Characterizing Workload Correlations in Multi Processor Hard Real-Time Systems

Ernesto Wandeler Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
E-mail: {wandeler,thiele}@tik.ee.ethz.ch

Abstract

Modern embedded systems are typically integrated as multi processor system on chips, and are often characterized by the complex behaviors and dependencies that system components exhibit. Different events that trigger such systems normally cause different execution demands, depending on their event type as well as on the task they are processed by, leading to complex workload correlations. For example in data processing systems, the size of an events payload data will typically determine its execution demand on most or all system components, leading to highly correlated workloads. Performance analysis of such complex system is often very difficult, and conventional analysis methods have no means to capture the possible existence of workload correlations. This leads to overly pessimistic analysis results, and thus to too expensive system designs with considerable performance reserves. We propose an abstract model to characterize and capture workload correlations present in a system architecture, and we show how the captured additional system information can be incorporated into an existing framework for modular performance analysis of embedded systems. We also present a method to analytically obtain the proposed abstract workload correlation model from a typical system specification. The applicability of our approach and its advantages over conventional performance analysis methods is shown in a detailed case study of a multi processor system on chip, where the analysis results obtained with our approach are considerably improved compared to the results obtained with conventional analysis methods.

I. Introduction

Today's complex real-time embedded systems are often comprised of a combination of different hardware and software components that are triggered by a number of different incoming event streams, and that communicate via some communication network. Such systems are often integrated as a Multi processor System on Chip (MpSoC) and many alternatives for partitioning, allocation and binding lead to a large design space.

During the system level design process of such a complex system, a designer is typically faced with the questions whether the timing properties of a certain implementation will meet the design requirements, what architectural element will act as a bottleneck, or what the on-chip memory requirements will be. One of the major challenges in the design process is therefore to analyze these essential characteristics of a system architecture in an early design stage, to support the choice of important design decisions before much time must be invested in detailed implementations.

Obtaining tight results for the above mentioned characteristics of a system design is not only difficult because of the heterogeneity and complexity of the complete system, but also because of the possibly complex behaviors and dependencies of the different components in the system. Different events may arrive on incoming event streams of a component, and they may create different resource demands, depending on their event type as well as on the task they are processed by. Due to this, there often exist complex correlations in such systems. We can thereby differentiate between correlations in the occurrence of different event types on incoming event streams, which we call event stream correlations (see [17]), and correlations of different resource demands that an event of a given type causes on different system components, which we call workload correlations. Highly correlated resource demands are for example typical in data processing systems, where the resource demand that an event creates on a component directly depends on the size of its payload data. In such a system, an event with a large payload will typically cause a large resource demand on most or all components it is processed on, and in turn, an event with a small payload will cause a small resource demand on most or all components.

The system-level analysis of such complex and heterogeneous systems with a high degree of internal correlations is currently mainly based on simulation, using for example SystemC, or trace based simulation as in [10]. These techniques allow modeling and simulation of complex systems in any level of detail. However, in terms of completeness, simulation based approaches do not allow to obtain worst-case results, because any concrete simulation-run can in general not guarantee to cover all corner cases. But most of all, simulation often suffers from long run-times and from a high set-up effort for each new architecture and mapping to be analyzed, and is therefore not very well suited for performance analysis in early design stages.

Other existing techniques use probabilistic models to describe real-time systems, see e.g. [16]. Such models can capture statistical characteristics of complex components, the results obtained from an analysis with these probabilistic models will however again

be of probabilistic nature. Hence, they are not applicable for the analysis of hard real-time systems.

In contrast to the above mentioned methods, formal analytical methods allow to obtain the hard bounded results that are needed for the analysis of hard real-time systems. A well-known technique to model real-time components are thereby Timed Automata [1]. In [6], it was shown that timed automata can be used as task models for event-driven systems and that the schedulability problem in such a model can be transformed to a reachability problem for timed automata and is thus decidable. While we could model complex component behaviors in any level of detail using this technique, the limitation to synchronous communication in timed automata would require to explicitly model all buffers in a stream based application with asynchronous communication. This however would usually turn the analysis effort for a complete system to be prohibitive.

In [14], an analytical framework for system level performance analysis was proposed that is based on classical scheduling results from hard real-time system design. This framework uses a number of well known abstractions to capture the timing behavior of event streams and provides additional interfaces between them. Very recently, effort was put into the refinement of component interaction models in this framework [13], the models however do not explicitly consider workload correlations yet. In [7], some methods were presented that try to address the problem of correlations present in complex embedded systems, but only the event stream correlations on incoming event streams are considered, and not workload correlations between different system components. A general disadvantage of the framework presented in [14], and hence of all methods based on it, is its confinement to a limited number of event stream timing behaviors. Due to this, the framework can only capture very limited aspects of event streams that exhibit arbitrary timing behaviors.

On the other hand, powerful abstractions have been developed in the domain of communication networks, to model flow of data through a network. In particular the theoretical framework called Network Calculus [9] provides means to deterministically reason about timing properties of data flows in queueing networks. Real-Time Calculus [15] extends the basic concepts of Network Calculus to the domain of real-time embedded systems and in [2] a unifying approach to modular system level performance analysis with Real-Time Calculus has been proposed. It is based on a general event model, allows for hierarchical scheduling and arbitration, and can take computation and communication resources into account.

The framework presented in [2] has been successfully used in several case studies for hard real-time system analysis (see e.g. [3]). However, since it is also not capable to capture the above mentioned correlations, that are often present in complex embedded systems, the results obtained from performance analysis with [2] are sometimes overly pessimistic. In [17], a new method was presented that extended the existing framework by allowing to characterize event stream correlations. And in this work, we present a new method that extends the same framework and that allows to characterize, capture and exploit workload correlations in the performance analysis of complex systems.

Contributions

- We provide an in-depth insight to the problem of performance analysis of distributed hard real-time systems with varying worst-case and best-case resource demands. And we point out the performance reserves that often exist in such systems due to workload correlations on different system components, that are typically not considered by existing performance analysis methods.
- We present a new abstract model that allows to characterize and capture existing workload correlations between different components in a system architecture, and we show how the so captured information of workload correlations can be incorporated into the framework presented in [2]. Further, we present an analytic method to extract the workload correlation information from a typical system model.
- We show the applicability of the presented model and methods in a detailed case study of a Multi processor System on Chip, where the analysis results obtained using the new methods are considerably improved compared to the results obtained using conventional methods.

II. Workload Correlations – An Application Scenario

We first present an application scenario of a multi processor system on chip (MpSoC) with workload correlations, that will serve us as a case study throughout the paper, and that will help us to visualize the effects and the importance of the presented methods. In Section V, we provide the complete analysis of this application scenario, and discuss the obtained results.

An overview of the MpSoC in consideration is depicted in Fig. 1. The figure shows a two-processor system on chip, that must serve two event streams, both entering and leaving independently through the I/O-interfaces of the chip. Both event streams have hard-real time processing requirements.

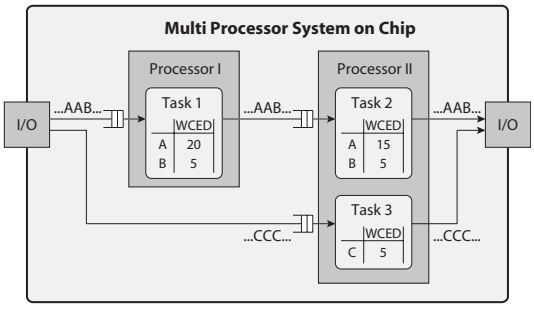

Fig. 1**.** A multi processor system on chip with correlating execution demands.

The first (upper) event stream is processed consecutively by the two tasks *Task 1* and *Task 2*, that are running on *Processor I* and *Processor II*, respectively. In the long-term average, the events on this stream arrive periodically with a period of 4ms. However, from time to time, bursts in the event arrival are possible on this stream. The events arriving on the first stream may be differentiated into two event types, with different execution demands. Events of type *A* have an execution demand of 20'000 cycles in *Task 1* and a demand of 15'000 cycles in *Task 2*. Events of type *B* on the other hand are less resource demanding and demand only 5'000 cycles, both in *Task 1* as well as in *Task 2*. Further let's

suppose that events of both types may arrive equally frequently and that we know nothing about possible correlations or arriving patterns of the two event types on the first event stream.

The second (lower) event stream is processed by *Task 3* that is running on *Processor II*. Events on this stream arrive periodically, with a period of 6ms and a maximal jitter of 1ms. The events on this event stream are all of type *C* and have an execution demand of 5'000 cycles in *Task 3*.

On *Processor II*, *Task 2* and *Task 3* are scheduled using a preemptive fixed priority scheduler, where, according to the *Rate Monotonic* scheduling scheme [11], *Task 2* (processing the stream with period $p = 4ms$) has the higher priority than *Task 3* (processing the stream with period $p = 6ms$). Since the processing of Task 3 has a lower priority than the processing of Task 2, the events on the second stream may experience a processing delay that depends on the interference from Task 2.

Let the speed of *Processor I* in the system be fixed to 6MHz. Our design problem will then be to determine upper bounds to the maximum delay that events on the second (lower) event stream experience during processing on the MpSoC, as a function of the processor speed of *Processor II*.

In this MpSoC, we nicely see the correlations of the workloads that the first stream is creating on the two processors: an event of type *A*, that creates a high demand of 20'000 cycles on *Processor I* again creates a high demand of 15'000 cycles on *Processor II*. On the other hand, an event of type *B* that only creates a demand of 5'000 cycles on *Processor I*, creates also a low demand of 5'000 cycles on *Processor II*.

In the next two sections, more details of this application scenario are provided in the example sections. The complete analysis, and the results obtained for our design problem are then presented and discussed in Section V.

Note: all results shown in this work were computed using discrete quantities, with a time quantization of 1ms and a resource quantization of 1'000 cycles. Therefore, in all computations and figures, one time unit equals 1ms, and one resource unit equals 1'000 cycles.

III. Modular Performance Analysis of Hard Real-time Systems

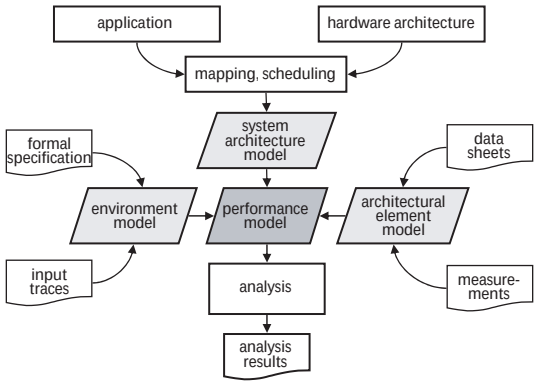

Fig. 2**.** Elements of Modular Performance Analysis and methods to obtain them.

Figure 2 shows an overview of an approach to Modular Performance Analysis (MPA). The central idea of MPA is to construct a performance model of a system, that captures the properties of the different system elements, such that they can be used for performance analysis. The approach described here is based on Real-Time Calculus [15], that itself is based on the theoretical framework called Network Calculus [9]. Network Calculus provides means to deterministically reason about timing properties of data flows in queueing networks. In the here described approach to MPA, performance models are built using three basic abstract elements: arrival curves model the environment (incoming and outgoing event streams), service curves model the capacity of architectural elements (resources) and performance components define the semantics of task execution in the system. The following sections give a short introduction to these basic elements and to the needed analysis methods.

A Event Stream Model (Environment Model)

Every event on an *event stream*, has a time when it occurs. This timing information of an event stream is captured using the concept of upper and lower *arrival curves* [5]:

DEFINITION 1 (ARRIVAL CURVES). *Let $R[s,t)$ denote the number of events that arrive on an event stream in the time interval $[s,t)$. Then, R, $\bar{\alpha}^u$ and $\bar{\alpha}^l$ are related to each other by the following inequality*

$$\bar{\alpha}^l(t-s) \leq R[s,t) \leq \bar{\alpha}^u(t-s), \forall s < t \tag{1}$$

where $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$.

In this model, the timing information of standard event stream models like *periodic*, *periodic with jitter*, *periodic with bursts*, *sporadic* or of any event stream with a known analytical timing behavior can be represented by an appropriate choice of $\bar{\alpha}^u$ and $\bar{\alpha}^l$ [2]. Moreover, it is also possible to determine the values of $\bar{\alpha}^u$ and $\bar{\alpha}^l$ corresponding to any given finite length arbitrary event trace, obtained for example from observation or simulation.

EXAMPLE 1. *Figure 3 shows the event stream models of the two input event streams that trigger the MpSoC in Fig. 1. From the arrival curves in the upper graph, we see that the first (upper) event stream can be modeled as a periodic stream with burst, with a period $p = 4ms$, a burst period $bp = 1ms$, a burst length $bl = 5ms$ and a maximum event inter arrival time $h = 7ms$. From the arrival curves in the lower graph, we see that the second (lower) event stream can be modeled as a periodic stream with jitter, where the period $p = 6ms$ and the jitter $j = 1ms$.*

B Resource Model (Architectural Element Model)

The capacity of a *resource unit* can be characterized using lower and upper *service curves* β^l and β^u [9]:

DEFINITION 2 (SERVICE CURVES). *Let $C[s,t)$ denote the number of processing or communication units available from a resource over the time interval $[s,t)$, then the inequality*

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s), \forall s < t \tag{2}$$

holds, where again $\beta^l(0) = \beta^u(0) = 0$.

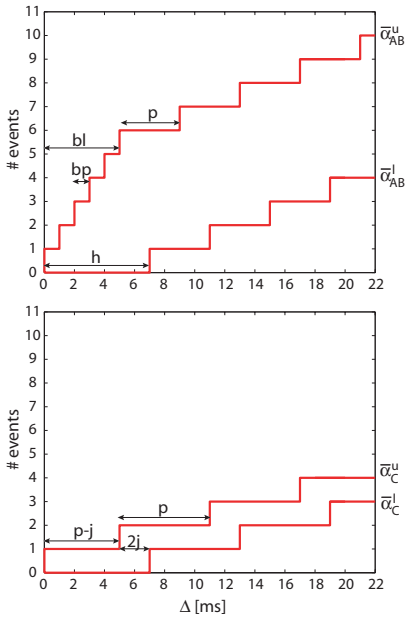

Fig. 3**.** The event stream model of the first (upper) and second (lower) input event streams that trigger the MpSoC in Fig. 1.

The service curves of a resource can be determined using data sheets, using analytically derived properties or by using measuring.

EXAMPLE 2. *Both processors of the MpSoC in Fig. 1 are initially unloaded. We therefore model both of them with as a periodic resource stream, with a quantization period* $p = 1ms$ *and a step height equalling the resource capacity during* $1ms$*. For* Processor I *with a processor speed of* $6MHz$*, this step height equals* 6 *resource units (*$6'000$ *cycles).*

C Performance Components and Real-time Calculus

Performance components are the basic building blocks to construct a performance model of a system. They model the application tasks in a system and define the semantics of how application tasks are executed on architectural elements. At the same time, they also build the basis for performance analysis.

In the presented framework, an incoming event stream, represented as a set of upper and lower arrival curves, flows into a FIFO buffer in front of a performance component. The performance component is then triggered by the events of this buffered incoming event stream. It generates events on an outgoing event stream, while being restricted by the availability of resources, represented as a set of upper and lower service curves. The outgoing event stream can again be represented as a set of upper and lower arrival curves, while the remaining resource capacity is represented as an outgoing set of upper and lower service curves.

We describe and analyze such a component using Real-Time Calculus [15]. Real-Time Calculus provides a set of formulas that relates incoming arrival curves α and service curves β to outgoing arrival curves α' and service curves β' in a performance component:

$$\alpha^{l\prime}(\Delta) = \min\{\inf_{0\le\mu\le\Delta}\{\sup_{\lambda\ge 0}\{\alpha^l(\mu+\lambda)-\beta^u(\lambda)\} + \beta^l(\Delta-\mu)\}, \beta^l(\Delta)\} \quad (3)$$

$$\alpha^{u\prime}(\Delta) = \min\{\sup_{\lambda\ge 0}\{\inf_{0\le\mu\le\lambda+\Delta}\{\alpha^u(\mu)+\beta^u(\lambda+\Delta-\mu)\} - \beta^l(\lambda)\}, \beta^u(\Delta)\} \quad (4)$$

$$\beta^{l\prime}(\Delta) = \sup_{0\le\lambda\le\Delta}\{\beta^l(\lambda)-\alpha^u(\lambda)\} \quad (5)$$

$$\beta^{u\prime}(\Delta) = \max\{\inf_{\lambda\ge\Delta}\{\beta^u(\lambda)-\alpha^l(\lambda)\}, 0\} \quad (6)$$

Further, in Real-Time Calculus, an upper bound to the maximum delay experienced by an event at a performance component is given by the following inequality:

$$delay \le \sup_{t\ge 0}\left\{\inf\{\tau\ge 0 : \alpha^u(t)\le\beta^l(t+\tau)\}\right\} \quad (7)$$

We may connect performance components into a network according to the model of a system architecture. Event flows that exit performance components (tasks) can be connected to an event flow input of another component and similarly, resource capacity that is not consumed by a performance component can be connected to the resource input of a lower priority performance component. Together with the models of all system resources (service curves) and incoming event streams (arrival curves), we obtain the performance model of a complete system.

D The Importance of Workload Transformations

As defined above, the arrival curves $\bar{\alpha}^l$ and $\bar{\alpha}^u$, used to model event streams, capture the number of events that arrive on an event stream in a given time interval. On the other hand, the service curves β^l and β^u denote the number of resource units (e.g. cycles) that are available on an architectural element in a given time interval. It becomes apparent, that in order to apply formulas (3)–(7), both the arrival curves and the service curves need to be expressed in the same base unit. For this, we can either transform the event based arrival curves $\bar{\alpha}^l$ and $\bar{\alpha}^u$ into resource based arrival curves, denoted by α^l and α^u. Or alternatively, we can transform the resource based service curves β^l and β^u into event based service curves, denoted by $\bar{\beta}^l$ and $\bar{\beta}^u$. It is important to note, that these transformations must be conservative, such that the analysis still provides hard bounded results.

In the most simple case, where only one type of events is processed in a task, and where each event always creates exactly the same, constant resource demand (i.e. the worst-case demand equals the best-case demand), resource based quantities can be transformed lossless into event based quantities and vice versa. Event based quantities simply need to be multiplied with the resource demand of a single event, and resource based quantities need to be divided by the same number.

In a more general case, this transformation is not as simple anymore, and we need more powerful methods to capture the relation between the number of events and the created resource demand in a task. Workload Curves, that were first introduced in [12] provide the needed expressiveness:

DEFINITION 3 (WORKLOAD CURVES). *Let* $W(u)$ *denote the*

total resource demand created in a task by u consecutive events of an incoming event stream. For every event sequence s, the lower workload curve γ^l and the upper workload curve γ^u satisfy the relation:

$$\gamma^l(v-u) \leq W[u,v) \leq \gamma^u(v-u), \forall u < v \tag{8}$$

Therefore, any sequence of *e* consecutive events, will create a total resource demand of at least $\gamma^l(e)$ and at most $\gamma^u(e)$ on an architectural element.

EXAMPLE 3. *We know nothing about possible correlations or arriving patterns of the different event types on the event streams triggering the MpSoC in Fig. 1. Therefore, the resource demand that* e *consecutive events could create, is only bounded by the worst (best) case, where all* e *events create the maximum (minimum) possible resource demand. This leads to the following workload curves for the different tasks in Fig. 1: $\gamma^l_{\mathrm{T1}}(e) = e \cdot 5$ resource units, $\gamma^u_{\mathrm{T1}}(e) = e \cdot 20$, $\gamma^l_{\mathrm{T2}}(e) = e \cdot 5$, $\gamma^u_{\mathrm{T2}}(e) = e \cdot 15$, and $\gamma^l_{\mathrm{T3}}(e) = \gamma^u_{\mathrm{T3}}(e) = e \cdot 5$.*

Using workload curves and their pseudo-inverse, arrival and service curves can be transformed from event based to resource based quantities and vice versa as follows:

$$\alpha^l(\Delta) = \gamma^l(\bar{\alpha}^l(\Delta)) \qquad \beta^l(\Delta) = \gamma^l(\bar{\beta}^l(\Delta)) \tag{9}$$

$$\alpha^u(\Delta) = \gamma^u(\bar{\alpha}^u(\Delta)) \qquad \beta^u(\Delta) = \gamma^u(\bar{\beta}^u(\Delta)) \tag{10}$$

$$\bar{\alpha}^l(\Delta) = \gamma^{u\,-1}(\alpha^l(\Delta)) \qquad \bar{\beta}^l(\Delta) = \gamma^{u\,-1}(\beta^l(\Delta)) \tag{11}$$

$$\bar{\alpha}^u(\Delta) = \gamma^{l\,-1}(\alpha^u(\Delta)) \qquad \bar{\beta}^u(\Delta) = \gamma^{l\,-1}(\beta^u(\Delta)) \tag{12}$$

E Performance Components and Workload Transformations

We mentioned before that all workload transformations must be conservative in order to guarantee hard bounded analysis results. Mathematically expressed, the following relation (and similar relations for all other arrival and service curves) must hold with all workload curves:

$$\bar{\alpha}^u(\Delta) \leq \gamma^{l\,-1}(\gamma^u(\bar{\alpha}^u(\Delta))) \tag{13}$$

And for general workload curves, where the upper curve γ^u does not equal the lower curve γ^l (i.e. the worst-case execution demand of all possible events does not equal the best-case execution demand), the right side of (13) will normally be substantially larger than the left side. This means, that while the application of one workload transformation (i.e. from event based to resource based quantities, or the other way around) in general leads to realistic hard bounds (depending of course on the accuracy of the workload curves), a two-way workload transformation (from event based to resource based quantities and back again to event based quantities, or the other way round) often leads to overly pessimistic, yet still valid bounds.

As a consequence of this, we must take care to use as few workload transformations as possible during a system performance analysis, and in particular we should omit two-way workload transformations whenever possible. Figure 4 shows a schema how to use a minimal number of workload transformations in the analysis of a performance component. The bright shaded data flows are resource based quantities, while the dark shaded data flows are event based quantities. In the two blocks labeled *WLT*, the workload transformations are applied according to (9)–(12), and in the two blocks labeled *RTC*, (3)–(6) are used to compute outgoing arrival and service curves in the respective base unit.

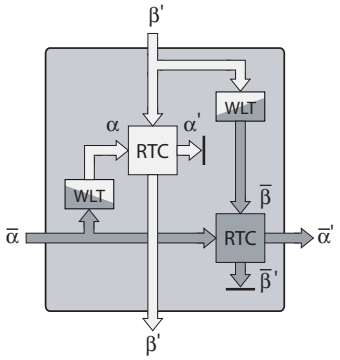

Fig. 4**.** Resource based (bright) and event based (dark) data flows in the analysis of a performance component.

In the analysis schema shown in Fig. 4, (9)–(12) are applied to the resource based curves α and β (in the bright block labeled *RTC*) as well as to their event based counterparts $\bar{\alpha}$ and $\bar{\beta}$ (in the dark block labeled *RTC*). Both sets of curves, α and β as well as $\bar{\alpha}$ and $\bar{\beta}$, model the same properties of a system, but both use a different method of abstraction (resource units vs. events), and both may therefore capture different aspects of the same system properties (see Example 4). However, in the analysis schema shown in Fig. 4, the outgoing resource based arrival curve α' as well as the outgoing event based service curve $\bar{\beta}'$ get lost after the analysis, together with the different aspects of system properties modeled by them. It would of course be possible to obtain an event based outgoing arrival curve $\bar{\alpha}'$ from a workload transformation of the outgoing arrival curve α' (and similar we could obtain a β' from $\bar{\beta}'$). But due to the two-way transformation step that would be introduced by this, this curve will normally be overly pessimistic, as we will see in Example 4.

EXAMPLE 4. *Figure 5 shows the event based (upper graph) and resource based (lower graph) outgoing arrival and service curves of Task 1 in the MpSoC in Fig. 1. The event based arrival curves $\bar{\alpha}_{\mathrm{T1}} = \bar{\alpha}_{\mathrm{AB}}$ (see Example 1) and the resource based service curves $\beta_{\mathrm{T1}} = \beta_{\mathrm{PI}}$ (see Example 2) served as initial input for the computation. With this input, all event based and resource based outgoing arrival and service curves were computed according to the analysis schema described above and depicted in Fig. 4. Additionally, the back-transformations of α'_{T1} and $\bar{\beta}'_{\mathrm{T1}}$ were computed by applying (9)–(12): $\beta'^{*}_{\mathrm{T1}} = \gamma_{\mathrm{T1}}(\bar{\beta}'_{\mathrm{T1}})$ and $\bar{\alpha}'^{*}_{\mathrm{T1}} = \gamma^{-1}_{\mathrm{T1}}(\alpha'_{\mathrm{T1}})$.*

Let us first discuss the event based upper outgoing arrival curve $\bar{\alpha}^{u'}_{\mathrm{T1}}$. Intuitively, we can say that the maximum possible event stream output (restricted by $\bar{\alpha}^{u'}$) will occur when the maximum possible event stream input (restricted by $\bar{\alpha}^u$) occurs together with the maximum possible resource availability (restricted by β^u). According to (4), the minimum possible resource availability (restricted by β^l) also plays a role (it restricts the maximum initial buffer fill level), but we will not consider it for the discussion here. From β^u_{T1} we know that in the best case there are 6 *resource units available per millisecond. And since the smaller event* B *of the input event stream of* Task 1 *only requires* 5 *resource units to be*

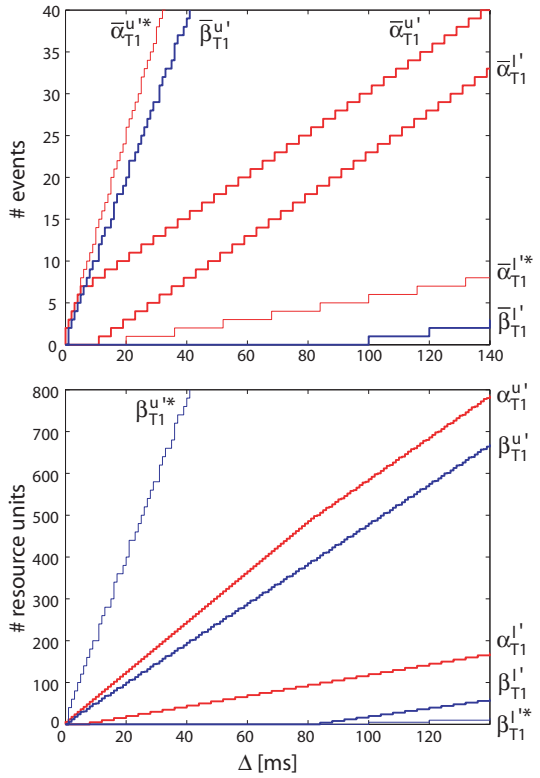

Fig. 5**.** The event based (upper graph) and resource based (lower graph) output curves of Task 1 in the MpSoC in Fig. 1.

processed, we know that we could in the best case process 1.2 *events per millisecond. This is expressed in* $\bar{\beta}^{u}_{\mathrm{T1}}$*. Because of this high event based processing capacity (based on the case where all arriving events are of the smaller type* B*), the maximum possible event stream output (*$\bar{\alpha}^{u'}_{\mathrm{T1}}$*) is only restricted by the availability of triggering input events (*$\bar{\alpha}^{u}_{\mathrm{T1}}$*), as we see in Fig. 5.*

Let us now concentrate on the resource based upper outgoing arrival curve $\alpha^{u'}_{\mathrm{T1}}$*. The worst possible resource demand is created by the input event stream when all arriving events are of the more resource demanding type* A*. In this case, a total execution demand of* $6 \cdot 20 = 120$ *resource units is created during a burst of only little more than* 5*ms. But in the same time, a maximum of only* $6 \cdot 6 = 36$ *resource units are available for processing, and therefore most of the arriving events will be buffered in the input buffer of* Task 1*. In this phase, the output event stream rate is restricted by the availability of resources. However, after a burst, the events on the input event stream will again arrive periodically, creating a maximum resource demand of* 20 *resource units every* 4*ms. And since in the same period a maximum of* 24 *resource units are available for processing, the input buffer of* Task 1 *will eventually be emptied again after a burst, leading into a phase where the maximum output event stream rate is restricted only by the availability of triggering input events. From the change of slopes of* $\alpha^{u'}_{\mathrm{T1}}$ *(and* $\beta^{l'}_{\mathrm{T1}}$*) in Fig. 5, we see that the first phase, where the output event stream is restricted by the availability of resources, is never longer than* 84*ms.*

When we are now using (12) to transform $\alpha^{u'}_{\mathrm{T1}}$ *back to* $\bar{\alpha}'^{*}_{\mathrm{T1}}$*, we must assume that with every investment of* 5 *resource units we could in the best case generate an event of type* B *on the output event stream. But by taking this assumption, we neglect the fact that for the computation of* $\alpha^{u'}_{\mathrm{T1}}$*, we assumed that all events are of type* A*. This decoupling of worst-case and best-case events leads to the very pessimistic upper bound* $\bar{\alpha}'^{*}_{\mathrm{T1}}$*. Similar thoughts can explain the pessimism of all other back-transformed curves in Fig. 5.*

From the results shown and discussed in Example 4, we clearly see that the resource based outgoing arrival curve α' of a performance component may carry valuable information (as for example the presence of a phase where the maximum available resources restrict the output event stream rate in Example 4) that may not be present in the event based outgoing arrival curve $\bar{\alpha}'$, or in the back-transformed curve $\bar{\alpha}'^{*}$. In the next section, we present a new method, that allows to consider the information contained in α' in the analysis of succeeding performance components.

IV. Workload Correlations and Performance Analysis of Multi Processor Systems

As we have seen in Section III–C, we may connect performance components to a network, to build the performance model of a complete multi processor system. In the existing framework, we may however only use event based arrival curves $\bar{\alpha}$ and resource based service curves β to interconnect performance components. The reason for this restriction is that outgoing resource based arrival curves of one component are not directly related to ingoing resource based arrival curves of another component, since the same event causes in general a different resource demand in two components, when it is processed by two different tasks. Similar thoughts can be made for event based service curves.

As a direct consequence of this restriction, existing methods have no means to use the information captured by an outgoing resource based arrival curve α' of one component in the analysis of a succeeding component. To overcome this problem, we will now present a new method that will allow us to directly transform the outgoing resource based arrival curve α' of one performance component into an ingoing resource based arrival curve of another performance component, by exploiting the knowledge of workload correlations existing between these components. No two-way transformation step will be needed for this, and therefore much of the information captured in α' will be preserved.

Figure 6 shows the performance model of the MpSoC of Fig. 1, where the outgoing resource based arrival curve of *Task 1* is directly connected to the ingoing resource based arrival curve of *Task 2*, using the new workload correlation based interconnection method.

The central part of this interconnection path is the box labeled *WCC*, where the resource based arrival curves are transformed, by the use of what we will call Workload Correlation Curves (WCC). Workload Correlation Curves allow to analytically characterize and capture existing workload correlations between two components:

DEFINITION 4 (WORKLOAD CORRELATION CURVES). *Let us suppose that* r_{T1} *resource units are invested in a task* T_1 *to process incoming events and generate outgoing events. Further, let* $W_{\mathrm{T1}\rightarrow\mathrm{T2}}(r_{\mathrm{T1}})$ *denote the total execution demand that is created*

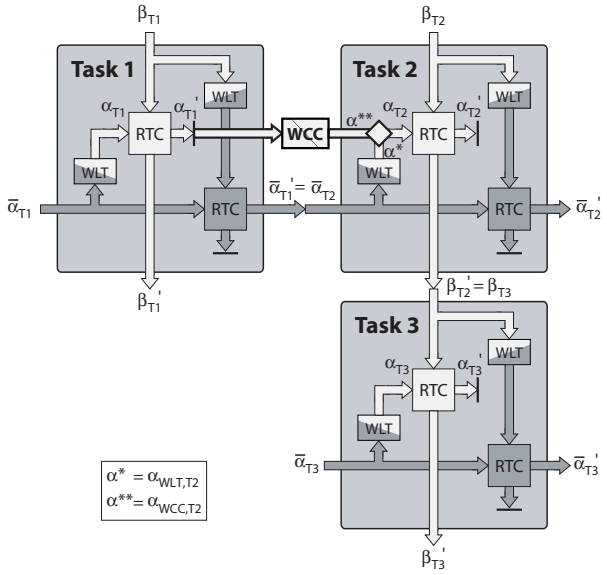

Fig. 6**.** The performance model of the MpSoC of Fig. 1, with the new component interconnection method highlighted.

in a task T_2 by the arrival of the events that were generated on the output of the preceding task T_1 by the investment of r_{T1} resource units. Then, for any number of invested resources r_{T1} in T_1, the lower workload correlation curve $\delta^l_{\mathrm{T1}\to\mathrm{T2}}(r)$ and the upper workload correlation curve $\delta^u_{\mathrm{T1}\to\mathrm{T2}}(r)$ satisfy the relation:

$$\delta^l_{\mathrm{T1}\to\mathrm{T2}}(v-u) \leq W_{\mathrm{T1}\to\mathrm{T2}}[u,v) \leq \delta^u_{\mathrm{T1}\to\mathrm{T2}}(v-u), \forall u < v \tag{14}$$

Therefore, whenever r_{T1} resources are invested in a task T_1, the outgoing events that are generated by this resource investment will create a total resource demand of at least $\delta^l_{\mathrm{T1}\to\mathrm{T2}}(r_{\mathrm{T1}})$ and at most $\delta^u_{\mathrm{T1}\to\mathrm{T2}}(r_{\mathrm{T1}})$ resource units in a succeeding task T_2.

EXAMPLE 5. *Figure 7 shows the workload correlation curves between* Task 1 *and* Task 2 *of the MpSoC in Fig. 1. Note that with the availability of a single resource unit (the 20^{th}),* Task 1 *can generate an event of type* A *on its output event stream, leading to an execution demand of* 15 *resource units in* Task 2*. But after this initial event of type* A*, the constant processing of events of type* B *(assuming that the input buffer of* Task 1 *never underflows) leads to the maximum workload on* Task 2*.*

Using workload correlation curves, outgoing resource based arrival curves of a task T_1 can be transformed directly into ingoing resource based arrival curves of a task T_2:

$$\alpha^l_{\mathrm{WCC,T2}}(\Delta) = \delta^l_{\mathrm{T1}\to\mathrm{T2}}(\alpha^{l\,\prime}_{\mathrm{T1}}(\Delta)) \tag{15}$$
$$\alpha^u_{\mathrm{WCC,T2}}(\Delta) = \delta^u_{\mathrm{T1}\to\mathrm{T2}}(\alpha^{u\,\prime}_{\mathrm{T1}}(\Delta)) \tag{16}$$

In parallel, as we see in Fig. 6, we can still obtain the ingoing resource based arrival curves $\alpha^l_{\mathrm{WLT,T2}}$ and $\alpha^u_{\mathrm{WLT,T2}}$ from a workload transformation of the ingoing event based arrival curves, by applying (9) and (10) to $\bar{\alpha}^l_{\mathrm{T2}}$ and $\bar{\alpha}^u_{\mathrm{T2}}$. And since all so obtained

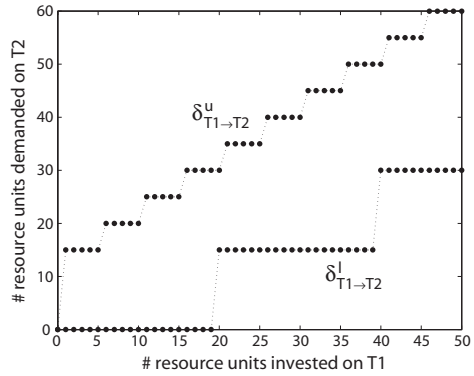

Fig. 7**.** The workload correlation curves between Task 1 and Task 2 in the MpSoC in Fig. 1.

ingoing resource based arrival curves, $\alpha^l_{\mathrm{WLT,T2}}$ and $\alpha^u_{\mathrm{WLT,T2}}$ as well as $\alpha^l_{\mathrm{WCC,T2}}$ and $\alpha^u_{\mathrm{WCC,T2}}$, are hard bounds, we can combine the curves by taking the pairwise maximum and minimum, respectively:

$$\alpha^l_{\mathrm{T2}}(\Delta) = \max(\alpha^l_{\mathrm{WLT,T2}}(\Delta), \alpha^l_{\mathrm{WCC,T2}}(\Delta)) \tag{17}$$
$$\alpha^u_{\mathrm{T2}}(\Delta) = \min(\alpha^u_{\mathrm{WLT,T2}}(\Delta), \alpha^u_{\mathrm{WCC,T2}}(\Delta)) \tag{18}$$

V. Case Study

In this section we analyze the MpSoC presented in Section II. We will in particular compute upper bounds to the maximum delay that events of the second (lower) input stream experience during processing on the MpSoC, as a function of the processor frequency f_{PII} of *Processor II*. Note that in the system analysis presented in this section, we neglect the delay that events experience by the I/O-interfaces and by the on-chip communication network.

For the system performance analysis, we use the performance model of the MpSoC, shown in Fig. 6. To model the processing capacity of the initially unloaded *Processor II*, we use a periodic service curve $\beta_{\mathrm{PII}} \equiv \beta_{\mathrm{T2}}$ with period $p = 1$ms and a step-height equalling $f_{\mathrm{PII}}/1'000'000$ resource units. The event based arrival curves $\bar{\alpha}_{\mathrm{T1}} \equiv \bar{\alpha}_{\mathrm{AB}}$ and $\bar{\alpha}_{\mathrm{T3}} \equiv \bar{\alpha}_{\mathrm{C}}$ (see Example 1) and the resource based service curves $\beta_{\mathrm{T1}} \equiv \beta_{\mathrm{PI}}$ (see Example 2) serve as further system inputs.

A Analytic Bound on the Maximum Delay

According to (7), we need α^u_{T3} and β^l_{T3} to compute an upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II* of the MpSoC.

We can immediately compute α^u_{T3} by applying (10) to $\bar{\alpha}^u_{\mathrm{T3}}$.

To obtain β^l_{T3}, we need in a first step to compute the outgoing arrival curves $\alpha^{u\,\prime}_{\mathrm{T1}}$ and $\bar{\alpha}^{u\,\prime}_{\mathrm{T1}}$ of *Task 1*. We compute these curves according to the analysis schema described in Section III–E, using (3)–(6) and (9)–(12). The resulting outgoing curves are shown in Fig. 5 and are discussed in Example 4.

According to the performance model shown in Fig. 6, the event based upper outgoing arrival curve of *Task 1* can directly be linked

to the event based upper ingoing arrival curve of *Task 2*: $\bar{\alpha}^u_{\mathrm{T2}} = \bar{\alpha}^{u'}_{\mathrm{T1}}$.

To compute the resource based upper ingoing arrival curve of *Task 2*, we first conventionally apply (10) to $\bar{\alpha}^u_{\mathrm{T2}}$, leading to $\alpha^u_{\mathrm{WLT,T2}}$. Applying (16) to $\alpha^{u'}_{\mathrm{T1}}$ will in turn lead to $\alpha^u_{\mathrm{WCC,T2}}$, and applying (18) to $\alpha^u_{\mathrm{WLT,T2}}$ and $\alpha^u_{\mathrm{WCC,T2}}$ will finally lead to α^u_{T2}. All these variants of resource based ingoing arrival curves of *Task 2* are shown in Fig. 8.

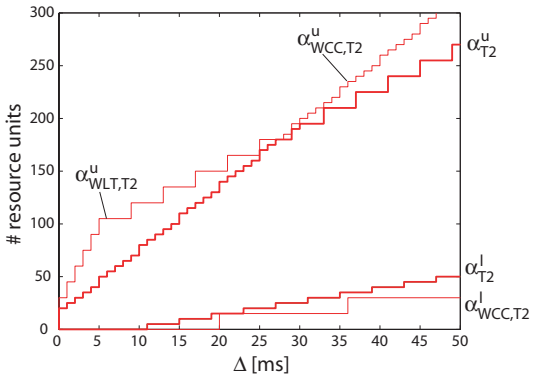

Fig. 8**.** All variants of resource based input arrival curves of Task 2 in the MpSoC in Fig. 1.

Then, to compute the resource based lower outgoing service curve of *Task 2*, we apply (5) to α^u_{T2} and β^l_{T2}, leading to $\beta^{l'}_{\mathrm{T2}}$, that can be linked to β^l_{T3}. This allows us finally to use (7) to compute an upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II*:

$$d_{\mathrm{T3}} \quad \leq \quad \sup_{t \geq 0} \Big\{ \inf\{\tau \geq 0 : \alpha^u_{\mathrm{T3}}(t) \leq \beta^l_{\mathrm{T3}}(t+\tau)\} \Big\}$$

To see the impact of the new presented methods to the obtained analysis results, we do the same performance analysis using the conventional analysis methods. There, we apply (5) directly to $\alpha^u_{\mathrm{WLT,T2}}$ and β^l_{T2}, leading to $\beta^{l'}_{\mathrm{WLT,T2}} \equiv \beta^l_{\mathrm{WLT,T3}}$. We then use again (7) to compute an upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II*:

$$d_{\mathrm{WLT,T3}} \quad \leq \quad \sup_{t \geq 0} \Big\{ \inf\{\tau \geq 0 : \alpha^u_{\mathrm{T3}}(t) \leq \beta^l_{\mathrm{WLT,T3}}(t+\tau)\} \Big\}$$

In Fig. 9, the resource based upper ingoing arrival curve α^u_{T3} is shown together with the resource based lower ingoing arrival curve $\beta^l_{\mathrm{WLT,T3}}$, that was obtained using conventional analysis methods, and the resource based lower ingoing arrival curve β^l_{T3}, that was obtained using the new presented methods that allow to incorporate existing workload correlations into performance analysis.

B Experimental Results

We used the above described analysis to compute the upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II*, for processor frequencies f_{PII} ranging from 6MHz to 20MHz, in 1MHz steps. The obtained results for the analysis using the conventional methods as well as

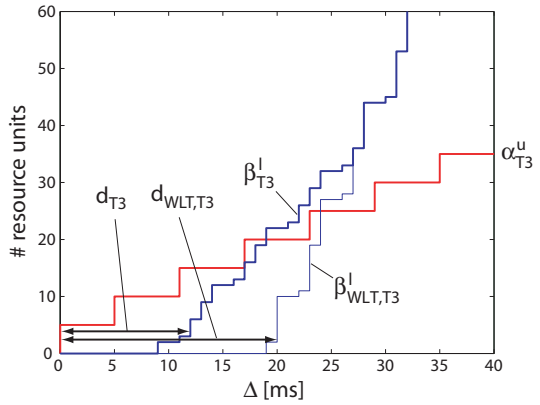

Fig. 9**.** The resource based upper ingoing arrival curve together with both, the conventional and the new variant of the resource based lower ingoing service curves of Task 3 in the MpSoC in Fig. 1. (For $f_{\mathrm{PII}} = 8$MHz)

for the analysis with the new presented methods are visualized in the graph in Fig. 10.

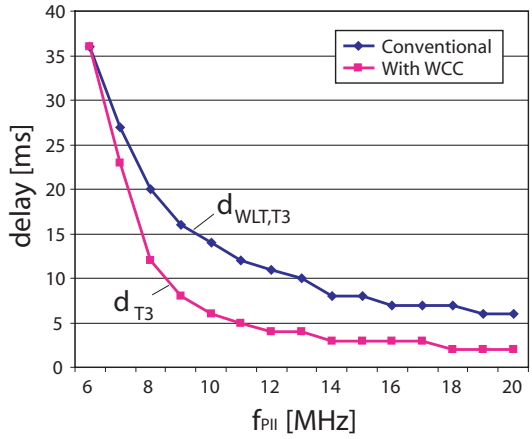

Fig. 10**.** Comparison of the analytic upper bound to the maximal delay experienced by an event during processing by Task 3 in the MpSoc in Fig. 1, when analyzed using conventional methods (upper curve), and when analyzed using the new presented methods that consider workload correlations (lower curve).

From the results shown in Fig. 10, we see that the use of the new presented analysis methods leads to considerably tighter analytic bounds for the investigated design problem. The obtained results clearly point out the performance reserves that often exist in complex systems due to workload correlations, and that are usually not considered by existing performance analysis methods. For example, while we need to run *Processor II* at a processor speed of 14MHz to guarantee an upper bound to the maximum delay of 8ms with existing performance analysis methods, we can guarantee the same bound already at a processor speed of 9MHz with the new presented analysis methods. This corresponds to a processor speed reduction of more than 35%.

VI. Computing Workload Correlation Curves

In this section, we present a method to analytically compute the workload correlation curves between two components. The presented method is based on a slightly reduced version of a task model that was first introduced in [18]. While we use this reduced version for the sake of simplicity, the method could also easily be extended to work with the full version of the task model introduced in [18].

A Task Model

We model the execution demand behavior of a task by a state-transition graph that relates incoming events to best-case and worst-case resource demands as well as to outgoing events.

DEFINITION 5 (TASK AUTOMATON). *A task automaton F_T is a tuple $(S, S^0, \Sigma_I, \Sigma_O, D, T)$, where S is a set of states, $S^0 \subseteq S$ is a set of initial states, Σ_I is the set of accepted incoming event types and Σ_O is the set of generated outgoing event types. Further, D is a demand function $D : S \times \Sigma_I^* \times \Sigma_O^* \times S \rightarrow [\mathbb{Z}_{\geq 0}, \mathbb{Z}_{\geq 0}]$. This function determines the upper and lower bound of the resources needed by a task in order to process an incoming event, generate an outgoing event and change its internal state. Finally, $T \subseteq S \times \Sigma_I \times \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \times \Sigma_O \times S$ is a set of transitions.*

The task always starts in an initial state, and if $s \xrightarrow{\sigma_I/[d_l,d_u]/\sigma_O} s'$ and if the task is triggered by the incoming event σ_I, the task has a resource demand of at least d_l and at most d_u resource units to emit an event σ_O and change its state from s to s'.

We can get the needed information to model a task from a formal specification of the task or from data sheets. Moreover, it may also be possible to obtain the information from program analysis of the task code.

EXAMPLE 6. *Figure 11 shows the task models of the two simple tasks running on* Processor I *and* Processor II *of the multi processor system on chip shown in Fig. 1.*

Both tasks are modeled using only a single state. Upon arrival of an event A, Task 1 *has an execution demand of* 20 *resource units to process the event and to generate again an event* A *on its output. But the processing of an event* B *only requires* 5 *resource units and leads again to an event* B *on the output of* Task 1.

In Task 2 *on the other hand, the processing of an event* A *only requires* 15 *resource units, while the processing of an event* B *again requires* 5 *resource units.*

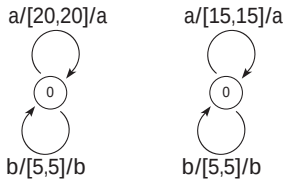

Fig. 11**.** The task models of Task 1 (left) and Task 2 (right) of the multi processor system on chip in Fig. 1.

The task models shown in Fig. 1 are of the most simple type of task automata we could think of, where F_T consists of only a single state with a self-loop for every accepted event type. The presented task model however also allows to model much more complex task behaviors including for example the modeling of caching effects. And with the full version of the task model introduced in [18], it is even possible to model for example arbitrary up- and down-sampling of event rates in tasks. An example of such a more complex task automaton is provided in [18].

B Workload Correlation Automaton

The first step towards computing the workload correlation curve of two task automata, is to build a so-called workload correlation automaton F_{WC}. This workload correlation automaton is built from the product automaton of the two task automata, where we have a transition between two states if and only if two transitions existed in the initial task automata, with the output event type of the first task automaton transition equalling the input event type of the second task automaton transition (we say that the first task automaton and the second task automaton synchronize on such transitions):

$$
\begin{aligned}
S_{WC} &= S_{\mathrm{T1}} \otimes S_{\mathrm{T2}} \\
S^0_{WC} &= S^0_{\mathrm{T1}} \otimes S^0_{\mathrm{T2}} \\
T_{WC} &= \{((u,v),[d_{l,\mathrm{T1}},d_{u,\mathrm{T1}}],[d_{l,\mathrm{T2}},d_{u,\mathrm{T2}}],(u',v'))| \\
&\quad (u,\sigma_{I,\mathrm{T1}},[d_{l,\mathrm{T1}},d_{u,\mathrm{T1}}],\sigma_{O,\mathrm{T1}},u') \in T_{\mathrm{T1}} \\
&\quad \wedge (v,\sigma_{I,\mathrm{T2}},[d_{l,\mathrm{T2}},d_{u,\mathrm{T2}}],\sigma_{O,\mathrm{T2}},v') \in T_{\mathrm{T2}} \\
&\quad \wedge \sigma_{O,\mathrm{T1}} = \sigma_{I,\mathrm{T2}}\}
\end{aligned}
$$

After building all transitions, we can remove all states and state trajectories that are not reachable from an initial state anymore. This finally leads to the workload correlation automaton F_{WC} that is described by the tuple $(S_{WC}, S^0_{WC}, T_{WC})$.

EXAMPLE 7. *Figure 12 shows the workload correlation automaton of the two task models shown in Fig. 11.*

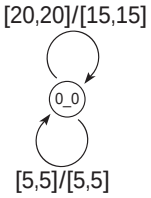

Fig. 12**.** The workload correlation automaton of the two task models shown in Fig. 11.

C Workload Correlation Curve

From the workload correlation automaton F_{WC}, we can compute the upper workload correlation curve $\delta^u_{\mathrm{T1}\rightarrow\mathrm{T2}}$ following a three step procedure.

Step 1:. On the transitions of the workload correlation automaton F_{WC}, we retain the lower resource demand of the first task *T1* and the upper resource demand of the second task *T2*, and discard the other information:

$$s \xrightarrow{[d_{l,\mathrm{T1}},d_{u,\mathrm{T1}}]/[d_{l,\mathrm{T2}},d_{u,\mathrm{T2}}]} s' \Rightarrow s \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s'$$

Step 2:. We then transform the so obtained automaton into an automaton where every transition corresponds to a created demand of one resource unit on the first task *T1* and where the weights on the transition corresponds to the created demand on the second task *T2*. For this, we first replace every transition $s \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s'$ with $d_{l,\mathrm{T1}} > 0$ with a state trajectory according the following rule:

$$s \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s' \Rightarrow s \underbrace{\xrightarrow{0} s_{(1)} \xrightarrow{0} \ldots \xrightarrow{0} s_{(d_{l,\mathrm{T1}}-1)} \xrightarrow{d_{u,\mathrm{T2}}}}_{d_{l,\mathrm{T1}} \text{ transitions}} s'$$

After this, we replace all transitions $s' \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s''$ with $d_{l,\mathrm{T1}} = 0$ as follows:

$$s \xrightarrow{w} s' \xrightarrow{0/d_{u,\mathrm{T2}}} s'' \Rightarrow s \xrightarrow{w+d_{u,\mathrm{T2}}} s''$$

Step 3:. In this automaton, we interpret the execution demand d as the weight of a transition. The weight $w^u(e)$ of the maximum-weight path with length e then equals the value of the upper workload correlation curve $\delta^u_{\mathrm{T1}\to\mathrm{T2}}(e)$. Techniques to efficiently find maximum-weight paths of any length in a weighted graph are described in [4] and [8] and an efficient algorithm with a computational complexity of $O(e|S||T|)$ was presented in [17].

To compute the lower workload correlation curve $\delta^l_{\mathrm{T1}\to\mathrm{T2}}(e)$, we must follow the same three step procedure as described above, but in *Step 1* we need to retain the upper resource demand $d_{u,\mathrm{T1}}$ of the first task *T1* and the lower resource demand $d_{l,\mathrm{T2}}$ of the second task *T2*. In *Step 3*, the value of the lower transfer curve $\delta^l_{\mathrm{T1}\to\mathrm{T2}}(e)$ then equals the weight $w^l(e)$ of the minimum-weight path with length e in the automaton obtained in *Step 2*.

VII. Conclusions

We introduced a new abstract model that allows to characterize and capture existing workload correlations between different components in a system architecture, and we presented an analytic method to extract this abstract workload correlation model from a typical system specification. We further presented the needed methods to incorporate the new model, and the system information captured by it, into an existing framework for modular performance analysis of embedded systems. We have shown the applicability of the presented model and methods in a detailed case study of a multi processor system on chip, where a considerable improvement of certain analytic worst-case bounds was achieved compared to conventional analysis methods. The improved analysis results clearly point out the performance reserves that often exist in complex embedded systems due to workload correlations that are usually not considered by existing analysis methods.

VIII. REFERENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] 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, March 2003.

[3] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5):641–665, April 2003.

[4] G. Cohen, D. Dubois, J. P. Quadrat, and M. Voit. A linear-system-theoretic view of discrete-event processes and its use for performance avaluation in manufacturing. *IEEE Transactions on Automatic Control*, AC-30(3), 1985.

[5] R. Cruz. A calculus for network delay. *IEEE Trans. Information Theory*, 37(1):114–141, 1991.

[6] C. Ericsson, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *In proc. of RTCSA'99*. IEEE Press, 1999.

[7] M. Jersak, R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded systems design. In *Proc. 7th Design, Automation and Test in Europe (DATE)*, 2004.

[8] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, (23):309–311, 1978.

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

[10] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

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

[12] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Proc. 7th Design, Automation and Test in Europe (DATE)*, 2004.

[13] R. Racu, K. Richter, and R. Ernst. Calculating task output event models to reduce distributed system cost. In *GI/ITG/GMM Workshop*, February 2004.

[14] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *IEEE Computer*, 36(4):60–67, April 2003.

[15] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[16] 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.

[17] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 450–459, 2004.

[18] E. Wandeler and L. Thiele. Abstracting functionality for modular performance analysis of hard real-time systems. In *Asia South Pacific Design Automation Conference (ASP-DAC)*, 2005.