A General Framework for Analysing System Properties in Platform-Based Embedded System Designs

Samarjit Chakraborty Simon Künzli Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH) Zürich
E-mail: {samarjit, kuenzli, thiele}@tik.ee.ethz.ch

Abstract

We present a framework for analysing various system properties pertaining to timing analysis, loads on various components and on-chip buffer memory requirements of heterogeneous platform-based architectures, in a single coherent way. Many previous analysis techniques from the real-time systems domain, which are based on standard event models, turn out to be special cases of our framework. We illustrate this using various realistic examples.

1 Introduction

The complexity of today's systems-on-chip (SoC) designs, coupled with issues like short time-to-market and low cost, have led to new design paradigms such as platform-based design [7]. These are based on the concept of *reuse* at several levels of abstraction, where designers rely on the use of intellectual property blocks or cores from some library (such as the IBM Blue Logic Core Library [5]), or on cores provided by a third-party vendor. Since such cores are already predesigned and verified, a designer can now concentrate on the overall system rather than the individual components, and also reduce the number of steps required to translate a system-level design into a final product.

In practice, these goals are however far from being realised. Assembling a hardware platform itself from a collection of cores remains an error prone and time consuming process [1]. For example, the interfaces and electrical characteristics of the different cores are different and hence integrating them might require additional interfaces or *adapters*. Analysing such system platforms to verify timing and other system properties pose a major challenge because they depend on the interfaces and properties (such as arbitration schemes on buses) of the different cores, and also on the RTOS and other components of the software platform. The problem gets aggravated in the context of embedded systems because of their generally heterogeneous architecture, where different scheduling and resource sharing strategies are used on the different buses and processors.

After the major architectural decisions have been made, the first step in a platform-based design requires the facility to integrate the cores into a high-level design, take into account the influence of the software platform, and then analyse such a design. Such an analysis involves verifying timing properties, identifying possible bottlenecks that might exist at a bus or a processor, and also estimate values of on-chip memory requirement, off-chip memory bandwidth, etc. However, currently there are almost no tools or methods which enable this in an easy and efficient manner. All the existing approaches rely on simulation (for example VCC [14] and Seamless [11]), and hence suffer from the problems of high running time, incomplete coverage and failure to identify corner cases. The last two problems are aggravated by the fact that in many cases system integrators do not have a full understanding of the functionality and the interfaces of the different cores, but only understand their high-level input/output behaviour. Therefore, if guarantees on system properties are required then some form of static formal analysis is inevitable.

Most of the work on the formal analysis of such systems exists in a disjoint form and do not offer a single unified framework for analysing system-level designs, especially in the presence of heterogeneity. It is only recently that some work in this direction is being done—for example, [8] analyses response times for static-priority process scheduling combined with TDMA bus protocol. But the goal of a general approach to analysing different system properties (including timing behaviour) of an arbitrarily complex and heterogeneous platform architecture still remains elusive.

Our contributions and relation to previous work. To address this problem, a general approach to timing analysis for heterogeneous systems was recently presented in [9] and [10]. It is based on identifying architectural components for which analysis methods are already known in the literature, and then combining these to obtain a compositional description of the complex system-level timing behaviour. The main contribution of this work is a method to adapt outgoing event streams from one component to match the input event model of another component which is required to process this outgoing stream. This gives a means for formally composing different architectural components and reasoning about the behaviour of the entire system.

The main drawback of this approach is that it can only accommodate standard event models like purely periodic, periodic with jitter, periodic with bursts, and sporadic. In practice, the event streams involved in a system usually do not conform to any of these standard models. But while analysing such systems, these streams are approximated by some standard model which minimizes the error. This introduces several modeling complexities, and when worst case bounds for a system are required, such approximations using standard event models give overly conservative bounds.

The analysis framework that we present in this paper is based on an event model which can accurately capture the characteristics of any arbitrary event stream. Given the trace of an event stream, it is possible to extract a number of parameters which represent the abstract timing characteristics of the stream in our model. We show that this framework can be used to analyse any arbitrarily complex and heterogeneous platform architecture and answer questions related to timing properties, on-chip memory requirements, and the load on different architectural components in a single coherent manner. Further, the results obtained by applying different scheduling strategies such as static priority, proportional share, time division multiplexing, and earliest deadline first on standard event models like periodic, periodic with jitter, sporadic, etc, turn out to be special cases of the results that can be obtained in our framework. The work in [9] and [10] is based on composing different *analysis domains* where each analysis domain is restricted to only standard event models. Our work extends and generalizes the concepts presented in [9, 10] and is not only restricted to timing analysis, but can also address other system properties in an uniform way.

The underlying theory behind our framework was originally developed in the context of performance evaluation of network processor architectures [12, 13]. However, there were two major shortcomings of the work presented in [12] and [13]: (i) it was not shown how the framework compares with the theoretical results from the real-time systems area, (ii) how closely do the performance evaluation results match those obtained by simulation. As already pointed out in [15], without a clarification concerning the above two issues, the applicability of the framework can not be fully established. In relation to the results in [12, 13], the work in this paper addresses the issue (i). Firstly, it shows that the framework is not only restricted to analysing network processor architectures, but is applicable to the more general domain of heterogeneous embedded system designs. Secondly, as already mentioned, it shows that many of the results based on standard event models from the real-time systems area can also be obtained within our framework. The second issue mentioned above, i.e. (ii), is addressed in detail in [3] and [4]. Although the basic framework presented here is the same as in [12, 13], the mathematical bounds used to deduce timing properties of event streams being processed by an architecture, or those used to compute the loads on various architectural components are tighter compared to our previous results. Further, these bounds pertain to event streams which span over $t = -\infty$ to $t = +\infty$ and therefore accurately capture event models like periodic, sporadic, etc., which do not have any specific starting time. The results in [12, 13] are concerned with event streams that have a fixed starting time (say $t = 0$) and are therefore appropriate for reasoning about event traces that arise in practical applications. But they can not accurately model an event stream which is specified, for example, as a stream with period equal to p and exsiting from $t = -\infty$ to $t = +\infty$.

The next section explains the basics of our event model, following which in Section 3 we describe the framework based on this model, and show how it can be used to analyse different properties of a system-level architecture. Finally, in Section 4 we present two examples using which we illustrate how the results from this framework generalize those that can be obtained from the traditional real-time systems area.

2 Event Models and Resource Capabilities

In this section we describe the underlying event model which forms the basis of our framework and also a means of modeling the processing capability of resources which process incoming event streams. It may be noted that in contrast to previous work on the performance evaluation of distributed embedded systems which relies on statistical bounds (see for example [6]) we are interested in analysing architectures and deriving worst case bounds on system properties like response times, on-chip memory requirements and loads on various components. We also show how standard event models such as periodic or periodic with jitter can be represented by our event model.

Event models. For a given event stream, let $R(t)$ denote the number of events that arrive in the time interval $[0, t]$. Further, assume that the number of events arriving within any interval of time is bounded above by a right-continuous subadditive function called the *upper arrival curve*, denoted by α^u. Similarly, a lower bound on the number of events arriving is given by a *lower arrival curve* α^l. R, α^u and α^l are related by the following inequality:

$$\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s),\ \forall 0 \leq s \leq t$$

Therefore, $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ can be interpreted as the minimum and maximum number of events arriving within *any* time interval of length Δ, respectively. Any standard event model can be represented in our model by an appropriate choice of α^l and α^u. For example, a periodic event stream with period p can be represented by an α^l and α^u, both of which are staircase functions of step width p and

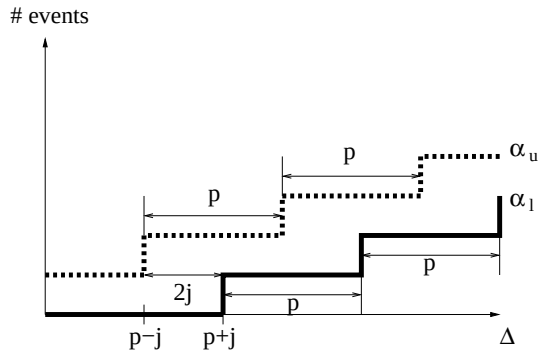

Figure 1. The upper and lower arrival curves of the class of event streams with period p and jitter j.

height 1, with $\alpha^l(t) = 0$ for all $t < p$ and $\alpha^u(t) = 1$ for all $t < p$. This is because within any time interval of length less than p, the minimum number of events that can be seen is zero, and within any time interval of length p^+, the minimum number of events that can be seen is equal to one. Similarly, the maximum number of events that can be seen within any time interval of length p and p^+ is one and two respectively.

Following the same reasoning, the class of event streams with period p and jitter j can be represented by an upper and a lower arrival curve of the form shown in Figure 1. Given any particular instance of such a periodic with jitter event stream, the corresponding upper and lower arrival curves would lie within the arrival curves shown in Figure 1, and therefore these curves represent the upper and lower bounds on the maximum and minimum number of events that can arrive within any time interval for any event stream with period p and jitter j. Alternatively, given the upper and the lower arrival curves of the class of event streams *periodic with jitter*, then it is possible to uniquely determine the period and the jitter values. Note that in Figure 1, if $j = 0$ then the upper and the lower arrival curves coincide and represent a purely periodic event stream with period p. Formally, these results can be stated in the form of the following theorem.

Theorem 1 *The upper and the lower arrival curves representing the entire class of event streams with period p and jitter j are unique.*

Similar representations of standard (abstract) event models like sporadic, or periodic with bursts, can also be given in terms of the upper and the lower arrival curves. At the same time, given any finite length arbitrary event trace and a real number Δ, it is possible to determine the values of $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ corresponding to the event trace, by sliding a window of length Δ over the trace and recording the minimum and maximum number of events lying within the window respectively. The upper and the lower arrival curves corresponding to the trace can be determined by following this procedure for different values of Δ.

Processing capability. Similar to the upper and lower arrival curves, we use β^u and β^l to denote upper and lower *service curves* of a resource with the following interpretation. If $C(t)$ denotes the number of processing units (might be in terms of processor cycles, time units, etc) available from the resource over the time interval $[0, t]$, then the following inequality holds. $\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s),\ \forall 0 \leq s \leq t$. Hence, $\beta^u(\Delta)$ and $\beta^l(\Delta)$ give an upper and lower bound on the resource capability over any time interval of length Δ.

3 Analysing System Properties

3.1 For a Single Resource

The Single Stream Case. An event stream entering a resource (such as a processor on which some processing function is implemented, or a communication resource such as a bus) gets processed (or transmitted in the case of a communication resource), thereby generating an outgoing event stream which might enter another resource for further processing. As a result, the processing capability (such as the processor or bus bandwidth) of the resource, as specified by its upper and lower service curves gets modified. In this section we formalize this notion and state the formulas for deriving the *outgoing arrival curves* and the *remaining service curves* from a specification of the incoming arrival curves and the original service curves.

Given an event stream which is specified by its arrival curves α^l and α^u and a resource which processes this event stream and its processing capability being specified by its service curves β^l and β^u. Let $\alpha^{l\prime}$ and $\alpha^{u\prime}$ denote the outgoing arrival curve of the (processed) event stream and $\beta^{l\prime}$ and $\beta^{u\prime}$ denote the remaining service curves of the resource. Then these curves are related by the following expressions.

$$\alpha^{l\prime}(\Delta) = \min\{\inf_{0\leq\mu\Delta}\{\sup_{\lambda>0}\{\alpha^l(\mu+\lambda) - \beta^u(\lambda)\} + \beta^l(\Delta-\mu)\}, \beta^l(\Delta)\} \quad (1)$$

$$\alpha^{u\prime}(\Delta) = \min\{\sup_{\lambda>0}\{\inf_{0\leq\mu<\lambda+\Delta}\{\alpha^u(\mu) + \beta^u(\lambda+\Delta-\mu)\} - \beta^l(\lambda)\}, \beta^u(\Delta)\} \quad (2)$$

$$\beta^{l\prime}(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{\beta^l(\lambda) - \alpha^u(\lambda)\} \quad (3)$$

$$\beta^{u\prime}(\Delta) = \min\{\inf_{\lambda>\Delta}\{\beta^u(\lambda) - \alpha^l(\lambda)\}, 0\} \quad (4)$$

These results are based on generalizing ideas from network calculus as applied to the domain of communication networks (see [2] for details), and hold specifically for infinite event streams that span over time $t = -\infty$ to $t = +\infty$. Therefore, these are suited for modeling event streams such as periodic, sporadic, etc., which do not have any specific starting time. For modeling finite length event traces, the relations used in [12, 13] may be used.

Processing Multiple Streams. When multiple event streams enter a resource, the processing capability of the resource is shared between these streams according to some scheduling strategy. The characteristics of each of the outgoing streams and the remaining processing capability of

the resource would depend on the scheduling strategy used. Here we derive these formulas for the case of static priority scheduling. For examples of other scheduling policies such as proportional share, the reader is refered to [12, 13].

Let us assume that there are n event streams entering a resource whose processing capability is bounded by the service curves β^l and β^u. Each event stream i is constrained by the arrival curves $\bar{\alpha}_i^l$ and $\bar{\alpha}_i^u$ and let the streams be ordered according to their priorities, i.e. stream 1 has the highest priority and stream n the lowest. For each event stream i, let w_i be its per event processing requirement on the resource. From now on, we will assume w_i to be defined in time units, i.e. the resource takes w_i time units to process each event of the stream i. To take these different processing requirements into account, we scale $\bar{\alpha}_i^l$ and $\bar{\alpha}_i^u$ appropriately before using Equations (1–4). Hence we have,

$$\alpha_i^u = w_i \bar{\alpha}_i^u, \qquad \alpha_i^l = w_i \bar{\alpha}_i^l, \qquad i = 1, \ldots, n \tag{5}$$

Similarly, each outgoing processed event stream has to be scaled back as follows:

$$\overline{\alpha}_i^{u\prime} = \lceil \alpha_i^{u\prime} / w_i \rceil, \qquad \overline{\alpha}_i^{l\prime} = \lfloor \alpha_i^{l\prime} / w_i \rfloor, \quad i = 1, \ldots, n \tag{6}$$

In the case of static priority scheduling, the resource processes the event streams in the order of decreasing priority, and the resulting arrival and service curves are computed using Equations (1–4). For the event stream 1, the service curves of the unloaded resource serve as an input. For the ith event stream, the input service curve is equal to the remaining service curve after processing the $(i-1)$th stream, for $i \geq 2$. This can be formally stated as follows: $\beta_1^u = \beta^u$, $\beta_1^l = \beta^l$, $\beta_i^u = \beta_{i-1}^{u'}$, $\beta_i^l = \beta_{i-1}^{l'}$, $i = 2, \ldots, n$, where $\beta_{i-1}^{u'}$ and $\beta_{i-1}^{l'}$ for $i = 2, \ldots, n$ are determined from β_{i-1}^u, β_{i-1}^l, α_{i-1}^u and α_{i-1}^l by applying Equations (3) and (4). Lastly, the remaining service curve after processing all the event streams is given as follows: $\beta^{u\prime} = \beta_n^{u\prime}$, $\beta^{l\prime} = \beta_n^{l\prime}$. This can be used to process other event streams, possibly using a different scheduling discipline, in a hierarchical manner.

3.2 For Multiple Resources

Our view of a platform architecture with multiple resources is the following. Event streams flow through a network of resources based on the order in which they need to be processed. In this process, the arrival curves associated with the event streams get modified in accordance with the Equations (1) and (2). The arrival curves of an outgoing event stream from a resource serve as input arrival curves to another resource. Properties of the event streams like periodicity, jitter, bursts, etc change as the stream flows from one resource to the next, and these are captured in the arrival curves. Similarly, the processing capability of a resource also changes as it processes an incoming event stream. The remaining processing capability, as captured in the upper and lower service curves is used to process other event streams.

This model of an architecture can be represented as a *scheduling network*. The nodes of this network represent event processing functions that are implemented on the various resources. The inputs to each such node are the arrival curves of an event stream that is to be processed, and the service curve of the resource, representing the processing capability available to the function that is being implemented on the resource. The outputs describe the resulting arrival curves of the processed event streams and the remaining service curves of the (partially) used resource. These arrival and service curves then serve as inputs to other nodes of the scheduling network. Note that "resources" in our framework refer to both communication (such as buses) and computation (such as processors) resources. The exact construction of the scheduling network for an architecture depends on the scheduling policies on the different architectural components, an example of which is shown in the next section.

Given a scheduling network corresponding to an architecture, it is possible to determine the timing properties (such as jitter and burst lengths) of the processed event streams from their outgoing arrival curves. Further, it is also possible to determine properties of the architecture such as the on-chip memory requirement and the loads on the different components such as processors and buses.

Let α^l and α^u be the lower and upper arrival curves of an event stream entering a node of a scheduling network whose input service curves are given by β^l and β^u. Then the maximum delay (or response time) experienced by an event at the resource represented by the service curves, and the maximum number of backlogged events from the stream that waiting to be processed can be given by the following inequalities.

$$delay \leq \sup_{t \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(t) \leq \beta^l(t+\tau)\} \right\} \tag{7}$$

$$backlog \leq \sup_{t \geq 0} \{\alpha^u(t) - \beta^l(t)\} \tag{8}$$

For a physical interpretation of these inequalities, we refer the reader to [2]. From inequalities (7) and (8), it is possible to compute the overall response time and backlog of an event stream by summing its delay and backlog values at the different nodes (of the scheduling network) through which the stream passes. Lastly, if β^u and $\beta^{l\prime}$ are the initial upper service curve and the final lower (remaining) service curves of a resource, then its maximum utilization can be given by: $utilization = \lim_{\Delta \to \infty} \frac{\beta^u(\Delta) - \beta^{l\prime}(\Delta)}{\beta^u(\Delta)}$. This can, for example, be used to identify potential bottlenecks that exist in a platform architecture.

4 Generalizing Standard Event Models

We now give two examples to show that in the case of heterogeneous system architectures, results from classical scheduling theory, that can be used to analyse standard

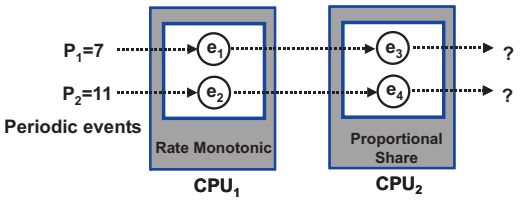

Figure 2. The system described in Example 2.

event models (like periodic, sporadic, etc.), can also be derived within our framework. The work in [10] considered a number of examples of heterogeneous platform architectures involving standard event models and different scheduling strategies and answered various questions related to timing analysis using a compositional approach. Our examples show that the same questions can also be answered using our framework.

Example 1 *Consider a purely periodic event stream entering a resource which requires a maximum of $e_{\max}$ and a minimum of $e_{\min}$ time units to process an event. The outgoing (processed) event stream is still periodic, but has a jitter equal to $e_{\max} - e_{\min}$.*

Let $R(t)$ denote the number of events that arrive at the resource over the time interval $[0, t]$ and $R'(t)$ denote the number of processed events that can be seen during the same interval. Then $R'(t) \leq R(t - e_{\min})$ and $R'(t) \geq R(t - e_{\max})$. Let the input event stream be constrained by the upper and lower arrival curves α^u and α^l respectively, and t and s be any time instants with $t \geq s$. Then, $R'(t) - R'(s) \leq R(t - e_{\min}) - R(s - e_{\max}) \leq \alpha^u((t-s) + e_{\max} - e_{\min})$. Similarly, $R'(t) - R'(s) \geq R(t-e_{\max}) - R(s-e_{\min}) \geq \alpha^l((t-s) - (e_{\max} - e_{\min}))$. Hence, the number of events that can be seen at the output within any time interval of length Δ is greater than or equal to the number of events that can be seen at the input over any time interval of length $\Delta - (e_{\max} - e_{\min})$, and is less than or equal to the number of events that can be seen at the input within any time interval of length $\Delta + (e_{\max} - e_{\min})$. This implies that the jitter of the output event stream increases by $(e_{\max} - e_{\min})$ over the jitter of the input event stream. If the input stream is purely periodic with a period p, then the output stream is periodic with period p and jitter equal to $(e_{\max} - e_{\min})$.

Example 2 *A system consists of two processors CPU_1 and CPU_2, on each of which two processes are implemented, as shown in Figure 2. Two purely periodic event streams 1 and 2, with periods $p_1 = 7$ and $p_2 = 11$ respectively are processed by the two processes implemented on CPU_1. The per event processing time for both the event streams is equal to 2. CPU_1 schedules the two processes processing streams 1 and 2 according to rate monotonic scheduling, and therefore stream 1 has higher priority over stream 2. The two outgoing, processed event streams are then processed by the two processes implemented on CPU_2, where the per event processing time is again equal to 2. CPU_2 implements proportional share scheduling and gives equal processor share to both the processes. Both CPU_1 and CPU_2 implement preemptive scheduling. What are the characteristics of the two processed event streams coming out of CPU_2?*

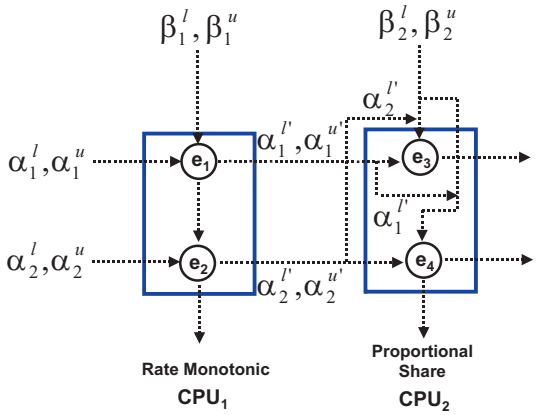

Figure 3. The scheduling network for the system described in Example 2.

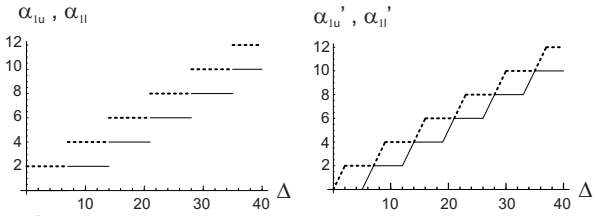

Figure 4. The upper and lower arrival curves of the incoming event stream 1 and the arrival curves of the processed stream coming out of CPU_1 (dotted line show the upper curve and the solid line shows the lower curve).

We use the arrival curves of the input event streams entering CPU_1 and from them compute the arrival curves of the final processed event streams coming out of CPU_2. These are then used to deduce the timing behaviour of the processed event streams. Figure 3 shows the scheduling network corresponding to the system. The entire processing capability of CPU_1 is available to stream 1 since this has the higher priority. This is represented by $\beta^u = \beta^l$, both being straight lines of slope 1 passing through the origin.

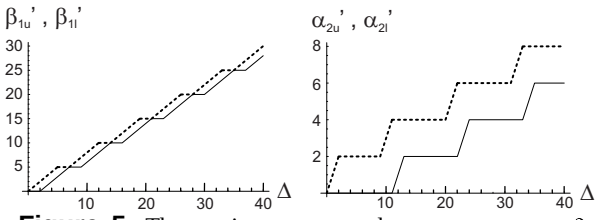

Figure 5. The service curves used to process stream 2 and the arrival curves of the processed stream coming out of CPU_1.

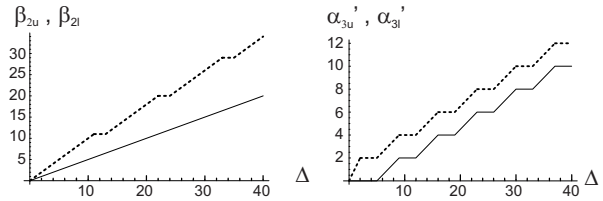

Figure 6. The service curves used to process stream 2 coming out of CPU_1 (indicated as stream 3) and the arrival curves of the processed stream (by CPU_2).

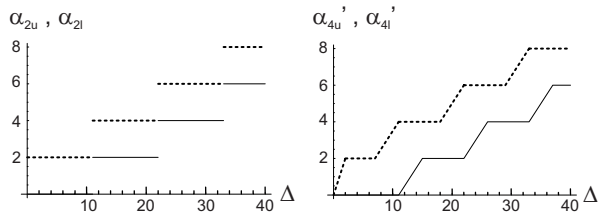

Figure 7. The arrival curves of the incoming event stream 2, and those of the finally processed stream coming out of CPU_2 (i.e. after being processed at CPU_1 and CPU_2).

Figure 4 shows the arrival curves of the stream 1 and those of the processed stream. As described in Section 2, note that the processed stream is still periodic with period 7. In Figure 4, the arrival curves of the input event stream represent the discrete stream, but since the Equations (1–4) hold for continuous streams, to interpret the right hand figure in Figure 4 as a discrete stream, a *floor* function should be applied to the lower curve and a *ceiling* function to the upper curve.

The remaining processing capability of CPU_1 that is available to stream 2 can be obtained by using Equations (3) and (4). These resulting service curves and the arrival curves of the processed stream are shown in Figure 5. As can be seen from this figure, the processed stream is still periodic with period 11 and but now has a jitter equal to 2.

In the case of CPU_2, because of the proportional share scheduling, both the streams are guaranteed a minimum service represented by a lower service curve, which is a straight line of slope 0.5 passing through the origin. The upper service curve for stream 3 (i.e. the processed stream 2) is equal to $\beta^u(\Delta) = \inf_{\lambda>\Delta}\{\beta(\lambda) - \alpha_4^l(\lambda)\}$, where β is the total unloaded processor capacity represented by a straight line of slope 1 passing through the origin, and α_4^l (which is equal to $\alpha_2^{l\prime}$)is the lower arrival curve of stream 4 (i.e. the processed stream 2 from CPU_1). The upper service curve available to stream 4 can be similarly computed from β and α_3^l (which is equal to $\alpha_1^{l\prime}$). Based on these service curves, the arrival curves of the processed streams 3 and 4 (by CPU_2) are given in (the right hand of) Figures 6 and 7. From these curves, it may be deduced that the processed stream 3 is periodic with period 7 and jitter 2 and stream 4 is periodic with period 11 and jitter equal to 4.

5 Conclusions

The framework presented in this paper allows for a formal analysis of different system properties in heterogeneous platform-based designs. Specifically, it extends the recent work presented in [9] and [10], and provides a single coherent way of deducing many results that can be derived using different event models and scheduling theoretic results from the domain of real-time systems.

References

[1] R. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz, M. Muhlada, W. Lee, F. White, and J.-M. Daveau. Automating the design of SoCs using cores. *IEEE Design & Test of Computers*, 18(5):32–45, 2001.

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

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

[4] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Comparing analytical modeling with simulation for network processors: A case study. In *DATE*, Munich, 2003.

[5] Blue Logic technology and CoreConnect bus architecture, IBM. `http://www.chips.ibm.com/bluelogic/`.

[6] A. Kalavade and P. Moghé. A tool for performance estimation of networked embedded end-systems. In *35th DAC*, 1998.

[7] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonolization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design*, 19(12), 2000.

[8] P. Pop, P. Eles, and Z. Peng. Bus access optimization for distributed embedded systems based on schedulability analysis. In *DATE*, 2000.

[9] K. Richter and R. Ernst. Model interfaces for heterogeneous system analysis. In *DATE*, 2002.

[10] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *39th DAC*, 2002.

[11] Seamless Hardware/Software Co-Verification, Mentor Graphics. `http://www.mentor.com/seamless/`.

[12] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration of network processor architectures. In *Network Processor Design: Issues and Practices, Volume 1*. Morgan Kaufmann Publishers, October 2002.

[13] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *39th DAC*, New Orleans, 2002.

[14] The Cadence Virtual Component Co-design (VCC). `http://www.cadence.com/products/vcc.html`.

[15] T. Wolf. *Design and Performance of a Scalable High-Performance Programmable Router*. PhD thesis, Department of Computer Science, Washington University in St. Louis, May 2002.