Long Term Trends for Embedded System Design

Ahmed A Jerraya
TIMA Laboratory
46 av. Felix Viallet 38031 Grenoble France
Ahmed.Jerraya@imag.fr

Abstract

An embedded system is an application specific electronic sub-system used in a larger system such as an appliance, an instrument or a vehicle. An embedded system is generally made of software (called embedded software) and a hardware platform. The evolution of technologies is enabling to the integration of complex platforms in a single chip (called System-on-Chip, SoC) including one or several CPU sub-systems to execute software and sophisticated interconnect in addition to specific hardware sub-systems.
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. This paper analyzes this evolution and defines long term roadmaps for embedded system design.

1. Introduction

Paradigm shift: from ASIC to SoC design.

90% of new ASICs already include a CPU in 130nm technology [IBS2002]. 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) [JW2004]. 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 network-on-chip (NoC). With such an evolution, future design methodologies will use (dedicated or programmable) processors as basic components instead of logic modules (gate, operator) used by the current methods. So the design of a SoC will consist of an assembly of processors executing tasks concurrently. 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.

Challenge in Raising Abstraction Levels for SoC

These new generations of designs are software intensive and feature both short market window and ever increasingly complex functionalities and reliability. 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. The designers will have a significant problem in delivering competitive products to the market that respect at the same time, short time-to-market, low cost, and complex and reliable designs for multi-processor SoC. The challenges clearly lay in innovating design methodologies.

Figure 1 shows the evolution of abstraction levels in chip design[1]. From layout level to RTL, the abstraction has been applied to hardware components and interconnections (Fig. 1. a). In terms of interconnection, wires are abstracted from metal lines at layout level to signals at Register Transfer Level (RTL).

Above RTL, the abstraction needs to be applied to both software and hardware components (Fig. 1.b). Because software components are running on processors, the abstraction of interconnection between software and hardware components is totally different from the existing abstraction of wires between hardware components. In fact, software components do not communicate with the external world via wires, but via function calls (device driver) or assembly instructions (load/store). HW/SW interface needs to handle two different interfaces: one on the software side using API and one on the hardware side

[1] Figure 1 is made of F. Schirrmeister's S graph that covers the abstraction levels from layout level to RTL.

IEEE COMPUTER SOCIETY

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. Thus, a new type of designer, i.e. HW/SW integration designer is required. In terms of maturity of design technique, HW/SW interface design is still one of bottlenecks in SoC design and, thus, needs to be mastered [DAC2004].

In case of multiprocessor SoC (MPSoC) design will be integrating multiple software/hardware subsystems. Application software tasks will be distributed over heterogeneous processors in MPSoC. They may communicate with each other via network-on-chip. In this case, the HW/SW interface and the CPU subsystem needs to handle the interaction between software tasks and network-on-chip. To enable such a communication, HW/SW interface gives to application software layer an abstract view of MPSoC architecture, i.e. a parallel programming model. HW/SW interface includes also network interface (for multiprocessor booting as well as inter-processor communication) to enable the interconnection with network-on-chip. In MPSoC design, HW/SW interface design becomes much more complicated than in the single processor case since parallel programming models and network interface need to be considered in addition to conventional HW/SW design. Thus, HW/SW interface design may become a significant bottleneck in MPSoC design. A recent case study in MPSoC design presents this problem [1]. Thus, the key challenge to heterogeneous MPSoC design will be HW/SW interface design.

Current practice of SoC design makes use of a sequential scheme. After the hardware platform design is finished, an operating system and/or middleware is chosen and tested on the hardware platform, then the software is ported on the operating system and/or middleware. In this practice, since the software design is done only after the hardware platform design is finished, the length of the design is extremely long and not acceptable in most cases [DAC2004].

Current practice to achieve reliable systems design is validation by physical prototype. However, it takes a long design cycle to obtain the physical prototype. Such a long design cycle prevents the designer from testing all the critical reliability properties thoroughly. To achieve reliable systems design, the designer needs to consider quality-of-service (QoS) from the beginning of system design to the implementation. The QoS covers three aspects: communication requirements (e.g. bandwidth; delay, jitter), reliability requirements (e.g. hard or soft real-time, communication ordering and delivery), and cost requirements (e.g. energy consumption, software code size).

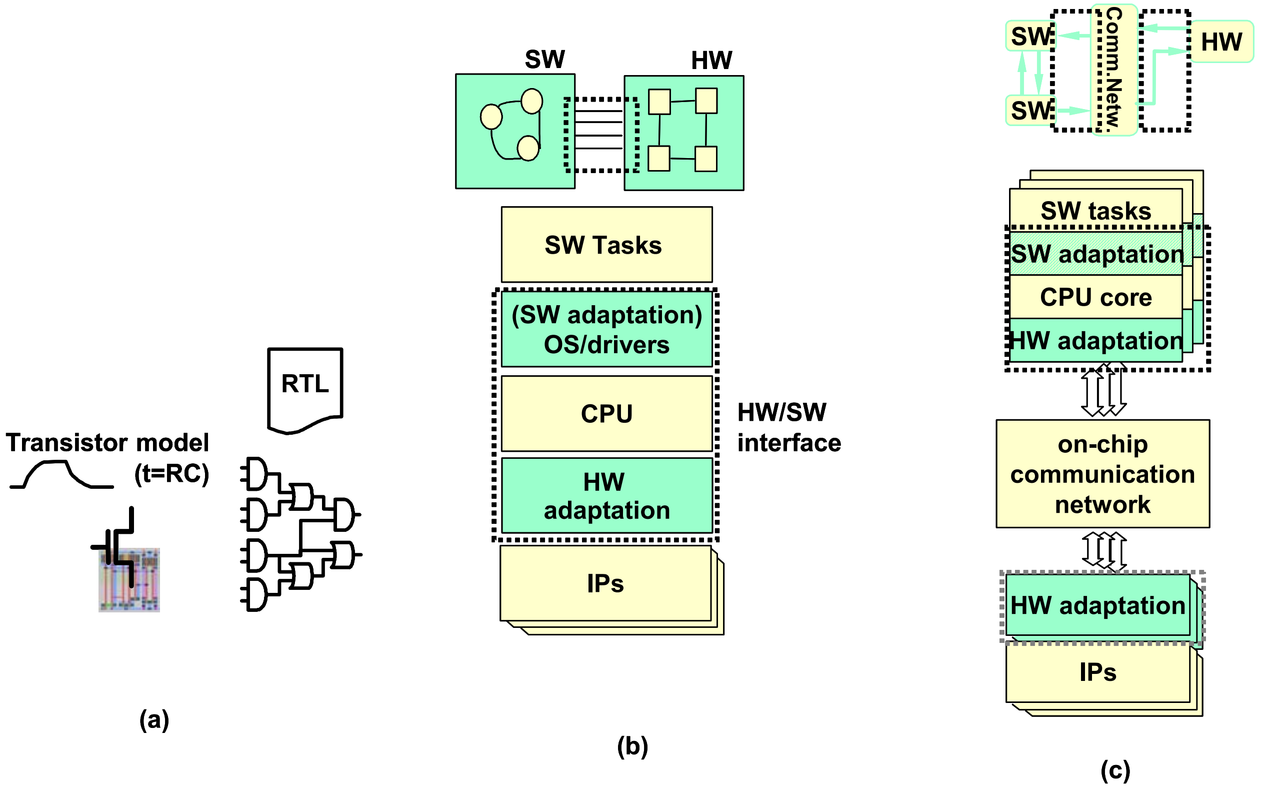

Figure 1. Evolution of abstraction levels in chip design

Hardware Dependant Software: The key technology for future designs

The key issues when integrating the parts of an SoC is the creation of a continuum between embedded software and hardware. This requires a new technology, namely Hardware dependent Software (HdS) for integrating embedded software to hardware platforms. The HdS concept is a promising innovative approach for the SoC integration. This concept facilitates

- Concurrent design of both hardware and embedded software leading to a shorter time-to-market.
- Modular design of hardware and software parts leading to a better mastering of complex systems.
- Ease global validation of SoCs including hardware and embedded software leading to a higher reliability and a better quality of services.

Even if the concept has existed for classical hardware software interfaces design. It is application to SoC brings new challenging issues that needs to be explored

- **HdS API as a Parallel Programming Model for SoC**
  The application programming interface (API) gives to upper software layer (i.e. application software) an abstraction of SoC architecture and HdS. In case of SoC, this is called HdS API. Compared with conventional parallel programming models (e.g. message passing interface, OpenMP, BSP, LogP, etc.) which are used on general purpose machines, the HdS API needs to allow to specify the application-specific design constraints of multi-processor SoC, i.e. quality of service constraints in terms of energy consumption, runtime, cost, reliability, etc.

- **Hardware dependent SW for Multiprocessor SoC**
  The HdS concept is quite well understood for single processor subsystems. The challenge will be to abstract complex heterogeneous multiprocessor platforms. In this case, HdS may become very complex. It needs to cover inter-subsystems communication. For multiprocessor systems, HdS API and HdS design becomes more complex. Thus, maintaining the same HdS API and designing HdS over different hardware platforms becomes a challenging task. To cope with the challenge, we need **a scalable/configurable architecture of HdS** to enable the abstraction of multiprocessor platforms. In case of heterogeneous multiprocessor, we may need to use a different HdS API for each sub-system. In this case, the overall programming model is made by the composition of different HdS APIs.

- **QoS-aware HdS API**
  Embedded software needs complex QoS (Quality of Service) requirements including high reliability from hardware platform. The current practice is to use an existing OS or middleware for validation of non functional properties of application software. These generally support real time and delay requirements. Unfortunately, other non functional properties such as inter-subsystem communication bandwidth, jitter and reliable communication are not supported. With current practice, QoS requirements of embedded software are validated only when the physical hardware prototype becomes available. Such a practice does not allow to monitor and to guarantee the required QoS in a systematic manner. Especially, the reliability of HdS design itself is hard to guarantee. Thus, to overcome the challenge, we need **a QoS-aware HdS API**.

- **HdS Customization**
  A fairly general HdS can be useful and applicable in a variety of applications. The gain of such a general HdS is that application software, hardware components and platform and middleware modules can be reused across different products, product families and even application domains. The drawback that comes along with generality is inefficiency. For applications that require only a small sub-set of the complete HdS functionality, the HdS will carry a tremendous overhead that cost-sensitive applications cannot tolerate. To mitigate this problem, we need to develop an HdS architecture that is highly configurable to allow to optimize and streamline an HdS instance for the particular needs of a given application. This is a central issue and critical to the success of HdS concept because without an efficient method to configure and optimize the HdS, the concept will not be accepted. Thus, we need to integrate the configurability of HdS right from the beginning into the development of the architecture and the modules.

- **HdS Verification**
  The correctness verification of HdS itself imposes new challenges because HdS needs to be verified according to a given hardware platform.

Of course, we cannot wait until the hardware prototype is available to carry out this verification. Thus, we need a new method of verifying the correctness of HdS design itself without using the physical prototype.

- **Standardization**
  Since HdS links separate teams that may belong to different companies or even market sectors, standardization will become a crucial issue. For example, a car maker would like to use a standard HdS-API to develop his software while keeping the right to select the hardware platform as late as possible. Currently, though the industry needs on HdS are getting more and more growing, there is a lack in standards on HdS technology including the standard of API and HdS design/verification method. Unlike classical embedded software, where there are standards or de-facto-standards regarding inter-process-communication (CORBA), operating system interface (POSIX), software analysis and design methodology (OOA and OOD using UML), etc. However, there is very few existing initiatives dealing with HdS for SoC. More recently, MIPI and SPIRIT initiatives are trying to fill this gap.

2. Preliminary: SoC Platform vs. Embedded Software

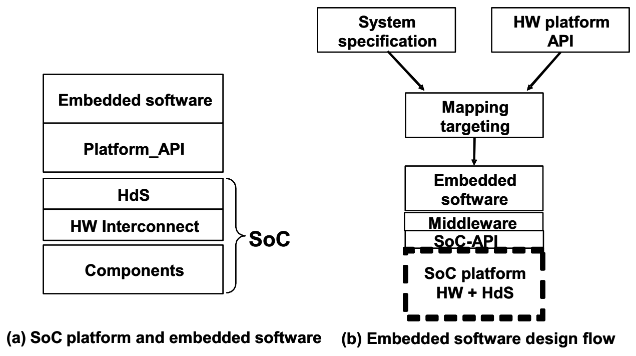

Figure 2. SoC platform vs. embedded software

In terms of design cost, the operating system and middleware (e.g. Qualcom BREW) are starting to consume a significant portion of system cost (e.g. code size, runtime, energy consumption, etc.). The reason why their overhead is growing is two-fold. They need to be ported on 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. Especially, embedded software needs more complex (Quality of Service) QoS (e.g. reliable inter-processor communication, control of energy consumption, etc.) from the hardware platform. For QoS reasons, many SoCs require a customization of a large part of these components. This is the case in multimedia and wireless for example.

Most people are asking whether the design techniques of embedded software can be applied to multi-processor SoC design. Our answer to this question is partially yes. In this section, we explain the relationship between SoC platform and embedded software to explain our viewpoint.

Figure 2 (a) shows the relationship. SoC can consist of two parts: SoC platform and embedded software. Embedded software assumes a fixed platform and runs on the platform. Platform API (e.g. OMAP API) provides a programming model to build embedded software. The API can be specific to the application and/or platform. The SoC platform consists of hardware components, interconnect, and hardware-dependent software (HdS). HdS is the low layer of software usually known as system software. HdS provides embedded software with Platform API. HdS is provided by SoC designer. Hardware interconnect layer consists of interconnects and on-chip network. Components are CPU, specific hardware, memories, etc.

Figure 2 (b) shows embedded software design flow. Given a system specification and a fixed hardware (SoC) platform (represented by platform API), the embedded software is reused, mapped and targeted on a possibly multi-processor hardware platform. A middleware layer (e.g. CORBA, JVM) can be designed to provide embedded software with abstract services.

As Figure 2 shows, embedded software is a part of the entire SoC. Thus, the design techniques of embedded software need to be applied to SoC design. However, they may not give a complete design methodology for SoC that covers the low level software layers hidden by the SoC API.

3. Design Challenge: Mixed HW/SW SoC Design

To give a complete design methodology for highly parallel multi-processor SoC, **we need a key enabling technology for mixed hardware/software design of multi-processor SoC**. To explain this argument, we present the advantages and limitations of embedded software design and current ASIC design approaches in terms of three criterions: (1) high performance and low power design, (2) short design cycle, and (3) best volume in the market window.

High performance/low power design: Hardware is preferred

Future embedded systems will have very high computation requirements. For instance, a digital cinema system (which supports MPEG2 encoding of 2000x1000 pixels video and full motion search with 128x128 search window) will need more than 32 TIPS (Tera instructions per second) as computation complexity. Assume that embedded 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 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.

To design high performance and low cost systems, mixed hardware/software design is needed. It exploits both the hardware capability of high computation and low power consumption and the software capability of flexibility. As a real example, a mixed hardware/software design (with algorithm optimization) of MPEG2 encoder can give a four-chip solution to the digital cinema application [1].

Short design cycle: SW is preferred

It is known that best margin can be achieved when the SoC enters a new market early [2]. In the case of derivative design, embedded software approach gives an advantage in reducing the design cycle (3-6 months) since software is flexible enough to add new functionality. ASIC design approach suffers from long design cycle (6-12 months in derivative design and 18-24 months in first design). However, in derivative design, embedded software approach may not cover all the necessary design steps since new functionality may need to be implemented as hardware core when software implementation does not satisfy the constraints of performance and/or power consumption.

Best volume in the market: Mixed Hardware/Software solution

To achieve best volume in a given market window, two aspects need to be considered: chip size and yield in chip production. In terms of chip size, ASIC approach will give a small chip size while embedded software approach may yield a bigger chip or even a chip set. In terms of yield in chip production, both ASIC and embedded software approaches have pros and cons. ASIC approach may suffer from low yield in the first few months of chip production until the learning curve of yield improves. However, the reduced chip size may improve the total chip production. Embedded software approach may give a good initial yield since an already proven SoC platform is reused. However, a larger chip size may reduce the effects of yield improvement.

Considering the advantages and limitations of embedded software and current ASIC approaches, our conclusion is that no single software or hardware approach can meet all the requirements. Mixed hardware/software design is required. The mixed hardware/software design methodology will achieve **a convergence of existing techniques of embedded software and ASIC**.

At present, there are few systematic design techniques to enable such a convergence. Thus, a breakthrough is required.

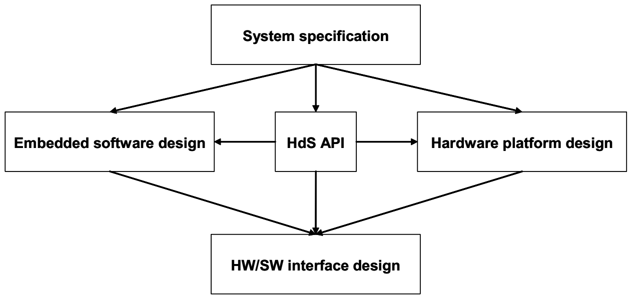

Figure 3. Concurrent HW/SW design flow

Figure 3 shows a simplified flow of mixed hardware/software design, where both software and hardware are designed concurrently.

Hardware/software concurrent development is important since it can reduce the time-to-market by enabling embedded software design and hardware platform design to be performed in a concurrent way.

Hardware/software co-development is enabled by hardware-dependent software. The HdS API (platform API shown in Figure 2) can be seen as a contract between embedded software designers and hardware platform designers. Thus, embedded software designers consider it to be an abstraction of the underlying execution platform and design the embedded software on top of it even before the hardware platform design is finished.

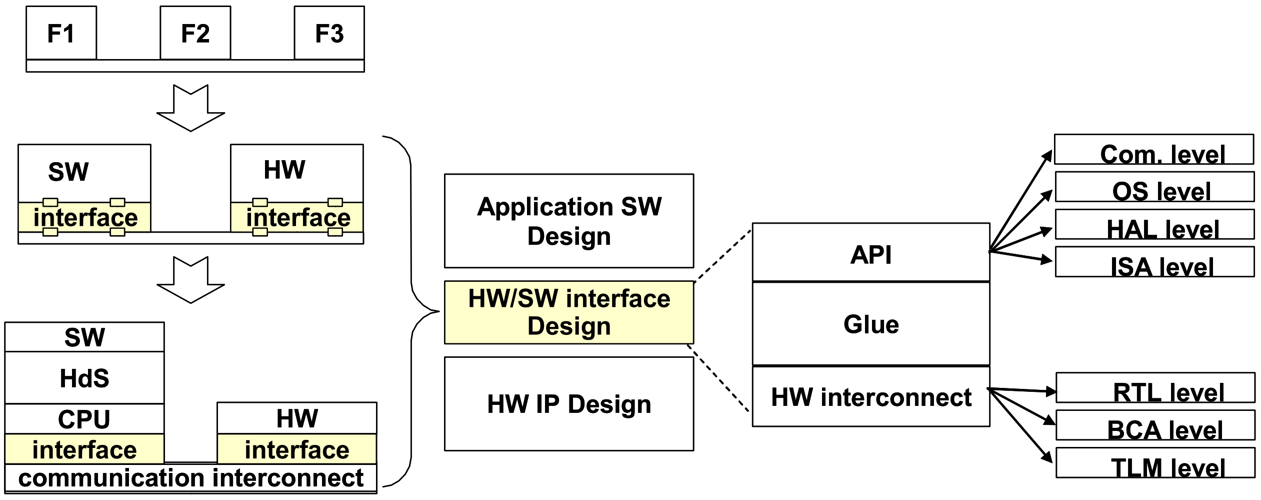

Figure 4. Abstraction levels of HW/SW Interfaces

To enable a true concurrent design of software and hardware, embedded software designers need to be able to validate their design even before the hardware platform design is finished. To achieve such an early design validation, the abstraction levels of both hardware and software need to be well defined. Figure 4 shows an example of potential abstraction levels of software and hardware. The figure shows four abstraction levels (communication, OS, HAL, and ISA) for software abstraction levels and three levels (RTL, BCA, and TLM) for hardware abstraction.

Well-defined abstraction levels enable to develop consistent simulation models of software and hardware. Early validation of embedded software can be done using the simulation models of hardware platform at different abstraction levels depending on the advancement of hardware platform design.

4. Long Term Trends

In this section, we present a roadmap for embedded SoC design. We expect that the abstraction continues to be raised as shown in Table 1. This table combines several studies. The first two columns resulted from a study of abstraction levels used in both hardware and software domains [4, 5]. Each line of this table corresponds to a specific abstraction layer. These are presented by increasing order of abstraction. The first columns give the concept abstracted by each abstraction level and a typical model used in the literature. The third column describes the technologies required to refine one abstraction layer to the next one in the table. According to the current state-of-arts and the difficulty of implementing the technology, we tried to build long term schedule (last column). Currently, transaction-level model (TLM) is being adopted as a higher abstraction level than RTL. TLM will serve to help to design HW/SW interface by accelerating the simulation of entire SoC [4].

SoC may have highly parallel architectures. Thus, after TLM, we expect that the trend will be to adopt parallel programming models as new abstraction levels for SoC design. Parallel programming models are required to describe the parallelism inherent in the application. In terms of design methodology, the significance of adopting parallel programming models is that SoC will be described in an architecture-neutral way that does not pre-determine specific software or hardware implementations.

There is a wide spectrum of parallel programming models depending on the level of abstractions [5]. Table 1 shows some of parallel programming models in the literature of parallel programming models. In terms of abstraction, there are six levels: interconnect and message control (equivalent to transport and session layers in OSI 7 layer model), communication (communication protocol, e.g. infinite or finite FIFO), mapping (physical location), decomposition (explicit or implicit representation of thread) and concurrency (explicit or implicit representation of concurrency).

The trend of adopting parallel programming models will be to adopt models from the most explicit ones to the least explicit ones. As the first parallel programming model to be adopted for SoC design will be some models like message passing interface [6] and OpenMP [7]. In terms of abstractions shown in Table 1, they gives an abstraction of interconnect/message control while fixing the details of communication, mapping, decomposition and concurrency. In terms of evolution of abstraction levels, these models give a nice continuum since TLM is an abstraction of interface and the next abstraction will be to abstract the interconnect.

Table 1 Roadmap of abstraction levels and key enabling technologies

To Abstract HW	To Abstract SW	Typical languages or models	Key Technologies	Year of marketing
Gate delay	All explicit	RTL	Logic optimization	1995
Interfaces wrappers	ISA, local architecture	TLM	HW-SW interfaces design & optimization	2005 - 2007
Interconnect/ Synchronization		Transport level/ MPI	Network-on-Chip -Network interface -Configuration	2005 - 2010
Communication		SDL	Communication computation Partitioning, communication synthesis	2010 - 2015
Mapping		BSP, LogP	Task allocation, scheduling automation	2015 - 2020
Decomposition		ParLog	Automatic partitioning	>>
Concurrency		Haskel, PPP	Execution model generation	>>

After transport and session layer models, the communication protocol will be abstracted while adopting some models like SDL. It is sure that some of those models have existed since dozens of years in other application areas like telecommunication. It is also noted that there are already a few people who try to apply those models like SDL, BSP, etc. to SoC design. However, the message of Table 1 is that adopting those models to SoC design will take a longer time that expected. It is because first we need to master key technologies such as HW/SW interface, network-on-chip, etc. Such technologies will allow to develop key enabling tools for those models, e.g. task partitioning, mapping, etc.

5. Summary

To cope with the paradigm shift from ASIC to highly parallel heterogeneous SoC, no single software and hardware approach can meet the requirements of system design: high performance/low cost, short design cycle, and best volume in the market. Thus, mixed hardware/software design is required. To enable such a design, a new design technology is required as a breakthrough. In this document, HW/SW interface technology is introduced as the breakthrough. HW/SW interface technology enables both hardware and software integration and concurrent hardware and software design, which are required for mixed hardware/software design of MPSoC. The roadmap of MPSoC design technology expects that the abstraction levels in chip design will continue to be raised by introducing high abstraction levels of parallel programming models and corresponding key enabling technologies.

Acknowledgement

This work is supported by French National project SystemC'Mantic and MEDEA+ project TOOLIP. The author would like to thank all the members of the System-Level Synthesis group of TIMA, and especially Sungjoo Yoo and Wander Cesário 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] J. Borel, "EDA: the Key Enabler for Future SoC Business", MEDEA+ Design Automation Conference 2003.
[3] M. Youssef, S. Yoo, A. Sasongko, Y. Paviot, A. A. Jerraya, "Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study", to appear in Proc. Design Automation Conference, 2004.
[4] A. Clouard, "Functional and Timed Transactional-Level SoC Models in SystemC 2.0", 5th European SystemC Users Group Meeting, Mar. 2002.
[5] D. Skillicorn and D. Talia, "Models and Languages for Parallel Computation", ACM Computing Surveys, vol. 30, issue 2, pp 123 – 169, 1998.
[6] The Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi/
[7] OpenMP, http://www.openmp.org/