International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:11, No:5, 2017

Improving the Performances of the nMPRA Architecture by Implementing Specific Functions in Hardware

Ionel Zagan, Vasile Gheorghita Gaitan

***Abstract*—**Minimizing the response time to asynchronous events in a real-time system is an important factor in increasing the speed of response and an interesting concept in designing equipment fast enough for the most demanding applications. The present article will present the results regarding the validation of the nMPRA (Multi Pipeline Register Architecture) architecture using the FPGA Virtex-7 circuit. The nMPRA concept is a hardware processor with the scheduler implemented at the processor level; this is done without affecting a possible bus communication, as is the case with the other CPU solutions. The implementation of static or dynamic scheduling operations in hardware and the improvement of handling interrupts and events by the real-time executive described in the present article represent a key solution for eliminating the overhead of the operating system functions. The nMPRA processor is capable of executing a preemptive scheduling, using various algorithms without a software scheduler. Therefore, we have also presented various scheduling methods and algorithms used in scheduling the real-time tasks.

***Keywords*—**nMPRA architecture, pipeline processor, preemptive scheduling, real-time system.

I. Introduction

THE question of whether preemptive systems are better than non-preemptive systems has been addressed for a long time. Field literature has provided partial solutions, but some issues like nondeterministic performance, scheduling cost and inefficient power consumption are still under discussion. Each of these solutions comes with its own advantages and disadvantages, depending on the predictability and efficiency of the system for which they have been implemented [1].

The following aspects have to be taken into account when performing an analysis of operating systems [2]; including the scheduler we are dealing with:

1) 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).
2) In non-preemptive scheduling, the problems generated by mutual exclusion are insignificant because the nature of the scheduling algorithm guarantees the exclusive access to shared resources. However, in preemptive scheduling, the implementation of complex protocols specific to the control mechanisms of shared resources is necessary in order to guarantee access to the shared resources and avoid priority inversion.
3) In hard real-time systems with non-preemptive scheduling, the jitter effect is at a minimum for all system tasks; this way, the control techniques for compensating and diminishing the negative effects of delays are simplified.
4) The non-preemptive implementation enables the use of stack sharing techniques, in order to save memory space for small embedded systems.

To discover and further pursue the research directions in the field of single-core and multi-core SoC CPU architecture, one needs to know if in doing so, the single-core architectures can be optimized in order to obtain maximum efficiency in real-time applications, as well as in those with low power consumption. Thus, using the CPU with a superior utilization factor, the predictable and deterministic control of a process specific to a real-time system (RTS) can be ensured.

This paper provides a schedulability analysis of the already existing scheduling algorithms and detailed description of the experimental results obtained during the tests performed on the nMPRA CPU architecture. The hardware implementation of schedulers as coprocessors represents a novelty for real-time systems and a true challenge in the field. The following issues are also taken in consideration: aspects characteristic to embedded real-time system, ensuring deterministic and predictable control of a process, and the real-time operating system (RTOS) characteristics and scheduling algorithms used in critical applications.

The nMPRA architecture can be successfully used in small applications for critical real-time and mixed-criticality systems. This implementation includes an integrated hardware scheduler called nHSE (Hardware Scheduler Engine for *n* tasks) controlled via its dedicated instructions [3], [4]. Tasks context switching is based on remapping the multiplied resources, such as Program Counter, Register File and Pipeline Registers [5]. The project has been implemented using Vivado 2015.4 design environment and the source code has been written in Verilog HDL.

This article is structured as follows: after a brief introduction in Section I. Section II presents a few models for

Ionel Zagan and Vasile Gheorghita Gaitan are with the Stefan cel Mare University of Suceava, Suceava, Romania and 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 (e-mail: zagan@eed.usv.ro, gaitan@eed.usv.ro).

the scheduling of real-time tasks, and Section III describes briefly the nMPRA architecture. Section IV describes the validation of the nMPRA architecture with the support of the static nHSE scheduler by submitting the waveforms characteristic to tasks context switching using the Virtex-7 development kit (subject of this article and the novelty for the proposed architecture); Section V presents related work and finally, Section VI adds the conclusions and directions for future research.

II. Non-Preemptive and Preemptive Tasks Scheduling

The present section will describe various algorithms for real-time task scheduling. Taking into account the restrictions for each set of tasks, each algorithm represents a scheduling solution. The implementation of these scheduling algorithms in hardware, increases admissibly the overall processor throughput, mainly because nMPRA 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. Throughout the present paper, each task τ_i is characterized by a WCET noted with C_i, a deadline D_i and period T_i. A deadline model is defined, compelling a D_i smaller or equal to T_i. 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; a higher value for P_i means a higher priority of that certain task.

A. Non-Preemptive Scheduling

By using the non-preemptive scheduling method, all context switching is eliminated, and, moreover, the architecture related cost coefficient decreases [1]. Under these circumstances, each task τ_i can be blocked for a period of time equal to B_i, representing the longest execution time of tasks with lower priority.

The reduction with one unit is necessary, because the new task has to be executed sooner with at least one unit. Taking into account (1), for a certain set of tasks, the most affected are the ones with high priority.

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

A feasibility study for a non-preemptive set of tasks proves difficult to perform, because it requires an analysis on a longer period of time. Bril et al. [6] proved that in non-preemptive scheduling, the WCET of a task may not appear in the first part of the execution.

Because the execution of high priority tasks is delayed, there is a scheduling anomaly called self-pushing phenomenon that does not allow meeting the established deadlines. Therefore, an analysis for a longer period of execution, called Li (Level-i Active Period defined in [6]), is necessary, at least until task τ_i, with priority P_i, completes execution.

Yao et al. showed in [7] that the analysis of non-preemptive tasks can be reduced to a single job, subject to the following conditions:

1. The task set τ_i is feasible under preemptive scheduling.
2. The relative deadlines D_i are lower than or equal to periods T_i.

Fig. 1 shows the scheduling without interrupts performed by the Deadline Monotonic algorithm for the set of tasks in Table I. It was noticed that τ_3 manages to meet the deadline, although the set of tasks cannot yet be scheduled, because τ_1 does not meet the conditions. Therefore, this set of tasks cannot be scheduled in a non-preemptive mode with none of the Rate Monotonic algorithms (RM) or Earliest Deadline First (EDF). Nevertheless, this scheduling scheme can be successfully used for those sets of tasks that have little use for the calculating unit.

TABLE I
Assigning the Preemption Thresholds

	C_i	T_i	D_i	P_i	θ_i
τ_1	2	7	6	3	3
τ_2	3	12	10	2	3
τ_3	7	22	17	1	2

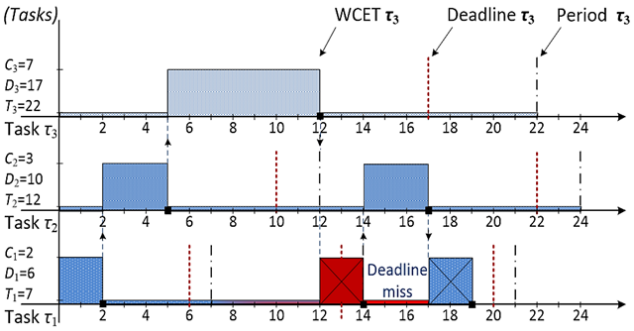

Fig. 1 The non-preemptive scheduling of tasks in Table I using the Deadline Monotonic algorithm

A main disadvantage of non-preemptive implementations is that it introduces an additional blocking factor for high priority tasks; nevertheless, there are many important advantages for adopting this type of scheduling.

B. Preemptive Scheduling

Preemptive schedulers introduce fluctuations for tasks execution times, reducing thus the predictability of the system. In the process of designing these types of schedulers, one has to take into consideration certain input costs introduced by [1]:

1) Scheduling – represents the time allocated to the scheduling algorithm.
2) Pipeline – sums up the clock cycles lost by instructions that have already been extracted and decoded, because the assembly line has to receive the instructions of the new task [8]; the time necessary for introducing the new task on the assembly line; the time needed to restore the assembly line for the interrupted task, when it resumes execution.
3) Cache-related – represents the time necessary for loading the cache line lost at the moment of context switching.
4) Bus-related – represents the time cycle introduced by the

operations of accessing the RAM memory, due to the cache miss effect.

The total sum of these times, or only part of them, represents the Architecture related cost that is significantly variable, depending on the context switching points. In analyzing the scheduling algorithms, one needs to take into account certain issues, such as the complexity of implementation, the effectiveness of the scheduling scheme [9], and the predictability of estimating the cost coefficient of the architecture.

The ***Preemption Thresholds model*** was first proposed by Wang and Saksena in [10]. According to this approach, each task τ_i is assigned a normal priority P_i and a preemption threshold $\theta_i \geq P_i$; the task may disable the preemptive system up to a specific preemptive threshold θ_i. Therefore, a context switch can only take place if the priority of the new task P_j is higher than the preemptive threshold θ_i of the task τ_i. This scheduling method represents a compromise between full preemptive and full non-preemptive scheduling. It is a normal situation because, if each priority threshold is considered equal to the priority of the task, the scheduler acts as a full preemptive; instead, if all priority thresholds are set as the maximum priority of the system, the scheduling algorithm becomes non-preemptive [1]. The preemption threshold is used in order to increase the priority of the task τ_i during execution. Even if task τ_i is interrupted by a different task with a higher priority, the priority of the task will remain the same. At the moment of activation, the priority of the task is the same as its nominal priority P_i; the task is inserted into the ready queue and waits until all tasks with higher priority $P_h > P_i$ are executed. At the time of execution, the task τ_i is assigned the priority θ_i and can only be interrupted by tasks τ_h with a higher priority $P_h > \theta_i$. Therefore, after completing execution, the priority of the task returns to its nominal value P_i.

Wang and Saksena proved that by appropriately setting the priority threshold, a good efficiency for the scheduling scheme and higher degree of CPU utilization can be achieved [10]. For example, by assigning the preemption thresholds θ_i for a set of tasks in Table I and using the Deadline Monotonic algorithm, a satisfying scheduling can be obtained. Thus, as can be seen in Fig. 2, a set of tasks, impossible to schedule with non-preemptive scheduling algorithms, can be successfully scheduled using the preemption threshold method.

One can notice that at the time t = 7, τ_1 can interrupt τ_3 because $P_1 > \theta_3$, and at the moment t = 12, τ_2 cannot interrupt τ_3 because $P_2 = \theta_3$. Because $P_1 > P_2$, task τ_1 is executed at the moment t = 14, even if the tasks τ_1 and τ_2 are in the READY state and do not yet have the preemptive thresholds activated.

According to the ***Task Splitting model***, a task τ_i is executed in the non-preemptive mode, and preemptions are allowed only in predefined points, called preemption points. Task τ_i is divided in m_i non-preemptive subjobs by certain well defined algorithms, resulting m_{i-1} preemptive points.

If a task with high priority reaches the READY state between two preemption points, the interruption of the current task will occur at the next preemption point [11].

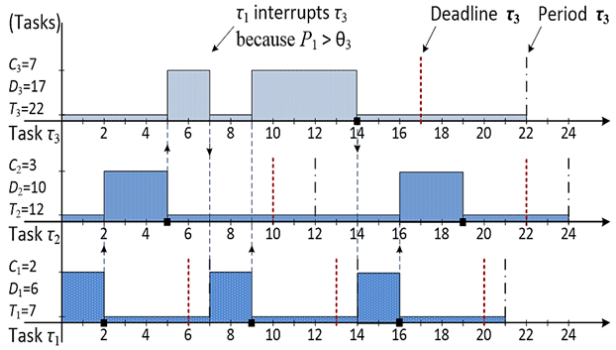

Fig. 2 Deadline Monotonic scheduling of the set of tasks in Table I with preemptive thresholds θ_i

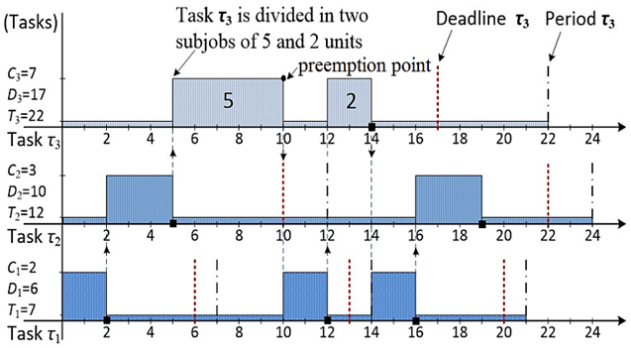

Fig. 3 Deadline Monotonic scheduling with the task splitting model for the set of tasks in Table I

Assuming that all jobs scheduled for a certain task have the same subjob division [1], and for each subjob k_{th} there is a WCET denoted by $q_{i,k}$, the WCET C_i can be defined in (2).

$$C_i = \sum_{k=1}^{m_i} q_{i,k} \tag{2}$$

In order to obtain an optimal scheduling for this model, the following parameters are important for every task: C_i, D_i, T_i, q_i^{max}, q_i^{last}, where the last two are defined in (3).

$$\begin{cases} q_i^{max} = \max\limits_{k\in[1,m_i]} \{q_{i,k}\} \\ q_i^{last} = q_{i,m_i} \end{cases} \tag{3}$$

The feasibility of a high priority task τ_k is affected by the length q_j^{max} of the longest subjob of every task τ_j with priority $P_j < P_k$. The response time is also influenced by the size q_i^{last} of the final subjob τ_i.

The feasibility analysis of a task set scheduled using the task splitting method can be performed in a similar manner, by using per-existing models and taking into account the following two differences. The first difference is represented by the period of the blocking factor B_i for each task with a lower priority than τ_i (4); the second difference is represented by the last non-preemptive period q_i^{last} of the task τ_i.

$$B_i = \max_{j:P_j<P_i} \{q_j^{max} - 1\} \quad (4)$$

Using the set of tasks in Table I and assuming that τ_3 is divided in two subjobs of five and two units, the scheduling performed using the task splitting method proves feasible, as shown in Fig. 3.

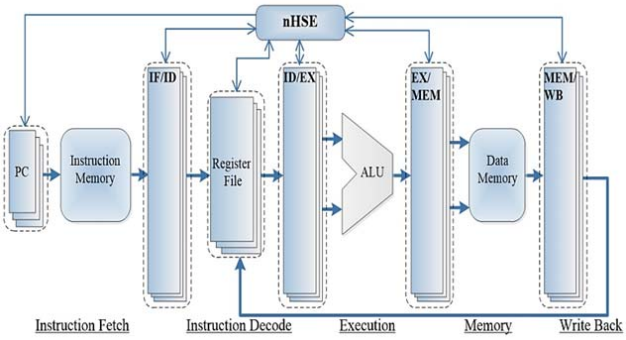

Fig. 4 Replication of resources of the nMPRA architecture [12]. PC-program counter, IF/ID-Instruction Fetch/Instruction Decode, ID/EX-Instruction Decode/EXecute, EX/MEM-EXecute/MEMory, MEM/WB-MEMory/Write Back stage

III. The nMPRA Architecture

The nMPRA architecture is based on remapping the multiplexed resources. So, an instance of the CPU will be called semi CPU (sCPUi for task *i*). Such a hardware instance includes its own PC register, general purpose registers, and pipeline registers. All sCPUi share the functional units of the nMPRA processor, such as the control unit, the logic and arithmetic unit, the hazard detection and data redirection unit [12]. The nHSE unit is a finite state machine, performing the static or dynamic scheduling of tasks, with inputs such as interrupts, deadlines, timers, mutexes or messages. The static scheduler is preemptive with static priorities. The dynamic scheduler provides the possibility to set the priority for each sCPUi which is deactivated at reset; in this case, only the sCPU0 remains active. The priority of a sCPUi can be changed dynamically by a dynamic scheduling algorithm, implemented either in software, at the sCPU0 level, or in hardware. Depending on the selected task *i* that runs on sCPUi, the nHSE scheduler manages the selection of the PC register and of the bank from the register file in the same way as the pipeline registers and any other storage element present in the pipeline. In the process of task context switching, the scheduler remaps these multiplied resources in order to restore the internal state of the data path and of the selected sCPUi control signals, as shown in Fig. 4.

The CPU implements the MIPS instruction set [13], adding additional instructions for the integrated scheduler nHSE.

This paper presents the results that demonstrate the implementation of task context switching operation using the nMPRA architecture and static nHSE defined in [4]. The static nHSE scheduler implements the Task Splitting method considering an nMPRA version with four sCPUi. This represents the novelty brought by the present article.

In the next section, we presented and described experimental results obtained from the practical implementation of this solution, and the benefits it brings compared to traditional processors.

IV. The Validation of the nMPRA Processor Using Virtex-7 Platform

This section demonstrates the functionality of the nHSE scheduler by validating the context switching performed by the Task Splitting algorithm using the FPGA xc7vx485tffg1761-2 circuit. To implement this scheduling model, it was necessary to extend the nHSE unit with a new configuration register named *grPrPointTS[0:3][31:0]* in order to define the preemption points for each task. This CPU architecture with five pipeline stages has been designed and implemented using the VC707 Evaluation Kit [14]. The nMPRA implementation is especially designed for minimizing the overhead generated by classical software schedulers for reducing the jitter effect and for eliminating the unpredictability in the case of handling asynchronous interrupts. During a clock cycle of the five stages pipeline nMPRA processor, the data stored in pipeline registers is processed by the functional units of the stage in question; the results are stored in the following pipeline registers or written in the specific bank of the Register File, as shown in Fig. 5.

For testing the nMPRA architecture with 4 sCPUi running at a frequency of 33MHz, it was necessary to synthesize and implement a SoC designed on the Virtex-7 FPGA VC707 Evaluation Kit produced by Xilinx.

For validating the MIPS instructions implemented by the nMPRA processor, the waveforms obtained from simulation and those acquired as a result of on-chip debugging with the ChipScope Analyzer have been pursued. The implementation is based on the project described in [15], a 32-bit MIPS processor which aims for conformance with the MIPS32 Release 1 ISA. The practical results presented in this section demonstrate the validity of the theoretical approach described in the previous chapter, so that the characteristics of the waveforms obtained during simulation correspond to the ones captured with the ChipScope Analyzer.

In order for the processor to interact with input/output ports, their mapping has been performed in the workspace of the data memory. Thus the ports corresponding to the UART communication and to the LCD screen can be accessed, as well as the digital inputs and outputs. In order to allow connections to a PC using the USB port, the development kit also contains a bridge Silicon Labs CP2103GM USB-to-UART (U44) device [14]. In the case of implementing the current SoC, via UART communication, the program instructions are transmitted from a PC to FPGA on-chip memory implemented with the IP Core Block Memory Generator, version 8.3.

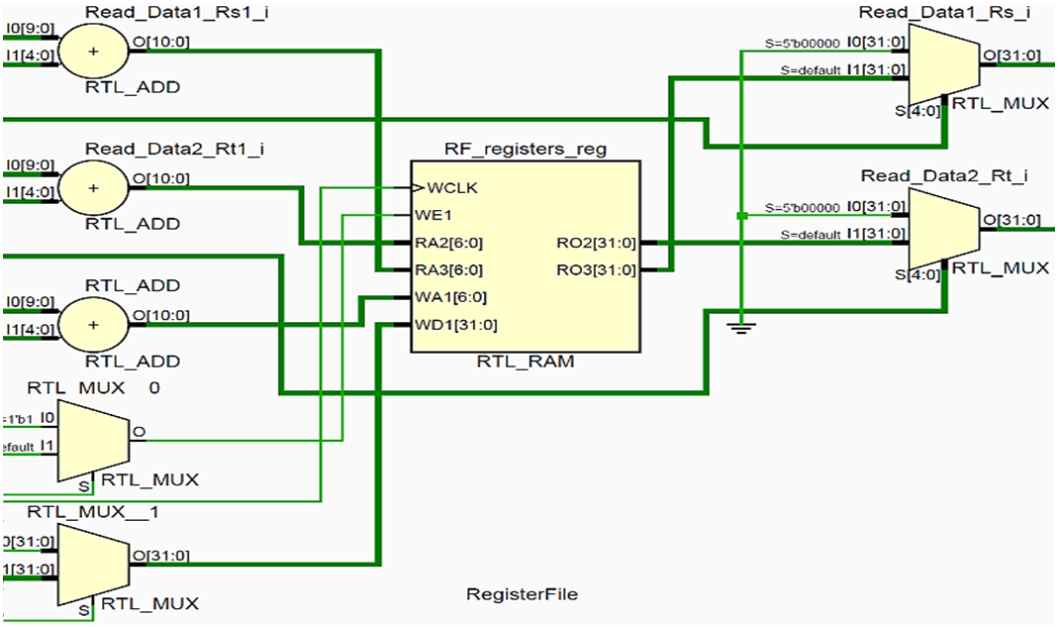

Fig. 5 Multiplying the Register File of the nMPRA architecture in a RTL representation

As for the boot procedure, Fig. 6 shows the waveforms specific to UART communication implemented by the hardware driver, capable of receiving and sending data with a predetermined transfer rate. Thus, using the ChipScope Analyzer it is possible to view and inspect the contents of the registers used for receiving MIPS instructions. For the FPGA circuit to receive every bit, including the start and stop bit, the oversampling mechanism has been used; therefore, the succession of bits for receiving the 0x58 (bin 01011000) byte could be observed. To do this, and considering the CPU's working frequency of 33 MHz and a UART frequency of 115.2kHz, a clock signal multiplied 16 times (*uart_tick_16x*) in relation to the clock signal used for the UART communication (*uart_tick*) was needed [15].

In order to test the access of an FPGA pin which commands an LED on the development platform, the signals of the memory data in the address space have also been mapped. In this case, the program (store instruction) performed a simple switch of a pin configured in the .xdc constraint file.

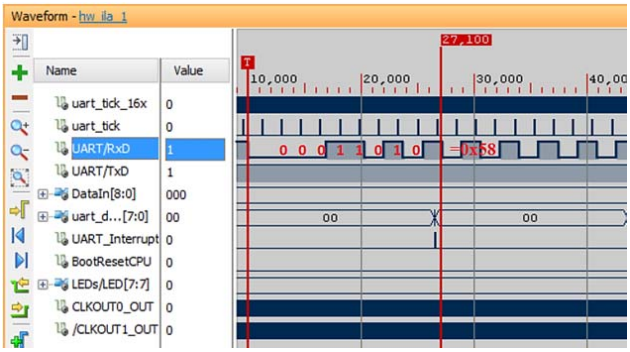

Fig. 6 Receiving the data through the RxD line

As shown in Fig. 7, waveforms are used to check the nHSE capacity to maintain the task contexts and to perform contexts switches within a time frame characteristic to real-time systems. The nMPRA architecture guarantees the execution of the new scheduled task starting with the next clock cycle, as we can see in Fig. 7, at the moment T6. Context remapping occurs after the non-preemptive subjob of sCPU2, if the Task Splitting preemptive scheduling algorithm implemented by nHSE performs a tasks context switching dictated through the nHSE_Task_Select[3:0] selector. This signal, along with those referenced in the following description, can be found in Fig. 7. Assuming that task 2 executed on sCPU2 semi processor is divided in two non-preemptive subjobs (14 and 6 clock cycles) by a certain well defined algorithm, one preemptive point will result, indicated by the T6 moment. The operation may, however, be delayed up to three clock cycles in case it is desirable that the active sCPUi completes the execution of the *sw* instruction, already present on the stages of the pipeline assembly line [16]. This instruction can be used both for inter-task synchronization and communication mechanisms and for accessing mapped ports in address spaces. 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 [4]. In comparison to the theoretical version, in the CPU validation version, two clock signals have been used, one for the pipeline registers (the internal logic of the scheduler) and for handling asynchronous external interrupts, and one for data and instruction memory. In order to synchronize with the program memory implemented on-chip, the *clock_mem* clock signal dedicated to memory runs at a high CPU frequency, and the signals for reading MIPS instructions from memory are modified on both fronts of the *clock_mem* clock.

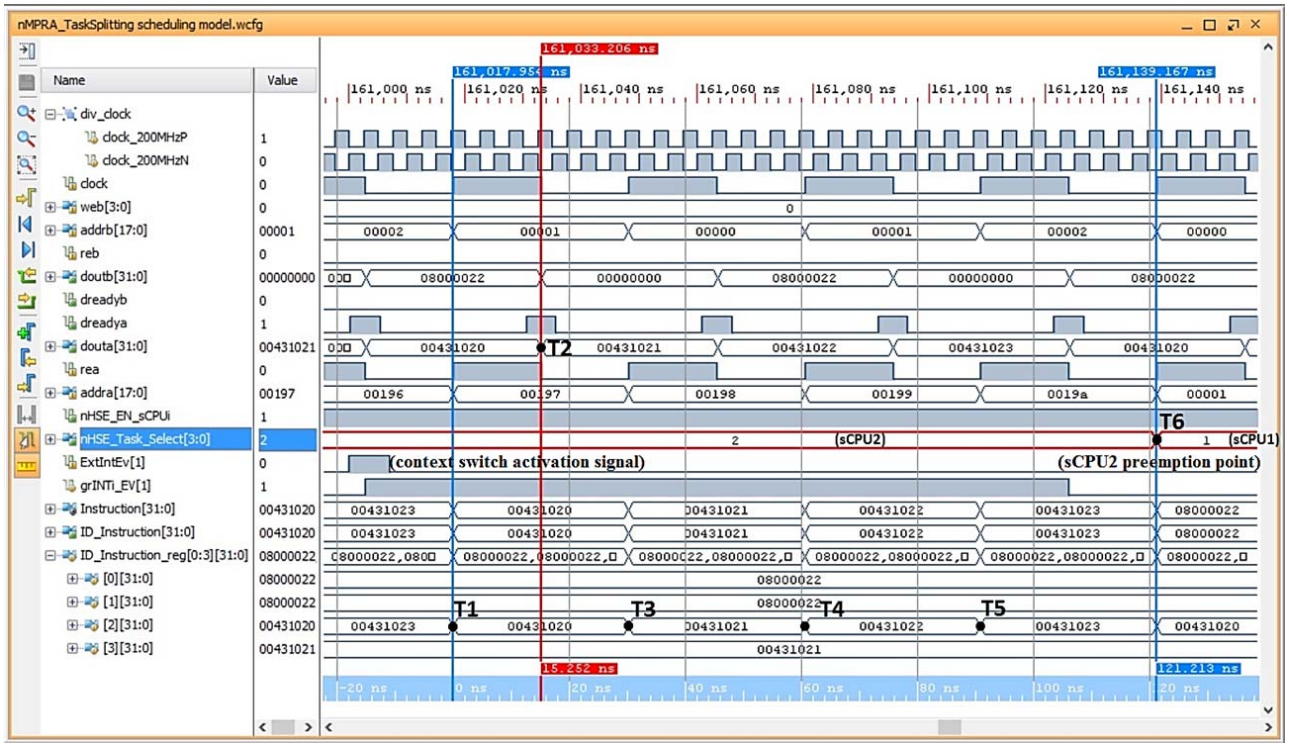

Fig. 7 The sCPU2 and sCPU1 context switching operation based on Task Splitting model in relation with the assigned activation signal *ExtIntEv[1]*; *clock_200MHzP, clock_200MHzN* - 200MHz differential signal clock; *clock* - nMPRA clock; *addra, addrb* - memory addresses; *rea, wea, reb, web* - Read/Write operation request; *dreadya, dreadyb* - data ready signals; *nHSE_EN_sCPUi* - nHSE enable signal; *nHSE_Task_Select[3:0]* - nHSE task selector; *ID_Instruction[31:0]* - wire type instruction; *ID_Instruction_reg[0:3][31:0]* - reg type sCPUi instruction

Fig. 7 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).

The waveforms corresponding to the Instruction[31:0], ID_Instruction[31:0] and nHSE_Task_Select[3:0] signals are also represented. The latter selects the PC, the bank from the Register File and the pipeline registers corresponding to each sCPUi. The *addra* and *addrb* signals are outputs of the memory controller that indicates the memory addresses accessed by the next transfer. These addresses are valid only when the *rea*, *wea*, *reb* and *web* signals are set to logic value 1. All operation on the data bus are synchronous with the CPU clock, the *dreadya* and *dreadyb* signals representing CPU inputs that indicate the completion of the current transfer; the following transfer can thus begin once with the next 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_reg[0:3][31:0] pipeline register containing, at a certain moment, the code for the instructions extracted for each sCPUi. At the T1 moment, the ID_Instruction_reg[2][31:0] register contains the 0x00431020 instruction, and at the T2 moment, the 0x00431021 instruction is extracted from memory from the 0x00197 address (*addra*) and sent to the Instruction Fetch/Instruction Decode pipeline register via Instruction[31:0] signals. Thus, the instruction is stored in the ID_Instruction_reg[i][31:0] register, where *i* is the sCPUi selected by the nHSE. We can observe how the ID_Instruction[31:0] signals transmit data from the ID_Instruction_reg[2][31:0] register, the ID_Instruction[31:0] pipeline output being wire type, not reg. This output is modified at the rising edge of the clock signal in connection to the nHSE_Task_Select[3:0] signals, the following instruction being retrieved at T3, T4 and T5 moments from the ID_Instruction_reg[2][31:0] register. The content of the ID_Instruction_reg[0][31:0] and ID_Instruction_reg[3][31:0] registers remains unchanged during simulation, because sCPU0 and sCPU3 are not selected for execution by the Task Splitting preemptive scheduling. Under these circumstances, the predictability of the CPU results from the outstanding performances obtained from context switching, handling external interrupts and from the simplicity of the architecture. 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.

V. Related Work

This chapter presents a brief description of a predictable processor architecture and a dynamic scheduling algorithm, which can be compared with the results presented in this paper using the nMPRA processor and the nHSE scheduler.

Kotecha et al. propose an innovative scheduling algorithm designed for RTOS, called Adaptive Scheduling Algorithm [17]. The proposed solution represents a scheduling algorithm based on a combination of EDF and Ant Colony Optimization (ACO). The authors present this dynamic algorithm as a real solution that could be used successfully in embedded systems, even in real-time applications. The solution is ideal in terms of Success Ratio and Effective CPU Utilization, obtaining good results in both underloaded and overloaded conditions. Presenting the measured execution time taken by each scheduling algorithms, the authors claim to get better performance criteria than for existing traditional algorithms. The aim of this project is to ensure the scheduling performance of periodic tasks in the preemptive mode in a single processor environment. The performed analysis and experiments reveal that the proposed algorithm is both fast and very efficient, because it can switch automatically between the EDF algorithm and the ACO based scheduling algorithm.

The MSparc architecture presented in [18] is a multithreaded processor based on block multithreading, designed to support architectural requirements for real-time systems. The proposed multithreaded processor is based on the SPARC standard, adapted to meet the system requirements. In order to provide the real time response, guaranteed by a minimal jitter, the authors choose to move the Round Robin scheduling algorithm from software to hardware. The main reason for implementing the MSparc project is to improve the reaction time for events with hard real-time constraints, preserving the predictable behavior.

VI. Conclusion and Future Work

The high performances obtained by the Task Scheduling algorithm implemented in hardware and the use of a particular processor architecture named nMPRA are the elements of originality and innovation that the present paper brings to the current state of research.

The Preemption Thresholds scheduling model can reduce the number of context switching, although the preemption cost represents a disadvantage; this cost is not easily estimated, because the number of context switching for every task cannot be accurately calculated. The Task Splitting cooperative scheduling model is the most predictable mechanism for estimating the preemption costs, because it can accurately estimate both the number of as well as the points (defined by the new nHSE registers *grPrPointTS[0:3][31:0]*), where context switching occurs.

By implementing the Task Splitting scheduling method in the hardware and with the support of the static nHSE scheduler, this project demonstrates the importance of dividing the execution of an instruction in stages and shows how CPU clock cycles can be saved using various scheduling models. Moreover, the time needed for context switching can be reduced by implementing a scheduler in hardware and by multiplying resources from the nMPRA architecture.

Due to the large dimensions of the project and the many connection wires and various interconnected modules, pipeline processors are difficult to design. To comply with time limitations, it was necessary for the data to be read and modified at the same period clock. Furthermore, it was very important to decide which of the registers are registered types and which components are clocked.

The performances and stability of the nMPRA architecture can be improved by designing a cache memory for optional data and by implementing a memory protection unit (MPU) for the hard real-time tasks, focusing on reducing the operating system overhead.

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.

References

[1] G. C. Buttazzo, "Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications," Third edition, Springer, 2011.
[2] W. Stallings, "Computer Organization and Architecture," 10th Edition, 2015.
[3] 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, IN, USA, pp. 1-5, May 2012.
[4] 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.
[5] I. Zagan, "Improving the performance of CPU architectures by reducing the Operating System overhead," in *The 3rd IEEE Workshop on Advances in Information, Electronic and Electrical Engineering AIEEE'2015*, pp. 1-6, 13-14 Nov. 2015, Riga, Latvia.
[6] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," in *Real-Time System*, pp. 63–119, 2009.
[7] G. Yao, G. C. Buttazzo, and M. Bertogna, "Feasibility Analysis under Fixed Priority Scheduling with Fixed Preemption Points," in *IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 71-80, Aug. 2010.
[8] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design, Revised Fourth Edition: The Hardware-Software Interface," Fourth Edition, 2011.
[9] M. Hwang, D. Choi, and P. Kim, "Least Slack Time Rate First: an Efficient Scheduling Algorithm for Pervasive Computing Environment," in *Journal of Universal Computer Science*, vol. 17, no. 6, pp. 912-925, 2011.
[10] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*, pp. 328-335, 1999.
[11] A. Burns, "Preemptive priority-based scheduling: an appropriate engineering approach," in *Advances in real-time systems*, pp. 225-248, 1995.
[12] I. Zagan, "Real-time evaluation of nMPRA CPU Architecture based on Multithreaded Execution,"in *8th International Conference on Computer Science and Information Technology (ICCSIT 2015)*, 10 - 11 Dec. 2015, Amsterdam, Netherlands.
[13] "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: 10-05-2016).
[14] www.xilinx.com/support/documentation/boards_and.../ug885_VC707_Eval_Bd.pdf, (Accessed: 17-08-2016).
[15] http://opencores.org/project,mips32r1, (Accessed: 12-09-2015).

[16] 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.
[17] K. Kotecha and A. Shah, "Adaptive scheduling algorithm for real-time operating system," in *IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 2109-2112, Jun. 2008.
[18] A. Metzner and J. Niehaus, "MSparc: Multithreading in Real-Time Architectures," in *Journal of Universal Computer Science*, pp. 1034-1051, vol. 6, no. 10, 2000.