Hardware Accelerated Scheduling in Real-time Systems

Lukáš Kohútka, Martin Vojtko, Tibor Krajčovič
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Bratislava, Slovakia
kohutka555@gmail.com, {martin.vojtko, tabor.krajcovic}@stuba.sk

***Abstract*—There are two groups of task scheduling algorithms in real-time systems. The first group contains algorithms that have constant asymptotic time complexity and thus these algorithms lead to deterministic task switch duration but smaller theoretical CPU utilisation. The second group contains complex algorithms that plan more efficient task sequences and thus the better CPU utilisation. The problem is that each task scheduling algorithm belongs to one of these two groups only. This is a motivation to design a real-time task scheduler that has all the benefits mentioned above. In order to reach this goal, we decided to reduce the time complexity of an algorithm from the second group by using hardware acceleration. We propose a scalable hardware representation of task scheduler in a form of coprocessor based on EDF algorithm. Thanks to the achieved constant time complexity, the hardware scheduler can help real-time systems to have more tasks that meet their deadlines while keeping high CPU utilisation and system determinism. Another advantage of our task scheduler is that any task can be removed from the scheduler according to the ID of the task, which increases expandability of the task scheduler.**

Keywords—FPGA; hardware acceleration; task scheduling; real-time systems; performance; task queue; coprocessor;

I. INTRODUCTION

Real-time systems represent a category of embedded systems that process real-time tasks, where success of the real-time task depends not only on the result of the computation but depends on the timing of the task too.

Even if we use a microcontroller with the highest performance, there is no guarantee that all tasks meet their deadlines. Therefore we use a real-time task scheduler that handles this problem. The ideal real-time task scheduler creates an optimal sequence of the tasks, in which the tasks will be computed and completed on time and has no overhead on CPU. More we use CPU for the scheduling, the less we use the CPU for effective computation and completion of the scheduled tasks.

This paper presents a novel real-time task scheduler implemented in a form of a coprocessor. We used an existing algorithm called Earliest Deadline First (EDF). Simulation and implementation results are shown in order to verify the correctness of the designed scheduler. We present a comparison of hardware and software implementation in terms of performance and efficiency of the scheduler.

II. RELATED WORK

Most of the existing task schedulers are implemented as a software module of an operating system. There exist static and dynamic schedulers. Static schedulers generate task order during the system initialization and dynamic schedulers generate task order during the system execution. The problem of the software implementations is that the task scheduling cannot be both fast and robust. The simple schedulers are fast but the possibility that the task set will be schedulable is smaller.

Besides existing software implementations, there exist hardware implementations too. We found that most of these solutions use the static RMS scheduling algorithm and some of them use EDF. They use architecture of task sorting based on multiplexer trees or FIFO structures which leads into issues with scalability [1-6].

III. COPROCESSOR BEHAVIOR

The hardware real-time task scheduler is designed in a form of a coprocessor unit that can cooperate with any CPU via custom instructions that take 2 clock cycles regardless of the number of the tasks in the system (see Figure 1).

The coprocessor uses EDF algorithm and thus orders the scheduled tasks according to the deadline of each task. Task with the lowest deadline should be the task that is being executed [7]. In addition to the EDF scheduling algorithm, the coprocessor allows us to remove any task from the scheduler according to the ID of the task. There are 2 instructions that are available to the user of the coprocessor:

- *task_schedule* – inserts a new task into the queue ordered according to the deadline.
- *task_kill* – removes an existing task from the queue according to the ID of the task.

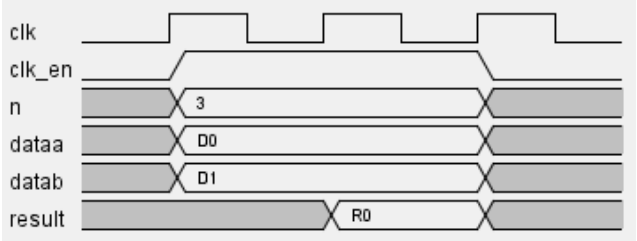

Figure 1. Coprocessor interface waveform

When we want to use an instruction of the coprocessor we

set the *clk_en* for the duration of 2 clock cycles. If $n = 1$ then the *task_schedule* instruction is used. The *result* is an output value that contains ID and deadline of the scheduled task with the lowest deadline. The ID is situated in the least significant bits and the deadline in the most significant bits of the *result*. The signal *dataa* contains input ID of a task. The input signal *datab* contains deadline of the new task and is needed for the *task_schedule* instruction only.

The coprocessor has also one pseudo-instruction called *nop* that does not change the list of the scheduled tasks but is useful when the operating system needs to know the ID or deadline of the task with the lowest deadline. We create the *nop* pseudo-instruction when we use the *task_schedule* instruction with maximum possible deadline.

IV. Hardware Architecture

The coprocessor contains two components: control unit and task queue. The control unit is responsible for generation of *nop* instruction during the second clock cycle and when the *clk_en* is zero. The task queue is responsible for sorting the tasks according the deadline.

The major part of the hardware task scheduler is the task queue. It contains tasks that are ordered by deadline. One task can be stored in one cell of the task queue and thus the number of cells in the queue represents the maximum number tasks in the system.

The scalability of the task queue in terms of area cost and performance is extremely important for the scalability of the whole task scheduler because the control unit has constant size but the size of the task queue is dependent on the maximum number of the tasks the system can handle.

In order to achieve the best possible scalability of the task scheduler, we propose a scalable architecture of the task queue which is based on parallel shift registers and pipelining approach. Each pipelining stage contains exactly K task cells that have common parallel input. The first pipeline stage receives input data directly from the current instruction and provides a task with minimal deadline as an output. Whenever we want to increase the capacity of the task queue we simply add new pipeline stages that contain the additional task cells. Thanks to this, the length of the coprocessor critical path is constant and thus it does not depend on the number of cells in the architecture. This means that the clock frequency of the whole computer system is not affected by the coprocessor with any size. In addition, the number of clock cycles needed for updating the choice of task that has minimal deadline is also constant because this task is always present in the first pipeline stage.

The chip area costs of the coprocessor depend on these parameters:

- N – maximum number of tasks in the system
- IDw – bit width of the task ID
- Dw – bit width of the task deadlines
- K – number of task cells in one pipeline stage

V. Verification and Synthesis Results

The verification and synthesis were executed for $N = 32$ tasks, $IDw = 5$ bits, $Dw = 20$ bits and $K = 8$ task cells. The verification was done by pre-layout simulations that were compared to software implemented behaviour of the task scheduler. The results of the hardware simulation are equivalent to the software output. The design was synthesized for Altera Cyclone II FPGA with these results:

- Logic elements: 4547 (7%)
- Combinational functions: 3912 (6%)
- Dedicated logic registers: 957 (1%)

VI. Conclusions

A novel hardware task scheduler with constant time complexity (2 clock cycles) is proposed. This scheduler is designed in a form of a coprocessor that implements EDF scheduling algorithm with support of "killing" any task. The designed hardware uses a scalable hardware architecture based on parallel shift registers and pipelining. The main benefits of this research are increased predictability, determinism and performance of the real-time systems, which leads to higher number of real-time tasks completed before their deadlines are met. The final effects are improved quality and reliability of the real-time systems.

We plan to test the coprocessor implemented in the FPGA as a part of a SoC that contains a Nios II soft core processor with some memory. The hardware task scheduler will be compared to the software equivalent of the same scheduling algorithm and test parameters.

Acknowledgment

This work was supported by the Slovak Research and Development Agency under the contract No. VEGA 1/1008/12.

References

[1] BLOOM, G., PARMER, G., NARAHARI, B., SIMHA, R.: Real-Time Scheduling with Hardware Data Structures, 2010.

[2] FERREIRA, C., OLIVEIRA, A. S. R.: Hardware Co-Processor for the OReK Real-Time Executive, 2010.

[3] CHANDRA, R., SINNEN O.: Improving Application Performance with Hardware Data Structures, 2010, ISSN 11783680.

[4] ONG, S. E., LEE S. C.: SEOS: Hardware Implementation of Real-Time Operating System for Adaptability, 2013, ISBN 9781479927951.

[5] FERREIRA, C., OLIVEIRA, A. S. R.: Hardware Co-Processor for the OReK Real-Time Executive, 2010.

[6] KOHOUT, P.: Hardware Support for Real-Time Operating Systems, 2002.

[7] BUTTAZZO, G. C.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 2011, ISBN 9781461406754, 9781461406761.