Fit the hardware to the algorithm with SystemC models

Bo Wu, CoWare, and Jens Reinecke, RWTH Aachen University, Germany - October 10, 2006

System-on-chip (SoC) designs commonly consist of one or multiple processors (e.g. DSP or reduced instruction set computing (RISC) processors), interconnects, memory sub-systems, DSP hardware accelerators, and peripherals such as direct memory access (DMA) controllers and memory management units (MMU). In order to cope with the complexity of such a design, engineers must perform several recurring design tasks. These include creation of an executable specification, architecture exploration, embedded software development, and hardware-software verification.

The traditional design flow is a sequential flow, where all design stages are separated. The algorithm designer finishes his or her work and delivers models of the DSP system and specifications to the hardware designer and the embedded software developer. Starting from these specifications, hardware and software development begins—almost from scratch—without the further benefit of the algorithm designer's experience. It is obvious that this traditional flow has many disadvantages including:

- The software developer has to wait for a first hardware prototype to implement his tasks.
- The software developer cannot take hardware features into account until late in the design process. This is unacceptable, because interconnects and memory architectures should be taken into account early in the design process. At minimum, they should be considered when partitioning the algorithm into hardware and software components. The algorithm designer might even need to take them into account when defining functional modules.
- The entire platform can be tested sufficiently only the first prototype is ready. Consequently, problems or inefficiencies are identified late in the design process. This can lead to a need to rebuild the prototype, a compromise in system performance, or even a re-spin of the chip.

These problems can be avoided by involving the algorithm designer in the early stages of the architecture design. This is enabled through electronic system level (ESL) design approach. In order for designers to accept an ESL-based approach, there must be an efficient and intuitive methodology for modeling complex platforms. This article introduces such an approach using a SystemC-based virtual hardware platform methodology.

Platforms are conceived in SystemC

Originally, SystemC was conceived as a means for implementing register-transfer-level (RTL) concepts in a C++ class library. Today, SystemC is used mainly for building models of a SoC platform early in the design, or as part of the verification flow. In both cases, SystemC provides easier ways to build models at higher levels of abstraction than any of the traditional HDLs. Some of the main concepts of the SystemC language are:

- Modules: Modules are functional components. They implement (concurrent) behavioral processes by describing certain functionality. Such a module can be, for example, the implementation of a
DSP sub-system.

- **Interfaces**: Interfaces contain declarations of functions for communication between modules. Interfaces do not implement any functionality; they only specify the communication model.

- **Channels**: Channels are the functional complement to an interface: The declared communication functions are implemented inside a channel. Hence, it can be said that a channel "connects" modules. Channels may be of different complexity, and may implement different types of protocols. When porting an algorithm to a SystemC platform, a designer may start from simple point-to-point channels.

- **Ports**: Ports are part of a module. They allow a channel to be connected to the module. Port declarations are provided with interfaces, which tell the module about the allowed communication functions associated with a port. By changing the interface for a port, the module may access another type of communication protocol.

*Figure 1* shows a simple example of a SystemC model. The system uses a so-called SystemC FIFO (sc_fifo()) channel, which behaves like a FIFO and provides some simple read and write functions for communication. The rest of the system consists of two SystemC modules. On the left-hand side we have a module modeling a transmitter. A receiver module is shown on the right-hand side. Both of the modules own a sc_fifo() port, which connects the modules to the channel. To forward data from the transmitter to the receiver, the transmitter module writes them to the sc_fifo() if there is free space inside. If data are available, the receiver reads them from sc_fifo().

![Simple Example of a SystemC Model](image)

1. **Simple Example of a SystemC Model**

The sc_fifo() channel is a simple channel, which can be used to model unidirectional point-to-point connections. It is a good channel to use when porting an algorithm to SystemC and performing the initial functional verification. An algorithm designer implementing an algorithm within a SystemC module does not have to care about the communication in detail. He just has to check the FIFO's status and calls the ports' read or write functions. The CoWare flow supports this simple protocol as one of the protocols used to connect DSP subsystems with a platform model.

A main differentiator in SystemC compared to traditional hardware description languages is transaction-level modeling (TLM), which offers a high abstraction of inter-module communication. It has four use cases: functional view (FV), architects view (AV), programmers view (PV), and verification view (VV). (See the reference at the end of this article for details.) The FV is the use case that algorithm designers can be directly involved in. The software and hardware designers can use the other three views. **SystemC Enables Early Algorithm Integration**

Our algorithm-to-platform design flow is structured as depicted in *Figure 2*. First, the algorithm designer implements the DSP system using CoWare's Signal Processing Designer (SPD), formerly known as SPW. For that purpose, algorithmic blocks may be assembled using SPD's C-based library blocks or imported from MATLAB into SPD (among other approaches).
2. Proposed Algorithm to Platform Design Flow

After exploring the performance of the DSP system, including all fixed-point quantization effects, the algorithm designer will arrive at a reference model of the targeted DSP system. The algorithm designer now starts to partition the model working with the hardware and software designers. The resulting hardware-mapped sub-systems are used to generate platform component models in the form of SystemC modules. These platform component models can be inserted into a virtual hardware platform as DSP hardware accelerators. In CoWare's design flow, this is done using the Platform Architect environment.

Simulating and analyzing the virtual hardware platform provides information on the quality of the partitioning and the chosen hardware architectures. If the results are not optimal, the partitioning in SPD, the selection of interconnects and memory architectures, etc. can be iteratively improved in the Platform Architect environment.

Because the generation of the DSP hardware accelerator models is an automated process, short iteration cycles can be achieved. DSP peripheral blocks from SPD can be partitioned, configured and generated in under an hour. In contrast, writing, rewriting and verifying manually-written SystemC code can take several days and require expert knowledge of SystemC. Once the virtual hardware platform meets the design goals, the hardware designer and the embedded software developer can start from a known, golden specification. The hardware designer has a starting point for the RTL implementation of the platform. The embedded software developer receives a developing and debugging platform much earlier than in the traditional flow. The algorithm-to-platform design flow can be subdivided into the following four flow phases.

**Partitioning**

The term “partitioning” describes the process of subdividing a complete system (called a “System View” in SPD) into smaller sub-systems (called "Detail Views" in SPD). Additionally, decisions are made about the upcoming mapping of the algorithm to hardware and software. SPD's Detail Views...
are the basis for generating DSP hardware peripheral models from SPD models. An example is shown in *Figure 3*.

**Platform Component Generation**

A prepared Detail View can be easily exported as a SystemC module and used as a platform component in Platform Architect. During the export, a number of configurable parameters are available, including:

- **Export Parameters**: These are used to configure the export session itself, e.g. the type of SystemC wrapper that will be generated.
- **Design Parameters**: These are used to select parameters that will be accessible after the export. Depending on the generated SystemC wrapper, they are provided as constructor arguments and/or as a part of a possible register interface of the wrapper.
- **Design Probes**: Probes can be attached to any port of a block in a Detail View. They can be used for debugging and verification purposes.

The generated DSP hardware peripheral model contains the highly optimized SPD data flow simulation executable, wrapped into a SystemC transaction level model, compliant with the SystemC Modeling Library (SCML) modelling standard. SPD simulation executables are designed to be used for simulating large DSP systems. As a result, the DSP peripheral model is efficient, even if the designer encapsulates large hierarchical designs.

The "Programmers View Transaction Level Modeling (PV TLM) Bus Wrapper" represents a Programmer's View target peripheral as shown in *Figure 4*. It offers a memory-mapped register interface. This register interface is based on a SCML memory model. Through this interface, data can be easily read and written. It also allows the designer to control the platform component.
4. Transaction-level bus wrapper

The automatically generated transaction-level bus wrapper owns two ports:

- A PV Target Port, which is connected to the platform's bus (if necessary, via a transactor) and linked to the wrapper's register interface
- An Interrupt Port, which can be directly connected to an interrupt controller

The automatically generated transaction-level bus wrapper's register interface consists of a set of 32-bit registers. These registers may be subdivided into three categories:

- Control Registers: They provide control of the wrapper. Through these registers, the embedded software developer can configure and control the wrapper and read out information on the module.
- Port Registers: These can be subdivided again into two types for each port of the original SPD model. There are registers for accessing a port's internal data buffer and registers providing information on a port's data type.
- Parameter Registers: These can be used to specify exported design parameters.

Besides the transaction-level bus wrapper, a SystemC FIFO wrapper can be generated separately or mixed with the transaction-level bus wrapper.

**ESL Platform Assembly**

It is easy to incorporate an exported model into a Platform Architect platform. The SPD export automatically generates a component library, which includes the generated module. This makes the module available within Platform Architect like any other IP library block.

**Embedded Software Development**

The final phase is the development of the embedded software. This includes implementation of algorithmic tasks as well as writing drivers for the exported SPD models. In case of the transaction-level bus wrapper, the software drivers perform the model's initialization and configuration as well as data flow control and synchronization. To ease the development of these drivers, the SPD model export automatically generates a skeleton for the software driver.

**Example: Wireless Platform Design**

We applied the proposed flow to an existing and fully verified SPD system: the Decoder Sub-System of the "WCDMA Downlink System" delivered with SPD's 3GPP WCDMA Library. The test case showed us that the described methodology can be performed in a short period of time. It also allowed the system architect to stay focused on important design challenges, rather than keeping busy with detailed modeling. *Figure 5* shows the chosen partitioning of the DSP system.
5. Partitioned Decoder Sub-System of the "WCDMA Downlink System"

The partitioned system included an exported test bench, six hardware-mapped sub-systems, and three software-mapped sub-systems that provide some control-like functionality.

This system was mapped to a virtual hardware platform that consisted of an ARM968E-S processor model (mainly used for data flow control purposes) and elements from CoWare's Generic IP Library (DMA controller, interrupt controller, and a shared memory for the storage of intermediate results). Figure 6 shows a virtual platform in Platform Architect, including the WCDMA Decoder Sub-System at the bottom of the diagram.

The platform used an Open Core Protocol (OCP) Bus interconnect. Other than that, we applied a point-to-point connection between one port of "Testbench" and one port of "Second Interleaver." (Figure 5 shows both blocks. In Figure 6, "Testbench" is shown as a separate block, but "Second Interleaver" is hidden inside the WCDMA Decoder Sub-System.) As the test bench merely delivers test data to the Decoder sub-system, it was appropriate to use an sc_fifo() connection here. By configuring the sc_fifo() connection to model zero latency and by not connecting the corresponding port to a bus, we achieve the intention of a test bench. There is no delay caused by the input data and there is no extra load on the bus. Additionally, this type of connection does not need any extra software drivers on the data flow controlling processor.

6. Virtual Platform in PA with 3GPP Decoder Sub-System exported from SPD (at the bottom)

During the export of design parameters, we were able to specify whether "Testbench" was a fixed or a random source. We were also able to specify the source file for the fixed signal source.

The platform depicted in Figure 6 was originally designed to model a system performing video/image
We added the WCDMA Decoder Sub-System as a sub-system to the platform. Now the platform "receives" the incoming data (as a signal source file for the "Testbench") and decodes them within the rest of the exported SPD models. The decoded data are converted and then displayed and sent to the corresponding module. The output data of the "CRC Fail" port are used to count the erroneous incoming data blocks. The number of these blocks is continuously monitored and updated within a block that was added from SPD’s Interactive Simulation Library (ISL).

Figure 7 shows what can be observed during simulation. The lower right shows the "received" and decoded image. Each time new data blocks are received, the corresponding pixels are added to the displayed picture and the block counter is increased within the ISL table above the display.

Summary
Due to today’s increasing complexity of digital signal processing systems, a design flow is needed to efficiently explore hardware solutions, so that the platform can fit the algorithm. The required flow has to be iterative with short design cycles to converge on an optimal solution in an appropriate amount of time. The platform design flow presented in this article meets these requirements.

The proposed design flow is intended to replace the time-consuming and sequential traditional design flow. After the first virtual platform is built, the iterative exploration can be done quickly. Both the hardware designer and the embedded software developer can further use the resulting virtual hardware platform to start their work almost simultaneously. By relaxing the dependencies they had on each other, time-to-market is reduced significantly.

References

About the authors
Bo Wu is a Senior Staff Solution Specialist for DSP Solutions at CoWare Inc. DSP Solutions span the entire range of technologies CoWare offers for modeling, simulation, and implementation of digital signal processing systems. He has over ten years of industrial experience in wireless system design areas and worked for Nortel Networks, AT&T Wireless, and Cadence Design Systems prior to joining CoWare. Bo holds Bachelor and Master degrees from Tsinghua University, Beijing, China, and Ph.D. degree from University of Victoria, BC, Canada. He can be reached at bwu@coware.com.
Jens Reinecke is with Electrical Engineering and Information Technology at the RWTH Aachen University, Germany. He is currently working on his Master Thesis at the Institute for Integrated Signal Processing Systems (ISS). Jens is experienced in ESL modeling and MP-SoC design through his studies at the ISS and his internship at CoWare participating in the development of the algorithm-to-platform design flow. He can be reached at jens.reinecke@rwth-aachen.de.