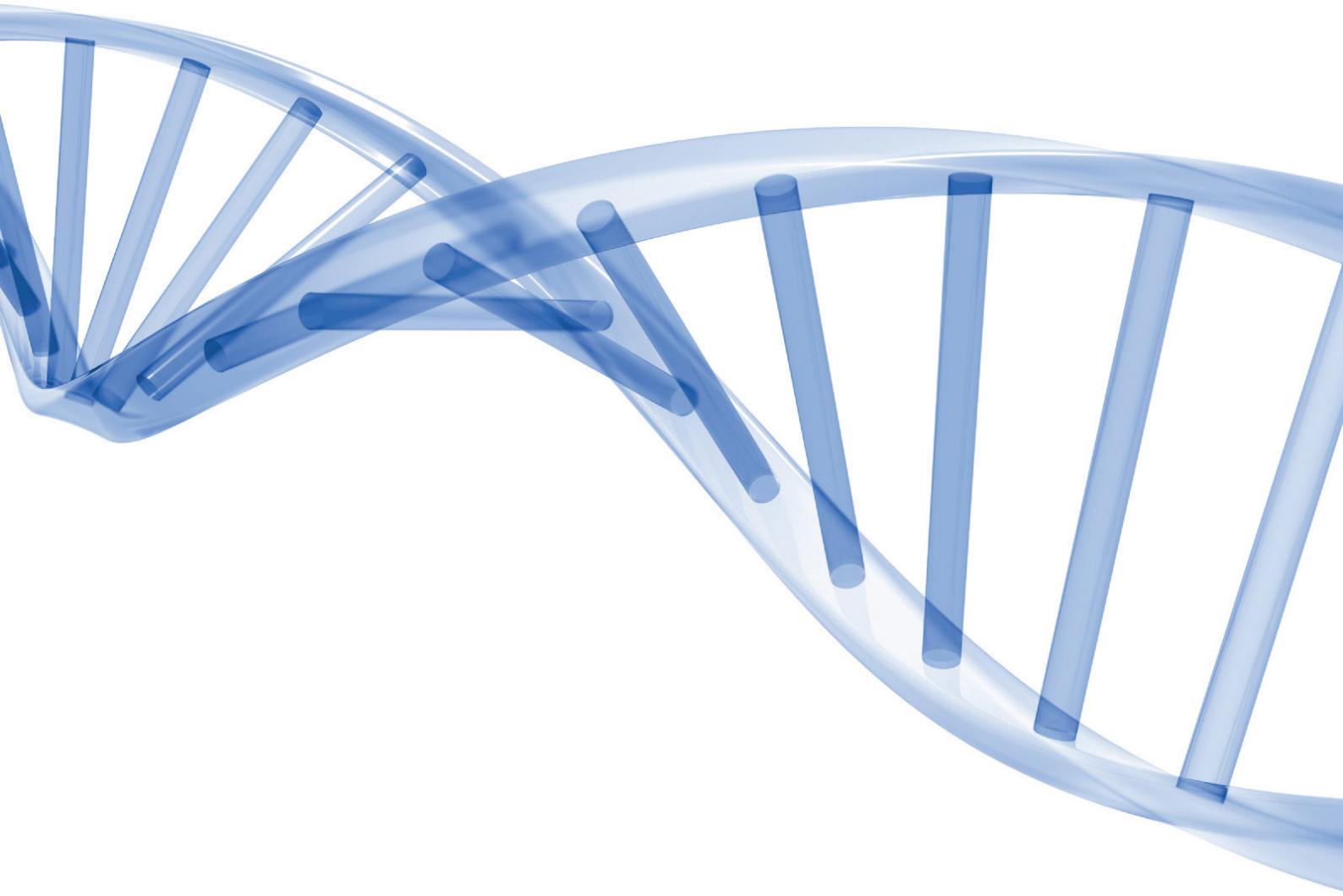
Timing Architects



Timing-Architects Embedded Systems GmbH

Effects of Task Priority Assignment in Embedded Multi-Core Real-Time Systems

Authors: Erjola Lalo, Dr. Michael Deubzer

www.timing-architects.com

Introduction

Task priority optimization is an old topic in embedded real-time systems theory. It started with single core processors where several heuristics and solutions are devised by different authors and problem was adequate solved. Nowadays, with the use of multi-core processor technology it became again an even more challenging topic. Although, multi-core processors lead to enhanced performance, reduced power consumption and efficient parallel processing of multiple tasks, it leads to software architecture design challenges and multi-core effects. In embedded real-time systems the shift from single-core to multi-core processors leads especially to communication based effects on timing such as inter-core communication delays and blocking times. In the following we describe what drives the motivation for optimization of task priorities and why an old topic becomes up-to date again.

What is a real-time System?

Real-time systems are composed by a number of tasks, managed by an operating system. The amount of time that takes to accept and complete an application's task is important. A real-time embedded system functions correctly only when computational results are correct and produced on time. Therefore, timing requirements such as deadlines are defined for the system's tasks to quantify what extent computational results are produced on time. A deadline represents the upper limit for the response time of a task. The response time is calculated as the difference between starting time and finishing time of the task. If a task produces the results after the deadline in a hard real-time system it can cause catastrophic consequences. These systems are generally found in safe-critical systems, where functional safety of the system is mandatory. In an embedded real-time system, tasks may be activated finite times. In every task's activation, an instance of it is released and the task is "ready". Hence, each task consists of a finite sequence of identical activities or jobs called instances. The activation time of the first instance is called offset. Each instance or job of a task is characterized by a period which is the distance between activation of one instance of the task to the activation of the next instance. Tasks with a regular activation are called periodic tasks, and tasks with irregular activations are called sporadic tasks. A system can have different types of tasks. Based on the period and offset, the system is categorized as synchronous (first instances of every task have the same offset) and asynchronous (first instances are activated on different times).

What is the Priority of a Task?

Priority is a scalar value mapped to every task and defines the order in which tasks are executed in a core by the scheduler of the operating system. The order of tasks and the allocation of tasks to cores are referred as scheduling. In priority based scheduling the order is defined by the priority. The highest priority ready task executes first on the core, the next higher priority executes after the highest priority finished, and so on. Static scheduling, meaning all instances of a task have the same priority and execute always on the same core, is introduced to increase the predictability of consequences coming from scheduling decisions. Hence, in static scheduling the scheduling decisions and the mapping to which core tasks have to execute are taken before the system begins to operate and it is required a prior knowledge about the system. The schedule of tasks running in a multi-core environment defines, although indirectly, the time in which computational results of each task are delivered. Hence, the right prioritization of tasks ensures an ordering of tasks where each task meets their deadlines. In real-time systems, preemption allows an urgent ready task to immediately execute once it is activated by preempting the execution of a running task. The operating system saves the context stack of the running task which is resumed after all higher priority ready tasks have finished. In preemptive scheduling, a ready task with higher priority can preempt a running task with lower priority at any time during execution. Although preemptive scheduling is used to increase in general the schedulability of the system, it might cause a high preemption overhead. This overhead consists of increase of memory size for saving context stack of all preempted tasks, increasing cache related delays and so on. Limited preemption was introduced for limiting preemption overhead where several authors proposed different approaches in implementation of it.

> „The right prioritization of tasks ensures an ordering of tasks where each task meets their deadlines"

In practical approaches, such as in OSEK real-time operating system, the limited preemption scheduling is implemented as cooperative scheduling. In cooperative scheduling, tasks do not preempt each other at any time during execution. They instead preempt each other at re-scheduling points where re-scheduling decisions are taken by scheduler. Therefore, tasks are grouped in cooperative groups based on an additional scalar value called priority threshold or priority grouping. Figure 1 shows limited preemption in a system with

two cooperative tasks running on the same core. The green color section shows the execution of the task. Each part in a task indicates the presence of a re-scheduling point in the end of the part. In this article we address task priority optimization in static scheduling in a multi-core processor for improving timing behavior of tasks of an embedded real-time system. We hereby consider both, preemptive and cooperative scheduling in multi-core processors. In addition, we list multi-core effects in timing regarding priority optimization and we propose an approach for assigning priorities for minimizing these effects. Priority optimization is part of Timing Architects Tool Suite.

Multi-Core Effects on Timing

In fixed priority scheduling in embedded multi-core real-time systems, a proper task priority assignment has to be done in a way that the system has minimal effects on timing. In multi-core systems, possible effects on timing occur because of task interferences and additional communication delays such as inter-core communications and buffering times. In the following paragraph, we show three practical scenarios when a near-optimal prioritization of tasks is sufficient and necessary for improving timing results:

1. Minimization of Task Response Times
2. Minimization of Task Start Delay
3. Minimization of Task Switching Overhead

Minimization of Task Response Time

Task response times on multi-core systems can be affected as a result of task interference due to access to common resources e.g. semaphores. The task requesting the semaphore first holds the semaphore. For example, two tasks A and B, which are running on different cores, can delay one or another because of exclusive hold of a semaphore. Thus, task A is delayed or blocked for execution because of waiting for the release of the semaphore hold by task B. In such circumstance the response time of task A increases due to the waiting time. In the worst case, the increased response time could lead to a deadline violation. Figure 1 shows the Gantt chart simulation result of dual core system with two tasks requesting one resource. The system is simulated and visualized using Timing Architects Tool Suite. In the first part is shown the simulation state of the semaphore SEM_1 and in the second part framed in blue is shown the simulation result of tasks in the system. The green color section shows the execution of the task. The red section shows the semaphore has reached the max value and the brown section shows the task is executing but with a deadline violation. Task priorities influence this effect by

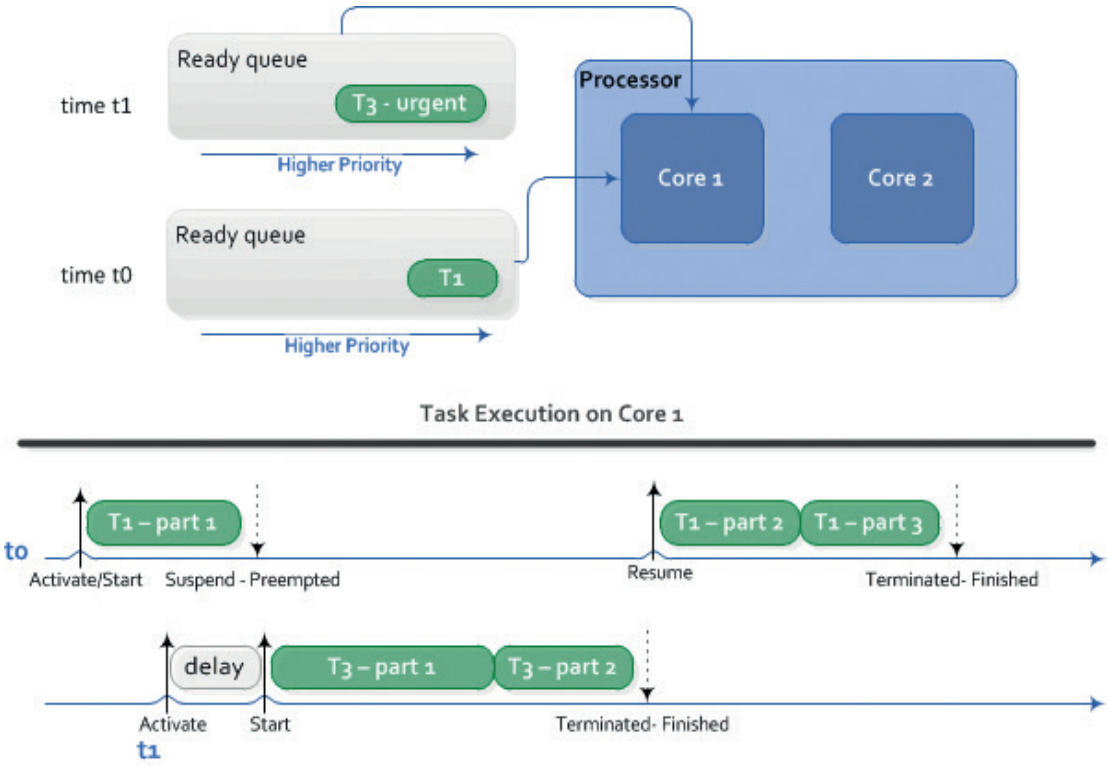


Figure 1 Limited Preemption.
Task T1 and Task2-urgent are cooperative and run on the same core. T1 is activated at time t0. No other task is ready at time t0, therefore T1 starts execution. At time t1, task T2-urgent is activated with a higher priority than of task T1 but with the same priority grouping. Hence, task T1 is preempted only at schedule point. After task T2-urgent finishes, task T1 resumes and starts execution.

> „Adjusted task priorities over multiple cores allow to reduce blocking times for shared resources."

changing which task executes first in the case multiple tasks are ready. Synchronized task priorities over multiple cores allow to synchronize access and therefore to block times for shared exclusive resources, too.

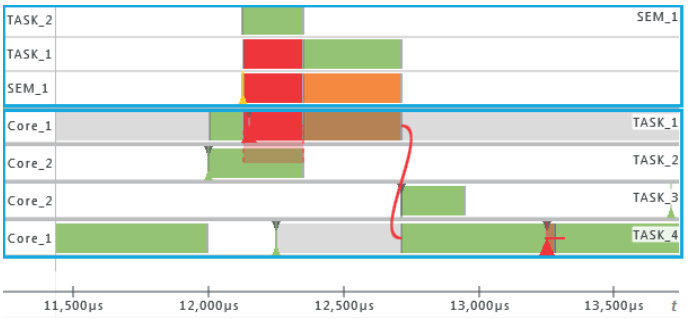


Figure 2 Task Interference – Timing Constraint Violation. TASK_1 and TASK_2 are running on different cores and both request semaphore SEM_1. TASK_2 gets first the lock on the semaphore and TASK_1 falls in a waiting state at t = 12,125 µs and cannot continue executing until the semaphore is released at t = 12,375 µs. As a result a deadline violation of TASK_1 occurs.

Minimization of Start Delay

Delays can occur because of preemptive and cooperative task suspensions. Hence, a task A with a lower priority is delayed as a result of activity of another task B with a higher priority. Task A delays while B is running on the core. Therefore, RT_( ) (Response Time) of task B increases. A practical scenario is shown in Figure 2. The red lines connecting instances of difference show the beginning of the interference from a higher priority to a lower priority task.

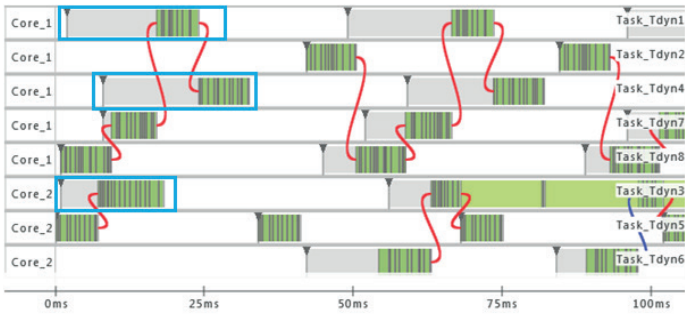


Figure 3 Task Interference - Start Delay. Task_Tdyn5 runs on the same core as Task_Tdyn3. It has a higher priority and it is the first which starts execution. Therefore, Task_Tdyn3 is delayed. The same occurs to other tasks framed in blue.

Minimization of Switching Overhead

A task can be preempted often during execution time because of higher priority tasks. More precisely, a chain of preemptions can occur. Task A is preempted by B;B is later preempted by C which is then preempted by D and so on. In all preemptions, the operating system has to save the context stack of each task. Task switching overhead consists in a time overhead and in an increase of the memory size. Figure 3 shows simulation results of a system that has high preemption overhead as a result of a bad priority assignment.

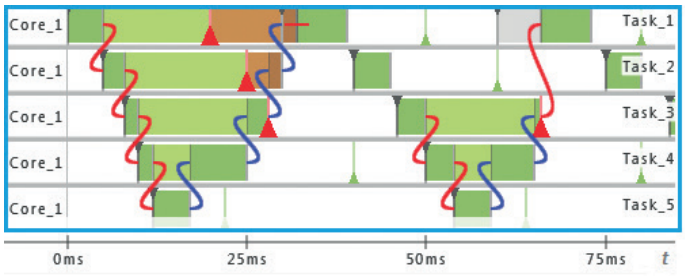


Figure 4 Hierarchical preemption and task switching overhead.

Relevance of Task Priority Optimization

Task priority optimization has concerned many authors in the research area, and several solutions were proposed and their optimality was proved. Deadline monotonic and rate monotonic are the most known heuristics for priority assignment for single core processor systems. Although, these heuristics are very powerful it is proven that they are not optimal for certain conditions in the system e.g. dependent task sets in multi-core systems. In multi-core task dependencies are evident. Precisely, there are two types of dependencies: Resource Dependency and Data Dependency. At a Resource Dependency tasks are running on separate cores and may have simultaneous exclusive request to the same resource. The task which requested the resource first holds the resource, as described above. Also when there is a higher priority task which requested the resource this task has to wait for the release of the lower priority task. On the other hand, Data Dependency is present when a task waits for the result of another task in order to execute or continue execution. The effect of priority in a system having these dependencies is obvious. Hence, the right prioritization can improve timing results of the overall system.

In order to provide a solution for this problem we proposed an optimization approach where we use genetic algorithms in combination with model-based timing simulation for assigning task priorities of a system. A timing model is a model based abstraction approach of a real-time embedded system where execution times of tasks as well as the activation behavior are described by probabilistic functions. We use an event-based simulator for evaluating the timing model and storing task state transitions events such as activate, start and terminate in a trace file with respective timestamps, also called trace. By applying metrics to the transition events of the trace, e.g. response time which represents the time between activation and termination of a task, and calculating statistical estimator to the metric values of

all different instances of a task, e.g. maximum, we have an indicator for the quality of a priority assignment. The power of genetic algorithm stands in the ability to search for priorities that strait forward algorithms do not find. Hence, it provides different solutions of priority assignment which are evaluated for defining their quality. We have seen that for complex systems with different types of dependencies our algorithm provides much better results compared to existing heuristics. In our approach, the optimization algorithm takes a given system abstracted in a timing model as an input and produces several solutions of the system with assigned priorities. The system is characterized by a set of properties and attributes. Properties are not changed during optimization; they serve as descriptive information of the system e.g. the task set with activation patterns and execution times, scheduling configurations, hardware configurations e.g. quartz of cores, etc. Attributes are systems characteristics that are changed by optimization algorithm. In attributes are included priorities of tasks and preempt ability (defines priority grouping of tasks; it groups tasks in cooperative and fully preemptive groups). The purpose of the algorithm is finding a near-optimal value of attributes in a way that timing results of the system are improved. Figure 5 describes the high-level overview of priority optimization algorithm.

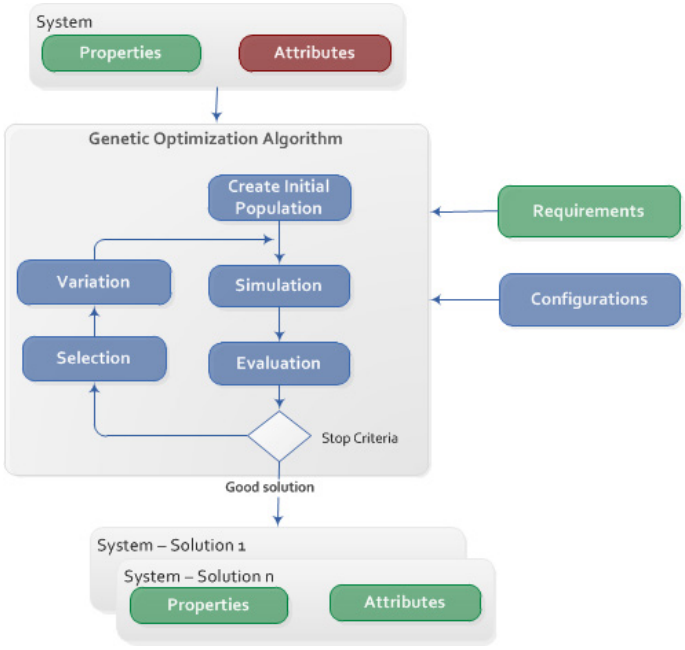


Figure 5 High-level overview of priority optimization algorithm

In genetic algorithms a set of solutions called the population is used to sample the exploration space. In our approach a solutions is a set of tasks with their attributes: priorities, preemptability, and metrics for timing evaluation. Genetic algorithm involves several steps called generations. In each generation, a number of solutions are produced. The first set of created solutions is called initial population. New solutions in next generations are created as a combination or modification of solutions of previous generations, referred as parents. This step of the algorithm is called variation. Priority optimization algorithm produces several solutions of priority assignment for the system taken as input. The best solutions are those which fulfill the configurations and requirements defined in the algorithm. Requirements have to be fulfilled by the algorithm for providing an acceptable prioritization. Such requirements for example are: minimization of response time for tasks of system S, the minimization of multi-core effects, i.e. memory requirement or communication delays, minimization of preemption overhead, etc. The algorithm can be configured using the following parameters: the stopping condition of the algorithm (number of generations, number of solutions produced), number of solutions generated in 1st generation (initial population), number of solutions generated in next generations, number of cooperative/ preemptive task groups, etc. In specific, the priority genetic algorithm runs through the following steps:

> "Reasonable specified requirements allow the algorithm to provide best task prioritization."

Create Initial Population

An initial population is created through uniform random generation of priorities for each task. In addition, new preemptability is assigned to each task by creating random generated cooperative clusters of tasks. A good initial population is a population with a variety of characteristics and differences between solutions.

Evaluation

Each solution produced by priority optimization algorithm is evaluated in evaluation step. Evaluation is the process of evaluating the quality of the produced solutions. It is quantified to what extent the requirements are fulfilled. When the expected requirements are fulfilled the solution is considered to be a valid candidate. Best solutions searched in each generation are ensured through a process called "fitness survival" of individuals. Fitness is a scalar value that measures the quality of solutions. During the evaluation a fitness value is assigned to each solution. In the fitness calculation is considered normed response time of each task. When response time of a task is higher than the deadline, a deadline violation is present. Response times of each task are adjusted through normalization in a common scale. Normalization of response times consists in an easy comparison of values.

In our approach, we use TA Tool Suite for the evaluation of each of the solutions generated by the genetic algorithm. Each solution is thereby evaluated regarding real-time properties, memory consumption and communication overhead.

Selection

The selection is a process that selects the solutions that will be part of the subsequent population.

Variation

It is used in order to create new solutions in each generation. Variation is performed by crossover and mutation, where each plays a different role in the genetic algorithm. Crossover combines attributes of two or more individuals for creating new solutions. In priority genetic algorithm we use single-cut point crossover operator. The operator selects a random cutting point in two parents; it combines information of two parents by producing two children with inherited genes from both parents. Mutation modifies the attributes of existing solutions to create new solutions. Mutation is needed to explore new states through avoidance of local minima. Crossover increases the average quality of the population.

The optimization algorithm stops when the stopping criterion is reached. In the end of the algorithm, several solutions of priority assignment of the system are delivered. For more technical details, we provide a comprehensive description in: Lalo Erjola et al.: Task Priority Optimization in Embedded Multicore Real-Time Systems; 2014; http://goo.gl/fAyyhl

Summary

Multi-core processors introduce new delay effects on calculation results of tasks in a real-time embedded system. In priority based scheduling these effects are minimized by assigning proper priorities to tasks. Priority in priority scheduling algorithms defines the schedule in which tasks are executed. The optimality of task priority assignment approaches in an embedded multi-core real-time system is strongly dependent on the characteristics of the system. Therefore, we developed an optimization algorithm which allows an automatic improvement of task and interrupt priorities in a very generic way. The product is used in several automotive tier one mass production systems, also in cooperation with the OEM.

About Timing Architects

Timing-Architects (TA) is an international operating high-tech company for development tools and methods, specialized in the optimization of software for embedded multi- and many-core real-time systems. For their innovative tool solution “TA Tool Suite” the company received multiple awards and is also recognized by experts as a leading player in research and development of multi-core embedded systems.

About the authors

Erjola Lalo is working as research assistant at OTH Regensburg, Germany. Her research project is priority, preemptability and schedule point placement optimization of multi-core real-time systems as participant in the Timing-Architects’ Trainee Program.

Dr. Michael Deubzer is CEO and co-founder of the Timing-Architects Embedded Systems GmbH. He received his doctor’s degree from the Technical University of Munich, writing his thesis on advanced scheduling algorithms for efficient and robust dynamic scheduling of next generation embedded systems, using multi-core processors.

Your contact at Timing-Architetcs:

Timing-Architects Embedded Systems GmbH
Bruderwöhrdstr. 15b
93055 Regensburg

Phone: +49 (0) 941 604 889 250
FAX: +49 (0) 941 604 889 259
Email: info@timing-architects.com

Timing Architects