Long Term Trends for Embedded System Design

Ahmed Amine JERRAYA
Laboratoire TIMA, 46 Avenue Félix Viallet, 38031 Grenoble CEDEX, France
Email: Ahmed.Jerraya@imag.fr

Abstract. An embedded system is an application specific electronic subsystem used in a larger entity such as an appliance, an instrument or a vehicle. The embedded system may embody the complete system functionality in several different ways---using software running on CPUs or in specialized hardware accelerators. The evolution of technologies is enabling the integration of complex platforms in a single chip; called System-on-Chip, SoC. Modern SoC may include one or several CPU subsystems to execute software and sophisticated interconnect in addition to specific hardware subsystems. Mastering the design of these embedded systems is a challenge for both system and semiconductor houses that used to apply only software strategy or only hardware strategy. In addition to classic software and hardware that can be designed by software and hardware engineers, SoC design requires the design of hardware- dependent software and software-dependent hardware. In order to meet performances requirements, these two parts need to be jointly designed. This requires a new kind of engineers that need knowledge in both hardware and hardware design to codesign these HW-SW interfaces. This paper analyzes this evolution and defines long term roadmaps for embedded system design.

1. Introduction

90% of new ASICs already include a CPU in 130nm technology [7]. Multimedia platforms (e.g. Nomadik and Nexperia) are already multi-processor system on chip (SoC) using different kinds of programmable processors (e.g. DSPs and microcontrollers) [5]. Heterogeneous cores are exploited to meet the tight performance and cost constraints. This trend of building heterogeneous multi-processor SoC will be even accelerated. SoCs will be composed of multiple, possibly highly parallel processors for applications such as mobile terminals, set top boxes, game processors, video processors, and network processors. Moreover, these chips will contain very sophisticated communication networks called networks-on-chips (NoC). So the design of a SoC will consist of an assembly of processors executing tasks concurrently. As a result, design methodologies must change their focus to selecting and using processors---either programmable or dedicated---as basic components rather than the logic modules, such as gates and ALUs, used by the current methods. It is easy to imagine that the design of a SoC with more than a hundred processors will become a current practice in few years, e.g. with 65nm technology in 2007. Compared with conventional ASIC design, such a multi-processor SoC is a fundamental change in chip design.

Published in proc. CEPA 2 Workshop – Digital Platforms for Defence, Brussels, Belgium, March 15-16, 2005

2. Why Application-Specific SoC?

Application requirements keep system designers busy by requiring that we develop different platforms for different design spaces. Some observers have speculated that the semiconductor industry is headed toward an ultimate universal chip that will provide all the computation power required by all applications. This would be an attractive proposition since using of standard platform allows us to avoid the cost and the pain of building specific HW/SW platforms. Of course, when possible this kind of solution should be developed. However, several forces will probably require us to use multiple platforms for quite some time.

With a particular application space, it is possible to build one or a few platforms that can be effectively used as the basis for many products within that space. This is the case of TI-OMAP and Nomadik for mobile terminals and Nexperia for digital TV [5]. So why not enhance one of these platforms to be used across multiple applications? Because these hardware platforms must meet several stringent design constraints---hard real-time performance, power consumption, and cost---all at the same time. If we did not have so many simultaneous constraints on the system, then it would be easier to apply general-purpose computing technology. If the constraints were not so tight---consider, for example, the needs of a 3G mobile handset---then once again it would be easier to use general-purpose solutions. But when the platform must deliver very high performance at very low power consumption and very low cost, the platform must be specialized to take advantage of characteristics of the application.

Furthermore, different applications have different combinations of requirements, which pushes us toward multiple architectures. Consider video compression, for example. Even when we use a common standard, such as AVC/H.264, we will need different platforms for different types of video compression systems. Digital cinema and a new crop of prosumer high-definition video cameras require very large frames. High-definition video compression requires more than 32 TIPS (Tera instructions per second) as computation complexity. A cell phone with a video camera will use much smaller frames and lower frame rates, requiring less computation, but will also need to meet much more stringent power consumption requirements. Cell phones must also be much more physically compact than high-end video cameras, pushing us toward more highly integrated architectures. As a result, we should expect to see different platforms even within this one application.

High-definition video recording illustrates the need for heterogeneous platforms. Assume that a pure software approach is used with a SoC platform consisting of programmable processors. In this case, to meet the computation requirement, 32,000 RISC processors running at 1GHz are needed on the SoC platform. Currently and in the near future, such a SoC platform may not be realisable in terms of chip area and, especially, power consumption. In terms of power consumption, such a platform has a significant limitation since it requires a large number of transistors and the leakage current that will be dominating more and more is proportional to the number of devices. When designed as a mixed hardware/software solution the same MPEG2 encoder can be implemented using only a four-processor solution in the case of the digital cinema application [1].

3. Hardware/Software Interfaces: The Design Bottleneck

The design of a system-on-a-chip generally requires developing complex software, entailing hundreds of thousands of lines of code, to run on the SoC platform. All that work must be accomplished between the competing constraints of a short time-to-market window and ever increasingly complex functionality. Current ASIC design approaches are hard to

scale to such a highly parallel multi-processor SoC. Designing these new systems by means of classical methods gives unacceptable realization costs and delays.

Traditional ASIC designers have a hardware-centric view of the system design problem. Similarly, software designers have a software-centric view. System-on-chip designs require the creation and use of radical new design methodologies because some of the key problems in SoC design lie at the boundary between hardware and software.

In order to allow for concurrent HW/SW design, we need abstract models of both software and hardware components (Fig. 1). Ideally one would like to start with a set of SW tasks communicating with a set of HW subsystems. Because software components run on processors, the abstraction needed to describe the interconnection between software and hardware components is totally different from the existing abstraction of wires between hardware components as well as the function call abstraction used to describe software. The HW/SW interface needs to handle two different interfaces: one on the software side using API and one on the hardware side using wires. This heterogeneity makes HW/SW interface design very difficult and time-consuming because the design requires the knowledge of both software and hardware and their interaction.

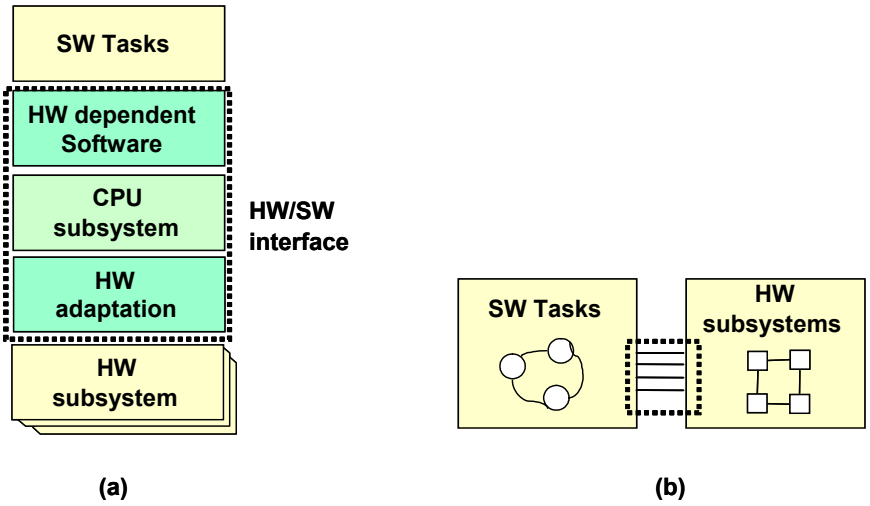

Figure 1. Evolution of abstraction levels in chip design

In general-purpose computer design, system designers must also consider both hardware and software, but the two are generally more loosely coupled than in SoC design. As a result, general-purpose computer systems generally model HW/SW interfaces twice. HW designers use a HW/SW interface model to test their hardware design, and software designers use a HW/SW interface model to validate the functionality of their software. **Using two separate models induces a discontinuity between hardware and software.** The result is not only a waste of design time but less efficient, lower-quality hardware and software. This overhead in cost and loss in efficiency are not acceptable for SoC design. A single HW/SW interface needs to be shared between both hardware and software designers. We believe that an additional type of designer, i.e. HW/SW system designer, is required to connect hardware and software design teams in an efficient way [2].

It is clear that there are two kinds of software:

- The application software: it may be designed by classical real-time software designers using specific analysis tools able to support complexity.
- The hardware dependent software; which adapts the application software to a CPU subsystem. In general-purpose computer systems this layer may use standard

components (e.g. operating system and middleware) that are designed to be ported to different hardware platforms. When such a solution is applied to SoC the operating system and middleware (e.g. Qualcom BREW) induce a significant overhead in system cost (e.g. code size, runtime, energy consumption, etc.). The overhead is caused by two reasons. They need to be ported to many different hardware platforms from uni-processor platform to heterogeneous multi-processor ones. They need also to implement full-featured functionality to support various embedded software.

A similar distinction exists on the hardware side where part of the design depends on software and needs to be isolated.

Designing the hardware/software interface layer requires designers that have detailed knowledge of both hardware and software. There aren't many such people in the world but they play a critical role in SoC design.

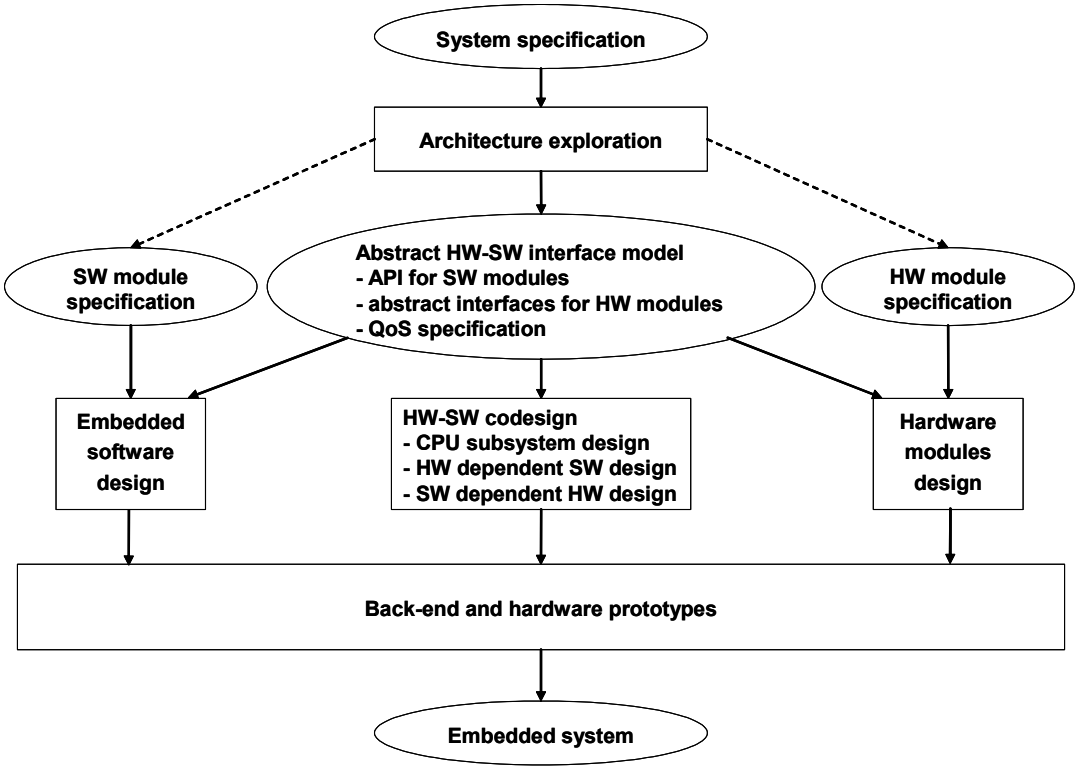

Figure 2. Concurrent HW/SW design flow

Figure 2 shows a simplified flow of mixed hardware/software design, where both software and hardware are designed concurrently. This scheme opens the design process to several new optimizations that were not possible when using the classical separate HW and SW design scheme. The most obvious optimization is a better adaptation of the CPU to both HW and SW interfaces. For example, new flexible processor technologies such as Tensilica [8] can be used to optimize performances of the HW/SW interfaces by introducing application-specific I/O operation. Another example may be the use of the capabilities of reconfigurable hardware, such as the Xilinx Virtex II Pro, to optimize hardware interfaces to an embedded CPU.

4. HW/SW interfaces codesign roadmap

HW/SW interfaces can be abstracted using different abstraction levels. Of course, the complexity of the codesign process will depend on the level at which the design starts.

Existing literature [4] and ongoing research work [8] [9] have identified clearly five abstraction levels that will constitute the key milestones for future research towards HW/SW codesign automation.

a) All explicit HW/SW interfaces: This is the currently used model for SoC design. HW is described as RTL modules. The CPU acts as the HW/SW interface and the software is binary code using an explicit memory and I/O architectures.
b) Abstract data transfer HW/SW interfaces model: At this level, the CPU is abstracted. HW and SW modules interact through exchanging transaction transported by explicit interconnect structure. This model is generally called Transaction Level Modeling (TLM). Several TLM languages exist, the most popular are those developed using SystemC [4]. Refining a TLM model down to RTL requires the refinement of the interfaces for different HW modules and the design of a CPU subsystem for each SW subsystem.

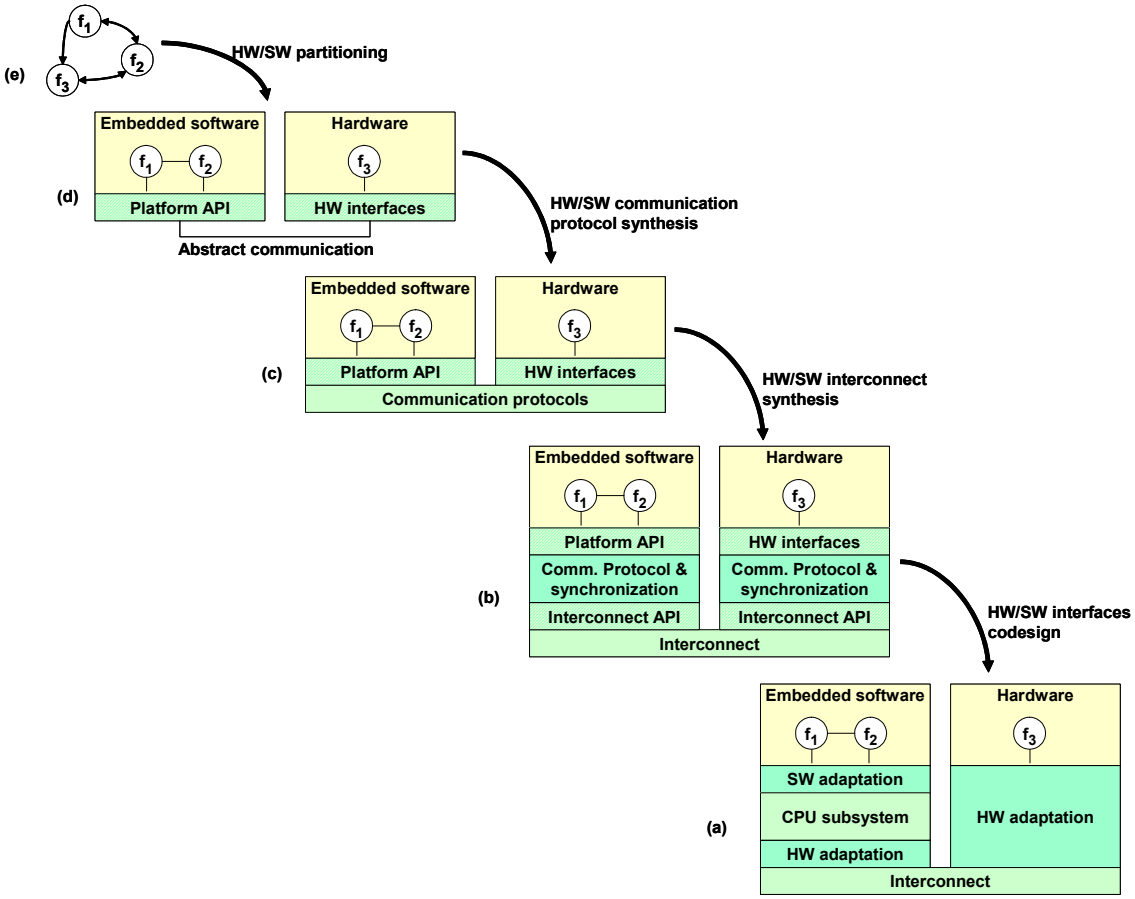

Figure 3. HW/SW interfaces abstraction levels and codesign steps

c) Abstract synchronization HW/SW interfaces model: At this level, the interconnect and synchronization are abstracted. The HW and SW modules interact by exchanging data following well defined communication protocols. MPI [4] is an example model that is to abstract synchronization and interconnect when specifying HW/SW interfaces. Refining an abstract synchronization HW/SW interfaces model requires first to design the interconnect (bus or network-on-chip) and fix the synchronization schemes. This will also need to refine the data transfer down to RTL.
d) Abstract HW/SW communication: At this level, the communication protocol is abstracted. The HW and SW modules interact by exchanging abstract data without assumption on the protocol used, the synchronization and the interconnect that will be implemented. SDL [4] is a typical model to abstract communication. Refining an SDL

model requires first to select a communication protocol, e.g. message passing or a shared memory and then go through all the refinement steps listed above.

e) Abstract HW/SW partitioning: The ultimate abstraction level is the functional model where HW and SW partitioning is not decided. Wide variety of models may be used to abstract HW/SW partitioning. This may range from sequential programming languages such as C/C++, concurrent languages, and higher level models such as algebraic notation, e.g. the B. language. Refining such a model requires first to fix which function needs to be software and which function needs to be hardware and perform all the refinements listed above.

So far SW design focused mainly on levels (b) to (e) and HW design tried to move the design to a higher level than (a). Of course, the ultimate goal would be to handle both HW and SW at all abstraction levels. A full HW/SW codesign scheme is detailed in Figure 3. Traditional HW/SW codesign research concentrated on HW/SW partitioning, but without solving the problem of abstracting the hardware platform. Rather than using ad-hoc models of hardware as has been done with traditional co-design, system-on-chip designs demand a well-thought-out approach to the hardware/software interface. The next step in automation is the synthesis of data transfer, then synchronization and interconnect, then synthesis of communication and finally HW/SW partitioning.

5. Summary

Moving from ASIC to highly parallel heterogeneous systems-on-chips requires a paradigm shift. Separate software and hardware approach cannot meet the requirements of system design: high performance/low cost, short design cycle, and best volume in the market. Thus, mixed hardware/software codesign is required. The enabler is a new technology---abstract HW/SW interface technology. It enables both hardware and software integration and concurrent hardware and software design. This technology requires a long term roadmap to cover different abstraction levels. In case of success, it will open new vistas leading to master the ever growing complexity of embedded systems.

Acknowledgement

The author would like to acknowledge the help received from Prof. Wayne Wolf through deep editing of this paper, and all the members of the System-Level Synthesis group of TIMA, and especially Sungjoo Yoo, Wander Cesário and Xi Chen for their help in preparing this document.

References

[1] Hiroe Iwasaki, et al., “Single-chip MPEG-2 422P@HL CODEC LSI with Multi-chip Configuration for Large Scale Processing beyond HDTV Level”, Proc. DATE (Designer’s Forum), 2003.

[2] M. Youssef, S. Yoo, A. Sasongko, Y. Paviot, A. A. Jerraya, “Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study”, in Proc. Design Automation Conference, 2004.

[3] A. Clouard, “Functional and Timed Transactional-Level SoC Models in SystemC 2.0”, 5th European SystemC Users Group Meeting, Mar. 2002.

[4] D. Skillicorn and D. Talia, “Models and Languages for Parallel Computation”, ACM Computing Surveys, vol. 30, issue 2, pp 123 – 169, 1998.

[5] A. A. Jerraya, W. Wolf, « Multiprocessor Systems-on-Chips », Morgan Kaufmann Publishers, ISBN 0-12-385251-X, September 2004.

[6] W. Wolf, “Computers as Components”, Morgan Kaufmann Publishers, ISBN 1-55860-541-X, 2001

[7] H. Jones, “Analysis of the relationship between EDA Expenditures and Competitive Positioning of IC Vendors for 2003”, http://www.edac.org/resources_profitability.jsp

[8] C. Rowen, “Engineering the Complex SoC”, Prentice Hall, 2004.