Published in the proceedings of the European Signal Processing Conference (EUSIPCO 2004), pp. 2015-2018, Vienna, 2004

DESIGN FLOW IMPROVEMENTS FOR EMBEDDED WIRELESS RECEIVERS

B. Knerr[1]*, P. Belanović*[1]*, M. Holzer*[1]*, G. Sauzon*[2]*, and M. Rupp*[1]

[1]Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Wien, Austria

[2]Infineon Technologies SMS
Operngasse 20b
1040 Wien, Austria

ABSTRACT

Complexity of modern communication systems, particularly in the wireless domain, grows at an astounding rate. The algorithmic complexity significantly outpaces the growth in complexity of underlying silicon implementations. Even more dramatically, algorithmic complexity outpaces design productivity. Primarily responsible for the second point is the fragmentation of the design process caused by the extremely heterogeneous nature of modern EDA tools required in wireless communication designs. Such design flows lead to numerous inconsistent design representations, each of them tackling only a specific aspect of the development while neglecting important properties for other design aspects. A permanent rewriting of the design description is required due to the iterative nature of the design process. To overcome such inconsistencies and speed up the process, we present in this paper an automated transition from a high level algorithmic description of a synchronous data flow model to a VSIA compliant Virtual Prototype for an industry-designed UMTS receiver. This automatic translation of algorithmic descriptions is embedded in a consistent design methodology supporting hardware/software co-design as well as efficient testing at all abstraction levels of the design process.

1. INTRODUCTION

Embedded systems, especially those designed for wireless communication applications, consist of an increasing software (SW) part with complex interfaces to the underlying hardware (HW). In general the design process can be coarsely divided into three consecutive stages [1]; it starts with the new technological concepts represented in a high level algorithmic description, for instance MATLAB from MATHWORKS (www.mathworks.com) or even just C-programming. Verification of the algorithms is typically performed as testing by simulation. The second major stage is the design specification with respect to which technologies should be used to map such algorithmic ideas into HW and SW. The decisions which standard components and intellectual property (IP) can be reused and which have to be newly developed are included in this stage. Once the system has been specified in detail, the implementation, as the third and final step, starts. This step mainly consists of building all required components that cannot be bought from outside vendors and join them with existing parts and, of course, programming the DSPs, FPGAs and designing ASICs.

Within this process chip manufacturers currently struggle with the so-called design gap, which can be split in two constituent parts: 1) the required complexity of algorithms in future communications systems (Shannon's law) increases more than the available complexity of silicon (Moore's law) [2] and 2) the available complexity of silicon rapidly outpaces design productivity, expressed as the average number of transistors designed per staff and month [3], resulting in an even wider design gap. To narrow this divergence, various measures have been proposed: 1a) including additional HW accelerators in conventional DSP architectures, 1b) allowing for strong HW parallelism, 2a) using a unified language that allows to overcome the difficulties when describing the design in appropriate fashion for the various design teams (www.systemc.org), 2b) re-using IP as much as possible (www.design-reuse.com, www.vsia.org), 2c) developing a Virtual Prototype (VP) as intermediate design step [4], and 2d) new testing strategies. Since the EDA tools available on the market may support only one or two of these speed-up measures at the same time, the performance in design productivity remains insufficient.

The most significant shortcoming within the explained design flow lies in the fact that the system descriptions at the three stages of the design process are fundamentally different, relying on different descriptions and thus making forward and backward communications between teams at least difficult if not impossible. Consequently, system descriptions are constantly reformatted and rewritten by the corresponding experts to incorporate input from the other teams. This mode of operation is error-prone, slow due to a lack of automatism and thus inefficient.

Clearly, significant increase in efficiency, reduction of time to market and improvement in quality can be achieved by providing a consistent design process covering all design teams in a their various design stages. Such a process requires a unified design environment, supporting all the teams equally and allowing them to work on a single system description containing all information at all levels. A framework meeting these requirements has been proposed in [5]. Its concept and implementation in form of a design database (DDB) with automatic translation tools allowing to convert algorithms from one into other abstraction levels as well as related work is briefly summarised in Sec. 2.

One of the major issues in the presented approach is based on the property of such translation tools to *understand* various abstraction levels and the corresponding dedicated system descriptions of the design. Refinement steps can take place within each of the representation levels or within the all-embracing system design description. To achieve this goal certain requirements exist, reported further on in Sec. 3, which a high level algorithmic description has to meet to enable an automatic translation to a lower abstraction level. In this context, HW/SW partitioning is performed, and a platform based design is introduced with a specific bus model. The VP technique is highlighted, its location in the design process, its significant impact on development time and related work in Sec. 3.2. In Sec. 3.3 the implementation of the automatic translation algorithms is presented in more detail. Sec. 4 depicts a design example of an industry-designed UMTS receiver chain, for which VPs has been generated and are currently in use in the development process. The last Sec. 5 summarises 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. CONSISTENT DESIGN METHODOLOGY

In order to make the design process consistent by providing a unified design environment but a the same time allow for various EDA tools as they are being favoured by the various design teams, a so-called single system description (SSD) was generated in form of a DDB [5]. Thus, each team applies its expertise and refines the SSD on its way from concept to realisation without leaving their accustomed language.

A similar approach with a centred database has been proposed

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

in [6] named model-integrated development. In this work a strong formalism for the translation of model-based system descriptions is introduced and a framework to develop translation tools is presented.

Note that at any point in time, the SSD 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. In this improved process, each team still requires its usual set of tools. Hence, the 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). An approach considering the aspect of a single system description has been made by the Open SystemC Initiative (OSCI)[7]. 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 SQL-DDB [8, 5]. A database representation is not bound to specific language constraints and thus offers great flexibility in capturing the miscellaneous aspects of a design. 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 DataBase Management Systems (DBMS) from Microsoft, IBM, Oracle and the open source DBMS MySQL.

The DDB is enriched by a set of dedicated in and out porting SW, the so-called translation tools for a selected subset of development environments that were favoured by the design teams involved. A further advantage of the DDB approach is that all information is contained there and will not be lost even if some EDA tools do not require all the design details. The core of the underlying DDB structure is shown in Fig. 1. It has been designed to generally fit system descriptions, with support for such concepts as modules or entities, their hierarchy and interconnections. In addition to such concurrent concepts, sequential parts of system descriptions, such as processes and operation sequences are also supported. The nomenclature of all concepts in the database structure implementing the single system description was strongly inspired by SystemC. All

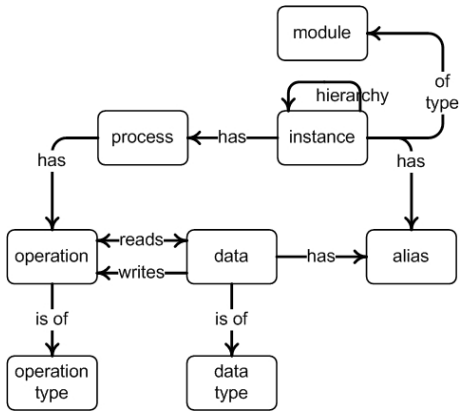

Figure 1: *The Design Database (DDB) structure.*

entities that make up the system are instances of modules. These instances form one or more layers of hierarchy. Each of the instances can contain one or more processes. All processes in the system run concurrently. Processes are internally sequential, formed by sequences of operations. Communications between instances, processes and operations is achieved through data. Data connecting several instances has several aliases; one within the context of each of the connected instances. An alias has an alias type, such as inport, outport, in-out port or internal signal. Data has a data type, such as a signal, variable or constant. Operation also has an operation type, such as addition (+), multiply- accumulation (MAC) or left bitwise shift (<<).

The most important aspect of an existing DDB system is the integration of each tool to the SSD. Since there exists only one system description, but a great variety of tools and DSLs, each with unique requirements, means of providing inputs for and incorporating outputs of all the tools must be provided by translational tools.

Fig. 2 depicts the DDB surrounded by the required translational tools each with dedicated ”in” and ”out” porting SW to incorporate the various EDA tools. The names in the figure are given as examples for EDA tools. Some of them have already been incorporated while others may be candidates for future implementations. The

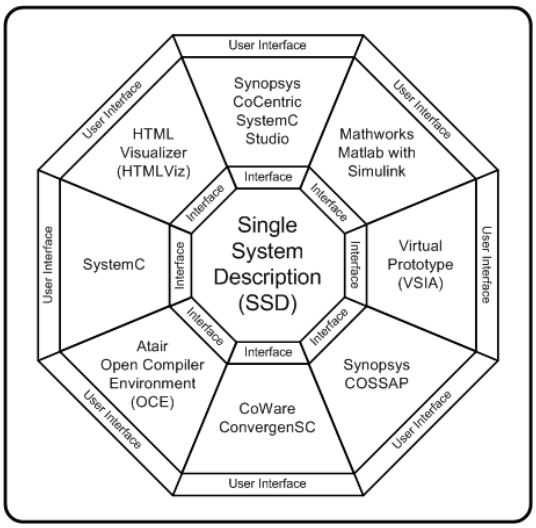

Figure 2: *Consistent design utilising an Single System Description.*

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.

3. AUTOMATIC TRANSLATION ALGORITHM

3.1 Automatic Understanding of System Level Designs

As Fig. 4 further ahead illustrates, the underlying concept of the translational tools is two-tiered, consisting of a parser and a scanner. Both parts together establish the System Description Interface (SDI). The parser is adapted to the EDA tool or the description language and translates the contained information to a line-based intermediate format (IF). A scanner tool processes the IF and enriches the DDB. The two-stage structure facilitates the development of new interfaces in such a way that the developer of the parser can neglect everything related to data base management, i.e. adding or deleting modules, removing redundant information, updating of attributes etc. This task is fulfilled by the scanner tool, which depends only on the IF and the current contents of the DDB.

The algorithm design team favours COSSAP as a graphical representation of the design at system level. To provide a complete representation of the COSSAP model within the DDB, the information on the architectural structure as well as on the functionality and behaviour of each component is extracted from the COSSAP project. A COSSAP project stores its model architecture in a file pair named after the top module, suffixed ’.v _arc’ and ’.v_ent’. The description language used in these files is VHDL compliant. A parser for VHDL has been implemented to create the IF representation of the model architecture (see Fig. 4 further ahead). This parser

is based on two open source tools: Flex, a lexical analyser, and Bison, a parser generator. The scanner processes the achieved IF and adds the information to the DDB. Each component of the COSSAP model has its own GenericC file containing the behavioural model. A parser for these files was implemented to make them ANSI-C compliant for further processing. Note that the developer of the GenericC files has to adhere to a guideline catalogue, to enable a smooth automatic transition to ANSI-C. These guidelines consist mainly of name conventions and have negligible impact on the COSSAP development routine. In particular the guidelines did not limit the creativity of the algorithmic design team and were thus quickly adopted throughout the team. While other SDIs exist (for example for reading SystemC designs), we will focus on the COSSAP SDI in this paper.

3.2 Virtual Prototyping Technique

The partitioning process transforms a system level specification into a heterogeneous architecture composed of HW and SW modules. Whereas HW development, and especially its testing, can be done rather independently from the SW development, development and testing of the SW has to wait until the HW has been designed, and in the case of an ASIC design, the first engineering samples have been manufactured.

Modelling the future DSP HW-architecture combined with a bus model and HW accelerators by SW (utilising an Instruction Set Simulator (ISS) to simulate the DSP), a software model of the entire design can be realised, generally called virtual prototype [9, 10, 4, 11]. In this technique SW reflects the behaviour of the HW and the HW interface via a bus model to the SW, as it will be realised later in HW. Such a VP can be implemented faster than the HW itself, because most of the HW implementation details can be neglected and high-level description languages can be used instead of HW description languages. Another advantage of a VP is its capability to serve as a reference (Golden-) model for the HW whose functionality is mirrored at the bus interface. A crucial point after the partitioning is a carefully designed interface between the various HW modules (HW accelerators or peripherals) and the SW modules, running on the DSP.

3.3 Automatic VP 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. 4). Note 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. 3). 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()'

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

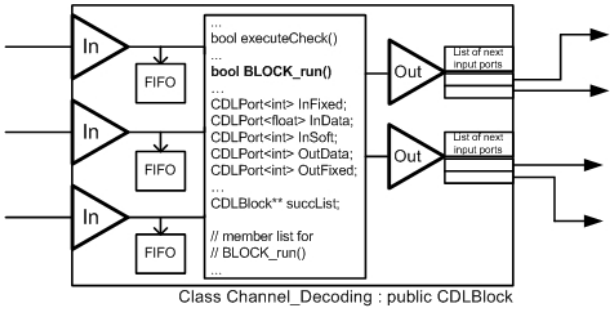

Figure 3: *A virtual prototype block class.*

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.

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. 4) 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.

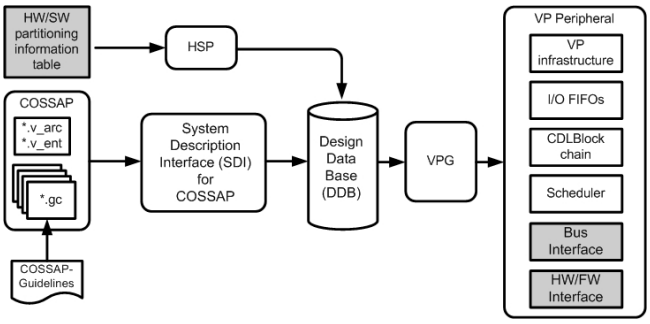

Figure 4: *Automatic process for VP generation.*

4. DESIGN EXAMPLE OF A UMTS RECEIVER

According to the design flow presented in Sec. 3 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. 5 depicts the DUD peripheral to expose the VP infrastructure of the automatically generated HW peripheral. This

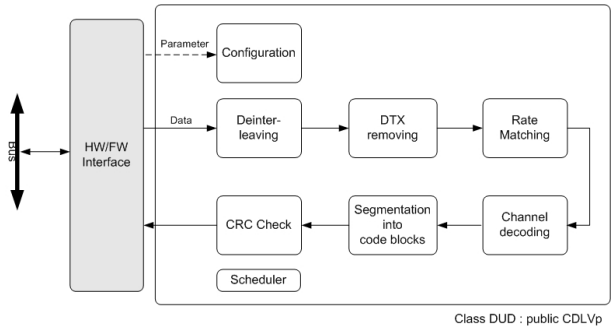

Figure 5: *VP Block diagram of the peripheral for decoding of user data (DUD).*

DUD consists of six blocks, representing the DSP for the decoding of user data according to 3GPP TS25.212. The *Configuration* 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

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.*

the top module, which instantiates, interconnects, and configures the blocks, has been accomplished. The time spend 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. 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 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 speed-up of development time is substantially.

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*

5. CONCLUSIONS

The automated environment for VP generation presented here has been successfully applied in an industrial design flow, showing significant speed-up 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] M. Rupp, A. Burg, and E. Beck. Rapid Prototyping for Wireless Designs: the Five-Ones Approach. *Signal Processing Europe 2003*, June 2003.

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

[3] P. Fisher and D. Cottrell. Emerging Standards in the Electronic Design Automation (EDA) Industry. In *Electronic Systems Design Seminar*, UC Berkeley, California, Oct. 1999.

[4] C. Valderrama. Virtual Prototyping For Modular And Flexible Hardware-Software Systems. *Design Automation for Embedded Systems Journal*, 2(2):267–282, 1997.

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

[6] 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, Jan. 2003.

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

[8] J.R. Groff and P.N. Weinberg. *SQL: The Complete Reference, Second Edition*. McGraw-Hill,Osborne, 2002.

[9] RASSP Education and Facilitation Program. Virtual Prototyping Using VHDL. Technical report, SCRA and DARPA, 1999.

[10] A. Hemani et al. System Level Virtual Prototyping of DSP SOCs Using Grammar Based Approach. *Design Automation for Embedded Systems Journal*, 5(3):295–311, 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.