SYSTEMC BASED ARCHITECTURE EXPLORATION OF A 3D GRAPHIC PROCESSOR

Tim Kogel, Andreas Wieferink, Heinrich Meyr
Integrated Signal Processing Systems
Aachen University of Technology, Germany
kogel,wieferin,meyr@iss.rwth-aachen.de

Andrea Kroll
Synopsys Inc.
Mountain View, CA
akroll@synopsys.com

Abstract - In this paper we propose a system level design and refinement methodology based on the SystemC class library. We address design space exploration and performance profiling at the highest possible level of abstraction. System level design starts with the initial functional specification and validation of the system behavior in SystemC. The refinement methodology covers architecture exploration and results in an executable System Architecture Model, which is able to generate the relevant profiling data and to verify if the chosen architecture meets the performance requirements.

We have applied this methodology to a 100 million gate design of a 3D graphic processor. In this project we were able to demonstrate the feasibility and define the final system architecture within 2 months. This 3D processor implements the ray-tracing rendering paradigm on one chip allowing real time rendering of 3D scenes with photo-realistic quality. Based on the results of this case study we present the benefits of our methodology to successively define a feasible system architecture coping with the processing and memory bandwidth requirements.

INTRODUCTION

One of the most challenging tasks in modern System-on-Chip design projects is to map a complex application onto a heterogeneous architecture in adherence to the specified performance and cost requirements. Driven by the ever increasing number of transistors integrated into one piece of silicon, the complete system functionality can be integrated on a single chip. Since power dissipation becomes a major design constraint in nearly all application domains, the designer has to trade carefully the flexibility of programmable architectures against the computational efficiency of dedicated hardware [14].

Under stringent power constraints, the required flexibility and performance is best delivered by a heterogeneous system architecture employing standard as well as application specific programmable architectures and dedicated hardware blocks. As a result, the designer faces a huge design space and has to compose a system architecture from various kinds of building blocks in order to meet the constraints of the specific application.

From the designers perspective, the overall performance of a system can be separated into the performance delivered by the selected algorithms on the one side and the performance of the underlying architecture on the other side. Hence the traditional design flow is subdivided into two decoupled phases. First the functionality of the system is defined during the algorithm exploration phase, which is usually performed with application specific tool and library support (e.g. SPW [5] and COSSAP [12] for wireless communications or OPNET [11] for networking). In the second phase of the design flow, the resulting system specification is implemented.

This traditional approach is no longer feasible for the design of large heterogeneous systems on a single chip, because quantitative architectural considerations are not taken into account prior to the implementation phase. Due to the high level of detail of implementation models, they are only optimized locally and system architecture tradeoffs and optimizations are not exploited. For that reason we propose an intermediate phase in the design flow, where the functionality of the system is mapped to the architecture in an abstract manner to enable architecture optimizations across heterogeneous computational components. Bringing both parts - the functional specification and the architectural specification - together is the most challenging task in the design process for on-the-edge systems.

The paper is organized as follows: first we discuss the related work before we introduce our system level design methodology. Key aspects of this methodology are to capture the systems behavior in an abstract and very efficient way and then add architectural details successively to that functional description. The applicability of our methodology is demonstrated by an industrial case study in section . Finally we briefly conclude the results of our work.

RELATED WORK

The issues of System Level Design have attracted a growing attention from both university and industry research teams. It is commonly accepted to cope with the growing system complexity by raising the abstraction level of the initial specification to explore architecture tradeoffs and guide HW/SW partitioning decisions.

The proposed specification languages and co-design frameworks cover all conceivable abstraction levels and application domains, but can be assessed by the following classification. The more powerful the co-design framework is in terms of accurate performance and cost estimation, automated HW/SW partitioning and synthesis, the higher is the specification effort and the more restricted is the framework to a certain application domain. Our approach intends to minimize the formal specification overhead and is therefore applicable for large scale designs and early feasibility considerations. In a very early design stage, where the feasibility of the design is still unproven, the effort to build a system model capable for automated co-design is often unjustifiable.

Starting from a formalized system description, complete co-design frameworks address architecture exploration, HW/SW partitioning and synthesis. Polis [4] from UC Berkeley and ArchiMate from Arexsys [3] fall in this category, since they are focused on automating the complete design flow for embedded systems. In contrast to such co-design frameworks we do not address automatic synthesis, but model the impact of the architecture on the highest possible level of abstraction. For verification purposes the hand-crafted HW and SW implementation models are integrated into our abstract architecture model via a co-simulation interface [1], which is capable of adjusting different levels of abstraction.

N2C [6] from CoWare relies on the RPC[1] paradigm, which will be part of the SystemC 2.0 specification, to build an executable system specification. From there N2C enables interface synthesis for rapid IP integration in SW centric designs, where the target architecture is a programmable core with accelerators connected to the system bus.

Commercially available HW/SW co-simulation tools like Seamless [10] are useful for the verification of implementation models. On the other hand they are not useful to guide architecture decisions in system level design due to the low abstraction level and low simulation speed.

[1] Remote Procedure Call

C/C++ based system modeling languages like SpecC [8] and SOCOS [7] are considered as the most promising vehicles to cope with the ever increasing complexity of SoC designs. The SystemC initiative [13] will probably establish a common platform for these kinds of system level design methodologies.

In this context the major contribution of this paper is a system level design methodology, which is based on the standard SystemC library. This methodology has proven to handle the enormous functional and architectural complexity of SoC designs at the edge of silicon feasibility.

SYSTEM LEVEL DESIGN METHODOLOGY

In this section we introduce our system level design methodology. The procedure of defining a system architecture can be subdivided into building an initial, pure functional executable specification and then finding a suitable architecture mapping.

Functional model As depicted in figure 1, usually the first step in defining a new application is to select the algorithms. This algorithmic exploration is performed with application specific tool and library support, where the system designer can concentrate on the algorithm development and profiling of the algorithmic performance.

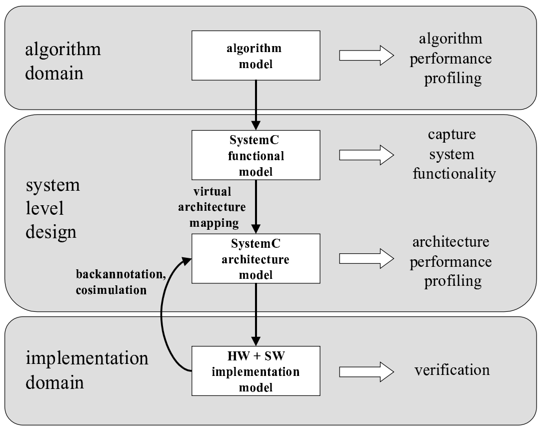
algorithm domain
algorithm model
algorithm performance profiling
system level design
SystemC functional model
capture system functionality
virtual architecture mapping
SystemC architecture model
architecture performance profiling
backannotation, cosimulation
implementation domain
HW + SW implementation model
verification

Figure 1: SoC design flow

Our methodology covers the system level design domain in figure 1, i.e. after the algorithmic exploration is finished and when the optimal architecture executing the functionality is to be defined. Generally, the SoC architecture can be seen as a set of parallel communicating blocks. To capture the complete functionality and to enable abstract architecture mapping, the functionality is first to be partitioned into SystemC modules. Metrics like minimization of information exchange between modules and algorithmic locality are used to guide partitioning decisions.

According to our methodology the SystemC process network is constructed to fulfill the following requirements in system level design:

simulation speed and modeling efficiency is mandatory to handle the high complexity of SoC designs.

separation of interfaces and behavior as motivated by Rowson [9] enables successive refinement of communication and structure.

separation of timing and behavior allows efficient profiling of functional blocks mapped to alternative architectures.

The first goal, high simulation speed and modeling efficiency, is achieved by raising the abstraction level of the HW model from the register transfer level (RTL) to an abstract system architecture level.

In the abstract model, functionality is partitioned into coarse grain SoC building blocks instead of scattering the functionality over numerous processes as often required for a synthesizable RTL description. The bit-true data representation of the RTL model is replaced by abstract data types, such that a whole set of functionally associated data is represented as a single token.

The key concept of raising the abstraction level is the introduction of a coarse grain time scale. For system performance profiling a time base is needed, but the high resolution of hardware clock cycles ruins simulation speed. Since the system state only changes on the arrival of a new token, the time base of our system model is given by a logical macro cycle. The macro cycle period corresponds to the minimal length of time between the arrival of two consecutive tokens.

At the end of this stage, the complete system behavior is captured by a network of communicating processes exchanging tokens. The simulation speed as well as the modeling efficiency (measured in lines of code) is at least two orders of magnitude better compared to the corresponding RTL model, which models the same functionality on a much higher level of architectural detail. The system model is now prepared for the annotation of timing information, which is described in the next section.

Virtual architecture mapping In the next design step, the functional model is mapped virtually to the intended target architecture in order to create a performance model of the resulting system architecture. The mapping is performed virtually by annotating the timing characteristics of the target architecture to the functional model, thus the methodology enables a very fast exploration of different design alternatives. The process of timing annotation is completely orthogonal to the functionality, hence the previously validated functional system behavior remains unchanged.

The methodology is based on the following observation: for performance profiling purposes, the basic timing characteristics of the target architecture can be expressed by the temporal relationship of consuming, processing and producing tokens.

- Pipelined architectures are able to consume and produce a token every cycle but introduce a static latency, which is determined by the number of pipeline stages.
- Data dependent modules show varying delays until the processing of the actual token is finished. In the case of a cache module for example, the processing delay of a cache read depends on whether the requested data set is in the cache or has to be fetched from the main memory.
- Resource shared modules and programmable architectures are blocked for a varying amount of cycles until the token is processed.

Our goal is to model the architecture specific timing independently from the behavior, thus a functionally correct system model can be easily mapped to different architectures. Therefore we have developed a set of channel models for the interconnection in the SystemC process network. They are all based on a lossless FIFO buffer according to the discrete event multiple-write, single-reader communication model. The channels provide methods to annotate the temporal relation between processes according to the above listed observations.

A statistical evaluation system is associated with the channel models, which produces histograms sampling the actual queue length of tokens on the channel. In case a FIFO queue between two processes grows very long, the reading process has not sufficient processing power to cope with the arriving tokens. In this way architecture bottlenecks can be detected and eliminated very early in the design flow before the time consuming implementation starts.

The key mechanism of virtual architecture mapping is separating behavior from timing aspects. In our model all timing related aspects are captured by the communication channels. Thus we achieve a threefold orthogonalization of system level design concerns in terms of function, interface and timing. In the next section the applicability of our system level design methodology is demonstrated by a large scale design example.

THE AVALON CASE STUDY

In this section we will introduce the AVALON project, where the proposed system level design methodology has been applied to a large scale industrial design project of a 3D graphic chip. We will first introduce and characterize the design and then show in detail the virtual mapping and profiling of one exemplary component.

Project overview and characterization Goal of the project is to show the feasibility of a real-time ray tracing [2] chip and estimate its cost in terms of power consumption and area. On the one side the required real-time frame rate of at least 25 frames per second must be fulfilled. On the other side the constraints introduced by the architecture and by the surrounding environment, that include limited processing power of the components, limited memory bandwidth, and limited bus capacity for data I/O and must be taken into account.

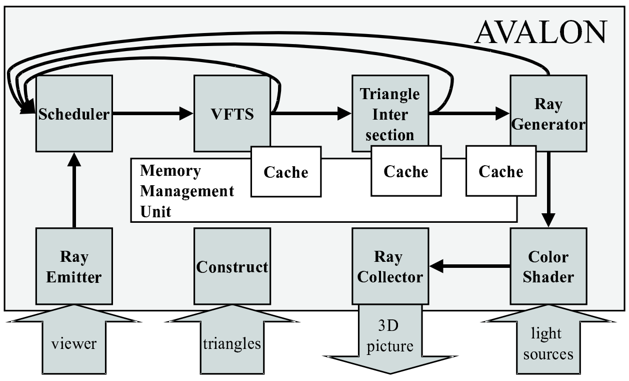

Figure 2: avalon system architecture

The first step is to specify the algorithms and define a coarse structure of the system as depicted in the system overview in figure 2. The 3D scene represented by a list of triangles is loaded into the *construct* module. Here the world is subdivided into a three dimensional grid data structure and all triangles are inserted into that grid. After this preparation, the *ray emitter* is ready to send rays into the scene in order to find the first triangle in the view of the spectator. The job of finding this triangle is performed jointly by the *very fast triangle search (VFTS)* and the *triangle intersection (TI)* units. The VFTS traverses the grid data structure along the ray direction in search for triangles and passes the located triangle list together with the ray to the TI. Here the ray is intersected with the triangles in the list and passed to the *ray generator* in case of a valid intersection result or otherwise back to the VFTS to search for further triangles. The ray generator then checks for the generation of new rays depending on the surface properties of the respective triangle like transparency or reflection and additionally sends rays to all light sources in the scene. Finally the color of the ray is determined by the *color shader* and all rays are collected in the *ray collector*.

For this design a very high processing power is needed to cope with the real-time requirements. Furthermore we have tremendous bandwidth requirements towards the main memory holding the scene information, which makes the use of caches mandatory. Thus each component in the feedback loops has a data dependent timing behavior, because not all data sets required to process the actual ray are available in the local caches. This results in a frame rate strongly depending on the actual scenario. A bottleneck in one part of the system within any loop results in an overall performance loss in terms of a dropping frame rate.

Functionality and architecture are tightly linked in this application domain and the simulation speed to validate the performance of the system must be very high. In the following we will describe the modeling steps we have introduced in section exemplarily for the triangle intersection unit. We present the successive refinement of the models from a pure functional description of the system towards an architecture. In the final stage the latency and throughput of each component can be observed and the performance of the distributed memory structure can be measured in the system context.

Architecture refinement and exploration In this section we describe, how the initial functional SystemC model is successively mapped to an architecture by refining the structure and annotating timing information.

The methodology of virtual architecture mapping is demonstrated exemplarily for the triangle intersection unit. At a first glance this component has the trivial functionality of calculating the intersection point between a ray and a list of triangles, which is nearest to the origin of the ray.

The component is much more interesting from the architectural point of view because of its very high processing and memory requirements. The component must accomplish up to $4 \cdot 10^9$ intersections per second to achieve the specified frame rate and picture resolution. Every intersection test has the complexity of 27 multiply and 14 add operations resulting in $164 \cdot 10^9$ FLOPS for this component. Additionally a data structure of 32 byte size representing the geometrical information for one triangle must be loaded for every intersection test, which corresponds to a memory bandwidth of 128 GB/s. Thus it is quite a challenge to find a feasible architecture for this component.

The successive steps of mapping the functional SystemC model to a virtual architecture model are depicted in figure 3. The starting point is the initial functional model in 3 a), where the functionality of the component is modeled with roughly 100 lines of C++ code which were written and validated within 2 days. The whole SystemC model on the pure functional level takes about 12 minutes to render a complete frame with a pixel resolution of 800 x 600 and a scene complexity of 140.000 triangles.

In a second step the functional model is refined to the intended architecture as

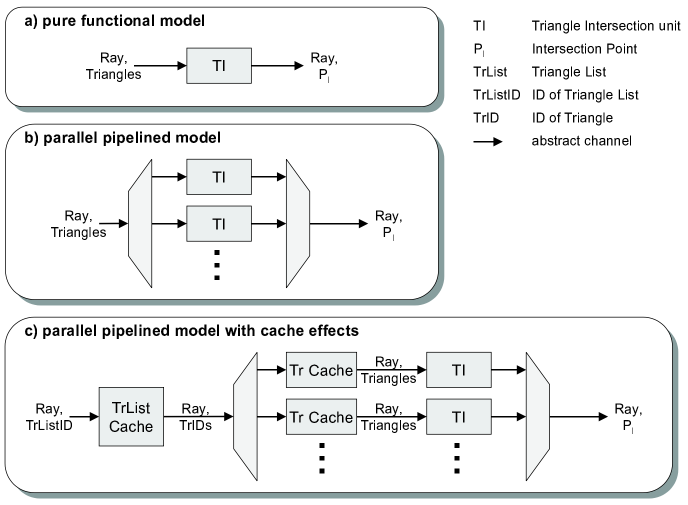

Figure 3: successive architecture refinement

depicted in figure 3 b). The required processing power can only be delivered by a set of parallel and deeply pipelined processing units. Now the functional TI model is instantiated several times within a structural SystemC module. Further components are added like a demultiplex component distributing arriving data tokens to the processing units and multiplexer merging the intersection results into the single output. The pipelined character of the processing units is modeled with the channel models introduced in the methodology section, which are instantiated between the TI and the multiplexer module. The tokens representing the intersection result are written to the channel with a static delay corresponding to the number of pipeline stages.

The refined structural model of the TI is parametrical with respect to the number of pipeline stages and the number of parallel functional units. It has been written and verified within only 2 days, since the verified functional model and the surrounding components remain untouched. The resulting simulation performance decreases due to the higher degree of detail and scales linearly with the number of parallel units.

On this level of detail we are able to profile the static peak performance of the triangular intersection unit and its influence on the resulting system performance, but the limited memory bandwidth and the influence of cache effects is not yet taken into account. For this purpose the architecture model is further refined as depicted in figure 3 c). In order to make cache effects visible we employ a cache model, which is generic in terms of cache size, block size, bus width, associative degree and replacement strategy. Now the architecture model considers the fact, that the triangle list containing only pointers to triangles is fetched from a cache and aftewards the triangle data structure is fetched from another cache. Furthermore in this application the data sequence has to be preserved, so that the cache units are stalled in case of a cache miss. This is modeled by suspending the cache process until the data is fetched. Now the performance influence of the dynamic delay introduced by data dependent cache hits and misses is modeled and allows precise information about the resulting system performance.

The cache aware system simulations now shows that a 2nd level cache is necessary to meet the performance and external memory bandwidth requirements. Furthermore we found out that in consideration of the cache effects the scheduling algorithm has to be adapted for optimal system performance. For the static performance model, a load balancing scheduling algorithm shows the best profiling results. Instead for the dynamic cache model a scheduling algorithm exploiting geometrical locality between rays and thus optimizing cache coherence obtains superior performance results.

The final system architecture model comprising about 4000 lines of C++ code has been built within 6 weeks by a team of 4 engineers, who are experienced in the methodology and the application domain. At the highest level of detail, the simulation instantiates about 200 SystemC processes and takes about 6 hours to render a complete frame. Equipped with this generic SystemC model, it took us another 2 weeks to adjust the generic parameters of the architecture model and finally define a system architecture, which delivers the required processing power and is feasible in terms of hardware cost and memory bandwidth.

CONCLUSION

In this paper we proposed a system level design methodology based on the SystemC library. Our approach is capable to capture the complete system functionality as well as important architecture features on the highest possible level of abstraction. The resulting modeling efficiency measured in lines of code and the simulation speed is at least two orders of magnitude better compared to an RTL architecture model. We have employed the outlined methodology during the design of a 100 million gate 3D graphic processor, whereby we were able to define a feasible system architecture. This work was accomplished within 2 months by a team of 4 engineers familiar with the application and the methodology. The final system architecture model also serves as an executable specification and as a fast co-simulation environment for the HW and SW implementation.

References

[1] A. Hoffmann, T. Kogel, H. Meyr. A Framework for Fast Hardware-Software Co-simulation. In *Proceedings of the European Conference on Design, Automation and Test (DATE)*, 2001.

[2] Andrew S. Glassner. *An Introdution to Ray Tracing*. Academic Publishers, 1989.

[3] Arexsys. *ArchiMate Architecture Generator, http://www.arexsys.com*.

[4] Balarin et al. *Hardware-Software Co-Design of Embedded Systems : The Polis Approach* . Kluwer Academic Publishers, 1997.

[5] Cadence. *SPW, http://www.cadence.com*.

[6] CoWare. *N2C, http://www.coware.com*.

[7] D. Desmet, D. Verkest, H. De Man. Operating System based Software Generation for Systems-on-Chip. In *Proceedings of the Design Automation Conference (DAC)*, 2000.

[8] D. Gajski, J. Zhu, R. Dömer, A.Gerstlauer, S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[9] J.A. Rowson, A. Sangiovanni-Vincentelli. Interface-Based Design. In *Proceedings of the Design Automation Conference (DAC)*, 1997.

[10] Mentor Graphics. *Seamless, http://www.mentor.com/seamless*.

[11] OPNET. *http://www.opnet.com*.

[12] Synopsys. *COSSAP, http://www.synopsys.com*.

[13] SystemC initiative. *http://www.systemc.org*.

[14] Theo A.C.M. Claasen. High Speed: Not the Only Way to Exploit the Intrinsic Computational Power of Silicon. In *Proceedings of the International Solid-State Circuits Conference*, 1999.