Synthesis Analysis and Evaluation of Hardware Scheduler based on Different Scheduling Algorithms

Ionel Zagan[1,2]
[1] Faculty of Electrical Engineering and Computer Science,
Stefan cel Mare University, Suceava, 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
zagan@eed.usv.ro

***Abstract*—The limitations of the current real-time operating systems are given by the CPU architecture, the memory, and the I/O subsystem, but also by the high-level languages and their compilers, and the unpredictable answer of the asynchronous interrupts. The most important issue of a real time system is resource scheduling: processor, memory, I/O ports and communication networks, when systems are distributed. The present article gives an overview of the state of the art of qualitative research in terms of tasks schedulability algorithms used for the nMPRA (Multi Pipeline Register Architecture) CPU architecture in real-time environments. By introducing a jitter of maximum three clock cycles for the contexts switching of tasks, the nMPRA processor proves a deterministic hardware implementation, due to the integrated nHSE (Hardware Scheduler Engine for *n* tasks). This paper presents and analyzes the current state of real time operation systems with hardware implementation functions.**

Keywords—Pipeline processing, Field programmable gate arrays, Architecture, Operating systems, Scheduling.

I. INTRODUCTION

Real-time operating systems (RTOS) are present in all embedded applications in economic and social areas. We can see that there are increasingly fewer systems that do not use one or more microprocessors. For this reason, research in this field has expanded, improving RTOS from all points of view, and thus guaranteeing outstanding performances for real-time applications. Processor development, the need for optimization and the way in which the processor is used, the scheduling algorithms, and RTOS, have enabled a resolution time of a few microseconds. All these factors have made time organization more efficient so as to guarantee both the predictability of task execution and meet the imposed time requirements. The result of the current technological developments was that, in most areas a net superior quantitative and qualitative factor has been achieved, as is the case for the industrial, automotive or medical sector.

The novelty brought by this paper relies in the tests performed for the validation of the priority scheduling model implemented in nHSE, using the FPGA Virtex-7 circuit. In addition to these tests designed to validate the preemptive scheduler, this paper also describes and analyzes some scheduling models and algorithms that can be implemented in hardware (nHSE) in order to meet the stringent requirements imposed by a critical real-time system (RTS). The nMPRA processor architecture [2], [3] is an innovative one with very low response times to external interrupts. Improving these times, as well as minimizing the time for task context switching, is the main research purpose of this paper.

Context switching is a key factor in real-time scheduling algorithms, because it enables the operating system to immediately allocate the processor to higher priority tasks. The present paper describes and compares the current approaches that have contributed to reducing the segmentation of the program implementation. It also presents effective methods for minimizing scheduling costs by eliminating unnecessary interrupts. In full-preemptive systems, the execution of the current task can be interrupted at any given moment by a task with a higher priority. The execution of the interrupted task will resume only when there are no more higher priority tasks ready to run. In some implementations, contexts switching can be completely forbidden in order to avoid unpredictable interferences among tasks, and also to improve the predictability of the system [4]. For some real time systems, the preemptive scheduler can be disabled for certain time cycles during the execution of critical sections, such as the Interrupt Service Routine (ISR).

The spatial isolation, required in critical embedded systems for protecting the state of critical execution threads, can be guaranteed by programming each task on different execution components, such as: execution threads (in a multi-thread architecture) or the core (in a multi-core processor). Therefore, guaranteeing the spatial isolation by using a processor for each task causes inefficient use of resources, making it a robust but unacceptable solution. To improve the performance of real-time operating systems, the temporary isolation of competing execution threads must be guaranteed. The predictability of the threads over time facilitates a tight margin for Worst-Case Execution Time (WCET), avoiding excessive or inefficient use of resources [5]. For processors in the field of mobile applications, designing multithreading or hyperthreading architectures enabled better processor time management, without the increase of processor operating frequency.

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

978-1-5386-1495-2/18/$31.00 ©2018 IEEE

This paper is structured as follows: the first section contains a brief introduction, and section II analyses similar projects published in the literature. Sections III and IV addresses the implementation in hardware of the nHSE module and nMPRA processor architecture including the experimental results, and section V describes several scheduling algorithms used in real-time systems with support of the proposed CPU architecture. Section VI concludes the paper presenting final conclusions.

II. Related Work

The following part describes and analyzes various CPU architectures proposed by the literature. The purpose of this analysis was to outline the new research directions for improving the RTOS with hardware implementation functions.

In [6] Rochange and Sainrat proposed changes in the pipelines of superscalar dynamic processors by slowing down the instructions between blocks. As can be seen in [6], the authors approached the idea of out-of-order speculative execution using a pipeline with six stages. The architecture has been designed so that the time spent by the base blocks is independent from one block to another. This is due to slowing down the instruction from a base block until the instructions in the previous block are executed.

Article [7] presents the architecture proposed by Nordstrom et al. Through this implementation, the authors provide valuable time-related recommendations for future processor architectures in real-time embedded systems. Starting from the principle of reducing the interference in shared resources, they recommend using separate cache memories for instructions and data, using a Least Recently Used (LRU) replacement algorithm and the so-called compositional architectures, such as ARM7 [8].

The basic idea of the Komodo project, presented by Kreuzinger et al. in [9], [10], is to use the Komodo Java-based multi-thread microcontroller to manage multiple real-time execution threads. The proposed architecture uses multiple PCs (Program Counter), instruction windows, stacks, and a signal unit, for managing a set of execution threaded activated through interrupts. To provide real time support, the authors improved the picoJava instruction set.

The Komodo Priority manager performs the scheduling of the execution threads with different priorities on a four-stage assembly line, ensuring a rapid switch of contexts [11]. In the case of jump or memory access instructions, which can cause the stagnation of the assembly line, the scheduler can assign the unused cycles to another execution thread. Therefore, even if the Komodo project allows other execution threads in the ready state to be executed when the assembly line is available, thus increasing the instruction interpolation factor, this paper does not describe any synchronization or communication mechanism. In [12], Al-Zawawi et al. proposed an architecture that can be partitioned in a set of virtual processors. The execution times of these processors are independent of each other, providing tasks that are being executed on virtual processors with an autonomous feature [13]. The architecture proposed by the authors presents its partitioning in a few high performance processors, several smaller performance processors, or a combination of the two extremes.

By parametrizing the Verilog HDL implementation, it was possible to analyze the FPGA resources needed for these extensions, thus performing a comparative analysis with other similar CPU implementations [14], [15]. Analyzing the general characteristics of various processor architectures, we can conclude that the development of real-time mechanisms and the scheduling of tasks with strict time conditions represent a current challenge in the field of both RTS and embedded systems, with their future version, Cyber-physical systems. In this context, the important role of hardware schedulers is highlighted; these schedulers take over the scheduling activity thus relieving the processor from this task. This is why, the proposed nMPRA architecture uses a hardware scheduler as part of the processor; the scheduler is controlled through integrated instructions transmitted through the assembly line.

III. nMPRA and nHSE Architecture – Concept and Theory of Operation

The proposed nMPRA architecture, with multiplied pipeline registers, uses the organizational structure of the MIPS32 architecture [16]. The low energy and power consumption features of the embedded systems that represent MIPS32 implementations, availability of development tools for embedded systems, mature architecture, coprocessor 2 at the developer's disposal, the open-source feature, and the fact that MIPS32 is a well-known architecture, all of these provide the MIPS processor with an important role in the embedded system industry. The nMPRA architecture is based on remapping the multiplexed resource. Thus, a MIPS pipeline was considered [17] and the PC, the pipeline registers and the GPR (General Purpose Registers) were multiplied *n* times (Fig. 1). All the tasks running on the CPU use the same datapath, Control Unit, Hazard Detection Unit, ALU (Arithmetic Logic Unit) and Forward Unit [18]. So, an instance of the CPU will be called semiprocessor (sCPUi for task *i*). The difficulties in this research occur at the level of the RTOS, due to its hardware embedded functionality and the fact that it highlights the performance improvements by suitable test programs.

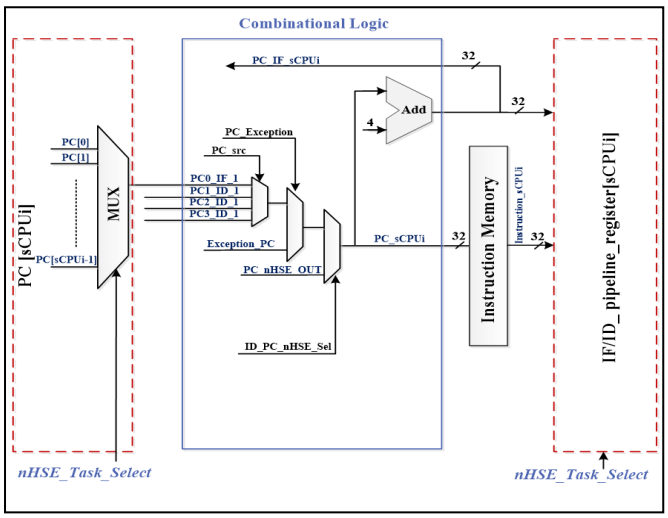

Fig. 1. Part of the combinational elements designed in Instruction Fetch stage and multiplication of PC and Instruction Fetch (IF)/Instruction Decode (ID) pipeline registers.

The paper makes the following contributions and improvements to the nMPRA implementation:

- The integration of the stages to the assembly line of the nMPRA architecture requires the multiplication of resources.
- The nMPRA processor has been designed using Verilog hardware description language with sCPU4, sCPU8 and sCPU16, scalable, parameterized, and well documented.
- Integration of MIPS32 processor basic modules in accordance to the nMPRA architecture specifications: the control unit, COP0, the memory controller, the hazard detection unit, the data forwarding unit and the ALU unit.
- The multiplication and multiplexing of all registers (PC, register file, pipeline registers, COP0 registers, Control unit registers and division unit, as well as any other memory element from the datapath) is performed.
- The implementation of the nMPRA processor with sCPU4, sCPU8 and sCPU16 (meeting the resource requirements), as well as the entire SoC project use the Virtex-7 development kit.

Due to the multiplication of resources, the guarantee regarding the spatial isolation of contexts, the performance of the entire system, and the need to meet the requirements of automotive (ISO26262) and industrial standards, the nMPRA architecture is highly recommended for real-time small-scale applications. The hardware implementation of scheduling schemes and significant reduction of task switching times have led to a competitive and innovative architecture. Moreover, unlike in other proposed implementations, we considered that in a multitasking system, inter-task synchronization and communication mechanisms are absolutely necessary. Tests have been performed in order to verify the functionality of the introduced mechanisms. Implementing a processor architecture without such mechanisms would be an unrealistic approach, as it is not a viable solution with practical applicability.

The nHSE scheduler uses a unified space for interrupts and tasks, and a scheduling rule whereby a high priority task cannot be interrupted by events assigned to low priority tasks. This research analyses both the waveforms obtained through the Vivado simulator and those provided by the PicoScope 2205MSO oscilloscope and ChipScope Analyzer. For this, the real data in the FPGA circuit, where the nMPRA circuit has been previously loaded as integrated part of the SoC project, have been read through the JTAG port, thus proving the hardware functionality of the architecture described in this research article.

Fig. 1 presents the way of interconnecting multiplied resources with combinational logic corresponding to the MIPS32 architecture. Although the PC register has been multiplied for each sCPUi, the selection logic remained the same, the program memory being accessed to extract the following valid MIPS32 instruction. This article discusses the aspects related to nMPRA functional processor blocks (Fig. 2), different scheduling algorithms suitable for nHSE, nMPRA processor and preemptive scheduler instructions attained in hardware, and the required registers for the implementation of the nHSE scheduler. For testing the project, various versions of the processor running at different working frequencies were implemented and synthesized, as well as a varying number of semiprocessors (4, 8 and 16). This was necessary due to the high working times with Xilinx tools (synthesizing, implementing, additional times introduced by In-system debug kernels) and to obtaining the resources (LUTs and FFs) needed for the hardware implementation of the processor.

IV. Schedulability Analysis of nMPRA Concept and Hardware Support

To ensure the compliance with the real-time characteristic, the scheduler uses a unified space for tasks and interrupts. Because interrupts inherit the priority of the task to which they are attached, using a priority-based scheduling rule that does not allow high priority tasks to be interrupted by lower priority tasks, the execution deadline for the hard real time external stimulus can be guaranteed.

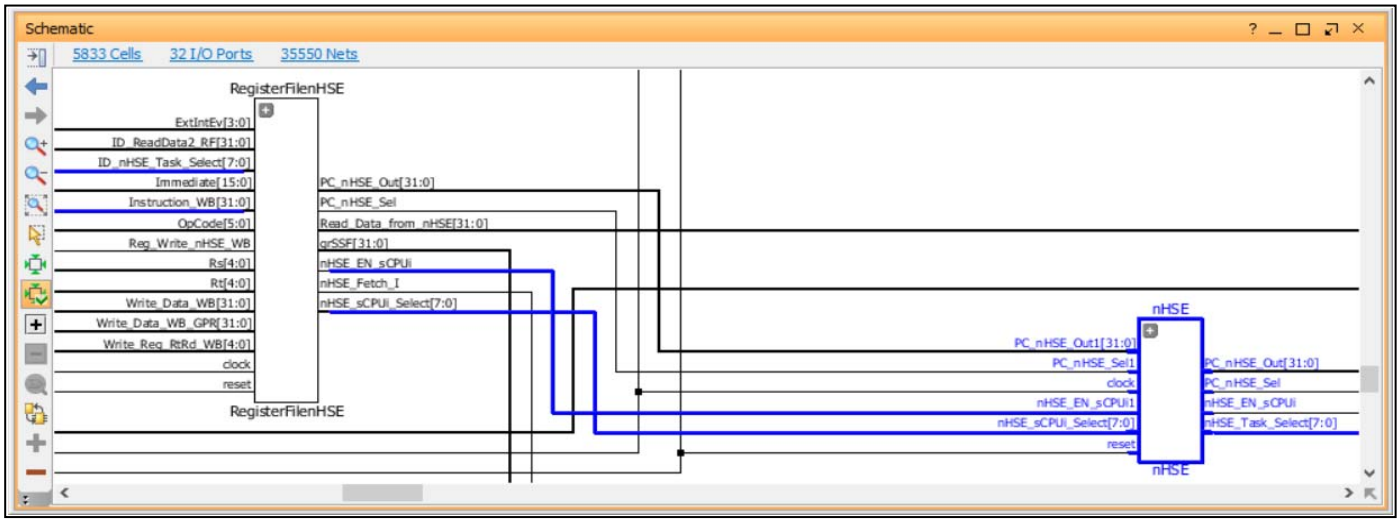

Fig. 2. Register-transfer level (RTL) representation of the RegisterFilenHSE module representing the COP2 Register File and nHSE module for generating the signals *nHSE_EN_sCPUi*[1:0], *nHSE_Task_Select*[7: 0], *PC_nHSE_Out*[31:0] and *PC_nHSE_Sel*.

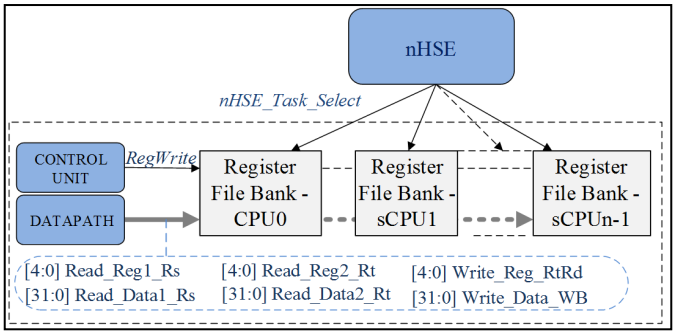

Fig. 3. Select the Registe File Bank for the sCPUi.

In the case of the nHSE scheduler, the activation and deactivation of the interrupt system are possible because they abide by the same execution mode as the tasks. Thus, the nHSE implements a strict prioritization rule, allowing all events to be captured and treated according to the priority of the issuing tasks. The advantage of interrupts that are not attached to tasks is that they are executed in their own Interrupt Service Thread (IST), not requiring context switching and register testing. Because the useful code is executed in the shortest time possible, this method is suitable for high priority interrupts that require a minimum response time, while being resource-intensive. Fig. 3 shows the internal formatting of the GPR (Register File) and its connection with the control unit and with the nHSE integrated hardware scheduler, in the case of the implementation based on the MIPS architecture.

The selection of the bank in the GPR is independent of the operations performed at the level of each semiprocessor. We remind the fact that in the datapath designed at the level of RTL for the nMPRA architecture, the control lines for each bank in the register file are separate. The register file for the nMPRA processor contains *NR_TASKS**32 general purpose registers of 32 bits each, and two ports for reading them, depending on the selected sCPUi.

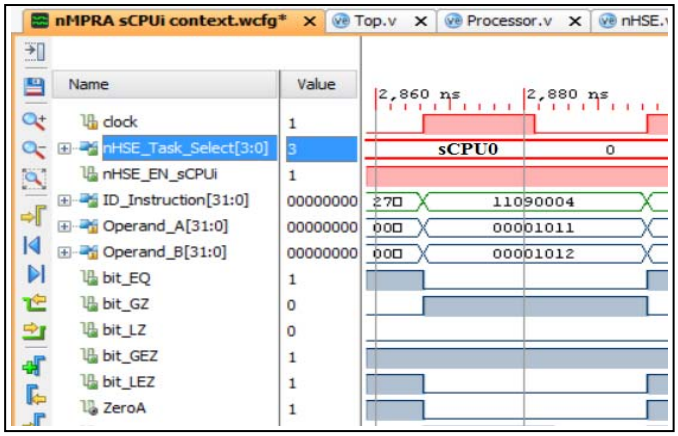

Fig. 4. Signals corresponding to the condition test unit.

Writing in the register file is performed depending on the semiprocessor selected by the scheduler [19]. The *Control_Unit* module (Fig. 3) represents the nMPRA processor control unit. This unit sets the control signals in the datapath for each instruction read from the memory.

Fig. 4 shows the clock signal (*clock* - *CLKOUT0_OUT*) of the nMPRA processor generated with IP Clocking Wizard 5.2 (*Clock_Generator* module). To synchronize with on-chip program memory, the signals used for the read and write operations are modified on the positive edge of the *clock_mem* signal, during implementation and testing of the nMPRA processor using different frequencies for this signal. This nMPRA clock signal (*clock*) has a lower frequency (33MHz), as compared to the memory clock signal (*clock_mem* – 66MHz). Both signals have a 50% filling factor. The frequency is $1/\Delta t$, while the period of the CPU clock is Δt = 30ns. Choosing the appropriate frequency is necessary to ensure a correspondence with the signal propagation time, through the processor logic.

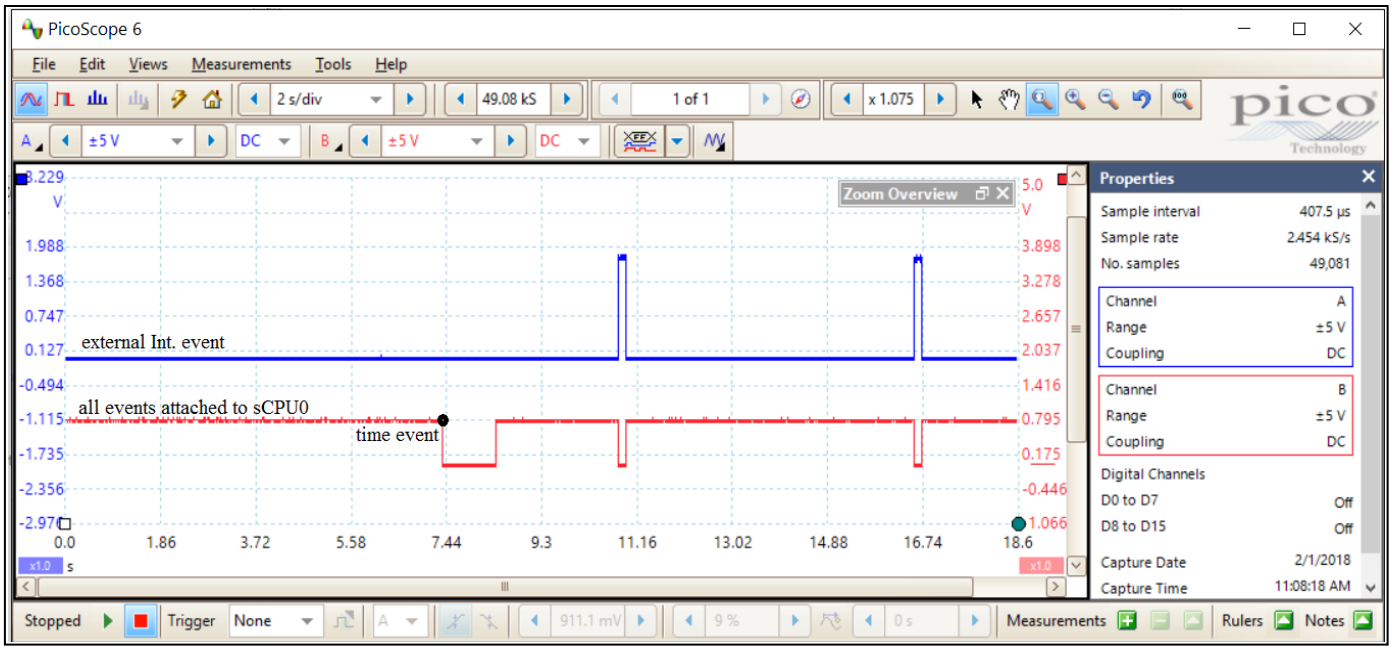

Fig. 5. Input signal corresponding to interrupt INT0 (*Channel A*) attached to sCPU0 semiprocessor and output signal LED[0] (*Channel B*).

On the first part of the waveforms presented in Fig. 4 we can see the processor clock waveform (*clock*), used for task switching in the nHSE block. In the second part of the figure, the inputs and outputs of the condition test unit are presented for two inputs of 32 bits and five output bits relative to the conditions required for the control unit. Also in the ID stage, the value for calculating the jump address, the values provided by these modules (*ID_JumpAddress*, *ID_BranchAddress*) are available for selection at the *PCSrccStd_Mux* multiplexer inputs.

Fig. 5 shows the oscilloscope capture for obtaining the events treated by the sCPU0 semiprocessor. As can be seen, 3 events are captured and treated (*Channel B*), namely a time event (*TEVi*) and two external interrupt events *IntEvi* (*Channel A*). This section describes the operations for testing and validating the datapath as well as the functional units shared by semiprocessors implemented in the nMPRA architecture. Using multiple types of instructions to cover all classes of operations, the CPU was tested by performing multiple Vivado 2016.4 design checks and various instruction sequences. It should be mentioned that MIPS32 does not have overflow detection for the multiplication and division operations. The 32-bit multi-cycle division module implements the multicycle division unit. In the case of logical division, 32 cycles are required to perform this operation in hardware. Since the writing of the result is performed in the HILO register and not in the pipeline, the pipeline can continue without interruption. When a following instruction attempts to access the HILO register, the assembly line will be stopped if the already launched partitioning operation is not yet complete. Detection of access to the HILO register is required because Read After Write (RAW) and Write After Write (WAW) hazards are possible during division operations. Therefore, readings and writings in the HILO register must be blocked, as long as the divider is occupied. To improve execution times, this logic can be placed on a previous pipeline level. The division unit is a finite state machine that can be available or unavailable, with the ability to detect whether overflows occur when signing operations are performed.

As can be seen in Fig. 6, the *nHSE_EN_sCPUi* selector represents the activation command for the sCPUi to which time events, such as deadline, watchdog timer, asynchronous external interrupts, mutex or message events, are attached.

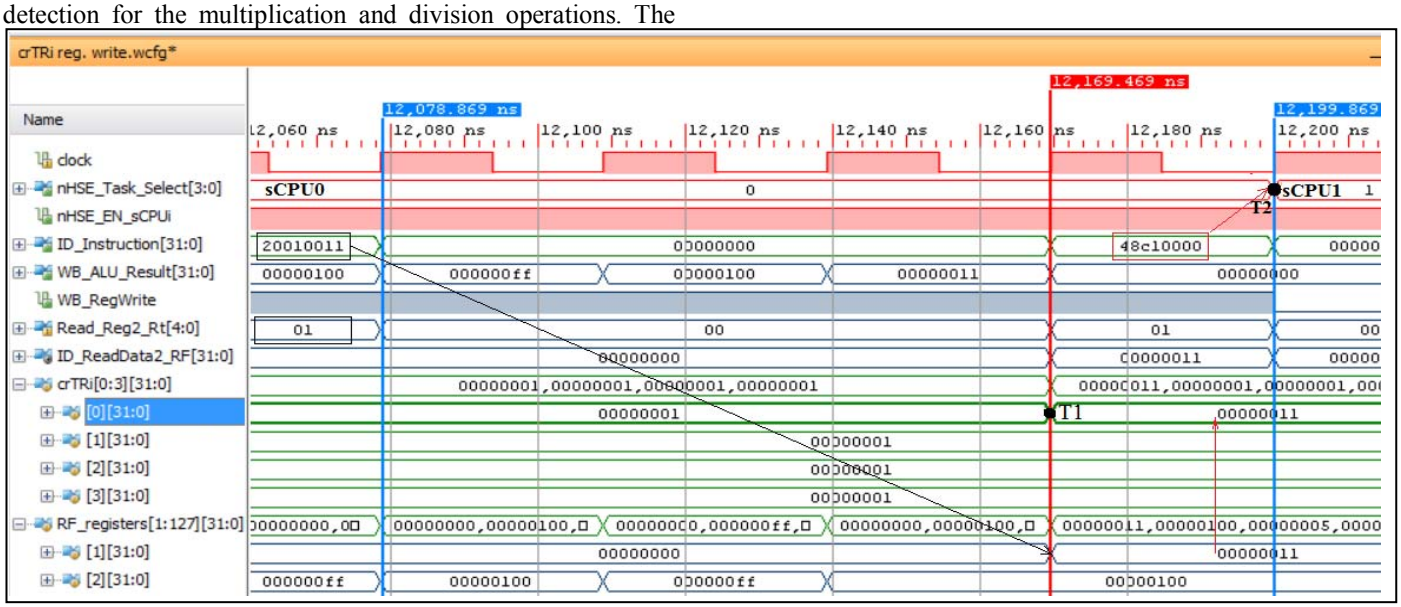

Fig. 6. The sCPU0 and sCPU1 context switching operation based on nHSE scheduler in relation with the preemptive priority scheduling model; *clock* - nMPRA clock; *nHSE_EN_sCPUi* - scheduler enable wire; *nHSE_Task _Select[3:0]* - scheduler task selection; *ID_Instruction[31:0]* – CPU instruction from ID pipeline stage; *WB_ALU_Result[31:0], WB_RegWrite, ReadReg2_Rt[4:0], ID_ReadData2_RF[31:0]* – pipeline assembly line signals and registers, *crTRi[0:3]* – sCPUi control task register; *RF_registers[1:127][31:0]* – Register File multiplied *n* times.

TABLE I. THE APPLICATION SEQUENCE USED TO VALIDATE THE CONTEXT SWITCHING OPERATION BASED ON NHSE SCHEDULER

Application code description	MIPS32 and nHSE instructions transmitted through the *ID_Instruction*[31:0] signals

sCPU0 is executed to threat a time event and execute *wait* instruction in order to dictate a context switch to the sCPU1 (Fig. 6.).	//*sCPU0 run*
	20010011 // 0010 00_00 000_0 0001_ 0000 0000 0001 0001;
	// *addi*, R[rt] = R[rs] + SignExtImm;
	00000000 // nop;
	48C10000 //0100 10_00 110_0 0001_0000000000000000;
	// *movcr* instruction copy r1 GPR (general purpose registers) in the register *crTRi*[sCPUi] COP2;
	// switch the contexts if sCPU0 has no validated and active events
	//context switch: sCPU0->sCPU1 (sCPU1 treats an event validated by *crTRi[1][31:0]* register)
	00000000 // nop;
	20010000 // *addi*, R[rt] = R[rs] + SignExtImm;
	00000000 // nop;

When the scheduler activates the semiprocessors through the *cr0MSTOP* signals, the block diagram decodes at the same time the *mrPRIsCPUi*, *crTRi*, *crEVi* and *grINT_IDi* registers for active sCPUi. At a certain moment, in the case of normal execution, only one sCPUi can be in the RUN state. This is possible through the nHSE scheduler, the task state, the event blocks, as well as the sync signals. The code section illustrated in Fig. 6 through the *ID_Instruction[31:0]* signals, tests the preemptive scheduling method implemented by the nMPRA architecture using the Virtex-7 development kit [20] and Verilog HDL. With an independent execution, the scheduler has entries for multiple events prioritized differently by each sCPUi through the *crEPRi* (Event Priority Register) control register. As can be seen in Fig. 6, at time moment T1, the execution of the CTC2 type instruction (Copy control word to coprocessor 2 (COP2)) *0x48c10000*, determines the change of the *crTRi*[0][31:0] control register with the value *0x00000011*. This value has been previously stored in the file register at *RF_registers[1][31:0]* location, via the *0x20010011* instruction (*addi* MIPS instruction) executed by sCPU0. The hardware solution for dealing with interrupts was implemented as an additional hardware block called the *Priority Encoder Block* that generates the ID of the highest priority interrupt.

The interrupt prioritization scheme has also been extended over events, thus becoming a hardware solution for any new type of event that can be attached to the nHSE architecture for treating the situation when multiple events become active; the following are few types of these events: time interrupts (*lr_Tevi*), watchdog generated events (*lr_WDEvi*), two deadline events (*lr_D1Evi, lr_D2Evi*), interrupts that can be attached to a task *i* (*lr_IntEVi*), mutexes used for accessing shared resources (*lr_MutexEvi*) and inter-task synchronization and communication events (*lr_SynEVi*). *crTRi* is a nHSE control register aimed at validating or inhibiting one of the seven events previously listed. As can be seen in Fig. 6, the multiplication of the memory elements from the pipeline registers greatly contributes to the increase of memory requirements for this architecture. The register file as well as the program counter are the support for the context switching operation in a clock cycle. After processing the *0x48c10000* instruction with the *wait Rj* mnemonic (CTC2 type), at time moment T2, the nHSE scheduler performs context switching between sCPU0 and sCPU1. The *Rj* register can be any of the r0-r31 registers, of the register file belonging to the executing sCPUi. Table I shows the code sequence executed by nMPRA, as can be seen in Fig. 6. The structure of the PC registers, of the trap cells for events and interrupts, together with the mechanism that automatically enables the loading of their treating routines, is based on an additional block implemented in hardware and on the interrupt and event priority encoder.

Fig. 7 presents the resource requirements for implementing the nMPRA processor with 8 sCPUi, including nHSE registers and inter-task synchronization and communication mechanisms. It is worth mentioning that these requirements include the resources used for the on-chip memory, UART, and for Human Machine Interface (HMI). In general terms, a LUT is actually a table that determines how the output is affected by any of the signals present at the inputs. Thus, a LUT consists of a RAM block that is indexed by its inputs.

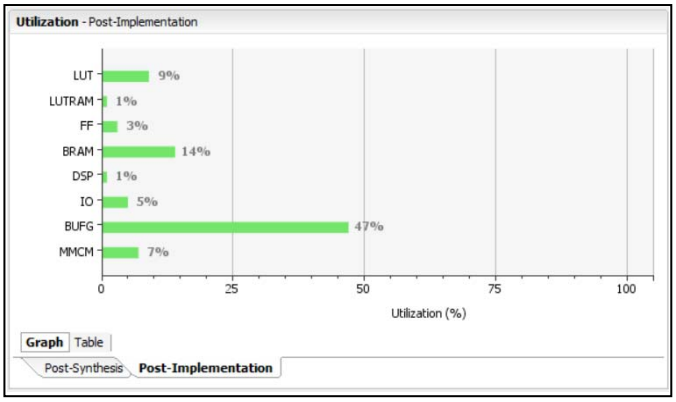

Fig. 7. Hardware resources for implementing nMPRA processor with 8 sCPUi including nHSE registers using xc7vx485tffg1761-2 FPGA circuit; *LUT*– Look-up Table; *LUTRAM* – Look-up Table as Memory; *FF* – Flip Flop; *BRAM* – Block RAM; *DSP* – Digital Signal Processing slices; *IO* –User I/O; *BUFG* – global clock buffers; *MMCM* – Mixed Mode Clock Manager.

The output of a LUT is the value of the RAM location indexed by the inputs. In the context of combinational logic, this is represented by the truth table, which effectively defines how the implemented circuit behaves. For slice definitions, it should be noted that the output of a LUT can optionally be connected to a Flip-Flop, and the groups of LUTs and FFs are referred to as slices.

V. SCHEDULABILITY ANALYSIS OF NHSE SCHEDULER BASED ON REAL-TIME TASKS SCHEDULING ALGORITHMS

This section presents the scheduling models and algorithms that underlie the program segmentation, as well as the optimization of the real time task execution. It also analyzes and describes several models so that the real time nHSE scheduler, validated in the present paper, can implement a scheduling algorithm suitable for both the nMPRA architecture and the set of tasks defined by the real time application. Throughout the present section, we will note with n the number of periodic or sporadic tasks that will be scheduled on a single processor. Each task τ_i is characterized by a WCET noted with C_i, a period T_i and deadline D_i.

A deadline model is fixed, compelling a D_i smaller or equal to T_i. Consequently, for scheduling purposes each task τ_i is assigned a priority P_i, used for selecting which of the ready for execution tasks can be scheduled by the nHSE module. A higher value for P_i (*mrPRIsCPUi* – nHSE register) means a higher priority of that certain task; in fact, tasks are ranked in descending order [21], so that $\forall\ i \mid 1 \leq i < n : P_i > P_i$+1. We consider that activation times are set at the moment of design and initialization of the nMPRA processor, and the task execution time must be equal to or less than C_i. To illustrate these, we propose in Table II a set of three tasks with their relative parameters.

TABLE II. PARAMETERS OF A SAMPLE TASK SET

	C_i	T_i	D_i
τ_1	2	7	6
τ_2	3	12	10
τ_3	7	22	17

A. *Preemptive Scheduling using the Deadline Monotonic Algorithm*

Fig. 8 shows the preemptive scheduling of the tasks set from Table II, using the Deadline Monotonic algorithm. It can be observed that the scheduling of this tasks set is not ideal, because the task τ_3 does not meet its deadline.

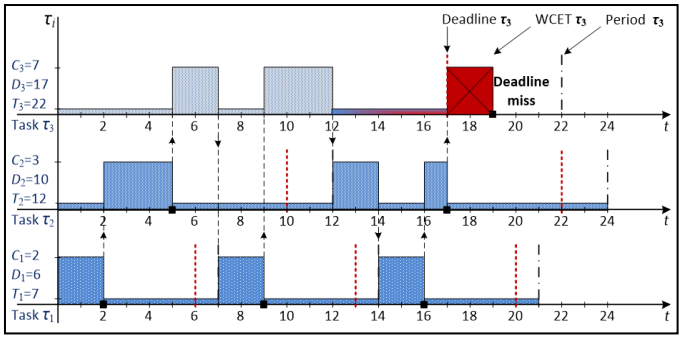

Fig. 8. The preemptive scheduling of tasks from Table II, using Deadline Monotonic algorithm.

B. *Deferred Preemptions Model*

This method implies assigning each task τ_i the longest interval q_i that can be non-preemptively executed. Depending on how many non-preemptive intervals are assigned, this model can be implemented in two different ways: the Floating model and the Activation-triggered model.

The first involves the defining by the programmer of the non-preemptive region through primitives inserted in the task code for activating and deactivating interrupts. The second model is characterized by the fact that the non-preemptive regions are triggered by the occurrence of a higher-priority task [21]. When a request occurs during the non-preemptive period, a timer is set for activating the preemptive mode after q_i interval. In both cases, non-preemptive intervals are not known at the time of the design, and for the sake of analysis, the most unfavorable cases are taken into consideration. For example, Fig. 9 shows the Deadline Monotonic scheduling by using the Deferred Preemptions method for the same set of tasks in Table II. Assuming that $q_1 = 0$, $q_2 = 0$ and $q_3 = 2$, the scheduling of the set of tasks is feasible, the non-preemptive period is observed until the context is switched in the favor of a task with a higher priority.

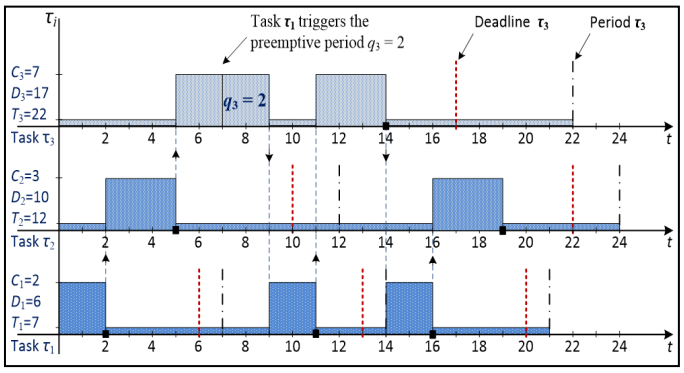

Fig. 9. Deadline Monotonic scheduling with Deferred Preemptions model for the set of tasks in Table II.

When non-preemptive intervals occur, and if a non-preemptive task τ_3 is executed, the task can be interrupted by a task τ_1 that will enter execution after a period q_3 from the moment of occurrence. This interruption can occur only once in the favor of a task with a lower priority [22], [23], where B_i is the longest non-preemptive interval of the tasks with a lower priority than that of the interrupted task (1). The B_i interval can be calculating using the following relation

$$B_i = \max_{j:P_j<P_i}\{q_j - 1\} \quad (1)$$

C. *Task Splitting Model*

If a set of tasks cannot be scheduled in a non-preemptive mode, there are several ways to divide the jobs in subjobs [21], in order to obtain feasible scheduling, if there is one. Therefore, it is very important to identify for each task τ_i the best locations where preemption points may be introduced by the nHSE scheduler, because there are sections, such as I/O operations, critical sections, or short loops, where task interruption is not desirable. In order to simplify this model, we considered the same set of tasks in Table II, assuming that τ_3 is divided in two subjobs of 5 and 2 units. The scheduling performed with the Deadline Monotonic algorithm, using the Task Splitting method proves feasible [24], as shown in Fig. 10. An optimal scheduling scheme, with minimum preemption costs, has to result from the feasibility analysis [25]. To implement this scheduling model with the nMPRA architecture, it was necessary to extend the nHSE module with a new nHSE registers in order to define the preemption points for each task (sCPUi). Based on the feasibility studies for determining the ideal scheduling scheme corresponding to nMPRA real-time architecture, we can draw the following conclusions. Using the Deferred Preemption scheduling model, the number of context switching may be exactly estimated; nevertheless, the points in which these switching occur cannot be established off-line. The Task Splitting cooperative algorithm is the most predictable scheduling mechanism for calculating the preemption points, because it can accurately evaluated both the number of as well as the execution points where context switching occur. The implementation of this scheduling scheme is done by inserting certain system functions in the source code; this involves increasing the jitter by inserting an additional time.

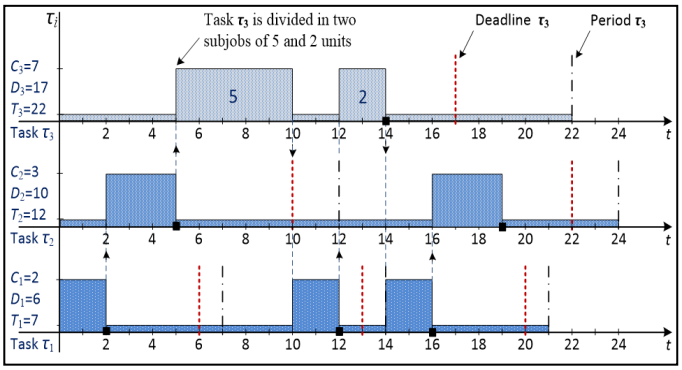

Fig. 10. Deadline Monotonic scheduling with the Task Splitting model for the set of tasks in Table II.

VI. CONCLUSION

The processor described in this research paper is the first version of full CPU implementation of the project described in [2]; furthermore, the author have taken into consideration that any MIPS32 pipeline storage element had to be multiplied as the other multiplied resources (such as the PC, GPR, pipeline registers).

Compared to the original implementation, the processor proposed in this article is compliant with MIPS32 instruction set architecture which enabled us to use coprocessor 2 support for the nHSE module, in accordance with MIPS CorExtend User Defined Instructions (UDI). The final conclusion of this article highlights the advantages of the RTOS hardware scheduling and the use of the nMPRA CPU architecture as support for the research presented in this scientific paper.

ACKNOWLEDGMENT

This paper was supported by the project "Demonstrator experimental de laborator bazat pe nHSE - sistem de operare de timp real integrat în hardware - implementat pe o arhitectură ZScale - RISC V", acronym: nHSE-RiscV, Contract no. 219PED / 2017, PN-III-P2-2.1-PED-2016-1460, PNCDI III, using the infrastructure 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.

REFERENCES

[1] T. T. Phuong, K. Ohishi, Y. Yokokura, and C. Mitsantisuk, "FPGA-based high-performance force control system with friction-free and noisefree force observation," IEEE Trans. Ind. Electron., vol. 61, no. 2, pp. 994-1008, Feb. 2014.

[2] V. G. Gaitan, N. C. Gaitan, and I. Ungurean, "CPU Architecture Based on a Hardware Scheduler and Independent Pipeline Registers," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 9, pp. 1661-1674, Sept. 2015.

[3] I. Zagan, V. G. Gaitan, "Improving the Performances of the nMPRA Processor using a Custom Interrupt Management Scheduling Policy," Advances in Electrical and Computer Engineering, vol. 16, no. 4, pp. 45-50, 2016, doi:10.4316/AECE.2016.04007.

[4] E. E Ciobanu Moisuc, A. B. Larionescu, and V. G. Găitan, "Hardware Event Treating in nMPRA," 2014 International Conference on Development and Application Systems (DAS), May 2014, Suceava, România, pp. 66-69, doi: 10.1109/DAAS.2014.6842429.

[5] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm, "The influence of processor architecture on the design and the results of WCET tools," Proceedings of the IEEE, vol. 91, no. 7, pp. 1038-1054, July 2003, doi: 10.1109/JPROC.2003.814618.

[6] C. Rochange and P. Sainrat, "A time-predictable execution mode for superscalar pipelines with instruction prescheduling," Proceedings of the 2nd conference on Computing frontiers, 2005, pp. 307-314, Ischia, Italy, 04 - 06 May 2005, doi: 10.1145/1062261.1062312.

[7] S. Nordstrom, L. Lindh, L. Johansson, and T. Skoglund, "Application specific real-time microkernel in hardware," Real Time Conference, 14th IEEE-NPSS, 4-10 Jun. 2005. doi: 10.1109/RTC.2005.1547468.

[8] S. Nordstrom, L. Asplund, "Configurable Hardware/Software Support for Single Processor Real-Time Kernels," International Symposium on System-on-Chip, Nov. 2007, doi: 10.1109/ISSOC.2007.4427426.

[9] J. Kreuzinger, R. Marston, T. Ungerer, U. Brinkschulte, and C. Krakowski, "The Komodo project: thread-based event handling supported by a multithreaded Java microcontroller," Proceedings 25th EUROMICRO Conference, vol.2, no., pp. 122-128, 1999, doi: 10.1109/EURMIC.1999.794770.

[10] J. Kreuzinger and T. Ungerer, "Context Switching Techniques for Decoupled Multithreaded Processors," 25th EUROMICRO Conference, Milano, 8-10 Sept. 1999, doi: 10.1109/EURMIC.1999.794476.

[11] Sun Microelectronics, "picoJava-I Microprocessor Core Architecture," Sun Microsystems, Mountain View, California, November 1996, Technical Report WPR-0014-01.

[12] A. El-Haj-Mahmoud, A. S. Al-Zawawi, A. Anantaraman, and E. Rotenberg, "Virtual multiprocessor: an analyzable, highperformance architecture for real-time computing," Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems, San Francisco, California, USA, pp. 213-224, 24 - 27 Sep. 2005, doi:10.1145/1086297.1086326.

[13] A. El-Haj-Mahmoud and E. Rotenberg, "Safely Exploiting Multithreaded Processors to Tolerate Memory Latency in Real-Time Systems," Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems, pp. 2-13, Sep. 2004, Washington DC, USA, doi:10.1145/1023833.1023837.

[14] C. A. Tănase, "An approach of MPRA technique over ARM cache architecture," 2016 International Conference on Development and Application Systems (DAS), Suceava, Romania, 2016, pp. 86-90. doi: 10.1109/DAAS.2016.7492553.

[15] E. E. Ciobanu, "The Events Priority in the nMPRA and Consumption of Resources Analysis on the FPGA, " Advances in Electrical and Computer Engineering, vol. 18, no. 1, pp. 137-144, 2018, doi:10.4316/AECE.2018.01017.

[16] D. A. Patterson, J. L. Hennessy, "Computer Organization and Design MIPS Edition: The Hardware/Software Interface," 5th Edition, Sep. 2013, ISBN: 9780124077263.

[17] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design, Revised Fourth Edition: The Hardware-Software Interface," Fourth Edition, 2011.

[18] http://opencores.org/project,mips32r1, (Accessed: 12-09-2015).

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

[20] VC707 Evaluation Board for the Virtex-7 FPGA, User Guide, XILINX, UG885 (v1.7.1) 12 August, 2016 https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf, (Accesat: Feb. 2017).

[21] G. C. Buttazzo, "Hard Real-Time Computing Systems", Predictable Scheduling Algorithms and Applications, Third edition, 2011.

[22] S. K. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in Proc. of the 17th Euromicro Conf. on Real-Time Systems (ECRTS'05), pp. 137-144, Palma de Mallorca, Balearic Islands, Spain, 6-8 July, 2005.

[23] G. Yao, G. C. Buttazzo, and M. Bertogna, "Bounding the maximum length of non-preemptive regions under fixed priority scheduling," In Proc. of the 15th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'09), pp. 351-360, Beijing, China, August 24-26, 2009.

[24] A. Burns, "Preemptive priority based scheduling: An appropriate engineering approach," S. Son, editor, Advances in Real-Time Systems, pp. 225–248, 1994.

[25] G. Yao, G. C. Buttazzo, and M. Bertogna, "Feasibility analysis under fixed priority scheduling with fixed preemption points," In Proc. of the 16th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'10), pp. 71-80, Macau, SAR, China, 23-25August, 2010.