Performance Evaluation of VLSI Platforms Using SystemQ

Sören Sonntag[1], Matthias Gries[2], Christian Sauer[2]
[1]Communication, [2]Corporate Research
Infineon Technologies, Munich
Soeren.Sonntag@infineon.com

Abstract: Modern VLSI systems exhibit increasing complexity. The size of a design not only grows, but the system architecture also becomes more and more heterogeneous and parallel. A key factor for a successful implementation is modeling and simulation of the design. However, modeling at register-transfer level (RTL) is not feasible anymore as a starting point. Already in the concept phase of a design quantitative estimations of the system's performance are required. To effectively support system engineers during this design phase we have developed a performance evaluation framework called SystemQ. By starting with a performance model we show how the system's behavior and structure can be refined systematically. SystemQ is implemented in SystemC and seamlessly supports the refinement of performance models down to transaction level and RTL.

1 Introduction

Recent trends in applications and technology lead to higher complexity in computation and communication of integrated systems. This leads, in turn, to more heterogeneous and parallel architectures to be developed. A key factor for a successful implementation is modeling and simulation of the design. However, modeling at register-transfer level (RTL) is not feasible as a starting point anymore. Tight time-to-market, cost, and power-dissipation constraints force the designer to estimate the performance of the system already very early in the design flow, namely in the concept phase. However, in this phase a detailed hardware description of the system is not available. Often, designers rely on back-of-the-envelope estimations. As complexity increases, new methodologies must be found to improve the quality of the design and decrease the design time.

Quantitative estimation (e. g. simulation) must be introduced early in the design flow, where system-level decisions have most impact. Abstractions must be found to speed up simulation times and enable system designers to evaluate many design alternatives. For these demands we have developed a performance evaluation framework called SystemQ [SGS05]. SystemQ enables the designer to efficiently evaluate different design alternatives. SystemQ is implemented in SystemC and provides a systematic path to architecture implementation, as shown with a complex packet processing system for the network access in this paper.

2 SystemC based Simulation

Our SystemQ framework relies on queuing theory [LZG+84] and is comprehensively described in [SGS05]. We chose a queuing-based approach because scheduling and workload dependencies are made explicit by the semantics of the formalism. Queuing problems can either be solved analytically or by simulation. The former, however, is a cumbersome task, especially for non-exponential arrival time distributions, and transaction-dependent service times as required in our case. Thus, we use simulation for evaluation.

Our SystemQ simulation environment is based on SystemC (*www.systemc.org*). We exploit SystemC's 2.0 communication support and its discrete event simulation capabilities for our timed queuing models by providing component and communication libraries. This way, performance information, such as latency or throughput of the overall system or individual building blocks, can be derived and potential bottlenecks can be identified. We base subsequent architectural refinement and implementation steps on SystemC's methodology.

Queuing systems consist of *queues* and *servers*. A queue is a waiting room where transactions are stored until a server becomes available to process the transaction. In SystemQ both types, queues and servers, are *sc_modules* that can be executed concurrently by using *processes*. Queues, modeled as *hierarchical channels*, are directly connected to servers. This way, queuing network semantics are enforced automatically at design time. Networks of queuing systems are composed from these elements by connecting ports with directed point-to-point *sc_channels*.

3 Supported Levels of Abstraction

Initially, the concept phase of the design flow starts with a system-level view of the design to be developed. At this time only general performance-relevant information will be needed by the system designer, i. e. the throughput of the system, the residence time of events in the system, maximum and mean memory usage, and resource utilization. Since SystemQ is based on queuing networks it is suited to reveal these properties of the system. With SystemQ's simulation approach we gain additional information since we can trace time dependencies, profile individual resources, and follow the processing of particular transactions through the model. Once the performance of the system has been determined and different architectural choices have been compared, more details come into play.

To address the designer's needs SystemQ allows three different refinement strategies, i. e. in structure, function, and communication as shown in Figure 1. This concept is also known as the separation of concerns principle in literature [KMN+00].

Functional refinement may add (and refine) the function of servers and/or introduce distinguishable transaction classes. *Structural refinement* decomposes queuing systems into queuing networks. This form of refinement is often motivated by incorporating details of the architecture in the model. *Communication refinement* adds precision to communication between queuing systems or changes communication semantically. This refinement is achieved using SystemC wrappers, described in detail in [GLMS02].

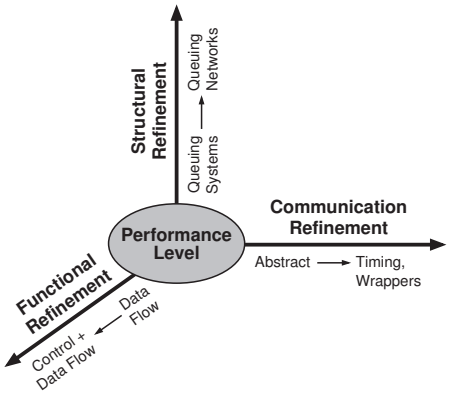

Figure 1: Orthogonal refinement strategies starting from performance level of abstraction

In the next section we show how to systematically refine a SystemQ model starting at the performance level of abstraction.

4 Refinement Example of a Packet Processing System

A domain well suited for evaluating scheduling aspects of a design is network processing. We exemplarily show in this section how to systematically refine a packet processing system for the network access. The chosen system is able to aggregate traffic of several digital subscriber lines (DSL) and to forward this traffic to the upstream core network. In downstream direction the system distinguishes packets per DSL customer and forwards them to the appropriate output port. Since DSL provides asymmetric data flow we have two processing elements for the downstream direction and just one processing element for the upstream direction.

The starting point of the refinement example is the performance level of abstraction. The most abstract model consists only of two queuing systems, i. e. one for the upstream direction and one for the downstream direction as shown in Figure 2. The functionality of the servers comprises three tasks, namely to fetch a packet from the attached queue, to wait for a determined amount of simulation time (*service time*), and to output the packet. The model is already feasible for determining the throughput of the system in upstream and downstream directions as well as for a rough latency estimation. However, this model is obviously very limited.

The first refinement step is a structural refinement by decomposition. Thereby both queuing systems will be replaced by queuing networks. This is shown in Figure 3, where the model consists of one queuing network per direction. The functionality of the queuing systems is still the same except that each queuing system has its own particular service time. An interesting concern is the modeling of the two parallel downstream processors. Without functionality the downstream ingress DMA cannot determine where to send the

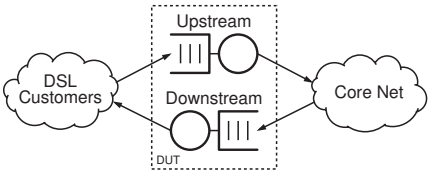

Figure 2: Abstract model of a packet processing system consisting of only two queuing systems

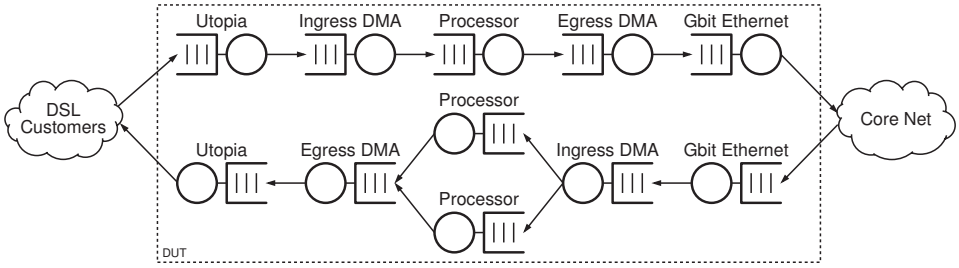

Figure 3: Top-level model of the packet processing system exhibiting queuing networks

packets to. For this output behavior an abstraction must be found. This might be an alternate behavior to balance the load of both processors or a random distribution.

The second refinement step shall therefore be of functional nature. Functionality is added for two reasons: To modify packets and to add control flow to the model. Control flow allows the downstream ingress DMA to choose the processor to send a packet to, e. g. per flow or per protocol. This level of abstraction allows several performance measures, e. g. minimum, maximum, or mean utilization of the queuing systems. However, as depicted in Figure 3, the current abstraction does not exhibit any dependencies between the upstream and the downstream direction. Indeed, no memory is modeled yet. Instead, the data is directly sent from the ingress DMA to the processor.

Therefore the third step (not shown) consists once more of structural refinement. Now a memory subsystem including priority arbiter and memory is made explicit in the model. To complete this refinement step the communication semantics must be changed. Until now only complete packets are sent through the system. Since packets will now be stored in the memory, the ingress DMAs will send only *packet pointers* to the processors.

Due to limited space the next refinement steps will only be enumerated. They consist of modeling more functionality of the servers, limiting the queue lengths and refining communication from point-to-point connections to buses. Thus, the transaction level of abstraction can easily be reached. In this level the strict separation of queues and servers will blur and queuing systems can be refined to transaction-level function blocks. Additional refinement steps are to further refine communication to functional correct transfers, to model even more queuing networks by decomposition, and finally to transform queuing systems into bit-true function blocks. Since SystemQ relies on SystemC's concept of communication with dedicated ports and interfaces, refinement can be continued until plain register-transfer-level code is present.

5 Related Work

Metropolis [BWH+03] is a general framework where arbitrary models of computation can be expressed. Its generality however makes the modeling effort more complex and less intuitive than by restricting the designer to one consistent representation for particular application domains.

Artemis [PHL+01] is targeted at media processing applications and is based on Kahn process networks. Extensions like virtual resources are required to enable scheduling analysis.

We recognize Click [KMC+00] as a modeling formalism for packet processing applications. Packet and resource scheduling are not part of the formalism. StepNP [PPB02] is written in SystemC and uses Click as input for applications, whereas architecture building blocks are SystemC modules at RTL or transaction level.

6 Conclusion

Based on our SystemQ approach we have presented a systematic refinement example of a simulation model. On a high level of abstraction the model is feasible to address the needs of VLSI platform architects already in the concept phase of the design flow. By systematic refinement more and more properties of the system can be revealed by adding details in terms of structure, functionality, and communication. Since SystemQ is implemented in SystemC it seamlessly supports the refinement of simulation models down to transaction level and RTL.

References

[BWH+03] F. Balarin, Y. Watanabe, H. Hsieh, et al. Metropolis: An Integrated Electronic System Design Environment. *IEEE Computer*, 36(4), April 2003.

[GLMS02] Thorsten Grötker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[KMC+00] E. Kohler, R. Morris, B. Chen, et al. The Click modular router. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[KMN+00] K. Keutzer, S. Malik, A. R. Newton, et al. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Transactions on CAD*, 19(12), 2000.

[LZG+84] E. D. Lazowska, J. Zahorjan, G. S. Graham, et al. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, 1984.

[PHL+01] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, et al. Exploring embedded-systems architectures with Artemis. *IEEE Computer*, 34(11), 2001.

[PPB02] P. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A System-Level Exploration Platform for Network Processors. *IEEE Design & Test of Computers*, 19(6), 2002.

[SGS05] S. Sonntag, M. Gries, and C. Sauer. SystemQ: A Queuing-Based Approach to Architecture Performance Evaluation with SystemC. In *SAMOS V*, *LNCS 3553*. Springer, 2005.