A refinement approach to design and verification of on-chip communication protocols

Hocine Mokrani, Rabéa Ameur-Boulifa
Institut Telecom, Telecom ParisTech, LTCI CNRS
Email: first_name.last_name@telecom-paristech.fr

I. Introduction

In this paper we focus on the exploration stage in the design of embedded systems. In particular, we study the circumstances under which communication refinement can be performed efficiently to achieve a satisfactory design quality. We present a new methodology for efficient system-level design space exploration of system-on-chip. The methodology provides an infrastructure to define a design at different levels of abstraction, as well as mathematical relationship between different system-levels.

Using the proposed methodology, the verification is driven with the design process in a joint way. We start with an abstract model for a basic protocol that can be formally verified with reasonable effort; this is then enriched with advanced features and details step-by-step to meet concrete model. The correctness of the enriched design is obtained from the correctness of its previous version and refinement relation between two consecutive versions.

II. Modeling, mapping and analysis

To begin with, the system model is given initially in an informal manner. This informal description is translated later into formal description which is amenable to analysis and verification. Usually, such a representation is based on states and transitions, it models an abstraction of the system behavior. The model should be free of any architectural constraints, and it should capture the essential properties of the application assuming an infinite resources.

Once the application is mapped onto an architecture, the first plateform becomes available. Then, the analysis is used to determine whether or not the chosen application-architecture combination satisfies the required design constraints. If the design constraints are not met, then the mapping process is reiterated with a different set of parameters until the desired results are obtained.

A. Application

The functional behavior of an application is described by the TML (Task Modeling Language) language. Using *tasks* and communication *channels* the TML model captures the parallelism available at application-level assuming unbounded physical resources. At this level of abstraction, there is no differentiation between hardware tasks and software tasks because no partitioning is defined yet. There is no data processing details inside the tasks. The language modeling a task consists of usual instructions (arithmetic/logic instructions, variable settings, tests, loops, etc), communication instructions (reading/writing abstract data samples in channels, sending/receiving events and requests) and computational instructions. The instructions within a task are totally ordered.

B. Architecture

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

C. Mapping

Once the application model is completed and an architecture is selected, the designer maps the application onto the architecture by applying rules. These rules can be provided by the designer. Example of rules: - Each *task* is mapped onto a process element. - If several tasks are mapped on a processor, this one must be endowed with a scheduler. - Each *channel* is mapped onto one-to-one communication elements. It is also possible to map a channel onto a combination of communication elements and memory elements, or by including blocks, such as a bridge or a controller.

Once the obtained results of the analysis are satisfactory, we proceed to the refinement step of communication units. So the level of abstraction is lowered further down by considering a more detailed instance for the communication protocol.

III. Communication Refinement

We illustrate the communication refinement by means of the following example. Consider the Producer-Consumer application of which the model TML is shown in Figure 1 (a). TASK1 does some computation and writes data to its output channel. TASK2 reads data from its input channel and does some computation. The two tasks are connected by the channel C1 of type **NBR-NBW** (Non-blocking Read - Non-Blocking Write, i.e, a memory).

Suppose that this application is mapped onto an architecture consisting of two processors CPU1 and CPU2 and a memory MEM, as depicted in Figure 1 (b). The TASK1 is mapped onto CPU1 and the TASK2 onto CPU2 and the channel is mapped

onto a bus with centralized arbitration. The bus arbitration scheme used here is named *daisy chain arbitration*. The two processors act as masters and the memory acts as slave.

The master or masters assert the signal REQUEST when they request the bus. The bus arbiter returns the GRANT signal, which passes through each of the masters which can have access to the bus, as shown in Figure 1 (b). Here, the priority of a master depends solely on its position in the daisy chain. If two or more masters request the bus at the same time, the highest priority master is granted the bus first, and then the GRANT signal is passed further down the chain. The signal RELEASE is used to indicate to the bus arbiter that the first master has finished its use of the bus. Holding REQUEST asserted indicates that another master wants to use the bus.

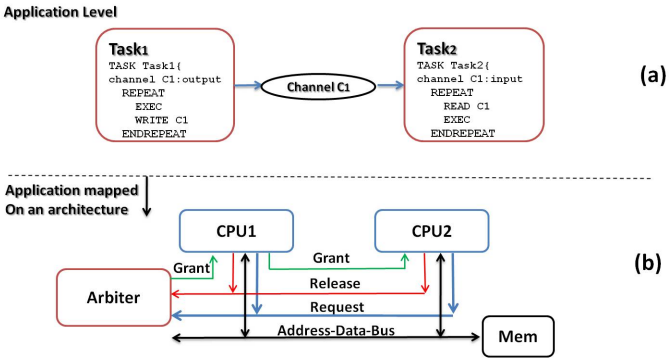

Fig. 1. (a) Producer-Consumer application, (b) replacing the channel with bus

Consider the application layer and the plateform layer two consecutive levels of refinement. Converting a simple channel to a concrete bus might increase the number of data-transfers and synchronization events between components. Our goal is to help the designers to check whether two systems given at different levels of abstraction preserve the communication semantics. That is to say no semantic inconsistencies is introduced during the refinement process. In order to define formally the notion of refinement wrt linear or tree semantics, so it captures the concept of execution, like the concept of "less non-deterministic", we need to give behavioural semantics of the related components and to formally characterize refinement steps.

IV. SEMANTICS

A system is behaviorally modeled as a set of concurrent state machines called labelled transitions systems one machine per component (task or channel). Each task is characterized by a set of states and a set of transitions between states. Each transition is labeled by an action representing the instruction executed by the component.

Definition 1: A labelled transition system (LTS) is a 4-tuple $\langle S, s_0, A, \rightarrow \rangle$ where: S is a non-empty set of states; $s_0 \in S$ is the initial state; A is a set of labels; $\rightarrow \subseteq S \times A \cup \{\tau\} \times S$ with $\tau \notin A$, is the transition relation.□

Thereby, the alphabet A consists of communications primitives (read and write) and computation function (exec).

The global behavior of the system is built by composition of the LTSs of all the components building it up. As example, the drawing depicted in Fig.2 is a part of the model of the application.

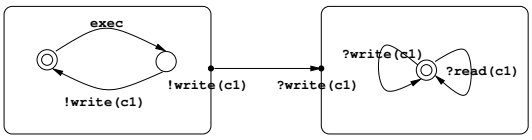

Fig. 2. Models for **Task1** (left) and channel **C1** (right)

There are various relations of equivalence and preorders on labelled transitions systems [1]. These semantics (linear time semantics or branching time semantics) are defined in terms of a function that associates to a process (an LTS) a set of the possible observations one could make when executing the process.

Let us consider here a semantics encoding the branching structure of a process by representing the set of actions that can be accepted after a trace is executed. Before giving the definition of this semantics called *Readiness*, let us first mention some definitions.

A function that returns an acceptance set : the set of possible actions that can be accepted after state s is reached.

Definition 2: The function $next_{LTS}(s)$ is defined by: $next_{LTS}(s) \stackrel{def}{=} \{e \in A \mid \exists s' \in S. s \xrightarrow{e} s'\}$.□

An LTS might perform a trace σ and arrives in a state for which X is the set of actions that are possible next. The set of all such pairs (σ, X) defines the readiness of LTS.

Definition 3: The readiness of an LTS is defined by: $Readiness(LTS) \stackrel{def}{=} \{(\sigma, X) \in A^* \times \mathcal{P}(A) \mid \exists s' \in S, s_0 \xrightarrow{\sigma} s' \wedge X = next_{LTS}(s')\}$. □

If LTS_1 and LTS_2 are LTSs, then we say that LTS_2 is readiness refinement of LTS_1, written $LTS_1 \sqsubseteq LTS_2$, if every acceptance of LTS_2 is also an acceptance of LTS_1.

Definition 4: $LTS_1 \sqsubseteq LTS_2$ if $Readiness(LTS_1) \supseteq Readiness(LTS_2)$. □

Building the models: We build the behavioural model of the whole systems at each level: we obtain a LTS (LTS_{App}) modeling the application and a LTS (LTS_{Pfm}) modeling the plate-form. We compute their respective readiness sets. We find out without any surprise that $Readiness(LTS_{App}) \nsubseteq Readiness(LTS_{Pfm})$. Indeed, we have $(exec, \{write\}) \in Readiness(LTS_{Pfm})$ but $(exec, \{write\}) \notin Readiness(LTS_{App})$. Therefore, $(LTS_{Pfm} \not\sqsubseteq LTS_{App})$. This is due among others to the choice of bus arbitration that is not very fair: the second master (the low-priority master) may be locked out indefinitely.

V. Conclusion and Future Directions

In this paper, we present a design flow for architecture exploration that supports formal refinement. Besides, of the designing the proposed methodology offers guarantees as for preservation of the properties along the flow. Compared with existing works works [4], [3] and [2], our work contributes at the level of correct-by-construction System-On-Chip.

As this work is in its initial stage, we need to examine more semantics and investigate the various properties which will be preserved by the different semantics.

In the future, we intend to fully automate the design of systems by automatically generating models and checking semantic consistencies between different levels.

References

[1] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra,Chapitre1*, pages 3–99. Elsevier, 2001.

[2] 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.

[3] Radu Marculescu, Ümit Y. Ogras, and Nicholas H. Zamora. Computation and communication refinement for multiprocessor soc design: A system-level perspective. volume 11, pages 564–592, 2006.

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