Design exploration and HW/SW rapid prototyping for real-time system design

Sylvain Huet, Emmanuel Casseau, Olivier Pasquier

▶ **To cite this version:**

HAL Id: hal-00079263

https://hal.archives-ouvertes.fr/hal-00079263

Submitted on 12 Jun 2006

Design exploration and HW/SW rapid prototyping for real-time system design

Sylvain Huet
LESTER Lab., CNRS FRE 2734, UBS
56321 Lorient Cedex, France
shuet@iuplo.univ-ubs.fr

Emmanuel Casseau
LESTER Lab., CNRS FRE 2734, UBS
56321 Lorient Cedex, France
emmanuel.casseau@iuplo.univ-ubs.fr

Olivier Pasquier
Polytech'Nantes, IREENA
BP50609, 44306 Nantes Cedex 3, France
olivier.pasquier@polytech.univ-nantes.fr

Abstract

Embedded signal processing systems are usually associated with real-time constraints and/or high data rates so that fully software implementation are often not satisfactory. In that case, mixed hardware/software implementations have to be investigated. However the increasing complexity of current applications makes classical design processes time consuming and consequently incompatible with an acceptable time to prototype. To address this problem, we propose a system-level design based methodology that aims at unifying the design flow from the functional description to the physical HW/SW implementation through functional and architectural flexibility. Our approach consists in automatically refining high abstraction level models through the use of an electronic system-level tool. We illustrate our methodology with the design of a wireless communication system.

1. Introduction

In this paper, we present a design approach which consists in automatically refining high abstraction level models. The methodology is based on an Electronic System-Level (ESL) design tool and aims at unifying the design flow from the functional description to the physical HW/SW implementation through functional and architectural flexibility. The paper is organized as follows. In section 2 we detail each step of the design flow we use, from the system specification to the prototyping and implementation. In section 3 our approach is applied to prototype a wireless communication system based on a Multiple Input Multiple Output (MIMO) transmission system [8]. The conclusions are given in section 4.
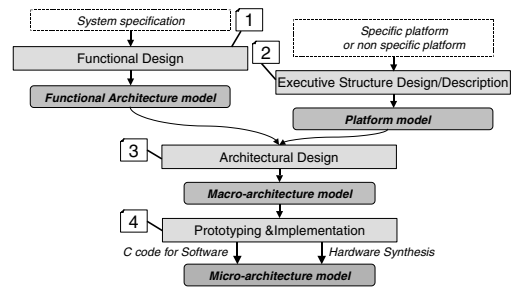

Figure 1. Methodology steps

2 Design flow

System-Level Design (SLD) flows are promising design approaches since they aim at automating the refinement of the specifications down to the implementation [9][11]. These flows are structured on the broadly accepted Y paradigm which aims at separating the functional model and the physical model of the systems. In this paper we put an SLD flow into practice with the view to reduce the "time to prototype" of digital signal processing systems. Even though we target a specific domain of application this flow can be extended to design a wide range of electronic systems.

Our methodology is structured on the basis of four main steps linked by three models: the Functional Architecture model, the Platform model and the Macro-Architecture model as presented on figure 1 and is now supported by Cofluent Studio [1], the ESL design tool we use. The originality of our approach lies in using this tool in conjunction with the High Level Synthesis [10] (HLS) tool GAUT [2] for the generation of the hardware parts of the system. The next four sub sections detail each step of the methodology.

2.1 Functional design

This step consists in defining the functional solution by extracting and exploring the parallelism of the specifications. This functional solution is depicted according to the functional architecture model (figure 3a). The solution is graphically described by a set of functions interconnected by relations [12]. The functions behaviors can be written either in C or SystemC [4]. This model is free of any implementation details and can be simulated to verify the system functionality. Some reference models are selected through this iterative validation process and will be considered as golden references for the next design steps.

2.2 Executive structure design/description

The executive structure design/description step aims at describing or designing the physical organisation of the platform on which the application will run. At this step, the platform is described as a set of processing elements (PE) interconnected by communication nodes (CN) (figure 2b). In the context of this paper, the prototyping platform we target is a Sundance platform [3]. This testbed is modular in terms of processing elements and communication network: it can embed several DSPs (in our case TMS320C6201, TMS32C6701, TMS32C6416 from Texas Instrument (TI) [5]) and FPGAs (in our case V1000E from Xilinx [7]) communicating through point to point communication links which can be either 20 Mbytes/s Sundance's Comm.-Ports (CP) or 200 Mbytes Sundance's Bus links (SDB).

2.3 Architectural design

The architectural design step, consists in mapping the Functional Architecture to the Platform Architecture. The Functional Architecture model is mapped to the Platform Architecture Model, that is to say Functions to Processing Elements and Relations to Communication Nodes. The resulting Macro-Architecture model is heterogeneous: it contains both functional and executive elements. Interfaces functions are also introduced to map Functional Relations to Communication Nodes. This is represented by the "Communication network" on figure 2c.

Like the Functional model, the Macro-Architecture model, also named virtual prototype, can be simulated [12]. This is very important for architecture exploration and performance estimations since this model describes both the behavior of the system and the hardware on which it runs. Thanks to the high level of abstraction of this model, rapid simulations can be performed to explore several platform configurations.
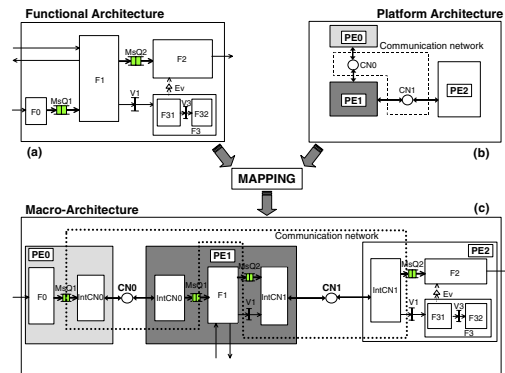

Figure 2. Description models

2.4 Prototyping and implementation

Finally, the Macro-Architecture is refined to the Micro-Architecture which is composed of the programming code for the microprocessors and the synthesizable description of the hardware parts of the system. Consequently two major ativities are involved to perform this task.

(1) Software generation. The code of the functions mapped onto software processors is generated in C using common services of Real Time Operating Systems (RTOS). Up to now two generation templates are available: either the VxWorks [6] or a generic RTOS application programming interface (API) could be used. In the last case, the API has to be implemented with the primitives of the targeted RTOS. As our platform embeds TI DSP, we have implemented this API with the DSP/BIOS primitives. At last platform specific code has to be written to solve the inter processing elements communication problem: in this model are also introduced interfaces functions (see "Communication network" on figure 2c) where we call our CP or SDB drivers.

(2) Hardware generation. Characterised Hand-written or IP hardware components can be used to implement the hardware parts. Nevertheless we promote a flexible approach based on HLS to generate them. The HLS tool we use is GAUT [2]. Its input language is a subset of behavioral VHDL but a SystemC entry will be soon available, hence unifying the languages used in the design flow. Beyond the synthesis of the algorithmic cores the HLS tool we use also synthesises our platform specific communication interfaces. Therefore, the synthesized function can be easily interfaced with others software or hardware processors of our rapid prototyping platform.

At this step of our design flow the system is refined enough to reach the entry points of the downstream tools we use: Code Composer from TI for the software parts [5], ISE/Foundation from Xilinx for the hardware parts [7].

3   Wireless communication system design

In this section our ESL based design methodology is illustrated by the design of a MIMO [8] wireless communication system. Functional and architectural exploration of such a system are discussed. Implementation issues are also presented.

3.1   Functional exploration

The system we target is composed of two parts: the emitter and the receiver side. The following cascading blocks can be identified for the emitter: a source coder, a channel coder, a mapper and a space time block coder; for the receiver: a combiner a demapper, a channel decoder and a source decoder. Therefore, the corresponding functional model of the system is composed of these eight blocks plus the channel to model the system environment. Thus when the functionality of each block has been validated, we dispose of an executable golden reference model of the system. At this stage of our flow models are used to explore and validate various alternatives for each functional blocks.

Beyond the functional results obtained at this stage, a profiler integrated in the ESL design tool allows to measure the relative computational complexity of each function. This metric can be used later for architecture exploration.

3.2   Architectural exploration

The data flow orientation of this application offers opportunities in term of architectural exploration. Our strategy consists in first trying a full software implementation mapped on a unique DSP. We profiled the timing behavior of this first prototype so that we are able to back annotate our macro architecture models with an accurate timing of the software implementation of each blocks. If this implementation does not fit the non functional specifications, in our case the throughput, the simulation of the timed macro architecture model allows the identification of the bottlenecks, then hardware accelerators are introduced. Thanks to the HLS tool we use, the design of these components is speeded-up.

3.3   Architecture performance

The simulations performed at the Architectural Design step allow us to conclude that the channel decoder is the bottleneck of the system for the fully software solution. Consequently we add an hardware accelerator to perform this task. From its behavioral algorithm we automatically synthesize four RTL IP with different timing constraints using either the CP or SDB communication links with GAUT [2]. With the back annotation of the macro architecture model

we have been able to estimate the performance of mixed hardware/software solutions and to conclude that the fully software solution offers very poor performances whereas the mixed hardware/software solution achieves data rates close to the rates of the hardware accelerator.

4   Conclusion

We have presented a system design approach that goes from the functional description to the physical HW/SW implementation through functional and architectural flexibility. Based on a Electronic System Level design tool, the aim of this approach is to facilitate and shorten the design cycle. The functional model of the system to be implemented is expressed at a high-level of abstraction. Profiling and estimates at the system-level as well as at the architectural one help the designer in his system distribution and architectural choices. The system is progressively designed according to a refining process that takes into account processing and communication element properties.

5   Acknowledgments

References

[1] Cofluent design, http://www.cofluentdesign.com.

[2] Lester, gaut, http://web.univ-ubs.fr/lester/www-gaut/.

[3] Sundance, http://www.sundance.com.

[4] Systemc, http://www.systemc.org.

[5] Texas instruments, http://dspvillage.ti.com.

[6] Vxworks, http://www.windriver.com.

[7] Xilinx, http://www.xilinx.com.

[8] H. Bolcskei and A. J. Paulraj. *Multiple-input multiple-output (MIMO) wireless systems*, chapter 90, pages 90.1 – 90.14. CRC Press, 2002.

[9] J. Calvez. *Embedded Real-Time Systems. A Specification and Design methodology*. John Wiley, 1993.

[10] D. Gajski, N. Dutt, A. C. Wu, and S. Y. Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[11] A. Gerstlauer, R. Dmer, J. Peng, and D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[12] R. L. Moigne, O. Pasquier, and J. Calvez. A graphical tool for system level modeling and verification with systemc. In *FDL*, 2003.