Workload Characterization Model for Tasks with Variable Execution Demand [*]

Alexander Maxiaguine   Simon Künzli   Lothar Thiele

Computer Engineering and Networks Laboratory

Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

E-mail: {maxiagui,kuenzli,thiele}@tik.ee.ethz.ch

Abstract

The analysis of real-time properties of an embedded system usually relies on the worst-case execution times (WCET) of the tasks to be executed. In contrast to that, in real world applications the running time of tasks may vary from execution to execution, e. g. in multimedia applications. The traditional worst-case analysis of the system then returns overly pessimistic estimates of the system performance. In this paper we propose a new effective method to characterize tasks with variable execution requirements, which leads to tighter worst-case bounds on system performance and better use of available resources. We show the applicability of our approach by a detailed study of a multimedia application.

1. Introduction

Minimization of cost and power consumption are important objectives in real-time embedded system design. Scheduling of tasks on processors has a great influence on these goals. The selection of the scheduling policy relies on the proper knowledge of task characteristics.

For real-time systems, a number of task models have been developed for schedulability analysis. In such models, behavior of a task is abstracted by a set of timing parameters. Typically, a task is characterized by an arrival pattern (sporadic, periodic etc.), deadlines and the execution time. Combined with other constraints, the parameters are used in schedulability tests to determine whether the system satisfies a set of given real-time requirements or not.

In schedulability tests for hard real-time systems, for which missing a deadline is regarded as a system failure, assumptions about a task's behavior must be made. In particular, in many existing analysis frameworks *each* instance of a task is assumed to take the worst case execution time (WCET) to finish. This assumption, although save, turns out to be too pessimistic for a large class of realistic applications. Loose bounds, obtained from the schedulability analysis under such an assumption, may lead to system implementations overprovisioned with resources and, as a result, having unreasonably high costs and/or power consumption.

There are many examples of applications (see e. g. [7]), in which task execution times have a great variability. In many cases, the worst case processing requirement happens rarely resulting in a high ratio of WCET to the average execution time of a task [9]. Scheduling such tasks, using, for instance, the rate-monotonic policy [8] under the WCET-assumption, results in a significant slack time.

To overcome this problem, some researchers (e.g. in [11], [3]) have proposed to model execution time of a task as a random variable. Using these models can improve processor utilization, but at the expense of permitting a certain (controlled) level of missed deadlines. It precludes employing these models for analysis of hard real-time systems.

In this paper, we propose a new approach to characterize the processing load produced by a task. On one hand, in comparison to the model assuming WCET for each task instance, our model provides *tighter bounds* on processing load generated by the task. This is because it can account for the variability of task execution time, i.e. it represents correlation between task execution times. On the other hand, in contrast to probabilistic approaches, our method provides *guaranteed bounds* on the task's processing load, since it still considers the worst case (and the best case) load, which is produced by a *sequence* of task executions. To achieve this, the model *implicitly* includes information about all possible request sequences for the task. The task behavior (with respect to the generated processing load) is abstracted by so called *workload curves*, which we introduce and formally define in Section 2. The proposed model is generic enough to be easily incorporated into existing scheduling frameworks, which we show in Section 3.

Related Work. Early task models for real-time analysis were annotated with a fixed WCET per job issued [8]. Then several new task characterization models were introduced to cope with variable task execution times. Using the System Properties Interval (SPI) model proposed in [13], a task

---

is characterized by its WCET and in addition with its best-case execution time (BCET). Every execution of a task will therefore have an execution time somewhere between its WCET and BCET. In the SPI model, as in the work of Wolf in [12], processes can have different modes with different intervals for execution times. Our research mainly relies on these concepts. We propose a method to characterize sequences of such process activations (i.e. modes) with bounds and show how these bounds can be used in the analysis.

Baruah [2] uses demand-bound functions to characterize variable workload imposed by sequences of tasks with conditional execution. This characterization model is orthogonal to our model in a sense that we do not try to model timing relations within tasks caused by conditional execution, but we are interested only in event *sequences*, where events of different types trigger tasks. Both models can be easily combined into a powerful analytical framework.

2. Characterization Model for Tasks

2.1. The Workload Curves

Let us consider a task, τ, that is executed on a processor. $[E_1, E_2, E_3, ...]$ is a sequence of events, that trigger the task τ. Each event E_i in that sequence is tagged with a certain type $t \in T$, where T denotes a finite set of all possible event types. The function $type(E_i)$ returns the type of ith event in the sequence.

Similar to [13], we can characterize the execution requirement imposed by an event type t with an interval $[bcet(t), wcet(t)] \in \mathbb{R}_{>0}$, where $bcet(t)$ and $wcet(t)$ denote BCET and WCET of t.

The following functions will tell us how many processor cycles will be consumed in the best case and in the worst case by any subsequence of k events starting from jth event in the event sequence:

$$\gamma^b(j,k) = \sum_{i=j}^{j+k-1} bcet(type(E_i)) \qquad j,k \in \mathbb{Z}_{>0}$$

$$\gamma^w(j,k) = \sum_{i=j}^{j+k-1} wcet(type(E_i)) \qquad j,k \in \mathbb{Z}_{>0}$$

Further, we define $\gamma^b(j,0) = 0$ and $\gamma^w(j,0) = 0 \ \forall j$. See Figure 1 for an example of sequence of events of different types. For this example event sequence $type(E_3) = a$ and $\gamma^b(3,4) = 5$.

The relations are not yet very useful for task characterization in general, because we will have to keep track of all possible triggering sequences. Therefore, we provide the following upper and lower bounds on the requested processing resources:

Definition 1 *Workload Curves*
An upper (or lower) workload curve $\gamma^u(k)$ (or $\gamma^l(k)$) gives
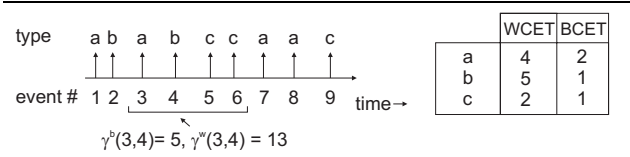

Figure 1. Event sequence with events of different types.

an upper (or lower) bound on the number of cycles that are needed to process any k consecutive activations of a task τ

$$\gamma^u(k) = \max_{\forall j \in \mathbb{Z}_{\geq 0}} \gamma^w(j,k) \qquad (1)$$

$$\gamma^l(k) = \min_{\forall j \in \mathbb{Z}_{\geq 0}} \gamma^b(j,k) \qquad (2)$$

In other words, any k consecutive executions of a task originate a processing demand of at most $\gamma^u(k)$ and at least $\gamma^l(k)$ processor cycles.

From the definition of workload curves, we can deduce some important properties:

- The workload curves are strictly increasing sequences.
- We can define the pseudo inverse $\gamma^{u-1}(e) = \max_{k \in \mathbb{Z}_{\geq 0}}\{k : \gamma^u(k) \leq e\}$ and $\gamma^{l-1}(e) = \min_{k \in \mathbb{Z}_{\geq 0}}\{k : \gamma^l(k) \geq e\}$. Then one can show that $\gamma^u(k) \leq e \Leftrightarrow \gamma^{u-1}(e) \geq k$ and $\gamma^l(k) \geq e \Leftrightarrow \gamma^{l-1}(e) \leq k$ and $\gamma^{u/l-1}(\gamma^{u/l}(k)) = k$.
- The worst case and best case execution times of a task equal $\gamma^l(1)$ and $\gamma^u(1)$, respectively.

The workload curves capture in a *compact way* all different possible sequences of task executions that can occur in an application. In that sense, the workload curve does not stand for an instance of task execution sequence, but for a class of execution sequences.

Note that the workload curves only take task activations into account. They are not based on any form of event timing. In order to include timing, we have to combine them with event models, which describe the temporal behavior of task activation.

To be applicable in hard real-time analysis the curves shall represent *guaranteed bounds*. It means that they shall be obtained by analytical means based on information available about the analyzed real-time application. This information may be explicitly contained in a specification of system and its environment and/or be derived from various constraints, which always hold true for all application's event patterns. In Subsection 2.2 we will show an example of how the workload curves can be obtained analytically.

Another way to construct the workload curves is by analysis of event traces (e.g. obtained from system's environment). It shall be used when it is not possible to guarantee constraints for the event patterns. In this case, the workload curves represent guaranteed bounds for this trace only (and therefore cannot be used for hard real-time analysis in general). However, they still can be very useful in many high-level analytical design exploration frameworks for systems with soft real-time constraints. This is because they provide a more precise abstraction of task behavior compared to single-valued and interval-based characterizations of task execution requirements. We will demonstrate this property in Subsection 3.2.

2.2. Illustrative Example

Example 1 *(Polling Task) Consider a task periodically polling for some event. If the event is detected, the task processes it with execution time e_p. Otherwise, the processing step is skipped (resulting in a shorter execution time e_c). The minimum, θ_{min}, and maximum, θ_{max}, inter-arrival times of the event stream, as well as the polling period T are known. T is smaller than θ_{min}, because we want to achieve a small response time for the task. For simplicity, let us assume that the task always finishes its execution before the next activation.*

For this problem setup we will now analytically derive the upper and lower workload curves. The maximum number of events detected in any k consecutive activations of the task is given by $n_{max}(k) = 1 + \left\lfloor \frac{k \cdot T}{\theta_{min}} \right\rfloor$, and the minimum number is given by $n_{min}(k) = \left\lfloor \frac{k \cdot T}{\theta_{max}} \right\rfloor$.

With these formulas we can easily find upper and lower bounds on the number of processor cycles that will be requested by the task in any k consecutive executions, by considering the execution requirements for the event processing e_p and e_c, respectively:

$$\gamma^u(k) = n_{max}(k)e_p + (k - n_{max}(k))e_c$$

$$\gamma^l(k) = n_{min}(k)e_p + (k - n_{min}(k))e_c$$

We can give tighter bounds than what would be achievable under the pessimistic assumption that all requests take WCET e_p to complete. In Figure 2 we can see the gain implied by the use of workload curves as grey-shaded areas. The uppermost curve named WCET shows the execution requirement if we perform a traditional worst-case analysis (with only single value for WCET), where the curve named BCET, represents the result of a best-case analysis (where all requests take e_c to compute). Using the upper and lower workload curve $\gamma^{u,l}$ we are therefore able to characterize a task execution behavior more precise than we could using the execution time interval-based approach.
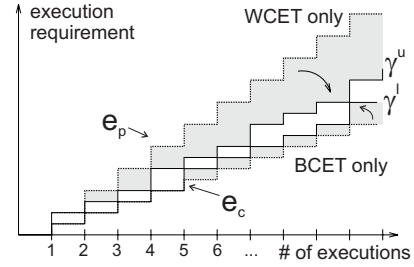
Figure 2. Workload curves for the polling task with $\theta_{min} = 3T$, $\theta_{max} = 5T$

3. Applications of Workload Curves

In this section we demonstrate how the workload curves can be applied for system-property analysis of real-time systems. We give two examples. The first example is based on a classical scheduling policy, the *rate-monotonic scheduling algorithm* [8]. We show how a well-known schedulability test for the rate-monotonic scheduler can be improved by using the workload curves. The second example shows how the workload curves can be applied in conjunction with a framework for performance evaluation of streaming applications to provide tighter bounds on buffer backlogs and delays of processed streams [5].

3.1. Rate-monotonic Schedulability Analysis

A rate-monotonic scheduler (RMS) uses a fixed-priority preemptive scheduling algorithm to schedule periodic task sets. The priorities are assigned to tasks based on their periods.

Lehoczky *et al.* [6] define a necessary and sufficient schedulability condition for RMS as follows. Given n periodic tasks $\tau_1, \ldots, \tau_n$ and the following expressions

$$W_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil \tag{3}$$

$$L_i = \min_{0 < t \le T_i} W_i(t)/t, \qquad L = \max_{1 \le i \le n} L_i$$

with C_i and T_i denoting WCET and period of task τ_i respectively, RMS can schedule τ_i iff $L_i \le 1$. The whole task set is schedulable iff $L \le 1$. Tasks are labeled with index i such that $T_1 \le T_2 \le \ldots \le T_n$, and have relative deadlines equal to their respective periods. (Refer to [6] for exact formulation and proofs of the schedulability condition.)

(3) gives the cumulative execution requirement generated by tasks $\tau_1, \ldots, \tau_i$ in the interval $[0, t]$. The term $C_j \cdot \lceil t/T_j \rceil$ denotes the execution requirement produced by a single task τ_j. It simply counts the number of arrivals of τ_j in the interval $[0, t]$ and multiplies this number by WCET of

the task. To achieve a tighter schedulability bound we need to replace the term $C_j \cdot \lceil t/T_j \rceil$ with the workload curve $\gamma_j^u(.)$, and pass the number of arrivals of τ_j given by $\lceil t/T_j \rceil$ as a parameter to the function.

$$W_i^*(t) = \sum_{j=1}^{i} \gamma_j^u(\lceil t/T_j \rceil) \qquad (4)$$

$$L_i^* = \min_{0 < t \le T_i} W_i^*(t)/t, \qquad L^* = \max_{1 \le i \le n} L_i^*$$

Since by the definition the function, $\gamma_i^u(k)$ returns the worst case execution requirement for any k consecutive instances of τ_i, and this requirement is less than or equal to kC_i, the following relations hold true

$$W_i^*(t) \le W_i(t) \quad , \quad L_i^* \le L_i \quad , \quad L^* \le L \qquad (5)$$

From (5) we can conclude that by using the workload curves in the RMS test we can obtain schedulability bounds that are at least as good as the bounds produced under the assumption of WCET for every task instance.

## 3.2. System-Level Performance Analysis of Streaming Architectures

Streaming architectures usually contain a number of processing elements (PEs) interconnected by buffers through which the PEs exchange streams of processed data. When such systems are designed, natural questions to answer are: How should the buffers be sized? How fast should the PEs be? System-level performance analysis aims at providing answers to these questions. Here we use an analytical framework which is based on the theory initially developed for analysis of integrated services networks, called *"Network Calculus"* (see [5] and references therein). The two central concepts of the framework are arrival and service curves.

An *arrival curve*, denoted by $\alpha(\Delta)$, gives an upper bound on the number of packets seen in the flow within any time interval Δ. A *service curve*, denoted by $\beta(\Delta)$, gives a lower bound on the amount of service that a flow is guaranteed to receive at a network node within any time interval Δ. Given the arrival and service curves of a flow processed by a node in the network (as shown in Figure 3a), it is possible to compute the upper bound on the backlog B produced by the flow in the queue in front of the node [5]:

$$B \le \sup_{\Delta \ge 0}\{\alpha(\Delta) - \beta(\Delta)\} \qquad (6)$$

Figure 3b gives an intuitive interpretation of (6).

In what follows, we take (6) *as an example* to demonstrate how the workload curves can be combined with arrival and service curves to enable a system-level performance analysis of architectures with multiple programmable elements (PEs). The application of workload curves to other results of the Network Calculus theory
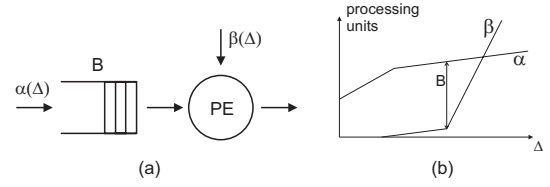
Figure 3. (a) A System consisting of a queue and a processing node (b) Graphical interpretation of (6) determining the upper bound on the backlog in front of a processing node

is straight forward and can be easily inferred from the provided example.

The idea of applying the Network Calculus theory to the analysis of real-time systems is not new. In [4] the authors have shown how the theory can be extended and used to address specific problems in system-level analysis of heterogeneous platform-based architectures. The notion of arrival curves is generalized to model *event flows*. (An event can be interpreted here in a wide sense: be it a packet, a sample of audio/video data or any other unit of work for a task executed on a processing resource.) A natural way to define the service curve for a task running on a programmable PE (like embedded processor) is to use minimum number of *processor cycles* supplied to the task by a processor scheduler in any time interval Δ.

The framework presented in [4] directly uses (6) to obtain the upper bound on the backlog in front of a PE. The backlog is expressed in terms of processor cycles (as opposed to number of packets, samples etc.) and denotes maximum *amount of work waiting in the queue* to be performed by the PE. We also note that, since in (6) the service curve is subtracted from the arrival curve, both curves shall be defined in common units i.e., processor cycles. To achieve this, in [4] the authors simply scale the event-based arrival curve (denoted by $\bar{\alpha}$) by a constant factor w i.e., $\alpha = w\bar{\alpha}$. For the worst-case analysis, we shall take w to be equal to WCET of a task. As we will show later in this section by experimental evaluation of an MPEG-2 decoder application, such an assumption may lead to substantial overestimation of needed processing resources.

In order to improve the bounds we employ workload curves for the conversion between number of events and associated processing cycles. The conversion and backward conversion for an arrival curve of a flow is shown in Figure 4.

We recast (6) in the following way.

$$\bar{B} \le \sup_{\Delta \ge 0}\{\bar{\alpha}(\Delta)) - \gamma^{u-1}(\beta(\Delta))\} \qquad (7)$$

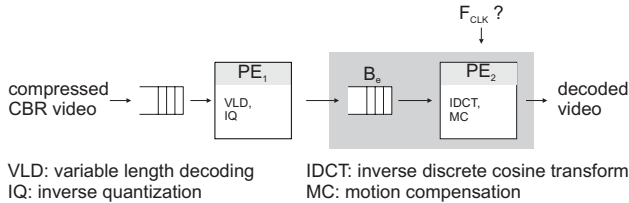where $\bar{B}$ denotes the maximum backlog measured in terms
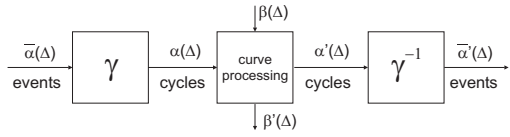
Figure 4. The composition of processing node and arrival curve conversion.

of *number of events* to be processed by a PE. We use the pseudo inverse of the upper workload curve $\gamma^{u-1}(.)$ to obtain the event-based service curve from the cycle-based service curve $\beta(\Delta)$.

We demonstrate the impact from applying the workload curves to performance analysis of streaming architectures by giving a case study of a realistic application.

Case Study. For the case study we selected an MPEG-2 decoder application. We consider an implementation of the MPEG-2 decoder on a streaming architecture consisting of two parallel embedded processors connected by a FIFO buffer. Figure 5 shows the architecture and the partitioning of the MPEG-2 decoder algorithm into two subtasks running on the respective processing elements (PEs). The subtask running on PE_1 performs the VLD and IQ functions on the incoming compressed video bitstream. Partially decoded *macroblocks* of video data on the output of PE_1 are sent via the FIFO buffer to the PE_2 running the second subtask, which performs the IDCT and MC functions on the incoming stream of macroblocks. We assume that no other tasks are executed by PEs, i.e. the MPEG-2 decoder subtasks receive the full processor capacity available on the PEs.

VLD: variable length decoding
IQ: inverse quantization

IDCT: inverse discrete cosine transform
MC: motion compensation

Figure 5. The architecture and mapping of the MPEG-2 decoder application on to it

Given the system described above, our goal is to determine the minimum required clock frequency of PE_2, F_{min}, such that the FIFO buffer between two PEs never overflows. We assume that the FIFO buffer size in macroblocks, b, is fixed and known. We also assume that the macroblock arrival process on the output of PE_1 is known and can be characterized by an arrival curve $\bar{\alpha}(\Delta)$.

To avoid overflow of the FIFO buffer we require that our target cycle-based service curve β satisfies the following constraint.

$$\beta(\Delta) \geq \gamma^u(\bar{\alpha}(\Delta) - b) \quad , \quad \forall \Delta \geq 0 \qquad (8)$$

In (8) we use the upper workload curve to obtain the worst-case processing requirement imposed by the the stream of macroblocks and "relaxed" by the buffer b.

Since we assume that the full processor resource is devoted to the decoding subtasks, the shape of the service curve β at PE_2 is given by $\beta(\Delta) = F\Delta$, where F denotes processor clock frequency. It follows that

$$F_{min}^{\gamma} = \max_{\Delta > 0} \left\{ \frac{\gamma^u(\bar{\alpha}(\Delta) - b)}{\Delta} \right\} \qquad (9)$$

If we knew *only* the WCET w of the task running on PE_2, then the upper workload curve would be determined as $\gamma_w^u(k) = wk$. Hence, if we characterize the task only with WCET, (9) can be written as

$$F_{min}^{w} = \max_{\Delta > 0} \left\{ \frac{w(\bar{\alpha}(\Delta) - b)}{\Delta} \right\} \qquad (10)$$

To see the impact from using the workload curves, we computed both values, F_{min}^{γ} and F_{min}^{w}. To get input data for the computations we had to use a simulator and obtain $\bar{\alpha}(\Delta)$ and $\gamma^u(k)$ curves by trace analysis. Using the simulator was necessary, because it is hard to derive analytically any useful constraints for a generic MPEG-2 stream of macroblocks. Patterns of macroblocks within the stream are not constrained (enough) by the MPEG-2 standard and dependent on a particular encoder implementation and on properties of the encoded video information.

We have performed simulations of the MPEG-2 decoder for 14 video clips. All the video clips were encoded with the following parameters: constant bit rate of 9.78 Mbit/s, main profile at main level, 25 fps, and resolution of 720×576 pixels.
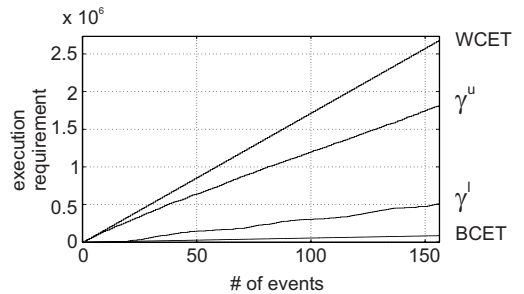
Figure 6. Workload curves

To determine $\bar{\alpha}(\Delta)$ and $\gamma^u(k)$, we analyzed traces obtained from the simulator using a window corresponding

to 24 full video frames. The resulting arrival and workload curves were obtained by taking maximum over all respective curves of individual video clips. Figure 6 shows the workload curves together with the WCET and BCET.

w was derived by taking $w = \gamma^u(1)$. Computed from the traces $\bar{\alpha}(\Delta)$, $\gamma^u(k)$, and w were plugged into (9) and (10) to calculate minimum clock frequency F_{min} of the PE_2 for the buffer size $b = 1620$ macroblocks (1 frame).

The results of the computations show that workload curves achieve significantly tighter bounds in comparison to conventional WCET-based characterization of tasks. We obtained $F_{min}^\gamma \approx 340MHz$, while $F_{min}^w \approx 710MHz$. This corresponds to over 50% of savings.

We have also performed system-level simulations with PE_2 running at the computed clock frequency F_{min}^γ. In Figure 7 we give the maximum backlogs that have been registered in the FIFO buffer in front of PE_2 during simulations. The backlogs are normalized to the FIFO buffer size. The plot shows results for all 14 video clips used in the experiments. Some of the bars in the plot are close to the maximum, which shows that the bounds obtained with our method, although based on worst-case analysis, represent sensible assumptions for system designers.

Our simulator consists of a transaction-level model of the architecture written in SystemC [10]. We used models of PEs based on *sim-profile* configuration of the SimpleScalar [1] instruction set simulator. Both PEs have an instruction sets similar to that used in MIPS3000 processors without floating point support. PE_1 is enhanced with special hardware support for video bitstream access, while PE_2 uses hardware acceleration of $IDCT$ function and a special block-based memory access mode.

4. Conclusions

In this paper, we have presented a new characterization method for tasks with variable execution demand. We have shown the application of this method in context of two different frameworks for real-time embedded systems analysis. In both cases, the application of the proposed task characterization method led to improved results compared to the conventional WCET-based analysis. The concept of workload curves is not restricted to the applications presented in this paper, but is generic enough to be embedded into other frameworks for real-time system analysis.

References

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[2] S. K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.
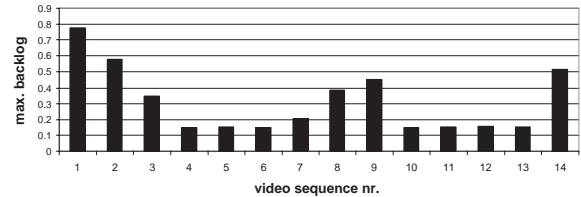
Figure 7. Maximal backlogs registered in the FIFO buffer in front of PE_2 running at computed clock frequency F_{min}^γ

[3] G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 279–288. IEEE Computer Society, 2002.

[4] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, pages 190–195, Munich, Germany, March 2003.

[5] J. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag, 2001.

[6] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium – 1989*, pages 166–171. IEEE Computer Society Press, 1989.

[7] Y.-T. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the 32nd ACM/IEEE conference on Design automation conference*, pages 456–461. ACM Press, 1995.

[8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[9] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pages 134–139. ACM Press, 1999.

[10] SystemC homepage. http://www.systemc.org.

[11] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real Time Technology and Applications Symposium*, pages 164 – 173. IEEE Computer Society, 1995.

[12] F. Wolf. *Behavioral Intervals in Embedded Software: Timing and Power Analysis of Embedded Real-Time Software Processes*. Kluwer Academic Publishers, 2002.

[13] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich. SPI – A System Model for Heterogeneously Specified Embedded Systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):397 – 389, August 2002.