Communication Refinement for SOC Design

Hocine Mokrani, Rabéa Ameur-Boulifa, Sophie Coudert
System-on-Chip Laboratory (LabSoC), GET/Telecom-Paris
2229, routes des Crêtes BP 193
F-06904 Sophia-antipolis Cedex
Email: first_name.last_name@telecom-paristech.fr

Emmanuelle Encrenaz-Tiphene
Laboratoire d'informatique de Paris VI
Université Pierre et Marie Curie (Paris VI)
F-75252 Paris Cedex 05
Email: emmanuelle.encrenaz@lip6.fr

***Abstract*— We present a methodology for efficient system-level design space exploration of system-on-chip (SoC). The methodology we proposed provides a formal approach for modeling and analysis system-level design. Indeed, it provides a mean to build models of architectures and application at an abstract level, to relate the two models with the mapping function, and to validate and analyze the performance of the resulting system.**

I. Introduction

To achieve a satisfactory design quality, along the increasing complexity of both the systems on chip (SoCs) and applications, design at high level of abstraction and verification methodologies are required. Because models are abstract, it is possible to speed up simulation and to apply formal analysis techniques. Our work focuses on formal verification capability to offer guarantees about a design. The methodology we advocate consists of developing abstract application models and architecture models, followed by a mapping step to relate the application model to the architecture. Afterwards, the system (application-architecture-mapping combination) is analyzed by using formal techniques. Once the system is validated, more details are introduced, the analysis is reiterated and so on. Our contribution: a new design dimension is added to the traditional methodology. The extended methodology involves formal design paradigm both to drive architecture refinement and exploration and to prove correctness of refinement steps.

II. Our Methodology

Our approach follows the general Y-chart scheme, as shown in Fig. 1, where application and architecture are provided separately. The application model drives the architecture design. Firstly, the designer studies the applications, makes some initial computation and analysis, and proposes a candidate platform architecture. By mapping the application onto the platform architecture, the designer evaluates and compares several instances of the platform. The evaluation indicates whether the selected plate-form and the design parameters satisfies given requirements which could be functional or non-functional. Separating application and architecture models allows designers to use a single application model to test different partitioning and map it onto a range of architecture models, possibly representing different instances of a single platform or the same platform instance at various abstraction levels. The strength of our methodology is to provide incremental design. It allows designer to start from a high-level plate-form (with abstract components), and gradually more detailed components will be introduced until convergence to the desired architecture.

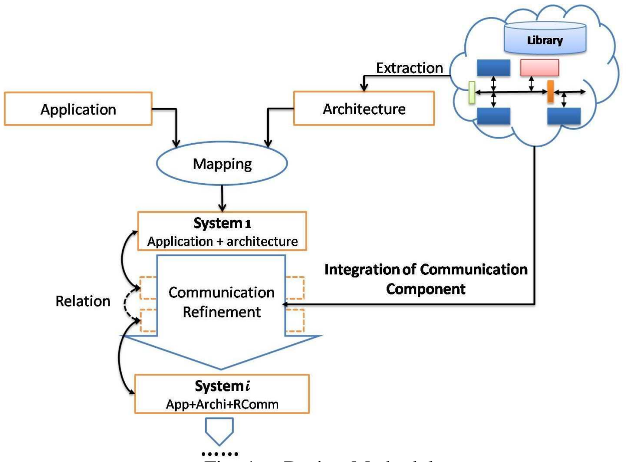

Fig. 1. Design Methodology

A. Application Modeling

To model the functional behavior of an application at high level of abstraction LabSoC has proposed a very simple language called TML [11] (Task Modeling Language). The TML model very close to Kahn processes network model [5] allows by using *tasks* and communication *channels* to capture the parallelism available at application-level assuming unbounded physical resources. At this level, there is no difference between hardware tasks and software tasks because no partitioning is defined yet. There is no data processing details inside the tasks. Operations within a task model are totally ordered. The proposed task modeling language consists of usual instructions (arithmetic/logic instructions, variable settings, tests, loops, etc), of communications instructions (reading/writing abstract data samples in channels, sending/receiving events and requests) and of computational instructions.

B. Architecture

An architecture is a set of interconnected hardware components. These components are the usual ones, they can be *Processing Elements (PE)* such as processors (standard or specific), hardware units, . . . , *Communication Elements (CE)* such as bus structures, *Memory Elements (ME)* such as RAM or FIFO buffer and *Interface Elements (IE)* such as Bridge, Arbiter, controller of interrupt.

C. Mapping

Once both an application model and an architecture model have been defined, mapping can be performed. The application is assigned to an architecture (final) after several iterations of refinement process which starts by the following rules: - Each *task* is mapped onto a process element (PE). This mapping can be many-to-one, in which case the tasks need to be scheduled by the PE. - Each *channel* is mapped onto one-to-one communication elements CE. This mapping can also maps the channels onto a combination of communication elements (CE) and memory elements (ME), possibly including user defined blocks, such as a bridge or controller. Typically, these do not have an equivalent element in the application model.

III. Refinement of Communication Channels

In its abstract form, the application is modeled as a set of communications processes free of any architectural constraints. As it is mapped onto an architecture the level of abstraction is moved down. So the communication mechanism must be taken into account since the behavior of the processes depends on the behavior of the communication channel. For instance, communication can be performed over a reliable channel connecting two processes with finite (or infinite) buffer. The level of abstraction is lowered further down by considering a more detailed instance for the channel, like bus architecture. The illustration is given in the example of the Fig. 2 (figure inspired from [7]). Consider an application which consists of two tasks connected via a channel as shown in Fig.2(a). In the first step, the application model is mapped to the target architecture. In this case, each task is mapped onto a single PE unit and the channel is mapped onto a finite buffer Fig. 2(b). In refinement step, the communication channel is transformed from the perfect buffer to a lossy medium Fig.2(c) where errors can occur.

In our framework, the *system* descriptions are given in an informal manner. These informal specifications need to be translated into formal models (e.g., process algebra, finite state machine, etc) which capture the essential properties of the application. The behavior of both the application and the architecture can be captured by the notion of traces. For an application a trace represents a sequence of computation and communication actions (operations) that are performed when it is executed. Concerning an architecture a trace a sequence of actions that are accepted to be executed on it. The mapping function translates the application traces into the architecture traces, i.e, the applications actions into more detailed architecture actions giving as a result traces permitted by the system.

Refining the communication medium from a simple channel to more detailed one will increase the communication operations and its relationship with computation. The idea is to

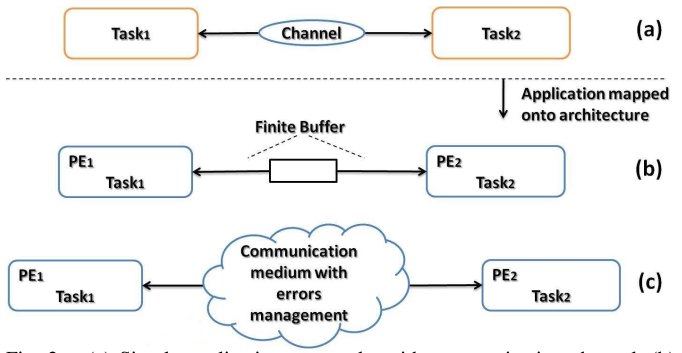
Task1
Channel
Task2
(a)
Application mapped onto architecture
Finite Buffer
PE1 Task1
PE2 Task2
(b)
Communication medium with errors management
PE1 Task1
PE2 Task2
(c)

Fig. 2. (a) Simple application: two-tasks with communication channel, (b) replacing channel with Finite Buffer, (c) adding errors

help designers check whether two systems at different levels of abstraction have any semantic inconsistencies that could have been introduced during the refinement process. So in order to ensure the compliance between the different levels of refinement, we need to give behavioral semantics of the related components and to formally characterize refinement steps. - The application model: set of traces that are generated by an application consisting of a sequence of observable actions performed by the application. - The architecture model: set of traces that are accepted by the architecture. - The mapping function as a series of well-defined steps (or rules) which transform an application trace to an architecture trace. Once an application is mapped onto an architecture the resulting system model will be set of traces that can be executed. As shown in Fig. 2 replacing the original communication by a concrete bus will involve several transformations which modifies its behavior. Typically, the data transfer action on an abstract channel will be converted to a sequence of bus transactions. Besides converting abstract data transfer to bus trace, it is necessary to preserve the communication semantics of the original model, i.e the synchronization between different components. For instance, on application level each abstract data transfer is independent from others. However, on a concrete bus they have to share the same medium and synchronization events. Therefore, additional synchronizations are required to avoid conflicts on the bus.

For more complex designs, like a design where several slaves components talk to several masters, the basic action of data transfer remain the same, but the refinement process requires more work. Indeed, it will take into account the arbitration mechanism, the interrupt controller, etc...

IV. Related work

There are various works and design methodology in the development of SoC that integrates formal verification of system in each level in the design process. Among these methods, we cite VeriAgent [9] which proposes an interface between UML description tools and formal verification tools with model checking, the Polis approach [3] seek the formal verification using FSMs, also we cite Metropolis [10], ArchAn [8], Ptolemy [2] and [6]. Some of these methodologies and works integrate formal method but no methodology provide mathematical relationship between different system-levels.

Alike Abrial's B Method [1] our approach dedicated for the SoC development is based on refinement technique allowing one to gradually introduce complexity into the design, incrementally verifying its correctness by automatic or interactive tools.

Much works has been done to design and experiment model of communication in architecture exploration. For example, [4] which proposes a practical approach to the communication synthesis for hardware/software systems. [6] which proposes a technique based on trace transformation for automate the communication refinement in architecture exploration.

V. Conclusion and Future Directions

The work presented in this paper is built upon methodology developed by our team LabSoC. It focuses on the integration of formal refinement in a system-level design methodology. The design methodology proposed supports refinement based design, and formal verification to prove correctness of refinement steps. We are currently formalizing the presented concepts (application, architecture, mapping) to support communication refinement in the mapping steps. We plan to automate this process by way of automatically generating refinement directives.

References

[1] J.-R. Abrial. *The B-book: assigning programs to meanings*. 1996.

[2] C.P. Cheng, T.Fristoe, and E.A. Lee. Applied verification: The ptolemy approach. *Technical Report No. UCB/EECS-2008-41*, April 19, 2008.

[3] F.Balarin, M Chiodo, P. GIUSTO, H.Hsieh, A. Jurecska, L. Lavagno, and al. hardware-software co-design of embedded system : the polis approach. may 1997.

[4] Denis Hommais, Frederic Petrot, and Ivan Auge. A practical tool box for system level communication synthesis. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 48–53, 2001.

[5] Gilles Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[6] Paul Lieverse, Pieter van der Wolf, and Ed Deprettere. A trace transformation technique for communication refinement. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 134–139, 2001.

[7] Radu Marculescu, Ümit Y. Ogras, and Nicholas H. Zamora. Computation and communication refinement for multiprocessor soc design: A system-level perspective. *ACM Trans. Design Autom. Electr. Syst.*, 11(3):564–592, 2006.

[8] Yves Mathys and André Châtelain. Verification strategy for integration 3g baseband soc. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 7–10. ACM, 2003.

[9] Edjard Mota, Edmund M. Clarke, Alex Groce, Waleska Oliveira, Marcia Falco, and Jorge Kanda. Veriagent: an approach to integrating uml and formal verification tools. *Electr. Notes Theor. Comput. Sci.*, 95:111–129, 2004.

[10] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pages 667–672. ACM, 2001.

[11] W.Muhammad, L. Apvrille, R.A. Boulifa, S. Coudert, and R. Pacalet. Abstract application modeling for system design space exploration. *Euromicro Conference on Digital System Design (DSD?06), Dubrovnik, Croatia*, August 2006.