SystemC Based Design of an IP Forwarding Chip with CoCentric System Studio

Tim Kogel

Institute for Integrated Signal Processing Systems

tim.kogel@iss.rwth-aachen.de

Denis Bussaglia

Synopsys Professional Services

denisb@synopsys.com

ABSTRACT

The ever increasing complexity and heterogeneity of modern System On Chip designs demands early consideration and exploration of architectural alternatives, which is hardly practicable on the low abstraction level of implementation models. In this paper, a system level design methodology based on the SystemC 2.0 library is proposed, which enables the designer to reason about the architecture on a much higher level of abstraction. Goal of this methodology is to define system architectures, which provide sufficient performance, flexibility and power efficiency as required by demanding application domains like wireless communications, broadband networking and multimedia applications. The methodology also provides capabilities for simulating multiple levels of abstraction simultaneously. This enables reuse of the simulation environment for functional verification of synthesizable implementation models against the abstract architecture model.

During a cooperation with Synopsys Professional Services, this methodology is integrated into CoCentric System Studio (CCSS) and applied to the development of a 2.5 GB IP forwarding chip with Quality-of-Service (QoS) support. In this paper we share our experiences with the latest SystemC 2.0 based features of CCSS, which is used as a common design platform for abstract architecture modeling, profiling and hardware implementation. During the architecture exploration phase we heavily employ the CCSS profiling capabilities to validate the performance of several architectural alternatives. Synopsys IP Telecom Workbench serves as a functional verification tool throughout the complete design process.

1 Introduction

One of the most challenging tasks in modern System-on-Chip design projects is to map a complex application onto a heterogeneous platform architecture in adherence to the specified flexibility, performance and cost requirements. Under stringent cost constraints, the required flexibility and performance is best delivered by a heterogeneous platform employing standard as well as application specific programmable architectures and dedicated hardware blocks, which are connected by a sophisticated communication topology. As a result, the designer faces a huge design space and has to compose a system architecture from various kinds of building blocks and communication resources in order to meet the constraints of the specific application.

The traditional approach comprising only two decoupled phases of textural specification and architecture implementation is no longer feasible for the design of large heterogeneous systems on a single chip, because quantitative architectural considerations are difficult to estimate on paper, prior to the implementation phase. Systems are either over-engineered, thus impacting the cost, or fail to deliver the expected performance.

Due to the high level of detail of implementation models, so far they can only be optimized locally and system architecture tradeoffs and optimizations are not exploited.
For that reason we advocate an intermediate *System Level Design* phase in the design flow, where the functionality of the system is mapped to the platform architecture in an abstract manner to enable architecture optimizations across heterogeneous computational components.

2 System Level Design Requirements

In the course of numerous design projects, we have identified the following methodical requirements in System Level Design, which are addressed by our approach.

- **high simulation speed and modeling efficiency** is mandatory to handle the high complexity of SoC designs.
- **separation of timing and behavior** allows efficient profiling of functional blocks mapped to alternative architectures.
- **separation of interfaces and behavior** according to the *interface based design* paradigm motivated by Rowson [3] enables successive communication and structural refinement as well as IP reuse.
- **incorporate hardware semantics** like reactivity, concurrency and determinism to express impact of the platform architecture.
- **seamless transition from system to gates** to avoid long iteration cycles caused by gaps in the design flow
- **intuitive visualization** comprising capabilities for system level debugging and performance analysis to enable efficient validation of the system model.

The foundation for our methodology is provided by the SystemC library [1], which is widely considered as the emerging EDA industry standard language for bringing together today's disjunctive worlds of system conceptualization and implementation. Hardware semantics as well

as interface based design are already incorporated into the 2.0 release of SystemC and also synthesis tools become commercially available. We have supplemented SystemC with a methodology specific library to enable System Level Design according to all requirements listed above.

3 Design Flow

Usually the first step in the design flow is to write a functional specification document, which specifies the intended functionality, features and interfaces as well as performance and cost requirements.

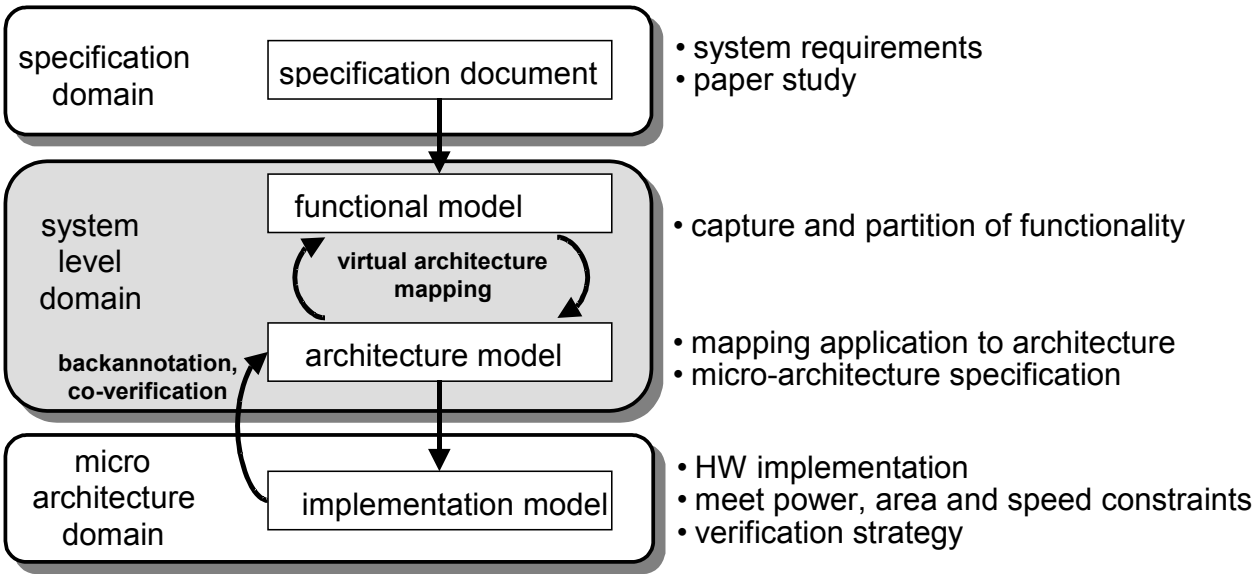

figure 1: System Level Design Flow

Our methodology [9] covers the system level design domain in figure 1, i.e. after the functional specification 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.

In the next design step, the functional model is mapped to the intended target architecture in order to create a performance model of the resulting system architecture. As explained in more detail in section 3.2, architecture 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.

3.1 Functional Modeling with SystemC

High simulation speed and modeling efficiency are achieved by modeling the hardware aspects of the system on the highest possible level of abstraction. Before modeling the hardware blocks on the highly detailed register transfer level (RTL) we create an abstract functional model.

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. Abstract Data Types (ADT) replace the bit-true data representation of the RTL model, 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 depends on the application and corresponds to the minimal duration between the arrivals of two consecutive tokens.

An example of a SystemC process network is given in figure 2 to illustrate our way of abstract system modeling. The code example of the SystemC process shows the interface of a module, where one incoming and one outgoing connection carry tokens of type IPPacket. The implementation of the class IPPacket is also depicted and shows some of the ADT elements.

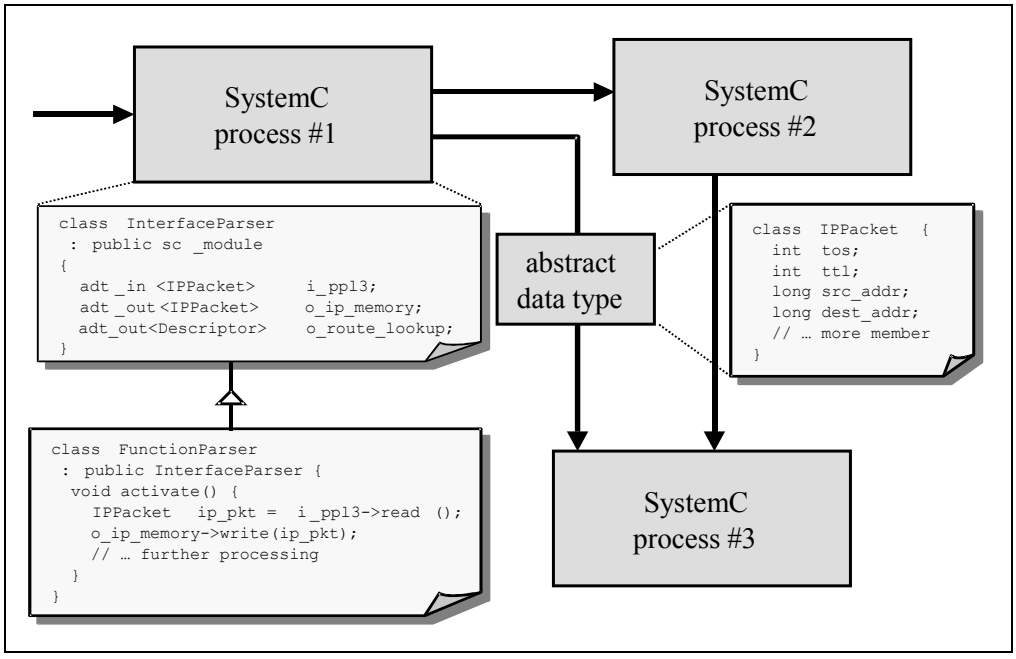

figure 2:Abstract SystemC Model

The interface class is specialized via C++ inheritance to a functional process class, which implements the functionality of the module. The functional process is activated on the arrival of every new token, reads the new token from the input port, processes the included data and writes

the generated tokens to the output ports. In that way, a strict separation between interfaces and behavior is achieved. Refinement of either the functionality or the internal structure of the module is done by inheriting from the same interface class Interface. Modification of the data elements exchanged between processes is done by adding or removing ADT elements. Both leaves the original interface definition untouched and thus preserves the compatibility between different refinement stages of one module throughout the system level design process. At the end of this stage, a network of communicating processes, which exchange ADT tokens, captures the complete system behavior. 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.

3.2 Virtual Architecture Mapping

In the next design step, the functional model is mapped 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 is preserved.

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 ADT tokens.

- Pipelined architectures are able to consume and produce a token every cycle and 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 are afflicted with an iteration interval, i.e. they are blocked for a varying amount of cycles during a 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 hierarchical SystemC 2.0 channel models for the interconnection of the modules. The channels are based on a loss-less FIFO buffer according to the discrete event communication paradigm and provide methods to annotate processing delay and iteration interval according to the above listed observations. The channels are implemented *hierarchically* (see chapter 11 of [2]), since they incorporate internal processes to implement the mechanism for timing annotation.

figure 3 illustrates the creation of an abstract architecture model from the combination of the original functional model together with the channels to capture the performance impact of the intended architecture. The right part of figure 3 shows the annotation of a processing delay t_D,

which is passed as a second parameter of the write() method of outgoing connections. Transparently the channel takes care, that this token does not arrive at the consumer process before the specified delay t_D. In the same way the, left part of figure 3 shows the annotation of an iteration interval t_I by calling the next() method of incoming connections. This causes the channel to suppress the arrival of tokens for the specified amount of cycles, which captures the effect of a resource-shared module.

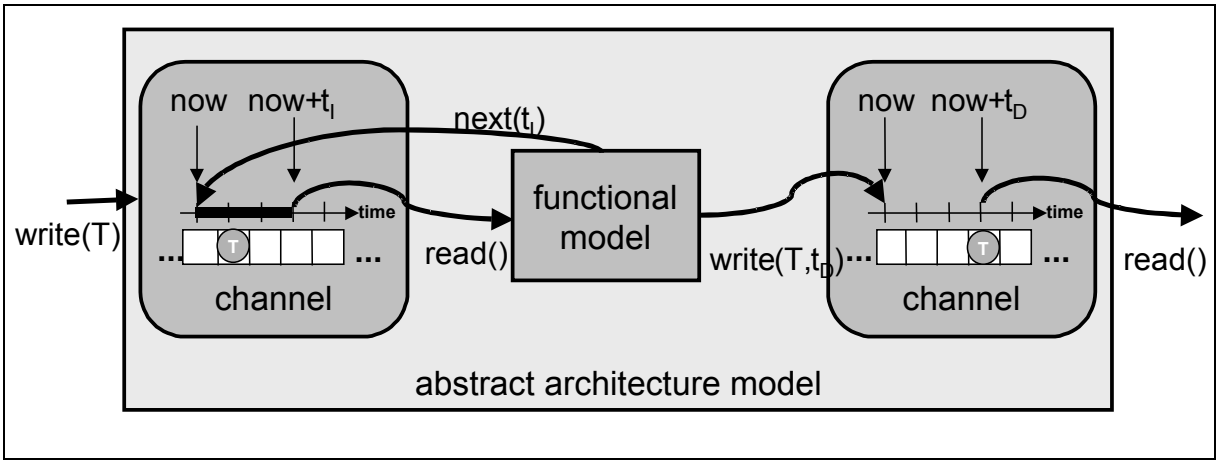

figure 3: Delay Annotation

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 consuming process has not sufficient processing power to cope with the arriving tokens, i.e. the processing power of the producing process. In this way architecture bottlenecks can be detected and eliminated very early in the design flow before the time consuming implementation starts.

3.3 Refinement and Co-Verification

After the system architecture is defined and the implementation phase is entered, the abstract model is successively converted into a synthesizable model, which realizes the same functionality on much higher level of detail. By using the CoCentric SystemC Compiler synthesis tool, the implementation phase can be performed in the same SystemC based design environment. This enables a smooth and stepwise transition from the abstract model to the synthesizable model.

Furthermore, the abstract system model can be reused as a functional verification environment for the synthesizable block implementation models. The co-verification of implementation models against their abstract reference significantly shortens the overall design time, since functional verification is the most time consuming task in the design of complex telecom applications [4]. As depicted for process 3 in figure 4, the basic idea is to plug the RTL implementation block per block into the abstract SystemC prototype, so realistic stimuli are

derived from the system context and comparison of the block implementation output against the abstract reference is performed conveniently on the higher abstraction level.

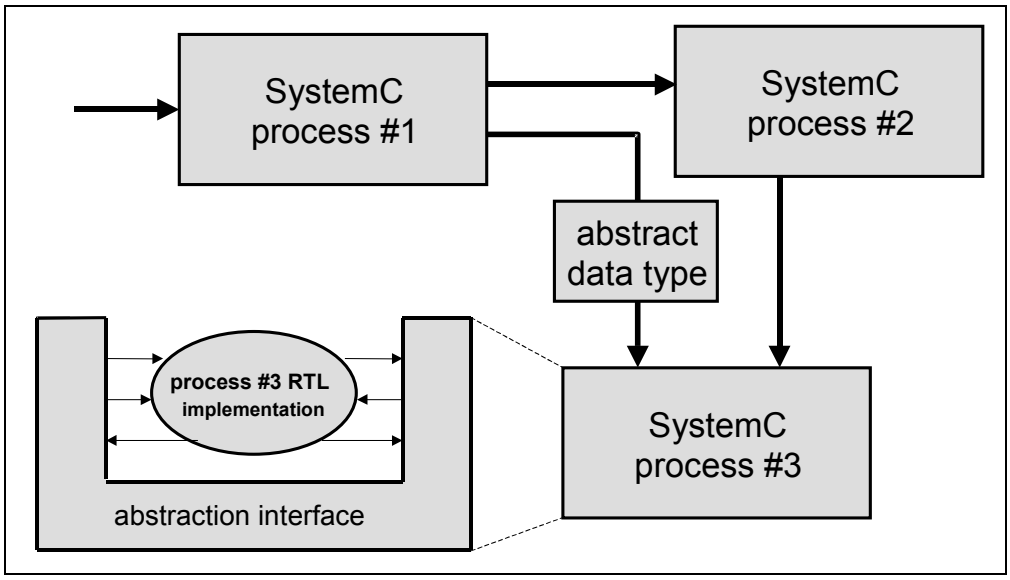

figure 4: functional co-verification environment

The key component to enable co-simulation of both abstract and synthesizable SystemC models is the abstraction interface, which is an extension of the SystemC 2.0 adapter concept (see chapter 12 of [2]). During simulation run-time, this interface bridges the different abstraction levels of data, time and protocol between the executable specification and the block implementation:

- The abstract data types are mapped to the binary format by assigning the ADT members to the respective field in their corresponding bit-true representation.
- The temporal resolution is refined from coarse-grain macro-cycles to hardware clock ticks by specifying the process activation ratio between the different time scales.
- The bus protocol behavior is added in terms of handshaking signals to feed the data stream slice by slice into the block implementation.

Concurrent execution of RTL implementation and abstract executable specification within the system context enables easy detection of implementation faults by comparing the abstract token output of both models. Together with the HDL co-simulation interface of System Studio (see section 4.4) the abstraction interface supports any VHDL, Verilog and SystemC implementation models.

4 The SystemC Design Environment

CoCentric System Studio serves as integrated SystemC design environment for all phases of the IP router development project. Although the outlined system level design methodology is based on the freely available SystemC library and therefore tool independent, we experienced significant advantages in using an integrated tool environment. In the following we will briefly

introduce the most important features of System Studio. Additionally we show, in what way System Studio can be easily customized to methodology specific needs.

4.1 CoCentric System Studio

System Studio is the environment used to model the complete system. SystemC models are captured in the System Studio environment and are organized into hierarchical libraries that can be shared among design projects. Using the design management capabilities of System Studio, the models can be organized into libraries for ease of reuse and maintainability across multiple project phases and distributed engineering teams.
Each model is stored in a database along with three different views of the model:

- A **symbol view**, which is a block and port representation of the model that can be instantiated into a block-diagram schematic.
- An **interface view** that defines and documents all port-related and parameter-related specifications for this model.
- An **implementation view** that contains the implementation of the module.

The tool generates source code from the interface and parameter descriptions of the interface so the developer can focus on the behavior of the model's implementation. The schematic entry as depicted in figure 5 allows graphical modeling of the overall structure, so the user is able to quickly specify and change the structure of the system in a very intuitive way.
In addition, other properties useful for design management are included in the model database, for example generation of html documentation, error checking and cross-highlighting of errors in the error message window, implementation and interface views, as well as integrated access to revision control systems such as RCS, SCCS, or ClearCase. However System Studio is also very cooperative with other tools, e.g. we use the open-source CVS revision control system [5], which supports Internet based access to the project repository.

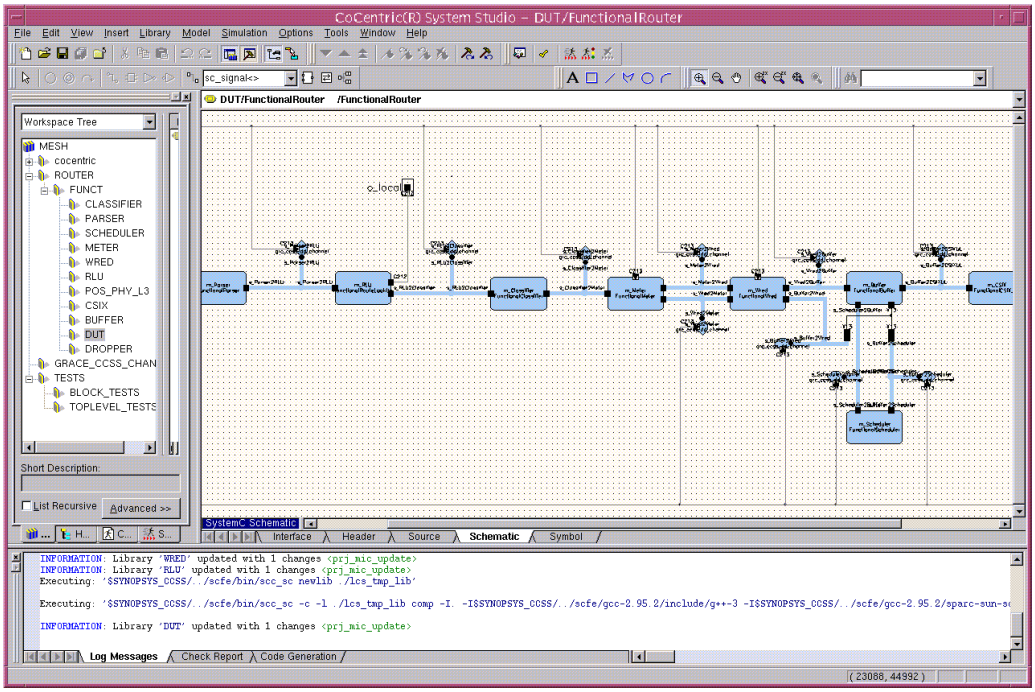

figure 5: System Studio Schematic Editor

As an extension to the original SystemC library, the SystemC simulation generated from System Studio can be controlled either interactively through the graphical Simulation Control Panel or in batch mode through a simulation script written in Tcl. Using the batch controlled mode, simulations can be run under control of a Tcl script that accomplishes a large number of simulations sweeping parameter ranges. Tcl scripts can also be used for parameter optimization. In this case, simulations are run in an iterative manner using results of one simulation run to compute new parameter values for the next iteration. Using Tcl scripts, a huge design space can be evaluated to achieve the most cost-efficient design.

4.2 System Debugging

Debugging of the initial executable system model is a very critical task, since it cannot be verified against another model but must be validated to capture the complete and correct functionality. There are two levels of debugging in System Studio: macro debugging and micro debugging.

Macro debugging occurs at the SystemC process level. During system simulation, processes are activated one by one. During macro debugging, the user can set breakpoints before (stop at) or after (stop after) one or several processes is activated. The simulation is stopped and the values for some parameters and signals can be observed at this stage. The values of the interconnections and the internal parameters are viewed according to levels. Specific values can be assembled in the Data Watch sheet and thus can be viewed jointly. When the simulation is stopped, the values of parameters and signals that are displayed with a white background are changeable; values that have a gray background are read-only.

Micro debugging means source-level debugging of the SystemC processes. A standard debugger such as gdb can be attached to a System Studio simulation. An intermediate layer is available between the standard debugger and System Studio to provide customized commands such as “sprint” to display values instead of a class for SystemC related classes. Macro- and micro-debugging capabilities offered in the Simulation Control Panel are closely integrated. For micro debugging, a “step-into” feature allows a source-code debugger to take over control of the simulation.

Furthermore, we have supplemented the native System Studio debugging features with a methodology specific debugger, which visualizes the SystemC simulation according to the Message Sequence Chart (MSC) principle. MSC tracing is a well known tracing mechanism usually employed in Specification and Description Language (SDL) based design environments like the TAU SDL suite [6]. The MSC representation provides a very intuitive visualization of a network of coarse-grain SystemC processes exchanging abstract data types (ADTs). SystemC modules are displayed by vertical lines, which are labeled with the module name at the top. Communication events are displayed as horizontal arrows between the lines of the emitting and receiving processes. The arrows are labeled with signal name, the bracketed time instance and the ADT type name. The debugger provides advanced filter mechanisms to systematically reduce the displayed data exchange to the currently interesting communication events.

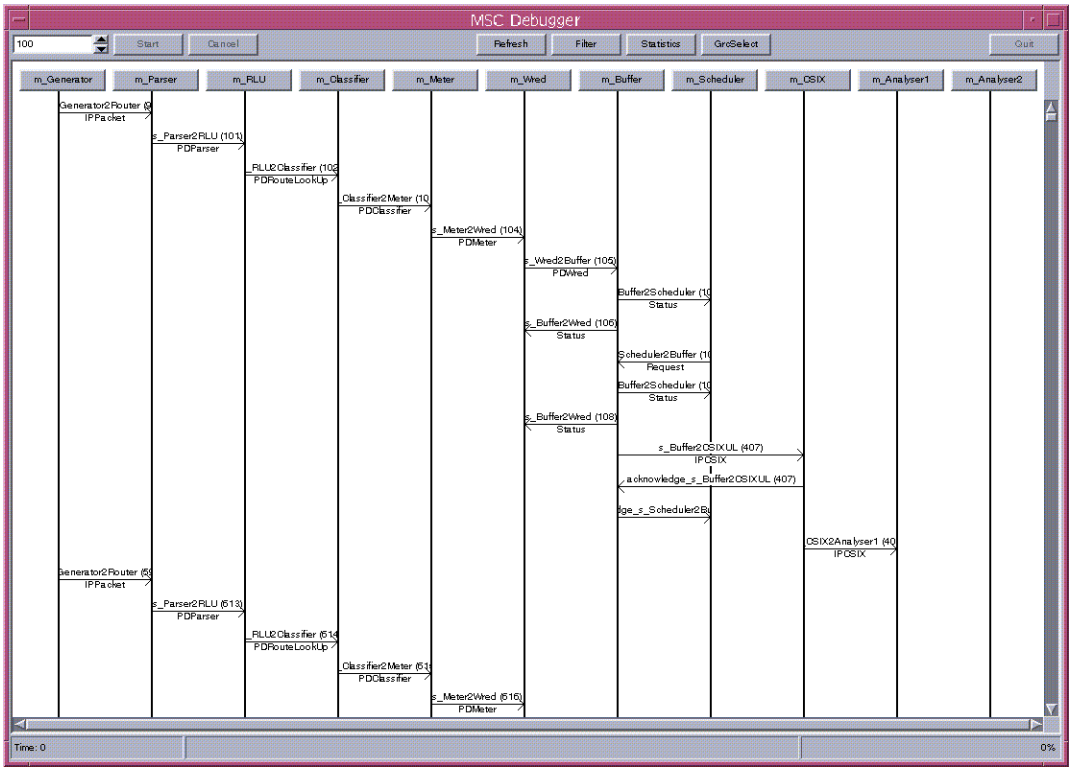

figure 6: Message Sequence Chart Debugger

Thus, debugging is achieved at all levels of granularity. First, the system level transactions are displayed in the MSC debugger. Next, cycle and process debugging is achieved through the macro debugging capabilities. Macro debugging capabilities include stop, step, continue (time),

pause, and breakpoints. Some signals can be monitored or traced. Finally, the source code of any model can be debugged in the framework of the overall simulation.

4.3 Performance Visualization

Performance is the key criterion during the investigation and the specification of the system architecture. For the proper analysis and understanding of the profiling results, the presentation of the simulation data is most important to reach design decisions. Data can be displayed using DAVIS, the Data Analysis and VISualization tool integrated in System Studio. Besides displaying values over time, data can be displayed in different formats such as scatter and eye diagrams and at logical levels. In addition, a monitor library provides additional interactive analysis and visualization capabilities, such as displaying utilization statistics of a bus as a pie chart, as vertical or horizontal bars, or as the histogram of a queue load. Objects in a design can be instrumented such that they can be watched during simulation. Any objects such as signals, ports, parameters, and in some cases local variables can be instrumented. These objects can then be dynamically visualized in the monitor

4.4 VHDL/Verilog Co-Simulation Interface

A co-simulation interface is introduced in System Studio to enable co-simulation with existing legacy code in Verilog or VHDL, and to verify the system behavior at block implementation level. Either the Verilog or VHDL code is embedded into the System Studio simulation, or the System Studio executable is slaved to any C-friendly environment. While importing Verilog or VHDL code into System Studio, clock and reset signals as well as the Verilog/VHDL and the System Studio wrapper are generated. By default, a one-to-one correspondence from input values to output values is assumed. The simulation can be called with or without the hardware debugger front-end to provide the usual Verilog or VHDL hardware environment to the user.

5 IP forwarding chip case study

In this section we will present the results of an IP forwarding chip design project, where the proposed system level design methodology has been applied in cooperation with Synopsys Professional Services. We will introduce and characterize the design and illustrate the virtual mapping methodology.

5.1 Project overview and characterization

The major objective of this project is the evaluation of the outlined system level design methodology with respect to the following requirements:

- Ensure performance as early as possible in the design flow to avoid late iterations
- Serving as a functional reference model of the system down to implementation
- Evaluate complex interactions between protocol layers
- Minimize cost for a given performance
- Evaluate architecture options: buffer access modes, speed, bus width, shared resources, data structures
- Evaluate complex algorithms, e.g. scheduling, WRED (weighted random early detection)

5.2 IP Router functional specification

In this section, the essential components of an ingress IP edge router and traffic manager design are elaborated. The functionality is focussed on IP layer, and protocol layer 2 functions are deliberately left out. Although the models considered are moderately complex, the overall purpose of the project is a feasibility study for the methodology and not a product quality chip design.

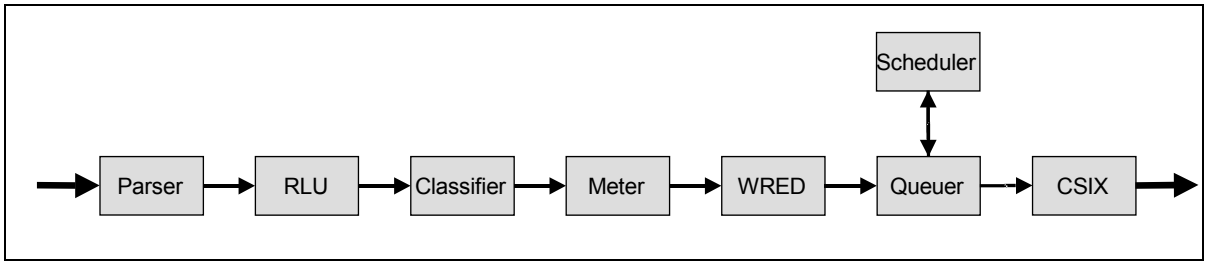

figure 7: functional view of IP forwarding chip

figure 7 displays the functional blocks of the IP router together with the basic inter-block communication:

- The Parser unit performs check of incoming packets and provides following units with packet descriptors, which hold all the relevant header information of the respective IP packet.
- The Route Lookup unit performs forwarding of IP Packets based on the longest match table search algorithm. The forwarding decision is based upon destination IP address and the contents of memory storing the routing table.
- The Classifier classifies incoming packets into Classes of Service (CoS), so the packets are processed according to their negotiated Quality of Service parameters by the following blocks.
- The Meter unit measures the IP packet rate and drops packets exceeding the negotiated traffic characteristics to protect the succeeding queuer unit from unfriendly traffic streams.
- The WRED unit also drops packets according to the weighted RED algorithm [7] to avoid throughput degradation due to congestion.
- The Queuer unit stores IP packets according to their class of service until they are forwarded to the CSIX unit
- The Scheduler unit decides on the basis of the priority and the actual fill status of the packet queues in the queuer unit.
- The CSIX unit segments IP packets into fixed size packets according to the standardized CSIX bus protocol [8] to interface the switch fabric.

According the methodology described in section 3, first a functional SystemC model is created from the functional specification document. As depicted in the System Studio snapshot in figure 5, the structure of the functional SystemC model matches the block diagram in specification document. Also the data flow through the model as displayed in the MSC debugger in figure 6 is straight forward except for the communication between the Queuer and the Scheduler units. This functional model is well suited to validate completeness and functional correctness, but does not yet impose any assumptions on the architectural realization.

5.3 Architecture Refinement and Exploration

The successive steps of mapping the functional SystemC model to a virtual architecture are explained with special emphasis on how the CCSS performance visualization features are used to drive architectural decisions and to define the final system architecture. In general, the architecture exploration can be separated into sub-phases, where the modeling granularity and accuracy is successively refined.

System Architecture
Given the constraints on latency, the architect can decide to have a more parallel than pipelined architecture. For instance, the classification can be performed in parallel with the route lookup. To model this effect, the top-level structure of the router does not need to be modified. Only the timing annotation changes in the route lookup and the classifier.

Block-Level Timing Budgets
An initial timing budget is defined for each module that is based on experience. Timing budget includes both pipeline delay and iteration interval. Based on simulations, the architect may relax some constraints, choose to tighten some others, or find out that a budget is required that is not realistic for a module. In the latter case, the architecture will be modified, e.g. resources are added or the algorithm is changed. Final budgets provide a requirement specification for later RTL implementation. The statistic evaluation of the channel library is used to detect bottlenecks in the system. The histograms depicted in **figure 8** show the number of tokens stored on the channels during a simulation run. The striped histogram shows a high incidence of long FIFO queues, which indicates a throughput bottleneck in the consumer block of the “pd_parser” channel: the consumer processing power does not catch up with throughput of the respective producer block.

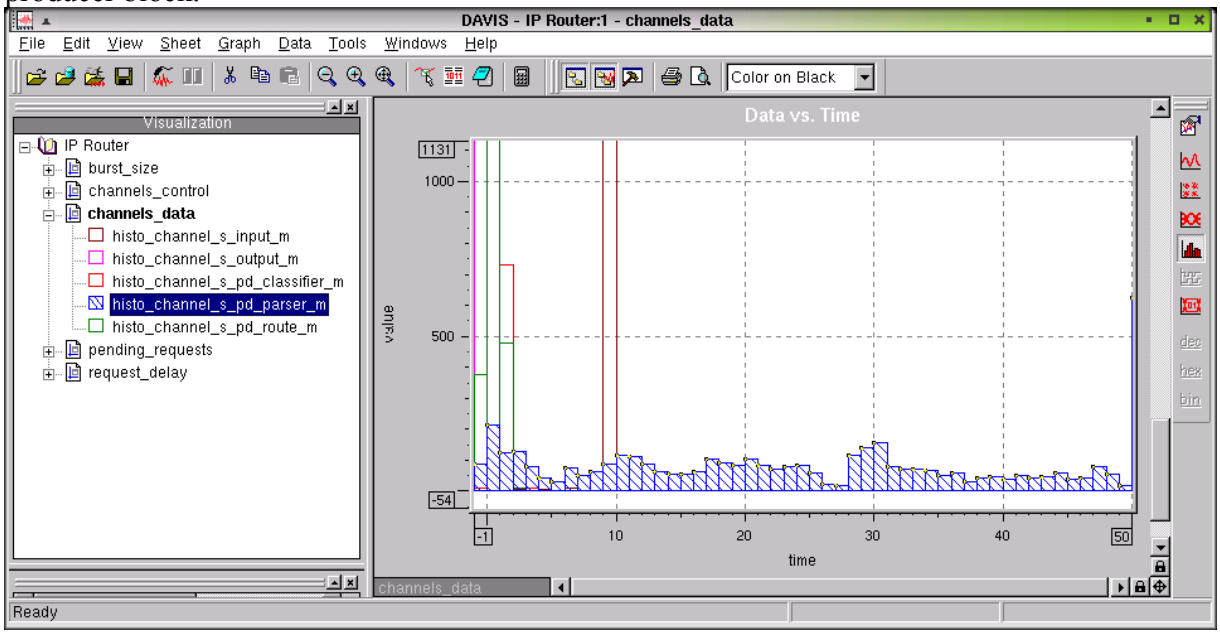

figure 8: channel statistic

A well-balanced system architecture provides sufficient processing power for all components, such that the FIFO fill level of all channels remains within moderate limits.

Algorithmic Exploration
Several scheduling schemes can be evaluated with high simulation performance, since the simulation is essentially running on a packet clock. By that the algorithmic performance can be evaluated in a realistic system context with respect to throughput, processing requirements and Quality of Service properties.

Block-Level Macro Architecture
Shared resources like a bus or a memory can be modeled at this stage. For instance as depicted in figure 9, most modules in the data path could make use of a shared packet descriptor memory instead of dedicated point-to-point communication. This would certainly reduce register cost, but would probably be not sustainable using a single bus/memory architecture. A trade-off between memory and register usage can be found by quickly evaluating a few options.

Block-Level Micro Architecture
Several options for the route lookup algorithm can be evaluated. They are tightly linked with the associated RAM usage (CAM, SRAM…). These memory components need not be modeled explicitly, but only the respective timing behavior (access time, latency…) is taken into account in the way the timing annotation is performed in the route lookup module. Should the route lookup module be reused in another system, the different implementation could be reused and only the interface would need to be updated.

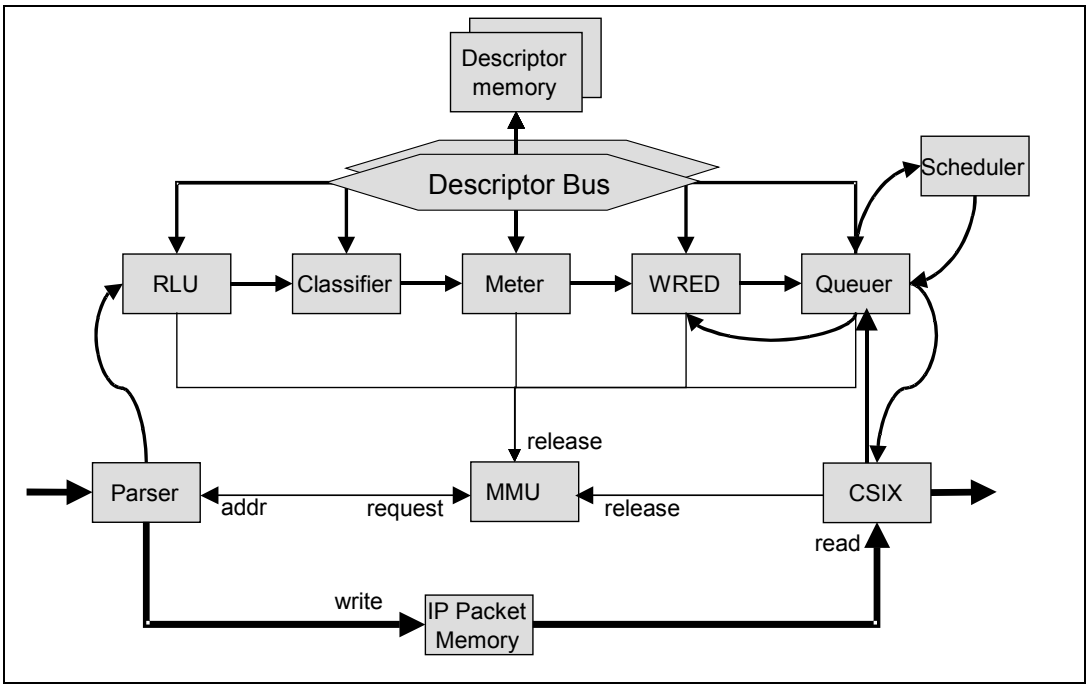

figure 9: abstract architecture view of IP forwarding chip

6 Conclusion

In this paper, a system level design methodology based on the SystemC library is presented. The outlined approach is well supported by the CoCentric System Studio development environment and capable to capture the complete system functionality as well as all performance relevant architecture features on the highest possible level of abstraction. The resulting modeling efficiency measured in lines of code and the simulation speed is about two orders of magnitude better compared to an RTL architecture model.

During a research cooperation with Synopsys Professional Services, the outlined methodology has been applied to the architecture conceptualization of an IP forwarding chip. By employing the proposed system level design methodology, a team of 3 engineers was able to demonstrate the feasibility and define a scalable and cost effective architecture within 2 months without sacrificing the algorithmic performance. The resulting system architecture model also serves as an executable specification and as a fast co-verification environment for the HW implementation.

In the second project phase, the HW/SW co-design aspects of the methodology will be applied to the IP Router case study. Here we will investigate the mapping of several computational tasks to Application Specific Instruction-set Processor (ASIP) cores to improve the system flexibility. The ASIP design will be performed using the LISA Processor Design Platform [10].

7 References

[1] SystemC initiative, http://www.systemc.org

[2] “Functional Specification for SystemC 2.0”

[3] J.A. Rowson, A. Sangiovanni-Vincentelli, “Interface-Based Design", DAC 1997

[4] A. Silburt et.al. “Accelerating Concurrent Hardware Design with Behavioural Modeling and System Simulation.” DAC 1995.

[5] Concurrent Versions System, www.cvshome.org

[6] Telelogic TAU, www.telelogic.com

[7] Random Early Detection Gateways for Congestion Avoidance, Floyd/Jacobson, 1993

[8] CSIX “Common Switch Interface Specification”, www.csix.org

[9] The GRACE++ Project, www.iss.rwth-aachen.de/Projecte/grace/index.html

[10] LISA Processor Design Platform, www.iss.rwth-aachen.de/lisa