Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

Predictable CPU Architecture Designed for Small Real-Time Applications – Implementation Results

Ionel Zagan, Vasile Gheorghita Gaitan

***Abstract*—The purpose of this paper is to describe and present the implementation results of nMPRA-MT processor concept designed for small real-time applications. Our target is to validate a fine-grained multithreading CPU architecture that uses replication and remapping techniques for the program counter, general purpose registers and pipeline registers. The new predictable CPU implementation is based on a hardware scheduler engine, being able to schedule dynamically a set of tasks on the five-stage pipeline assembly line. Using a FPGA device from Xilinx, we validate the innovative nMPRA-MT processor, interleaving different types of threads into the pipeline assembly line, providing predictability and hardware-based isolation for hard real-time threads. Mechanisms for synchronization and inter-task communication are also taken into consideration.**

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

I. Introduction

One of the present-day tendencies in the automotive and industrial field is to migrate towards more and more complex multithreading and multi-processor microcontroller architectures. The complexity of the applications in the field already mentioned require the design and implementation of some greater computing power hardware systems. This trend is first of all the result of the need to carry out more complex, reliable and safer applications. Because increasing the operating frequency is not always an effective solution due to the energy consumption of mobile applications, one of the options is to incorporate in the same silicon chip a number of similar computational kernels so as to allow a more efficient management of time, tasks and implicitly of energy consumption.

A parameter with a negative influence on the performance on a real-time system is the over-control due to the operating system [1]. The scheduling algorithm and task context switching operations may significantly influence the scheduling limit for those systems with a high frequency of task switching. This is the reason for which, in parallel with the improvement of software scheduling algorithms we also aimed at implementing them in the hardware, specifically to relieve the processor of the scheduling activity and to diminish the over-control specific to the operating system.

Ionel Zagan
Computers, Electronics and Automation Department, Ștefan Cel Mare University of Suceava
Romania
zagan@eed.usv.ro

Vasile Gheorghiță Găitan
Computers, Electronics and Automation Department, Ștefan Cel Mare University of Suceava
Romania
gaitan@eed.usv.ro

The present paper validates hardware-implemented real-time processor architecture on a FPGA-based development platform [2], which may be an important solution to mixed-critically systems.

Based on a hardware-implemented real-time scheduler, this paper comes with a realistic and feasible alternative to the already existing solutions, making the use of time more efficient and also ensuring the predictability of hard thread execution. The solution presented in this paper is a new processor which can execute dynamic scheduling algorithms without the need to have a software-implemented operating system [3], representing an innovative solution for real-time systems where time is a decisive factor of correctness. The execution of the scheduling blocks in the hardware, eliminates over-control due to the operating system, thus improving the scheduling limit of task set and the performance of the overall system. Since the processor architecture with integrated hardware scheduler is one based on resource multiplexing where the memory consumption used in the implementation varies in proportion to the number of used tasks, it is important to indicate that this architecture is intended for industrial and automotive embedded applications where the number of tasks lies in the range [8, 32]. More often than not, a number of tasks varying around value 16 is more than enough for most of such small applications. In the automotive field, the number of tasks used in Powertrain or Safety applications vary between 6 and 12. We therefore consider that the use of nMPRA-MT architecture for such projects is a realistic one. The added efficiency and improving usage safety, in accordance with Standard ISO26262, are the main features that support this architecture.

Multi Pipeline Register Architecture - Fine-grained Multithreading (nMPRA-MT) offers an excellent solution to this issue, because tasks and functions have individual contexts that are being hardware-managed, lacking any additional over-control which may lead to penalties in the performance time of applications. If such an architecture is being used, the user is absolved of the necessity to implement a software algorithm to check stack integrity. The round-robin or preemptive scheduling block which nMPRA-MT can implement with the help of time integrators and of hardware activated tasks (HAT), closely resembles the mechanism implemented in AutosarOS which inherits all the features from OSEK/VDX standard (Open systems and the corresponding interfaces for automotive electronics/ Vehicle Distributed eXecutive) [4].

This paper is organized as follows: section II describes similar papers in the field of safety-related real-time embedded systems. Subsequently, section III gives an overview of the original nMPRA architecture, while the requirements for hardware support are presented in section IV. In section V we describe the validation of the nMPRA-MT project and in section VI we conclude the paper with a presentation of the future work.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

II. Related work

This chapter presents few similar architecture and scheduler implementations that perform the development of real-time kernel primitives in hardware, achieving predictability and hardware based isolation for hard real-time threads.

The aim of Merasa Project presented in [5] and [6] was to implement a predictable processor architecture to be used successfully in hard real-time embedded systems. MERASA processor has a hardware designed for mixed-criticality systems, focused at the multi-kernel level. The proposed architecture is based on the SMT technique (Simultaneous Multithreading), being able to execute at the same time hard real-time (HRT) and non real-time (NHRT) threads.

The main features of this project is the predictibility of HRT thread execution and the efficient analysis of the WCET (Worst Case Execution Time) coefficient for each separated thread.

The FlexPRET project presented in [7] is a predictable processor designed for mixed-criticality systems. Authors have fundamented their work on fine-grained multithreading in order to design a new processor which interlaces an arbitrary number of threads on the assembly line. By classifying threads into two categories, hard-real time thread (HRTT) and soft-real time thread (SRTT), FlexPRET architecture manages to guarantee hardware isolation for HRTT threads and to use efficiently the processor by means of SRTT threads. If there is no HRTT thread scheduled for the execution, the available cycles may be used by SRTT threads in the round-robin mode. In this way, the dynamic scheduler has the advantage of using all processor cycles to the detriment of a more hard to get WCET.

The disadvantage of this implementation is that the processor cannot avoid pipeline assembly line stalls, when the scheduler executes instructions every clock cycle from the same thread.

The basic idea of the Komodo project presented by Kreuzinger et al. [8] is to use the multi-thread Komodo Java-based microcontroller to manage real-time execution multiple threads. Therefore, the proposed architecture uses multiple stacks, program counters, instruction windows and a signal unit to manage a set of execution threads activated by means of isolations. In order to offer a real-time support, the authors have improved picoJava instructions set.

Scheduling execution threads of various priorities on a four-stages assembly line is carried out by Komodo Priority manager, ensuring a quick change of contexts. In what concerns transfer and memory access instructions which may lead to assembly line interrupts, the scheduler may assign unused cycles to another execution thread.

Therefore, although Komodo project allows other execution threads in the ready state to be executed when the assembly line is free, thus increasing the instructions interleaving factor, this paper does not describe any synchronization or communication mechanism.

III. Overview of the nMPRA architecture

The Multi Pipeline Register Architecture (nMPRA) implemented for n threads is based on a hardware implemented scheduler as an integral part of the processor entitled Hardware Scheduler Engine (nHSE). The nMPRA concept replaces the stack saving classical method with a remapping technique [9], which uses the replication of program counter, register file and pipeline registers for n threads, as shown in Figure 1.

The major advantage of the nMPRA project is that, although it was designed for the single thread operating mode, the assembly line is not affected by the remapping operation of task contexts and implicitly the performance of the processor is not down-graded. The nMPRA processor based on the five stage pipeline assembly line is designed to execute the MIPS instruction set, implementing new instructions for task scheduling operations (we will use the terms “thread” and “task” interchangeably). The original architecture presented in [9], guarantees the execution of the new scheduled task starting with the next clock cycle or in 3 clock cycles when we wish to ensure memory consistency.

In order to implement the nMPRA project, the authors use nHSE with static scheduling algorithms for tasks, interrupts, and events. nHSE is directly responsible for the remapping operation of the pipeline registers set and of the registers file. Therefore, on the ascending front of a clock cycle operate the datapath control and pipeline control units, and on the descending front operates the hardware-based real-time scheduler [10]. Complete isolation of contexts and the small response time to events are the main features of nMPRA architecture. These features enable the integration of the project into hard real-time systems, guaranteeing real time performances necessary to these systems.

Operating at a frequency of 50 MHz, nMPRA implementation can achieve context commutation or answer to an external event in a time interval between 20 and 60ns, which confirms the proper operation of the theoretical model.

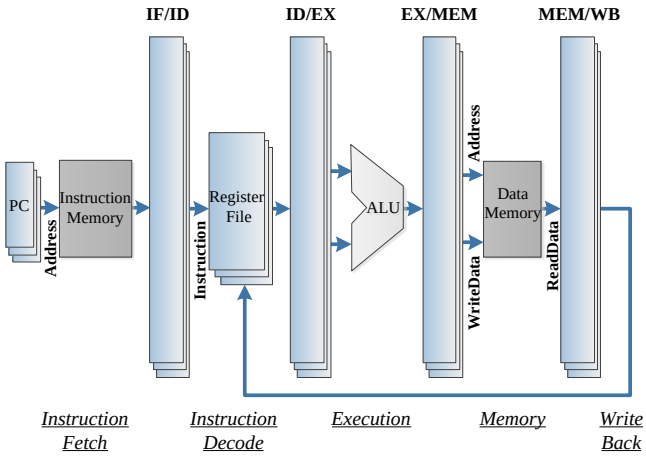

Figure 1. Replication of resources of the nMPRA architecture. PC-program counter, IF/ID-Instruction Fetch/Instruction Decode stage, ID/EX-Instruction Decode/EXecute stage, EX/MEM-Execute/MEMory stage, MEM/WB-MEMory/Write Back stage.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

IV. Proposed nMPRA-MT architecture and hardware support

The nMPRA-MT architecture implemented for n tasks, called Multi Pipeline Register Architecture - Fine-grained Multithreading, extends the nMPRA project presented in [9] and [10]. Our implementation classifies the high priority real-time tasks as hard threads–HT and low priority tasks as soft threads–ST [11]. The HTs are threads that need hardware-based isolation and the STs represent threads for which the results produced after the deadline do not cause a critical effect. To guarantee the predictability, we propose a new dynamic hardware-based real-time scheduler that executes HTs every two clock cycles; completely eliminating stalls from the pipeline assembly line. Thus, nHSE offers hardware-based isolation for HTs and schedules the STs to use the free CPU cycles. Introducing a new block in the Instruction Decode pipeline stage that compares two 32-bit values and outputs the information on them, the forwarding unit is able to eliminate any delays caused by hazards at the expense of a larger execution latency of two clock cycles per instruction [11].

The new nMPRA-MT architecture uses the basic idea of the FlexPRET processor presented in [7], interleaving HTs and STs into a five-stage pipeline assembly line. Unlike this architecture, nMPRA-MT implementation is based on the remapping process of task contexts which is totally transparent to the user, generating only a 3 clock cycles over-control when the software application uses sw instructions. If applications do not use this type of instruction then the overcontrol is zero.

Beside the processor-coprocessor type architectures, the nMPRA-MT architecture has the major advantage of not needing additional time for pipeline arbitration and nor does it generate delays due to the data transfer between the scheduler and the processor, because the scheduler is an integral part of the processor. Compared to other schedulers implemented in dedicated processors, nHSE has the advantage of processing external or interrupts events in the hardware scheduling block without generating additional overcontrol.

A. *Pipeline and Thread Scheduling*

Because most operating systems use the stack to save the function contexts and because the stack is a common memory zone, it is not excluded that during the course of the application this stack will be accidentally corrupted and the contexts of certain functions will turn inconsistent. Most dedicated microcontrollers implement as a safety measure SEC-DED (Single Error Correction, Double Error Detection) control codes to detect and correct certain errors for the Flash and RAM memory [12]. But these mechanisms perform just a verification of the data integrity and they do not guarantee their consistency, too. The corruption of certain contexts may be successfully detected only if there are mechanisms which could check overall variables and memory contexts from both points of view. AutosarOS partially implements this requirement, using mostly memory protection mechanisms so as to restrict access to memory from unauthorized code areas. It is for this reason that the project proposed by nMPRA-MT based on resource multiplexing normally performs task contexts remapping in just one clock cycle or in 3 clock cycles when we wish to ensure memory consistency. Guaranteeing a complete isolation of hardware contexts for HTs, we can say that project nMPRA-MT is a feasible implementation for small real-time applications.

Figure 2 exemplifies how nHSE introduces on the assembly line on 1st, 3rd and 5th stages a HT thread noted with τ_0. In performing this scheduling, in the detriment of the latency of an instruction at two clock cycles for HT thread execution, we avoid the worst cases of assembly line stalling. On the 2nd and 4th levels we introduced another HT thread using the round-robin scheduling algorithm, managing thus to fully feed the pipeline. The nHSE scheduler manages the assembly line threads according to their type, HT or ST and controls the introduction without penalities of a new task on the assembly line, taking advantage of the clock cycles still unused. In case data hazard appears, the hazard detection unit performs data redirection for each separate thread.

The state of each thread scheduled by the nHSE, is stored in a special STATE register. Every thread τ_n has its own state that can be IDLE, SLEEP or RUN, and a unique identifier (ID) which is an integer number from 0 to n-1, that identifies each thread with different priorities on the occurrence of an event. The scheduling algorithm implemented by nHSE executes all HT or ST threads according to the scheduling scheme that have passed the feasibility test. In what concerns the example shown in Figure 2, the τ_0 thread depicted in red has the highest priority, having the ID equal to 0, while the τ_1 thread depicted in blue is also a HT thread with the ID equal to 1. Thereby, the lowest priority thread, τ_{n-1}, corresponds to a ST thread having the ID equal to n-1. The STATE and ID registers of the active threads, the block dedicated to the events and synchronization logic, together with the dynamic scheduler, constitute the actual state of the processor. Based on these registers, every thread may have a priority on a scale from 0 to n-1, where 0 is the highest and n-1 is the lowest priority, and a state stored in the STATE register. The state of each thread can be changed by a dynamic scheduling algorithm implemented in the nHSE.

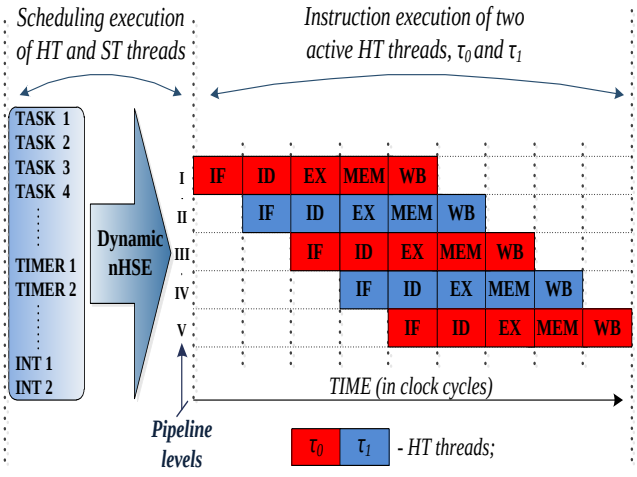

Figure 2. Fine-grained nMPRA-MT pipeline architecture and Dynamic nHSE scheduler, IF-Instruction Fetch stage, ID-Instruction Decode stage, EX-EXecute stage, MEM-Memory stage, WB- Write Back stage.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

The scheduler is a finite state machine (FSM) implemented into the hardware that can validate or suspend the execution of a HT or ST threads, based on the implemented scheduling algorithm. If an external event is attached to a HT thread, on the occurrence of this event the nHSE identifies and executes the thread identified by using the ID register [12]. If there are no events or HT threads active, the nHSE must execute the ST threads in order to eliminate the wasted CPU cycles.

B. *Inter-task synchronization and communication*

The hardware implementation of inter-task synchronization and communication mechanisms does not introduce an unacceptable increase of the overall WCET. Thus, the proposed nMPRA-MT architecture is able to execute multithreaded workloads using the above mentioned mechanisms, increasing the efficiency of implementation that support parallel execution of HTs combined with additional STs. In order to do this, a set of registers with fast access is used to implement the synchronization and communication mechanisms [10].

The hardware support for the synchronization mechanism is represented by Mutex Register File (MRF). This special registers contains the mutex state and the identifier of the proprietary thread. Sharing resources for all HTs and STs imply that mutex registers (MR) can be accessed from any thread under the control of the nHSE.

In order to avoid the priority inversion problematic scenario, the priority of a ST thread can be changed dynamically by a dynamic scheduling algorithm implemented in the nHSE, but cannot become an HT thread. Each thread has a hardware block implemented in the nHSE that generates signals stored in the Enable Mutex Register (EMR) every time whenever MRF is modified. When one or several threads wait for a mutex, the nHSE verify the EMR and MRF every clock cycle based on the scheduling algorithm implemented in the hardware.

However, from a hard real-time point of view, it is important to avoid resource sharing among HT and ST threads, because the nHSE must perform the dynamic scheduling operations introducing additional clock cycles. Therefore, a HT thread may have to wait for freeing a mutex held by a ST thread, affecting the predictability of the system and also increasing the WCET.

In order to implement the communication mechanism, nMPRA-MT architecture uses a number of *s* event registers (ER), each having (1+2*n*+*k*) bits, that compose the Events Register File (ERF). As we can see in Figure 3, every event register uses one bit to store the event state, a set of bits to memorize the ID of the source and destination threads, and the last bits are used to store the message.

When a thread sends a message using the ERF, the nHSE read the ERF signals and schedule the thread to which the message is addressed. The ERF can be accessed from any threads; therefore, they are resources shared for all type of threads. If a thread receives a message, it is necessary to identify the source of that event.

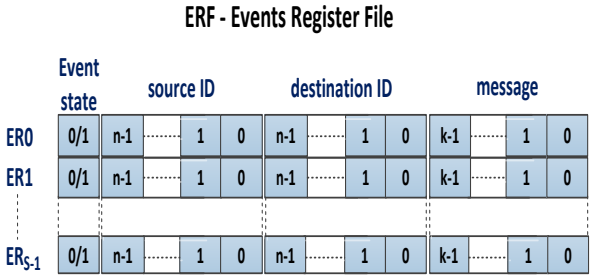

Figure 3. Events state, IDs of the source and destination tasks and the corresponding messages stored by Events Register File [10].

In order to avoid the unacceptable numbers of clock cycles used for searching in the ERF to find the source of the message, the search is made based on a Content Addressable Memory (CAM) principle [10]. Nevertheless, the nHSE implemented in the hardware can guarantee timing predictability for a set of feasible tasks, the list of HT threads being bounded by the relative deadline periods.

However, from an architectural point of view, the implementation of the synchronization and communication mechanisms is based on the atomic instruction, providing good performance in order to satisfy the individual deadline of tasks, and predictable blocking times for HT threads.

C. *Forward Unit and Hazard Situations*

For most of the real-time operating systems, the execution of the same source code in a different number of clock cycles is generated by the hazard situations appeared in the pipeline assembly line.

The proposed nMPRA-MT implementation is a fine-grained thread-interleaved pipeline architecture, capable of fetching instructions from different threads on every clock cycle. If data hazards appear when different HTs or STs are interleaved in the pipeline assembly line, a dedicated forwarding unit is implemented in order to solve all situations. In these cases, the data is redirected directly to the execution stage, avoiding the performance degradation of the pipeline assembly line. If the nHSE fetches instructions from the same HT or ST thread every two clock cycles, then hazard situations eliminated by the specific Forwarding Unit are likely to appear.

When the instructions executed in the pipeline belong to four different threads, there are no possible hazard situations. If the nHSE fetches in a continuous manner the instructions from a particular ST thread, it is possible to stall the pipeline assembly line in order to wait for the data processed by the previous instruction. In these cases, the wasted clock cycles due to the unsolved hazard situations appeared on STs execution, does not affect system predictability.

Because the HTs are scheduled every two cycles in the worst case situations preserving the predictable behavior, the proposed nMPRA-MT architecture is recommended for small real-time applications.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

V. Validation of the nMPRA-MT architecture

This section describes the operations carried out in order to test and validate the nMPRA-MT concept. This architecture was implemented and validated using a FPGA Virtex6 on a ML605 reference board, produced by Xilinx, as we can see in Figure 4. The processor code was RTL described in the standard Verilog 2001. At first, various implementation options of the nMPRA-MT processor have been validated for a 10MHz, 50MHz and 75MHz operation frequency. For validation, three scheduler versions were used. The first processor version supports the schedule of 4 tasks, the second supports the schedule of 8 tasks, while the last one of 16 tasks, all of them validated for 50MHz operation frequency. The generation of some intermediate implementations of 4 and 8 tasks respectively was necessary first of all due to the large period of time needed for the synthesis using Xilinx Design Suite 14.5. development tools, but also to detect the use of resources for various processor configurations.

The nMPRA-MT architecture uses pipeline registers replication, of the program counter and register file summing up to 1.08 kB for each thread. Multiplexing this resource for 16 tasks requires 17.25 kB of RAM memory.

A. *The Impact of Various Configuration Models on FPGA Resources*

Although the nMPRA-MT is a resource multiplexing architecture, the costs for the implementation of such a project are convenient compared to other commercial architectures. Still, we should mention that such an implementation would make sense for a reasonable number of tasks.

The implementation of this architecture for a large number of tasks would call for the synthesis of a logic in which the propagation time would be groundlessly high and thus the working frequency would significantly drop.

Table I presents the memory needed for three possible implementations of 4, 8 and 16 tasks where the maximum nesting depth of functions is on 8 levels.

Figure 4. HT thread jitter measurement on nMPRA-MT processor using Virtex-6 FPGA ML605 Evaluation Kit – Xilinx.

We have chosen these values to specifically present the memory consumption for an extreme version of the implementation: 16 tasks and an call depth of the register file on 8 levels. Taking into consideration the data presented in Table I, we may surely state that the memory needed to implement the nMPRA-MT processor is more than acceptable, given the fact that the microcontrollers used nowadays employ hundreds of KB of RAM memory.

TABLE I. MEMORY REQUIREMENTS FOR DIFFERENT CONFIGURATION OF THE NMPRA-MT PROCESSOR

nMPRA-MT configuration (number of tasks):	Memory required for:		Total memory required for nMPRA-MT (including PC)
	pipeline registers	general registers	
4	0.295kB	4kB	4.31kB
8	0.59kB	8kB	8.62kB
16	1.18kB	16kB	17.25kB

The resource that uses most memory is the register file and the pipeline registers. In the testing version specific to the nMPRA-MT architecture, the register file includes separate contexts for the tasks as well as for the functions calls. This allows us to get an additional speed gain which results from eliminating the need to save and read stack parameters. In the case of a general usage application, we may state that the nesting of functions calls depends very much on the type of application. A task may be divided into code areas which use intensively the processor, mixed code which uses the processor and the peripheral devices and code which accesses mainly the peripheral devices. For these types of codes, the common number of nested functions calls would be around 8, 6 and 4 respectively. Taking into consideration that this architecture does not need an operating system whose functions would overload the stack, the implementation with a nesting depth on 6 levels would be a reasonable solution. If a function context has 32 registers on 32 bits, the implementation of a register file which would ensure independent contexts for 6 nested functions for 16 tasks would need 12.28 kB of RAM memory.

Table II presents the energy consumption for the three implementations with hardware support for 4, 8 and 16 tasks respectively. Static power represents the energy consumed by FPGA at start-up, when the device is configured with the operating logic, but lacking any clock activity. This value is the sum of the static power value, which is the energy consumed by the device at start-up but lacking the application logic, and the design static power, representing the energy consumed of the implemented logic lacking any clock activity.

TABLE II. POWER SUMMARY FOR DIFFERENT CONFIGURATION OF THE NMPRA-MT PROCESSOR

Nr. of tasks	Clock +Logic	Signals +IO	MMCM	DSP	Static power	Total mW
4	27.10	34.58	77.30	0.74	3425.89	3565.61
8	55.93	50.68	77.30	0.77	3427.29	3611.97
16	87.32	61.15	77.30	0.75	3428.60	3655.13

The flip-flop (FF) and lookup-table (LUT) usages for three possible implementations are shown in Figure 5, the percentage indicating only for information purposes the components usage degree for the FPGA equipment used (Xilinx Virtex-6 xc6vlx240t-1ff1156), depending on the nHSE and the synchronization and communication mechanisms included.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

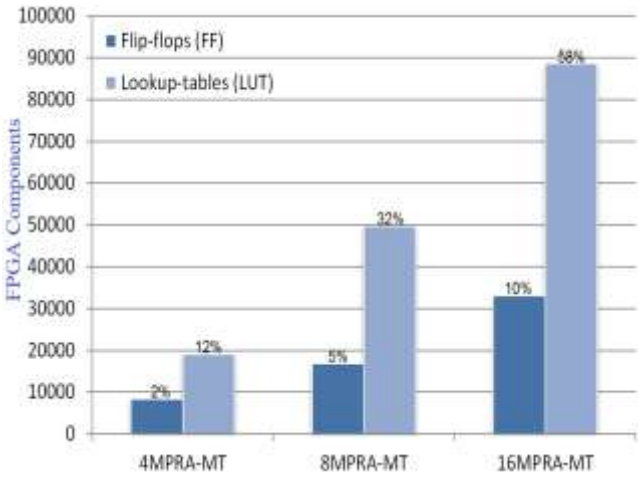

Figure 5. FPGA resource usage for different nMPRA-MT processor configurations.

As an illustration, the first implementation with 4 threads is named 4MPRA-MT, the second 8MPRA-MT corresponds to 8 threads and the last one, 16MPRA-MT has 16 threads, all of them with a nesting depth of functions on 4 levels. The resource difference between the original nMPRA version presented in [10] and 4MPRA-MT shows the cost of fine-grained multithreading and flexible thread scheduling, with a 35% increase in FFs and a 6% increase in LUTs.

The resource difference shows the cost of adding of a new forwarding module and control logic that requires more multiplexing based on thread IDs and STATE bits stored for each thread. The increase in LUTs is caused mainly by the additional control logic necessary for scheduling in the predictable mode HTs every two clock cycles. The eight thread version 8MPRA-MT is recommended for mixed-criticality applications, where hardware-based isolation is necessary. In this case, the cost of processor areas gains a more complex functionality, such as different memory hierarchy, more mutexes or peripherals.

B. Experimental Results and WCET Analysis

The system clock has been designed with the help of the LogiCore Clocking Wizard 3.6 generator from the ISE Design. For testing, the PLL block was powered by a differential input signal of a 200MHz oscillator, capable of generating a variable clock frequency ranging from 10 to 700 MHz. For the frequencies used in testing the processor, this PLL block has generated a maximum 120ps jitter. This jitter must be taken into consideration when determining the time limitations used for tracking the routes through FPGA. A major advantage of using the PLL generating block is that it implicitly global clock buffers (BUFG) from Virtex6 which are specific to clock generation, the main characteristic in generating the clock.

The main feature of these blocks is that they support a large fan-out (more than 100.000), necessary to clock tracking through all logical items of the processor. In implementing the nMPRA-MT version with 8 tasks, the fan-out reached the value of 18.000. The third channel marks the answer of the scheduler to an asynchronous external event acquired from pushing a button from the ML605 board.At a 10 MHz frequency, the jitter may be of maximum 133.5 ns, depending on the moment of the event. The internal logic of the nHSE block requires at most 33.5 ns to perform the context scheduling and remapping frequency.

As we can see in Figure 6, for that case in which the program executes sw instructions at an operating frequency of 10MHz, the jitter of the nMPRA-MT scheduler is of 340 ns. This happens because the clock used in the synchronization of the nHSE scheduler is out of phase with 240 degrees as compared to the processor clock used in memory and pipeline registers synchronization [10]. Due to the impedance inadequacy among the output levels of the Virtex6 architecture and the input level of the oscilloscope, we may detect some reflections that can be also vied in Figure 6.

In order to calculate the WCET of our processor design, we used the static analysis organized in two stages. For the beginning, we used the Fibonacci sequence to test our fine-grained multithreading nMPRA-MT implementations. This test bench, however, did not use all the instructions implemented in the nMPRA-MT architecture.

The second test bench we have created loads four values from memory and uses them to obtain the WCET for a feasible set of HT and ST threads. In some situations, when an HT thread is executed every two clock cycles in order to meet the deadline, some instructions depend on the previous one, while some do not. Each hazard situation is solved by the dedicated forward unit, obtaining the WCET for the second test bench. At the end of the tests, we can read the results stored in memory in order to check if we obtain the correct values.

Even when using the isolation, the computed WCET for the HT threads can be difficult to bind because the predictability of the architecture depends, at the same time, on the scheduling scheme implemented in hardware by the nHSE, pipeline ordering, synchronization and communication mechanisms. Due to a dynamic interaction between the various threads executed at the same time on the pipeline assembly line, the WCET is difficult to obtain because the real WCET significantly differs from the computed execution times.

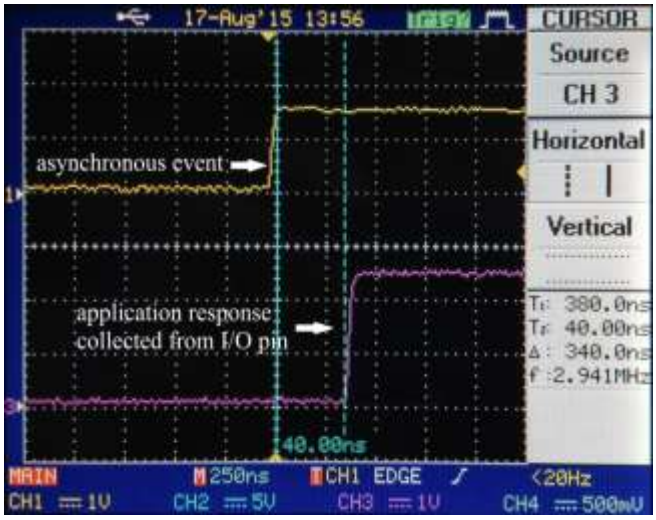

Figure 6. Jitter of the highest HT priority thread in relation with the assigned external asynchronous event.

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

C. *Design Methodology*

For implementation purposes, the nMPRA-MT project was subdivided into modules which were previously created and tested, being brought together to obtain a functional processor. The nMPRA project presented by Gaitan et al. in [10] had a similar design to the fine-grained multithreading architecture presented in this paper. For this reason, we decided to continue the nMPRA design, designing a new Hazard Detection Unit, a Forward Unit, and then make substantial changes to the data path, control, and pipeline registers. The nHSE was also completely different from the nMPRA design, and thus a new dynamic nHSE was designed. Once all the modules were completed and tested, we obtained the CPU module that joins all of the individual components with logic.

In Verilog, all modules operate simultaneously, being a challenging step to make changes in the data path and adding a new dynamic scheduler. This means that we had to make sure that our control unit signals had to command the correct modules at the correct time. The register file and pipeline registers were clocked on the positive edge and the nHSE on the negative edge.

Once all of the pipeline modules were created and fully debugged, we executed the test programs in order to debug our working nMPRA-MT processor and make sure that the hazards were being handled correctly, complying with all the instructions. Our implementation had to be able to simultaneously work on five different instructions on the pipeline assembly line at the same time. Therefore, we had to detect if there appear any hazard situations if the nHSE schedules a HT or ST thread at every two clock cycles, or if a branch instruction was correctly evaluated.

In conclusion, as a future work we will calculate a tight WCET for complete benchmarks and industrial applications running on the nMPRA-MT processor with both WCET tools.

VI. Conclusion and future work

The nMPRA-MT concept leads to all pipeline registers multiplexing so as to ensure complete isolation of software contexts and of hardware contexts too, which comprise internal signals of the processor according to the HT or ST executed instructions and thread. Therefore, the implementation proposed in this paper is a deterministic architecture as compared to SMT processors, which may expose additional overcontrol if the program does not expose an ILP (Instruction Level Parallelism).

Complete isolation of contexts and the unique time of response to events are different characteristics of nMPRA-MT architecture, making it possible to use this project in small dimensions mixed-criticality real time systems, ensuring the real time behavior necessary to such systems.

In order to improve functionality as compared to original implementation [9], the processor includes a solid support for the protection and operation using critical resources for HT and ST threads. The implementation of these mechanisms in hardware capable of operating in the same time interval with the scheduling algorithm, contributes to designing a more powerful processor architecture from its functionality point of view, without reducing its response times.

In conclusion, we may say that using a nMPRA-MT architecture with 16 tasks is fully accounted for, due to the benefits it brings, while implementation cost/effectiveness increases. Table I presents the memory necessity for various configurations of the nMPRA-MT architecture just as it was described in section V.

The architecture proposed in this paper may be improved. In the nMPRA-MT architecture, the dimensions of the memory consumed for the implementation of the register file is proportional to the number of HT and ST threads. It would be useful for the ST thread dedicated memory to be used as general usage memory with quick access, thus reducing the total architecture implementation cost. Otherwise, introducing HT dedicated scratchpad memories and the implementation of an efficient memory controller would bring significant improvements to architecture predictability and a better WCET.

Another interesting idea would be to develop a scheduler which would also support multi-core nMPRA-MT architectures. This approach would bring more dynamism in practice, offering the possibility to use the processor in performing multi-core processes or distributed systems.

Acknowledgment

This paper was supported by the project "Increasing the competitiveness of the EURONEST ICT&Hub Regional Innovation Cluster and stimulating interactions between members to develop high tech products and services" - Contract no.: 1CLT/800.020/19.05.2014, project co-funded from European Social Fund through Sectorial Operational Program Increase of Economic Competitiveness 2007-2013.

References

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

[2] Kumthekar, B.; Benini, L.; Macii, E.; Somenzi, F., "In-place power optimization for LUT-based FPGAs," Design Automation Conference, 1998. Proceedings, pp.718,721, 19-19 June 1998.

[3] Biondi, A.; Melani, A.; Marinoni, M.; Di Natale, M.; Buttazzo, G., "Exact Interference of Adaptive Variable-Rate Tasks under Fixed-Priority Scheduling," Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on , vol., no., pp.165,174, 8-11 July 2014, doi: 10.1109/ECRTS.2014.38.

[4] OSEK. OSEK/VDX Operating System Specification 2.2.1. OSEK Group, http://www.osek-vdx.org, 2003.

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

[6] Wolf, J.; Gerdes, M.; Kluge, F.; Uhrig, S.; Mische, J.; Metzlaff, S.; Rochange, C.; Cassé, H.; Sainrat, P.; Ungerer, T., "RTOS Support for Parallel Execution of Hard Real-Time Applications on the MERASA Multi-core Processor," Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on , vol., no., pp.193,201, 5-6 May 2010, doi: 10.1109/ISORC.2010.31.

[7] 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.

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-29

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

[9] 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.

[10] 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.

[11] Gaitan, Nicoleta Cristina; Zagan, Ionel and Gaitan, Vasile Gheorghita, "Predictable CPU Architecture Designed for Small Real-Time Application - Concept and Theory of Operation" International Journal of Advanced Computer Science and Applications(IJACSA), 6(4), 2015.

[12] Zhu Ming; Xiao Li Yi; Luo Hong Wei, "New SEC-DED-DAEC codes for multiple bit upsets mitigation in memory," VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on , vol., no., pp.254,259, 3-5 Oct. 2011, doi: 10.1109/VLSISoC.2011. 6081647.

[13] Gaitan, Nicoleta Cristina; Gaitan, Vasile Gheorghita; Ungurean, Ioan; Zagan, Ionel, "Methods to Improve the Performances of the Real-Time Operating Systems for Small Microcontrollers," Control Systems and Computer Science (CSCS), 2015 20th International Conference on , vol., no., pp.261,266, 27-29 May 2015.

[14] B. Meakin, G. Gopalakrishnan, "Hardware Design, Synthesis, and Verification of a Multicore Communication API ", SRC TECHCON, 2009 - cs.utah.edu.

[15] Dodiu Eugen, "Real-Time Hardware Scheduler for FPGA Based Embedded Systems", Ph.D. dissertation, University Stefan cel Mare of Suceava, Romania, 2013.

About Authors:

Ionel Zagan received the Eng. degrees in computer science from the Stefan cel Mare University of Suceava, Suceava, Romania, in 2005. He is currently a Ph.D. student with the Department of Computers, Electronics and Automation, Stefan cel Mare University of Suceava Romania. His current research interests include real-time systems, microcontrollers and pipeline processors with parallel execution of tasks. Mr. Zagan is a member of the IEEE Computer Society.

Vasile Gheorghita Gaitan received the MS and PhD degrees from the „Gheorghe Asachi" Technical University of Iasi, Romania in 1984 and 1997, respectively. He is currently professor with the Computers, Electronics and Automation Department, "Stefan cel Mare" University of Suceava, Romania. His main research interests include real time scheduling, embedded middleware, digital systems design with FPGAs, fieldbuses and embedded system application. He is a member of the IEEE and a member of the IEEE Computer Society.