Modular Performance Analysis Using Real-Time Calculus

Rebekka Roßberg
Institute of Embedded Systems/ Real-Time Systems
Ulm University
rebekka.rossberg@uni-ulm.de

Abstract

The system-level analysis, and therefore the optimization of embedded real-time systems, plays an important role in the design process. One way to perform such an analysis, is by using the powerful, yet simple, framework of modular performance analysis (**MPA**), which utilizes the formal analysis methods of real-time calculus (**RTC**) to evaluate various performance criteria of hard real-time systems. This analysis is done by building a system performance model, which sums up all the information, that is needed for the following performance analysis. In this paper an overview on RTC, MPA and their use is given and illustrated by a sample system.

I. INTRODUCTION

When designing an embedded real-time system there are many questions we need to be answered already in the early stages of the design cycle: What architecture to choose? What scheduling to use? How powerful do the CPUs need to be? Is the performance of the underlying execution platform sufficient to fulfill the given deadlines? These questions can be answered best by using system-level performance analysis.

In general there are different categories of approaches for evaluating embedded systems. Above all there are holistic, simulation based and formal analysis methods for system-level performance [1]. The latter category contains the modular performance analysis based on real-time calculus, which this paper is dedicated to.

With MPA, we want to have an analysis method, that can work with only little information about the system [1]. Additionally, we need to be able to evaluate, and therefore compare, numerous possible system models within a short time. This specifically sets the need for a short analysis time. Furthermore, we want the analysis process to be able, to deal with all kinds of different applications, environments, resource sharing strategies and hardware platforms [2], that a distributed, heterogeneous and highly complex multiprocessor embedded real-time system might offer. As a real-time system is defined as a system, that can always guarantee to complete its task before a given deadline [3], we need to know the guaranteed maximum response times of the system. However, having completely accurate results, knowing average case response times and being able to model details, is not really important yet.

A. About the Paper

Initially we will characterize the system data (section II) that is needed in order to perform the MPA and model this data using arrival and service curve sets in section III and section IV. In section VI we will build the system performance model out of HW/SW-components (section V). This model we are then going to analyze on performance (section VII) and sensitivity (section VIII) properties, in order to find answers to important design questions regarding the embedded real-time system. Finally, we are going to discuss (section IX) whether MPA really is a suitable framework for the context specified above.

While not proposing any new research, a general overview of MPA and RTC is given. It shall be an introduction into the topic and a guide for utilizing MPA to evaluate own embedded real-time systems. The paper is limited to the evaluation of hard real-time systems. Those are systems, in which a single missed deadline may lead to a complete failure of the system. Therefore, hardware and software have to operate strictly within the given time limits.

Most chapters will be accompanied by the example of an embedded real-time system. Here we will analyze, and later on optimize, the properties of a simple message-receiver device to illustrate the use of MPA. This example was made using the RTC Toolbox for Matlab. Therefore, the plots are designed using Matlab. The example itself is constructed by myself, yet inspired by [4].

B. Terminology

First up, we want to introduce the three main concepts of this paper, along with their correlations between each other.

Network calculus is a “mathematical approach to model network behavior” [5] and thus an approach to model the flow of data through the network. It provides a general formal framework, as well as useful tools, to compute the worst case bounds on different characteristics of the communication network, such as queuing delays and required buffer sizes [6]. To reveal those bounds, network calculus integrates information about the packet scheduling algorithms, the incoming events and much more [5].

When combining the results of network calculus with the theory of the min/max-plus linear system (see section A) and the theory of real-time scheduling, we end up with **RTC**: a framework, specifically designed for analyzing hard heterogeneous real-time systems [5]. Hence, most of the theorems of network calculus can now also be adapted to real-time systems. Here the incoming event streams and available system capacities are described using curves, which can then be transformed, whenever they are being processed at any system component. In this way hard upper and lower bounds of the embedded real-time system's characteristics, like delays and buffers, can be computed. The RTC framework was first proposed by [5] and later on extended by the framework of MPA in [7].

MPA defines a specific, RTC based, approach to perform a system-level performance analysis of a distributed embedded real-time system. Here "performance" is defined, as to what extent the system meets its answer time requirements [2]. Therefore, the performance analysis is the process of determining these properties. To characterize end-to-end delays, response time and other system properties, MPA now cuts the system into its components in a modular way. Afterwards all the data is combined into one big abstract system performance model, along with information about the environment, the components itself and the architecture. This model can be analyzed now using the formal possibilities of RTC to gain insight into the system, as well as to obtain hard upper and lower bounds to various system properties [8].

II. COLLECTING DATA FOR MODELLING THE SYSTEM

Before being able to build a system model, we must first collect some data about the application and the available resources.

A. General Approach

Quite little information is needed for constructing a system performance model, yet some data is fundamental for the following analysis. What are the specifications of the incoming event streams, including their patterns, periods and deadlines? Which of these events can occur in parallel? (Note that this information needs to be given by formal specifications, not from observation or simulation [1].) What resource demands do the tasks have? How are the processing capacities and patterns of the available system resources specified? What scheduling technique to use?

Here we can have very different degrees of detail, as sometimes we can design the new system pretty much freely, while other times we already got specific components and architectures to use given. Yet, in case we can freely design the system, we can now just set these parameters arbitrarily and later on, for example in section VIII, optimize them.

B. Sample System: A Simple Message Receiver

To illustrate the use of MPA we will design our own little embedded system. First the requirements to this system have

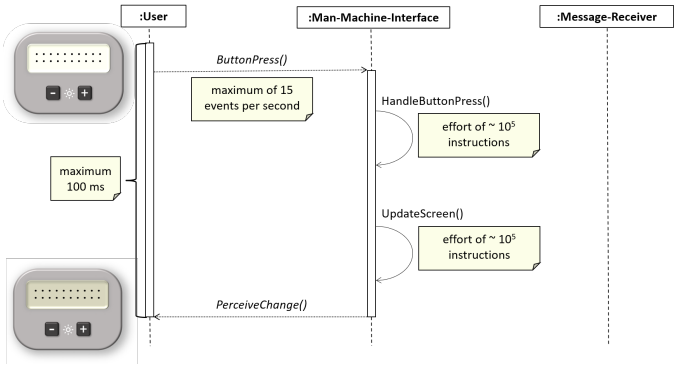

Fig. 1. Annotated UML Sequence Chart of the Use Case "changing the screen's brightness"

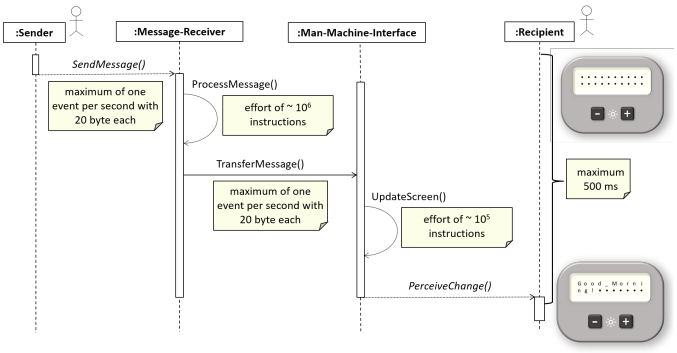

Fig. 2. Annotated UML Sequence Chart of the Use Case "receiving a message"

to be collected. We want to construct a simple gadget that can receive messages via a signal. Each of the messages consists out of one to 20 ASCII characters that are displayed to the device screen once the message got received. Besides receiving messages, the device's user can also adjust the screen's brightness by pressing the "brighter" or the "darker" button. Other functionalities will not be implemented, especially it will not be possible to compose or send any own messages.

Therefore, the system has only two use cases that can be described using UML sequence charts (see Figure 1 and Figure 2).

III. ARRIVAL CURVES

Event stream models describe how a system is being used by the environment: How often do events of a specific event stream arrive? Do these arrivals follow any kind of pattern? How much input data is provided to the system? How much data is generated by the system and fed back to its environment after the processing [3]? To answer these questions, we have to construct an own event stream model for each stream of the system, by specifying the event bound functions.

A. Event Bounds

We define the differential arrival function $R[s,t)$ as the total number of events that arrive within the time interval $[s,t)$: $s,t \in \mathbb{R}$ [9]. While R models one concrete trace of an event

stream, in the framework of RTC we usually characterize an event stream by its arrival curves [9]:

$$\overline{\alpha}(\Delta t) = [\overline{\alpha}^{u}(\Delta t), \overline{\alpha}^{l}(\Delta t)]$$

These bounds are related by

$$\forall t \leq s \leq 0 : \overline{\alpha}^{l}(t-s) \leq R[s,t) \leq \overline{\alpha}^{u}(t-s)$$

meaning that all possible traces of this event stream are located between the lower arrival curve $\overline{\alpha}^{l}(\Delta t)$ and the upper arrival curve $\overline{\alpha}^{u}(\Delta t)$. Thus, these curves set guaranteed lower and upper bounds on the number of incoming events at this particular event stream over any time interval of length Δt [10].

In contrast to $\overline{\alpha}(\Delta t)$, $\alpha(\Delta t)$ describes the amount of resource capacity an event stream demands over a period of time[9]. This specification is needed for most use cases and can be obtained by

$$[\alpha^{u}(\Delta t), \alpha^{l}(\Delta t)] = [\gamma^{u}(\overline{\alpha}^{u}), \gamma^{l}(\overline{\alpha}^{l})]$$

Here $\gamma(e) = [\gamma^{u}(e), \gamma^{l}(e)]$ defines the minimum and maximum work, that is required per number of incoming event **e** [3]. Usually γ can be simplified to the basic linear function $\gamma(e) = e \cdot WL$ with the task-specific workload of a single event being **WL**[1].

B. Handling Different Arrival Patterns

Various event arrival patterns with deterministic timing behavior, such as sporadic, periodic and periodic with jitter, can be efficiently abstracted into curves [7], [11].

In this paper we will focus on periodic event streams. Within these streams there will always be exactly one event occurring within the period **p**. In case we are observing a periodic event stream with a jitter **j**, that period p is guaranteed to be in $[p-j, p+j]$ [3]. It is important to notice, that those jitters do not accumulate: the 10th event will for example arrive within $[9p-j, 10p+j]$, not within $[9p-9j, 10p+10j]$. Furthermore, the minimum distance **d** between the arrivals of two events needs to be specified whenever dealing with a jitter.

The arrival curves without jitter (Figure 3) and with jitter (Figure 4) can now be modeled graphically or in the form of equations [11]:

$$\overline{\alpha}^{l}(\Delta t) = \left\lfloor \frac{\Delta t - j}{p} \right\rfloor$$

$$\overline{\alpha}^{u}(\Delta t) = \min\left\{ \left\lceil \frac{\Delta t + j}{p} \right\rceil, \left\lceil \frac{\Delta t}{d} \right\rceil \right\}$$

Those functions can also be approximated linearly to facilitate the upcoming calculations, yet this leads to less accurate results [1].

C. Arrival Curves of the Sample System

As the sequence charts Figure 1 and Figure 2 define the maximum amount of incoming events per event stream and time interval, we can easily model these streams using curves (see Figure 5). The workload of the various tasks are also given by the sequence charts. Therefore, we could construct the resource based arrival curves as well.

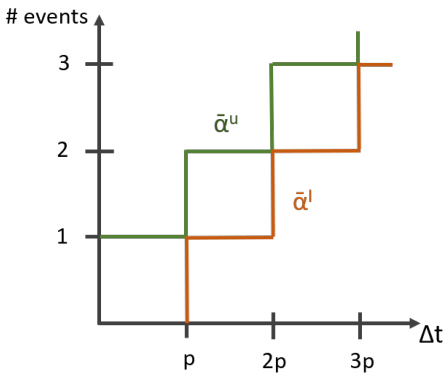

Fig. 3. Arrival curve with period p

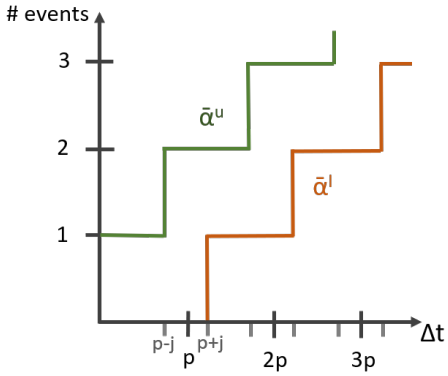

Fig. 4. Arrival curve with period p and jitter j

IV. SERVICE CURVES

Resource models provide information about the processing capacities of the resources that are available within a system. Thus, they characterize the service that can be provided by the system [3]. For every resource, whether communicational or computational, an own resource model has to be constructed.

A. Resource Bounds

We define the differential service function $C[s,t)$ as the total number of available resource units, e.g. processor cycles or transmittable bits on a bus, within the time interval $[s,t)$: $s,t \in \mathbb{R}$ [9]. While C models one concrete availability of a

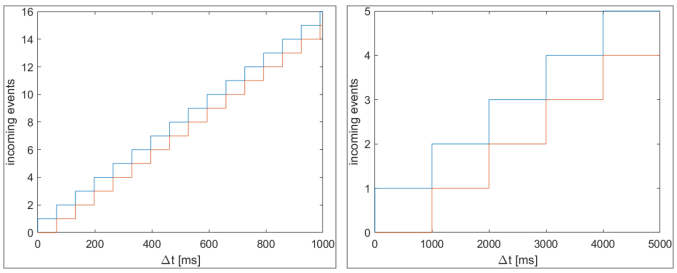

Fig. 5. Arrival curves of brightness events (left) and message events (right)

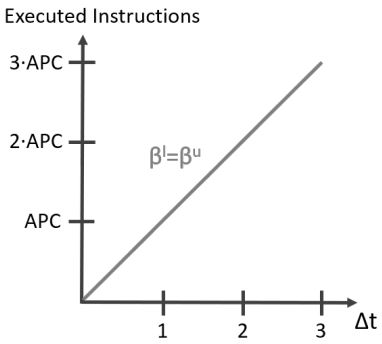

Fig. 6. Service curve of a simple CPU with full availability

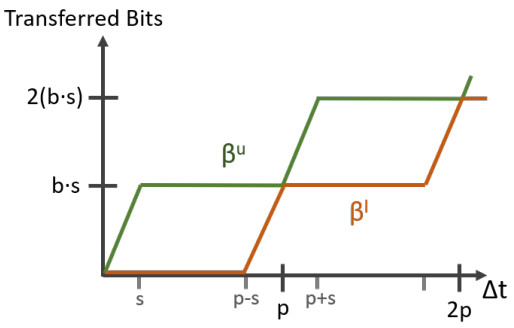

Fig. 7. Service curve of a simple TDMA bus

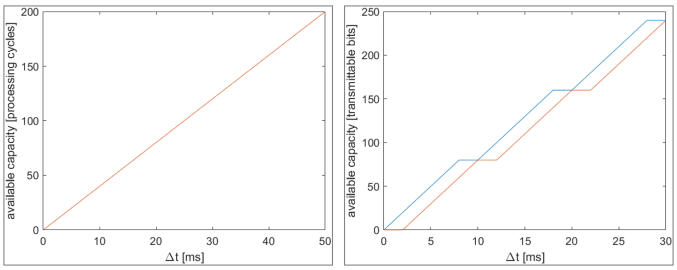

Fig. 8. Service curves of the sample system's 4 MIPS CPU (left) and it's TDMA bus (right)

resource, in the framework of RTC we usually characterize a resource by its service curves [9]:

$$\beta(\Delta t) = [\beta^u(\Delta t), \beta^l(\Delta t)]$$

The bounds are related by

$$\forall t \leq s \leq 0 : \beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s)$$

meaning that all possible processing capacities of this component are located between the lower service curve $\beta^l(\Delta t)$ and the upper service curve $\beta^u(\Delta t)$. Therefore, these curves set guaranteed lower and upper bounds on the processing units available at this resource over any time interval of length Δt [10]. In an analogue way to $\overline{\alpha}(\Delta t)$ we can also define $\overline{\beta}(\Delta t)$ as

$$[\overline{\beta}^u(\Delta t), \overline{\beta}^l(\Delta t)] = [(\gamma^l)^{-1}(\beta^u), (\gamma^u)^{-1}(\beta^l)]$$

$\overline{\beta}(\Delta t)$ directly depends on the number of events that can be processed per Δt, yet it only has very few use cases.

B. Handling Different Resource Availability Patterns

The capacities of various resources, such as resources with full availability, but also with fluctuating service capacities, can be efficiently abstracted into curves [3]. When dealing with a simple unloaded processor with full availability, the service curve can be modeled by the linear function

$$\beta^u(\Delta t) = \beta^l(\Delta t) = APC \cdot \Delta t$$

with **APC** being the constant number of available processing cycles per time interval (see Figure 6) [3].

A popular example for resources with fluctuating availability are time division multiple access (**TDMA**) busses. When communicating via a TDMA bus, information can be transmitted with a period **p** for **s** time units using the full bus bandwidth **b**. Afterwards there is a waiting period of $p - s$ time units until the bus gets allocated again (see Figure 7) [3].

C. Service Curves of the Sample System

Here we choose to use fully available CPUs as computational resources, with 4 or 5 million executable instructions per second (**MIPS**), Additionally we use a TDMA bus as a communicational resource. The available service of the bus is given by the bandwidth of 20 kilobits of transferrable information per second (**kbps**), a period of 10 and a slot length of 8. The service curves of these resources are given by Figure 8.

V. HW/SW-COMPONENTS

A real-time embedded system consists of components, that provide the service capacity of the resources to the event streams, in order to give them the supplies they need for executing their tasks. Whenever an event stream now arrives at a computational or communicational component, an application task is interpreted and executed [1]. This processing usually leads to changes in the event arrival and service curves. So how can we derive the properties of the outgoing streams α' and β' from the properties of the incoming streams α and β?

Let's suppose we have a simple system with only one single event stream arrival curve α and one resource with service capacity β like in Figure 9 [4] [5].

The pair of output arrival curves describes the event rates after this processing step, while the pair of output service curves describes the remaining service capacity [1]. The relations between α, β, α' and β' and thereby the values of $\alpha'$and β' depend on the processing semantics of the component. In general, they can be modeled as

$$\alpha' = f_\alpha(\alpha, \beta)$$

$$\beta' = f_\beta(\alpha, \beta)$$

where the function **f** is given for each component specifically and depends on the processing semantics of the modeled com-

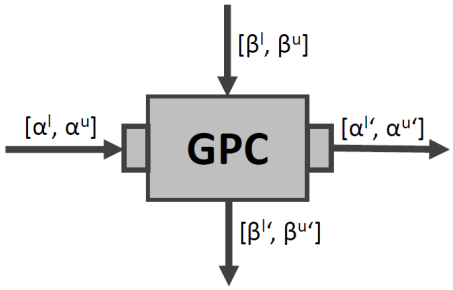

Fig. 9. Single GPC with its incoming and outgoing streams

ponent [1]. These processing semantics need to be provided in the application or resource description.

In the framework of MPA we are mostly working with abstract components. Here the input and output streams are no concrete traces, but always just upper and lower borders for possible traces, like for example $\alpha' = [\alpha'^u, \alpha'^l]$ [1].

Greedy processing components (**GPC**) are a common example for such components in real-time embedded systems. A GPC is triggered by an incoming event, directly instantiates a task and then processes the event, and thereby its task, in a greedy fashion in first-in-first-out order [9]. Of course this process is restricted by the resource availability β. The main advantage of using GPCs is, that we can easily compute the semantics of the component using the min/max-plus algebra (see section A) [1]. Thus, we will from now on assume that all of the components in this paper are GPC.

With a GPC the resulting event streams are [1]:

$$\alpha'^u = \min\left\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\right\}$$

$$\alpha'^l = \min\left\{(\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l\right\}$$

While the remaining service capacities are:

$$\beta'^u = (\beta^u - \alpha^l)\overline{\oslash}0$$

$$\beta'^l = (\beta^l - \alpha^u)\overline{\otimes}0$$

VI. CONSTRUCTING THE SYSTEM PERFORMANCE MODEL

Usually a system needs more than just one component. Therefore, a clever way to systematically link all those different components together is needed, as we want to enable resource sharing, as well as a pipelined stream of data, in the system performance model. This can be done by aligning multiple resources horizontally and multiple event streams vertically [1].

A. Flow of Data

Each event stream is typically processed by a sequence of HW/SW components. The horizontal order of the steams determines the order, in which the event streams will pass by the resources [1]. At every intersection between an event and a resource stream, a component will be instantiated, in case that event stream needs to fulfil a task at this resource.

B. Resource Sharing — Impact of different Scheduling Policies

An important step for designing a real-time embedded system is choosing a scheduling scheme for the resources, as there are usually several event streams that need to use the same component at one time. In the framework of MPA, various scheduling and arbitration policies can be used, like for example fixed priority, proportional share, TDMA, generalized processor sharing or earliest deadline first scheduling [1]. All of these policies have different techniques to distribute the available resources, in particular the remaining service capacities β', among the event streams in need.

Earlier, different approaches were needed when analyzing different classes of arbitration strategies, such as dynamic, static or hierarchical ones, as their event bounds need to be characterized in complete different ways. Yet since [12] there is a unified request bound available for all kinds of analysis purposes in real-time scheduling theory. Therefore, the response time analysis can now be done with the same unified equation, identically what arbitration strategy is used in the system.

Following there are some examples of common scheduling policies and their uses in MPA.

1) Preemptive Fixed Priority Scheduling: In a system with a fixed priority scheduling strategy, the processor always executes the task with the highest priority of the queue first. As those priorities are static, they do not change during execution time. A preemptive scheduler additionally has a clock and will stop the execution of the task after a certain amount of time units have passed, to put the task back into the queue and continue with the next one. Meanwhile, a non-preemptive scheduler does never interrupt the execution of tasks and lets them run until they voluntarily unblock the processor again [3].

In preemptive fixed priority scheduling, tasks are preempted by higher priority tasks [12]. The vertical order of the event streams within the system model determines which event stream will get the highest service capacity and therefore determine the priorities of the event streams [1]. In general urgent or highly frequent tasks should be prioritized. The lower priority event streams below can then only use the remaining service capacity to fulfill their tasks at the same processor [3]. As MPA is especially suitable for modeling fixed priority scheduling, this will be the primary used scheduling technique of the paper.

2) Non-Preemptive Fixed Priority Scheduling: Fixed priority scheduling with a non-preemptive strategy can also be used in the framework of MPA, yet the integration into the system model is more complex than integrating its preemptive equivalent. However, since [9] and [13] efficient and sufficiently general approaches are known to analyze systems with such arbitration behavior. Still it is not possibly to just simply use GPUs here, as the components are not supposed to just process all inputs in a greedy manner one after another. Instead, there is the need for abstracting the scheduling strategy from the system-level components like shown in Figure 10.

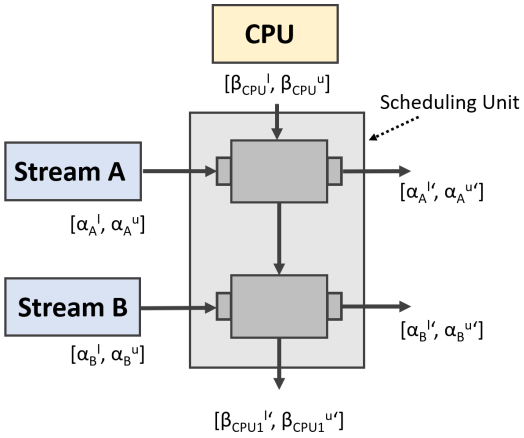

Fig. 10. System Performance Model scheduled using a more complex arbitration technique. This could for example be Fixed Priority Non-Preemptive or Earliest Deadline First Scheduling. The scheduling unit abstracts from the real components and manages what the component will process first.

*3) Earliest Deadline First Scheduling (**EDF**):* Similar to non-preemptive fixed priority scheduling, when using EDF scheduling, there is also the need to abstract the arbitration strategy from the components themselves (see Figure 10). Luckily there is great tool support that allows us to treat EDF components just like any "normal" GPC. However, there now is the need to additionally include the deadline of every task into the calculation. Using these deadlines, the demand bound function can be specified as the deadline shifted request bound function [12].

C. System Performance Model of the Sample System

Starting with the input data collected in section II we can now construct a possible performance model of our system. Here we decide to actually propose two different models in order to compare them later on: model A with two CPUs of 4 MIPS each connected by a bus (Figure 11), as well as model B of only one 5 MIPS CPU (Figure 12).

After mapping the tasks to their corresponding resources using the information from Figure 1 and Figure 2, we define all components to be GPCs. As a scheduling technique we choose preemptive fixed priority scheduling and give the brightness event stream a higher priority than the message event stream, as brightness events can occur more often and need quicker processing, in order for the system to seem responsive.

VII. SYSTEM PERFORMANCE ANALYSIS

Now we can analytically evaluate the given system performance model regarding various performance criteria using MPA. The main assignment will be to find out, whether the system-level timing properties of this specific implementation meet the specified timing requirements [8]. The exact analysis methods may slightly vary for different abstract components, but always remains deterministic. Following, the performance analysis methods for a system out of GPCs are presented.

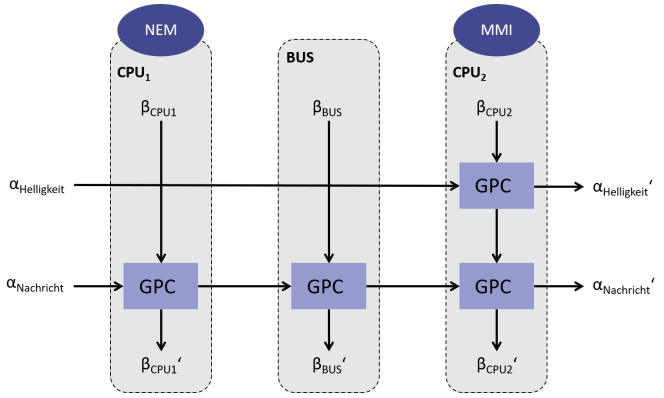

Fig. 11. System Performance Model of Architecture A

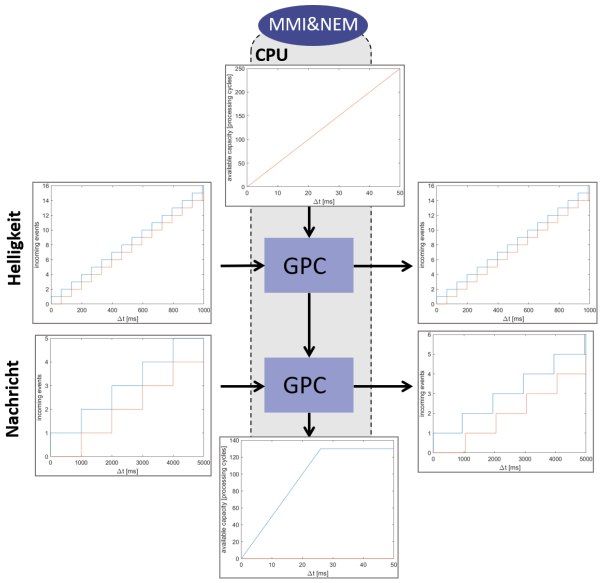

Fig. 12. System Performance Model of Architecture A with the plotted input and output streams

The reason why the following equations use the Infimum and Supremum, instead of just using the minimum or maximum, can be found in [14]. Both, the delay and the buffer space calculations, consider α^u and β^l, as $d_{\max}$ and $b_{\max}$ occur whenever the maximum load of events arrives at the time of minimum resource availability. Visually these sizes can be interpreted as Figure 13.

However, the system performance model is able to provide even more insights to a system. For example analyzing the characteristics of the outgoing service curves, can expose the individual utilization of resources. Besides that, the throughput of the event streams can also be calculated quite easily using the data of the system performance model [1].

A. Maximum Delay Guarantees

The maximum delay $d_{\max}$ experienced by any event on the event stream with arrival curve α when being processed at a

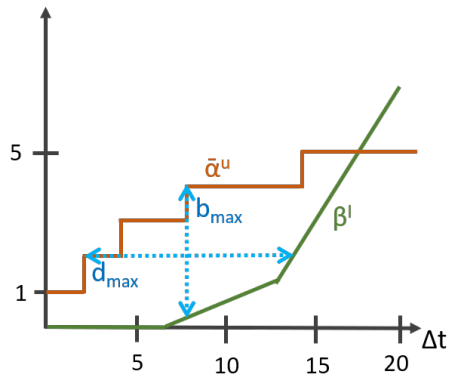

Fig. 13. Graphically determining $d_{\max}$ and $b_{\max}$

single GPC with service curve β is bounded by [9]:

$$d_{\max} \leq Del(\alpha^u, \beta^l)$$

$$Del(\alpha^u, \beta^l) := \sup_{\lambda \geq 0} \left\{ \inf_{\tau \geq 0} \{\alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\} \right\}$$

The end-to-end delay experienced by an event with arrival curve α that is being processed by **n** GPCs with the service curves β_i, $i \in \{0, \ldots, n\}$ is furthermore bounded by:

$$d_{\max} \leq Del(\alpha^u, \beta_1^l \otimes \cdots \otimes \beta_n^l)$$
$$\leq Del(\alpha^u, \beta_1^l) + \cdots + Del(\alpha^u, \beta_n^l)$$

The calculation of the hard upper bounds to the maximum delays are quite conservative, yet they guarantee, that nothing every exceeds those bounds [9].

B. Maximum Buffer Requirements

We often also need to know how much storage space the internal buffers need, in order for them to always be able to temporarily store incoming events, that can not yet be processed. This maximum required buffer space $b_{\max}$ of a GPC with service curve β processing an event stream with arrival curve α is bounded by [9]:

$$b_{\max} \leq Buf\left(\alpha^u, \beta^l\right)$$

$$Buf\left(\alpha^u, \beta^l\right) := \sup_{\lambda \geq 0} \left\{\alpha^u(\lambda) - \beta^l(\lambda)\right\}$$

In the case of several consecutive tasks that all use the same shared memory, the required buffer space can be reduced to

$$b_{\max} \leq Buf(\alpha^u, \beta_1^l \otimes \cdots \otimes \beta_n^l)$$
$$\leq Buf(\alpha^u, \beta_1^l) + \cdots + Buf(\alpha^u, \beta_n^l)$$

C. Performance Analysis of the Sample System

Both, architecture A and architecture B have the same buffer requirements. In the sample system architecture A delays a brightness event by 50ms and a message event by 395ms. Meanwhile, architecture B delays a brightness event by 40ms and a message event by 580ms. As 580ms exceed the specified maximum delay of 500ms per message event, only architecture A is suitable for our scenario.

VIII. SYSTEM SENSITIVITY ANALYSIS

Next up we are going to find out how sensitive or robust the system is to a change in event input sizes. Furthermore, we want to know where possible bottlenecks are and how to choose optimal processor or bus dimensions.

A. How robust is the system?

A robust system is not sensitive towards changes in the event stream rates: An increase of the input data rate of any event stream by a few percents does not cause significant changes in the systems end-to-end delays [1]. Specifically the new end-to-end delays to not exceed the specified maximum delay times of the system applications. In order to increase the system's robustness we need to look for and eliminate bottlenecks.

B. Where is the bottleneck?

The resource, that the system is most sensitive to, when changing its capacity, is a potential bottleneck resource. If we now slightly higher the capacity of a bottleneck resource, without chancing the capacities of any other resources, the end-to-end delay of at least one event stream immediately decreases, and therefore the system quality increases [1].

C. Using the Results of MPA to Make Design Decisions: Improving Architectures

When designing a new embedded real-time system, we want to find out what architecture works best for the specific use case. Therefore, we want to try out numerous possible system models and compare them in therms of their performances and robustness. section VII and section VIII give us a great opportunities to do so, using the system performance model. We can now quickly propose, analyze and compare various systems. These systems may be extremely different, for example when trying out opposed architectures, or just slightly different, for example when decreasing a CPU's performance by just a few percent to see if we can make the system cheaper, without risking any disadvantages. Additionally, we can try out different scheduling strategies or change the communication structure between the resources, to develop new, and thus maybe improved, system models [1].

In the end it is the goal to find a system that, fulfills the timing requirements of all applications and is robust enough in respect to changes, while being as cheap as possible and therefore as small-dimensioned as possible.

D. Sensitivity Analysis of the Sample System

After we chose architecture A to be sufficient to the timing requirements in section VII, we now want to investigate on the architecture's robustness and further improve it. With Figure 14 and Figure 15 we realize that the architecture is actually quite robust to increasing event stream rates, yet it is unable to handle more than 20 brightness events per second or more than 4 message events per second.

Increasing the capacities of CPU1 (Figure 16), the bus (Figure 17) or CPU2 (Figure 18), all immediately lead to improved delays, at least when it comes to message events. As

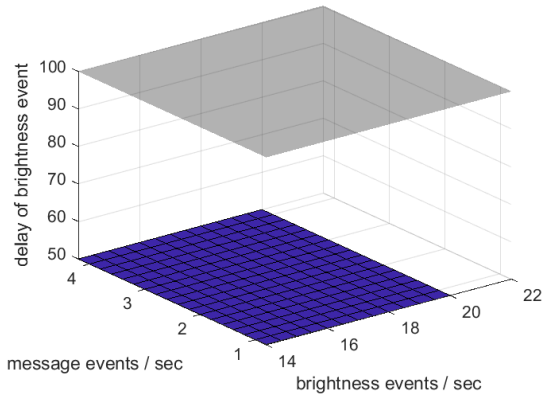

Fig. 14. Impacts associated to increasing the rate of $\alpha_{Brightness}$

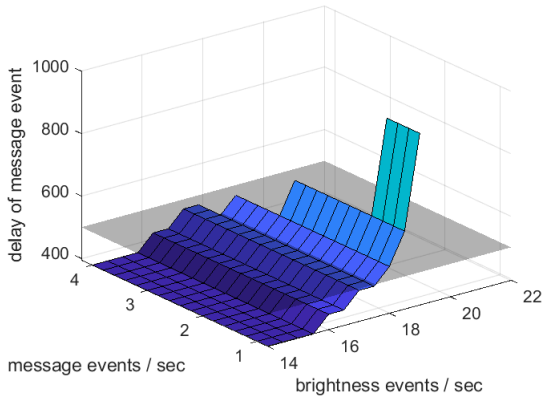

Fig. 15. Impacts associated to increasing the rate of $\alpha_{Message}$

the delays decrease the fastest, when increasing the resource capacity of CPU3, we choose this resource to be worthy of further improvements.

IX. DISCUSSION

A. Advantages and Disadvantages of using a Formal Analysis Method

In contrast to simulation based, probability based or holistic analysis methods for embedded real-time systems, formal methods can guarantee the correctness of their results [1]. The analysis will surely reveal the worst case bounds of system properties such as end-to-end delays. Therefore, it

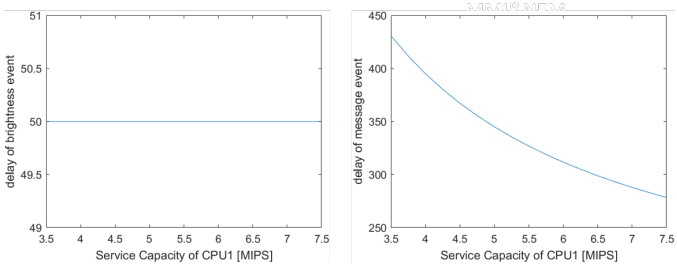

Fig. 16. Impacts associated to increasing the service capacity of CPU1

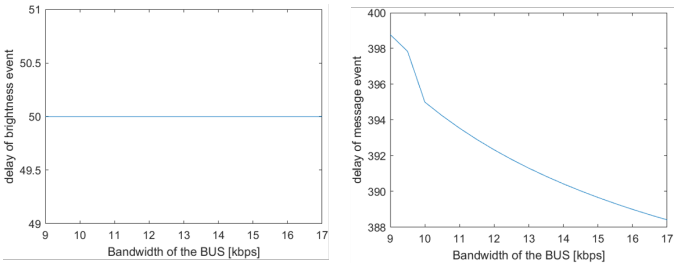

Fig. 17. Impacts associated to increasing the bandwidth of the BUS

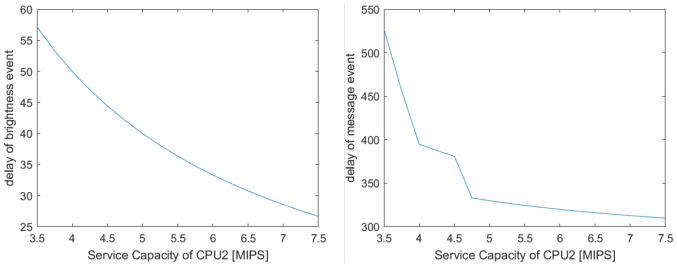

Fig. 18. Impacts associated to increasing the service capacity of CPU2

is guaranteed that any system architecture accepted by the analysis, also fulfills all system requirements in reality [1]. On the other hand, system analysis using formal methods often leads to very pessimistic and in general not tight bounds, due to a lack of details to include in the analysis and limited modeling capacities [8] [9]. Using formal methods, we can only compute hard upper and lower bounds to the real system characteristics, which makes this method only suitable to analyze hard real-time systems only. A system architecture might be rejected due to bad worst case guaranties, even if they are never actually reached in execution and therefore the system in fact fulfills all requirements[1].

Other significant advantages of the formal analysis are the fast analysis run-time, as well as the low set up effort for new architectures [1]. Similar models can be easily and quickly constructed and compared, as this merely involves reconnecting the event and service flows between the components. In this way the technique is sufficiently productive even with a really high time-to-market pressure in the domain of embedded real-time systems and the interactive nature of the design process of such systems [4] [15].

Besides that, formal analysis methods are not suitable for systems with a dynamic and complex interactions, as this architecture can not be incorporated into the model properly [1].

B. Advantages of Using MPA

In contrast to many other formal analysis methods of embedded systems, the modular performance analysis relies neither on standard arrival patterns of events, nor on classical scheduling [16]. In general the MPA can work really efficiently with only really few and abstract data about the system. The low demand on detailed information, makes MPA a lightweight analysis method [1]. Furthermore, this leads to a

high degree of generality and modularity, as well as an easy analyzability of system performance models [1].

The MPA and RTC framework is easy to use, especially when working with the RTC Toolbox for Matlab. It is a compositional method which simplifies the analysis of heterogeneous systems a lot. In addition, MPA can represent the majority of different event arrival patterns, whether arbitrary or not [9].

C. Disadvantages and Limitations of Using MPA

As we are abstraction from the time domain to the time interval domain, it is difficult to accurately exploit implicit timing correlations between event arrivals on different event streams within the MPA framework [1].

In MPA and RTC it is impossible to model state-dependent behavior, as state information cannot be modeled naturally. For example, scheduling policies, that depend on the fill-level of buffers just cannot be implemented in the framework of MPA. Yet, thanks to [16] we can interface RTC curves with state-based models using automata and hence also model states in this new extended framework.

X. CONCLUSION

The framework of modular performance analysis proofs to be a powerful analyzing tool for hard distributed embedded real-time systems, that is capable of doing fast evaluations on system structures. As shown in section IX using MPA with RTC has some major advantages and is useful for various contexts, especially for the use in early design phases of an embedded system to make important design decisions, as MPA can deal with only few and abstract information about the system [1].

The real-time calculus framework furthermore empowered MPA by formal analysis methods and efficient computation strategies for obtaining correct and accurate performance analysis results. Even though the computed bounds on system performance characteristics, such as end-to-end delays and buffer requirements, are often quite pessimistic, they provide guaranteed hard bounds [1].

Although there are some limits to MPA and RTC, they both are really valuable approaches to the formal analysis during the design process of real-time embedded systems. Therefore, they are still subjects of ongoing research. Hence, there is a good chance, that some problems described in section IX will be solved at least partially within the next few years.

If enough information about the system and time to do the analysis are provided, it might also be an option to use MPA combined with a simulation based method for obtaining even better insights into the system [1].

APPENDIX A
THE MIN/MAX-PLUS CALCULUS

Real-time calculus uses the min/max-plus algebra for computational purposes. Instead of the usual algebraic structure of $(\mathbb{R}, +, \cdot)$, we are now dealing with $(\mathbb{R} \cup \infty, +, \cdot)$ as an algebraic structure. Luckily we do not need to deep dive into the characteristics of the algebra here, knowing and understanding the following operations on functions f and g is sufficient for being able to use RTC as a calculation tool:

Min-Plus Convolution:

$$(f \otimes g)(\Delta t) = \inf_{0 \leq \lambda \leq \Delta t} \{f(\Delta t - \lambda) + g(\lambda)\}$$

Min-Plus Deconvolution:

$$(f \oslash g)(\Delta t) = \sup_{\lambda \geq 0} \{f(\Delta t + \lambda) - g(\lambda)\}$$

Max-Plus Convolution:

$$(f \overline{\otimes} g)(\Delta t) = \sup_{0 \leq \lambda \leq \Delta t} \{f(\Delta t - \lambda) + g(\lambda)\}$$

Max-Plus Deconvolution:

$$(f \overline{\oslash} g)(\Delta t) = \inf_{\lambda \geq 0} \{f(\Delta t + \lambda) - g(\lambda)\}$$

Proofs and more detailed information on the min/max-plus algebra can be found in [17]. These equations can be quite complicated to compute, but the RTC Toolbox for Matlab is here to take care of them. This toolkit is a great help when it comes to performing MPA for any kind of use case scenarios.

REFERENCES

[1] E. Wandeler, "Modular performance analysis and interface based design for embedded real time systems," Ph.D. dissertation, Swiss Federal Institute of Technology Zürich, 2006.

[2] L. Thiele, "Performance analysis of distributed embedded systems," in *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*. Association for Computing Machinery, 2007, pp. 7–9.

[3] P. Marwedel, *Embedded System Design*. Springer Dodrecht Heidelberg London New York, 2011, pp. 213–217, 2nd Edition.

[4] M. V. Ernesto Wandeler, Lothar Thiele and P. Lieverse, "System architecture evaluation using modular performance analysis: A case study," in *International Journal on Software Tools for Technology Transfer*, vol. 8, 2006, pp. 649–667.

[5] S. C. Lothar Thiele and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, 2000, pp. 101–104.

[6] *Non Preemptive Static Priority in Network Calculus: Accuracy and Integration with P-GPS*, 2012.

[7] S. K. Samarjit Chakraborty and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 190–195.

[8] E. Wandeler and L. Thiele, "Abstracting functionality for modular performance analysis of hard real-time systems," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '05. Association for Computing Machinery, 2005, p. 697–702.

[9] D. B. Chokshi and P. Bhaduri, "Modeling fixed priority non-preemptive scheduling with real-time calculus," in *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008, pp. 387–392.

[10] M. Moy and K. Altisen, "Arrival curves for real-time calculus: The causality problem and its solutions," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2010, pp. 358–372.

[11] A. M. Ernesto Wandeler and L. Thiele, "On the use of greedy shapers in real-time embedded systems," *ACM Trans. Embed. Comput. Syst.*, pp. 4–8, Apr. 2012.

[12] F. Slomka and M. Sadeghi, "Beyond the limitations of real-time scheduling theory: a unified scheduling theory for the analysis of real-time systems," in *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, 2021, pp. 201–236.

[13] W. M. Sofack and M. Boyer, "Non preemptive static priority with network calculus: Enhancement," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Springer, Berlin, Heidelberg, 2012, pp. 258–272.
[14] R. L. Cruz, "A calculus for network delay. i. network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, 1991.
[15] M. Verhoef and J. Hooman, "Evaluating embedded system architectures," in *Boderc: Model-based design of high-tech systems*, 2007, pp. 151–159.
[16] S. C. Linh T.X. Phan and P. Thiagarajan, "A multi-mode real-time calculus," in *Real-Time Systems Symposium*, 2008, pp. 59–69.
[17] J.-Y. L. Boudec and P. Thiran, *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001, vol. 2050, pp. 125–156.