports memory accesses of type word, halfword and byte. The data memory bus is synchronous, used to access the RAM on-chip memory. It uses a minimum number of control signals and a simple protocol, in order to ensure that the data and instruction memory is accessed in writing, in a single clock cycle; the access to memory is performed on the positive edge of the clock signal.

III. PRIORITIZING AND TREATING EVENTS BY THE NHSE SCHEDULER

The nHSE is a finite state machine which has inputs for events, such as interrupts, deadline, watchdog timers, timers, mutexes, messages, and self-support execution. This implementation allows a very fast context switching that is possible due to the remapping of the active running task context with the scheduled task; the jitter is minimized in order to provide an accurate predictability behavior.

The nHSE architecture implements a hardware block with the role of arbitrating and announcing the sCPUi that has attached the event in question either directly, or through the active scheduler (static or dynamic). This block validates the command signals for each sCPUi. The events representing input signals for the nHSE module are the following: interrupts, the timer generated event, the event generated by exceeding deadline 1, the event generated by exceeding deadline 2, the event generated by the watchdog timer, the events generated by mutexes and the synchronization events.

The dynamic scheduler represents the support for the dynamic scheduling that enables the priority switching of a task, including that of a sCPUi. This scheduler is deactivated on reset only by the sCPU0. The nHSE module contains one register with the identifier corresponding to each sCPUi, one register with the priority set for the sCPUi used only by the dynamic scheduler, and a global register containing the identifier for the active sCPU that can be inhibited during the execution of atomic instructions. The sCPU0 will aways have the highest priority that is priority 0.

The crEPRi[i] register presented in Table 1, represents the priorities attached to each event that can be validated or not at the level of each sCPUi. Thus, each sCPUi can have different priorities for time events, interrupts, mutexes and synchronization events through messages.

The Pri_TEvi, Pri_WDEvi, Pri_D1Evi, Pri_D2Evi, Pri_IntEvi, Pri_MutexEvi and Pri_SynEvi bits groups represent the priorities attached to the categories of events validated or inhibited in the crTRi control register. Thus, when an event occurs, the corresponding bit will be set from the crEVi register.

The fact that interrupts have fixed priorities should be emphasized; the grINT_IDi[0] interrupt has the highest priority and the grINT_Idi[p-1]interrupt has the lowest; p is the number of interrupts from nMPRA. Although the priority of interrupts is fixed [9], they can be attached to any sCPUi and, at the level of each sCPUi, they can have different priorities given by the priorities set in the crEPRi register. The selection of the interrupt with the highest priority is performed through a hardware module that implements the priority encoder for interrupts.

IV. Experimental Results

The project has been implemented using the VC707 Evaluation Kit produced by Xilinx and Vivado 2015.4 design environment and the source code has been written in Verilog HDL. The implementation is based on the project described in [10], a 32-bit MIPS processor which aims for conformance with the MIPS32 Release 1 ISA. Figure 2 shows the clock_200MHzP and clock_200MHzN clock signals which represent the 200MHz differential signal available at the output of the SIT9102 oscillator and the clock signal of the nMPRA processor (clock) generated through the PLL block obtained with IP Clockind Wizard 5.2 (Rev. 1).

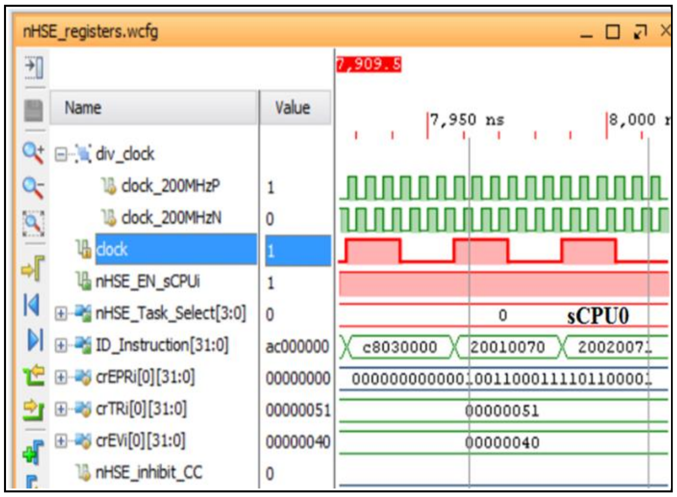

Fig. 2. The registers of the nHSE hardware integrated scheduler

The nMPRA processor architecture, using Virtex7 development kit, is defined and validated in the present paper, without describing the entire SoC project. Particular attention was paid to the nHSE real time scheduler, to improving the execution predictability by partially or completely eliminating hazards from the pipeline and minimizing the jitter for task context switching [11][12][13][14]. Compared to the theoretical version, in the version used to validate the processor, two clock signals were used. One clock cycle was used both for the pipeline registers, the register file, the internal logic of the scheduler and for treating external asynchronous interrupts [15]. The second clock cycle was used for the instruction and data memory.

The waveforms corresponding to the nHSE_EN_sCPUi, nHSE_Task_Select[3:0], ID_Instruction[31:0], crEPRi[0][31:0], crTRi[0][31:0], crEVi[0][31:0] and nHSE_inhibit_CC signals are also represented. The nHSE module generates the activation signals for all sCPUi semiprocessors through the nHSE_Task_Select[3:0] selector and the nHSE_EN_sCPUi validation signal; it can be inhibited under certain conditions, by the logic of the n events.

TABLE I. Assigning Priorities to Multiple Events using the crEPRi Control Register

31..21	20..18	17..15	14..12	11..9	8..6	5..3	2..0
-	Pri_MutexEvi	Pri_MutexEvi	Pri_IntEvi	Pri_D2Evi	Pri_D1Evi	Pri_WDEvi	Pri_TEvi

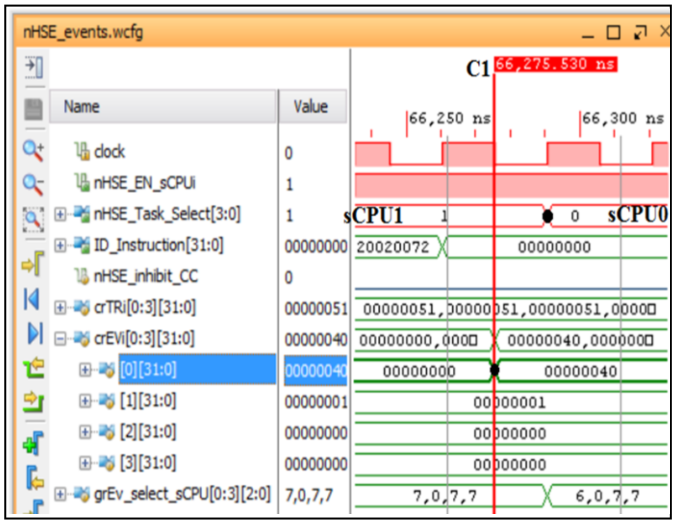

Fig. 3. The nHSE scheduler dictates a context switch for sCPU0 to treat the the synchronization event through messages

The grEv_select_sCPU[0:3][2:0] registers store the events treated by each sCPUi at any time moment. As can it be seen in Figure 3, the moment marked by the cursor C1 indicates the occurrence of a synchronization event through messages; the event is stored by the crEVi[0] = 0x00000040 register. Even if the sCPU1 semiprocessor treats a time event (grEv_select_sCPU[1] = 0), the scheduler performs the context switching operation because sCPU0 has the highest priority and the nHSE_inhibit_CC signal does not prevent the switching of the semiprocessors. The value 6 stored in the grEv_select_sCPU[0] register indicates the fact that sCPU0 treats a synchronization event through messages, validated through the crTRi[0] register. The crTRi[0] = 0x00000051 register validates time events, interrupt generated events, and events generated by the mechanism of communicating through messages.

As we can see in Figure 3, the contexts switch operation is guaranteed in one clock cycle. In a four sCPUi version as the one used for obtaining the waveforms in the present article, we can observe the ID_Instruction[31:0] pipeline register containing, at a certain moment, the code for the instructions extracted for each sCPUi.

Figure 4 represents the situation when there is a time event; time moment T1 represents the moment in which context switching is performed; the data stored in the pipeline registers are saved during the transition from sCPU0 to sCPU1. At time moment T2, the first instruction corresponding to sCPU1 (ID_Instruction[31:0] = 0x20020001) is extracted. This switch takes place under the strict command of the nHSE static scheduler, through the nHSE_Task_Select[3:0] and nHSE_EN_sCPUi nHSE signals, the time needed for switching contexts is no more than one clock cycle.

The crEPRi[0:3][31:0] register stores the priorities of the 7 events validated through the crTRi[0:3][31:0] registers. Thus, at the level of the semiprocessor sCPU0 corresponding to the validated events, the following priorities are already set: Pri_TEvi=3’b001, Pri_IntEvi=3’b000 and Pri_SynEvi=3’b010. A smaller value represents a higher priority, Pri_IntEvi=3’b000 being the event with the highest priority. The priority level of each category of events can be changed dynamically through the instructions dedicated to the nHSE scheduler, in relation to the requirements of the real time system.

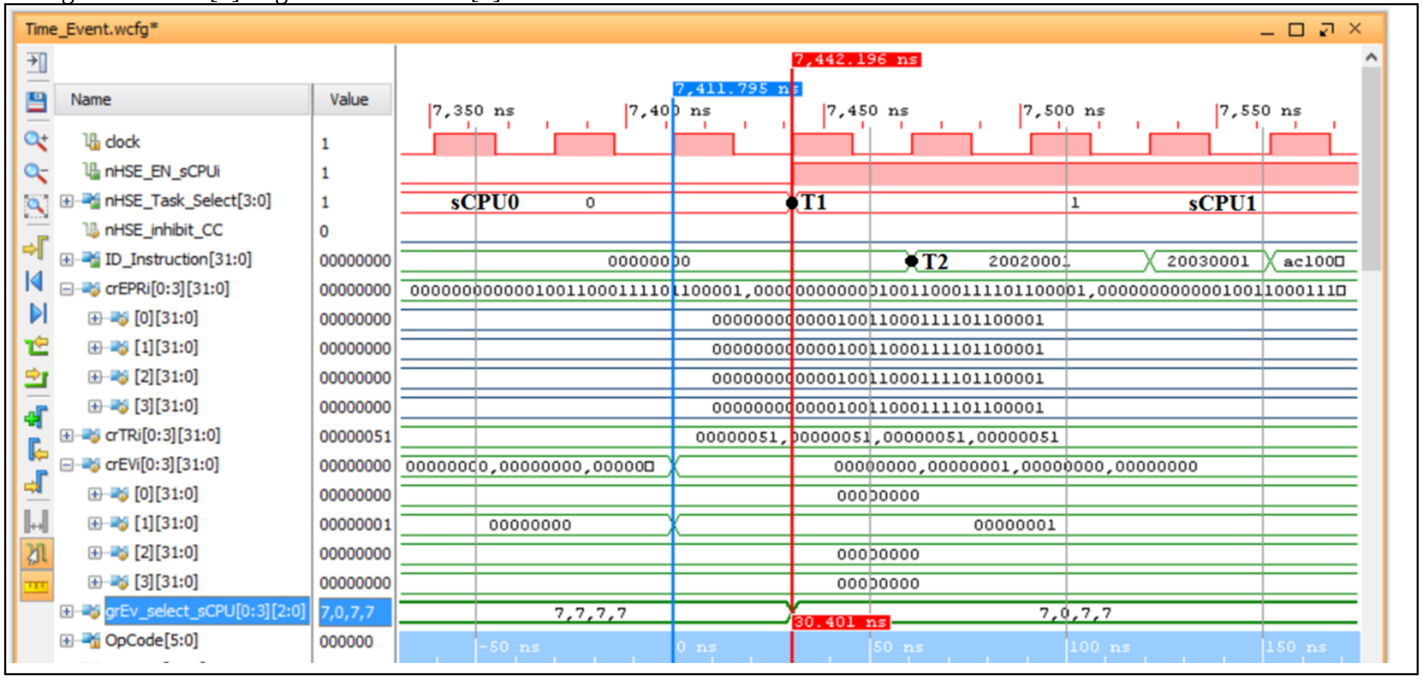

Fig. 4. The treating of an event using the nHSE architecture; clock - nMPRA clock; nHSE_EN_sCPUi - nHSE enable signal; nHSE_Task _Select[3:0] - nHSE task selector; nHSE_inhibit_CC – context switch inhibit signal; ID_Instruction[31:0] - wire type instruction; crTRi – enable event register; crEVi – events register; grEv_select_sCPU - current event identifier

The grINT_PR global register implemented in the nHSE scheduler stores the number of the interrupt with the highest priority, the selection being performed in hardware. The nMPRA architecture guarantees the execution of the new scheduled task starting with the next clock cycle, as we can see in Figure 4, at the moment T1.

We remind that all sCPUi share the same functional units, such as ALU, the control unit, the condition unit, the unit for hazard detection, and the redirection of data unit, so that the data path must guarantee the hardware isolation and the consistency of sCPUi contexts [16]. At a 33MHz frequency, the scheduler answer to an time related event may be around 30.401ns (one clock cycle). It can be said that the experimental results demonstrate the practical implementation of the theoretical aspects, therefore obtaining very low times for events handling and context switching operations.

The aim of this test is to verify and validate the custom interrupt management scheduling policy implemented in nHSE, and emphasize the added performance brought by the nMPRA processor in comparison to the theoretical elements presented in section III. Because the datapath is shared by all sCPUi implemented in the nMPRA processor, their contexts must be preserved and made available at any time moment to the real-time nHSE scheduler.

The goal of this implementation is not to describe a complete solution of the data path, but to validate the practical implementation of the nMPRA architecture and of the nHSE scheduler, using a flexible and competitive FPGA development platform. To design the nHSE module and to obtain better performances brought by the nMPRA, an analysis of events handled through software as well as hardware was necessary in the case of treating interrupts in a classical computing system.

V. Related Work

This chapter presents a brief description of two predictable processor architectures which can be compared with the results presented in this paper using the nMPRA processor.

The Merasa concept [17] was developed to obtain a processor architecture which can be successfully used in hard real-time embedded systems. Concerning the architecture, each core can have only one hard real-time (HRT) execution thread and an arbitrary number of non-HRT (NHRT) execution threads. Each core is made up of two scratchpad memories; one of the memories is dedicated for data and the other for instructions (D-ISP and DSP); the data integrity is ensured by individual allocation of a subnet of banks cache for each task. The priorities for execution threads are fixed and the scheduling policy chosen is round robin. Taking into consideration that the embedded systems have limited resources available, the Merasa architecture must offer an optimal cost for the implementation of an average number of HRT and NHRT execution threads, including their synchronization and communication mechanisms. If the HRT thread is suspended, pending an external interrupt, time event, or sharing a resource with another HRT or NHRT task, its dedicated assembly line will remain unused and it will negatively influence the performance of the entire system.

In [18], Andalam proposes a new predictable architecture called ARPRET. The ARPRET architecture is implemented and synthesized on the Xilinx ML-403 FPGA device, obtaining predictability by projecting a particular soft-core coupled with a hardware accelerator, called the Predictable Functional Unit. Thus, time behavior for models and programs becomes most important because, in order to guarantee that a hard real-time system behaves according to the model, their characteristics must be preserved during compilation.

VI. Conclusion and Future Work

The nMPRA architecture is versatile and very flexible because of the following reasons: the prioritization of multiple events attached to a sCPUi, the wait instruction which enables the implementation in hardware of a logical OR between these events, and the implementation in hardware of synchronization and inter-task communication mechanisms. However, the priorities of tasks, interrupts, and synchronization mechanisms can be ordered in any way, in order to meet the requirements of the real time application, whose central element can successfully be the nMPRA processor.

Each task is executed based on its own context, without depending on other system tasks or scheduler actions. For inter-task communication, nMPRA can ensure the implementation of queues, enabling data to be safely transferred between tasks. The hardware implementation of queues is flexible and can be used to obtain a number of objectives, including simple data transfers and synchronization through mutexes. By sending and receiving data using queues, the events specific to the queues implemented in classical CPU architectures can be used.

The performances of the nMPRA architecture can be improved by designing a cache memory for optional data and a memory protection module for unauthorized access, thus satisfying the constraints of the hardware isolation of tasks. The following papers should consider both the improvement of the memory architecture and the comparison with other similar implementations. The negative collateral effect generated by obtaining these outstanding performances, essential for mixed criticality real-time systems, is the memory consumption for multiplying in hardware the multiplexed resources, such as PC, register file and pipeline registers.

Acknowledgment

This work was partially supported from the project “Integrated Center for research, development and innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for fabrication and control”, Contract No. 671/09.04.2015, Sectoral Operational Program for Increase of the Economic Competitiveness co-funded from the European Regional Development Fund.

This paper has been prepared with the financial support of the project BG58/2016: “Development and integration of a mobile tele-electrocardiograph within the monitoring and diagnosis system of patients GreenCARDIO”, acronym: m-GreenCARDIO, PNCDI III, P2 - increasing competitiveness through RDI Romanian economy, Transfer of knowledge to the economic agent. Contract no. PN-III-P2-2.1-BG-2016-0463.

REFERENCES

[1] G. C. Buttazzo, ”Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications,” Third edition, pp. 13–30, Springer, 2011. ISBN: 978-1-4614-0675-4

[2] W. Stallings, “Computer Organization and Architecture,” 10th Edition, pp. 263–272, 2015. ISBN: 978-0134101613

[3] V. G. Gaitan, N. C. Gaitan, and I. Ungurean, “CPU Architecture Based on a Hardware Scheduler and Independent Pipeline Registers,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 9, pp. 1661–1674, Sept. 2015. doi:10.1109/TVLSI.2014.2346542

[4] D. A. Patterson and J. L. Hennessy, “Computer Organization and Design, Revised Fourth Edition: The Hardware-Software Interface,” Fourth Edition, pp. 330–379, 2011. ISBN: 978-0-12-374750-1

[5] I. Zagan, “Improving the performance of CPU architectures by reducing the Operating System overhead,” in 3rd IEEE Workshop on Advances in Information, Electronic and Electrical Engineering AIEEE’2015, Riga, Latvia, pp. 1–6, 13–14 Nov. 2015. doi: 10.1109/AIEEE.2015.7367279

[6] E. Dodiu and V. G. Gaitan, “Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – concept and theory of operation,“ in IEEE EIT International Conference on Electro-Information Technology, Indianapolis, USA, pp. 1–5, May 2012. doi:10.1109/EIT.2012.6220705

[7] “MIPS® Architecture For Programmers Volume I-A: Introduction to the MIPS32® Architecture,” Revision 3.02, Mar. 2011, Available: https://courses.engr.illinois.edu/cs426/Resources/MIPS32INT-AFP-03.02.pdf. (Accessed: May 2016).

[8] I. Zagan and V. G. Gaitan, “Predictable CPU Architecture Designed for Small Real-Time Applications – Implementation Results,” in: 3rd International Conference on Advances in Computing, Electronics and Communication (ACEC), 10 - 11 October 2015 / Zurich, Switzerland, pp. 143-150, ISBN: 978-1-63248-064-4. doi:10.15224/ 978-1-63248-064-4-29.

[9] S. Kelinman and J. Eykholt, “Interrupts as threads,” ACM SIGOPS Operating Syst. Rev., vol. 29, no. 2, pp. 21–26, Apr. 1995. doi:10.1145/202213.202217

[10] I. Zagan and V. G. Gaitan, “Improving the Performances of the nMPRA Processor using a Custom Interrupt Management Scheduling Policy,” in Advances in Electrical and Computer Engineering (AECE), ISSN 1582-7445, Volume 16, Issue 4, pp. 45-50, 2016. doi:10.4316/AECE.2016.04007

[11] E. E Moisuc, A. B. Larionescu, and V. G. Gaitan, “Hardware Event Treating in nMPRA,” in 12rt International Conference on Development and Application Systems – DAS, Suceava, Romania, pp. 66-69, 15–17 May, 2014. doi:10.1109/DAAS.2014.6842429

[12] E. Dodiu, V. G.Gaitan, and A. Graur, “Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – architecture description”, in IEEE 35’th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics, Croatia, pp. 859-864, 24 May 2012. INSPEC Accession Number: 12865464

[13] L. Andries and V. G. Gaitan, “Dual Priority Scheduling algorithm used in the nMPRA Microcontrollers,” in 18th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, pp. 43-47, 2014. doi:10.1109/ICSTCC.2014.6982388

[14] L. Andries, V. G. Gaitan and E. E. Moisuc, “Programming paradigm of a microcontroller with hardware scheduler engine and independent pipeline registers - a software approach,” in 19th International Conference on System Theory, Control and Computing (ICSTCC), Cheile Gradistei, Romania, pp. 705-710, 2015. doi: 10.1109/ICSTCC.2015.7321376

[15] I. Zagan and V. G. Gaitan, “Schedulability Analysis of nMPRA Processor based on Multithreaded Execution,” in 13rt International Conference on Development and Application Systems – DAS, Suceava, Romania, pp. 130-134, May 19–21, 2016. doi:10.1109/DAAS.2016.7492561

[16] N. C. Gaitan, I. Zagan, and V. G. Gaitan, “Predictable CPU Architecture Designed for Small Real-Time Application - Concept and Theory of Operation,” International Journal of Advanced Computer Science and Applications, vol. 6, no. 4, 2015. doi: 10.14569/IJACSA.2015.060406

[17] T. Ungerer et al., Merasa: Multicore execution of hard real-time applications supporting analyzability, IEEE, Micro, vol. 30, no. 5, pp. 66–75, 2010. doi: 10.1109/MM.2010.78

[18] S. Andalam, Predictable platforms for safety-critical embedded systems, Thesis, The University of Auckland, 2013.