Automatic Generation of Virtual Prototypes

P. Belanović, M. Holzer, B. Knerr, and M. Rupp
Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Vienna, Austria
{pbelanov,mholzer,bknerr,mrupp}@nt.tuwien.ac.at

G. Sauzon
Infineon Technologies
Operngasse 20b, 1040 Vienna, Austria
guillaume.sauzon@infineon.com

Abstract

Virtual prototyping as an embedded system design technique has the potential to significantly increase efficiency of the design process. An environment for automatic generation of virtual prototypes (VPs) directly from algorithmic-level descriptions is presented here. It is implemented as part of a unified design methodology and produces VSIA compliant VPs. When applied to an industrial design flow of a UMTS receiver, this environment for automatic generation of VPs produced significant speedups over traditional manual VP creation, with savings in the order of hundreds to thousands of person-hours.

1. Introduction

Complexity of modern embedded systems, particularly in the wireless communications domain, grows at an astounding rate. This rate is so high that the algorithmic complexity now significantly outpaces the growth in complexity of underlying silicon implementations, which proceeds according to the famous Moore's Law. Furthermore, algorithmic complexity even more rapidly outpaces design productivity, expressed as the average number of transistors designed per staff/month [12, 7]. In other words, current approaches to embedded system design are proving inadequate in the struggle to keep up with system complexity.

Hence, a number of new system design techniques with potential to speed up design productivity are intensively researched. One of these techniques known as virtual prototyping speeds up the design process by enabling development of hardware and software components of the embedded system in parallel.

Development of a comprehensive design environment for automatic generation of virtual prototypes (VPs) from an algorithmic-level description of the system is presented here. Section 1.1 describes the concept of a VP in closer detail, including a review of related work. The design environment for automatic generation of VPs is presented and described in detail in Section 2. Results of applying this automatic environment in an industrial product development process are shown in Section 3. Finally, conclusions are drawn in Section 4.

This work has been funded by the Christian Doppler Pilot Laboratory for Design Methodology of Signal Processing Algorithms.

1.1. Virtual Prototype Concept

System descriptions at algorithmic level contain no specific implementation details. Hence, before implementation of the system can begin, the algorithmic description is partitioned, i.e. each component in the description is assigned to software or hardware implementation.

Traditionally, implementation of hardware components proceeds from this point. Development of software modules, however, can begin only once all required hardware design is complete. This is due to the fact that the design of software components must take into consideration the behaviour of the underlying hardware. Hence, a significant penalty is incurred in the length of the design process. Virtual prototyping is a technique which can eliminate most of this penalty and thus dramatically shorten the development cycle, as shown in Figure 1. Furthermore, such design effort savings are achieved with every iteration of the design cycle, i.e. revision of the algorithmic-level description.

A VP is a software model of the complete system, fully representing its functionality, without any implementation details. In this work we consider VPs which additionally include full definitions of hardware/software interfaces found in the system, including the required architectural information, but still no details of the actual implementation of any component. Hence, such VP components can be used to mimic the hardware/software interface which will be found in the final implementation, as shown in Figure 2. This enables parallel development of the component's hardware implementation and the development of software compo-

Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping (RSP'04)
1074-6005/04 $20.00 © 2004 **IEEE**

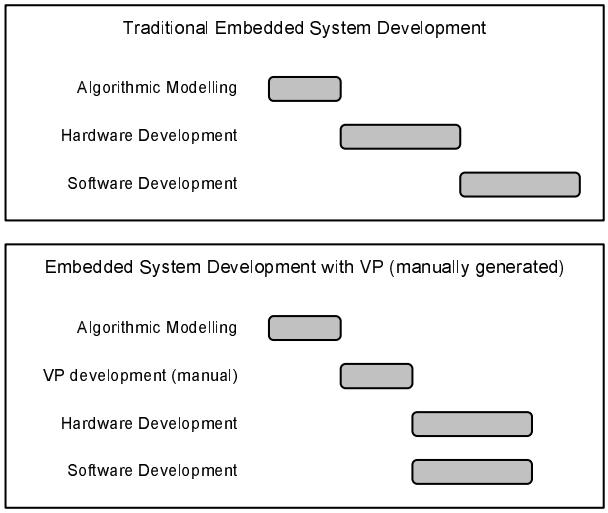

Figure 1. Shortening of the design cycle by the VP technique

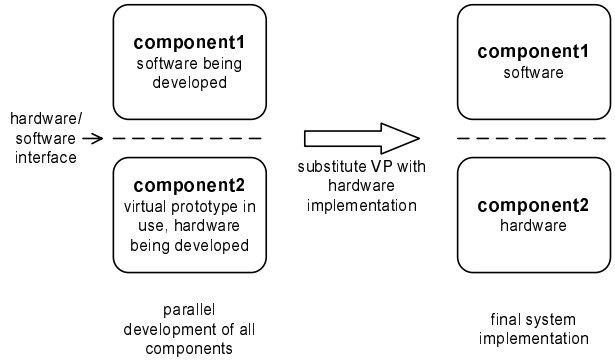

Figure 2. System development using a VP

nents which interact with it in the final system. Once both software and hardware development is completed, the VP component is simply substituted with the real hardware implementation which provides the same behaviour at the hardware/software interface.

Virtual prototyping offers numerous improvements to the design process. First and foremost, it allows parallel development of *all* components in the system, resolving all interface dependencies. Furthermore, it allows testing of software components which interface with hardware against the known hardware/software interface. Finally, a VP allows verification of the hardware implementation itself, making sure the hardware indeed provides correct interface to external components as it was designed for at the algorithmic level.

Very importantly, creation of a VP for a system component requires a relatively small design effort, compared to

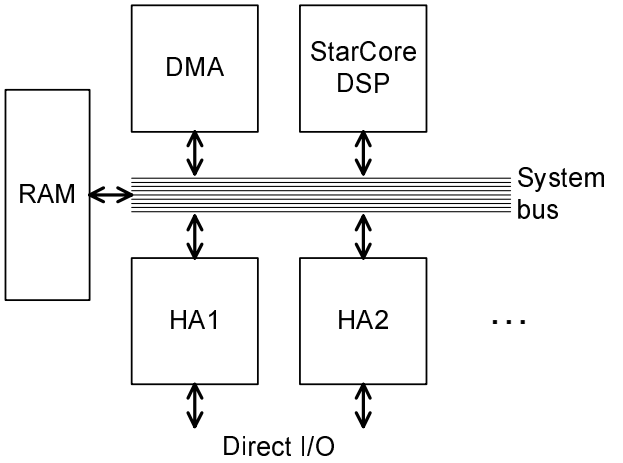

Figure 3. Target hardware platform

that of a full hardware or software implementation. This is due to the relaxed requirement of the VP to recreate behaviour only at component boundaries, allowing all other implementation details to be overlooked.

1.2. Model of Hardware Platform

The hardware platform targeted in this work is an SoC based around a StarCore DSP, as represented in Figure 3. In addition to the StarCore DSP, the platform includes a number of Hardware accelerator (HA) blocks. Just like the DSP, the HA blocks are connected through the system bus to all the other components. Additionally, HA blocks have direct I/O interfaces, which can be used to connect to components outside the embedded SoC, such as the antenna sub-system. The platform also includes a bank of Random Access Memory (RAM) for use by all the system components. Direct Memory Access (DMA) services are provided to the DSP, as well as to the HA blocks by a dedicated DMA controller, which is also connected to the system bus.

After hardware/software partitioning is performed on the algorithmic-level description of the system, the components assigned to hardware are implemented as separate HAs, whereas components assigned to software are implemented as DSP code blocks.

1.3. Related Work

Much of the current research effort in raising the efficiency of the embedded system design process focuses on introducing unified design methodologies that cover the entire design cycle [11, 8]. Such methodologies have the potential to increase design efficiency, reduce time to market and improve quality of the embedded system product. The design environment for automatic generation of VPs which

is presented here has been developed as an integrated part of just such a unified design methodology [10].

The simulation environment used in this work is that defined by the VSI Alliance (VSIA simulation interface standard) [1]. This environment is highly suitable to simulation of VPs because of its simple (non event-driven) scheduling, which is adequate for this application and produces superior simulation performance.

Previous work on VPs has for the most part focused on their use, in the hardware/software co-simulation of the embedded system [4, 2, 6]. While these efforts are targeted towards increasing the efficiency and quality of the design process through novel modifications of the co-simulation process, they ignore the significant gains achievable by automatic generation of VPs.

The approach presented in [5] considers automatic generation of VPs and achieves a speedup in the order of 5 to 8 times that of manual VP creation. However, this VP generation environment requires, in addition to the algorithmic description of the component, a formal description of its GLObal Control, Configuration and Timing (GLOCCT), the need for which is eliminated in our approach.

Furthermore, this design environment considers only Synchronous Data Flow (SDF) models, whereas our approach extends this to Dynamic Data Flow (DDF) models, adaptable to both the dynamically configurable sample rates of all sub-modules of a VP component, as well as the changing flow of data between these sub-modules.

Also, the VP environment presented here implements the entire VP homogeneously, in automatically generated VSIA compliant C++ code, rather than separating DSP code in C and GLOCCT code in VHDL.

2. Automatic VP Generation Environment

As described earlier, design of an embedded system proceeds from the algorithmic-level description towards the system's final implementation firstly through a partitioning process, followed by the creation of a VP and finally hardware or software implementation of each individual component.

The process of VP generation is typically performed manually, through rewriting of the VP from the algorithmic-level description. However, when the VP design environment is integrated into a unified design methodology, it is possible to make VP generation a fully automated process. This helps eliminate human errors and drastically decrease the time needed to create a VP, in turn deriving maximum possible efficiency gain promised by virtual prototyping. This is illustrated in Figure 4.

The automatic VP generation environment presented here is depicted in Figure 5. The process of automatically generating a VP component from that component's

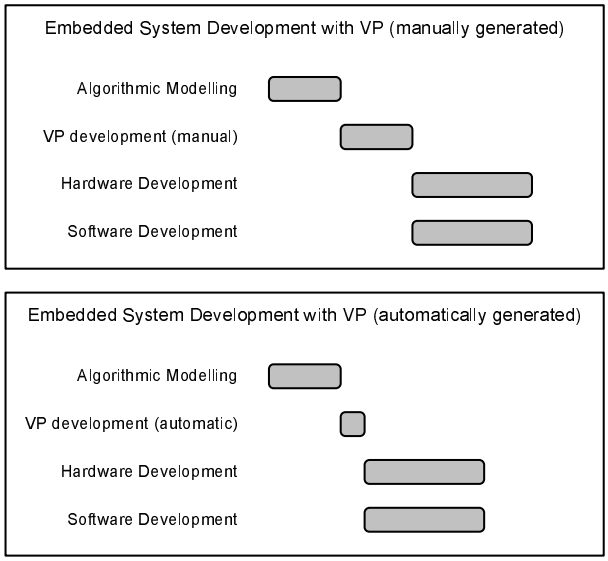

Figure 4. Shortening of the design cycle by automating VP generation

algorithmic description consists of two parts. First, the algorithmic description of the entire system (encompassing all its components) is read into the Single System Description (SSD). This also includes partitioning of the system by labelling each system component for implementation in hardware or software. The second step in the process is the generation of all parts of the VP component from the SSD.

2.1. Processing the Algorithmic Description

The environment for automatic generation of VPs presented here is based on processing algorithmic descriptions created in the COSSAP environment. Nevertheless, the VP environment is in principle independent of languages and tools used for algorithmic modelling and can, due to its modular structure, easily be adapted to any language or tool.

COSSAP descriptions contain separate structural/interconnection and functional information. The structural and interconnection information in the COSSAP description is VHDL-compliant and is read into the SSD by the System Description Interface (SDI). The SDI comprises a VHDL-compliant Parser module as well as a Scanner module which manages the database structure within the SSD.

The functional information in COSSAP descriptions is written in GenericC (extension to ANSI C proprietary to the COSSAP environment) and formatted in accordance with specific guidelines. These guidelines ensure compatibility of the GenericC code with tools in the second phase of

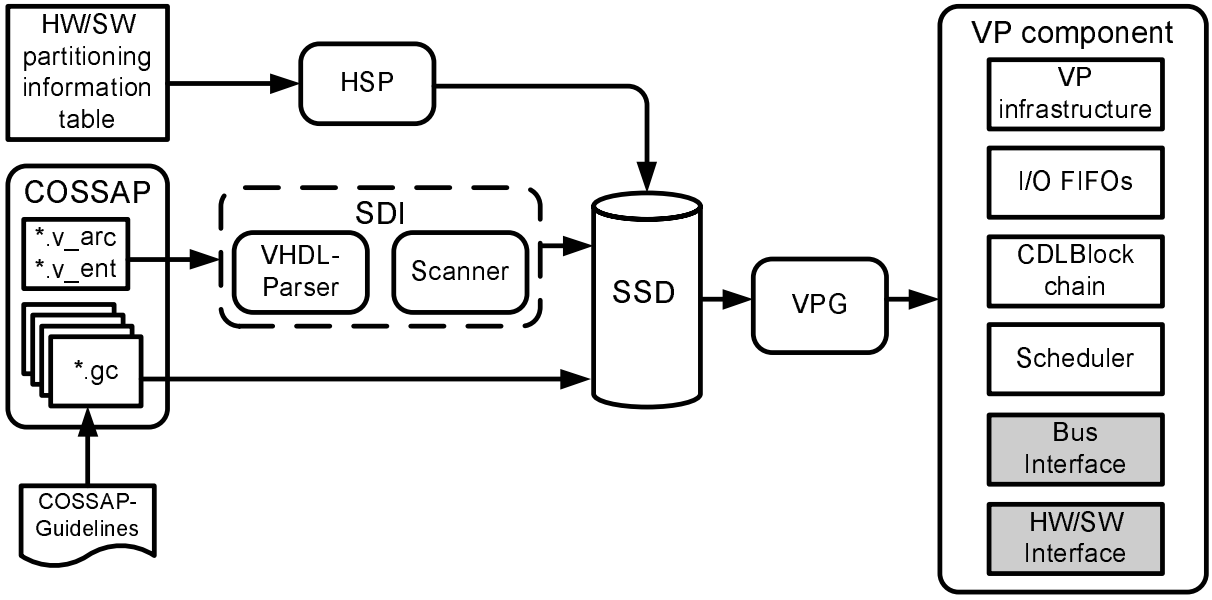

Figure 5. Design environment for automatic generation of VPs

the automatic VP generation. Suitably formatted functional component descriptions are placed directly into the SSD.

After the complete algorithmic system description is processed into the SSD, it is necessary to perform hardware/software partitioning before VP components for all hardware components can be generated. This task is performed by the Hardware/Software Partitioning (HSP) tool. Manually created hardware/software partitioning decisions, stored in textual form, are integrated into the SSD by the HSP tool. Once system partitioning is performed, the first phase of the VP generation process is complete.

2.2. Virtual Prototype Generation

A VP component is composed of several parts, as shown in Figure 5. VP infrastructure is a class library containing all elements needed to construct a VP component. First-In First-Out (FIFO) blocks facilitate transfer of data within a VP component, between its sub-modules. These FIFO blocks have capabilities of writing data passing through them into test-vector files, as well as inserting data into the VP component from similar test files. Hence, these modified FIFO blocks enable testing not only at VP component boundaries, but also within them.

CDLBlock chain is a set of all sub-modules of a VP component, interconnected to recreate the original structure found in the algorithmic description. Each VP component contains a Scheduler, which governs the execution of all sub-modules, while maintaining the flow of data through the FIFO blocks. Finally, each VP component contains two elements which are by definition application-specific and need to be created manually. The first of these is the interface to the system bus which connects all system components (see Figure 3). The second manually created element is the hardware/software interface block, which controls the transfer and packaging of data between the software running on the system's CPU and the VP component.

The second phase of automatic VP generation is performed by the Virtual Prototype Generator (VPG) tool. This tool extracts all necessary structural information for the particular component from the SSD and creates the CDLBlock chain accordingly, using the VP infrastructure and FIFO blocks. Relevant functional information in the SSD is code-styled to be compliant with the VSIA standard and the C++ language and is then integrated into the VP component. Following these steps, the automatically created VP component can be manually customised to a particular system bus, CPU and communications protocols, before being used.

3. Results

The environment for automatic generation of VPs presented here has been applied to an industrial design flow of a UMTS receiver in order to estimate its performance benefits. To this end, performance of the automatic VP generation has been compared to that of the usual manual creation of VPs. Both processes start from a completed algorithmic model in COSSAP and result in a fully functional VP in complete simulation with bus and CPU models.

As mentioned in Section 2.1, COSSAP descriptions contain separate structural/interconnection and functional information. Hence, the design effort (measured in person-

Component	*Structural*	*Functional*	*Total*
DPE	8	25	33
SYNC	17	39	56
DUD	12	43	55

Table 1. Design effort for manual VP creation

hours) for VP creation is composed of processing the structural/interconnection part of the description, processing the functional part of the description and creation of the interface to the system bus. The first two steps can be performed manually or automatically, whereas the interface to the system bus has to be created manually in both approaches. Hence, the design effort for the interface to the system bus has not been taken into account in this comparison.

Design effort for the manual approach is shown in Table 1, with all values in person-hours. The three components for which the results are presented, Delay Profile Estimator (DPE), Synchronisation (SYNC) and Decoding of User Data (DUD), are all parts of the complete, industrially developed UMTS receiver. As can be seen from these results, design effort varied between components, due to their various complexities and code lengths.

Automatic generation of VP components for each of these system blocks took a negligible amount of time (in the order of several seconds). Hence, automatic VP generation produced savings equal to the values shown in Table 1, in total exceeding 140 person hours for these three components.

However, with each new revision of the system description at the algorithmic level, the VP generation process has to be repeated. In this industrial UMTS receiver design flow, some system components have well in excess of 50 revisions. Hence, taking into consideration the revision cycles, total savings achieved by the automatic VP generation environment are expected to reach thousands of person-hours.

4. Conclusions

The automatic environment for VP generation presented here has been successfully applied in an industrial design flow, showing significant speedup in creation of VPs, with savings in the order of hundreds to thousands of person hours. Simultaneously, this approach also eliminates human-related errors, thus improving design quality. Additionally, work presented here shows better performance benefits, increased flexibility and wider applicability compared to previously presented automatic techniques.

Future work on the presented environment includes generation of VPs in standards other than VSIA, such as SystemC [9] and ConvergenSC [3], as well as processing of algorithmic descriptions developed in environments other than COSSAP.

References

[1] U. Bortfeld and C. Mielenz. Whitepaper C++ System Simulation Interfaces, July 2000.

[2] J. Cockx. Efficient Modelling of Preemption in Virtual Prototype. In *International Workshop on Rapid System Prototyping RSP 2000*, pages 14–19, Paris, June 2000.

[3] CoWare ConvergenSC Products. https://www.coware.com/convergensc.

[4] C. Hein, J. Pridgen, and W. Kleine. RASSP Virtual Prototyping of DSP Systems. In *Design Automation Conference DAC'97*, pages 492–497, 1997.

[5] A. Hemani, A. K. Deb, J. Öberg, A. Postula, D. Lindqvist, and B. Fjellborg. System Level Virtual Prototyping of DSP SOCs Using Grammar Based Approach. *Design Automation for Embedded Systems*, 5(3):295–311, 2000.

[6] A. Hoffmann, T. Kogel, and H. Meyr. A Framework for Fast Hardware-Software Co-simulation. In *Design, Automation and Test in Europe DATE'01*, pages 760–765, Munich, 2001.

[7] International Sematech. International Technology Roadmap for Semiconductors, 1999. Austin, Texas.

[8] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. In *Proceedings of the IEEE*, volume 91, pages 145–164, January 2003.

[9] Open SystemC Initiative. www.systemc.org.

[10] P. Belanović, M. Holzer, D. Mičušík, and M. Rupp. Design Methodology of Signal Processing Algorithms in Wireless Systems. In *International Conference on Computer, Communication and Control Technologies CCCT'03*, pages 288–291, July 2003.

[11] M. Rupp, A. Burg, and E. Beck. Rapid Prototyping for Wireless Designs: the Five-Ones Approach. *Signal Processing Europe 2003*, 83(7):1427–1444, July 2003.

[12] R. Subramanian. Shannon vs. Moore: Driving the Evolution of Signal Processing Platforms in Wireless Communications. In *IEEE Workshop on Signal Processing Systems SIPS'02*, October 2002.