Guest Editorial: Special Issue on Concurrent Hardware and Software Design for Multiprocessor SoC

System-on-Chip (SoC) design complexity threatens continuation of current design schemes. In fact, designing SoCs requires concurrent design of complex embedded software and a sophisticated hardware platform that may include several heterogeneous CPU subsystems. The lack of early coordination between different teams belonging to different cultures causes delay and cost overheads that are no longer acceptable for the design of embedded systems. Programming models have been used to coordinate software and hardware communities for the design of classic computers. A programming model provides an abstraction of HW–SW interfaces and allows concurrent design of complex systems composed of sophisticated software and hardware platforms. Examples include API at different abstraction levels, RTOS libraries, drivers, typically summarized as hardware-dependent software. This abstraction smoothes the design flow and eases interaction between different teams belonging to different cultures, hardware, software, and system architecture. The abstract hardware/software interface model defined in the early stage of the design process will then act as a contract between separate teams that may work concurrently to develop the hardware and the software parts. In addition, this scheme eases the integration phase, since both hardware and software have been developed to comply with a well-defined interface. The main benefit of concurrent hardware and software design methods are to reduce design time and cost, in addition to making the handling of complexity easier.

1. MULTIPROCESSOR SYSTEM-ON-CHIP (MPSOC)

Of new ASICs in 130-nm technology, 90% already include a CPU. Multimedia platforms (e.g., Nomadik and Nexperia) are already multiprocessor system-on-chip (SoC) using different kinds of programmable processors (e.g., DSPs and microcontrollers) [Jerraya and Wolf 2004]. Heterogeneous cores are exploited to meet the tight performance and cost constraints.

This trend of building heterogeneous multiprocessor SoC will even accelerate. 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.

Industrial products show that multiprocessing is no more the domain of high-end research labs. A survey made by Embedded System Design [Turley 2005] has shown that only 52.4% of projects are made of a single CPU, while 20% of SoC include two CPUs, and 25% include three or more CPUs. The survey also shows that only 40% of multiprocessor projects use a homogeneous configuration, i.e., the same processor duplicated. The other 60% use a mix of different CPUs, e.g., DSP and microcontrollers. Even if this study deals with a general

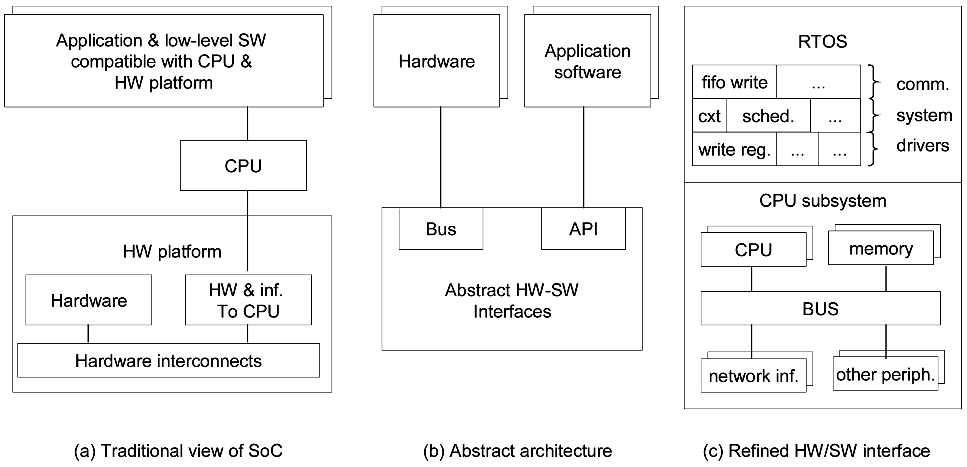

Fig. 1. HW/SW interface refinement for a general SoC architecture.

embedded system design and not single-chip designs, the trend is clear: future SoC will be heterogeneous MPSoC that integrates multiple CPU types (DSP, microcontrollers) on the same chip. This may get even more complex by using specific instances of reconfigurable processors, where each CPU will be tailored to the software functions it will execute [Turley 2005].

2. DESIGN PRACTICES

Designing and programming heterogeneous multiprocessor SoC is now becoming a major challenge for several reasons [Bouchhima et al. 2005]. In conventional SoC design approaches, hardware and software are usually considered separately as two "monolithic" blocs, with the CPU as ultimate interface between the two (Figure 1a). This traditional view of the SoC architecture has many disadvantages in a heterogeneous MPSoC context. In fact, the monolithic aspect makes the dependencies inside and across the two sides (HW and SW) implicit, very mangled and spread among unstructured elements. This hardly limits the flexibility (i.e., the ability to cope with new architectures) and leads, generally, to inefficient designs and huge overhead, in case heterogeneous processors are used with multiple software stacks.

Ideally, one would like to separate hardware-design issues from software design-related aspects. The abstract architecture concept (Figure 1b) allows such decoupling by describing SoC architecture as a set of hardware and software subsystems communicating through abstract hardware/software interfaces.

The abstract HW/SW interface hides the RTOS that provides the programming model for SW teams and the CPU subsystems that provide bus interfaces for hardware design teams.

The design of hardware and software components may then be done by two separate teams while the integration step will consist in using an existing platform or refining the abstract hardware/software interface. This includes the selection of RTOS and the CPU subsystems required to execute the software. In case of heterogeneous MPSoC, the software architecture may require the use of multiple software stacks requiring different RTOS [Wolf 2006].

In practice, hardware and software are designed by separate teams. This scheme leads to one of two situations:

(1) The hardware and software are designed concurrently with no early coordination. This will lead to an expensive integration step and increases both cost and time-to-market.
(2) The hardware is designed first and the software is then designed in a second step using the hardware platform to validate the software. This leads to a longer design cycle, which conflicts with time-to-market constraints imposed by most MPSoC applications.

3. THIS ISSUE

The seven papers in this issue cover a broad spectrum of approaches aimed at enabling concurrent hardware and software design.

The first two papers introduce complete methodologies and flows to design an application-specific MPSoC starting from a high-level model of an application. Xu and Wolf base their approach on Network-on-Chip concept to produce an MPSoC architecture providing the specification of abstract hardware/software interfaces. Kangas et al. start with UML and provide and use a library-based approach to produce application-specific architecture mapped on FPGA platform that includes a distributed application software and a physical implementation of an MPSoC architecture.

The third and fourth papers deal with hardware and software integration by targeting high-level software applications on a hardware platform. Hua, Qu, and Bhattacharyya propose an energy-efficient compilation approach to map a software application given as a sequential program on an MPSoC architecture. Hessel et al. tackle the case of a software application given as multiple threads using a parallel programming model. They generate an application-specific RTOS to adapt the high-level software to the application.

The fifth and sixth papers deal with early validation techniques allowing the cosimulation of high-level hardware and high-level software before the availability of the detailed models of hardware/software interfaces. These techniques may also be used for accelerating the simulation of complex design. Ou and Prasanna make use of an architecture-level model to abstract both hardware and software. This allows them to speed up the cosimulation process and allows for architecture exploration in the case of configurable multiprocessor SoC. Loghi, Poncino, and Benini make use of an abstract multiprocessor simulation platform to evaluate the effect of architectural decisions and, especially, cache coherence trade-offs in shared memory.

The paper by Lapalme, Aboulhamid, and Nicolescu introduces an infrastructure to ease the development of concurrent hardware and software design methodologies. They propose a ".Net" as a standard to enable creation and customization of simulation tools for mixed hardware/software applications.

AHMED JERRAYA
TREVOR MUDGE