Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 238 (2010) 81–95
www.elsevier.com/locate/entcs

Cost Analysis for Embedded Systems: Experiments with Priced Timed Automata*

Tolga Ovatman[1]

Department of Computer Engineering
Istanbul Technical University
34469 Maslak, Istanbul, Turkey

Aske W. Brekling[2] Michael R. Hansen[3]

Department of Informatics and Mathematical Modelling
Technical University of Denmark
2800 Lyngby, Denmark

Abstract

Analysis of resource consumption of embedded systems is a major challenge in the industry since the number of components that can be included in a single chip keeps getting bigger. In this paper, we consider simple models of embedded systems and the automated analysis about timing and memory access costs of those models. In order to achieve this, a basic model is built using priced timed automata and some resource consumption scenarios are verified. Even though the experiments are performed on small and basic models, we believe we have taken a basis step in showing that it is promising to use priced timed automata and Uppaal Cora as a model checking tool in reasoning about resource consumption of embedded systems.

Keywords: Embedded systems, priced time automata, cost analysis.

1 Introduction

An embedded system consists of a combination of hardware and software components, and perhaps additional mechanical or other parts, designed to perform a dedicated process. Since the system is dedicated to a specific task, it can be optimized during the design phase aiming at reducing the size and cost or increasing the reliability and performance.

* This work is partially funded by the Erasmus Student Exchange programme, ARTIST2 (IST- 004527), MoDES (Danish Research Council 2106-05-0022) and the Danish National Advanced Technology Foundation under project DaNES.

[1] Email: ovatman@itu.edu.tr
[2] Email: awb@imm.dtu.dk
[3] Email: mrh@imm.dtu.dk

doi:10.1016/j.entcs.2010.06.006

The complexity of the hardware-software systems increases as technology gets more advanced. The more complex systems become the harder it gets to provide guarantees for systems' properties. For embedded systems, it is a particular challenge to provide guarantees about resource consumption, such as bounds on the use of processing time and memory.

A traditional embedded system platform has many different parameters and the *design space* is the complete collection of all possible configurations of these parameters. *Design space exploration* is the phenomenon of searching for desired solutions among a huge variety of possible designs. An important aspect is to understand and reason about the trade-off among different design parameters. It is clear that even for simple cases such reasoning becomes a complicated matter.

There exist different frameworks for modelling and analysis of hardware-software systems. ARTS [8], for example, is a simulation framework that supports designers in analyzing designs before the system is implemented. However, ARTS cannot be used to guarantee properties as it does not investigate the full state space. Other approaches exists for analysis of designs for system on chips (SoC), e.g. [5,2], but ARTS is the main inspiration for our work, where we experiment with *Priced Timed Automata* [10] for modelling simple components having costs and we use Uppaal Cora [6] for automated reasoning about costs. Uppaal Cora is a member of the Uppaal[3] family that supports simulation as well as verification in terms of model checking. There exist other work on modelling Multiprocessor System on Chips (MPSoC) using Uppaal [4] and optimization work on task graph scheduling using Uppaal Cora [10].

In our experiments we restrict ourselves by considering a simple system model involving a processing element with a very basic local memory, a real time operating system, an external memory and an application consisting of a number of tasks as illustrated in Figure 1. An intelligent home device like an intelligent vacuum cleaner may fit into this structure. Even though the system is simple, it is a complicated matter to analyze the trade-off between aspects of timing and memory access. Using Uppaal Cora for solving this problem, we shall device cost-optimal solutions to a collection of scenarios. These optimal schedules cannot be extracted using "standard" scheduling theory. These analyses can be seen as the introductory steps towards using formal analysis for cost analysis concerning resource consumption of hardware/software systems.

The rest of the paper is organized as follows: The next section introduces a simple embedded system with some scenarios we would like to analyze. Section 3 gives a brief summary of priced timed automata. Section 4 introduces our formal model for systems. In Section 5 the experiments and formal analysis of the scenarios are presented, and the last section contains a conclusion of the study.

2 A Simple Embedded System

In Figure 1, we show the major components of the embedded systems we shall model and analyze in the following sections.

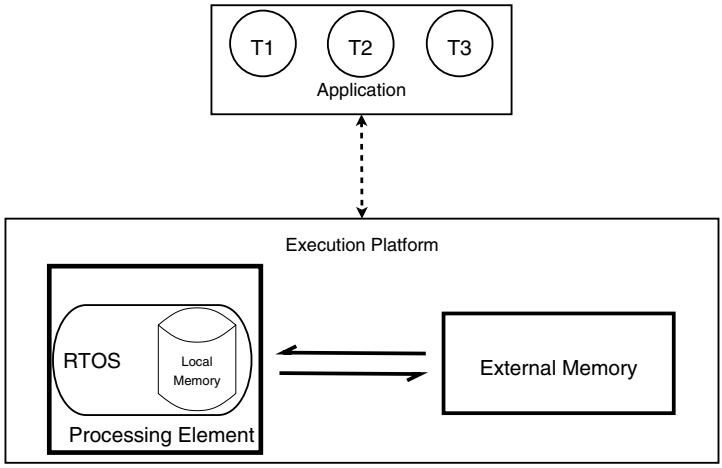

Fig. 1. An informal model of an embedded system

The system consists of a *processing element* with a *local memory*, an *external memory*, an *operating system* and an *application* consisting of a number of *task*, where just three tasks are shown in the figure. We shall consider scheduling of tasks taking cost of using the processing element and the local and external memories into account. In particular, we shall consider the following aspects:

Memory usage: The local memory is cheaper to use than the external memory, i.e. if the data for a task already exists in the local memory then the cost of executing the task is cheaper than if it first has to access the external memory. We want to schedule tasks so that they meet their deadlines as cheaply as possible, and therefore, the best scheduling strategy will not necessarily be earliest deadline first [7], but rather it would depend on the data dependencies among tasks.

Waiting time of tasks: First of all every task should meet each of its deadlines. In addition to that we shall impose another cost parameter concerning its waiting for processing time, as some tasks may be more important than others and the cost of making important tasks wait should be high.

Combined costs: In a realistic analysis, it is important to consider more than one of the properties of the system and analyze the total utilization of the systems resources using different configurations. In our examples, there will be trade-offs between optimizing the scheduling wrt. the cost of the different memory accesses and optimization wrt. the cost concerning the waiting times for tasks.

Optimization of costs: The goal is to find cost-optimal use of resources. The cost criteria may involve waiting time of tasks, running time and memory usage. We use Uppaal Cora to find the optimal schedule with respect to given cost criteria. Our general aim is to show that priced timed automata can be used for modelling embedded systems and Uppaal Cora can be used for design space exploration.

3 Priced Timed Automata

We will now give a brief overview of priced timed automata [9]. Priced timed automata are extensions to timed automata [1] addressing the calculation of prices for different runs of a timed automata. In order to do this, *cost rates* $\dot{c}$ can be added to locations and *costs* c to transitions. A cost rate $\dot{c} == x$ denotes that c grows constantly with rate x in a location. The cost $c = x$ denotes that x is added to the total cost of c when the transition is taken.

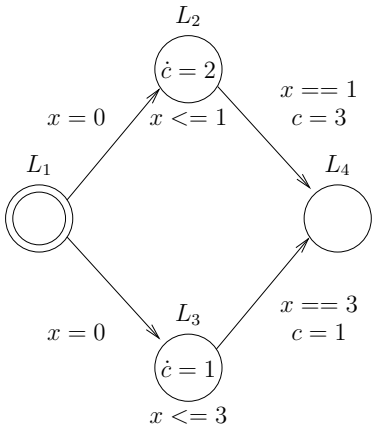

Fig. 2. Example of a priced timed automaton

Looking at Figure 2, we can see a simple priced timed automaton with four locations and four transitions which can lead to different runs of the automaton from the initial location L_1 to location L_4. In one run it will spend 3 time units in location L_3 due to the *invariant* $x \leq 3$ in the location and the *guard* $x == 3$ on the transition leaving L_3, where x is a (real-valued) clock which is set to 0 in both transitions leaving L_1. The other run over L_2 has a similar explanation.

Examining the figure, we can see that locations L_2 and L_3 have cost rates denoted with $\dot{c}$ that represents the increase of the total cost per time unit. In our case, the total cost will be incremented by 2 for each time unit that has been spent in location L_2 and 1 for each time unit that has been spent in L_3. We can also see that taking the transition $L_2 \rightarrow L_4$ has cost 3 and taking the transition $L_3 \rightarrow L_4$ has cost 1.

There are two principally different runs of the priced timed automaton in Figure 2:

$$\alpha = L_1 \rightarrow L_2 \rightarrow L_4$$
$$\beta = L_1 \rightarrow L_3 \rightarrow L_4$$

and the total cost for these runs are:

$$cost(\alpha) = 5$$
$$cost(\beta) = 4$$

In this example the infimum cost for all runs leading to location L_4 is $cost_{inf} = cost(\beta) = 4$, and UPPAAL CORA is able to compute such infimum costs.

3.1 *Using* UPPAAL CORA *on Cost Models for Embedded Systems*

We conduct cost analysis for embedded systems using UPPAAL CORA. Firstly, we provide a priced timed automata model - implemented in UPPAAL CORA - for embedded systems where different costs (e.g. tasks' waiting time) are incorporated. UPPAAL CORA can then make verification based on a query given in the UPPAAL Requirement Specification Language. The basic property we want to verify in our experiments is that all tasks of the system are able to run a specified number of times while meeting all their deadlines. When this query returns *Property is Satisfied*, there can be any number of traces in the computation tree which leads to the specified situation. An option that provides the user with the *best* trace can be enabled in UPPAAL CORA. The *best* trace is the one with infimum cost. When specifying systems, the main idea is to "under-specify" your models leaving degrees of freedom, e.g. in terms of non-determinism in scheduling decisions, so that UPPAAL CORA can determine which decisions will leads to the *best* trace.

4 Model of the Simple Embedded System

In this section we present a model of an embedded system using priced timed automata. Considering Figure 1, a system can be thought of as a parallel composition of an application and an execution platform:

$$System \mathrel{\widehat{=}} Application \parallel ExecutionPlatform$$

where an application is a parallel composition of a number of tasks (τ_i):

$$Application \mathrel{\widehat{=}} \|_{i=1}^{n} \tau_i$$

An execution platform is a parallel composition of a number of hardware components (COM) and a real-time operating system ($RTOS$) that coordinates the tasks while considering the the situation of the hardware components[4]. This can be expressed as follows:

$$ExecutionPlatform \mathrel{\widehat{=}} \|_{j=1}^{m} COM_j \parallel RTOS$$

In the following we will give the main characteristics only of tasks, hardware components and the real-time operating system.

Our model will handle non-preemptive periodic tasks with offset times and specified run times. Each task may perform a number of memory accesses in each period

[4] A system could be modelled using a real-time operating system for each processing element, however, in this work we only analyze systems with a single processing element

and it will not perform a memory access if the desired variable resides in local memory. Some properties of a task is shown in Figure 3 with their symbols and variables in the UPPAAL CORA model.

	period	runtime	offset	number of times to be run
Symbol	π	ρ	o	θ
Variable	*period*	*run_time*	*offset*	*nr_runs*

Fig. 3. Attributes of tasks

The real-time operating system will only deal with task scheduling and memory access operations, and our model will be constructed as a special component that deals with precisely these operations.

An instance of a hardware component *COM* can be used to represent any physical hardware component of the system such as a processing element or a memory. This model is responsible for receiving a triggering signal from another component indicating that an input is present, process this input, and send the result to another component. Hardware components are connected to each other in a specified sequence, describing the layout of the execution platform. The operating system initiates and terminates this sequence of signals.

4.1 A Simple Example

In this section, we consider the complete model of a system consisting of one task τ, one real-time operating system *os* and one processing element *pe* that is directly connected to an external memory *mem*. This system is described as follows:

$$\begin{aligned} System &= Application \parallel ExecutionPlatform \\ Application &= \tau \\ ExecutionPlatform &= os \parallel pe \parallel mem \end{aligned}$$

Our system model is composed of parallel running automata each of them representing a specific component of the system. Communication between these automata is handled via global variables and communication channels.

Communication between the hardware components and the operating system is illustrated in Figure 4, where `SIG` is an array of channels and the placement in this array denotes the resource for which it is destined, e.g. `SIG[PE]` is destined for the processing element.

Communication between the execution platform and the application in this example is shown on Figure 5, where `READY`, `RUN`, `FINISH` and `IO` are the channels used for this communication.

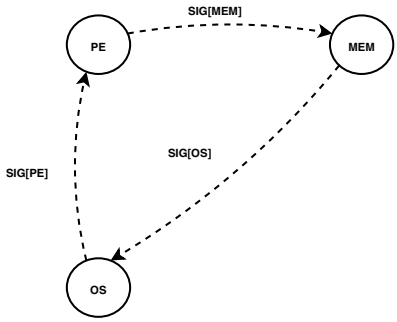

Fig. 4. Communication diagram between hardware components

Fig. 5. Communication diagram between the application and the execution platform

4.1.1 Model for a task

The priced timed automaton for τ is given in Figure 6. It resides in the `START` location before it start the cyclic behavior when the `offset` time has elapsed. When the automaton enters the `IDLE` location it waits for a new period to start before it moves to `READY`. While moving to `READY` it signals the *os* component indicating that it has become ready. In `READY` the task waits until a `RUN` signal has been issued from *os* before moving to `RUNNING`. In `RUNNING`, the task moves to `MEM` whenever it needs to issue a memory access and takes one of the transitions back to `RUNNING` depending on the presence of the data in the local memory. After the last memory access, the task moves to `IDLE`, issuing a `FINISH` signal to *os*, where it waits for its next period.

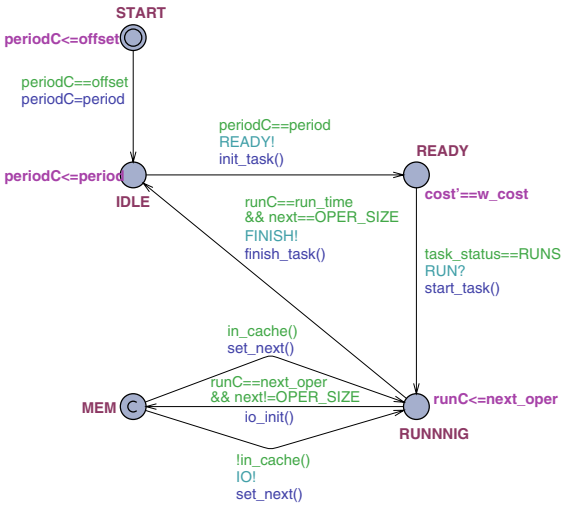

Fig. 6. Priced Timed Automata model of τ

4.1.2 Model for a real-time operating system

The automaton for *os* is given in Figure 7. The purpose of *os* is to handle the communication between the application and the execution platform. This automaton starts in the `IDLE` location and waits for tasks to issue `READY` signals. Based on those signals it makes a scheduling decision and moves to the `SCHEDULED` location, from which it issues a `RUN` signal to the task that has been scheduled and moves to the location `TASK_RUN`. It then triggers the component chain by issuing a `SIG[PE]` signal to the *pe* and moves to `WAIT_COM`, where it waits for the output signal from the component chain to arrive before moving to `COM_FIN`. Here, it can either finish execution by receiving a `FINISH` signal from the task and move to `IDLE` or it can receive an IO signal from the task on which it moves back to `TASK_RUN`. The automaton does not react to `READY` signals when making or executing a scheduling decision.

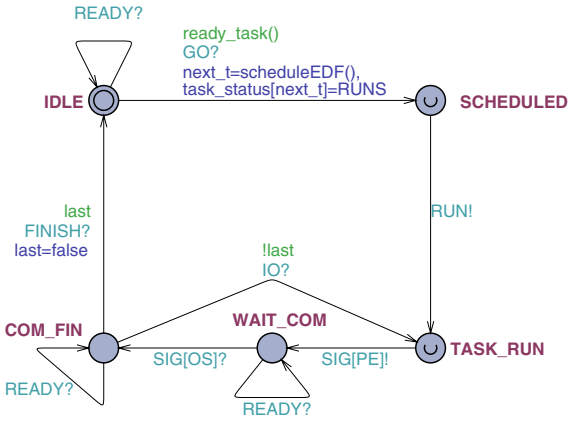

Fig. 7. Priced Timed Automata model of *os*

The assignment `next_t=scheduleEDF()` indicate on which transition the *os* would make a scheduling decision. In our experiments we want Uppaal Cora to find the optimal schedule. Therefore, the assignment `next_t=scheduleEDF()` is removed in order to allow a non-deterministic choice of any ready task as its scheduling decision. Using the *best*-trace generator the optimal schedule is extracted.

4.1.3 Model for a processing element

The automaton for *pe*, shown in Figure 8, waits for a triggering input signal `SIG[PE]` from another component (*os* in this simple example) in the `IDLE` location. The location `PROCESS` models the actual processing, and in location `SEND` it spends some time communicating the output and signal the result to *mem* using `SIG[MEM]`.

4.1.4 Model for a memory component

The automaton for *mem* in Figure 9 starts in the `IDLE` location by waiting for a triggering input signal `SIG[MEM]` from another component (*pe* in this case). In `PROCESS` it does some *processing*, which is accessing data in this case. Finally, in `SEND` it can spend some time communicating the output and signals the result to

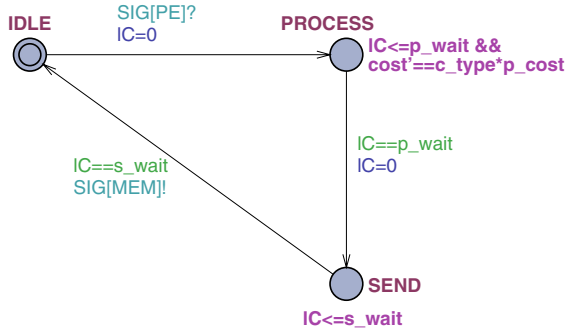

Fig. 8. Priced Timed Automata model of *pe*

the next component in the chain, which in this case is *os*, using `SIG[OS]`.

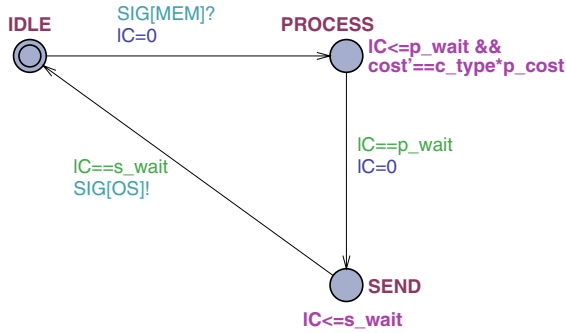

Fig. 9. Priced Timed Automata model of *mem*

4.2 The costs model

We will consider two kinds of costs:

(i) The cost of a task waiting for processing time. This is modelled using the cost rate `w_cost` in the `READY` location of the task automaton.

(ii) The cost of accessing a hardware component, e.g. a memory component. This is modelled using the cost rate `p_cost` in the `PROCESS` location of the automaton for that component.

The problem to be solved is finding a cost-minimal schedule so that every task meets every deadline.

4.3 Analysis of a known case

In order to check whether our model is meaningful, we have tested the model on an example where we know the result. We have tried to achieve the optimal schedule in the following system definition which consists of three tasks running on a processing element without any memory access. These periodic tasks will run for a specific time on the system and Uppaal Cora's optimization will provide the schedule of these tasks depending on cost assignment.

$$\begin{aligned} \mathit{System} &= \mathit{Application} \parallel \mathit{ExecutionPlatform} \\ \mathit{Application} &= \tau_1 \parallel \tau_2 \parallel \tau_3 \\ \mathit{ExecutionPlatform} &= \mathit{os} \parallel \mathit{pe} \end{aligned}$$

The application consists of three tasks whose periods (π) and running times (ρ) are in Figure 10. The optimal schedule computed by UPPAAL CORA should be an earliest-deadline first schedule.

	π	ρ
τ_1	8	1
τ_2	5	2
τ_3	10	4

Fig. 10. Properties of tasks that will be used for validation

Using the task parameters shown in Figure 10 it is easy to verify that the optimal trace provided by UPPAAL CORA:

$$\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_3\tau_2\tau_1\tau_2\tau_3\tau_2\tau_1\tau_2\tau_1\tau_3\tau_2$$

is, in fact, an earliest-deadline-first schedule.

5 Experiments

We will now experiment with the model. In one experiment we will focus on waiting time and memory accesses. Another experiment will consider optimization over the scheduling of the three tasks and produce a unique schedule that minimizes the costs. Our system definition for these experiments is:

$$\begin{aligned} \mathit{System} &= \mathit{Application} \parallel \mathit{ExecutionPlatform} \\ \mathit{Application} &= \tau_1 \parallel \tau_2 \parallel \tau_3 \\ \mathit{ExecutionPlatform} &= \mathit{os} \parallel \mathit{pe} \parallel \mathit{mem} \end{aligned}$$

The characteristics for the three tasks can be seen in Figure 11. We have already given explanations for π (period), ρ (runtime), o (offset) and θ (number of iterations). In each period, each task has two memory accesses, unless the variable is in the local memory. The values for μ_1 and μ_2 are the durations for the first and second memory access, respectively. The values for υ_1 and υ_2 are the names of the variables used in connection with the first and second memory access, respectively.

Figure 12 shows the schedules for the system using rate-monotonic and earliest-deadline-first scheduling algorithms [7].

	π	ρ	o	θ	μ_1	μ_2	υ_1	υ_2
τ_1	50	20	30	6	2	2	A	B
τ_2	60	20	10	5	1	3	B	C
τ_3	100	30	0	3	4	5	C	D

Fig. 11. Properties of tasks that will be used through verification experiments

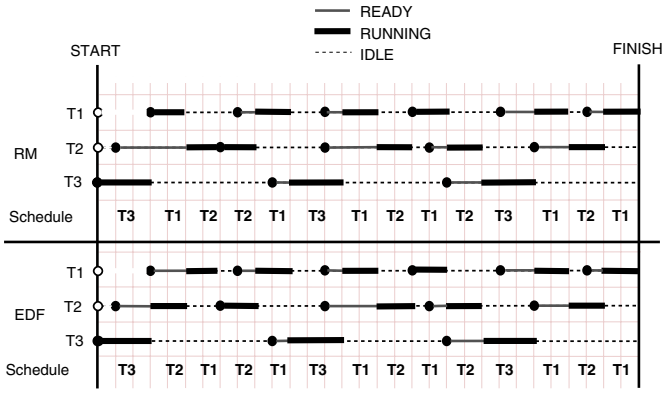

Fig. 12. Running schemes of tasks in Figure 11 when scheduled with RM and EDF

In the following experiments, system definition and task properties are as in Figure 11. We will experiment with different cost criteria concerning waiting time and memory access.

5.1 Reasoning about timing and memory access

Scenario and Purpose. In this experiment we try to see which scheduling policy (rate monotonic or earliest deadline first) will yield a better performance regarding costs for waiting time and memory access.

Modelling. The cost rate of the `READY` location, where a task is waiting, is indicated by $\dot{c}_w$ in Figure 13. The cost rate of the `PROCESS` location, indicated by $\dot{c}_m$, determines the costs while a component (in this case memory) is processing. Notice that memory access is more important than waiting times of tasks in this example.

Verification. We have used Uppaal Cora to find the total cost of the specified scenario:

$$\text{Rate-monotomic scheduling (RM)} : \sum_{i=1}^{3}(cost^w(\tau_i) + cost^m(\tau_i)) = 900$$

$$\text{Earliest deadline first scheduling (EDF)} : \sum_{i=1}^{3}(cost^w(\tau_i) + cost^m(\tau_i)) = 930$$

Analysis. Our results show that RM gives a lower total cost because:

	$\dot{c}_w$	$\dot{c}_m$
τ_1	2	10
τ_2	1	10
τ_3	1	10

Fig. 13. Waiting and memory access cost rates for the first scenario

- The $\tau_1\tau_2$ scheduling of the RM causes τ_2 to reuse the variable B from Figure 11 and hence reduce the memory access cost
- In RM τ_1 waits less and since it costs more when τ_1 waits, the waiting time cost is reduced as well.

5.2 Optimization of costs

Scenario and Purpose. We are now going to let Uppaal Cora find the schedule with infimum cost in two cases with different costs.

Modelling. In the first case, we choose to introduce a trade-off between memory access and task waiting time. Consider the cost-assignment scheme in Figure 14. Waiting-time cost-rates give higher precedence to τ_3 and then τ_2, while memory access cost-rates arrange them vice versa, since it is cheaper for τ_1 than τ_2 to bring variable B to the local memory, and similar for τ_2 and τ_3 concerning variable C.

	$\dot{c}_w$	$\dot{c}_m$
τ_1	1	5
τ_2	2	10
τ_3	3	15

Fig. 14. Waiting and memory access cost rates for the first case

In the second case we will focus on the cost of the waiting time of a specific task. As seen in Figure 15 our cost rate assignments gives the lowest priority, in terms of waiting time, to τ_3.

	$\dot{c}_w$	$\dot{c}_m$
τ_1	3	5
τ_2	3	10
τ_3	0	15

Fig. 15. Waiting and memory access cost rates for the second case

In both cases, our aim is to explore a part of the design space and achieve better cost solutions than a use of standard scheduling principles would give.

Verification. After verification of the first case, the diagrams with schedules produced using an EDF-algorithm and by UPPAAL CORA can be seen in Figure 16. Please note that each horizontal square in the schedule diagrams represent a duration of 10 time units. The cost of the optimal run produced by UPPAAL CORA is

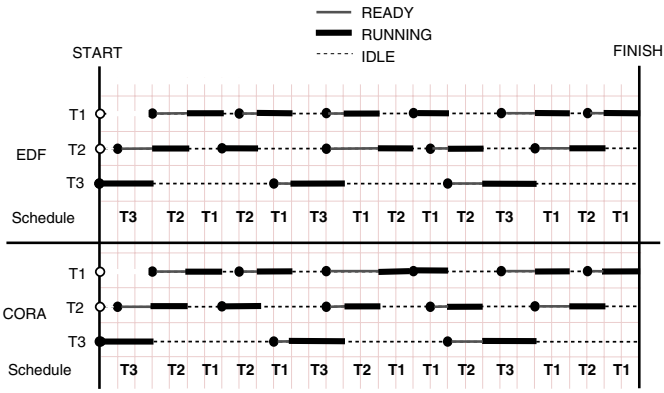

Fig. 16. Schedule diagrams of the tasks in Figure 14

for the first case

$$\sum_{i=1}^{3}(cost^{w}_{inf}(\tau_i) + cost^{m}_{inf}(\tau_i)) = 1005$$

After the verification of the second case, the diagrams with the schedules produced using and EDF-algorithm and by UPPAAL CORA can be seen in Figure 17. The cost of the optimal run produced by UPPAAL CORA is for the second case

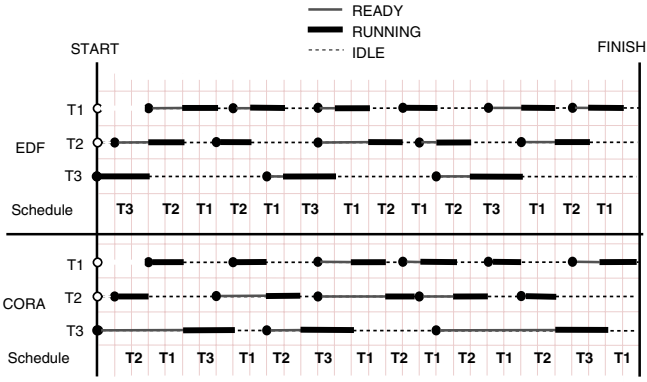

Fig. 17. Schedule diagrams of the tasks in Figure 15

$$\sum_{i=1}^{3}(cost^{w}_{inf}(\tau_i) + cost^{m}_{inf}(\tau_i)) = 1045$$

Analysis. The results of the first case in Figure 16 show that the schedule provided by UPPAAL CORA differs at one point only from that of the EDF algorithm i.e. the third time τ_1 and τ_2 are run. UPPAAL CORA determines that the schedulability

property can be upheld by choosing τ_2 instead of τ_1, which the EDF-algorithm would choose. This choice leads to a smaller cost.

For the second case, examining UPPAAL CORA's schedule in Figure 17, three main points can be observed:

i. τ_3 is scheduled after τ_2 if it is possible.

ii. τ_2 is scheduled after τ_1 if it is possible.

iii. The only case where the above two strategies do not apply is the first run, where these strategies would lead to a missed deadline of τ_3.

With a 2 GHz Pentium 4 machine with 512 MB of RAM, it takes about 1 second running verifications on our model. This duration applies for experiments without optimization in Sections 5. It takes about 25 seconds to perform verification on the same model when optimization is performed by UPPAAL CORA.

6 Summary

The aim of this ongoing work is cost analysis of embedded systems, and in this paper, we have a simple model comprising tasks running on an execution platform consisting of hardware components such as processing elements and local and external memories and a real-time operating system. We have modelled the system as priced timed automata. In the model we have introduced costs for tasks' waiting time and memory access, and we have used UPPAAL CORA to give cost-optimized schedules for systems. The results indicate that the approach could be fruitful in connection with design space exploration of embedded systems.

Acknowledgement

Comments and suggestions from Jan Madsen are greatly appreciated.

References

[1] Alur, R. and D. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.

[2] Andreas, G. B., *Processor/memory co-exploration on multiple abstraction levels* (2003).

[3] Behrmann, G., A. David and K. G. Larsen, *A Tutorial on* UPPAAL, Lecture Notes in Computer Science **3185** (2004), pp. 200–236.

[4] Brekling, A., "Modelling and Verification of MPSoC," Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU (2006).

[5] Kim, S., C. Im and S. Ha, *Efficient exploration of on-chip bus architectures and memory allocation*, in: *International Conference on Hardware/Software Codesign and System Synthesis* (2004), pp. 248–253.

[6] Larsen, K., G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson and J. Romijn, *As cheap as possible: Efficient cost-optimal reachability for priced timed automata*, Lecture Notes in Computer Science **2102** (2001), pp. 493+.

[7] Liu, C. L. and J. W. Layland, *Scheduling algorithms for multiprogramming in a hard-real-time environment*, Journal of the ACM **20** (1973), pp. 46–61.

[8] Mahadevan, S., M. Storgaard, J. Madsen and K. Virk, *Arts: a system-level framework for modeling mpsoc components and analysis of their causality*, in: *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (2005), pp. 480–483.

[9] Rasmussen, J. I., K. G. Larsen and G. Behrmann, *Priced timed automata: Algorithms and applications*, Lecture Notes in Computer Science **3657** (2005), pp. 162–182.

[10] Subramani, K., K. G. Larsen and J. I. Rasmussen, *On using priced timed automata to achieve optimal scheduling*, Formal Methods in System Design **29** (2006), pp. 97–114.