Impact in the Parallel Processing of IHM-Plasma Using the Earliest-Deadline-First Algorithm for the Task-Scheduler Realized by Hardware

Igor Krause¹, Leandro P. Dantas², and Salvador P. Gimenez¹

¹Igor Krause and Salvador Pinillos Gimenez, Fei University Center, São Bernardo do Campo, Brazil
²Leandro Poloni Dantas, INSPER/SENAI, São Paulo, Brazil
e-mail: eng.igor.krause@gmail.com

Abstract—This work studies the impact in the parallel processing of the Interlocked-Hardware-Microkernel (IHM) Plasma microprocessor (IHM-Plasma) by implementing the Earliest-Deadline-First (EDF) algorithm by hardware in the task-scheduler block that belongs to those task-based operational systems, such as that related to the real-time operational systems (RTOS). IHM-Plasma presents Reduced Instruction Set Computer (RISC) architecture. Results show that the EDF algorithm running in IHM-Plasma’s hardware has increased the number of tasks executed per second by up to 174% compared to the same algorithm running in software. The developed work has great potential use in Hard Real-Time Systems and others where a rigid control of deadlines is essential and hold many tasks.

Index Terms—Microkernel by Hardware; Parallel Processing; Microprocessors Earliest-Deadline-First Algorithm.

I. INTRODUCTION

A Real-Time Operating System (RTOS) is responsible for the task-scheduling, dispatching, and communication between tasks of a Real-Time System (RTS). The scheduler defines the tasks order of execution. The dispatching performs the hardware resources relocation for the next task (context switching). The context switching is a key factor in real-time scheduling algorithms, because it may hold the Central Processing Unit (CPU) to immediately run the higher priority tasks [1].

The objective of a RTOS is to provide to the system real-time behavior, satisfy temporal requirements, and allow multiple tasks to be run continuously, regularly, and robust [2]. Nowadays, RTOS is an indispensable component for building RTS, once time-critical applications are becoming more present in the systems, making it difficult to guarantee its real-time performance [3].

One approach seen in literature to enhance the performance of RTOS is implementing some or all RTOS’s functions into hardware [3]. That is, adding some hardware into the system that is dedicated to perform one or more functions of the RTOS, releasing the software to spend clock cycles running these functions. This approach brings many advantages to the system, such as greatly reducing the event trigger latency and reducing the scheduling overhead (the time the CPU spends running the scheduling algorithm) [4].

Several approaches have been used to implement the task-scheduler and/or context-switching of an operational system into hardware [4][5][6][7]. Researchers in the field are more and more concluding that implementing some of the RTOS components into hardware brings leading advantages to the system [4]. The implemented hardware has the capacity of increasing the parallel processing of information, which may result in decreasing the response time of the system [4].

In the study [4], the scheduler algorithm of a RTOS is hardware implemented in a five-stage MIPS architecture. In this approach, the program counter, pipeline and general-purpose registers are replicated for each task within the system.

The study [5] proposes an integration of a FPGA with the CPU. The task-switching mechanism is implemented in the FPGA. This is an interesting solution because it allows the scheduling algorithm to be swapped after implementation, which is not true in ASIC implementations.

There are also some extreme solutions like [6]. It proposes a methodology to convert the CPU and the instruction memory into equivalent hardware, which results in each task and handler being synthesized into an independent hardware module. As the tasks don’t need a CPU to be run anymore, they can be executed parallelly. A manager module controls the run/stall of the tasks and handlers’ modules. The study shows that the methodology results in very low system latency, but at the cost of very large hardware consumption.

The work [1] studies the current available implementation functions of a hardware scheduler in RTS. It also analyzes different scheduling algorithms that can be implemented in hardware in order to check their behavior related to the RTS requirements.

There is even a study [3] that proposes a hardware/software co-design to improve the reliability of the system’s task-scheduling.

In the studies seen in literature, implementing some functions of the RTOS into hardware always results in the increase of the system performance. The issue in these approaches is the size of the implemented hardware: the state of art solutions proposes a copy of the Internal Register Bank (IRB) and Program Counter (PC) register for each task of the system, add new pipeline states for the CPU, and modify the instruction set of the microprocessors [1][4][5][6][7][8][9][9].

But recently, an innovative approach was developed to implement key functions of RTOS into hardware of Reduced Instruction Set Computer (RISC) CPUs, entitled “Interlocked Hardware-Microkernel” [10][11]. In this study, the new approach was implemented into a Microprocessor without Interlocked Pipeline Stages (MIPS) architecture microprocessor called Plasma, giving rise to the “IHM-Plasma” microprocessor. This solution stands out for consuming much less hardware resources (logic modules, special function registers, register banks, memories, etc.) as compared to the other similar approaches described in the literature [1][4][5][6][7][8][9], without modifying its buses and without adding any instruction to its instructions set. IHM microarchitecture was responsible for transferring functions of the task-based operational system, in which it was originally implemented by software, like task-scheduling and dispatching in hardware resources. The task scheduler hardware executes the preemptive round-robin algorithm and the dispatching the
context switching processes, in order to perform parallel processing with the intrinsic assignments of the Plasma microprocessor, aiming to boost its processing capacity in terms of reducing the time spent to execute the task-switching process [11].

Remarkable results were obtained by the IHM-Plasma in relation to those observed of the Plasma running task-based operational system by software, as for instance. The time taken to perform the task-switching process by the IHM-Plasma was 98.5% reduced in comparison to the Plasma CPU running the traditional task-based operational system by software [10][11]. To reach these results, the number of Adaptive Look-Up Table (ALUT) consumed in the implementation of the IHM-Plasma microprocessor into an Intel Cyclone V FPGA is increased 351% in comparison to the implementation of the original Plasma microprocessor.

But as the software needs to be changed in a software scheduling operational system in order to change the task-scheduling algorithm executed, the hardware needs to be changed in a hardware scheduling operational system in order to change the task-scheduling algorithm executed. In this scenario, the motivation of this study is to propose a new task-scheduler block by hardware for the IHM-Plasma running the Earliest-Deadline-First (EDF) algorithm, in order to allow the IHM Architecture to perform another option of scheduler algorithm and verify the impact in its processing performance in relation to the Plasma microprocessor running the traditional task-based operational system by software.

II. IHM-PLASMA ARCHITECTURE

The IHM-Plasma was proposed to boost the processing performance of the Plasma microprocessor by using the parallel processing concept when it is used to run task based operational systems, such as the Real-Time Operational Systems (RTOS) [11].

In a conventional task-based operational system, the microkernel controller is responsible for the task-switching process and the task-scheduler is responsible for deciding what will be the next task to be run, usually depending of its priority. Both processes are performed by software running different algorithms, that define part of the microkernel of the task-based operational system. Thus, usually the tasks to be run can be interrupted before completing their executions, in order to the microprocessor can execute another task. So, to avoid mainly the data overwriting in those same special function registers, in those same registers of the register bank and in those same contents of the memories that are used by the different tasks and consequently generating fails in the electronic system to be controlled, usually each task uses in a separate memory block, entitled “Task-Control-Block (TCB)”, in order to store the context of the task (values of the registers and content of that memories that define the conditions of the task, when it was interrupted) [10][2].

Besides, one of the bottlenecks of task-based operational systems (software solution) is the time spent to run the task-scheduling algorithm periodically, whose main objective is to verify continuously the priorities of the tasks in order to decide what of them must be run after the finishing of the execution of the current task [2]. Another bottleneck of task-based operational systems is the time that it takes to perform the context switching in order to enable the next task chosen to be run. During this process, the CPU is responsible for saving the conditions (values) of registers of the Internal Register Bank (IRB) and Program Counter (PC) into the corresponding TCB of the next task to be run to the registers of IRB and PC to that it can now be run [2].

In order to try to overcome these bottlenecks, IHM microarchitecture has proposed the use of another IRB, working in parallel to that already existent. While one of them is being used by CPU to run the current task, the other is being used by IHM to receive the data of the context of the next task to be run, in which are obtained of its respective TCB, in which were previously stored on it by the IHM. After that, the task switching process occurs, in which CPU is paused, in order to IHM selects and makes available the other IRB, in which contain the context of the next task to be run, so it can run the next task to be run taking into account the correct context of this task. Besides, at the same time, IHM uses the IRB that was being used by the CPU to again to define the new context of the next task to be run and so on [10][11].

Comparative studies between the Plasma and IHM-Plasma have shown that IHM-Plasma (hardware solution) was able to reduce the context switching time to 2 clock cycles, while that performed by the Plasma (software solution) was of 136 clock cycles and thus providing more time for CPU to perform the tasks than carry out the processes of identifying the next task to be run and tasks switching [10][11]. Therefore, we can conclude that the use of parallel processing concepts by transforming part of the functions of task-based operational system, that is done by software, in hardware resources that are capable of performing these same functions, can be used in order to remarkably improve the processing performance of the microprocessors. Fig. 1 illustrates the task-switching processes as a function of time in clock cycles, being make by the Plasma CPU, running a task-based operational system by software (Fig. 1a), and by IHM-Plasma CPU, using parallel processing concepts (hardware solution) (Fig. 1b), regarding 3 tasks.

It is important to highlight that the IHM-Plasma was conceived by adding three main logic blocks to the Plasma microprocessor: I- Task Scheduler Block; II- Context Manager Block; III- Time and Control Block.

A. Scheduler Block

The scheduler block is responsible for deciding what is the next task to be run by CPU. The scheduler block analyzes the priorities of all tasks that must be run and it uses the Round-Robin scheduling algorithm. For each task, it is created a vector holding its priority, and these vectors are compared withing a “binary tree”. These vectors are composed by 21 bits. These 21 bits are divided in 4 parts: I- The first part is composed by the bit 0
Fig. 1 Task-switching processes performed by the Plasma CPU (software solution) (a) by the IHM-Plasma CPU using parallel processing concepts (hardware solution) (b)

informing if the task is active; II- The second one is composed by the bits from 1 to 12 and informs the time the task is waiting to be run; The third part is composed by the bits from 13 to 15 and indicate the priority of the task; The last one is composed by the bits from 16 to 20 and indicate the number of the task which the vector belongs. The binary tree of the IHM-Plasma compares the value in the bits 0 to 15 of the vectors, and defines the task with biggest value in these bits to be the next task run [10][11]. Fig. 2 illustrates how the vector analyzed by the binary tree of IHM-Plasma is defined.

B. Context Manager Block

The context manager block is responsible for saving the context of the current task at the moment it was interrupted so that another task can be run. Besides, it is responsible to restore the context of the next task to be run by the CPU from the corresponding TCB, that was previously saved in the respective TCB by the IHM, when this task was interrupted so another could be run. The context of each task is composed by the values of the registers of IRB and PC that were defined when it was interrupted so that another one could be run. The context manager block uses a finite state machine (FSM) to control data flow during the storing process (backup) of the values of the registers of IRB and PC at the moment that a task was interrupted in its respective TCB and another FSM to control the data flow during the process of reading the values of the respective TCB to posteriorly write them in IRB that was selected by the IHM to CPU can run the next task [11].

C. Time and Control Block

The time and control block is responsible for performing the communication between IHM, the Plasma CPU and the random access memory (RAM), that is used to store part of the task-based operational system of IHM-Plasma in order to we can run it and perform the experiments to verify its functioning. This block runs the Round-Robin algorithm to perform all activities of the IHM (context restoration, context backup, and task switching) [11].

The time and control block has a third FSM, entitled “state_motor”. It is responsible for controlling all the activities of the IHM (management of the context information, pausing the CPU, IRB switching, task-switching, etc.) [11]. Fig. 3 illustrates the state diagram that defines the FSM “state_motor”.

D. Main Plasma CPU Changes to implement IHM-Plasma

First of all, the PC electronic circuit of the Plasma microprocessor was changed in order to IHM block could define the address of the next instruction to be run at the moment the IHM-Plasma perform a task-switch [11]. Fig. 4 illustrates the fourth input, represented in dashed lines, that was added to the multiplexer of the PC electronic circuit, in order to IHM can define to the Plasma CPU the address of the next instruction to be run, that was saved in the TCB corresponding to the next task to be run.

In Fig. 4, the clk signal indicates the clock cycle, the D and Q input and output indicate the input and output of a D flip-flop, and the ALU indicates the Arithmetic Logic Unit from the Plasma CPU.

The second modification in the Plasma CPU in order to conceive IHM-Plasma was the implementation of a second IRB, in which it is identical to that already existing in the Plasma CPU. While one of them is used by the Plasma CPU to run the current task, the other one is used by IHM block,
in the same time and performing parallel processing by hardware, in order to it can store the context of the next task to be run by the Plasma CPU [11]. Fig. 5 illustrates the new IRB with its respective electronic circuit that were added to the original IRB of the Plasma CPU, represented by the light gray.

In Fig. 5, the A1 and A2 inputs define the addresses of the registers the IRB are going to read and exhibit their values on the outputs RD1 and RD2. The A3 input defines the address of the register the IRB is going to write the value defined by the input WD3 if the input WE3 is active.

E. Ram Modifications

It was implemented nine new SFR into RAM of the IHM-Plasma to help control and configure IHM [11]. Tab. 1 presents the name and the purpose of each SFR added in the IHM-Plasma.

The RAM was divided in three parts aiming the organization of the data according to their functions. The first and second parts store data and software and are accessed only by CPU. The third one is accessed by the CPU when the IHM is disabled, storing data and software, and is accessed by the IHM when the IHM is enabled, storing the TCB of the tasks of the operational system [11].

III. RTM-PLASMA ARCHITECTURE

The Real-Time Microkernel (RTM) architecture is based on IHM architecture. It was conceived to allow the IHM Architecture to perform the task-scheduling according to another option of algorithm. This study replaced the task scheduler block of the IHM architecture with a new block that runs the Earliest-Deadline-First (EDF) algorithm. In this scheduling algorithm, the task’s priority is inversely proportional to the remaining time to meet its deadline (time limit by which a task should be completed). It is an algorithm more focused on Hard Real-Time Systems, where the miss of a deadline may cause catastrophic damage to the system. Thus, besides implementing a new task scheduler, it was also necessary to change the “time and control” block [11][2].

A. Task Scheduler Block

In the RTM architecture, the deadline of each task is established by the software, through the use of a new SFR entitled “task_deadline”. This SFR is defined by 32 bits. This 32 bits are divided in 4 parts: I- The first part is composed by the bits from 0 to 24, and contains the value of the desired deadline in clock cycles; II- The second one is composed by the bit 25 that informs if the task is periodic or aperiodic; III- The third part is composed by five bits (from 26 to 30, defining 32 tasks in the maximum) in which is responsible for defining what is the task of 32 possible; IV- The last one is defined by the bit 31, that is responsible for enabling the assignment of the deadline to the corresponding task. Fig. 6 illustrates how the “task deadline” SFR is defined.

The control of the deadline of a task is made using a register and a counter down for each task. The register is defined by the SFR “task deadline”. Every time that a task is activated or restarted, the register updates the value of the counter down. The counter down holds the remaining time for the task to meet its deadline. It is decreased of one unit every clock cycle. Fig. 7 shows a simplified schematic of the deadline controller of a task.

For each task, it is created a vector holding its priority, and these vectors are compared within a “binary tree”. These vectors are composed by 30 bits. These 31 bits are divided in 3 parts: I- The first part is composed by the bits from 0 to 24 and contains the remaining time that the task still has to meet its deadline, in clock cycles; II- The second one is composed by the bit 25, that is responsible for informing if the task is active or not; The last one is composed by the bits from 26 to 30, in which they indicate the number of the task which the vector belongs. The binary tree concept is used to RTM-Plasma. It is responsible for performing comparisons between the value of the bits from 0 to 25 of the vectors of different tasks in order to define what is the
task that presents the lowest value. That task that presents the lowest value is defined as that with the highest priority. Fig. 8 illustrates how the vector analyzed by the binary tree of RTM-Plasma is defined.

The new scheduler block implemented in the RTM-Plasma has a flag intitled “change_task” that is responsible for informing to the time and control block that there is a task with a higher priority than the current task that are being run by CPU. If this flag is activated, the RTM architecture performs the task-switching.

B. Time and Control Block

In the IHM architecture, the context restoration is started in some clock cycles before the end of time-slice. The time-slice is an interruption generated by a timer that defines the period of time the CPU must execute a task before performing a new task-switching. The lower this interruption’s period, faster is the system’s response and better is the deadline control of the system, but reduces the time left for the CPU runs the tasks of the operational system. The task-switching process in the IHM-Plasma is predetermined to occur at the end of time-slice.

RTM-Plasma does not use time-slice to control the task-switching. The context restoration of a task starts after the context manager block finishes the context backup. The task-switching process happens at the moment that the scheduler block sends the electric signal “change_task” informing there is a task with higher priority than the current task that is being run by the CPU. The next task is defined the moment the context backup is over. The task-scheduling performed by the RTM-Plasma IHM (hardware solution) has a better response time and deadlines control because the task-scheduling is updated every clock cycle. Once started the context restoration of the next task, it will necessarily be the next task executed by the CPU [11].

The original FSM “state_motor” was also changed to suit the behavior of IHM architecture to implement EDF algorithm. Fig. 9 shows the new state diagram of the new FSM “state_motor” that is used in the RTM-Plasma.

C. Additional IHM-Plasma Changes

The RTM architecture has a signaling electronic system that generates an interruption to the microprocessor when a task is not finished within its time limit (deadline). This allows to the programmer to develop an interruption handling more appropriate for each system.

The SFR are changed to suit the algorithm EDF used by the RTM-Plasma. One SFR is added in order to establish the deadline of the tasks, and another SFR is added in order to inform if a task did not meet its deadline. Tab. 2 presents the SFR that were added to RTM architecture, and describe their functions.

<table>
<thead>
<tr>
<th>Field</th>
<th>Task</th>
<th>State</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>31</td>
<td>30</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 6 Format of the SFR “task_deadline”

IV. RESULTS

The RTM architecture was implemented based on IHM-Plasma microprocessor, creating the RTM-Plasma. Its experimental electrical characterization and its real functioning were performed in FPGA, through the didactic kit “DE1-SoC” from Terasic®, which contain a FPGA “Cyclone V” from Intel® [12][13], running a task-based operational system to perform comparative studies between the task-switching performed by the Plasma CPU (software solution) with the task-switching performed by the RTM-Plasma hardware (hardware solution).

The objective in the simulations is to compare the time the CPU is free to run the operating system’s tasks and the time the CPU must spend running task-switching functions. As what the task performs itself is not what we want to measure but the time it is being run by the CPU, the tasks used in the operational system are very simple and run in a loop. It
begins in a long loop running no operation, which simulates the CPU running some instructions. When this first loop is over, the task turns on a GPIO pin to indicate to the oscilloscope that the task was run, enter a small loop performing no operation to allow enough time for the oscilloscope detect the activation of the GPIO pin, and then the GPIO pin is turned off. The task is over and it jumps to the beginning of the loop.

The task-based operational system designed by this work allows the user to set it to perform the task-switching process by the Plasma CPU (software solution) or by the RTM-Plasma hardware (hardware solution). When a task is created, it is assigned the desired value for its deadline by the programmer. When the task is started or when the task is finished, the deadline value is loaded with the established value. In scheduling by software (Plasma), the deadline values of all tasks are reduced only when the CPU runs the scheduler algorithm. In scheduling by hardware (RTM-Plasma), the deadline value is reduced every clock cycle, which makes it more precise than software scheduling.

A. Task-Switching

The first study aims to experimentally measure by using the FPGA kit, the times used to perform the task-switching processes between the task based operational systems running the EDF algorithm by software, done by the Plasma CPU (software solution), and by the proposed solution by this study, done partially by hardware by RTM-Plasma microprocessor. The conventional task switching process (software solution) is performed in three steps: I- The first one, the Plasma microprocessor performs the context backup process of the interrupted current task (it reads the content of IRB of the current task and it writes them in its respective TCB); II- The second step, Plasma CPU runs the task scheduling algorithm (EDF) to decide what is the next task that will must be run by it; III- The last one, the Plasma CPU performs the context restoration, by reading the contents of the respective TCB of the next task to be run and writing them in IRB in which it is not being used by the Plasma CPU. It is important to highlight that we have tried to implement the task switching algorithm by software to be run by the Plasma CPU as similar as possible to the one implemented in the hardware of RTM-Plasma, to make the comparison process between these two approaches as fair as possible.

The results have shown that Plasma CPU is able to use 68 clock cycles to perform the context backup of the current task that was interrupt, 1170 clock cycles to perform EDF scheduler algorithm by software, and 72 clock cycles to perform the context restoration of the next task to be run by the Plasma CPU, resulting a total of 1310 clock cycles. This experiment was done considering 3, 7 and 14 tasks, respectively. Therefore, for all cases, we have observed that the quantities of clock cycles for all process done are always the same. This same process done by Plasma CPU by software was performed by RTM-Plasma in only 2 clock cycles. Tab. 3 presents the comparison of the task-switching process performed by the software solution performed by the Plasma CPU and by the hardware solution done by RTM-Plasma, respectively.

Therefore, we conclude the remarkably reduction of the time used to perform the task switching when we used the parallel processing (hardware solution). So, this approach can be considered an alternative procedure to be used to significantly boost the processing performance of the microprocessors thinking on task-based operation systems.

B. System Performance

Additional two studies were performed in order to experimentally measure the task based operation system performances of the considered approaches (software by Plasma CPU and hardware by RTM-Plasma solutions): I- The first one has aimed to measure the quantity of tasks run by second or the frequency of the execution of the tasks as a function of the time slice used by software solution; II- The second study has have the objective of measuring the quantity of tasks run by second or the frequency of the execution of the tasks as a function of the sleep time used by both solutions (software and hardware). In these two studies were considered 3, 7 and 14 tasks to be run by the task-based operation system. All tasks were identical, and their functions have performed the activation of a bit of an output port of FPGA kit and posteriorly they have put themselves to sleep. Fig. 10 illustrates the execution frequencies of all tasks when they were run by the Plasma CPU as a function of the time-slice used by software solution, regarding different number of tasks (3, 7 and 14). The continuous line represents the hardware solution and the dashed line represents the software solution.

Analyzing Fig. 10, we observe that the execution frequencies of all tasks being run by the Plasma CPU, regarding the software solution, reduce as the time slice decreases. When the time-slice is reduced to 2\(^{10}\) (1024) clock cycles, the task-based operation system run by software by the Plasma CPU is not able to run the tasks, regarding different number of tasks to be run. This occurs due to the task-based operation system run needs 1310 clock cycles to perform the task switching process and in these cases it has only available 1024 clock cycles. Besides, we can see that the minimum time-slice to be considered by the software solution is approximately 2\(^{11}\) (2048) clock cycles. In contrast to that was found by the software solution, we observe that the hardware solution by the fact of using EDF algorithm is independent of the time-slice. Furthermore, we notice that the hardware solution done by RTM-Plasma is capable of boosting the processing performance in 37% considering 3 tasks, 109% regarding 7 tasks and 174% considering 14 tasks for a time-slice of 2\(^{11}\) clock cycles.

Fig. 11 and 12 illustrates the execution frequencies of all tasks when they were run by the Plasma CPU (Fig.11) and by the IHM-Plasma hardware (Fig.12) as a function of

<table>
<thead>
<tr>
<th>Task-Switching</th>
<th>Period (clock cycles)</th>
<th>Time Reduction (times)</th>
<th>Time Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>1310</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hardware</td>
<td>2</td>
<td>655</td>
<td>99.85%</td>
</tr>
</tbody>
</table>
The RTM-Plasma CPU is capable of improving in 127% the processing performance (execution frequency of 1.52 Hz) in comparison to the one (execution frequency of 0.67 Hz) observed by the software solution done by Plasma CPU. Similar results can be found regarding 3 and 7 tasks to be run.

CONCLUSIONS

This work studies the impact in the processing performance of the IHM-Plasma, by using a tasks scheduler running Earliest-Deadline-First Algorithm, implemented by hardware, utilizing the parallel processing concepts.

The RTM-Plasma CPU is capable of improving in 127% the processing performance (execution frequency of 1.52 Hz) in comparison to the one (execution frequency of 0.67 Hz) observed by the software solution done by Plasma CPU. Similar results can be found regarding 3 and 7 tasks to be run.

ACKNOWLEDGEMENTS

The authors acknowledge FEI University Center, CNPq (Grant 307804/2019-4) for the financial support, and Mackinca DHW (Brazil) by making the FPGA kit available to performing the experimental electrical characterization.

REFERENCES


