System level performance analysis – the SymTA/S approach

R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter and R. Ernst

Abstract: SymTA/S is a system-level performance and timing analysis approach based on formal scheduling analysis techniques and symbolic simulation. The tool supports heterogeneous architectures, complex task dependencies and context aware analysis. It determines system-level performance data such as end-to-end latencies, bus and processor utilisation, and worst-case scheduling scenarios. SymTA/S furthermore combines optimisation algorithms with system sensitivity analysis for rapid design space exploration. The paper gives an overview of current research interests in the SymTA/S project.

1 Introduction

With increasing embedded system complexity, there is a trend towards heterogeneous, distributed architectures. Multiprocessor system on chip designs (MpSoCs) use complex on-chip networks to integrate multiple programmable processor cores, specialised memories and other intellectual property (IP) components on a single chip. MpSoCs have become the architecture of choice in industries such as network processing, consumer electronics and automotive systems. Their heterogeneity inevitably increases with IP integration and component specialisation, which designers use to optimise performance at low power consumption and competitive cost. Tomorrow's MpSoCs will be even more complex, and using IP library elements in a 'cut-and-paste' design style is the only way to reach the necessary design productivity.

Systems integration is becoming the major challenge in MpSoC design. Embedded software is increasingly important to reach the required productivity and flexibility. The complex hardware and software component interactions pose a serious threat to all kinds of performance pitfalls, including transient overloads, memory overflow, data loss and missed deadlines. The International Technology Roadmap for Semiconductors, 2003 Edition (http://public.itrs.net/Files/2003ITRS/Design2003.pdf) names system-level performance verification as one of the top three codesign issues.

Simulation is state of the art in MpSoC performance verification. Tools from many suppliers support cycle-accurate cosimulation of a complete hardware and software system. The cosimulation times are extensive, but developers can use the same simulation environment, simulation patterns and benchmarks in both function and performance verification. Simulation-based performance verification,

however, has conceptual disadvantages that become disabling as complexity increases.

MpSoC hardware and software component integration involves resource sharing that is based on operating systems and network protocols. Resource sharing results in a confusing variety of performance runtime dependencies. For example, Fig. 1 shows a CPU subsystem executing three processes. Although the operating system activates T_1, T_2 and T_3 strictly periodically (with periods P_1, P_2 and P_3, respectively), the resulting execution sequence is complex and leads to output bursts.

As Fig. 1 shows, T_1 can delay several executions of T_3. After T_1 completes, T_3 – with its input buffers filled – temporarily runs in burst mode with the execution frequency limited only by the available processor performance. This leads to transient T_3 output burst, which is modulated by T_1's execution.

Figure 1 does not even include data-dependent process execution times, which are typical for software systems, and operating system overhead is neglected. Both effects further complicate the problem. Yet finding simulation patterns - or use cases - that lead to worst-case situations as highlighted in Fig. 1 is already challenging.

Network arbitration introduces additional performance dependencies. Figure 2 shows an example. The arrows indicate performance dependencies between the CPU and DSP subsystems that the system function does not reflect. These dependencies can turn component or subsystem best-case performance into system worst-case performance – a so-called scheduling anomaly. Recall the T_3 bursts from Fig. 1 and consider that T_3's execution time can vary from one execution to the next. There are two critical execution scenarios, called corner cases: the minimum execution time for T_3 corresponds to the maximum transient bus load, slowing down other components' communication; and vice versa.

The transient runtime effects shown in Figs. 1 and 2 lead to complex system-level corner cases. The designer must provide a simulation pattern that reaches each corner case during simulation. Essentially, if all corner cases satisfy the given performance constraints, then the system is guaranteed to satisfy its constraints under all possible operation conditions. However, such corner cases are extremely difficult to find and debug, and it is even more difficult to find simulation patterns to cover them all. Reusing function verification patterns is not sufficient because they do not

148

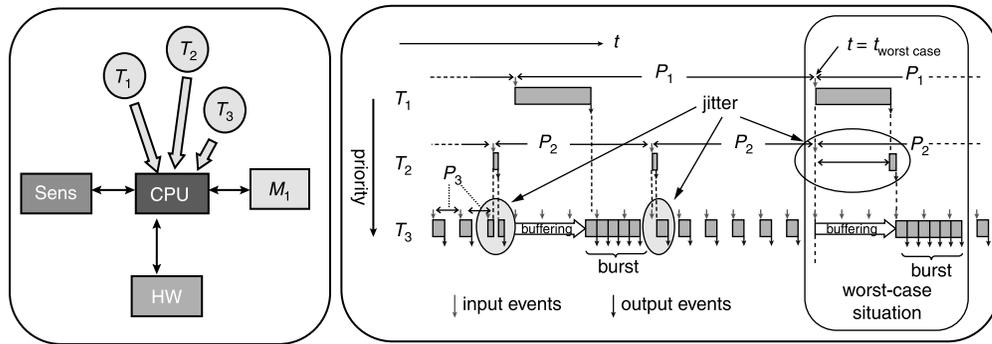*IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005*
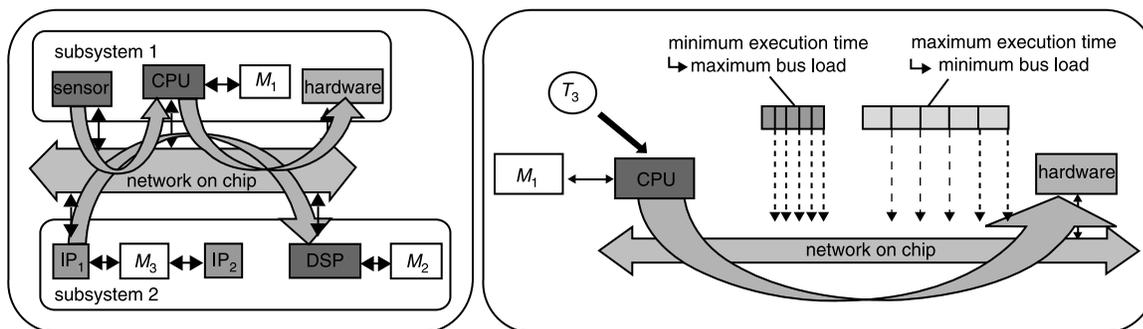
Fig. 1 *CPU subsystem*

Fig. 2 *Scheduling anomaly*

cover the complex nonfunctional performance dependencies that resource sharing introduces. Reusing component and subsystem verification patterns is not sufficient because they do not consider the complex component and subsystem interactions.

The system integrator might be able to develop additional simulation patterns, but only for simple systems in which the component behaviour is well understood. Manual corner case identification and pattern selection is not practical for complex MpSoCs with layered software architectures, dynamic bus protocols and operating systems. In short, simulation-based approaches to MpSoC performance verification are about to run out of steam, and should essentially be enhanced by formal techniques that systematically reveal and cover corner cases.

Real-time systems research has addressed scheduling analysis for processors and buses for decades, and many popular scheduling analysis techniques are available. Examples include rate-monotonic scheduling and earliest deadline first [1], using both static and dynamic priorities, and time-slicing mechanisms like TDMA or round-robin [2]. Some extensions have already found their way into commercial analysis tools, which are being established, e.g. in the automotive industry to analyse individual units that control the engine or parts of the electronic stability program.

The techniques rely on a simple yet powerful abstraction of task activation and communication. Instead of considering each event individually, as simulation does, formal scheduling analysis abstracts from individual events to event streams. The analysis requires only a few simple characteristics of event streams, such as an event period or a maximum jitter. From these parameters, the analysis systematically derives worst-case scheduling scenarios, and timing equations safely bound the worst-case process or communication response times.

It might be surprising that, up to now, only very few of these approaches have found their way into the SoC (system-on-chip) design community by means of tools. Regardless of the known limitations of simulation such as incomplete corner-case coverage and pattern generation, timed simulation is still the preferred means of performance verification in MpSoC design. Why then is the acceptance of formal analysis still very limited?

One of the key reasons is a mismatch between the scheduling models assumed in most formal analysis approaches and the heterogenous world of MpSoC scheduling techniques and communication patterns that are a result of (*a*) different application characteristics; (*b*) system optimisation and integration which is still at the beginning of the MpSoC development towards even more complex architectures.

Therefore, a new configurable analysis process is needed that can easily be adapted to such heterogeneous architectures. We can identify different approaches: the holistic approach that searches for techniques spanning several scheduling domains; and hierarchical approaches that integrate local analysis with a global flow based analysis, either using new models or based on existing models and analysis techniques.

2 Formal techniques in system performance analysis

Formal approaches to heterogeneous systems are rare. The 'holistic' approach [3, 4] systematically extends the classical scheduling theory to distributed systems. However, because of the very large number of dependencies, the complexity of the equations underlying the analysis grows with system size and heterogeneity. In practice, the holistic approach is limited to those system configurations which simplify the equations, such as deterministic TDMA

networks. However, there is, up to now, no general procedure to set-up and solve the holistic equations for arbitrary systems. This could explain why such holistic approaches are largely ignored by the SoC community even though there are many proposals for multiprocessor analysis in real-time computing.

Gresser [5] and Thiele *et al.* [6] established a different view on scheduling analysis. The individual components or subsystems are seen as entities which interact, or communicate, via event streams. Mathematically speaking, the stream representations are used to capture the dependencies between the equations (or equation sets) that describe the individual components timing. The difference to the holistic approach (which also captures the timing using system-level equations) is that the compositional models are well structured with respect to the architecture. This is considered a key benefit, since the structuring significantly helps designers to understand the complex dependencies in the system, and it enables a surprisingly simple solution. In the 'compositional' approach, an output event stream of one component turns into an input event stream of a connected component. Schedulability analysis, then, can be seen as a flow-analysis problem for event streams that, in principle, can be solved iteratively using event stream propagation.

Both approaches use a highly generalised event stream representation to tame the complexity of the event streams. Gresser uses a superpositional event vector system, which is then propagated using complex event dependency matrices. Thiele *et al.* use a more intuitive model. They use numerical upper and lower bound event arrival curves for event streams and similar service curves for execution modeling.

This generality, however, has its price. Because they introduced new stream models, both Thiele and Gresser had to develop new scheduling analysis algorithms for the local components that utilise these models; the host of existing work in real-time system cannot be re-used. Furthermore, the new models are far less intuitive than the ones known from the classical real-time systems research, e.g. the model of rate-monotonic scheduling with its periodic tasks and worst-case execution times. A system-level analysis should be simple and comprehensible, otherwise its acceptance is extremely doubtful.

The compositional idea is a good starting point for the following considerations. It uses some event stream representation to allow component-wise local analysis. The local analysis results are, then, propagated through the system to reach a global analysis result. We do not necessarily need to develop new local analysis techniques if we can benefit from the host of work in real-time scheduling analysis.

A key novelty of our unique SymTA/S approach is that we use intuitive standard event models (Section 3.2) from real-time systems research rather than introducing new, complex stream representations. Periodic events or event streams with jitter and bursts [7] are examples of standard models that can be found in the literature. Our SymTA/S technology lets us extract this information from a given schedule and automatically interface or adapt the event stream to the specific needs within these standard models, so that designers and analysts can safely apply existing subsystem techniques of choice without compromising global analysis.

3 SymTA/S approach

SymTA/S [8] is a formal system-level performance and timing analysis tool for heterogeneous SoCs and distributed systems. The application model of SymTA/S is described in Section 3.1. The core of SymTA/S is our recently developed technique to couple local scheduling analysis algorithms using event streams [9, 10]. Event streams describe the possible I/O timing of tasks. In our compositional performance analysis methodology [11, 12], input and output event streams are described by standard event models which are introduced in detail in Section 3.2. The analysis composition using event streams is described in Section 3.3. A second key property of our compositional approach is the ability to adapt the possible timing of events in an event stream. The event stream adaptation concept is described in Section 3.4.

3.1 SymTA/S application model

A task is activated due to an activating event. Activating events can be generated in a multitude of ways, including expiration of a timer, external or internal interrupt, and task chaining. Our existing approach assumes that each task has one input FIFO. A task reads its activating data from its input FIFO and writes data into the input FIFO of a dependent task. A task may read its input data at any time during one execution. We therefore assume that the data need to be available at the input during the whole execution of the task. We also assume that input data is removed from the input FIFO at the end of one execution.

A task needs to be mapped on a computation or communication resource to execute. When multiple tasks share the same resource, then two or more tasks may request the resource at the same time. To arbitrate request conflicts, a resource is associated with a scheduler which selects a task to which it grants the resource out of the set of active tasks according to some scheduling policy. Other active tasks have to wait. Scheduling analysis calculates worst-case (sometimes also best-case) task response times, i.e. the time between task activation and task completion, for all tasks sharing a resource under the control of a scheduler. Scheduling analysis guarantees that all observable response times will fall into the calculated [best-case, worst-case] interval. We therefore say that scheduling analysis is conservative. We assume that a task writes its output data at the end of one execution. This assumption is standard in scheduling analysis.

Figure 3 shows an example of a system modelled with SymTA/S. The system consists of two resources each with two tasks mapped on it. R1 and R2 are both assumed to be priority scheduled. Src1 and Src2 are the sources of the external activating events at the system inputs. The possible timing of activating events is captured by so-called *event models*, which are introduced in Section 3.2.

3.2 SymTA/S standard event models

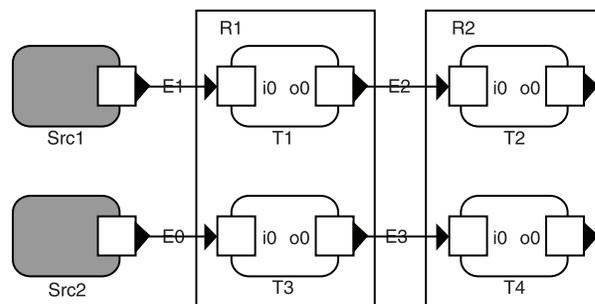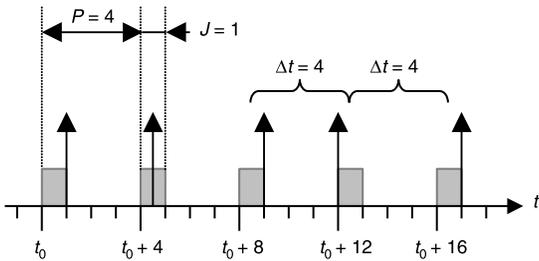Event models can be described by sets of parameters. For example, a *periodic with litter* event model has two

Fig. 3 *System modelled with SymTA/S*

Fig. 4 *Example of an event stream that satisfies event model* $\mathcal{P} = 4, \ \mathcal{J} = 1$

parameters $(\mathcal{P}, \mathcal{J})$ and states that each event generally occurs periodically with period $\mathcal{P}$, but that it can jitter around its exact position within a jitter interval $\mathcal{J}$. Consider an example where $(\mathcal{P}, \mathcal{J}) = (4, 1)$. This event model is visualised in Fig. 4. Each grey box indicates a jitter interval of length $\mathcal{J} = 1$. The jitter intervals repeat with the event model period $\mathcal{P} = 4$. The Figure additionally shows a sequence of events which satisfies the event model, since exactly one event falls within each jitter interval box, and no events occur outside the boxes.

An event model can also be expressed using two event funηctions $\eta^u(\Delta t)$ and $\eta^l(\Delta t)$.
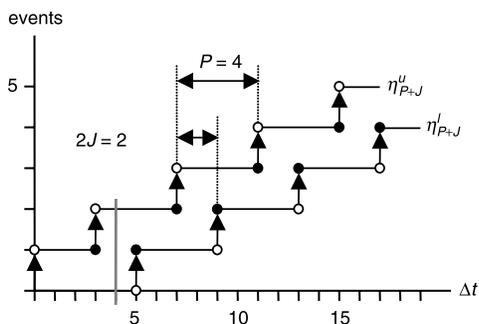
Definition 1 (Upper event function): The upper event function $\eta^u(\Delta t)$ specifies the maximum number of events that can occur during any time interval of length Δt.

Definition 2 (Lower event function): The lower event function $\eta^l(\Delta t)$ specifies the minimum number of events that have to occur during any time interval of length Δt.

Event functions are piecewise constant step functions with unit-height steps, each step corresponding to the occurrence of one event. Figure 5 shows the event functions for the event model $(\mathcal{P} = 4, \mathcal{J} = 1)$. Note that at the points where the functions step, the smaller value is valid for the upper event function, while the larger value is valid for the lower event function (indicated by dark dots). For any time interval of length Δt, the actual number of events is bound by the upper and lower event functions. Event functions resemble arrival curves [13] which have been successfully used by Thiele *et al.* for compositional performance analysis of network processors [14]. In the following, the dependency of η^u and η^l on Δt is omitted for brevity.

A *periodic with jitter* event model is described by the following event functions $\eta^u_{\mathcal{P}+\mathcal{J}}$ and $\eta^l_{\mathcal{P}+\mathcal{J}}$ [12]:

$$\eta^u_{\mathcal{P}+\mathcal{J}} = \left\lceil \frac{\Delta t + \mathcal{J}}{\mathcal{P}} \right\rceil \qquad (1)$$

Fig. 5 *Upper and lower event functions for event model* $\mathcal{P} = 4, \ \mathcal{J} = 1$

$$\eta^l_{\mathcal{P}+\mathcal{J}} = \max\left(0, \left\lceil \frac{\Delta t - \mathcal{J}}{\mathcal{P}} \right\rceil \right) \qquad (2)$$

To get a better feeling for event functions, imagine a sliding window of length Δt that is moved over the (infinite) length of an event stream. Consider $\Delta t = 4$ (grey vertical bar in Fig. 5). The upper event function indicates that at most two events can be observed during any time interval of length $\Delta t = 4$. This corresponds, for example, to a window position between $t_0 + 8.5$ and $t_0 + 12.5$ in Fig. 4. The lower event function indicates that no events have to be observed during $\Delta t = 4$. This corresponds, for example, to a window position between $t_0 + 12.5$ and $t_0 + 16.5$ in Fig. 4.

Let us further introduce distance functions $\delta^{min}(N \geq 2)$ and $\delta^{max}(N \geq 2)$, which return the minimum (respectively, maximum) distance between $N \geq 2$ consecutive events in an event stream.

Definition 3 (Minimum distance function): The minimum distance function $\delta^{min}(N \geq 2)$ specifies the minimum distance between $N \geq 2$ consecutive events in an event stream.

Definition 4 (Maximum distance function): The maximum distance function $\delta^{max}(N \geq 2)$ specifies the maximum distance between $N \geq 2$ consecutive events in an event stream.

For *periodic with jitter* event models we obtain:

$$\delta^{min}(N \geq 2) = \max\{0, (N-1) * \mathcal{P} - \mathcal{J}\} \qquad (3)$$

$$\delta^{max}(N \geq 2) = (N-1) * \mathcal{P} + \mathcal{J} \qquad (4)$$

For example, the minimum distance between two events in a *periodic with jitter* event model with $(\mathcal{P} = 4, \mathcal{J} = 1)$ is 3 time units, and the maximum distance between two events is 5 time units.

If in a periodic with jitter event model, the jitter is larger than the period, then two or more events can occur at the same time, leading to bursts. To describe a bursty event model, the *periodic with jitter* event model can be extended with a d_{min} parameter that captures the minimum distance between events within a burst. A more detailed discussion can be found in [12].

Additionally, sporadic events are also common [11]. We model sporadic event streams with the same set of parameters as periodic event streams. The difference is that for sporadic event streams, the lower event function $\eta^l(\Delta t)$ is always zero. The maximum distance function $\delta^{max}(N \geq 2)$ approaches infinity for all values of N [12]. Note that jitter and d_{min} parameters are also meaningful in sporadic event models, since they allows to accurately capture sporadic transient load peaks.

Event models with this small set of parameters have several advantages. Firstly, they are easily understood by a designer, since period, jitter etc. are familiar event stream properties. Secondly, the corresponding event functions and distance functions can be evaluated quickly, which is important for scheduling analysis to run fast. Thirdly, as we will see in Section 3.3.2, compositional performance analysis requires the modelling of possible timing of output events for propagation to the next scheduling component. Our event models allow us to specify simple rules to obtain output event models (Section 3.3.1) that can be described with the same set of parameters as the activating event models. Therefore, we do not have to depart from our event models independent of size and structure of the composed

system (hence the term 'standard'). This makes our compositional performance analysis approach very general.

3.3 Analysis composition

In our compositional performance analysis methodology [11, 12], we alternate local scheduling analysis and even model propagation during system-level analysis. This requires the modelling of possible timing of output events for propagation to the next scheduling component. In the following, first we explain the output event model calculation. Then we present our compositional analysis approach.

3.3.1 Output event model calculation:
Our event models allow us to specify simple rules to obtain output event models that can be described with the same set of parameters as the activating event models. The output event model period obviously equals the activation period. The difference between maximum and minimum response times (the response time jitter) is added to the activating event model jitter, yielding the output event model jitter (5):

$$\mathcal{J}_{out} = \mathcal{J}_{act} + (t_{resp,max} - t_{resp,min}) \tag{5}$$

Note that if the calculated output event model has a larger jitter than period, this information alone would indicate that an early output event could occur before a late previous output event, which obviously cannot be correct. In reality, output events cannot follow closer than the minimum response time of the producer task. This is indicated by the value of the *minimum distance* parameter.

3.3.2 Analysis composition using standard event models:
In the following, we explain our compositional analysis approach using the system example in Fig. 3. Initially, only event models at the external system inputs are known. Since an activating event model is available for each task on $R1$, a local scheduling analysis of this resource can be performed and output event models are calculated for $T1$ and $T3$ (Section 3.3.1). In the second phase, all output event models are propagated. The output event models become the activating event models for $T2$ and $T4$. Now, a local scheduling analysis of $R2$ can be performed since all activating event models are known.

However, it is sometimes impossible to perform system level scheduling analysis as explained above. This is shown in the system example in Fig. 6.

Figure 6 shows a system consisting of two resources, R_1 and R_2, each with two tasks mapped on it. Initially, only the activating event models of T1 and T3 are known. At this point the system cannot be analysed, because on every resource an activating event model for one task is missing, i.e. we need to calculate response times on R_1 to be able to analyse R_2. On the other hand, we cannot analyse R_1 before analysing R_2. We call this problem 'cyclic scheduling dependency'.

One solution to this problem is to initially propagate all external event models along all system paths until an initial
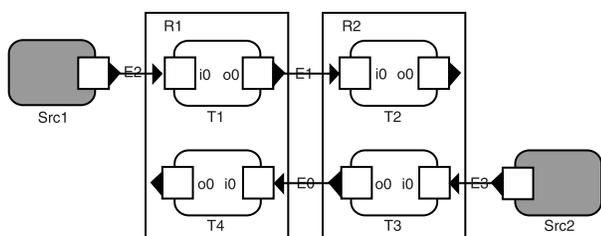
activating event model is available for each task [15]. This approach is safe since, on one hand scheduling cannot change an event model period, and on the other hand, scheduling can only *increase* an event model jitter [7]. Since a smaller jitter interval is contained in a larger jitter interval, the minimum initial jitter assumption is safe.

After propagating external event models, global system analysis can be performed. A global analysis step consists of two phases [12]. In the first phase local scheduling analysis is performed for each resource and output event models are calculated (Section 3.3.1). In the second phase, all output event models are propagated. It is then checked if the first phase has to be repeated because some activating event models are no longer up-to-date, meaning that a newly propagated output event model is different from the output event models that was propagated in the previous global analysis step. Analysis completes if either all event models are up-to-date after the propagation phase, or if an abort condition, e.g. the violation of a timing constraint has been reached.

3.4 Event stream adaptation

A key property of our compositional performance analysis approach is the ability to adapt the possible timing of events in an event stream (expressed through the adaptation of an event model [12]) There are several reasons to do this. It may be that a scheduler or a scheduling analysis for a particular component requires certain event stream properties. For example, rate-monotonic scheduling and analysis [1] require strictly periodic task activation. Alternatively, an integrated IP component may require certain event stream properties. External system outputs may also impose event model constraints, e.g. a minimum distance between output events or a maximum acceptable jitter. Such a constraint may be the result of a performance contract with an external subsystem [16]. Event stream adaptation can also be done for the sole purpose of traffic shaping [12]. Traffic shaping can be used to reduce transient load peaks, for example, to obtain more regular system behaviour. Practically, we distinguish event model adaptation from event model *shaping* in SymTA/S [17]. Adaptation is required to satisfy an event model constraint, while shaping is voluntary to obtain more regular system behaviour. We have currently implemented two types of event adaptation functions (EAF): a periodic EAF produces a periodic event stream from a *periodic with jitter* input event stream. A d_{min}-EAF enforces a minimum distance between output events.

4 Complex embedded applications

Compositional performance analysis as described so far is not applicable to embedded applications with complex task dependencies. This is because it uses a simple activation model where the completion of one task directly leads to the activation of a dependent task. However, activation dependencies in realistic embedded applications are usually more complex. A consumer task may require a different amount of data per execution than produced by a producer task, leading to multi-rate systems. Task activation may also be conditional, leading to execution-rate intervals. Furthermore, a task may consume data from multiple task inputs. Tasks with multiple inputs also allow cyclic dependencies to be formed (e.g. in a control loop).

In this Section, we focus on multiple inputs (both AND- and OR-activation) and functional cycles [18]. We skip multi-rate systems and conditional communication, since these features have not yet been incorporated into

Fig. 6 *Example of a system with cyclic scheduling dependency*

SymTA/S. The reader interested in their theoretical foundations is referred to [19].

4.1 Basic thoughts

The activation function of a consumer task C with multiple inputs is a Boolean function of input events at the different task inputs. A restriction we impose is that activation must not be invalidated due to the arrival of additional tokens [20]. This means that negation is not allowed in the activation function. Consequently, the only acceptable Boolean operators are AND and OR, since an input is negated in all other commonly used Boolean operators (NOT, XOR, NAND, NOR).

To perform scheduling analysis on the resource to which task C is mapped, activating event functions for task C have to be calculated from all input event functions. In the following we demonstrate how to do this for AND- and OR-activation using our standard event models (Section 3.2). An extended discussion covering event models in general can be found in [19].

4.2 AND-activation

For a consumer task C with multiple inputs, AND-activation implies that C is activated after an input event has occurred at each input i. An example of an AND-activated task with three inputs is shown in Fig. 7.

Note that AND-activation requires input data buffering, since at some inputs data may have to wait until data have arrived at all other inputs for one consumer activation. We will refer to this source of buffering as AND-buffering. We also use the term 'token' [21] to refer to the amount of data required for one input event.

4.2.1 AND-activation period:
To ensure bounded AND-buffer sizes the period of all input event models must be the same. The period of the activating event model equals this period:

$$\mathcal{P}_i \overset{!}{=} \mathcal{P}_j \; ; \; i, j = 1 \ldots k \Rightarrow$$
$$\mathcal{P}_{AND} = \mathcal{P}_i \; ; \; i = 1 \ldots k \qquad (6)$$

4.2.2 AND-activation jitter:
To obtain the AND-activation jitter, let us consider how often activation of the AND-activated task can occur during any time interval Δt. Obviously, during any time interval Δt, the port with the smallest minimum number of available tokens determines the minimum number of AND-activations. Likewise, the port with the smallest maximum number of available tokens determines the maximum number of AND-activations.

The number of available tokens at port i during a time interval Δt depends on both the number of tokens arriving during Δt and on the number of tokens that arrived earlier, but did not yet lead to an activation because tokens at one or more other ports are still missing. This is illustrated in the
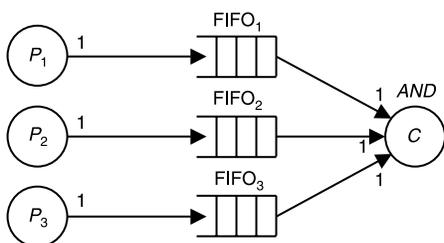

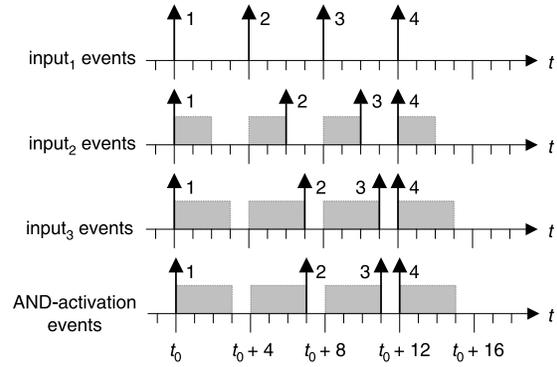**Fig. 7** Example of an AND-activated task C

Fig. 8 AND-activation timing example

following example. Let us assume that our task in Fig. 7 receives tokens at each with the following period with jitter input event models:

$$\mathcal{P}_1 = 4, \; \mathcal{J}_1 = 0$$
$$\mathcal{P}_2 = 4, \; \mathcal{J}_2 = 2$$
$$\mathcal{P}_3 = 4, \; \mathcal{J}_3 = 3$$

Figure 8 shows a possible sequence of input events that adhere to these event models, and the resulting AND-activation events. The numbering of events in the Figure indicates which events together lead to one activation of AND-activated task C.

As can be seen, the minimum distance between two AND-activations (activations 3 and 4 in Fig. 8) equals the minimum distance between two input events at input 3, which is the input with the largest input event model jitter. Likewise, the maximum distance between two AND-activations (activations 1 and 2 in Fig. 8) equals the maximum distance between two input events at input 3. It is not possible to find a different sequence of input events leading to a smaller minimum or a larger maximum distance between two AND-activations. From this we can conclude that the input with the largest input event jitter determines the activation jitter of the AND-activated task, i.e.

$$\mathcal{J}_{AND} = \max\{\mathcal{J}_i\} \; ; \; i = 1 \ldots k \qquad (7)$$

This statement also remains true if the first set of input events do not arrive at the same time (as is the case in Fig. 8). A proof is given in [19]. Calculation of the worst-case delay and backlog at each input due to AND-buffering can also be found in [19].

Note that in some cases it may be possible to calculate phases between the arrival of corresponding tokens in more detail, e. g. through the use of inter-event-stream contexts (Section 5.3). It may then be possible to calculate a tighter activating jitter if it can be shown that a certain input cannot (fully) influence the activation timing of an AND-activated task, because tokens at this input arrive relatively early. This is particularly important for the analysis of functional cycles (Section 4.4).

4.3 OR-activation

For a consumer task C with multiple inputs, OR-activation implies that C is activated each time an input event occurs at any input of C. Different to AND-activation, input event models are not restricted, and no OR-buffering is required, since a token at one input never has to wait for tokens to arrive at a different input in order to activate C. Of course, activation buffering is still required.

IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

153
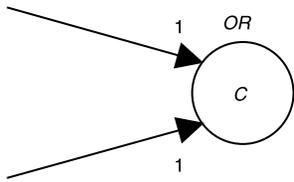
Fig. 9 *Example of an OR-activated task C*

An example of an OR-activated task with two inputs is shown in Fig. 9. Let us assume the following *periodic with jitter*, event models at the two inputs of task C.

$$\mathcal{P}_1 = 4, \ \mathcal{J}_1 = 2$$
$$\mathcal{P}_2 = 3, \ \mathcal{J}_2 = 2$$

The corresponding upper and lower input event functions are shown in Fig. 10. Since each input event immediately leads to one activation of task C, the upper and lower activating event functions are constructed by adding the respective input event functions. The result is shown in Fig. 11a.

Recall a key requirement of compositional performance analysis, namely that event streams are described in a form that can serve both as input for local scheduling analysis, and can be produced as an output of local scheduling analysis for propagation to the next analysis component (Section 3.3.2). Owing to the irregularly spaced steps (visible in Fig. 11a), the exact activating event functions cannot be described by a *periodic with jitter* event model, and thus cannot serve directly as input for local scheduling analysis. Furthermore, after local scheduling analysis a
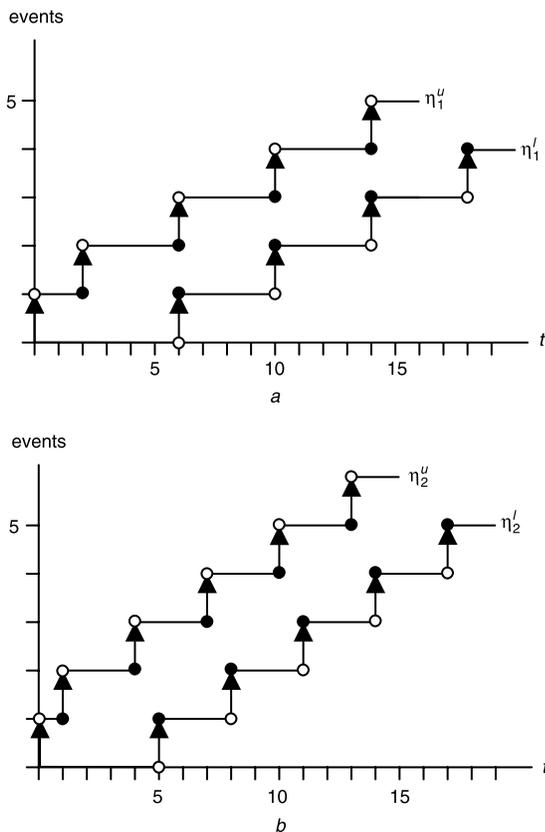
Fig. 10 *Upper and lower input event functions in our OR-example*

a OR input 1 $(\mathcal{P}_1 = 4, \mathcal{J}_1 = 2)$
b OR output 2 $(\mathcal{P}_2 = 3, \mathcal{J}_2 = 2)$
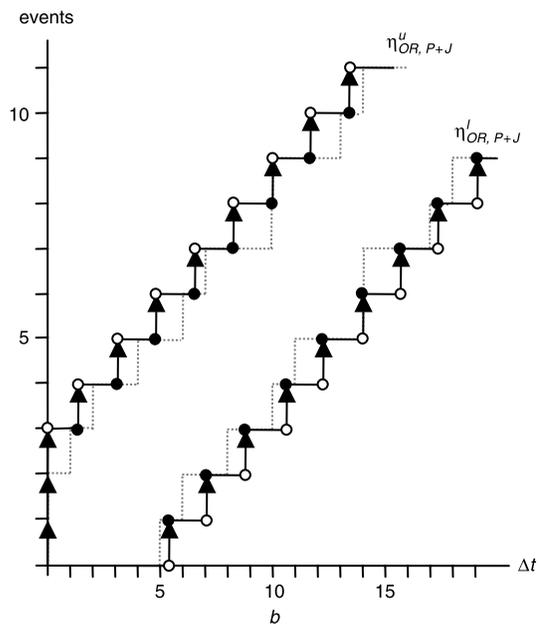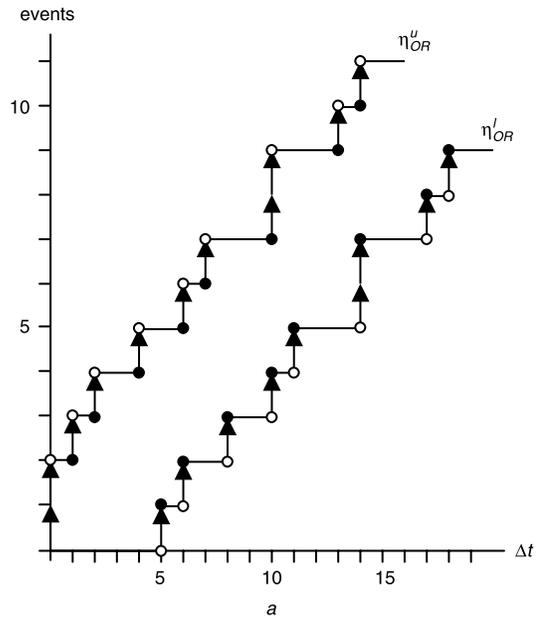
Fig. 11 *Upper and lower activating event functions in our OR-example*

a Exact
b Periodic with jitter approximation

periodic with jitter output event model has to be propagated to the next analysis component. We need an activation jitter in order to calculate an output jitter (Section 3.3.1). Therefore, we need to find conservative approximations for the exact activating event functions that can be described by a *periodic with jitter* event model $(\mathcal{P}_{OR}, \mathcal{J}_{OR})$. The intended result is shown in Fig. 11b (the exact curves appear as dotted lines).

4.3.1 OR-activation period:
The period of OR-activation is the least common multiple $\mathrm{LCM}(P_i)$ of all input event model periods (the *macro period*), divided by the sum of input events during the macro period assuming zero jitter for all input event streams:

$$\mathcal{P}_{OR} = \frac{\mathrm{LCM}(P_i)}{\sum_{i=1}^{n} \frac{\mathrm{LCM}(P_i)}{\mathcal{P}_i}} = \frac{1}{\sum_{i=1}^{n} \frac{1}{\mathcal{P}_i}} \qquad (8)$$

4.3.2 OR-activation jitter:
A conservative approximation for the exact activating event functions with a *periodic with jitter* event model implies the following inequalities:

$$\left\lceil \frac{\Delta t + \mathcal{J}_{OR}}{\mathcal{P}_{OR}} \right\rceil \geq \sum_{i=1}^{n} \left\lceil \frac{\Delta t + \mathcal{J}_i}{\mathcal{P}_i} \right\rceil \quad (9)$$

$$\max\left(0, \left\lfloor \frac{\Delta t - \mathcal{J}_{OR}}{\mathcal{P}_{OR}} \right\rfloor\right) \leq \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{\Delta t - \mathcal{J}_i}{\mathcal{P}_i} \right\rfloor\right) \quad (10)$$

To be as accurate as possible, we are interested in the minimum jitter that satisfies inequalities (9) and (10). It can be shown that the minimum jitter that satisfies (9) and the minimum jitter that satisfies (10) are the same [19]. In the following, the upper approximation (9) is used to calculate the OR-activation jitter. Since the left and right sides of this inequality are only piecewise continuous, it cannot be simply transformed to obtain the desired minimum jitter. The solution used here is to evaluate (9) piecewise for each interval $[\Delta t_j, \Delta t_{j+1}]$, during which the right side has a constant value $k_j \in \mathbb{N}$. For each constant piece of the right side, a condition for a *local jitter* $\mathcal{J}_{OR,j}$ is obtained that satisfies inequality for all $\Delta t : \Delta t_j < \Delta t \leq \Delta t_{j+1}$.

For each constant piece of the right side, (9) becomes

$$\left\lceil \frac{\Delta t + \mathcal{J}_{OR,j}}{\mathcal{P}_{OR}} \right\rceil \geq k_j \; ; \; \Delta t_j < \Delta t \leq \Delta t_{j+1}, k_j \in \mathbb{N}$$

Since the left side is monotonically increasing with Δt, it is sufficient to evaluate it for the smallest value of Δt, which approaches Δt_j, i.e.

$$\lim_{\epsilon \to +0} \left\lceil \frac{\Delta t_j + \epsilon + \mathcal{J}_{OR,j}}{\mathcal{P}_{OR}} \right\rceil \geq k_j \; ; \; k_j \in \mathbb{N}$$

$$\Leftrightarrow \lim_{\epsilon \to +0} \left\lceil \frac{\Delta t_j + \epsilon + \mathcal{J}_{OR,j}}{\mathcal{P}_{OR}} \right\rceil > k_j - 1$$

$$\Leftrightarrow \lim_{\epsilon \to +0} (\mathcal{J}_{OR,j} + \epsilon) > (k_j - 1) * \mathcal{P}_{OR} - \Delta t_j$$

$$\Leftrightarrow \mathcal{J}_{OR,j} \geq (k_j - 1) * \mathcal{P}_{OR} - \Delta t_j \quad (11)$$

The global minimum jitter is then the smallest value which satisfies all local jitter conditions. As already said, η_{OR}^u displays a pattern of distances between steps which repeats periodically every macro period. Therefore, it is sufficient to perform above calculation for one macro period. An algorithm can be found in [22].

4.4 Cyclic task dependencies
Tasks with multiple inputs allow us to build cyclic dependencies. A typical application is a control loop, where one task represents the controller and the other task a model of the controlled system. A task graph with a cycle is shown in Fig. 12.
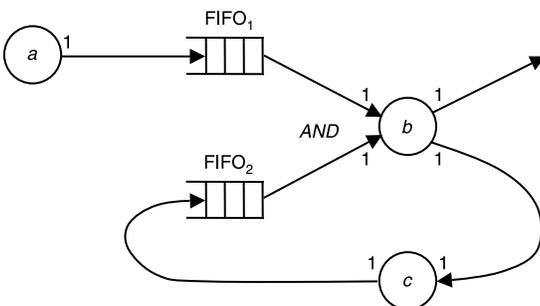
Fig. 12 *Example of a cyclic dependency*

We assume that tasks with multiple inputs in cycles are AND-activated, and that they produce one token at each output per execution. This implies that at least one initial token must be present inside the cycle to avoid deadlock [21], and that the number of tokens inside the cycles remains constant. Consequently, the period of the cycle-external event model determines the period of all cycle tasks. Finally, we assume exactly one cycle-task with one cycle-external and one cycle-internal input. All other cycle-tasks only have cycle-internal inputs. These restrictions allow us to concisely discuss the main issues resulting from functional cycles. A much more general discussion can be found in [19].

In Section 4.2 we established that the activation jitter of an AND-activated task is bounded by the largest input jitter. As was the case for cyclic scheduling dependencies (Section 3.3.2), we have to start system analysis with an initial assumption about the cycle-internal jitter of the AND-activated task, since this value depends on the output jitter of that task, which we have not calculated yet. A conservative starting point is to initially assume zero internal jitter. We can now iterate analysis and event model propagation around the cycle, hoping to find a fix-point.

However, if only one task along the cycle has a response time which is an interval, then after the first round of analysis and event model propagation the internal input jitter of the AND-activated task will be larger than the external input jitter. In our compositional performance approach, this larger jitter will be propagated around the cycle again, resulting in an even larger jitter at the cycle-internal input of the AND-activated task (Section 3.3.2). It is obvious that the jitter appears unbounded if calculated this way.

The problem boils down to the fact that event model propagation as presented so far captures neither correlations between the timing of events in different event streams, nor the fact that the number of tokens in a cycle is fixed. Therefore, the activation jitter for the AND-activated task is calculated very conservatively.

4.5 Analysis idea
Cycle analysis requires detailed consideration of the possible phases between tokens arriving at the cycle-external and the cycle-internal inputs of the AND-activated task. The solution that we propose in the following has the advantage to require only minor modifications to the feedforward system-level analysis already supported by SymTA/S. The idea goes as follows.

We initially assume that the cycle-internal input cannot increase the activation jitter of the AND-activated task. This allows us to 'cut' the cycle-internal edge, rendering a feedforward system which can be analysed as explained in Section 3.2.2. We then calculate the time it takes a token to travel around the cycle, and reason about the validity of the initial assumption.

In the following, the idea is explained for cycles with one initial token. Let us assume an external *periodic with jitter* event model with period $\mathcal{P}_{ext}$ and jitter $\mathcal{J}_{ext}$. Let us define t_{ff}^{min} and t_{ff}^{max} to be the minimum (respectively, maximum) sum of worst-case response times of all tasks belonging to a cycle (the 'time around the cycle') as obtained through analysis of the corresponding feedforward system. Let us further assume that after analysis of the corresponding feedforward system, $t_{ff}^{max} \leq \mathcal{P}_{ext}$.

At system startup, the first token arriving at the cycle-external input will immediately activate the

AND-concatenated task together with the initial token already waiting at the cycle-internal input. No further activation of the AND-activated task is possible until the next token becomes available at the cycle-internal input of that task. If feedforward analysis was valid, then this will take between t_{ff}^{min} and t_{ff}^{max} time units.

The maximum distance between two consecutive external tokens is $\delta_{ext}^{max}(2) = \mathcal{P}_{ext} + \mathcal{J}_{ext}$ (4). From $t_{ff}^{max} \leq \mathcal{P}_{ext}$ follows that it is not possible that the 2nd external token arriving as *late* as possible after the 1st external token has to wait for an internal token.

The 3rd external token can arrive at most $\delta_{ext}^{max}(3) = 2 \times \mathcal{P}_{ext} + \mathcal{J}_{ext}$ after the 1st external token. Therefore, if both the 2nd and the 3rd external tokens arrive as late as possible, then the 3rd arrives $\mathcal{P}_{ext}$ after the 2nd. From $t_{ff}^{max} \leq \mathcal{P}_{ext}$ follows that the 3rd external token arriving as late as possible after the 1st external token cannot wait for an internal token, even if the 2nd external token also arrived as late as possible. This argument can be extended to all further tokens. We infer that no external token arriving as late as possible has to wait for an internal token.

Activation of task b also cannot happen earlier than the arrival of an external token. Therefore, the activating event model of task b is conservatively captured by the external input event model (12). We conclude that our approach is valid for a cycle with $M = 1$ initial token, for which $t_{ff}^{max} \leq \mathcal{P}_{ext}$:

$$\mathcal{P}_{act} = \mathcal{P}_{ext} \; ; \; \mathcal{J}_{act} = \mathcal{J}_{ext} \qquad (12)$$

In [19] it is shown that the approach presented in this Section is also valid for a cycle with $M > 1$ initial tokens, for which $(M - 1) * \mathcal{P}_{ext} < t_{ff}^{max} \leq M * \mathcal{P}_{ext}$. In [19] it is also shown how to extend the approach to nested cycles. In SymTA/S, the feedforward analysis is performed for every cycle, and the required number of initial tokens is calculated from t_{ff}^{max}. This number is then compared against the number of cycle-tokens specified by the user in the same manner as any other constraint is checked.

5 System contexts

Performance analysis as described so far can be unnecessarily pessimistic, because it ignores certain correlations between consecutive task activations or assumes a very pessimistic worst-case load distribution over time.

We have therefore added advanced performance analysis techniques, taking correlations between successive computation or communication requests as well as correlated load distribution into account, in order to yield tighter analysis bounds. Cases where such correlations have a large impact on system timing are especially difficult to simulate and hence are an ideal target for formal performance analysis. We call such correlations 'system contexts'.

In Section 5.1, using an example of a hypothetical set-top box, we review the assumptions made by a typical performance analysis, called 'context blind analysis'. Then, we show the analysis improvements that can be obtained when considering two different types of system contexts separately and also in combination: intra-event stream contexts, which consider correlations between successive computation or communication requests (Section 5.2), and inter-event stream contexts, which consider possible phases between events in different event streams (Section 5.3). The combination of both system contexts is explained in Section 5.4.
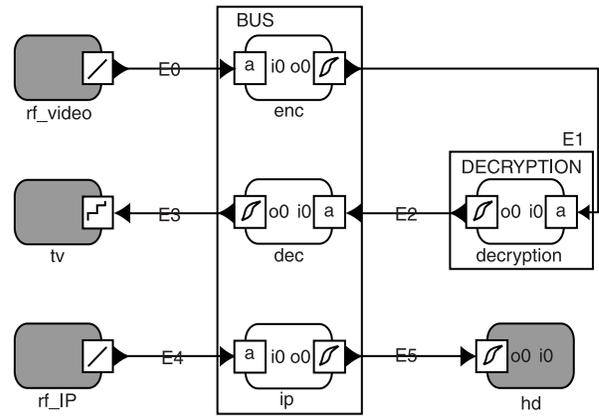

Fig. 13 *Hypothetical set-top-box system*

5.1 Context blind analysis

The SoC implementation of a hypothetical set-top box shown in Fig. 13 is used as an example throughout this Section. The set-top box can process MPEG-2 video streams arriving from the RF-module (rf_video) and sent via the bus (BUS) to the TV (*tv*). In addition, a decryption unit (DECRYPTION) allows us to decrypt encrypted video streams. The set-top box can additionally process IP traffic and download web content via the bus (ip) to the hard disk (hd).

We will focus on worst-case response time calculation for the system bus. We assume static priority-based scheduling on the bus. The priorities are assigned as follows: $enc > dec > ip$. MPEG-2 Video frames are assumed to arrive periodically from the RF-module. The arrival period is normalised to 100. The core execution and communication times of the tasks are listed in Table 1.

The worst-case response time of *ip*, calculated by a context blind analysis, is 170. As can be seen in Fig. 14, even though a data dependency exists between enc and dec, which may even out their simultaneous activation, a context blind analysis assumes that in the worst-case all communication tasks are activated at the same instant. Furthermore, even though MPEG-2 frames may have different sizes depending on their type, a context blind analysis assumes that every activation of enc and dec leads to a maximum transmission time of one MPEG-2 frame.

5.2 Intra-event stream context

Context-blind analysis assumes that, in the worst-case, an every scheduled task executes with its worst case execution time for each activation. In reality, different events often activate different behaviours of a computation task with different WCET, or different bus loads for a communication task. Therefore, a lower maximum load (and a higher minimum load) can be determined for a sequence of successive activations of a higher-priority task if the types of the activating events are considered. This in turn leads to

Table 1: Core execution times

Task	CET
enc	[10, 30]
dec	[10, 30]
ip	[50, 50]
decryption	[40, 40]

156

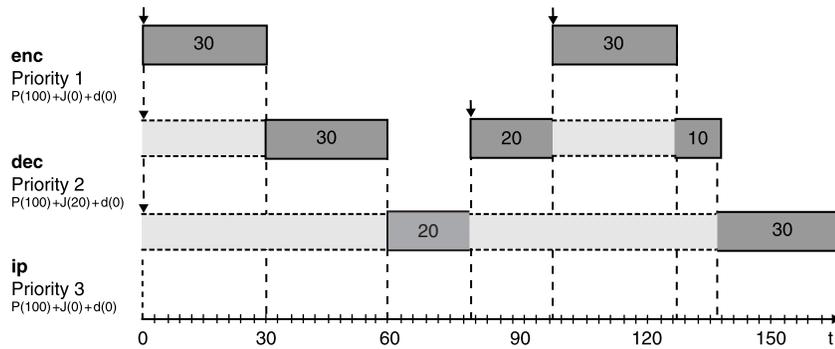*IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005*

Fig. 14 *Worst case response time calculation for ip without contexts, using Sym TA/S*

a shorter calculated worst-case response time (and a longer best case response time) of lower-priority tasks. We call the correlation within a sequence of different activating events an 'intra-event stream context'.

Mok and Chen introduced this idea in [23] and showed promising results for MPEG-streams where the average load for a sequence of I-, P- and B-frames is much smaller than in a stream that consists only of large I-frames, which is assumed by a context-blind worst-case response time analysis. However, the periodic sequence of types of activating events was supposed to be completely known.

In reality, intra-event stream contexts can be more complicated. If no complete information is available about the types of the activating events, it is no longer possible to apply Mok and Chen's approach. Mok and Chen also do not clearly distinguish between different types of events on one hand, and different task behaviours, called modes [24], on the other. However, this distinction is crucial for subsystem integration and compositional performance analysis. Different types of events are a property of the sender, while modes are a property of the receiver. Both can be specified separately from each other and later correlated. Furthermore, it may be possible to propagate intra-event stream contexts along a chain of tasks. It is then possible to also correlate the modes of consecutive tasks.

We extended intra-event stream contexts by allowing minimum- and maximum-conditions for the occurrence of a certain type of events in a sequence of a certain length n, in order to capture partial information about an event stream. n is an arbitrary integer value. A single worst-case and a single best-case sequence of events with length n can be determined from the available min- and max-conditions that can be used to calculate the worst- and best-case load due to any number of consecutive activations of the consumer task. In [25], we have extended static-priority pre-emptive response-time calculation to exploit this idea.

Let us apply this approach to our set-top box example. Suppose that the video stream sent from the RF to the bus is encoded in one of several patterns of I-, P- and B-frames (IBBBBB, IBBPBB, IPBBBB . . .), or that several video streams are interleaved. Therefore, it is impossible to provide a fixed sequence of successive frame types in the video stream. However, it may be possible to determine min- and max-conditions for the occurrence of each frame type.

The communication times of tasks enc and dec depends on the received frame type. I-frames have the largest size and lead to the longest execution time, P-frames have the middle size and B-frames have the smallest size. Therefore, the mode corresponding to the transmission of an I-frame has the largest communication time and the mode corresponding to the transmission of a B-frame has the lowest communication time.

Having both intra-event stream context information and modes of the consumer tasks, we can determine a weigh-sorted worst case sequence of frame types with length n. The reader interested in knowing our algorithm to exploit min- and max-conditions is referred to [25].

Now we can determine for l successive activations of enc and dec the worst case load produced on the bus. This is performed, by iterating through the weight-sorted sequence starting from the first event, adding up loads until the worst case load for l activations has been calculated. If l is bigger than n, the sequence length, we go only through l mod n events and add the resulting load to the load of the whole sequence multiplied by l div n.

In Fig. 15, assuming that the worst case sequence of frame types with length 2 is IP and that the transmission time for an I-frame is 30 and for a P-frame is 20, we show the calculated worst case response time of ip, when considering the available intra-event stream context information. As can be seen, for both tasks enc and dec, the produced load on the bus due to a transmission of successive MPEG-2 frames is smaller than in the context-blind case (see Fig. 14]). This leads to a reduction of the calculated worst-case response time of ip: 150 instead of 170.
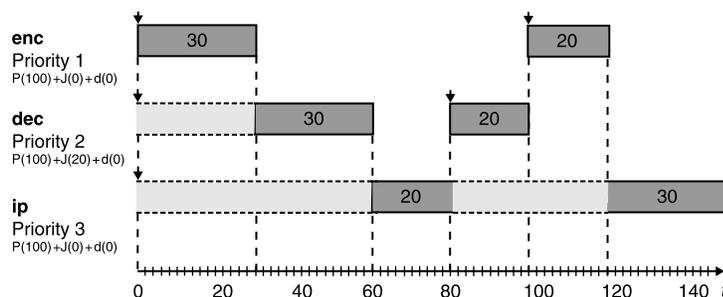

Fig. 15 *Worst case response time calculation for ip considering intra contexts*

enc
Priority 1
P(100)+J(0)+d(0)

30   30

dec
Priority 2
P(100)+J(20)+d(0)

30

ip
Priority 3
P(100)+J(0)+d(0)

10   30   10

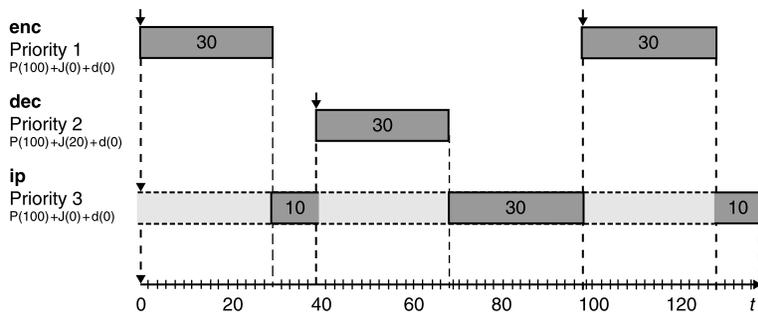0   20   40   60   80   100   120   *t*

Fig. 16 *Worst case response time calculation for ip considering inter contexts*

5.3   Inter-event stream context

Context-blind analysis assumes that all scheduled tasks sharing a resource are independent and that in the worst-case all tasks are activated simultaneous. In reality, activating events are often time-correlated, which rules out simultaneous activation of all tasks. This in turn may lead to a lower maximum number (and higher minimum number) of interrupts of a lower-priority task through higher-priority tasks, resulting in a shorter worst-case response time (and longer best-case response time) of the lower priority task. We call the correlation between time-correlated events in different event streams an inter-event stream context.

Tindell introduced this idea for tasks scheduled by a static priority pre-emptive scheduler [26]. His work was later generalised by Palencia and Harbour [27]. Each set of time-correlated tasks is grouped into a so-called 'transaction'. Each transaction is activated by a periodic sequence of external events. Each task belonging to a transaction is activated when a relative time, called 'offset', elapses after the arrival of the external event.

To calculate the worst-case response time of a task, a worst-case scenario for its execution must be build. Tindell [26] showed that the worst-case interference of a transaction on the response time of a task occurs at the critical instant which corresponds to the most delayed activation of a higher-priority task belonging to the transaction. The activation time of the analysed task and all higher-priority tasks have to happen as soon as possible after the critical instant.

Since all activation times of all higher-priority tasks belonging to a transaction are candidates for the critical instant of the transaction, the worst-case response time of a lower-priority task has to be calculated for all possible combination of all critical instants of all transactions that contain higher priority tasks, to find the absolute worst-case.

Let us apply Tindell's approach to our set-top box example. Owing to the data dependency between enc, decryption, and dec, these tasks are time-correlated. The offset between the activations of enc and decryption corresponds to the execution time of enc. Base on this offset and the execution time of decryption, we can calculate the offset between the activations of enc and dec.

To show in isolation the analysis improvement due to inter-event stream contexts, we will assume for now that all video-frames are I-frames. Figure 16 shows for the inter-event stream context case the calculated worst case response time of ip due to interrupts by enc and dec. As can be seen, a gap exists between successive executions of enc and dec. Since ip executes during this gaps, one interrupt less of ip is calculated (in this case through enc). This leads to a reduction of the calculated worst-case response time of ip: 140 instead of 170.

In Fig. 17, analysis improvements with inter-event stream context information in relation to the context-blind case are shown as a function of the offset between enc and dec, which is equal to the execution time of the decryption unit.

Curve (i) shows the reduction of the calculated worst-case response time of dec. Depending on the offset, dec is either partially offset value < 30), completely (offset value > 70) or not interrupted at all by enc (offset value between 30 and 70). The latter case yields a maximum reduction of 50%.

Curves (ii)–(vii) show the reduction in the calculated worst-case response time of ip for different IP traffic sizes. The reduction is visible in the curves as dips. If no gaps exists between two successive executions of enc and dec, no worst-case response reduction of ip can be obtained (offset value < 30 or > 70). If a gap exists, then sometimes one interrupt less of ip can be calculated (either through enc or dec), or there is no gain at all (curves (iv) and (vi)). Since the absolute gain that can be obtained equals the smaller worst case execution time of enc and dec, the relative worst-case response time reduction is bigger for shorter IP-traffic.

An important observation is that inter-event stream context analysis reveals the dramatic influence that a small local change, in our example the speed of the decryption unit reading data from the bus and writing the results back to the bus, can have on system-performance, in our example the worst-case transmission time of lower-priority IP traffic.

5.4   Combination of contexts

Inter-event stream contexts allow us to calculate a tighter number of interrupts of a lower-priority task through higher-priority tasks. Intra-event stream contexts allow us to calculate a tighter load for a number of successive
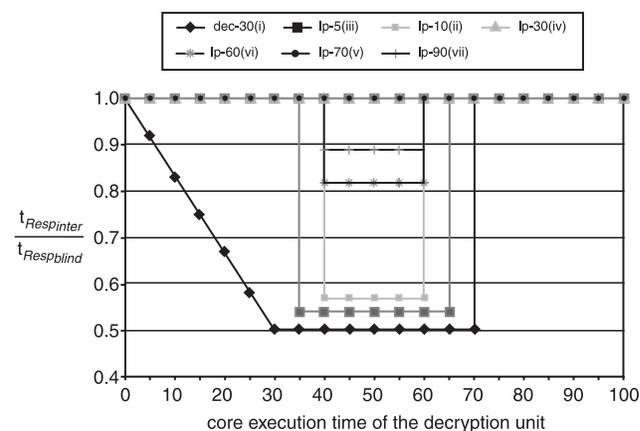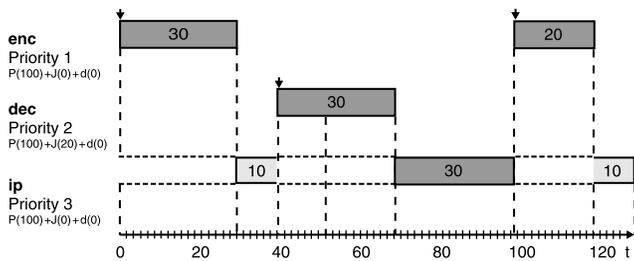

$\frac{t_{Resp_{inter}}}{t_{Resp_{blind}}}$

core execution time of the decryption unit

Fig. 17 *Improved worst case response time calculation due to inter contexts*

Fig. 18 *Worst-case response time calculation for ip with combination of contexts*

activations of a higher-priority task. The two types of contexts are orthogonal: the worst-case response time of a lower-priority task is reduced both because fewer high-priority task activations can interrupt its execution during a certain time interval and because the time required to process a sequence of activations of each higher-priority tasks is reduced. Therefore, performance analysis can be further improved if it is possible to consider both types of contexts in combination. This is shown in Fig. 18 for the worst-case response time calculation of *ip*: 130 instead of 170.

In Fig. 19, we show analysis improvements considering both inter- and intra-event stream contexts in relation to the context-blind case as a function of the offset between enc and dec. Curve (i) shows the reduction of the calculated worst-case response time of dec. Since dec is interrupted at most once by enc, and the worst-case load produced due to one activation of enc is the transmission time of one I-frame, no improvement is obtained through the context combination in comparison to curve (i) in Fig. 17.

Curves (ii)–(vii) show the reduction of the calculated worst-case response time of ip for different IP traffic sizes. When comparing curves (ii) and (iii) (IP traffic sizes of 5 and 10) to curves (ii) and (iii) in Fig. 17, it can be seen that no improvement is obtained through the context combination. This is due to the fact that, ip is interrupted at most once by enc and at most once by dec. Therefore, as in case (i), the calculated worst-case load produced by the video streams is the same no matter whether the available intra-event stream context information considered or not.

Curve (iv) shows that for an IP traffic size of 30 no improvements are obtained through the context combination
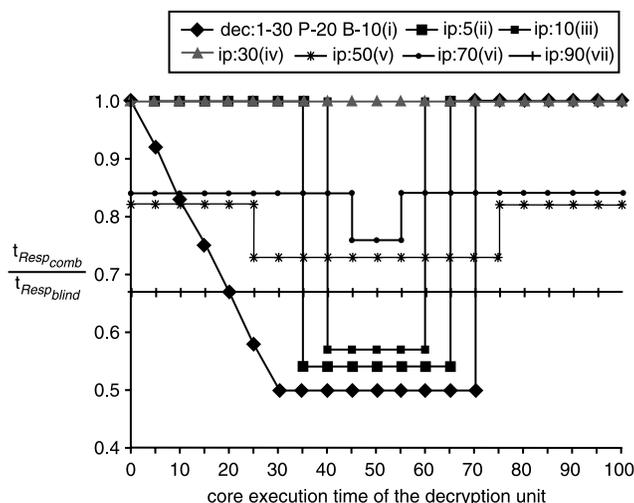
in comparison to the context-blind case. This is due to the fact that, for all offset-values, ip is interrupted exactly once by enc and exactly once by dec, and that the calculated worst-case load produced by the video streams due to one activation is the same no matter whether intra-event stream contexts are considered or not.

Curves (v) and (vi) show that for IP traffic sizes of 50 and 70 improvements are obtained as a result of the context combination in comparison to both the intra- and inter-event stream context analysis. Let us focus on curve (v). Since intra- and inter-event stream contexts are orthogonal, the reduction of the calculated worst-case response time of *ip* due to the intra-event stream context is constant for all offset values. Since no reduction due to inter-event stream context can be obtained for an offset value of 0 (equivalent to the inter-event stream context-blind case), we are sure that the reduction shown in the curve for this offset value is only a result of the intra-event stream context. On the other hand, the additional reduction between the offset values 25 and 75 is obtained due to the inter-event stream context.

Curve (vii) shows that, for an IP traffic size of 90, even though the intra-event stream context leads to an improvement (see curve (vii) in Fig. 17), the improvement due to the intra-event stream context dominates, since no dip exists in the curve, i.e. no additional improvements are obtained due to the context combination in comparison to the intra-event stream context case.

This example shows that considering the combination of system contexts can yield considerably tighter performance analysis bounds compared to a context-blind analysis. Equally importantly, this example reveals the dramatic influence that a small local change can have on system performance. Systematically identifying such system-level influences of local changes is especially difficult using simulation due to the large number of implementations that would have to be synthesised and executed. On the other hand, formal performance analysis can systematically and quickly identify such corner cases. All these results took a couple of milliseconds to compute using SymTA/S.

6 Design space exploration for system optimisation

In this Section we will give a brief overview about the evolutionary design space exploration and system optimisation techniques used in SymTA/S. We will first describe system parameters which can be subject to optimisation and how they can be composed to define the search space. Then we will give some examples of metrics expressing desired or undesired system properties, forming so-called optimisation objectives. Finally, we will explain the design space exploration loop performed in SymTA/S.

6.1 Search space

The search space and the optimisation objectives can be multidimensional, which means that several system parameter can be explored simultaneously to optimise multiple objectives. Possible search parameters include:

* mapping of tasks onto different resources
* changing priorities on priority-scheduled resources
* changing time slot sizes and time slot order on TDMA or round robin scheduled resources
* changing the scheduling policy on a resource
* modifying resource speed.

Fig. 19 *Analysis improvements due to combination of intra and inter contexts*

IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

159
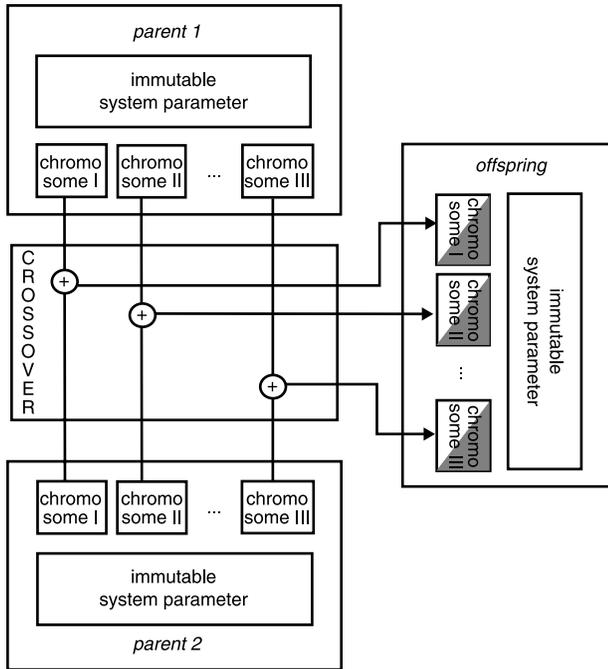
Fig. 20 *Functionality of crossover operator in SymTA/S*

Since EAF*s* in SymTA/S allow us to control the timing of events and data between connected components (see Section 3.4), additional exploration is possible using systematic traffic shaping. Thereby, d_{min}-EAF*s*, allowing us to extend the minimum distance between successive output events, are of particular interest. We will see in Section 8.2 that they can be used to weaken the global impact of bursts, which can lead to interesting optimisation results.

The compositional structure of SymTA/S allows a flexible coding of the search space. Search parameters can be defined very precisely. They can be limited locally to one or several components, or can be of global scope. The combination of a search parameter and its scope is called a 'chromosome' in the context of evolutionary algorithms. Chromosomes form modular entities and can be combined arbitrarily to span the search space. An individual, representing a specific system configuration, consists of immutable system parameters and a set of chromosomes, which represent the variable system parameters. This modular design supports the explicit combination of local and global exploration techniques. For example, the designer can optimise the TDMA slot sizes on a single resource while allowing system-wide traffic shaping, or optimise the priority assignments on all priority scheduled resources in the system while varying the speed of a single resource.

Each chromosome carries the variation operators necessary for combination with other chromosomes of its type. In SymTA/S we currently use the most popular operators: mutation and crossover. The operators are applied chromosome-wise. Figure 20 illustrates the functionality of the crossover operator.

6.2 Optimisation objectives

Optimisation objectives can be any kind of metric defined on desired or undesired properties of the considered system.

Note that some metrics only make sense in combination with constraints. Each individual is associated with a fitness vector containing one entry for every concurrent optimisation objective. We use the following notation:

$R = $ maximum response time of a task or
maximum end-to-end latency along a path

$D = $ deadline (task or end-to-end)

$\omega = $ constant weight > 0

$k = $ number of tasks or
number of constrained tasks/paths in the system

and define the following example optimisation objectives:

1. minimisation of the (weighted) sum of completion times

$$\sum_{i=1}^{k} \omega_i R_i$$

2. minimisation of the maximum lateness

$$\max(R_1 - D_1, \dots, R_k - D_k)$$

3. maximisation of the minimum earliness

$$\min(D_1 - R_1, \dots, D_k - R_k)$$

4. minimisation of the (weighted) average lateness

$$\sum_{i=1}^{k} \omega_i (R_i - D_i)$$

5. maximisation of the (weighted) average earliness

$$\sum_{i=1}^{k} \omega_i (D_i - R_i)$$

6. minimisation of end-to-end latencies
7. minimisation of jitters
8. minimisation of the sum of communication buffer sizes.

The choice of the metric for optimisation of a specific system is very important to obtain satisfying results. Example metrics 4 and 5, for instance, express the average timing behaviour of a system with regard to its timing constraints. They might mislead an evolutionary algorithm and prevent it from finding system configurations fulfilling all timing constraints, since met deadlines compensate linearly for missed dead-lines. For systems with hard real-time constraints, metrics with higher penalties for missed deadline and fewer rewards for met deadlines can be more appropriate, since they lead to a more likely rejection of system configurations violating hard deadline constraints. The following example metric penalises violated deadlines in an exponential way and can be used to optimise the timing properties of a system with hard real-time constraints:

$$\sum_{i=1}^{k} c_i^{R_i - D_i}, c_i > 1 \; constant$$

Performing a multi-objective optimisation in SymTA/S usually leads to the discovery of several Pareto-optimal.

Definition 5 (Pareto-optimal): Given a set V of k-dimensional vectors $v \in \mathbb{R}^k$, a vector $v \in V$ dominates a vector $w \in V$ iff for all elements $0 \leq i < k$ we have $v_i \leq w_i$ and for at least one element l we have $v_l < w_l$.

A vector is called pareto-optimal iff it is not dominated by any other vector in V.

Pareto-optimal solutions represent a certain tradeoff between two or more objectives, leaving it to the designer to decide which solution to adopt. In our case, individuals
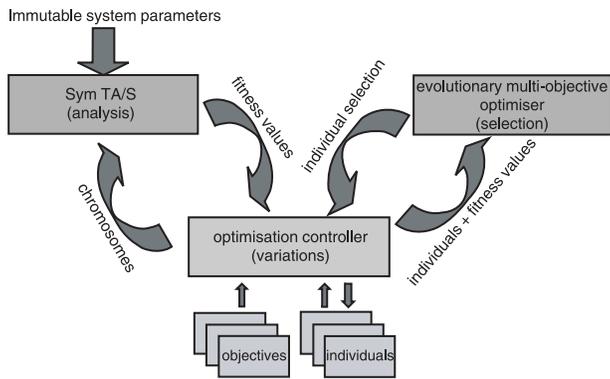
Fig. 21 *Design space exploration loop*

with pareto-optimal fitness vectors represent the different system design trade-offs.

6.3 Design space exploration loop

Figure 21 shows the design space exploration loop performed in SymTA/S The optimisation controller, is the central element. It is connected to SymTA/S, which performs the analysis of the individuals, and to an evolutionary multi-objective optimiser. The latter is responsible for the problem-independent part of the optimisation problem, i.e. elimination of individuals and selection of interesting individuals for variation. Currently, we use FEMO (fair evolutionary multiobjective optimiser) [28] and SPEA2 (strength Pareto evolutionary algorithm 2) [29] for this part. Both are coupled via PISA (platform and programming language independent interface for search algorithm) [30]. Note that the problem-specific part of the optimisation problem is coded inside the chromosomes and their variation operators.

An example for a variation operator is order crossover [31]. It is applicable for priority assignments coded as lists, in which each entry corresponds to the priority of a specific task. The offspring inherits the priority assignments of the tasks between two randomly chosen positions in the priority list from the first parent. The remaining priorities are inherited from the second parent, beginning at the first position of its priority list, starting from the second chosen position and skipping over all priorities already assigned in the offspring.

Example:

Parent 1	:	1	2	3	4	5	6
Parent 2	:	3	2	6	5	4	1
Cross Pts	:			*		*	
Offspring	:	6	1	3	4	5	2

Before the exploration loop is started, SymTA/S is initialised with the immutable part of the system architecture. In order to analyse a design alternative represented by an individual, its chromosomes are transformed into commands and applied to SymTA/S. This completes the system design, which can then be analysed by SymTA/S. After analysis the optimisation controller requests the system parameters necessary to determine the fitness values according to the optimisation objectives. This procedure is performed for every individual currently considered. The individuals and their fitness vectors are then sent to the evolutionary multi-objective optimiser. On the basis of the fitness values the optimiser creates two sets. One set contains individuals selected for elimination, the other contains individuals selected for variation

(mutation and crossover). These sets are communicated to the optimisation controller, which deletes eliminated individuals and performs the requested mutation and crossover operations. The next iteration is then started with the surviving and newly created individuals.

Note that the selection of individuals for elimination and variation depends on the used multi-objective optimiser. For instance, FEMO [28], eliminates all dominated individuals in every iteration and pursuits a fair sampling strategy, i.e. each parent participates in the creation of the same number of off-springs. This leads to a uniform search in the neighbourhood of elitist individuals.

The performance of the search procedure in SymTA/S is affected by the search strategy of the optimiser, the coding of the chromosomes and their variation operations as well as the choice of the optimisation objectives. As far as the optimizer is concerned, it is known that no general purpose optimization algorithm exists that is able to optimize effectively all kinds of problems [32].

7 Sensitivity analysis

Most analysis techniques known from literature give a pure Yes/No answer regarding the timing behaviour of a specific system with respect to a set of timing constraints defined for that system. Usually the analyses consider a predefined set of input parameters and determine the response times, and thus the schedulability of the system.

However, in a realistic system design process it is important to get more information with respect to the effects of parameter variations on system performance, as such variations are inevitable during implementation and integration. Capturing the bounds within which a parameter can be varied without violating the timing constraints offers more flexibility for the system designer and supports future changes. These bounds shows how sensitive the system or system parts are to system configuration changes.

Liu and Layland [1] defined a maximum load bound on a resource that guarantees the schedulability of that resource when applying a rate monotonic priority assignment scheme. The proposed algorithm is limited to specific system configurations: periodically activated tasks, tasks with deadlines at the end of their period and tasks that do not share common resources (like semaphores) or that do not intercommunicate.

Later on, Lehoczky *et al.* [33] extended this approach to systems with arbitrary priority assignment. However, his approach does not go beyond the limitations mentioned above. Vestal [34] proposed a fixed-priority sensitivity analysis for tasks with linear computation times and linear blocking time models. His approach is still limited to tasks with periodic activation patterns and deadlines equal to the period. Punnekkat *et al.* [35] proposed an approach that uses a combination of a binary search algorithm and a slightly modified version of the response time schedulability tests proposed by Audsley and Tindell [7, 36].

In the following we give a brief overview about the sensitivity analysis algorithm and the analysis models and metrics used in SymTA/S. As already mentioned above, different approaches were proposed for the sensitivity analysis of different system parameters. However, all can perform only single resource analysis as they are bounded by local constraints (tasks deadlines). Due to a rapid increase in system complexity and heterogeneity, the current distributed systems usually have to satisfy global constraints rather than local ones. End-to-end deadlines or global buffer limits are an example of such constraints. Hence, the formal approaches used for the sensitivity

analysis at resource level cannot be transformed and applied at the system level, as this implies huge effort and less flexibility.

Our sensitivity analysis framework combines a binary search technique and the compositional analysis model implemented in SymTA/S. As described in Section 3, SymTA/S couples the local scheduling analysis algorithms into a global analysis model.

Since deadlines are the major constraints in real-time systems it makes sense to measure the sensitivity of path latencies. As the latency of a path is determined by the response times of all tasks along that path, and the response time of a task directly depends on its core execution time, we consider the following issues as important metrics for the sensitivity analysis:

1. Maximum permissible variation of the core execution time of a task without violating the system constraints or the system schedulability.
2. Minimum speed of a resource. The decrease of a resource speed directly affects the core execution times of all tasks mapped on that resource but also reduces the energy required by that resource.

Variation of task execution/computation times: The search interval is determined by the current WCET value $t_{core,max}$ and the value corresponding to the maximum load bound on the resource holding the task. If we denote by R_{load} the current load on the resource R and by $R_{load,max}$ the maximum load bound on resource R, then the search interval is

$$[t_{core,max}; t_{core,max} + \mathcal{P} \times (R_{load,max} - R_{load})]$$

where $\mathcal{P}$ represents the activation period in the case of periodic tasks or the minimum interarrival period in the case of sporadic tasks. If, for the current system configuration, the constraints are violated or the system is not schedulable, then the search interval is $[0; t_{core,max}]$.

The algorithm selects the middle interval value and verifies if the constraints are satisfied for the configuration obtained by replacing the task WCET value with the selected value. If *yes*, then the second half of the interval becomes the new search interval, otherwise the first half of the interval is searched. The algorithm iterates until the size of the search interval becomes smaller than a specific predefined value.

Variation of resource speed: The same algorithm is applied to find the minimum resource speed. If, for the current configuration, the constraints are satisfied and the system is schedulable, then the search space is determined by $[R_{speed,min}; R_{speed}]$, where R_{speed} is the current speed factor (usually 1) and $R_{speed,min}$ is the speed factor corresponding to the maximum resource load bound. Otherwise, the search space is $[R_{speed}; R_{speed,max}]$, where $R_{speed,max}$ is the speed factor corresponding to the minimum resource load bound (below 1%).

The ideal value for the maximum resource load bound is 100%. We performed experiments on different system models and we observed that for load values $> 98\%$ the runtime of the sensitivity analysis algorithm drastically increases. This is due to an increase of the analysed period (busy period) in the case of local analysis scheduling algorithms. However, a resource load $> 98\%$ is not realistic due to variations of the system clock frequency or other distorting factors.

8 System-on-chip example

In this Section using SymTA/S, we apply the techniques from the preceding Sections to analyse the performance of a system-on-chip example shown in Fig. 22.

The embedded system in Fig. 22 represents a hypothetical SoC consisting of a micro-controller (uC), a digital signal processor (DSP) and dedicated hardware (HW), all connected via an on-chip bus (Bus). Dsp and uC are equipped with local memory. The HW acts as an interface to a physical system. It runs one task (*sys_if*) which issues actuator commands to the physical system and collects routine sensor readings. *sys_if* is controlled by task ctrl, which evaluates the sensor data and calculates the necessary actuator commands. ctrl is activated by a periodic timer (tmr) and by the arrival of new sensor data (AND-activation in a cycle). We assume two initial tokens in the cycle.

The physical system is additionally monitored by three sensors (sens1 – sens3), which produce data sporadically as a reaction to irregular system events. These data are registered by an OR-activated monitor task (mon) on the uC, which
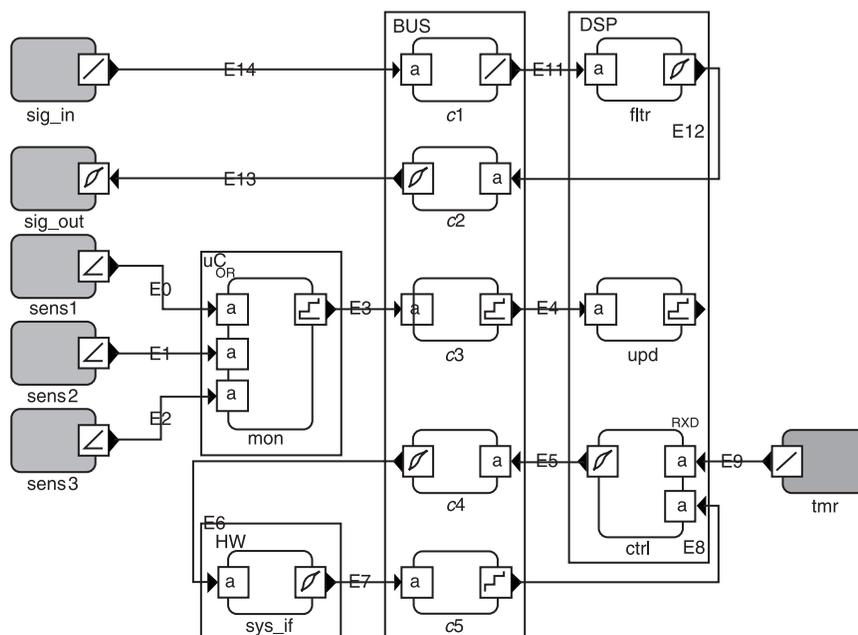

Fig. 22 *System on chip example*

162

IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

Table 2: Core execution and communication times

Computation			Communication	
task	C		task	C
mon	[10, 12]		$c1$	[8, 8]
sys_if	[15, 15]		$c2$	[4, 4]
fltr	[12, 15]		$c3$	[4, 4]
upd	[5, 5]		$c4$	[4, 4]
ctrl	[20, 23]		$c5$	[4, 4]

decides how to update the control algorithm. This information is sent to task upd on the DSP, which updated parameters into shared memory.

The *DSP* additionally executes a signal-processing task (fltr), which filters a stream of data arriving at input sig_in and sends the processed data via output sig_out. All communication, except for shared-memory on the DSP, is carried out by communication tasks c1–c5 over the on-chip Bus. Core execution times for each task are shown in Table 2.

We assume the following event models at system inputs (Table 3).

To function correctly, the system has to satisfy a set of path latency constraints (Table 4). Constraints 1 and 3 have been explicitly specified by the designer. The 2nd constraint implicitly follows from the fact that the cycle contains two initial tokens. Constraint 3 is defined for causally dependent tokens [20]. We shall also impose a maximum jitter constraint at output *sig_out* (Table 5).

8.1 Analysis

We will use static priority scheduling both on the DSP and the Bus. The priorities on the Bus (respectively, DSP) are assigned as follows: $c1 > c2 > c3 > c4 > c5$ and $fltr > upd > ctrl$.

Table 3: Event models at external system inputs

Input	s/p	$\mathcal{P}_{in}$	J_{in}	$d_{min,in}$
sens 1	s	1000	0	0
sens 2	s	750	0	0
sens 3	s	600	0	0
sig_in	p	60	0	0
tmr	p	70	0	0

Table 4: Path latency constraints

Constraint no.	Path	Maximum latency
1	sens 1, sens 2, sens 3 → *upd*	70
2	sig_in → sig_out	60
3	cycle (ctrl → ctrl)	140

Table 5: Output jitter constraint

Constraint no.	Output	Event model period	Event model jitter
4	sig_out	$\mathcal{P}_{sig\_out} = 60$	$J_{sig\_out,max} = 18$

Performance analysis results were obtained using SymTA/S [8]. In the first step, SymTA/S performs OR-concatenation of the output event models of sens1–sens3 and obtains the following *sporadic* activating event model for task *mon*:

$$\mathcal{P}_{act} = \mathcal{P}_{OR} = 250, \ \mathcal{J}_{act} = \mathcal{J}_{OR} = 500$$

The large jitter is due to the fact that input events happening at the same time lead to a burst of up to three activations (we assume no correlations between sens1 and sens3). Since task mon is the only task mapped onto uC, we can now perform local scheduling analysis for this resource, in order to calculate the minimum and maximum response times, as well as the output event model of task mon. The results of this analysis are shown in Table 6.

The worst-case response time of task mon increases compared to its worst-case core execution time, since later activations in a burst have to wait for the completion of the previous activations. The output jitter increases by the difference between maximum and minimum core execution times compared to the activation jitter. The minimum distance between output events equals the minimum core execution time.

At this point, the rest of the system cannot be analysed, because on every resource activating event models for at least one task are missing. SymTA/S therefore generates a conservative starting-point by propagating all output event models along all paths until an initial activating event model is available for each task. SymTA/S then checks that the system cannot be overloaded in the long term. This calculation requires only activation periods and worst-case core execution times and thus can be done before response-time calculation.

System-level analysis can now be performed by iterating local scheduling analysis and event model propagation. SymTA/S determines that task ctrl belongs to a cycle, checks that AND-concatenation is selected, and then proceeds to analyse the corresponding feedforward system. SymTA/S executes until a fix-point for the whole system has been reached, and then compares the calculated performance values against performance constraints.

Table 7 shows the calculated response times of the computation and communication tasks with and without taking into account inter contexts. We observe that the exploitation of context information leads to much tighter response time intervals in the given example. This in turn

Table 6: Scheduling analysis results on uC

Task	s/p	Activating EM	r	s/p	Output EM
mon	s	$\mathcal{P}(250) \ \mathcal{J}(500)$ $d(0)$	[10, 36]	s	$\mathcal{P}(250) \ \mathcal{J}(526)$ $d(10)$

Table 7: Context blind and sensitive analysis

Comp. task	$Resp_{blind}$	$Resp_{sens}$	Comm. tasks	$Resp_{blind}$	$Resp_{sens}$
mon	[10, 36]	[10, 36]	$c1$	[8, 8]	[8, 8]
sys_if	[15, 17]	[15, 15]	$c2$	[4, 12]	[4, 4]
fltr	[12, 15]	[12, 15]	$c3$	[4, 16]	[8, 12]
upd	[5, 22]	[5, 22]	$c4$	[4, 28]	[8, 20]
ctrl	[20, 53]	[20, 53]	$c5$	[4, 32]	[8, 32]

IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

163

Table 8: Constraint values for context blind and sensitive analysis

No.	Constraint	Inter-context-blind	Inter-context-sensitive
1	sens 1, sens 2, sens 3 → upd	74	70
2	sig_in → sig_out	35	27
3	cycle (ctrl → ctrl)	130	120
4	$J_{\text{sig\_out,max}} = 18$	11	3

reduces the calculated worst-case values for the constrained parameters. Table 8 shows that, in contrast to the inter-context blind analysis, all system constraints are satisfied when performance analysis takes inter-context into account. In other words, a context blind analysis would have discarded a solution which is in reality valid.

8.2 Optimisations

Let us now try to optimise our example architecture Optimisation objectives are the four defined constraints. We try to minimise the latencies on paths 1–3 and the jitter at output sig_out.

In the first experiment our search space consists of the priority assignments on the BUS and the DSP. Table 9 shows the existing Pareto optimal solutions. In the first two columns, tasks are ordered by priority, highest priority on the left. In the last four columns, we give the actual value for all four constrained values. The best reached values for each constraint are emphasised.

As we can observe, there are several possible solutions, each with its own advantages and disadvantages. We also

observe that in each solution one constraint is only barely satisfied. A designer might want to find some alternative solutions where all constraints are fulfilled with a larger margin to the respective maximum values.

We extend our search space by using a shaper at the output of task mon. It is making sense to perform traffic shaping at this location, because the OR-activation of mon can lead in the worst-case scenario to bursts at its output. That is, if all three sensors trigger at the same time, mon will send three packets over the BUS with a distance of 10 time units, which is its minimum core execution time. This transient load peak affects the overall system performance in a negative way. A shaper is able to increase this minimum distance in order to weaken the global impact of the worst-case burst.

Table 10 shows Pareto optimal solutions using a shaper at the output of mon extending the minimum distance of successive events at the output of mon to 12 time units, and thus weakening the global impact of the worst-case burst. The required buffer for this shaper is minimal, because at most one packet needs to be buffered at any time.

We observe that several new solutions are found. Not all best values for each constraint from the first attempt are reached, yet configurations 3 and 5 are interesting since they are more balanced regarding the constraints.

8.3 Sensitivity analysis

We applied the sensitivity analysis algorithms presented in Section 7 to the Pareto optimal system configurations obtained in Section 8.2 The Δ values show the maximum permissible changes in tasks execution/computation times. Table 11 presents the current task execution times and the Δs obtained for the system configurations described in Table 9.

Table 9: Pareto optimal solutions

No.	Bus tasks	DSP tasks	con. 1	con. 2	con. 3	con. 4
1	c1, c2, c3, c4, c5	upd, fltr, ctrl	55	42	120	18
2	c1, c2, c4, c3, c5	upd, fltr, ctrl	59	42	112	18
3	c2, c1, c4, c5, c3	upd, fltr, ctrl	63	42	96	18
4	c1, c2, c3, c4, c5	fltr, upd, ctrl	70	27	120	3

Table 10: Pareto optimal solutions: shaper at mon output

No.	Bus tasks	DSP tasks	con. 1	con. 2	con. 3	con. 4
1	c2, c1, c3, c4, c5	upd, fltr, ctrl	**59**	42	120	18
2	c1, c2, c4, c3, c5	upd, fltr, ctrl	63	42	112	18
3	c3, c2, c1, c4, c5	fltr, upd, ctrl	64	42	120	11
4	c2, c1, c5, c4, c3	upd, fltr, ctrl	67	42	**96**	18
5	c2, c3, c1, c5, c4	fltr, upd, ctrl	68	**31**	134	**7**

Table 11: Sensitivity analysis of tasks execution/computation times

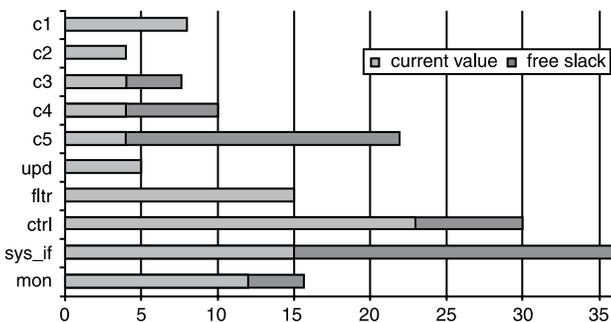|	c1	c2	c3	c4	c5	upd	fltr	ctrl	sys_if	mon
WCET	8	4	4	4	4	5	15	23	15	12
No.	$\Delta c1$	$\Delta c2$	$\Delta c3$	$\Delta c4$	$\Delta c5$	Δupd	$\Delta fltr$	$\Delta ctrl$	Δsys_if	Δmon
1	0	0	1.11	3.33	10	0	0	**7**	**13**	**5**
2	0	0	**3.66**	**6**	**18**	0	0	**7**	**21**	3.66
3	0	0	2.33	2.5	2.5	0	0	**7**	9	2.33
4	0	0	0	3.33	13.5	0	0	**7**	13	0

Fig. 23 *Slack values corresponding to task core times*

Figure 23 shows the current task times and the slack values corresponding to No.2 in Table 11.

As future work we will implement the values obtained by the sensitivity analysis as optimisation objectives in the exploration framework presented in Section 6.

9 Conclusion

The component integration step is critical in MpSoC design since it introduces complex component performance dependencies, many of them cannot be fully overseen by anyone in a design team. Finding simulation patterns covering all corner cases will soon become virtually impossible as MpSoCs grow in size and complexity, and performance verification is increasingly unreliable. In industry, there is an urgent need for systematic performance verification support in MpSoC design.

We have seen that the host of work in formal real-time analysis can be nicely applied to individual, local components or subsystems. However, the well established view on scheduling analysis has shown to be incompatible with the component integration style which is common practice in MpSoC design due to heavy component reuse. The recently adopted event stream view on component interactions represents a significant improvement for all kinds of system performance related issues.

First, the stream model elegantly illustrates the consequences of (*a*) resource sharing and (*b*) component integration for two of the main sources of complexity. This helps to identify previously unknown global performance dependencies, while tackling the scheduling problem itself locally where it can be overseen.

Secondly, the use of intuitive stream models such as periodic events, jitter, burst and sporadic streams, allows us to adopt existing local analysis and verification techniques. In particular, SymTA/S provides automatic interfacing and adaptation among the most popular and practically used event stream models. In other words, SymTA/S is the enabling technology for the reuse of known local component design and verification techniques without compromising global analysis.

In this paper, we have surveyed the basic ideas underlying the SymTA/S technology. We subsequently introduced a variety of features that enable the analysis of complex embedded applications which can be found in practice. This includes multi-input tasks with complex activation functions, cyclic functional dependencies between tasks, systems with mutually exclusive execution modes, and correlated task execution (intra-and inter-contexts). These powerful concepts make SymTA/S a unique performance analysis tool that verifies end-to-end deadlines, buffer over-underflows and transient overloads. SymTA/S eliminates key performance pitfalls and systematically guides the designer to likely sources of constraint violations.

The analysis with SymTA/S is extremely fast (10 s for the system in Section 8, including optimisation). The turnaround times are within seconds. This opens the door to all sorts of explorations, which is absolutely necessary for system optimisation. SymTA/S uses genetic algorithms to automatically optimise systems with respect to multiple goals such as end-to-end latencies, cycles, buffer memory and others. Exploration is also useful for sensitivity analysis in order to determine slack and other popular measures of flexibility. This is specifically useful in systems which might experience later changes or modifications, a design scenario often found in industry. We have carried out a large set of experiments that demonstrate the application of SymTA/S and the usefulness of the results.

We have already applied the technology in case studies in co-operation with industry partners in telecommunications, multimedia, and automobile manufacturing. The cases had a very different focus. In one telecommunications project, we resolved a severe transient-fault system integration problem that not even prototyping could solve. In the multimedia case study, we modelled and analysed a complex two-stage dynamic memory scheduler to derive maximum response times for buffer sizing and priority assignment. In several auto-motive studies, we showed how the technology enables a formal software certification procedure. The case studies have demonstrated the power and wide applicability of the event flow interfacing approach. The approach scales well to large, heterogeneous embedded systems including MpSoC. The modularity allows us to customise SymTA/S libraries to the specific needs of our partners.

We consider the SymTA/S approach to be a serious alternative or supplement to performance simulation. The unique technology allows comprehensive system integration and provides much more reliable performance analysis results at far less computation time.

10 References

1 Liu, C.L., and Layland, J.W.: 'Scheduling algorithm for multi-programming in a hard-real-time environment', *J. ACM*, 1973, **20**, pp. 45–61
2 Jensen, C.L.E., and Tokuda, H.: 'A time-driven scheduling model for real-time operating systems'. Proc. 6th IEEE Real-Time Systems Symp. (RTSS 1985) IEEE CS Press, pp. 112–122
3 Tindell, K., and Clark, J.: 'Holistic schedulability analysis for distributed real-time systems', *Microprocess. Microprogr.*, 1994, **40**, pp. 117–134
4 Gutierrez, J.J., Palencia, J.C., and Harbour, M.G.: 'On the schedulability analysis for distributed hard real-time systems'. Proc. 9th Euromicro Workshop on Real-Time Systems, Toledo, Spain, June 1997, pp. 136–143
5 Gresser, K.: 'An event model for deadline verification of hard real-time systems'. Proc. 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland, 1993, pp. 118–123
6 Thiele, L., Chakraborty, S., and Naedele, M.: 'Real-time calculus for scheduling hard real-time systems'. Proc. Int. Symp. on Circuits and Systems (ISCAS), Geneva, Switzerland, 2000
7 Tindell, K.W.: 'An extendible approach for analysing fixed priority hard real-time systems', *J. Real-Time Syst.*, 1994, **6**, (2), pp. 133–152
8 Hamann, A., Henia, R., Jersak, M., Racu, R., Richter, K., and Ernst, R.: 'SymTA/S – symbolic timing analysis for systems'. http://www.symta.org/
9 Richter, K., and Ernst, R.: 'Event model interfaces for heterogeneous system analysis'. Proc. Design, Automation and Test in Europe Conf. (DATE'02), Paris, France, Mar. 2002
10 Richter, K., Ziegenbein, D., Jersak, M., and Ernst, R.: 'Model composition for scheduling analysis in platform design'. Proc. 39th Design Automation Conf., New Orleans, USA, June 2002
11 Richter, K., Jersak, M., and Ernst, R.: 'A formal approach to MpSoC performance verification', *Computer*, 2003, **36**, (4)
12 Richter, K., Racu, R., and Ernst, R.: 'Scheduling analysis integration for heterogeneous multiprocessor SoC. Proc. 24th Int. Real-Time Systems Symp. (RTSS'03), Cancun, Mexico, Dec. 2003

IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

165

13 Cruz, R.L.: 'A calculus for network delay', *IEEE Trans. Inf. Theory*, 1991, **37**, (1), pp. 114–141

14 Thiele, L., Chakraborty, S., Gries, M., and Künzli, S.: 'Design space exploration of network processor architectures', in Franklin, M., Crowley, P., Hadimioglu, H., and Onufryk, P. (Eds.): 'Network Processor Design Issues and Practices' (Morgan Kaufmann, 2002) **1**, chap 4, pp. 55–90

15 Richter, K.: 'Compositional performance analysis'. PhD thesis, Technical University of Braunschweig, 2004

16 Tindell, K., Kopetz, H., Wolf, F., and Ernst, R.: 'Safe automotive software development'. Proc. Design, Automation and Test in Europe (DATE'03), Munich, Germany, Mar. 2003

17 Technical University of Braunschweig: 'SymTA/S – symbolic timing analysis for systems'. http://www.symta.org

18 Jersak, M., and Ernst, R.: 'Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. Proc. 40th Design Automation Conf., Annaheim, USA, June 2003

19 Jersak, M.: 'Compositional performance analysis for complex embedded applications' PhD thesis, Technical University of Braunschweig, 2004

20 Ziegenbein, D.: 'A compositional approach to embedded system design'. PhD thesis, Technical University of Braunschweig, 2003

21 Lee, E.A., and Messerschmitt, D.G.: 'Synchronous dataflow', *Proc. IEEE*, 1987, **75**, (9), pp. 1235–1245

22 Houri, M.Y.: 'Task graph analysis with complex dependencies' Master's thesis, Institute of Computer and Communication Networks Engineering, Technical University of Braunschweig, 2004

23 Mok, A., and Chen, D.: 'A multiframe model for real-time tasks', *IEEE Trans. Softw. Eng.*, 1997, **23**, (10), pp. 635–645

24 Ziegenbein, D., Richter, K., Ernst, R., Thiele, L., and Teich, J.: 'SPI – A system model for heterogeneously specified embedded systems', *IEEE Trans. Very Large Scale Integr. (VLSI) Syt.*, 2002, **10**, (4), pp. 379–389

25 Jersak, M., Henia, R., and Ernst, R.: 'Context-aware performance analysis for efficient embedded system design'. Proc. Design Automation and Test in Europe, Paris, France, Mar. 2004

26 Tindell, K.W.: 'Adding time-offsets to schedulability analysis'. Technical Report YCS 221, Univ. of York, UK, 1994

27 Palencia, J.C., and Harbour, M.G.: 'Schedulability analysis for tasks with static and dynamic offsets'. Proc. 19th IEEE Real-Time Systems Symp. (RTSS98), Madrid, Spain, 1998

28 Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., and Deb, K.: 'Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimisation problem'. Parallel Problem Solving from Nature, PPSN VII, 2002

29 Zitzler, E., Laumanns, M., and Thiele, L.: 'SPEA2: Improving the strength Pareto evolutionary algorithm'. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001

30 Bleuler, S., Laumanns, M., Thiele, L., and Zitzler, E.: 'PISA – a platform and programming language independent interface for search algorithms'. http://www.tik.ee.ethz.ch/pisa/

31 Davis, L.: 'Applying adaptive algorithms to epistatic domains'. Proc. 9th IJCAI, Los Angeles, CA, 1985, pp. 162–164

32 Wolpert, D.H., and Macready, W.G.: 'No free lunch theorems for optimisation', *IEEE Trans. Evol. Comput.*, 1997, **1**, (1), pp. 67–82

33 Lehoczky, J., Sha, L., and Ding, Y.: 'The rate monotonic scheduling algorithm: Exact characterization and average case behavior'. Proc. Real-Time Systems Symp., 1989, pp. 201–209

34 Vestal, S.: 'Fixed-priority sensitivity analysis for linear compute time models', *IEEE Trans. Softw. Eng.*, 1994, **20**, (4), pp. 308–317

35 Punnekkat, S., Davis, R., and Burns, A.: 'Sensitivity analysis of real-time task sets'. Proc. 3rd Asian Computing Science Conf., ASIAN'97, 1997, 72–82

36 Audsley, N.C., Burns, A., Richardson, M.F., Tindell, K., and Wellings, A.J.: 'Applying new scheduling theory to static priority preemptive scheduling', *Softw. Eng. J.*, 1993, **8**, (5), pp. 284–292

166

IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005