Paper Title

A Formal Approach to MpSoC Performance Verification

Authors

Kai Richter, IEEE Member, Marek Jersak, IEEE Member, and Rolf Ernst, IEEE Fellow
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
Braunschweig, Germany

Keywords

MpSoC design, system integration, performance verification, formal scheduling analysis

Abstract

Communication-centric Multiprocessor Systems-on-Chip (MpSoC) will dominate future chip architectures. They will be built from heterogeneous HW and SW components integrated around a complex communication infrastructure. Already today, performance verification is a major challenge that simulation can hardly meet, and formal techniques have emerged as a serious alternative. The article presents a new technology that extends known approaches to real-time system analysis to heterogeneous MpSoC using event model interfaces and a novel event flow mechanism.

Biography

Kai Richter is a research staff member and a PhD candidate in the team of Rolf Ernst at the Technical University of Braunschweig, Germany. His research interests include real-time systems, performance analysis, and heterogeneous HW/SW platforms. He recieved a Diploma (Dipl.-Ing.) in electrical engineering from the University of Braunschweig, Germany. Contact him at kair@ida.ing.tu-bs.de.

Marek Jersak is a research staff member and a PhD candidate in the team of Rolf Ernst at the Technical University of Braunschweig, Germany. His research interests include real-time embedded systems, multi-language design, and model transformations. He recieved a Diploma (Dipl.-Ing.) in electrical engineering from the Aachen Institute of Technology, Germany. Contact him at marek@ida.ing.tu-bs.de.

Rolf Ernst is a full professor at the Technical University of Braunschweig, Germany, where he heads the Institute of Computer and Communication Network Engineering. His main research interests are embedded system design and embedded system design automation. He recieved a PhD in electrical engineering from the University of Erlangen-Nürnberg, Germany. Contact him at ernst@ida.ing.tu-bs.de.

Coordinating Author - Contact Person

Kai Richter
IDA - Institut fuer Datentechnik und Kommunikationsnetze (**IDA** - Computer and Network Engineering)
Technical University of Braunschweig
Hans-Sommer-Strasse 66
D-38106 Braunschweig
Germany

Phone: +49 (0) 531 - 391 - 3724
Fax: +49 (0) 531 - 391 - 4587
Mailto: kai.richter@tu-bs.de

A Formal Approach to MpSoC Performance Verification

Kai Richter, Marek Jersak, Rolf Ernst
Technical University of Braunschweig,
Braunschweig, Germany

MpSoC - The Architecture of Choice

Multiprocessor Systems-on-chip (MpSoC) integrate multiple programmable microcontrollers and digital signal processors (DSPs) with specialized memories and complex intellectual property (IP) components on a single chip using complex on-chip networks. Such heterogeneous MpSoCs have become the architecture of choice in major industries such as network processing, consumer electronics, or automotive systems. The ever increasing heterogeneity is an inevitable result of component specialization and IP integration. Component specialization and optimization are needed to achieve the required performance at low power consumption and acceptable cost.

Fig. 1 shows the Viper processor for multimedia applications [DJR01] which is based on the Philips Nexperia platform [Nex02]. Many of the key components are either reused or supplied externally, such as the MIPS (www.mips.com) and TriMedia (www.trimedia.com) processor cores. Tomorrows MpSoCs will be even more complex, and using such IP library elements in a "cut&paste"-like design style is the only way to reach the necessary design productivity.

Systems integration is becoming the major challenge in MpSoC design. The complexity of HW and SW component interactions is a serious thread to all kinds of performance pitfalls including transient overloads, memory overflow, data loss, and missed deadlines. According to the latest ITRS roadmap [ITRS01], system-level performance verification is among the top three co-design issues.

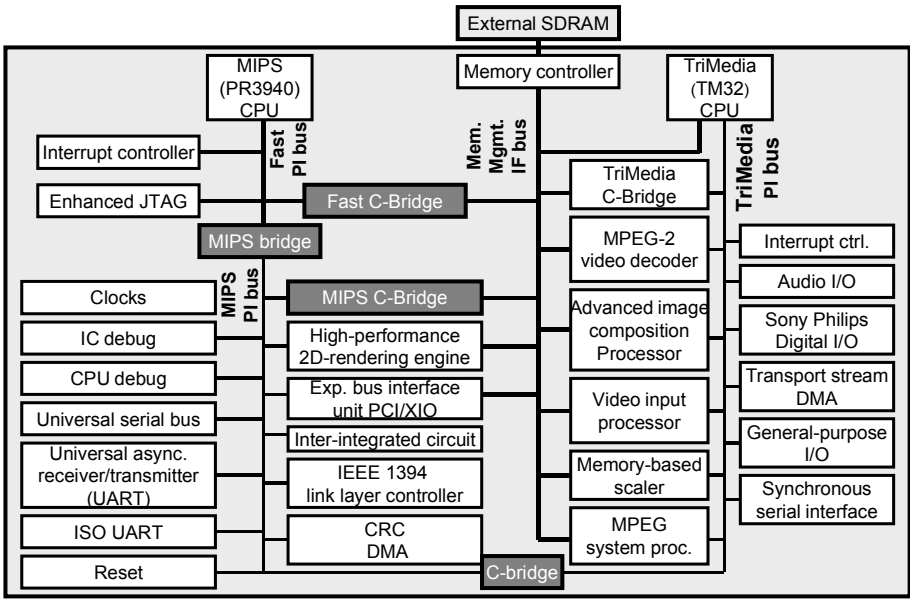

Figure 1 The Viper Processor combines a MIPS RISC processor, a TriMedia TM32 VLIW DSP, weakly programmable co-processors, fixed function co-processors, and different types of memories and caches (omitted in the figure). A complex network of bridged high-speed and peripheral buses connects these components.

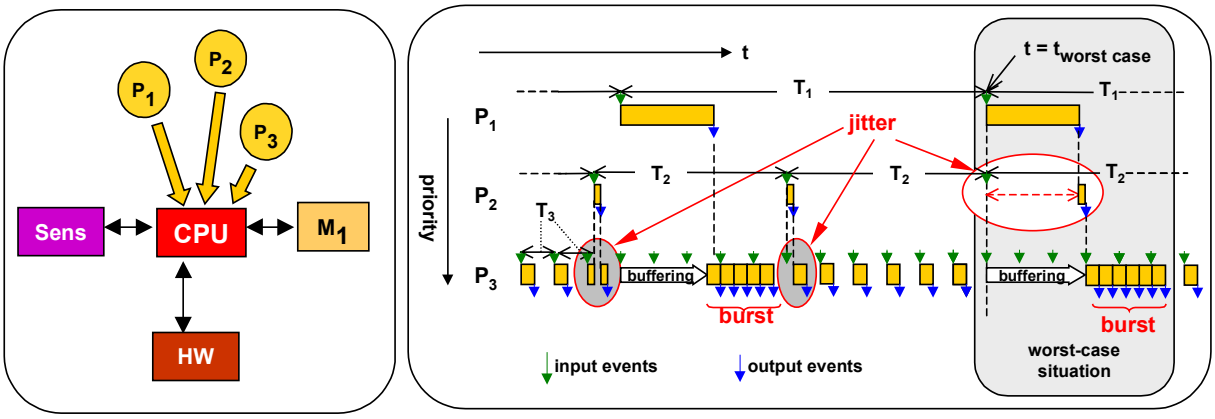

Figure 2 "Performance Corner Cases": subsystem containing three tasks which are activated periodically. Scheduling is preemptive and follows static priorities. The execution sequence is rather complex. Although activated periodically, P_3 (with its input buffers filled) temporarily runs in burst mode with an execution frequency which is only limited by the available processor performance. This burst execution will lead to a transient P_3 output burst which is modulated by P_1 execution. In the highlighted situation all processes request the CPU at the same time, resulting in longest possible, i.e. worst-case preemtion time for all lower priority processes.

Performance Simulation - Can It Get The Job Done?

Simulation is state of the art in MpSoC performance verification. Tools such as Mentor Seamless CVE (www.mentor.com/seamless) or Axys MaxSim (www.axysdesign.com/products/products_maxsim.asp) support hardware and software co-simulation of the complete system. The co-simulation times are extensive but as a major advantage performance verification and function verification can use the same simulation environment, simulation patterns, and benchmarks. Simulation patterns are use cases that support system understanding and debugging. However, there are critical, *conceptual* disadvantages of simulation-based performance verification.

HW and SW component integration includes resource sharing which is based on operating systems and network protocols. Resource sharing results in a confusing variety of *performance run-time dependencies*. Consider the system in Figure 2. Although activated periodically, the execution sequence of the three processes on the CPU is quite complex leading to output bursts. The figure does not even include data dependent process execution times. Finding simulation patterns leading to the high-lighted worst-case situation is already challenging. Network arbitration (Figure 3) introduces an additional performance dependency (indicated by the green arrows) between the subsystems which is not reflected in the system

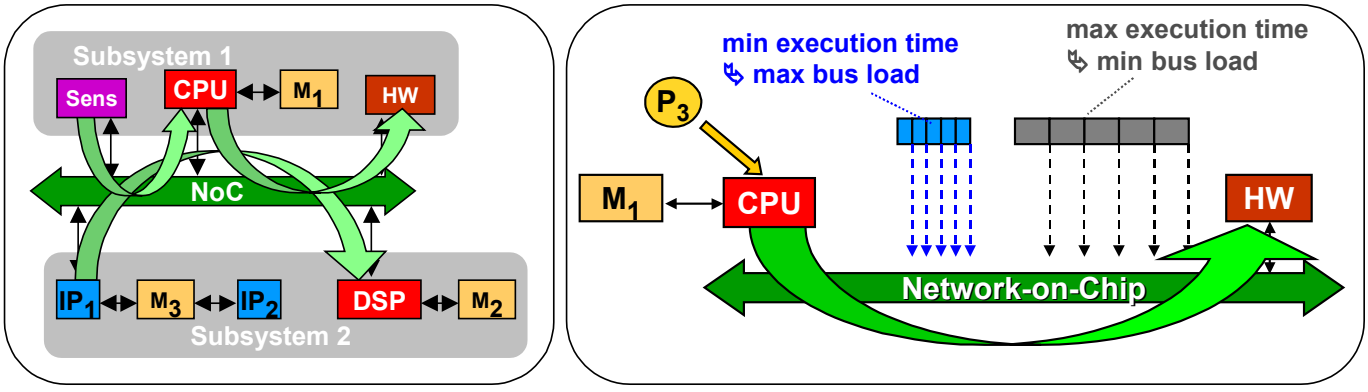

Figure 3 "Scheduling Anomaly": P_3 is a driver process reading data from M_1 and -during bursts- iterating at maximum speed. Furthermore, P_3 has a non-constant execution time, e.g. due to input data dependencies. The shorter the process execution time, the shorter the distance of packets to be sent over the bus and the higher the transient bus load. So, the minimum execution time corresponds to the maximum transient bus load slowing down communication of other components.

function. This dependency can be sophisticated turning component or subsystem "best case" performance into system "worst case" performance - a so called scheduling anomaly. Such transient run-time effects leading to complex *system-level corner cases* are extremely difficult to find and to debug.

Using abstract and scheduling-aware performance simulation tools such as Cadence VCC (www.cadence.com/products/vcc.html) provides rough estimates on system performance quickly but does not help in reliably covering system-level corner cases. Furthermore, VCC only uses *typical* execution times rather than considering intervals defined by the best-case and worst-case bounds.

System-Level Corner Cases - Who provides the simulation patterns?

Where do we get the stimuli from to cover *system-level corner cases*?

(a) Reusing *function verification patterns* is not sufficient since they do not cover the complex non-functional performance dependencies introduced by resource sharing.
(b) Reusing *component or subsystem corner cases* is not sufficient, either, since 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 where the component behavior is well understood. For complex MpSoC with layered SW architectures, dynamic bus protocols, and operating systems manual system-level corner case identification is no practical option. In conclusion, todays simulation-based approaches to MpSoC performance verification are running out of steam.

Industrial Consequences - What Do We Pay?

With embedded system design gradually moving from a core-centric SoC to a communication centric MpSoC design style, the demands on flexibility and reactivity of on-chip interconnect are continuously increasing. To meet these requirements at acceptable cost, sophisticated multi-hop dynamic communication network protocols have been proposed [BDM02] to be optimized using communication statistics. Interestingly in practice, there seems to be a second development in the opposite direction, towards conservative and less efficient communication protocols like TDMA (time division multiple access) where non-functional component dependencies -like those in Figure 3- are minimized and communication timing becomes straighforwardly predictable. An example is the Sonics SiliconBackplane Micronetwork (www.socinsinc.com). A similar development towards conservative protocols is found in distributed systems such as TTP (time-triggered procotol) in automotive and aerospace electronics. Such protocols enforce *static bus access* patterns, and the run-time behavior of each communication can be independently verified. This simplicity in integration, however, comes at a significant performance price which grows with system complexity. It increases buffer sizing requirements and, hence, also response times, and does not adapt to dynamically changing load situations which are typical for reactive embedded systems. It will therefore be highly difficult to scale such a conservative design style to future communication-centric MpSoC with complex network protocols.

Formal Techniques - A Promising Alternative

When simulation falls short, formal approaches become more attractive. Such approaches allow a systematic verification based on well-defined models. Other than simulation, formal analysis *guarantees full performance corner-case coverage* by its very nature and provides guaranteed bounds for critical performance parameters. The literature on formal performance or timing verification distinguishes two subproblems: (a) formal process or task performance analysis, usually in the form of *process running time analysis*, and (b) resource sharing analysis, also known as *scheduling or schedulability* analysis which is based on process running times. Most of the solutions for scheduling analysis are limited to *individual components* or *homogeneous subsystems*. Therefore, a major shortcoming of existing techniques is that they are currently not applicable to complex heterogeneous MpSoC as we will see after reviewing the existing work.

Process Running Time and Communication Analysis

There exist two roots for process running time analysis: in real-time system analysis for software processes, and in HW/SW co-design for rapid HW performance estimates. Process running time analysis basically includes (a) program path analysis to find out what is going to be executed, and (b) architecture modeling - inluding pipelines and caches- to determine the time spend for execution on this path. The enormous progress in this field over the last 10 years already leading to industrial large scale applications in the aircraft industry, e.g. the Absint tool (www.absint.com), can not be mentioned here. We summarize that process running time analysis provides conservative upper and lower bounds (intervals) for individual process running times as well as bounds for the communication between processes [Wol02]. These running time and communication behavior lay the foundation for the next analysis level, modeling of shared resources.

Scheduling Analysis

Scheduling analysis for processors and buses has been another focus of real-time systems research for decades, and there are many popular scheduling analysis techniques available, e.g. RMS (rate-monotonic scheduling) and EDF (earliest deadline first) using static and dynamic priorities [LiL73], time-slicing mechanisms like TDMA (time-division multiple access) or RR (round-robin) [JLT85], or static order scheduling [LeM87]. Since then, many extensions have been proposed, some found their way into commercial analysis tools such as TriPacific RAPID RMA (www.tripac.com/html/prod-fact-rrm.html), Livedevices Real-Time Architect (www.livedevices.com/realtime.shtml), and TimeSys TimeWiz (www.timesys.com/index.cfm?bdy=tools_bdy_model.cfm).

The techniques rely upon a simple yet very 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 sequences. Only few simple and intuitive key characteristics of event sequences such as an event period or a maximum jitter are needed. From these parameters, worst-case scheduling scenarios are *systematically* derived, and efficient algorithms allow to safely bound the worst-case process or communication *response times*.

The event stream model impressively displays the consequences of resource sharing, as Figure 2 shows. A periodic input event stream (task activation) is transformed (by the scheduling) into an event stream with burst at the components output. A larger system with more priority levels generates even more complex event sequences.

Complex Event Streams - Too Complex?

The example leads to an interesting observation. Due to complex run-time interdependencies, even systems with periodic process activation produce outputs which are no longer periodic. In case of the system in Figure 3, the output event stream of CPU which is no longer exactly periodic turns, i.e. is *propagated* into the input event stream of the network, where it experiences additional distortion due to communication scheduling on the shared network. In order to solve the system-level performance verification problem including send and receive buffer sizing, we could try to find suitable analysis techniques which can deal with these event streams *propagated* through the network of components. However, protocols and corresponding analysis techniques that can handle such complex input event streams efficiently are extremely rare and produce even more "distorted" output streams which -again- need to be fed into the connected receivers, and so on. It is obvious that very soon the event sequences in the corresponding merged communication streams become incredibly complex, and the existing scheduling and analysis techniques can not be applied. In other words, the complex event streams are not compatible with the existing analysis techniques.

In effect, global scheduling analysis of complex systems is currently not possible, since the known subsystem techniques can not be reasonably combined, mainly due to input-output event stream incompatibilities. There exist few "holistic" analysis approaches providing solutions also for special classes of distributed systems [TiC94, PEP02], but they are limited in their scope and scalability.

Recently, a different view of global scheduling analysis has been adopted. The individual components and subsystems are seen as entities that interact, i.e. communicate, via event streams. Schedulability analysis, then, becomes a *flow analysis problem* for event streams that can, in principle, be solved iteratively by event stream propagation -just as explained above. In order to tame the complexity of the event streams, one approach is to generalize the event model in an event vector system [Gre93] or with upper- and lower-bound arrival curves [TCN00]. The work in [TCN00] successfully applies event model approximation using arrival curves to network processor design. However, both approaches introduce a new event stream representation and, thus, require new scheduling analysis techniques for the local components.

However, we don't necessarily need to develop new local analysis techniques but can benefit from the host of work in real-time scheduling analysis. Even if input and output streams seem to be totally incompatible, we can see that the number of P_3's output events in Figuge 3 can be easily bounded over a larger time interval. The bursts only occur temporarily, representing a *transient* overload within a *generally periodic* event stream, i.e. some key characteristics of the original periodic stream remain even in the presence of heavy distortion.

In our work, we have developed a technology that allows us to (a) *extract* this key information from a given schedule, and to (b) automatically *interface* or *adapt* the event stream such that the existing subsystem analysis techniques can be safely applied.

Event Stream Interfacing - Taming Complexity

Figure 4(a) shows a relatively simple event stream interfacing scenario. It converts a periodic event stream with jitter into a sporadic stream (some analysis techniques need this), requiring only a minimum of math. The jitter events are characterized by a period (T_X) and a jitter (J_X). The jitter bounds the maximum time deviation of each event with respect to a "virtual" reference period. In other words, each event is allowed to be at most $J_X/2$ "earlier" or "later" than the reference. The required sporadic input event model has only one parameter, the minimum interarrival time ($t_{Y,min}$) between any two events in the stream, thus bounding the maximum transient event frequency. Now, imagine two successive events in the original stream, the first being as late as possible ($t + J_X/2$) and the second as early as possible ($t + T_X - J_X/2$). The minimum distance between two dedicated events in the output event stream is thus $t_{Y,min} = T_X - J_X$.

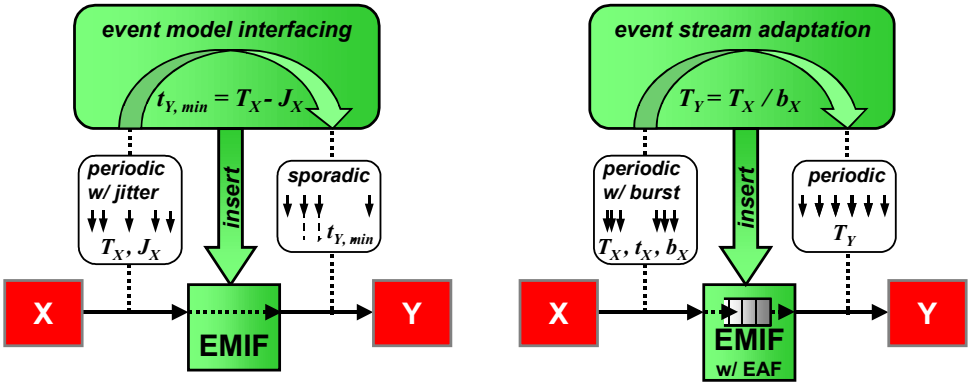

Figure 4 (a) and (b) illustrate the use of EMIFs and EAFs to interface periodic events with jitter into the sporadic event model, or to adapt periodic bursts into the purely periodic events, respectively.

With respect to the existing analysis techniques and practically important input stream characteristics, we can identifiy two *event model classes*:

- *Periodic event models* are particularly useful when the event timing shows a generally periodic behavior. Periodic signal sampling as typically found in signal processing and control applications is an example for purely periodic events. Periodic events with *jitter* originate from a purely periodic source but the timing has been distorted by process preemption or network congestion (as in the above example). Jitters that exceed the original period lead to *event bursts*.
- *Sporadic event models* define the second class of event models in most situations, where the source is not periodic, e.g. user generated service requests or management tasks which appear seemingly irregular. However, having no information at all would not allow any performance analysis. We need at least some information to bound the overall system load, e.g. the minimum interarrival time between any two events. Besides the basic model of sporadic events, there is the model of *sporadic events with bursts* allowing higher transient event frequencies.

Our new technology provides adaptation support for all possible event model combinations. The intuitive event flow principle using comprehensible yet rigorous event models and model adaptation support the integrator in controlling complex event streams in system integration. It gives tremendous help in understanding, analyzing, and optimizing the dynamic behavior of the complex component interactions. The interfacing and adaptation is based on establishing mathematical relations between the involved streams or models. For readings on the mathematical details of event models and their transformations, we refer to [RiE02]. For this article, we will focus on the practical impact of event stream adaptation on system design and analysis.

There are situations -as known from Fig. 4(a)- in which the event stream itself, i.e. the timing properties of the actual events remains unchanged while only the *mathematical representation*, i.e. the underlying event model is transformed. We refer to such transformations as *Event Model InterFaces* (EMIF). EMIF transformation requires that the given timing of any possible event sequence in one model can be directly captured by the other model's parameters. If such direct model transformation is not possible, then the actual timing of the events in the stream, i.e. the stream itself must be adapted. An example is a periodic stream with jitter or burst that must be adapted to a component that expects periodic events. This is performed by an *Event Adaptation Function* (EAF) that is automatically inserted in an EMIF to make the streams match. Practically speaking, EAFs correspond to buffers which are inserted at the component interface to make the system working *and* analyzable. Using EAFs, buffer sizing and buffering delay calculation is automatically performed during adaptation and is used in global system analysis.

Figure 4(b) shows the resynchronization of a periodic event stream with burst into a purely periodic stream. Again, the math is relatively simple. The sought-after parameter T_Y of the purely periodic stream is the average period of the bursty stream, and is given by $T_Y = T_X / b_X$. The burst event model [TiC94] captures a number of b_X events within a period of T_X.

Likewise, sophisticated interfaces and adaptations possibly requiring appropriate design elements are available for all other combinations of event models. Furthermore, the library of existing EMIFs and EAFs is easily extendible to cover other event models and streams potentially used by designers of sophisticated high-performance subsystems or necessary to integrate some complex IP components. These EMIFs and EAFs form the foundation for a novel and very promising system-level performance analysis procedure.

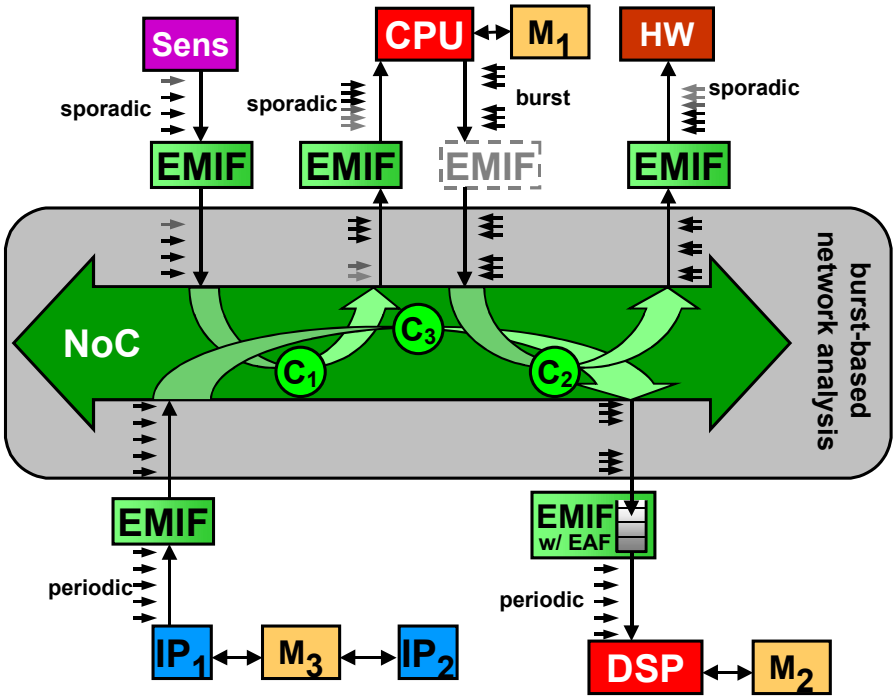

Figure 5 Event stream view of complex component interactions after integration. The merged network traffic on the logical channels C_1, C_2, and C_3 consists of periodic, bursty, and sporadic event streams. Event model interfaces at the network inputs map the key input stream characteristics into the burst model parameters, thereby enabling the application of a known network performance analysis from literature. The channels interfere, and the network output is distorted due to congestion and needs to be interfaced to the event models required by the individual components or subsystems. The periodic DSP functions additionally require adaptation (EAF) to run efficiently as planned.

Interface and Propagate!

We can now reliably verify the performance of the heterogeneous system from Figure 3. The Sensor (Sens) sporadically sends new data to P_1 on the CPU using the logical channel C_1, while P_3 sends bursts of requests through C_2 to the fixed function component HW. Simultaneously, the signal processing subsystem 2 is using the network. IP_1 periodically sends data over channel C_3 to the DSP which also implements a periodic scheduling to efficiently run a set of signal processing applications.

The network can implement any protocol for which an appropriate analysis technique is avaliable. In fact, this freedom really widens the design space, since the host of work on real-time analysis covers a variety of network protocols including very complex dynamic arbitration schemes. For this example, we assume that the input event streams to the network must comply with the model of periodic events with burst in order to use a known analysis technique. The required event model interfaces at the network inputs are illustrated in Figure 5. Only the burst stream from the CPU already meets the required model, the other two input streams require an EMIF. Next, the network is analyzed using the known technique, and the distorted output event streams are obtained. In other words, the input streams are *interfaced* and *propagated* through the nework analysis.

Finally, the output event streams are interfaced to the input models required by the individual receiver components. The DSP requires periodic input data to fit the given implementation of purely periodic scheduling. This situation is already known from Figure 4(b), an EMIF with an appropriate EAF is inserted at the network output.

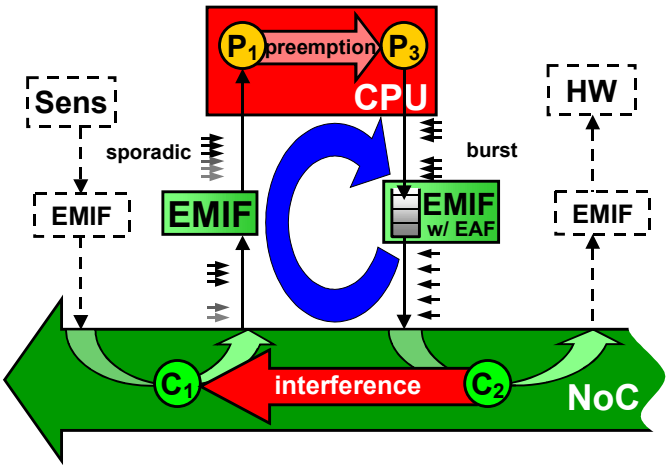

Figure 6 System integration can result in cyclic event stream dependencies which are not reflected in the system function. Appropriate buffering when inserting EMIFs breaks up such cycles. Putting the buffers in the right place significantly improves system performance and memory optimization.

The example shows that using event model interfacing in system integration gives tremendous relieve in system-level performance analysis when dealing with complex event stream dependencies spaning several heterogeneous components. The required event models are either constrained by the local component and subsystem analysis techniques -such as the network in our system-, or are given by a component's implementation itself -for instance, the periodic DSP schedule or the maximum frequency (sporadic event model) of the HW component.

Cyclic Event Stream Dependencies

As an additional performance pitfall in MpSoC design, system integration can introduce cyclic event stream dependencies which are not obvious and difficult to detect if component details are not considered. Figure 6 highlights a non-functional event stream dependency cycle in our system that is only introduced by communication sharing. Upon reception of new sensor data the CPU activates process P_1 which -due to preemption as in Fig. 2- influences the execution timing of process P_3. P_3's output, in turn, enters the network on channel C_2 where it *now interferes* with the arriving sensor data on channel C_1. It is the interference of the two functionally independent channels C_1 and C_2 that closes the dependency cycle, since the subsystem of Figure 2 was originally cycle-free.

Such cycles are analyzed by iterative propagation of event streams until the event stream parameters converge or until some deadline is missed or a buffer limit is exceeded. This iteration process terminates, since the event timing uncertainty (i.e. the best-case to worst-case event timing interval) grows monotonously with every iteration. For cases where there is no convergence, we have developed a mechanism that uses EAFs to break up the dependency cycle in order reduce the timing uncertainty to enforce convergence. Cyclic dependencies have been thoroughly investigated in [RZJ+02]. It should be noted that these event flow cycles are not an artifical result of global analysis but practically exist as we have seen in the example.

Application

So far, we have the methods and a simple tool for analysis interfacing and developed a library with analysis algorithms to configure a global analysis model. This is a big step towards global formal analysis but, at this time, there is still expert modeling knowledge needed. However, we have already applied the technology to three case studies together with an automotive, a multi-media, and a telecom industrial partner with each case having a very different focus. In the telecommunication project, we could resolve a severe system integration problem with transient faults which was not found even in prototyping. In the multi-media case study, we have modeled and analyzed a complex two-stage dynamic memory scheduler to

derive maximum response times for buffer sizing and priority assignment. Last but not least, we have shown how the technology enables a formal software certification procedure for automotive manufacturers.

Conclusion

Heterogeneous HW/SW component integration leads to increasing problems in MpSoC performance verification. We gave small examples demonstrating that seemingly simple systems can already lead to complex, non-intuitive subsystem interdependencies including cycles which are not reflected in the system function. Finding these by simulation is very time-consuming and increasingly critical. New performance verification solutions are urgently needed as future communication-centric MpSoC will create unprecedented integration complexity.

Starting with abstract event stream models that are used in real-time system scheduling analysis, we developed a global event flow interfacing technique which couples local scheduling analysis techniques into a coherent global performance analysis. The scheduling analysis techniques can be applied to programmable cores and their SW as well as to HW components, a requirement for any MpSoC verification. We consider it a serious alternative to performance simulation at far less computation time and higher result safety. First realistic applications demonstrate the power and the wide applicability of the approach.

At the current state, application of the global analysis technique still requires some expert knowledge to set up analysis and guide the analysis process, but we are working on an automated process using libraries of analysis techniques. Here, we can profit from the host of work in real-time systems analysis

References

[BDM02] L. Benini and G. DeMicheli. *Networks on Chips: A New SoC Paradigm*. IEEE Computer, Januar 2002.

[DJR01] S. Dutta, R. Jensen, and A. Rieckmann. *Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems*. IEEE Design&Test of Computers, Sep.-Oct. 2001, pp. 21-31.

[Gre93] K. Gresser. *An Event Model for Deadline Verification of Hard Real-Time Systems*. Proceedings of 5th Euromicro Workshop on Real-Time Systems, pp. 118-123, Oulu, Finnland, 1993.

[ITRS01] International Technology Roadmap for Semiconductors, 2001 Edition. http://public.itrs.net/Files/2001ITRS/ESH.pdf

[JLT85] E. Jensen, C. Locke, and H. Tokuda. *A Time-Driven Scheduling Model for Real-Time Operating Systems*. Proceedings of 6th IEEE Real-Time Systems Symposium (RTSS85), pp. 112-122, San Diego (CA), USA, 1985.

[LeM87] E. A. Lee and D.G. Messerschmitt. *Static scheduling of synchronous data flow programs for digital signal processing*. IEEE Transactions on Computers, vol. 36(1), pp. 24-35 , Jan. 1987.

[LiL73] C. L. Liu and J. W. Layland. *Scheduling algorithms for multiprogramming in a hard-real time environment*. Journal of the ACM, vol. 20(1), pp. 46–61, 1973.

[PEP02] P. Pop, P. Eles, and Z. Peng. *Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems*. Proceedings of International Symposium on Hardware/Software Codesign (CODES02), pp.187-192, Estes Park (CO), USA, 2002

[RiE02] K. Richter and R. Ernst, *Event model interfaces for heterogeneous system analysis*. Proceedings of Design, Automation and Test in Europe (DATE02) Conference, pp. 506-513, Paris, France, 2002

[RZJ+02] K. Richter , D. Ziegenbein, M. Jersak, and R. Ernst, *Model composition for scheduling analysis in platform design*. Proceedings of Design Automation Conference (DAC02), pp. 287-292, Paris, France, 2002

[TCN00] L. Thiele, S. Chakraborty, and M. Naedele. *Real-time Calculus For Scheduling Hard Real-Time Systems*. Proceedings of International Symposium on Circuits and Systems (ISCAS 2000), pp. 101-104, Geneva, Switzerland, vol. 4, March 2000.

[TiC94] K. Tindell and J. Clark. *Holistic schedulability for distributed hard real-time systems*. In Euromicro Journal, Vol. 40, pp. 117-134, 1994

[Wol02] F. Wolf. *Behavioral Intervals in Embedded Software*. Kluwer Academic Publisher, Boston, 2002