An FPGA Based Scheduling Coprocessor for Dynamic Priority Scheduling in Hard Real-Time Systems

Jens Hildebrandt, Dirk Timmermann

University of Rostock, Institute of Applied Microelectronics and Computer Science, Richard-Wagner-Str.31, D-18119 Rostock, Germany
{hil, dtim}@e-technik.uni-rostock.de

Abstract. In this paper we present a scheduling coprocessor device for uniprocessor computer systems running a real-time operating system (RTOS). The coprocessor shortens the scheduling time of the operating system by performing dynamic priority computation for all tasks in parallel and making a task selection according to these priorities at a higher speed than a software solution would do. This paper starts with a survey of related work and gives a motivation for the development of the proposed coprocessor architecture. We describe the architecture of our deterministic scheduling coprocessor and an efficient FPGA implementation and give a performance evaluation.

1 Introduction

In multitasking real-time operating systems (RTOS) the scheduler has a twofold impact on system performance. By determining the order of task execution according to a specific scheduling algorithm the scheduler of an RTOS bears a huge part of responsibility for the timeliness of the whole real-time system. Furthermore, the scheduler belongs to the most often called operating system functions. Thus, scheduler function execution time makes up a considerable share of overall operating system overhead. One way to decrease system overhead while using powerful scheduling methods is to support scheduling by dedicated hardware benefiting from higher execution speed and excessive parallelism.

In recent years, some work has been done in the area of hardware based scheduling or, more generally, hardware implemented operating system functions. In [1] and [3] an entire real-time kernel implemented in hardware is described for use in single- and multiprocessor systems. The scheduler incorporated in that device uses static priority preemptive scheduling. A scheduling coprocessor for multiprocessor real-time systems is presented in [2]. This device supports static as well as online scheduling and is designed to accelerate scheduling in the Spring operating system kernel [6]. The architecture proposed in [5] is aimed at a different field of application.. This work describes a scalable Earliest-Deadline-First (EDF) scheduler for ATM networks. Here, the scheduler inserts incoming data packets into an output queue according to deadlines derived form the guaranteed bandwidth of the input channel.

2 The Scheduling Coprocessor Architecture

The scheduling coprocessor proposed in this paper is intended for use in single-processor real-time systems running a real-time operating system (RTOS). The coprocessor supports dynamic priority scheduling and uses the same basic architecture for a whole class of scheduling algorithms by changing only minor functional blocks specific to the desired scheduling method. The coprocessor can easily be adapted to actual requirements (number of tasks, parameter resolution) via design parameters during synthesis. The device periodically computes task priorities and performs a selection by these values of the next task to run. Task switching itself has to be done by software as this operation is very processor specific and depends on the operating system as well. This split of the scheduler ensures a high degree of independence from the hardware and software environment the coprocessor works in.

The basic architecture of the scheduling coprocessor is given in [Fig. 1.]. Task parameters and task state information are loaded into internal registers via a simple interface with data and address lines, chip select, and read/write control lines. A scheduling operation is started by a trigger pulse on the *tck* input. End of operation and hence the availability of a valid scheduling result is signaled by an active interrupt output *sched_int* . A second interrupt output, *err_int*, gets active whenever a deadline violation by one ore more tasks is detected. Following an interrupt a read access to the coprocessor delivers the number of either the next task to execute or, in case of an error interrupt, of a task that will miss or already has missed its deadline.

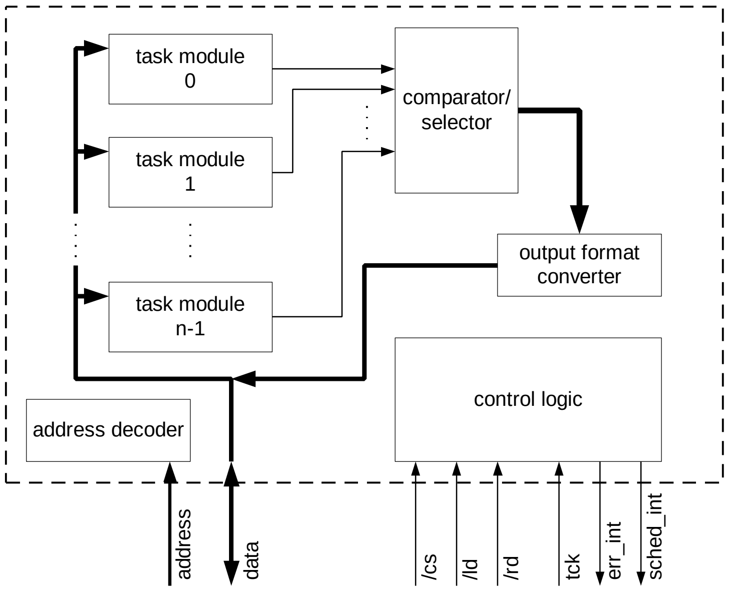

Fig. 1. Basic architecture of the scheduling coprocessor

The main components of the coprocessor are the *task modules* and the *comparator/selector* unit.

Each *task module* represents one task and contains registers for task parameters and task state. Based upon this information the dynamic priority of the task is computed inside the *task module* according to the implemented scheduling algorithm. The computation is started by the external *tck* signal. The result is shifted out of the *task module* serially under control of the *comparator/selector* module.

The maximum number of tasks that can be scheduled is determined by the number of *task modules* contained inside the coprocessor device. Task parameters can be changed during run time. Thus, a time multiplexed usage of one *task module* for two or more tasks which are not active at the same time is possible although this requires a careful system analysis and some additional operating system overhead at run time.

The *comparator/selector* module compares the priority values of tasks computed inside the *task modules* and selects, according to the particular scheduling algorithm, either the smallest or the biggest value. The information which *task module* outputs that value is coded in the output vector of the *comparator/selector* .

The *comparator/selector* works bit-serially, i.e. it compares the priority values of all tasks in parallel by reading them bit by bit. By this, execution time of a compare-and-select operation is independent of the number of tasks actually taking part in the scheduling process. Instead, execution time is determined by the bit-width of priority values which is constant for a given implementation of the coprocessor. This deterministic behavior is important for real-time systems as it eases exact timing analysis of a system.

3 Implementation

The coprocessor architecture described in the previous section was realized in several versions using different scheduling algorithms in a XILINX Virtex FPGA. The most advanced one implements the Enhanced Least-Laxity-First (ELLF) [4] scheduling algorithm. Therefore, *task modules* comprise two preloadable downward counters, one of them for the remaining run time of the task and the other one for the remaining time until deadline. The difference of these values is loaded into a shift register and shifted out to the *comparator/selector* bit by bit. Alternatively, the remaining time until deadline can be loaded into the shift register which is required in a second phase of the compare/select process. Thus, as these two parameters have to be shifted out consecutively, a complete scheduling operation for this implementation of ELLF takes

$$N = 2w+2 \tag{1}$$

clock periods, where w is the bit-width of the laxity and time-until-deadline values. The two extra clock cycles are needed for decreasing the counters and loading the new laxity value into the shift register. In Table 1. execution times and resource requirements for different combinations of maximum number of tasks and parameter resolution of this ELLF implementation are given.

Table 1. Resource requirement and execution time for ELLF implementations with different maximum task numbers and parameter resolutions (using Xilinx Virtex 1000)

Max. tasks	Parameter resolution, bit	Equiv. gate count	Execution time at max. clock rate, µs	Max. clock rate, MHz
16	12	70199	0.750	34.650
16	16	76849	1.005	33.838
20	12	78234	0.774	33.575
20	16	86540	1.019	33.340
32	16	116852	1.079	31.488

4 Conclusions

The scheduling coprocessor architecture described in the previous sections is suitable for implementations of different dynamic priority scheduling algorithms. It provides a fast and deterministic means for dynamic priority computation and hence makes this algorithms effectively usable for real-time operating systems. Usage of FPGA devices makes it even possible to configure the scheduling coprocessor at run time for different algorithms according to the actual requirements of the operating system. Due to the universal architecture, this can be done without changing the software routines that access the coprocessor.

References

1 Adomat, J. et al.: Real-Time Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems. Proceedings of the Euromicro RTS'96 Workshop. IEEE Computer Society, Los Alamitos (1996) 164-168

2 Burleson, W. et al.: The Spring Scheduling Coprocessor: A Scheduling Accelerator. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 7, No. 1, (March 1999) 38-47

3 Furunäs, J. et al.: RTU94 – Real Time Unit 1994 – Reference Manual. Technical Report, Mälardalens Real-Time Research Centre (1998)

4 Hildebrandt, J., Golatowski, F., Timmermann, D.: Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems. Proceedings of the 11th Euromicro Conference on Real Time Systems. IEEE Computer Society, Los Alamitos (1999) 208-215

5 Kim, B.K., Shin, K.G.: Scalable Hardware EDF Scheduler for ATM Networks. Proceedings of the 18th IEEE Real-Time Systems Symposium. IEEE Computer Society, Los Alamitos (1997) 210-218

6 Stankovic, J., Ramamritham, K.: The spring kernel: A new paradigm for real-time systems. IEEE Transactions on Software Engineering. IEEE Computer Society, Los Alamitos (May 1992) 54-71