Faster Complex SoC Design by Virtual Prototyping

M. Holzer, B. Knerr, P. Belanović, M. Rupp
Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Wien, Austria

and

G. Sauzon
Infineon Technologies
Operngasse 20b
1040 Wien, Austria

ABSTRACT

Development of complex System on Chip (SoC) for modern communication systems has become more and more challenging. A designer has to bridge the gap between the requirements of the system at algorithmic level and its cycle true hardware description; the latter being synthesized for an Application Specific Integrated Circuit. This development requires numerous sub-steps, i.e., several intermediate descriptions on different abstraction levels of the same system. One of those levels can be an architecture level description implemented purely in software in form of a Virtual Prototype. This paper shows how Virtual Prototyping has been integrated in a design flow by a consistent design methodology and what advantages in terms of design efficiency have been gained.

Keywords: Single System Description, Virtual Prototyping, System on Chip, Design Methodology, UMTS Receiver

1. INTRODUCTION

In the last decades new development approaches and tools have allowed the design of ever more complex hardware (HW). There exist tools for the transition from transistor design to gate level design, automatic place and route, and the use of hardware description languages (HDL) like VHDL. Finally, synthesis tools, operating on HDLs like VHDL, pushed the design productivity forward again. With those approaches it is possible to realize several components of a system on one chip. Such Systems on Chip (SoC) consisted at first only of a single processor, HW accelerators, memory and a bus system. Rather simple HW accelerators emerged to more sophisticated Application Specific Integrated Processors (ASIP)[1]. Newest SoC implementations utilize already more than three processing units. All these efforts are needed to provide high processing power solutions, demanded by the complexity of modern signal processing algorithms[2, 3]. On the other hand, design productivity, which has been increased as mentioned before stays behind the tight time-to-market requirements[4]. With this growing number of implemented complexity the number of needed test cases is increasing, whereas verification takes already up to 70% of the design time[5]. Thus all additional verification efforts regarding separated platforms especialy developed for testing should be reused for a consistent design process.

The process from converting a design from an algorithmic description to a corresponding cycle-true HW description cannot be performed in one step; several intermediate representations are needed. Careful selection and proper operation of these design steps as well as the knowledge of transition methods so called *algorithmic refinement* is crucial for a consistent design flow and thus a successfully project. Several properties of abstraction layers are proposed, as they can be time related (e.g. un-timed, timed functional, bus cycle accurate, cycle true[6]), data related (e.g. float and fixed point representation), and communication related (e.g. Synchronous Data Flow (SDF), Transaction Level Modelling (TLM)[7], Open Core Protocol International Partnership OCP)[8]). A common abstraction layer is the architectural view of a system. Tools like CoCentrics System Studio from Synopsys (www.synopsys.com), and ConvergenSC from CoWare (www.coware.com) provide features for simulation and evaluation of architectures. In spite of such tools, it remains up to the designer to describe the architectural level, as well as to perform the required transitions manually, being an undesired time consuming and error-prone process.

This paper presents in Section 2 Virtual Prototyping in general followed by the description of a specific Virtual Prototype (VP), which can be automatically generated from a high level system description. This automatic generation is integrated in a consistent design methodology as described in Section 3. The translation process is reported in Section 4. Section 5 depicts a design example of an industry-designed UMTS receiver chain, for which VPs have been generated. The last Section 6 summarizes the achieved goals and points out the future work in this field, such as extending the framework to other high-level languages and refining existing translation algorithms.

2. VIRTUAL PROTOTYPING

2.1 Related Work

A VP is an all SW model of a general embedded system consisting of HW and SW parts. Usually such a VP

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

model defines the interfaces and the underlying behavior of a system under design, whereas the abstraction of the implemented behavior can differ. Several VPs of a system under design may exist as long as each fulfils its role in the design process[9]. Previous implementations of VPs have for the most part focused on their use, in the hardware/software co-simulation of the embedded system [10, 11]. While these early efforts are targeted towards increasing the efficiency and quality of the design process through novel modifications of the co-simulation process, a transition method (even a manual one) from an algorithmic description to the VP is not shown. The approach presented in[12] considers automatic generation of VPs and achieves a speedup in the order of five to eight times compared to a manual VP creation, but is does not consider the architectural needs of complex SoC, consisting of several processing units, and a bus system. Those architectures of complex SoC are similar to the architectures used in Printed Board Designs.

2.2 Specialised Virtual Prototype

In the following we present an automatic generation method for a VP tailored for platform based designs. In such case supporting the specifics of a DSP and its surrounding HW components a cycle-true simulation of the entire hardware system is sufficient. However, the simulation time in the context of a UMTS receiver design with very high complexity is too high and the design of such a cycle-true VP would have to wait until the entire system is specified to that level of detail. A simpler VP mirroring the hardware exactly only at the software interface is much more useful. It is thus only necessary to model the processing unit by an Instruction Set Simulator (ISS) running the SW part of the design. Additionally, a complex SoC includes also other modules accessible by the software like memory and the register interface of hardware accelerators. Those interfaces have to be modelled as well. Thus, the remaining parts of the SoC namely the signal processing algorithms parts later being implemented as hardware accelerators do not have to be modelled cycle-true. Even better, in order to provide consistent refinement steps, the high level algorithmic signal processing description, that has been already written at the design start, can be reused. Thus, this VP is not only a simulation platform for verification of the system, the hardware descriptions can be further refined by the designer or synthesized with high level synthesis tools, already providing a rapid prototype of the system.

The implementation of such a VP representation needs a simulation environment, that allows for simulation of parallel processes. Hardware description languages like SystemC and VHDL can be used for that task, because they provide statements for concurrent processing. For the presented VP another simulation interface has been chosen, that has been proposed by the Virtual Socket Interface Association (VSIA)[13]. This VSIA simulation uses a static scheduling, achieving faster simulation compared to the event based simulation of SystemC and VHDL.

3. CONSISTENT DESIGN METHODOLOGY INCORPORATING VP

For a consistent design process it is necessary to provide a unified design environment and at the same time to allow usage of various EDA tools as they are being favoured by the design teams. This is possible by a so-called single system description (SSD) and can be implemented in form of a design database(DDB). In[14] a centred database is proposed a so called model-integrated development. In that work a strong formalism for the translation of model-based system descriptions is introduced and a framework to develop translation tools is presented.
Another approach considering the aspect of a single system description has been made by the Open SystemC Initiative (OSCI)[15]. SystemC is a language that allows not only to describe designs at algorithmic level but also at architectural level. However, SystemC cannot provide the interfaces to other development environments itself. At best, companies base their tools on SystemC, enriching the design with their IP, like ConvergenSC (www.CoWare.com) regarding bus architecture exploration. These enrichments suffer from the same incompatibility to other environments, resp. languages.

A better solution of an SSD is the implementation in the form of a relational DDB[16]. Such a database representation is not bound to specific language constraints and thus offers great flexibility in refining the SSD on its way from concept to realisation. Additional advantages of the DDB approach are fast access, data security by the capability to grant permissions to the developers, a high popularity as well as compatibility with major Data Base Management Systems (DBMS) from Microsoft, IBM, Oracle and the open source DBMS MySQL. The DDB allows the teams to have insight into the current description of the system using their favoured tools and a general HTML based tool that allows to view tables, lists and structural descriptions of the design, thus avoiding any communication obstacles.

This unified design environment has to seamlessly integrate all the existing tools required by the design teams, as they are provided by many EDA tool manufacturers (for example, SPW from CADENCE, CoCentric System Studio from SYNOPSYS, or ConvergenSC from CoWare) and their domain specific languages (DSL). The most time consuming aspect of an existing DDB system is the integration of these tool to the DDB. Figure 1 depicts the DDB surrounded by the required tools each with dedicated interfaces to incorporate the various EDA tools and stays open for incorporating other tools as the empty tool box in the figure indicates. The design teams provide inputs, such as desired system behaviour and structure, constraints, and tool options. In addition, the designers receive outputs, such as status of the system description, results of simulations, estimates of HW costs, timing and similar. Some of the tools are those currently used by the design teams, while others are specially written to perform missing tasks, either automatically or manually by designers and thus making refinement steps of the design consistent.

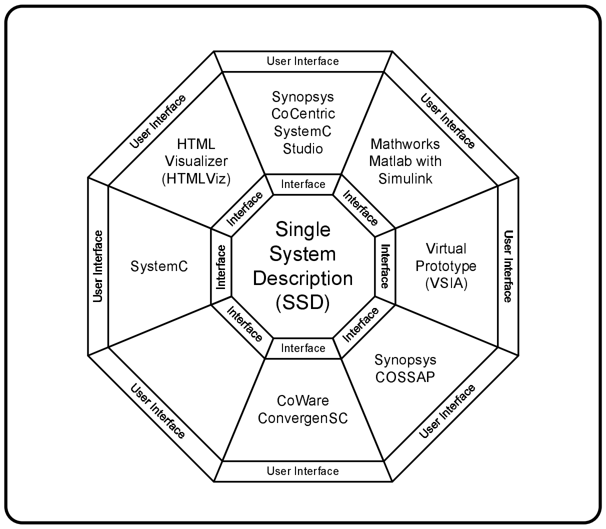
User Interface
Synopsys CoCentric SystemC Studio
HTML Visualizer (HTMLViz)
Mathworks Matlab with Simulink
Interface
Single System Description (SSD)
SystemC
Virtual Prototype (VSIA)
Synopsys COSSAP
CoWare ConvergenSC

Figure 1: *Consistent design utilizing an Single System Description.*

4. VIRTUAL PROTOTYPE GENERATION

At this point, the entire model is represented in the DDB using the translational tools. After the decisions of which component shall be realised in HW, which in SW, has been made [1], this information is stored in a file, which is automatically processed by a tool that extends the DDB. It is possible to flag each instance of the system description as *HW*, *SW* , or yet *UNDEFINED* supporting a future automatic partitioning tool.

A DSP structure enriched by peripheral HW accelerators communicating via a common bus was selected for the design requiring the VP to reflect the DSP as well as to support its peripheral communication. In order to build such a VP, an object-oriented environment in C++ has been created, containing classes for blocks, ports, intercommunication FIFOs, and scheduling, the so-called *VP infrastructure* of the peripheral (Fig. 3). The VP peripheral just mimics the exact HW behaviour at the bus interface, the internal implementation details (FIFOs, scheduler) are only utilisable in HW with reservations. While this implementation implies a specific HW platform, much importance was put on the fact that this platform is rather general, a DSP with a common bus structure for its HW accelerator units as it is typically used in UMTS receiver designs.

For each module that will be implemented in HW, a C++ file pair is created automatically, consisting of header and class file. These module classes are derived from the CDLBlock class, which is the centre of the VP peripheral infrastructure (see Fig. 2). Each derived class instantiates its ports and FIFOs, for each input port a FIFO is provided. The core of each block is the 'BLOCK_run()' method. In this procedure the extracted and now ANSI-C compliant algorithmic description from the COSSAP project is inserted automatically. Furthermore, an additional function is provided to verify that sufficient data for at least one execution is available at the input ports supporting the scheduling capabilities of the VP design.

[1]Note that this decision is still done by humans and thus error-prone.

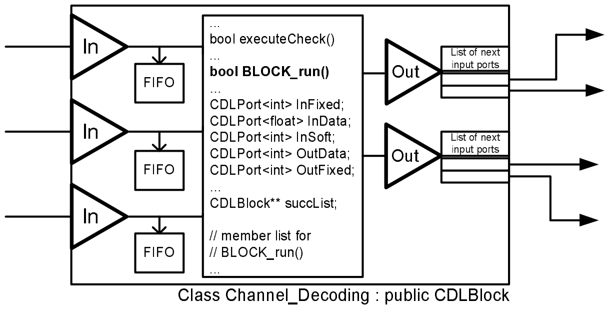

Figure 2: *A virtual prototype block class.*

A C++ file pair for the top module, containing the infrastructure of the peripheral, is created automatically. Within the constructor of this class, all instances characterised as HW are instantiated and interconnected building up the *CDLBlock chain*. This top module class has a member function acting as a scheduler. Assuming a directed graph structure of the architecture, we implemented the scheduler obeying a recursive depth-first search algorithm. On this account, each block manages a list of its successors which has been set up when the interconnections of all instances were established. The *scheduler* is responsible for calling each block as often as possible, i.e. as long as enough data is supplied at the input ports. Note that the two grey constituents of the VP *HW/FW* and *Bus interface* (Fig. 3) are not part of the automatic generation procedure as the algorithmic description does not contain any information concerning this matter.

For verification purposes, we provide several test modes. Each block can be triggered independently to copy all output values to one or several files (dump mode) or to read all its input values from files (pump mode) or both. As the top module is derived from the block class, too, this also is true for all peripherals. Thus, it is possible to feed in data at arbitrary positions during run-time allowing to speed up testing significantly since testing of specific modules does not require now to run the entire VP but only those blocks under test.

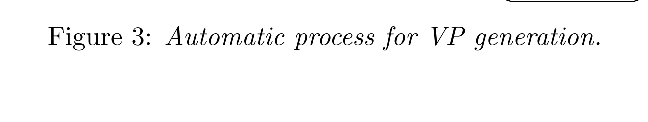

Figure 3: *Automatic process for VP generation.*

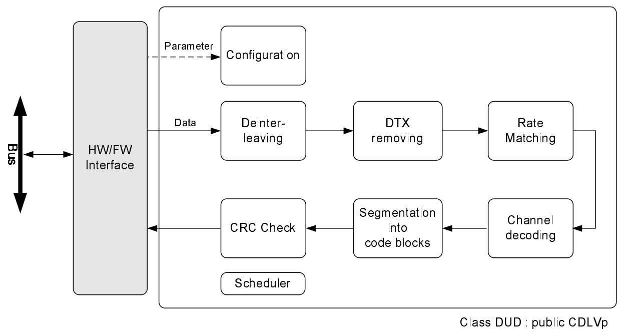

Figure 4: *VP Block diagram of the peripheral for decoding of user data (DUD).*

5. DESIGN EXAMPLE OF A UMTS RECEIVER

According to the design flow presented in Sec. 4 the entire COSSAP project for a UMTS baseband processing unit (UMTS BPU) has been analysed by the SDI and imported to the DDB. After partitioning of the algorithmic description of the UMTS BPU into SW and HW components, ten VP peripherals have been generated. Four of them assemble the UMTS cell phone receiver part, which has been selected for further consideration. Each of the four parts takes over the complete or partial functionality of a) delay profile estimation (DPE), b) frame and slot synchronisation (SYNC), c) rake receiver (RAKE) and d) decoding of user data (DUD). As an detailed example, Fig. 4 depicts the DUD peripheral to expose the VP infrastructure of the automatically generated HW peripheral. This DUD consists of six blocks, representing the DSP for the decoding of user data according to 3GPP TS25.212. The *configure* block collects the initial parameters of the six SP blocks, which cannot be passed by the HW/FW interface during runtime and delivers them to each block to set up the specific behaviour, for instance switching the data processing from a slot to a frame basis. Note that the COSSAP developer has the possibility to change the parameters of a block before each block invocation. In the VP as well as in the real HW implementation it is only possible to update the parameters of the blocks between two peripheral invocations. The *HW/FW interface* communicates via the on-chip bus with the Starcore DSP, configures the DUD peripheral, fills the FIFOs of the *Deinterleaving* block with data and triggers the *scheduler* function.

To achieve the time savings due to the presented automation, it was necessary to port the HW peripherals manually from the COSSAP project to the VP representation (except of the rake because of its extraordinary size). Within this process the expenditure of human labour, measured in person hours (PH), for translating the behavioural model of each module into a C++ file pair has been documented. We assume the time expense for this task is proportional to the lines of code (LOC) of the GenericC files of the modules. The results are listed in Table 1, showing also that the efficiency (avg. PH/LOC= 0.017) of the manual work is roughly constant for each module. In a second step, the creation of the C++ files for the top module, which instantiates, interconnects, and configures the blocks, has been accomplished. The time exposure is assumed to be proportional to the number of instantiated blocks (NIB), further on referred to as infrastructural complexity. The results are listed in Table 2 showing an avg.PH/NIB= 2.0.

HW peripheral	LOC	PH	PH/LOC
DPE	1653	25	0.015
SYNC	2422	39	0.016
DUD	2398	43	0.018

Table 1: *Algorithmic complexity and time expense of the manually generated VP peripherals.*

HW peripheral	NIB	PH	PH/NIB
DPE	5	8	1.60
SYNC	7	17	2.43
DUD	6	12	2.00

Table 2: *Infrastructural complexity and time expense of the manually generated VP peripherals*

As the HW/FW interface has to be described manually with respect to the underlying architecture, its development time is not taken into account for the evaluation of the time reduction. Assuming that the averaged values of the PH/LOC and PH/NIB ratios obtained from UMTS receiver are reliable metrics for the remaining peripherals of the whole UMTS BPU, it is possible to estimate the design effort for the manual generation of all VP peripherals. An amount of 774PH for 121NIBs and 31266LOCs was estimated. Employing the presented automatic approach each of the VP peripherals is generated within seconds. The industrial development routine revealed revision levels for the peripherals reaching easily depths of more than fifty. And at each revision version, in which especially the infrastructure (e.g. number of ports, connections) has been changed to a certain degree, this approach exposes its superior performance, applying the automatic mapping procedure. The time savings accumulate to many thousand person hours over an entire design. Its application in industry proved that the modification of an existing COSSAP project necessary to be compliant to the GenericC guidelines causes negligible effort, while the speedup of development time is substantially.

6. CONCLUSIONS

The automated environment for VP generation presented here has been successfully applied to an industrial design flow, showing significant speedup in creation of VPs, with savings in the order of hundreds of person hours. Simultaneously, this approach also eliminates human-related errors, thus improving quality. Additionally, work presented here shows better performance benefits, increased flexibility and wider applicability compared to previously presented automated techniques. Future work on the presented environment includes generation of VPs in standards other than VSIA, such as SystemC, as well as processing of algorithmic descriptions developed in environments other than COSSAP. Also, direct binding of the presented VP environment with more of the numerous commercial algorithmic and architecture level tools is expected to increase automation and thus significantly gain design efficiency.

REFERENCES

[1] A. Hoffmann and H. Meyr. *Architecture Exploration for Embedded Processors with LISA*. Kluwer Academic Publishers, 2002.

[2] R. Subramanian. Shannon vs. Moore: the Digital Signal Processing in the Broadband Age. In *IEEE Communication Workshop*, Aptos, California, May 1999.

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

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

[5] B. Bailey. The Waking of the Sleeping Giant – Verification, April 2002. www.mentor.com/consulting/techpapers/mentorpaper_8226.pdf.

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

[7] L. Cai and D. Gajski. Transaction Level Modeling in System Level Design. Technical report, Center for Embedded Computer Systems, 2003.

[8] N. Weyrich A. Haverinnen, M. Leclercq and D. Wingard. Whitepaper SystemC based SoC Communication Modeling for the OCP Protocol, Oct. 2002.

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

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

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

[12] 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.

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

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

[15] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[16] 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.