Predictable CPU Architecture Designed for Small Real-Time Application - Concept and Theory of Operation

Nicoleta Cristina GAITAN, Ionel ZAGAN, Vasile Gheorghita GAITAN
Faculty of Electrical Engineering and Computer Science
Stefan cel Mare University of Suceava, Romania
Suceava, Romania

***Abstract*—The purpose of this paper is to describe an predictable CPU architecture, based on the five stage pipeline assembly line and a hardware scheduler engine. We aim at developing a fine-grained multithreading implementation, named nMPRA-MT. The new proposed architecture uses replication and remapping techniques for the program counter, the register file, and the pipeline registers and is implemented with a FPGA device. An original implementation of a MIPS processor with thread interleaved pipeline is obtained, using dynamic scheduling of hard real-time tasks and interrupts. In terms of interrupts handling, the architecture uses a particular method consisting of assigning interrupts to tasks, which insures an efficient control for both the context switch, and the system real-time behavior. The originality of the approach resides in the predictability and spatial isolation of the hard real-time tasks, executed every two clock cycles. The nMPRA-MT architecture is enabled by an innovative scheme of predictable scheduling algorithm, without stalling the pipeline assembly line.**

Keywords—fine-grained multithreading; hardware scheduler; pipeline; hard real-time; predictable

I. INTRODUCTION

The spectacular development of the embedded systems in the past few years confirm their importance and major impact on the present-day scientific, technological, and socioeconomic areas. Their importance is rendered by the extensive applicability area (including real-time and low-power applications) of the present research in fields like automotive, robotics, and industrial control. The difficulties encountered while developing this approach are generated by the design process aimed at obtaining a predictable architecture.

Real-time systems (RTS) are those systems which provide a correct response within a predetermined time [1]. This predetermined time, considered a deadline, generates a classification of the embedded RTS in:

- Soft RTS – missing the deadline does not cause a critical effect;
- Hard RTS - missing the deadline causes a hazard situation.

The application always requires the properties that define the RTS, even if the system does not. The main feature of the embedded RTS is to ensure deterministic and predictable control of a process. In critical applications, obtaining a correct answer after the deadline is insufficient and cannot be taken into consideration. Depending on the consequences of missing a deadline, real-time tasks are classified into three categories [1]:

- Hard: if the missing of a deadline results in catastrophic effects, the task can be called hard real-time task;
- Firm: a task can be considered firm if the results produced after the deadline are not used within the system and do not involve damage;
- Soft: real-time tasks can be considered soft, if the results produced after the deadline may be used within the system, even if they degrade the system performance.

The limitations of the current Real-Time Operating System (RTOS) are rendered by the complex approach of the design level of the CPU, the memory, the I/O subsystem, and the high level languages and compilers. The time varying behavior of the RTOS implemented in software implies an unpredictable response for interrupts. For the most commercial RTOS, the execution of the same instructions in a variable number of cycles is generated by hazards. In order to eliminate this impediment, CPU architectures have developed deeper pipelines with out of order speculative execution and dynamic scheduling, while memories have been organized hierarchically by using cache memories on multiple levels.

In real-time systems, a single processor must execute multiple tasks with different priorities using an appropriate scheduling model. As a consequence, the system must meet safety and certification requirements, which are specified using standards for real-time kernels. Based on these constraints, in recent years, there has been intensive research on hardware scheduling of real-time systems. Among the many approaches regarding software and hardware scheduled [2][3][4][5][6], the actual implementations must provide hardware based isolation by means of a real-time operating system. While the commercial RTOS reduce significantly the hardware costs, these new approaches must be verified and certified, a process which is not simple since the system introduces overhead for task switching and execution time monitoring. Although using sophisticated mechanisms, in a real time system, it is difficult to determine the task's Worst Case Execution Time (WCET).

European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

The predictability of the current CPU implementations depends on the task scheduler, the pipeline ordering, the branch prediction, memories and caches.

Some fine-grained multithreaded CPU implementations can preserve spatial isolation, having inadequate thread scheduling algorithms.

The XMOS X1 project can fully utilize the processor, activating simultaneously four threads; still, a dynamic scheduling may reduce temporal isolation of active threads [5].

In the PTARM implementation [7], the authors isolate each hard thread; the disadvantage is that these exactly four threads can be constantly active in the five stage pipeline architecture.

The FlexPRET, presented in [8], is a fine-grained multithreaded processor designed for mixed-criticality systems. This implementation stalls the pipeline assembly line, when the scheduler executes instructions, every clock cycle, from same thread.

By incorporating RTOS functionality into hardware, we obtain a performance improvement of the entire system, guaranteed by the appropriate benchmark programs.

As a consequence, field-programmable gate array (FPGA) devices, with a high capacity of the logic gates and efficient prices [2], represent a hardware support for embedded RTOS. For this reason we propose a hardware implementation for RTOS functionalities, using the FPGA systems [6].

This paper is structured as follows: in section I is presented an introduction, the fine-grained architecture nMPRA-MT is presented in section II and the dynamic scheduler is described in section III. Similar works are described in section IV and final conclusions, including future work, are inserted in section V.

II. THE FINE-GRAINED NMPRA-MT ARCHITECTURE

In order to obtain an innovative implementation with predictable behavior, we propose a fine-grained multithreading processor, with spatial and temporal isolation between tasks. This architecture implemented for n tasks, called Fine-grained Multithreading - Multi Pipeline Register Architecture (nMPRA-MT), extends the Multi Pipeline Register Architecture (MPRA) project presented in [3] and [4]. This concept replaces the stack saving classical method with a remapping algorithm, which uses the replication of resources such as program counter, register file and pipeline registers, for n threads, τ_n, as shown in Fig. 1. Our CPU architecture uses the basic idea of the FlexPRET project presented in [8]. To ensure predictability, we propose a new scheduler architecture, which executes hard real-time tasks every two clock cycles. Implementing a new forwarding unit, our architecture is able to eliminate stalls from pipeline assembly line.

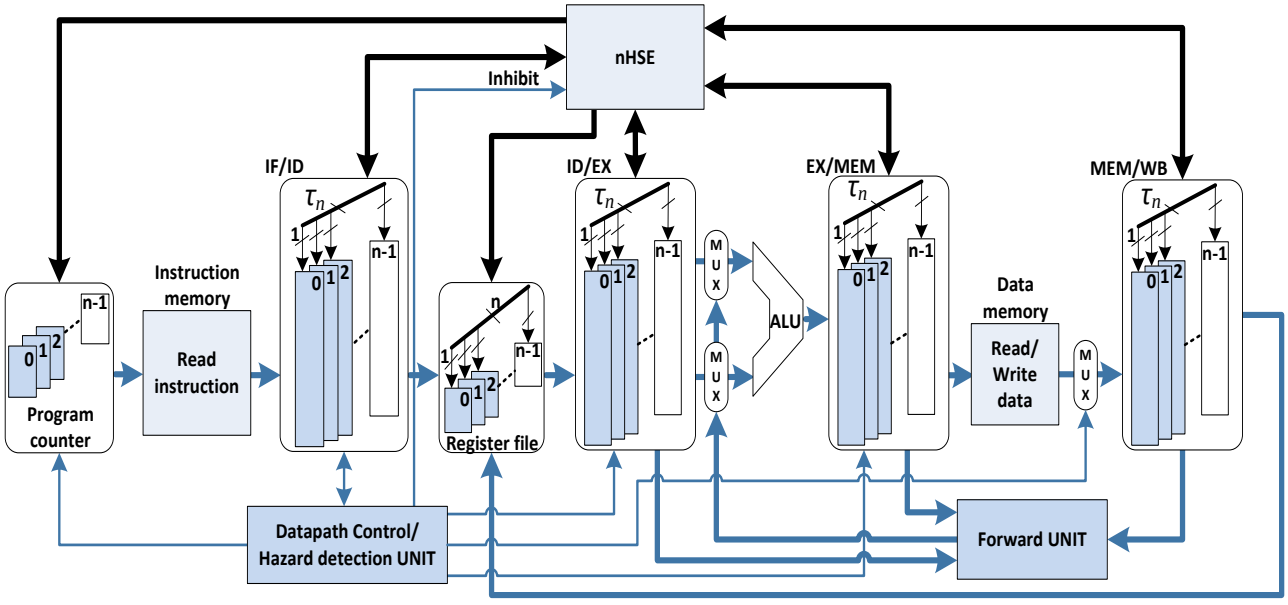

Fig. 1. Five stage fine-grained nMPRA-MT architecture with resource replication (PC, register file, pipeline registers)

The proposed architecture nMPRA-MT is designed to implement the MIPS instruction set, adding new instructions, required for tasks scheduling operation. This original architecture enables the execution of the new scheduled task, starting with the next clock cycle. In order to implement the nMPRA-MT project, we use a hardware scheduler engine for n threads (nHSE), with dynamic scheduling algorithms for tasks, interrupts, and events. The aim of the nHSE is to interleave instructions from different threads in the pipeline assembly line. The scheduler dynamically controls an arbitrary number of threads within the pipeline levels.

The concept of the fine-grained multithreading can be defined as the technique to fetch, each clock cycle, instructions from different hardware threads, allowing instructions from multiple threads to be interleaved dynamically in the pipeline.

The nMPRA-MT is a fine-grained multithreaded architecture, designed for the hard real-time system requirements. In order to preserve the performance of the

classical pipeline implementation, we use a new innovative pipelined assembly line containing five stages. This allows the simultaneous execution, within the pipeline levels, of four instructions from different threads, thus the final execution report is an instruction per clock cycle. Being a fine-grained thread-interleaved pipeline implementation, the nMPRA-MT is able to execute the instruction from different threads at any given time. If the instruction is fetched from different threads, no hazard situations are possible, but if the scheduler fetches instructions from the same thread every two clock cycles, it is possible to meet hazard situations resolved by the Forward Unit. As can be seen in Fig. 1, a dedicated Hazard Detection Unit was designed in order to detect any structural hazards, data hazards or control hazards. With the Forward Unit, it is no longer possible to stall an instruction from a thread which has already been fetched, if it depends on data which has not been calculated yet, as presented in the next chapter.

Hard RTS have a critical behavior in terms of real time, where the hardware-based isolation and the predictability of the threads are very important characteristics. In this scope, our implementation classifies the high priority tasks as hard threads–HT. Soft real-time tasks–ST, are tasks for which the results produced after the deadline do not cause a critical effect. By scheduling an arbitrary set of tasks with nHSE, our architecture supports hardware-based isolation for HTs, and offers STs the unused CPU cycles.

The nMPRA-MT architecture has the interrupt system presented in [3]. This interrupt handling is completely distributed and flexible in the system, allowing us to prevent the unpredictable situations generated by the interrupt service routines (ISRs). In order to obtain a minor jitter, the interrupts are attached to a HT or a ST task, inheriting the type and scheduling method used by nHSE. This represents an important feature of the embedded systems, because it is not necessary to flush the pipeline on their appearance. If the interrupt is assigned to an HT having the highest priority, the execution starts without affecting the pipeline assembly line.

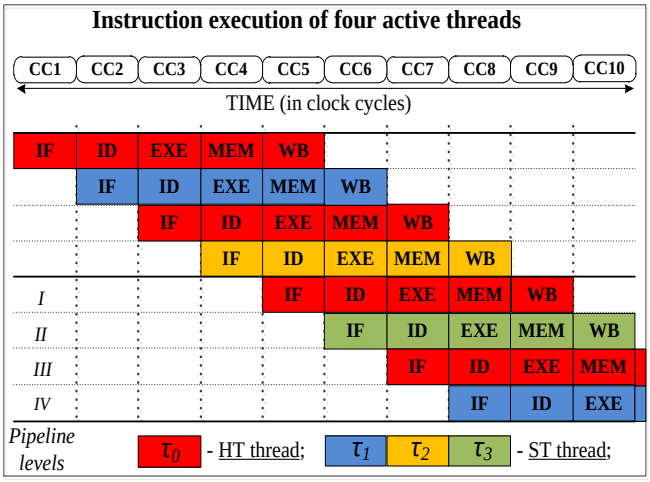

Fig. 2. Instructions execution by fine-grained nMPRA-MT pipeline architecture. IF-Instruction Fetch stage, ID-Instruction Decode stage, EX-EXecute stage, MEM-Memory stage, WB- Write Back stage

In order to minimize the number of clocks per instruction execution, a data path pipelined scheme with five stages was designed. As can be seen in Fig. 1, the multiplexor which selects the data to be written in the Register file, through the MEM/WB pipeline register, is placed immediately after the data memory. Because the data is stored on the rising edge of the CPU clock in the Memory pipeline stage, while the reading operation of the instruction is made on the falling edge of the clock, situation hazards can be avoided when the CPU executes multiple instructions from the same thread every two cycles.

Gaitan et al. present in [9] the basics of the nMPRA architecture for n tasks using the remapping algorithm with the replication of resources. Thus, each task scheduled by the nHSE has a distinct program counter (PC), a separated bank in the Register file, and a distinct set of pipeline registers (IF/ID - Instruction Fetch-Instruction Decode stage, ID/EX - Instruction Decode-Execute stage, EX/MEM - Execute-Memory stage, and MEM/WB - Memory-Write Back stage). When a new task is placed on the pipeline, the context switch can be accomplished in only one clock cycle. An instance of the processor represents a semi CPU (sCPUi for task i); all the resources of the processor are shared by every sCPUi, except the sCPU0. The sCPU0 instance is the only one active after power-on or reset, being able to access the scheduler configuration and task registers. The IDLE, SLEEP, and RUN state for each thread scheduled by the nHSE, is stored in a special STATE register. Along with the ID priority register assigned for each thread, an appropriate scheduling algorithm is implemented and executed for all HT or ST threads. In the nHSE scheduling scheme, every τ_n has its own ID and STATE register. The τ_0 thread, with an ID equal to 0, has the highest priority, and the lowest priority thread, τ_{n-1}, corresponds to an ID equal to n-1. As can be remarked in Fig. 1, the replication of the program counter, register file and a set of pipeline registers is made for τ_n threads. The threads filled in blue (τ_0, τ_1 and τ_2) are in the RUN state and share the resources of the processor pipeline, but the threads left blank (τ_{n-1}) are in the IDLE or SLEEP state.

III. Dynamic Scheduling Implemented by nHSE

The nMPRA-MT architecture is equipped with a dynamic scheduler built into an FPGA device. The preemptive scheduling algorithm, implemented by the nHSE, is able to perform fast task switching, without flushing or stalling the pipeline assembly line.

The nHSE is a finite state machine (FSM) implemented into the hardware, having an independent execution handling of the input events, such as interrupts or timers [4]. As can be noted in Fig. 1, the presence of the Inhibit signal from the IF/ID pipeline register to the nHSE block, signals the presence of the load and store instructions. This mechanism is implemented to guarantee that memory access is atomic.

Our CPU is structured to be a five stage MIPS pipelined processor, (IF - Instruction Fetch, ID - Instruction Decode, EXE - Execute and MEM - Memory, WB - Write Back), maintaining the performance of pipeline computing.

In Fig. 2, the τ_0 thread depicted in red, is introduced for execution in pipeline levels I and III.

Through this scheduling, at the expense of an execution latency of two clock cycles for τ_0, the worst cases of pipeline stagnation are avoided.

In levels II and IV, three ST threads (τ_1, τ_2 and τ_3) have been introduced by using the round-robin scheduling algorithm, achieving a full charge of the CPU pipeline. Thus, the nHSE manages all the HT and ST threads, depending on the type and state of each owner. For an optimal use of the CPU cycles, the τ_1, τ_2, and τ_3 are scheduled when the HT threads are in the IDLE or SLEEP state.

The Hazard detection unit and Forward unit blocks, depicted in Fig. 1, detects if there are hazard situations and forward data when is necessary. In the example presented in Fig. 2, it is possible to solve all hazard situations, because the τ_0 is scheduled every two cycles.

A data hazard which stalls the pipeline is not possible in nMPRA-MT, because the new nHSE does not schedule in a continuous manner the instructions from a particular thread τ_i. In other CPU implementations, it is possible to waste clock cycles, when the pipeline must be stalled in order to wait for the data processed by the previous instruction of the same thread. If data hazards appear when different τ_i are interleaved in the pipeline, the forwarding unit redirects the data from the memory directly to the executing stage, avoiding the stagnation of the pipeline.

In the example outlined in Fig. 2, the nMPRA-MT executes 10 instructions from τ_0, τ_1, τ_2 and τ_3 in the same number of clock cycles. Having an arbitrary number of hard and soft threads, the calculation of a WCET is a simpler action if the scheduled algorithm is the appropriate one. If τ_i proves to be schedulable after the feasibility analysis, the nMPRA-MT becomes a predictable architecture, providing, as well, hardware-based isolation for the HT threads.

In order to obtain a minimal jitter accepted by the real-time application, each thread maintains its own program counter, general-purpose registers, and pipeline registers. The nHSE decides which thread to fetch the next instruction from, according to schedule algorithm and deadlines.

Because the number of threads scheduled exceeds the number of pipeline levels, it is necessary to know which thread will meet its deadline earlier. Based on the ID and STATE registers, it is possible to execute without penalties a new instruction from a different τ_i.

The processor must include a solid support when working with critical resources. For this reason, the hardware support dedicated to the mutex and simple semaphores, represents an improvement on the predictability of the system.

In order to guarantee the timing predictability of each HT, a constant scheduling frequency is required in order to meet an individual deadline. This scheduling constraint is not required for ST because the results produced after the deadline of the threads may still be used. For the HT tasks in real-time systems, the upper bound is fixed at compile time, considering also the time spent with the interrupts execution. Based on the feasibility study of a set of tasks, the WCET analysis must confirm the safe upper bounds for the HT.

The development of a new application was necessary because the present architecture extends the instruction set of the traditional MIPS processor, adding specific instructions for the dynamic scheduling of the fine-grained multithreading architecture. In order to implement all the modules of the CPU in VERILOG, the logic structures which operate concurrently must be synchronized so as to obtain the right behavior of the system. For this reason, the external clock must control the correct blocks at the right time. To validate the correct functionalities of the real-time nMPRA-MT architecture, the traditional MIPS compilation tools can be used, without many changes.

IV. Related Work

This chapter provides a brief description of a few similar architecture and scheduler implementations, regarding the development of the real-time kernel primitives in hardware.

We start with the XMOS project, proposed by May, in [10]. The author has implemented an XMOS processor which can use the entire CPU even if the number of active tasks is less than four. A variation in the number of active tasks reduces the temporal isolation and produces difficulties in calculating WCET. The new XMOS architecture, presented in [11], allows the architects to build systems with multiple Xcore, providing a communication mechanism based on messages between all the Xcores within the kernel. Implementing a communication protocol between multiple cores via links, XMOS can be successfully used in multicore systems, dedicated boards or distributed systems.

The PTARM project presented in [7], guarantees spatial isolation for each thread. This is possible if there are at least four threads constantly active to fully utilize the processor. The PTARM has five stages of pipeline, requiring at least four threads to keep the processor completely occupied. The tasks are scheduled in the pipeline using the round-robin algorithm. PTARM is recommended for hard real-time systems because it has a constant frequency, but cycles are lost if less than four threads are executed.

The new CPU architecture presented in [8], is a fine-grained multithreaded processor designed to support the architectural techniques for mixed-criticality systems. FlexPRET supports an arbitrary number of interlacing threads controlled by a new scheduler.

Threads are classified as hard (hard real-time thread - HRTT) or soft (soft real-time thread - SRTT). FlexPRET supports hardware isolation for HRTT, while allowing the SRTT to efficiently use the CPU.

In [12], the authors proposed a new processor called JOP-Plus. This processor may be used for embedded systems, even in real-time applications. This is because, most of the code is written in SystemJ, the programming language used for designing the concurrent distributed software systems. The main reasons why the Java language has not been introduced in real-time applications are:

- Code execution needs a Java Virtual Machine (JVM) as an additional layer between the processor and the Java code;
- The mechanism's automatic garbage collection makes the response of Java programs to be unpredictable;
- The structure of Java concurrent programs is completely non-deterministic.

Register file (RF) 16x16 is used for temporary storage of the operands used by the Concurrency and Reactivity Control Flow (CRCF) code. Unlike GALS-JOP, the data memory is in the 16 bit format and stores the CRCF memory data structures. The JOP-Plus architecture has improved memory organization compared to the GALS-JOP basic project, eliminating the instruction jump-table. The new developed processor outperforms the SystemJ execution platform while having the most efficient use of the FPGA, being optimal in embedded real-time applications.

Al-Zawawi et al., in [13], propose an architecture which can be divided into a set of virtual processors. The execution time of these processors has a separate context, providing a composite time for the tasks which run on a virtual processors. The architecture allows us to partition it into a few processors with higher performance or more processors with lower performance or a combination of the two extremes.

Rochange and Sainrat, in [14], propose changes in the dynamic superscalar processor pipeline lines by slowing (stalling) the instructions between blocks. They realized that the time consumed by the basic blocks is independent from one block to another, by stalling the instruction extraction in a basic block until the instructions from the previous block are executed.

V. Conclusion and Future Work

The current paper extends the processor architecture presented in [3] and [4], implementing original new solution for the nMPRA and nHSE real-time behavior. The scheduler architecture has been implemented with a dynamic algorithm, providing predictability and hardware based isolation for the HT. The necessities for implementing RTOS in hardware are: jitter reduction, improved response time for external events, reduction of CPU, memory footprint and eliminating as much of the execution overhead given by the scheduler as possible.

The nMPRA-MT project is a powerful architecture based on its proprieties:

- The nMPRA-MT architecture is a predictable one because the nHSE eliminates stalls from the pipeline assembly line while executing the HT every two clock cycles;
- The pipeline is not reset because it is not necessary to restore/save the context due to the replication of resources (PC, file registers and pipeline registers);
- It uses a strong statement that a task can wait for different types of events (time, mutex, private event, interrupts timers for deadlines, etc.);
- Switching between tasks is usually accomplished in a single machine cycle, maximum three machine cycles when working with global memory;
- It comes with a distributed controller which inherits the priority interrupts of that interrupt task;
- It supports dynamic scheduling algorithms.

However, the architecture is susceptible to improvements such as:

- The implementation of nHSE as a coprocessor to use the existent professional compiler facilities, such as MIPS and ARM Cortex-Mx;
- Improving predictability in case of data hazards which can provide different times for the same instruction execution;
- Increase the parallelization of the instructions' execution through optimal scheduling algorithms;
- Improved response time to simultaneous multiple events and multiple interrupts by using priority encoding and transferring them directly to the event handlers;
- The explicit definition of a memory and peripherals model.
- The implementation of a wide variety of possible configurations for the processor in FPGA.

Acknowledgment

This paper was supported by the project "Sustainable performance in doctoral and post-doctoral research PERFORM–Contract no. POSDRU/159/1.5/S/138963," project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

References

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

[2] Shawash, J.; Selviah, D.R., "Real-Time Nonlinear Parameter Estimation Using the Levenberg–Marquardt Algorithm on Field Programmable Gate Arrays," Industrial Electronics, IEEE Transactions on, vol.60, no.1, pp.170-176, Jan. 2013.

[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," 2012 IEEE EIT International Conference on Electro-Information Technology, Indianapolis, IN, USA, 6-8 May 2012, ISBN: 978-1-4673-0818-2, ISSN: 2154-0373

[4] Gaitan, V.G.; Gaitan, N.C.; Ungurean, I, "CPU Architecture Based on a Hardware Scheduler and Independent Pipeline Registers" Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, no.99, pp.1,1 doi: 10.1109/TVLSI.2014.2346542.

[5] May, D., The XMOS XS1 Architecture. XMOS, 2009.

[6] Liu, I., Reineke, J., & Lee, E. A. (2010, November). A PRET architecture supporting concurrent programs with composable timing properties. In Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on (pp. 2111-2115). IEEE.

[7] I. Liu, J. Reineke, D. Broman, M. Zimmer, and E. Lee, "A PRET microarchitecture implementation with repeatable timing and competitive performance," in Proc. of the 30th IEEE International

Conference on Computer Design (ICCD), 2012, pp. 87–93.

[8] Michael Zimmer, David Broman, Chris Shaver, and Edward A. Lee. FlexPRET: A Processor Platform for Mixed-Criticality Systems. Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS), Berlin, Germany, April 15-17, 2014 (accepted paper).

[9] E. Dodiu, V.G. Gaitan and A. Graur, "Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – architecture description," IEEE 35'th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics, Croatia, 24 May 2012, ISSN: 1847-3946, ISBN 978-953-233-069-4.

[10] May, D., "The XMOS Architecture and XS1 Chips," Micro, IEEE , vol.32, no.6, pp.28,37, Nov.-Dec. 2012, doi: 10.1109/MM.2012.87.

[11] https://www.xmos.com/download/public/xCORE-Multicore-Microcontrollers-Overview%281.3%29.pdf.

[12] Nadeem, M.; Biglari-Abhari, M.; Salcic, Z., "JOP-plus - A processor for efficient execution of java programs extended with GALS concurrency," Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific , vol., no., pp.17,22, Jan. 30 2012-Feb. 2 2012, doi: 10.1109/ASPDAC.2012.6164940.

[13] A. El-Haj-Mahmoud, A. S. Al-Zawawi, A. Anantaraman, and E. Rotenberg, Virtual multiprocessor: an analyzable, highperformance architecture for real-time computing, in CASES '05. New York, NY, USA: ACM, 2005, pp. 213–224.

[14] Rochange and P. Sainrat, A time-predictable execution mode for superscalar pipelines with instruction prescheduling, in CF '05. New York, NY, USA: ACM, 2005, pp. 307–314.