[Downloaded from www.aece.ro on Thursday, July 03, 2025 at 23:21:58 (UTC) by 172.70.100.102. Redistribution subject to AECE license or copyright.]

Improving the Performances of the nMPRA Processor using a Custom Interrupt Management Scheduling Policy

Ionel ZAGAN[1,2], 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*
zagan@eed.usv.ro

***Abstract*—A quantitative and qualitative increase in production has been obtained in most fields through the development of CPUs and real-time systems based on them. Such is the case in the industrial sector where the automation process relieved partly or wholly the human activities needed in the manufacturing process. This is mainly due to time sharing in embedded real-time systems and to pseudo-parallel execution of tasks in the implementation of a single central processing unit. The present article presents the validation of the nHSE (Hardware Scheduler Engine) scheduler implemented in hardware by using a FPGA Xilinx Virtex-7, Vivado development platform, and the Vivado Simulator. In this context, our main contribution relates to a custom interrupt management scheduling policy implemented in hardware at the nHSE level, in order to provide predictable execution for asynchronous interrupts. By reducing the jitter when handling with asynchronous interrupts and completely eliminating the uncertainties of the scheduling limit for the set of tasks, a significant improvement of the overall system's predictability has been obtained.**

***Index Terms*—field programmable gate arrays, pipeline processing, architecture, scheduling, operating systems.**

I. INTRODUCTION

One of the fundamental requirements of a real-time system (RTS) is the determinism of executing real-time critical tasks [1]. In RTSs based on software schedulers, the overhead generated by the operating system and the clock cycles needed for context switching are only a few parameters that can generate an increase of the jitter and the missing of deadlines. The hardware implementation of schedulers represents a novelty for real-time systems and a true challenge in the field [2].

In classical systems, the tasks of the application can be interrupted at any moment by the drivers dedicated to the interrupts. In real time operating systems, this approach can generate unpredictable interrupts of critical tasks, causing the non-compliance with the execution deadlines [3].

In real-time operating systems (RTOS), two approaches can be outlined for dealing with interrupts generated by external devices. The first method consists in associating each source of interrupts to an aperiodic or sporadic task. This task, scheduled as any other in the system, is responsible for dealing with the associated interrupts. This way, the time required for treating interrupts is automatically included in the scheduling mechanism. As the scheduler selects first the critical tasks, there are certain situations in which the tasks treating interrupts are not executed immediately. The incorrect implementation of this method, or an erroneous initialization of task parameters required by the scheduling algorithm (priority, deadline, type or criticality), can generate data loss in RTOS.

A different approach for implementing the treating interrupts routine is that of interrupting the current task and immediately executing the interrupt service routine (ISR). This method minimizes the latency of interrupts; nevertheless, the costs involved must be taken into account, so that to ensure the feasibility of the task set in the system.

This paper describes the implementation of Multi Pipeline Register Architecture (nMPRA) processor [4-6] using the FPGA Virtex-7 circuit, presenting the experimental model of the real-time scheduler implemented in hardware that implements a custom interrupt management scheduling policy specific to RTSs (subject of this article and the novelty for the proposed architecture). The project has been implemented using the VC707 Evaluation Kit [7] produced by Xilinx and Vivado 2015.4 design environment and the source code has been written in Verilog HDL. The nMPRA project is the result of the following: outstanding performance in the case of context switching, architectural simplicity, implementing specific deadlines in hardware, inter-task synchronization and communication implemented in hardware, predictable execution of real time tasks, treating external interrupts and time related events with minimum jitter and handling general exceptions.

The novelty of this paper resides in the innovative mechanism of handling interrupts integrated in the nHSE hardware module. Thus, the interrupts can be programmed in the same way as tasks, guaranteeing the feasibility of the system even when the interrupts are handled in real time. This paper presents an innovative processor implementation named nMPRA representing a feasible and realistic alternative to the already-existing solutions in the field, because it makes the best use of time and minimizes the jitter. For this to be obtained, the field-programmable gate array (FPGA) devices [8-9] with a high capacity in logic

This work was supported in part by the 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.

Digital Object Identifier 10.4316/AECE.2016.04007

1582-7445 © 2016 AECE

[Downloaded from www.aece.ro on Thursday, July 03, 2025 at 23:21:58 (UTC) by 172.70.100.102. Redistribution subject to AECE license or copyright.]

gates, available today at acceptable prices [10-11], represents a hardware support for the development of embedded real-time operating systems [12].

The present paper is structured as follows: after a brief introduction in section I, section II sums up a few useful methods for treating interrupts used in classical computing systems; section III describes the nMPRA processor and the mechanism for treating interrupts implemented in hardware, and finally, section IV focuses on the conclusions and future research.

II. Interrupts Problem in Classical Computing Systems and Related Work

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. Thus, the nMPRA architecture, together with the nHSE scheduler implemented in hardware, remove partially or totally the events handled through software in the case of treating an interrupt. One of the problems in treating interrupts is that the CPU must wait for an indefinite period of time so that the I/O module is prepared for the transmission and reception of data. During this standby period, the CPU must constantly interrogate the status of the I/O module; consequently, the performances of the entire system decrease [3]. To avoid this, the CPU could issue a command for the integrated module and only afterwards introduce other tasks in execution. Thus, at the end of each instruction cycle, the CPU checks for interrupts. When the I/O module is prepared to issue data towards the CPU, it will send the CPU an interrupt request; the CPU resumes the current task, saves the context of the current program (PC and CPU registers) and performs data transfer with the peripheral device. At the end of treating the interrupt, the CPU restores on a stack the context of the previously saved program and resumes its execution. Since interrupts can appear unpredictably in the running program, the context saving operation must include all information necessary for restoring the interrupted program. ISR is not a routine called by the running program and moreover, these two program sequences can belong to different users. In conclusion, in the process of treating interrupts, two main problems can occur: the processor identifying the I/O module that generated the event and the processing of simultaneous interrupts.

The occurrence of an interrupt triggers both hardware and software events. As it can be seen in Fig. 1, the following event sequence will result in case an I/O device initiates a data transfer:

- The I/O device emits an interrupt signal to the CPU;
- The CPU completes the execution of the current instruction, checks for the occurrence of an interrupt, and, if necessary, sends a confirmation signal to the device that generated the interrupt;
- The CPU saves on stack the information of the program that runs at the moment when the interrupt occurs. This information includes the (PSW – Program Status Word) and the location of the next instruction to be executed (PC – Program Counter). The CPU loads in the PC register the address for treating interrupts corresponding to the event that triggered it. In the next instruction cycle, the CPU continues to extract the instruction addressed by the new PC; thus, the program for treating interrupts gains control;
- After saving the program's current context on the system stack, the stack pointer is updated with a new value. In order to treat the interrupt, an examination is required, regarding either the condition relating to the I/O operation, or the event that caused the interrupt. This operation may need sending additional commands or confirmation signals to the I/O device;
- Treating the interrupt;
- After the interrupt has been successfully treated, the context restoring begins by copying the register previously saved on the stack. Finally, the CPU status contained in the PSW register is restored from the stack. At this point, the next instruction extracted and decoded belongs to the program interrupted by the ISR.

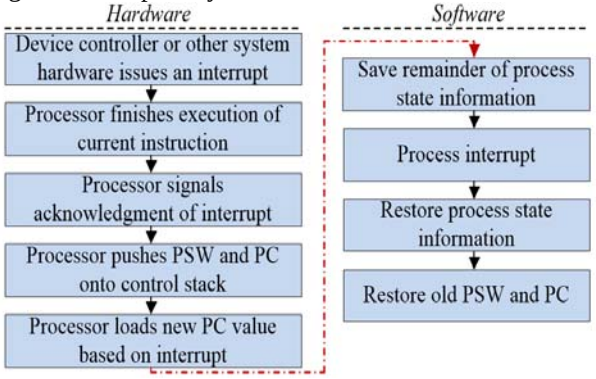

Figure 1. General scheme of processing an interrupt in a classical computing system [3]

The MSparc architecture presented in [13] is a custom 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.

The project Komodo presented by Kreuzinger et al. in [14], is a Java-based multithreading microcontroller for handling multiple real-time threads. The concept uses multiple program counters and instruction windows, stack register sets, and a signal unit to manage a set of threads triggered by interrupts. In order to provide real-time support and schedule multiple threads with different priorities using the proposed four-stage pipeline architecture, the picoJava instruction set is improved. Because multithreading microcontroller supports fast context-switching, if a branch or memory access causes pipeline stagnation, the Komodo Priority manager can schedule another thread to use the unallocated cycles.

III. Proposed Interrupt Handling System Implemented by nHSE Scheduler and nMPRA Support

The advantage of the interrupts handling system implemented by the nMPRA is that it is not necessary to use a dedicated controller for task selection and interrupts management [12].

[Downloaded from www.aece.ro on Thursday, July 03, 2025 at 23:21:58 (UTC) by 172.70.100.102. Redistribution subject to AECE license or copyright.]

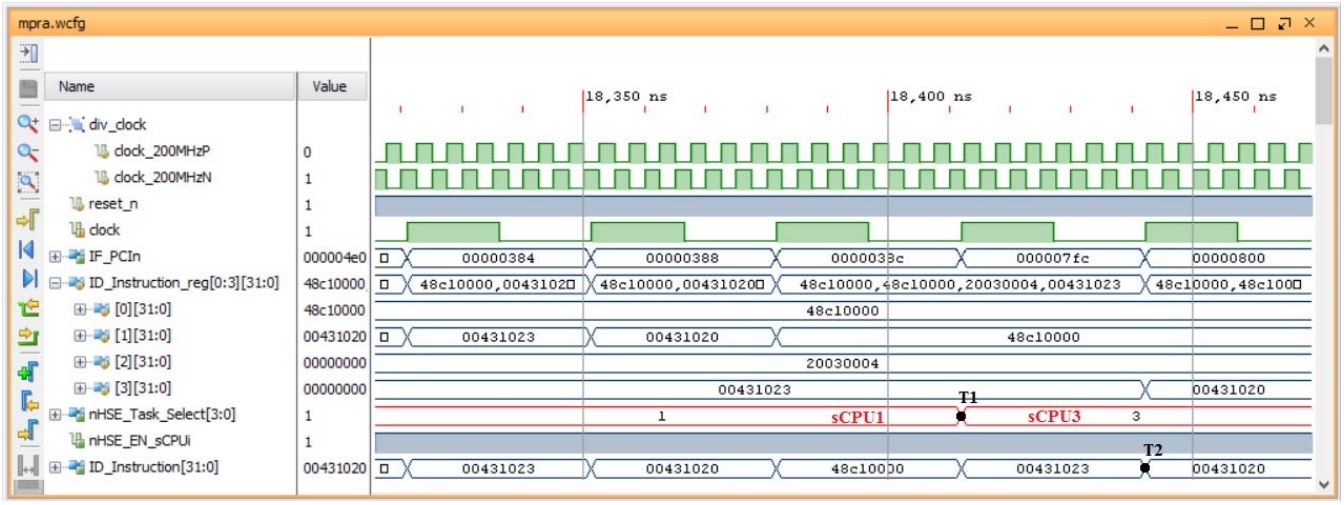

Figure 2. The sCPU1 and sCPU3 context switching operation; clock_200MHzP, clock_200MHzN - 200MHz differential signal clock; reset_n – reset signal; clock - nMPRA clock; IF_PCIn – program counter; ID_Instruction_reg[0:3][31:0] - reg type sCPUi instruction; nHSE_Task_Select[3:0] - sCPUi selector; nHSE_EN_sCPUi - nHSE enable signal; ID_Instruction[31:0] - wire type sCPUi instruction

The innovation elements introduced by this paper are the special results obtained from the synthesis, implementation and testing of the nMPRA processor, by using the FPGA Virtex-7 development kit from Xilinx. Vivado simulator and ChipScope analyzer were used in order to validate the nMPRA architecture, the instructions dedicated to the nHSE scheduler and measurement of the jitter when an external asynchronous interrupt occurs. The predictable response is ensured even when external asynchronous interrupts occur, when the CPU load is at the upper limit. The interrupt latency and context switching represent characteristics typical to the nMPRA processor, these coefficients having acceptable values for a deterministic architecture [15]. This section describe and validate a custom CPU architecture that includes a real-time scheduler implemented in hardware that totally or partly eliminates the overhead generated by functions specific to classical operating system. All nMPRA components are designed for absolute determinism and outstanding performances.

The aim of the tests performed using a competitive hardware platform was the practical implementation of the following: the nHSE hardware integrated scheduler, the module treating asynchronous external interrupts, and the System - on - Chip project used in testing the CPU. The CPU implements the MIPS instruction set [16], adding additional instructions for the integrated scheduler nHSE.

The architecture of the nMPRA processor is based on a five stages assembly line, enabling the simultaneous execution of up to five instructions on different stages [17]. In the best case scenario, when hazard situations do not occur on the assembly line, the execution report is one clock cycle/instruction. The implementation is based on the project described in [18], a 32-bit MIPS processor which aims for conformance with the MIPS32 Release 1 ISA. The MIPS32 architecture sets a new standard of performance for embedded processors on 32-bit. This architecture is the foundation of MIPS technologies for the next generation of high-performance processors based on this design, and it also features compatibility for the MIPS64 architecture. The MIPS architecture is extremely stable due to the robust set of instructions. The instructions are scalable from 32 bits to 64 bits, with a wide range of software development tools and consistent support from numerous licensed MIPS technologies.

Due to the fact that the new nMPRA architecture relies on multiplexing multiplied resources, such as Program Counter, Register File and Pipeline Registers, this structure forms a typical MIPS architecture which we will call semi CPU (sCPU). An *i* instance of this semiprocessor will be called semiprocessor *i* (sCPUi). The scheduler of the nMPRA processor, called nHSE is the central module of this architecture [15]. The implemented and validated architecture uses a unified space for interrupts and tasks, with a custom interrupt management scheduling policy. The scheduler also has a timer block for each task which can be configured within the boot procedure. Fig. 2 represents the context switching operation of sCPU1 and sCPU3, when the scheduler performs a preemptive scheduling algorithm based on priorities, at a CPU working frequency of 33 MHz. Thus, when a context switch occurs concurrently with IF_PCIn signals, the content of the ID_Instruction_reg[0:0][31:0] pipeline registers can be analyzed. If the program jumps at a different address, the efficiency of the assembly line decreases; the assembly line is thus loaded, starting with the first instruction from the new segment of the executed code program. The address of the next instruction will be updated, according to the selection of the nHSE scheduler, the control unit and the hazard detection unit; this instruction is stored in the PC register and the current instruction corresponding to the selected sCPUi will be extracted from the program memory. The reset_n signal and the 200 MHz differential clock signal of the Virtex-7 development kit produced by Xilinx can also be traced. With this signal and the IP Clocking Wizard 5.2 (Rev. 1), the 33 MHz clock signal of the nMPRA processor is obtained.

In the simulation presented in Fig. 2, one can see the context switching between sCPU1 and sCPU3 at time moment T1; this operation was dictated by the nHSE_Task_Select [3:0] selector and the nHSE_EN_sCPUi activation signal. Thus, under the direct command of the nHSE scheduler, the wire ID_Instruction[31:0] signals send the active MIPS or nHSE instruction forward in the pipeline. Moreover, the process of selecting the instructions stored in

[Downloaded from www.aece.ro on Thursday, July 03, 2025 at 23:21:58 (UTC) by 172.70.100.102. Redistribution subject to AECE license or copyright.]

the ID_Instruction_reg[1] and ID_Instruction_reg[3] pipeline registers as well as the 0x00431023-*subu*, 0x00431020-*add* and 0x48c10000-*wait* instructions (belonging to sCPU1) and 0x00431023-*subu* and 0x00431020-*add* (belonging to sCPU3) can be seen. The content of the ID_Instruction_reg[0] and ID_Instruction_reg[2] registers remains unaltered, because sCPU0 and sCPU2 are not selected for execution. The pipeline registers must memorize the data and the control signals corresponding to the instruction executed by each sCPUi. When resuming the task executed by sCPU3, the consistency of the restored data can be checked, following that, at time moment T2, the 0x00431020 instruction is extracted by IF_PCIn from address 0x000007fc.

The characteristics of the nHSE are the following: the number of designed and scheduled tasks must be within the limit of available resources of the FPGA circuit; each task has assigned a priority; for a high flexibility of the application, each task can have external interrupts attached; the task with the higher priority is selected by the scheduler and inserted in execution (if it is not blocked or suspended), using a unified space for interrupts and tasks; interrupts can be assigned to tasks so that tasks inherit both the priority and the characteristics; there are separate times for each task implemented in hardware and two types of deadlines (the first is an alarm and the second is a fault).

Analyzing the scheme of treating interrupts, we encountered the following problem: what happens if all interrupts are attached to the same sCPUi and occur simultaneously? In designing the nHSE scheduler, the following two solutions to this problem were implemented: the first one, also the simplest, is the software solution. It is a versatile solution and does not require additional hardware modules, because the interrupt priorities can be easily switched [19]. One disadvantage is the delays introduced by the test blocks and routines for treating interrupts in the case when more interrupts are attached to the same sCPUi and occur simultaneously. Moreover, the delay generated by the test blocks depends on the position of the test block moment; the second solution involves an additional hardware block. This paper presents the experimental results obtained from testing this solution. When one or more interrupts occur, the block implementing the priority encoder will generate an adequate number for the interrupt with the highest priority attached to the sCPUi. As can be seen in Fig. 3, the displacement of the trap cell is supplied by the new hardware block implemented in the nHSE scheduler; the control is thus transferred to the interrupt handler. For consistency, for each interrupt, the time delay of the decision blocks will remain the same. It is a rapid solution, but it requires an additional hardware block whose complexity is generated by all interrupts from the CPU and by the possibility of attaching them to different sCPUi.

Usually, situations of inverted priorities occur when the tasks with higher priority are suspended by interrupts assigned to tasks with lower priority. In order to eliminate this disadvantage, the nMPRA architecture uses task and interrupt unification in the same address space [20].

Fig. 4 shows the registers of the nHSE scheduler, with a four sCPUi configuration (sCPU0, sCPU1, sCPU2 and sCPU3) and four external interrupts ExtIntEv[0:3]; for reasons concerning space, only the ExtIntEv[0] interrupt attached to sCPU0 has been presented. In the upper part of the simulation, three control registers and one global register implemented by nHSE can be noticed in hexadecimal and binary format; the multiplication by 4 of the control and global registers is generated by the 4 sCPUi, as follows:

- crTRi[0:3][31:0] memorizes the events validated or inhibited by the *wait* nHSE instruction, one bit for each of the seven events (from right to left, the events generated by the timer, by the watchdog, by deadline 1 and deadline 2, by interrupts, mutexes and synchronizing events, are validated (1) or inhibited (0)). Thus, the 0x00000011 code indicates the fact that time events and interrupt-generated events are validated for each sCPUi;
- The crEVi[0:3][31:0] registers contain 32 bits for each sCPUi, with the role of signaling the occurrence of an event validated by the crTRi register;
- crEPRi[0:3][31:0] represents the registers that provide the priorities for events, by allocating 3 bits for each event (in Fig. 4, the P0 bits group represents the highest priority assigned to interrupts, and the P1 bits group represents the priority of the time event);
- grINT_IDi[0:3][31:0] is the register that selects the task ID to which the interrupt has been attached. In the present exemplification, it results an interrupt for each sCPUi.

The prioritization scheme implemented in this scheduler architecture selects the active event category with the highest priority, in order to treat it. The level of priorities for each category of events can be changed dynamically, depending on the requirements of the mixed-criticality real-time system. If several events are active in the selected category, another selection must be performed in order to find the event that will be treated effectively first. In the example shown in Fig. 4, the interrupt type event has the highest priority with a value of crEPRi[0][14:12]=3'b000.

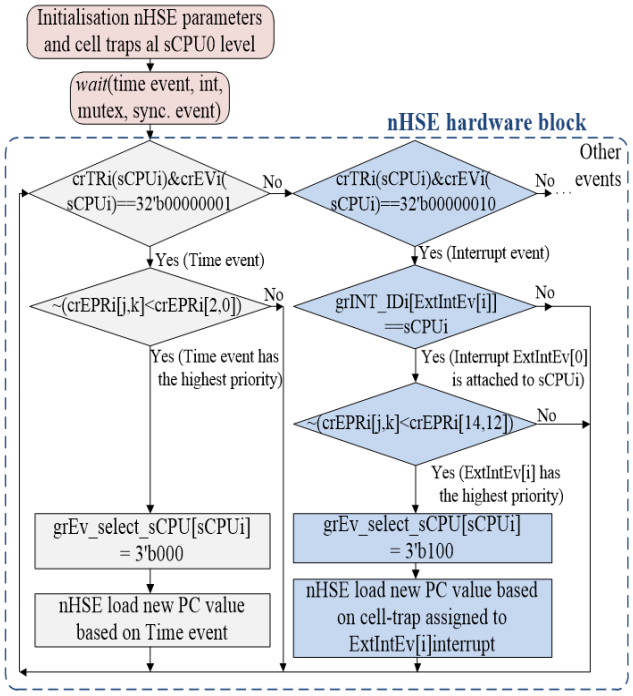

Figure 3. The hardware solution for treating interrupts implemented in the nHSE scheduler

[Downloaded from www.aece.ro on Thursday, July 03, 2025 at 23:21:58 (UTC) by 172.70.100.102. Redistribution subject to AECE license or copyright.]

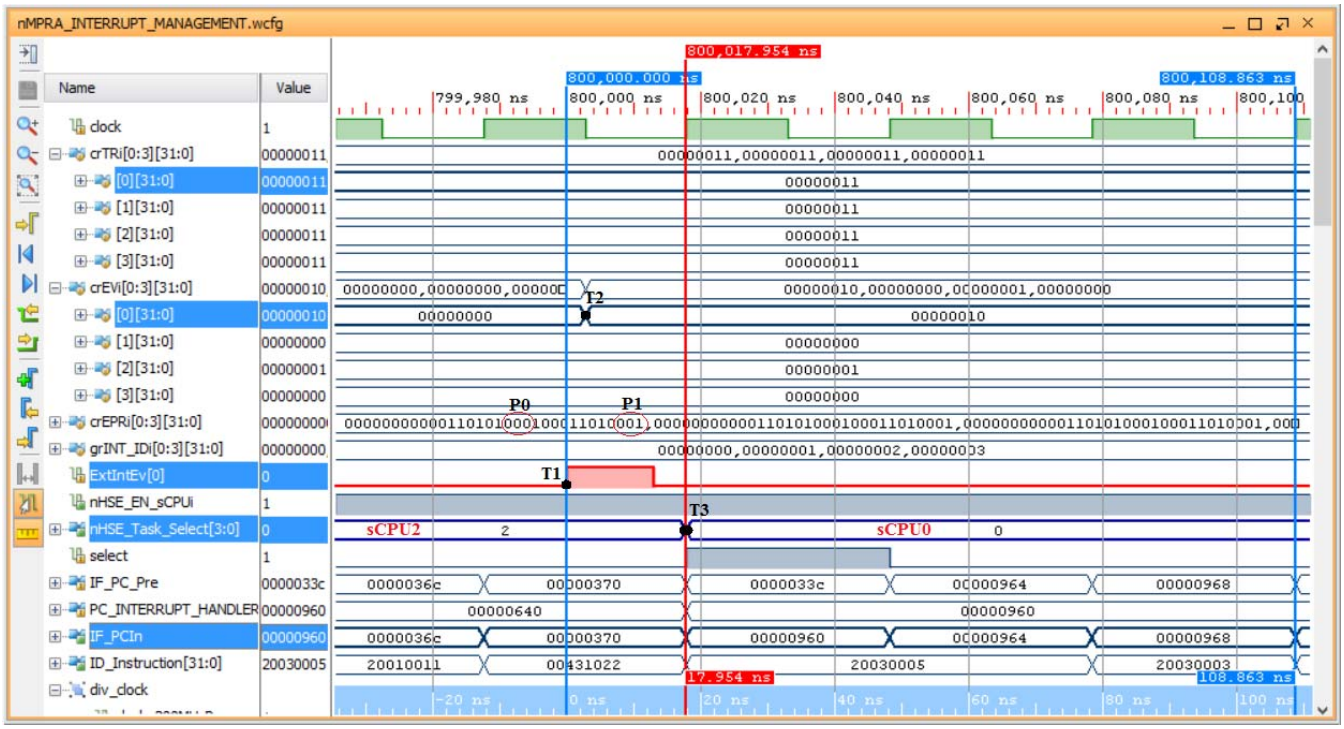

Figure 4. The response of the nHSE scheduler to treating an asynchronous external interrupt; ExtIntEv[0] - external interrupt signal; select – interrupt cell trap activation signal; PC_INTERRUPT_HANDLER – interrupt handler routine pointer associated with ExtIntEv[0] event

At level sCPU0, the time event is also enabled through the corresponding bit crTRi[0][0]; nevertheless, because the value of the event priority is set at crEPRi[0][2:0]=3'b001, the event will be scheduled after the treatment of the interrupt (crEPRi[0][14:12]< crEPRi[0][2:0]). Further on in this paper, we are describing the waveforms in the case when the scheduler responds to the occurrence of an asynchronous external interrupt. The setting of the ExtIntEv[0] signal generated by Vivado simulator and the answer of the scheduler can be observed by activating the high priority sCPU0 to which the event is attached.

The content of the general register grInt_IDi[0][31:0]=32'h00000000 indicates the fact that the interrupt ExtIntEv[0] is assigned to the sCPUi with ID=32'h00000000, corresponding to sCPU0, in the case of the static scheduler used for this simulation.

As it can be observed in Fig. 4, the signal for the asynchronous external interrupt ExtIntEv[0] was set at moment T1, and the bit crEVi[0][4] is set at moment T2, indicating the occurrence of an interrupt. Because sCPU2 is in execution, at the next positive edge of the clock signal, the context switching between sCPU2 and sCPU0 is performed under the command of the nHSE_Task_Select[3:0] selector and its nHSE_EN_sCPUi activation line. By context switching from moment T3, thus checking the performance brought by the hardware implementation of the scheduling architecture presented in this paper, the reaction of the system will be observed.

The global prioritization of the event system involves the existence in the system of seven types of various criticality events, along with the circuit for decoding and selecting the event with the highest priority. The crEPRi [0:3][31:0] registers provide the priorities of major events and, with the help of the demultiplexors activated by signals corresponding to the validated event, the priority of the event category is assigned, thus generating signals for handling the event with the highest priority. Thus, all events, single in their category, such as time-related events, have associated a trap cell indicating the routine of treating that time event. In addition, each interrupt has also attached a trap cell, the addresses of handling routines being loaded by sCPU0 into the pointer register, at startup, after reset. In case one of these events occur, the task associated to it will become active, and the IF_PCIn register corresponding to sCPU0 is automatically loaded with the content of the PC_INTERRUPT_HANDLER=0x00000960 pointer register, thus leading to the execution of the routine associated with the event. The return address contained in the IF_PC_Pre register, prior to loading the address of handling the routine of the occurred event, is automatically saved in a backup register associated to the sCPUi. If other events for sCPUi occur after completing the execution of the time event handling routine, the routine address for the event with the highest priority from among the remaining ones is automatically in IF_PCIn. The IF_PC_Pre register proceeds to the address for extracting the following instruction, when the *select* signal is deactivated.

As we can see in Fig. 4, the contexts switch operation is guaranteed in one clock cycle. At a 33MHz frequency, the scheduler answer to an asynchronous external event may be around 17.95ns (0.6 clock cycle). The internal logic of the nHSE block needs at least 15ns to perform task scheduling and remapping sequence of the contexts. It can be said that the experimental results demonstrate the practical implementation of the theoretical aspects, therefore obtaining very low times for interrupt handling and context switching operations. The ID_Instruction[31:0] pipeline signal transmit the active instruction, so that the MIPS instructions 0x20010011-*addi* *(R[Rt=1]=R[Rs=0]+ SignExtImm(0011))* and 0x00431022-*subu* belong to sCPU2, and 0x20030005-*addi* and 0x20030003-*addi* correspond to sCPU0. The execution of the program

[Downloaded from www.aece.ro on Thursday, July 03, 2025 at 23:21:58 (UTC) by 172.70.100.102. Redistribution subject to AECE license or copyright.]

containing these instructions validates the datapath, the mechanism for treating interrupts projected in hardware at the level of the nMPRA architecture, as well as the instructions dedicated the nHSE scheduler.

The mechanism for the management of interrupts, their dynamic attachment to semi-processors, the dynamic nHSE scheduler implemented in hardware, and the separate prioritization of the events group for each sCPUi are all part of the nMPRA architecture, a real and flexible implementation for mixed criticality systems. The procedure described above is valid for every event in the system, and for the events that may be more than one in their category, as is the case of external interrupts, the events generated by mutexes and those generated by communication events, the implementation of an additional hardware block is necessary. This is because the implementation of the control logic for selecting the event with the highest priority through the grINT_IDi, grMutexi and grERFi registers is necessary (grMutexi is the register that selects the task ID to which the mutex has been attached and grERFi is part of the event register file and defines an event).

The range of applications that can be used by the nMPRA processor includes the following: control applications in the field of industry, automotive, data communication equipments, medical devices, a wide range of other embedded safety-critical applications. A detailed comparison between the nMPRA architecture and other similar projects can be found in [6].

IV. Conclusion

The innovation elements introduced by this paper are rendered by the practical implementation of a custom interrupt management scheduler, based on the nMPRA architecture. Tests conducted and presented in section III justify the use of CPU in embedded systems, where there is a need for a superior computing power in order to run a high level application. At the same time, we must ensure very low response times, so as to guarantee the real time feature in the execution of the tasks, and the calculation of WCET coefficients.

Following the practical results and the implementation of a preemptive scheduling algorithm, the nMPRA architecture can successfully satisfy the requirements of real-time systems. The jitter of the hardware scheduler at a frequency of 33MHz is of 17.954ns, and the clock cycle is of 30.3030ns.

As future work, we aim to present the way in which the scheduler instructions are implemented at the level of co-processor 2, and the experimental results obtained from including in hardware the nHSE dynamic scheduler that enables sCPUi priority switch during execution.

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, pp. 13–30, Springer, 2011. ISBN: 978-1-4614-0675-4

[2] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, “FlexPRET: A processor platform for mixed-criticality systems,” in 20th IEEE Real-Time and Embedded Technology and Applications Symposium - RTAS, pp. 101–110, Apr. 2014. doi:10.1109/RTAS.2014.6925994

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

[4] 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

[5] 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

[6] 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

[7] www.xilinx.com/support/documentation/boards_and.../ug885_VC707_Eval_Bd.pdf, (Accessed: Aug. 2016).

[8] J. Shawash and D. R. Selviah, “Real-Time Nonlinear Parameter Estimation Using the Levenberg–Marquardt Algorithm on Field Programmable Gate Arrays,” IEEE Trans. Ind. Electron., vol. 60, no. 1, pp. 170–176, Jan. 2013. doi:10.1109/TIE.2012.2183833

[9] M. Shahbazi, P. Poure, S. Saadate, and M. R. Zolghadri, “FPGA-Based Reconfigurable Control for Fault-Tolerant Back-to-Back Converter Without Redundancy,” IEEE Trans. Ind. Electron., vol. 60, no. 8, pp. 3360–3371, Aug. 2013. doi:10.1109/TIE.2012.2200214

[10] M. Shahbazi, P. Poure, S. Saadate, and M. R. Zolghadri, “Fault-Tolerant Five-Leg Converter Topology With FPGA-Based Reconfigurable Control,” IEEE Trans. Ind. Electron., vol. 60, no. 6, pp. 2284–2294, Jun. 2013. doi:10.1109/TIE.2012.2191754

[11] T. T. Phuong, K. Ohishi, Y. Yokokura, and C. Mitsantisuk, “FPGA-Based High-Performance Force Control System With Friction-Free and Noise-Free Force Observation,” IEEE Trans. Ind. Electron., vol. 61, no. 2, pp. 994–1008, Feb. 2014. doi:10.1109/TIE.2013.2266081

[12] 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 – IJACSA, vol. 6, no. 4, 2015. doi:10.14569/IJACSA.2015.060406

[13] A. Metzner and J. Niehaus, “MSparc: Multithreading in Real-Time Architectures,” Journal of Universal Computer Science, vol. 6, no. 10, pp. 1034–1051, 2000. doi:10.3217/jucs-006-10-1034

[14] J. Kreuzinger, R. Marston, Th. Ungerer, U. Brinkschulte and C. Krakowski, “The Komodo project: thread-based event handling supported by a multithreaded Java microcontroller,” in 25th EUROMICRO Conference, Milano, vol. 2, pp. 122-128, 1999. doi: 10.1109/EURMIC.1999.794770.

[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] “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).

[17] 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

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

[19] 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

[20] 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