A service based component model for composing and exploring MPSoC platforms

Tranberg-Hansen, Anders Sejer; Madsen, Jan

Published in:
2008 First International Symposium on Applied Sciences on Biomedical and Communication Technologies

Link to article, DOI:
10.1109/ISABEL.2008.4712584

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Tranberg-Hansen, A. S., & Madsen, J. (2008). A service based component model for composing and exploring MPSoC platforms. In *2008 First International Symposium on Applied Sciences on Biomedical and Communication Technologies* (pp. 1-5). IEEE. https://doi.org/10.1109/ISABEL.2008.4712584

A Service Based Component Model for Composing and Exploring MPSoC Platforms

Anders Sejer Tranberg-Hansen and Jan Madsen
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark
e-mail: {asth, jan}@imm.dtu.dk

***Abstract*—This paper presents an abstract service based modelling method for use in performance estimation and design space exploration of Multi Processor System On Chip (MPSoC) based systems. The method provides the infrastructure for composing abstract hardware and software models of stream based systems which can be used to produce detailed quantitative information regarding runtime properties of a given system through simulations. The method is based on a service oriented model of computation which is a modified version of Hierarchical Coloured Petri Nets.**

I. Introduction

During the last few years, advances within VLSI technologies have introduced a new paradox. In order to harvest the full potential of these advances, the increasing architectural design complexity must be addressed. This calls for novel design methods which allow early, fast and accurate design space exploration making it possible to evaluate and select the best suited configuration of a given system without compromising the overall cost and time-to-market constraints of the system.

This paper presents an abstract service oriented hardware and software modelling method for use in the design space exploration of stream based Multi-Processor System on Chip (MPSoC) systems. The method enables designers to construct abstract models of execution platforms, consisting of both hardware and software components, making it possible to perform a quantitative analysis and evaluation of a given platform at an early stage in the design phase. The method is simulation based and built upon a model of computation which is a modified version of Hierarchical Coloured Petri Nets (*HCPN*) [1].

The method delivers an infrastructure for use in MPSoC system level design space exploration by providing well defined entities that allow the construction of *system models* modelling both hardware and software. Figure 1 gives an overview of the method. A system model of an MPSoC system is constructed by mapping the contents of one or more *application models* onto the processing elements of a *platform model*. A platform model of a given target architecture is composed of one or more *component models*, each modelling e.g. a processing element, a memory structure, an inter-connect structure etc. Component models are implemented by *service models* [2]. A service model models the behaviour of a given component by the availability of a number of services. Depending on the level of abstraction used, a service can represent anything from

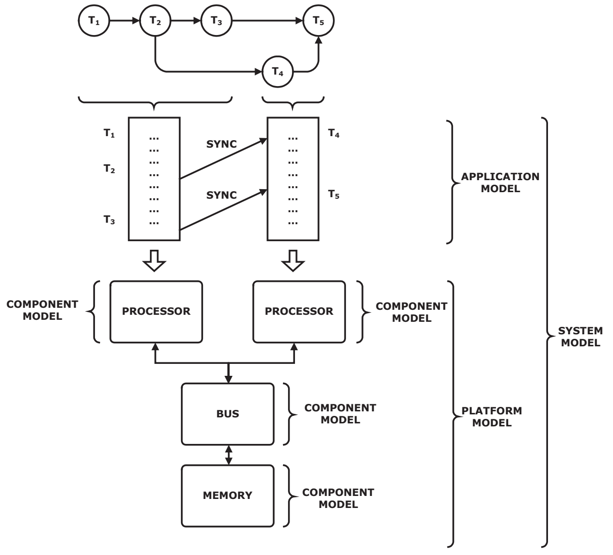

Figure 1. Overview of the proposed modelling method.

a compound task over arithmetic operations to actual low-level processor instructions offered by the given component. Application models thus represent the behaviour of the target application by a trace of requests to the services offered by the service model onto which the application is mapped.

The simulations performed at system level, using a system model, enable designers to extract detailed information regarding runtime properties of the applications and the architecture on which these are running e.g. execution profiles, resources utilization, memory usage, communication channel utilization, stalls and their causes, etc. and, thus, directs the designer to the best suited configuration of the system.

II. Related Work

A number of modelling methods and frameworks for use in design space exploration of MPSoC systems have been seen within recent years [3], [4], [5], [6], [7].

MILAN [3] and Metropolis [4] both aim to provide a framework in which models can be described at multiple levels of abstraction and gradually refined. They both allow multiple models of computations to co-exist within the framework. However, the means are different. MILAN defines a number of simulator models which can be integrated into a common model whereas Metropolis focuses on defining common communication semantics which allow the different models to communicate. In contrast to these generic approaches, our method aims at providing support for models described at different levels of abstraction and a gradual refinement of these, using a unified model of computation in order to reduce the communication overhead and simplify the required control logic during simulations.

Artemis [6], and the sub-project Sesame [8] also focus on design space exploration but are targeting stream based applications only. Application and architecture models are implemented using two different models of computation and simulated in parallel, letting the application model generate a trace of events as input to the architecture model. The functionality of an application is modelled in the application model and only the resource access, latencies, and communication constraints etc. are modelled in the architecture model. Our method employs a variant of trace driven simulation in which a trace of service requests is used to model the applications. However, in our case, both the functionality of applications, resource access, communication channels, etc. are modelled within the same model.

In [5] and [7] two different approaches that rely on static methods of analysis in order to perform design space exploration are presented. In theory, these approaches eliminate the need for simulations in order to predict performance. However, in most cases, the accuracy of these approaches only justifies their use in the very early stages of design space exploration where they can be used to reduce the number of potential candidate architectures.

Service models were originally presented for hardware/software co-design in [2]. The basic idea of service models is very intuitive from a hardware designer's point of view and it is a very appealing method of modelling hardware/software co-design problems. However, the original service models are only intended for modelling a single top level hardware component and, thus, cannot be used for modelling MPSoC systems. This issue is addressed in the method presented in this paper and the basic service model approach is heavily extended making service models usable in the current context.

The advantages of the modelling method presented in this paper are the high flexibility, the refinement possibilities of models, and the high level of accuracy which is obtainable. Service models have well defined interfaces and together with the hierarchical composition properties inherited from HCPNs, it is possible to combine models described at different levels of abstraction into one model, allowing a gradual refinement of the details of a model, or interchanging models, in order to investigate different implementations. At the same time a number of properties of service models simplifies the searching for enabled binding elements during the simulations of models, which is an extremely time consuming task in traditional HCPN models. This makes it possible to achieve much faster simulation times compared to traditional HCPN models. However, this is not the subject of the current paper and will be presented in a future paper.

III. Service Models

A service model is an abstract model of a hardware component implementing the behaviour of the component by offering a set of services. Each service can represent anything from, in the most abstract case, a compound task or functionality over arithmetic operations to actual instructions of a processor. The service model defines the services that are provided, how their behaviour is implemented as well as their latency.

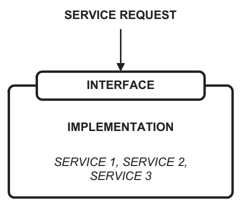

Figure 2. A Service Model, composed of an interface and an implementation, offering three abstract services: *Service 1*, *Service 2* and *Service 3*.

Figure 2 illustrates the basic principle of service models. Service models are composed of one or more *service model interfaces* and a *service model implementation*. The service model interfaces provide a uniform way of accessing the services offered by the service model as well as the possibility for multiple service models to communicate with each other. The service model implementation implements the detailed behaviour of the component being modelled by defining the operation of each service offered by the model. Service models inherit the hierarchical properties of HCPNs and, thus, a service model can be composed of several other service models. The uniform interface combined with the hierarchical properties play a vital role for the flexible refinement possibilities found in the service model concept, making it possible to use multiple levels of abstraction in different parts of the model and to interchange models easily, e.g. in order to increase the level of accuracy or in order to use a different implementation of a sub-component.

In order to introduce the basic concepts and the compositional properties of service models, the simple architecture, which can be seen in figure 1 is considered. The architecture consists of two identical processors, a bus and a shared memory.

A. The Service Model Interfaces

In order to provide a uniform way of accessing the services offered by a service model, abstracting away the details of the implementation and to allow a seamless connection of service models, two types of service model interfaces are defined:

1) The *passive service model interface*
2) The *active service model interface*

The passive interface defines which services can be requested and provides means for requesting services from the service model as well for indicating when the requested service has completed its execution. Thereby, the passive service model interface provides a uniform way of requesting the services offered by a service model, acting as an input interface. The active service model interface allows a service model to request services from other service models. This is done by connecting the active service model interface to a passive service model interface of another model, and thus, the active service model interface acts as an output interface.

It is a mandatory requirement for a service model to implement at least one passive service model interface in order for the service model to export its services. Active service model interfaces, on the other hand, are optional to implement and are only required if the service model needs to request services offered by other service models via their passive service model interfaces. The use of service model interfaces makes it possible to abstract away the details of the implementation of a given service model. In this way, multiple service model versions of a given hardware component, possibly at various levels of abstraction, can be constructed and interchanged freely in the design space exploration phase.

As an example of the use of interfaces, the processor models from the architecture shown in figure 1 define both a passive and an active service model interface. The passive interface allows application models to access the services of the processor, i.e. modelling the execution of the application on the processor. The active interface is used to connect the processor to the bus model, allowing the processor to gain access to the shared memory.

B. Service Model Implementations

The service model implementation defines the actual behaviour of the service model by implementing the functionality, the latency and the resource requirements of each service offered by the service model. During the simulation of a service model, a service is requested by placing a *service request* in the service model via the passive service model interface. A service request specifies the requested service, a list of arguments which can be empty, and a unique request number used to identify the service request by the simulation engine e.g. to annotate the execution time of the service request. The argument list can be used to provide input arguments to the implementation of a service or to allow the modelling of data operand dependencies by letting the arguments specify one or more data operands that must be present before the service can be executed. If the requested service is available and ready for execution, the service model will start executing the service request. Depending on the implementation of the service model, an arbitrary number of service requests can be processed in parallel e.g. modelling pipelines, VLIW, SIMD, and super scalar architectures. The completion of the processing of a service request is indicated via the passive service model interfaces through which it was originally requested. The processed service request will be either consumed by the model that originally requested it or removed and annotated by the simulation engine as will be discussed in section V.

In order to support a gradual refinement of the implementation of a service model, it is possible to compose the service model implementation as a hierarchical model. Thus, multiple implementations of a specific subpart of the model can be easily inter-changed in order to investigate different implementations or to allow a gradual refinement of the model. The level of accuracy used to describe the service model implementation determines how accurate the service model implementation is.

Thus, to summarize, the basic structure of the service model implementation dictates the timing and concurrency properties of the model. However, the actual behaviour of a service is implemented by associating actions with the services of the model. If service actions are combined with the possibility of adding arguments to the service requests using the argument list, the actual behaviour or operation of the service can be implemented in the model allowing e.g. the implementation of an actual addition of two values, or the possibility of modelling data operand dependencies, etc. This emphasizes the great potential of the method with respect to flexibility and accuracy. It is possible to refine models to a level where they can be used as e.g. cycle accurate instruction set simulators if needed.

C. Platform Models

Platform models are used to model hardware architectures and are composed of one, or more, services models each modelling a hardware component of the target architecture and specify how the service models are inter-connected. In this way platform models can represent arbitrary target architectures. In order to increase the level of productivity, component libraries are used. If a specified component does not exist in the component libraries used, then it must be designed and implemented manually. The platform model of a given target architecture is implemented as a service model having one or more passive service model interfaces as illustrated in figure 3. The services offered by a platform model are specified by the internal service models of which it is composed. The services of the platform model are accessible through the passive service model interfaces from e.g. the application models which are mapped to the platform.

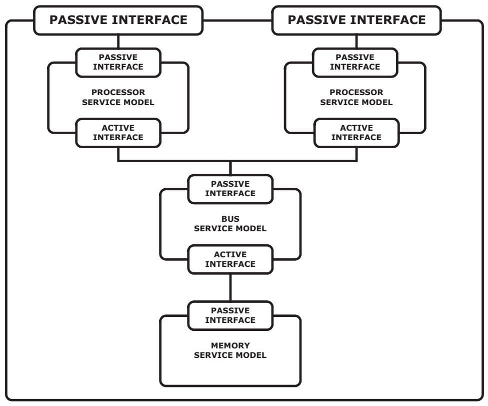

Figure 3. The resulting platform model of the example platform from figure 1. Two processor models (active) and a memory model (passive) are connected via a shared bus model.

Service models can be connected in serial and/or in parallel. A serial composition only involves two service models and, thus, is the simplest form of connection. In the example platform, shown in figure 3, the active service model interface of the bus service model is connected directly to the passive service model interface of the memory service model forming a serial composition. Similarly, the two processors are connected to the same passive service model interface of the bus service model forming a parallel composition.

There are no restrictions on how inter-component communication is modelled nor on the level of abstraction that is used, implying that, in principle, all types of inter-component communication methods are supported. Communication between service models within the same hierarchical level requires that the two service models *must* implement an active and passive interface, respectively. Thereby, the model implementing the active interface is allowed to communicate with the service model implementing the passive interface by requesting its services. If full duplex communication is required, two separate connections are needed and the service models involved must implement both a passive and an active service model interface. Communication across hierarchical levels, such as in the case of the example shown in figure 3 where passive interfaces are connected directly, is supported by allowing two passive or two active interfaces to be connected directly. The two interfaces are then combined forming a single logical interface, used e.g. to export an interface of a sub-component to the top level model.

IV. Application Models

In order for the method to be usable as the infrastructure in design space exploration, it is a mandatory constraint that it can capture the behaviour of the applications running on the architecture being modelled. In the method presented, applications are represented by an application model which is specified as a trace of service requests, each requesting a service offered by the execution platform onto which the application has been mapped. Service requests can be implemented at various levels of abstraction ranging from the most abstract case, such as the request of a function over arithmetic operations, to instructions and actual machine code if desired. This property gives the modelling approach a high degree of flexibility and the interpretation of the service requests is directly determined by the implementation of the service model on which the application runs. The use of service requests implies that there are no constraints on the type of applications supported, as long as an entity capable of translating the application into a trace of service requests exists and that the services requested are provided by the platform which executes the application.

V. System Models

A system model is composed of an application model mapping and a platform model. Simulations performed on system models enable designers to extract detailed information about the runtime properties of a given system, which can be used for design space exploration. Figure 4 illustrates how a system level simulation is carried out. The overall control of the system level simulation is concentrated in the entity referred to as the *simulation engine*. The simulation engine is responsible for controlling the model by e.g. issuing initialization and cycle increment commands corresponding to the tick of a clock in a synchronous system.

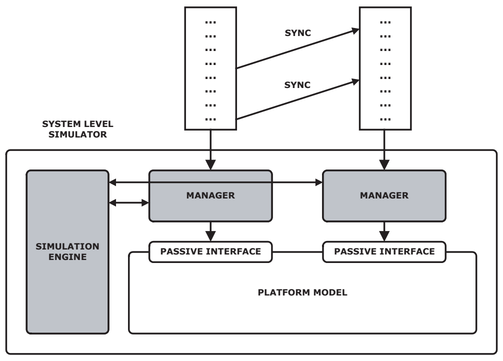

Figure 4. The system level simulation framework. The simulation engine is responsible for the overall control. Each passive service model interface of the platform model is being managed by a manager entity responsible for requesting services modelling the execution of applications.

In order to model the execution of applications, a *manager* is associated with each passive service model interface of the underlying platform model. The manager is responsible for requesting the services specified by the service request trace which represents the application that has been mapped to the passive service model interface that is being managed by the manager. The task of the manager can be compared to the task of a scheduler. However, in most cases the task of the manager will be quite trivial because the trace of service requests to be requested is generated during the compilation of the application prior to runtime. The manager is also responsible for extracting and annotating the execution time of each service request of the application trace when it has been executed, i.e. when it is indicated by the passive interface being managed, and for removing the service request from the model.

The managers thus collect information about the execution time of the applications running on the platform. In order to extract other runtime information, such as memory usage, resource occupancy, stalls and their causes, etc., it is possible to associate custom event loggers with the individual models. Thereby, the designers of the model can determine which properties should be logged, and, thus, the method does not limit the type of properties which can be extracted.

VI. Experimental Results

The presented modelling method has been used for design space exploration of an experimental audio processing platform developed at Bang & Olufsen ICEpower a/s. The platform is based on a small application specific pipelined processor denoted the *SVF* processor[1] and the results are

[1]The SVF processor is a synchronous fixed point processor having an instruction set designed to be optimized for running applications composed of state variable filter structures (*SVF*) [9]

presented in [10]. The modelling method was used to find the best possible mapping of a streaming audio application onto various configurations of the platform consisting of one to five SVF processors connected, either using point-to-point connections or via a shared bus, and in terms of throughput and utilization of the platform. Furthermore, the effect of being able to perform forwarding of data operands is also investigated through three different versions of the SVF processor model. The first is modelled without forwarding capabilities, the second version with full forwarding capabilities, and, lastly, the third version is modelling the actual hardware implementation.

Table I
UTILIZATION (η) AND CAUSE OF STALLS. *R* DENOTE RECEIVE STALLS EXPERIENCED WHEN A PROCESSOR NEEDS DATA FROM ANOTHER PROCESSOR, *D* DENOTE DATA DEPENDENCY STALLS CAUSED BY MISSING FORWARDING CAPABILITIES.

		SVF#NoFWD			**SVF#FWD**		
		η	**R**	**D**	η	**R**	**D**
SVFx5	**P1**	55.6%	0%	100%	100%	0%	0%
BUS	**P2**	55.0%	2%	98%	99.4%	100%	0%
	P3	54.5%	5%	95%	98.8%	100%	0%
	P4	53.9%	7%	93%	98.2%	100%	0%
	P5	53.4%	9%	91%	97.6%	100%	0%

The highest throughput was obtained when each of the application sub-blocks was executed on its own processor, corresponding to pipelining the application. The utilization of the processors was influenced mainly by two factors. The first was the mapping of the application. In the configurations with multiple processors, a balanced mapping of the application sub-blocks resulted in the highest utilization of each processor. The second, and most important factor, was the architecture of the SVF processor itself. Table I shows the utilization ratio of each processor of the platform configuration which had the highest throughput, the **SVFx5 BUS**. This configuration consists of five SVF processors connected via a shared bus. It clearly shows the importance of implementing forwarding capabilities (**SVF#NoFWD** vs. **SVF#FWD**). As can be seen, the utilization ratio increases dramatically when forwarding is implemented, implying that the throughput of the overall platform can be increased.

An even more interesting result is that the time spent on design space exploration using the current method compared to the traditional method employed at Bang & Olufsen ICEpower a/s, where RTL descriptions are used, is reduced significantly. The time needed for constructing a system model with the presented method, compared to writing an RTL description, is much shorter and more straightforward due to the higher abstraction level used. The result is a reduction of the time needed for the type of design space exploration presented in [10] from weeks to hours.

VII. Conclusion and Future Work

This paper has presented an abstract service based hardware and software modelling method for use in design space exploration and performance estimation of MPSoC based systems. The method provides means for a flexible construction of models and allows models described at different levels of abstraction to co-exist. At the same time, the obtainable level of accuracy is very high ranging from the modelling of basic properties and behaviours to actual data values and word sizes, determined solely by the level of abstraction used to describe the model.

The execution semantics currently employed in the simulation engine of the method only supports the modelling of hardware components consisting of a single clock domain. It is of vital importance that the execution semantics of the modelling framework is extended to support multiple clock domains in order to reflect real world MPSoC systems.

Another urgent issue that needs to be addressed is the extension of the application model to include a more detailed modelling of the dynamic properties of the application which is currently handled by the manager entities during simulation. The idea is to extend the use of service models to also cover the modelling of applications e.g. for modelling operating systems, middleware, etc., and let these generate the trace of service requests dynamically.

Acknowledgements

This work was supported by DaNES (Danish National Advanced Technology Foundation) and ARTIST2 (IST-004527). The authors explicitly want to acknowledge the industrial case provided by the DaNES partner Bang & Olufsen ICEpower a/s.

References

[1] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Springer-Verlag, 1992.

[2] J. Grode and J. Madsen, “A Unified Component Modeling Approach for Performance Estimation in Hardware/Software Codesign”, vol. 1. IEEE, 1998, pp. 65–69.

[3] A. Bakshi, V. K. Prasanna, and A. Ledeczi, “MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems,” *SIGPLAN Not.*, vol. 36, no. 8, pp. 82–93, 2001.

[4] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, “Metropolis: An Integrated Electronic System Design Environment,” *Computer*, vol. 36, no. 4, pp. 45–52+4, 2003.

[5] M. F. S. Oliveira, a. Eduardo W. Bri F. A. Nascimento, and F. R. Wagner, “Model Driven Engineering for MPSoC Design Space Exploration,” in *SBCCI ’07: Proceedings of the 20th annual conference on Integrated circuits and systems design*. New York, NY, USA: ACM, 2007, pp. 81–86.

[6] A. Pimentel, L. Hertzbetger, P. Lieverse, P. van der Wolf, and E. Deprettere, “Exploring Embedded-systems Architectures with Artemis,” *Computer*, vol. 34, no. 11, pp. 57–63, 2001.

[7] A. L. Varbanescu, H. Sips, and A. Van Gemund, “PAM-SoC: A Toolchain for Predicting MPSoC Performance,” *Lecture Notes in Computer Science*, vol. 4128 LNCS, pp. 111–113, 2006.

[8] A. Pimentel, C. Erbas, and S. Polstra, “A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels,”*IEEE Transactions on Computers*, vol. 55, no. 2, pp. 99–112, 2006.

[9] H. Chamberlin, *Musical Applications of Microprocessors*. Indianapolis, IN, USA: Sams, 1980.

[10] A. S. Tranberg-Hansen, J. Madsen, and B. S. Jensen, “A Service Based Estimation Method for MPSoC Performance Modelling,” *2008 International Symposium on Industrial Embedded Systems*, pp. 43–50, 2008.