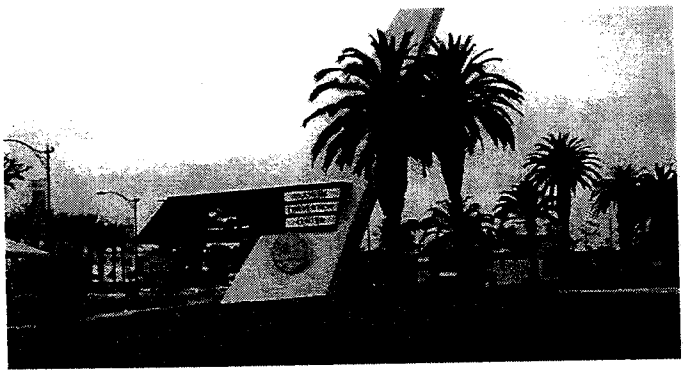
DAC 2005 Workshop

UML-SoC 2005

UML for SoC Design



Anaheim Convention Center
Anaheim, CA, USA
June 12, 2005

edited by:
Luciano Lavagno, Politecnico di Torino, Italy
Wolfgang Mueller, Paderborn University, Germany

Design and Synthesis of Reusable Platforms with Programmable Interconnects

Ananda Shankar Basu[1], Marcello Lajolo[1], and Mauro Prevostini[2]

[1] NEC Laboratories America, Princeton NJ 08540, USA
[2] ALaRI, University of Lugano, Lugano, Switzerland

Abstract. Platform based design requires to restrict the number of possible design choices in order to make it possible to come up with programmable solutions able to cope with the current complexity of System-On-Chip (SoC) designs. Nowadays there is a general consensus toward the fact that an effective Electronic System Level (ESL) design methodology must provide a specific support for platform specification, hardware/software partitioning and programmatic interfaces synthesis in order to allow designers to exploit the potentials of state-of-the-art technologies. In this work we present a methodology that leverages on UML for building new architectural platforms to be used in the system design process. We show how our methodology can allow to reuse pre-designed platforms by adding new architectural components and by customizing their interconnections.

1 Introduction

The proposed methodology leverages on the ACES codesign enviroment [1] which aims to assist the designer in the hardware/software partitioning and architecture selection phases. In [1] we showed how a functional specification of a system can be mapped onto a target architectural template that can be selected from a library of pre-designed templates. No support for the creation of new architectural templates was provided in that work.

In this paper we present a methodology that allows to fill this gap by providing the means to design new architectural templates. The main aim is to allow designers to specify new architectural platforms by combining pre-designed platforms and customize them by adding and removing selected architectural components like CPUs, buses and hardware units. Specific Intellectual Property (IP) blocks can also be specified at the functional level and synthesized in the final platform in order to provide programmable run-time configuration and interconnection mechanisms to the overall architecture.

2 The proposed design flow

Figure 1 shows the platform generation flow in our methodology. Starting from the top, on the left we have the functional specification of the components that

have to be inserted in the new architectural platform. On the right we have the specification of the basic architectural components that we want to have in the final platform. These two diagrams are then input into the codesign environment described in [1] and the following hardware/software partitioning and interface synthesis steps performed in this environment result in the diagram shown on the bottom of Figure 1 which corresponds to a new platform that can be added into the library of architectural templates provided by ACES.

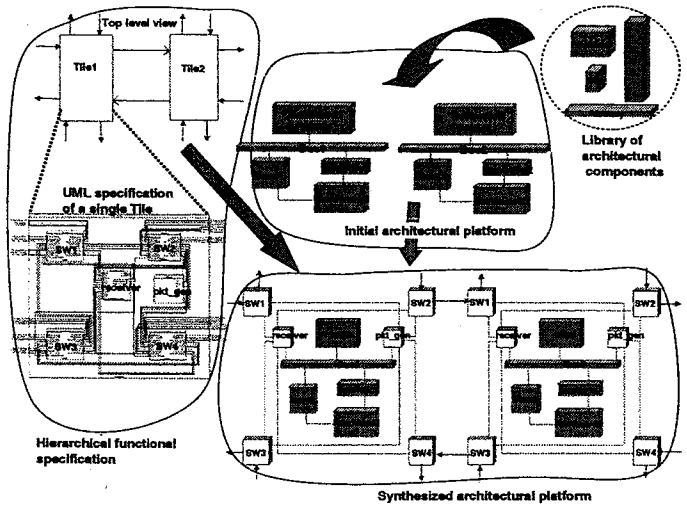


Fig. 1. Platform generation from UML specifications.

2.1 Platform configuration using object model diagrams

Object model diagrams are used for specifying the platform configuration in terms of both internal hardware/software features as well as in terms of topology and protocol used for interconnecting different sub-architectures. This is a hierarchical description containing a set of class instances modeling several aspects of the platform like the behavior of hardware IP cores, RTOS configuration routines, memory modules, glue logic and device drivers for connecting different hardware and software IPs in the platform.

The top-left picture of Figure 1 shows an example of a hierarchical object model diagram describing the communication wrappers of two tiles that are part of a 2-dimensional network on chip architecture. The two tiles are identical and consist of four switches used for implementing the communication with the four adjacent tiles in the network. A receiver unit processes incoming packets from all four switches and a sender unit delivers a packet to another tile in one of the four possible directions.

2.2 Platform specification using deployment diagrams

The deployment diagram can be used to specify a platform architecture in our proposed methodology. Normally the user has to select an architecture from a list of predefined platforms to be considered in the codesign phase. These predefined platforms are shown as an interconnection of hardware resources like CPUs, hardware elements and memories, connected by means of buses or dedicated point-to-point connections. Depending on the user's choice to map a design module either to hardware or software, the modules are deployed to the corresponding elements in the architectural platform. With respect to [1], we propose an additional stage in our design methodology where the platform can also be specified graphically by the user, making use of the UML deployment diagram. In addition to the predefined platforms, we also provide the basic resources like the CPUs, switches, hardware elements, buses, bus bridges, etc., from which the user can select, and connect them using communication paths to build his own platform. Consistency checks are necessary in order to ensure the semantic correctness of the usage of the architectural components as well as their interconnection protocol. The design components can then be deployed to the nodes in the architectural platform. Information from the deployment diagram would then be exported to the codesign environment for further steps to cosimulation.

The top-right picture of Figure 1 shows an example of deployment diagram for a dual processor architecture consisting of two identical single processor systems with a V850 CPU, a processor bus, a hardware unit connected to the bus through a dedicated bus interface and a memory module. The two single processor units are left completely unconnected in this diagram and their communication infrastructure will be specified in the following phase.

2.3 Platform generation from UML specifications

The new architectural platform is then automatically generated using hardware/software partitioning and interface synthesis support provided by ACES. This new platform is represented using another deployment diagram as it is shown on the bottom of Figure 1. In this diagram, modules represented in white are the class instances present in the initial object model diagram. These components have been synthesized by the codesign tool depending on the mapping specified during the hardware/software partitioning phase. In this case all modules have been mapped in hardware and hence they result in peripherals connected to the only bus in the tile. The components in gray are instead the architectural units that were already present in the original platform and among them there are the *mappable* units (CPUs and dedicated hardware components) that will be available for hardware/software partitioning experiments when this diagram will be selected for a new codesign step.

3 A simple bridge design example

In order to understand the basic steps that take place in our platform generation flow, we focus on the design of a basic unidirectional bridge used to

connect two distinct single-processor bus architectures. Figure 2 depicts the different phases of a platform configuration in which we start from an architectural template, shown in the top-right part of the picture, consisting of two identical pre-interconnected single processor templates. In this example, we want to build a dual processor system in which the two single processor templates are connected by a single unidirectional bridge from Bus_1 to Bus_2.

The functionality of the bridge is specified at the behavioral level in the object model diagram shown on the top-level part of the picture. The bridge has been decomposed in two different units: $BrgUp_{12}$ which handles the slave side of the bridge connected to Bus_1 and $BrgDown_{12}$ which handles the master side of the bridge connected to Bus_2. The behavior of both $BrgUp_{12}$ and $BrgDown_{12}$ can for example be described using UML state diagrams. In this functional specification stage, both units of the bridge contain specific address, data and RnW ports that represent a naming convention used to identify the bus ports that will have to be connected to a specific bus in the final configuration of the template. $BrgDown_{12}$ contains also request (Req_2) and acknowledge ports (Ack_2) for the communication with the bus arbiter, since it is a master on the bus to which it is connected.

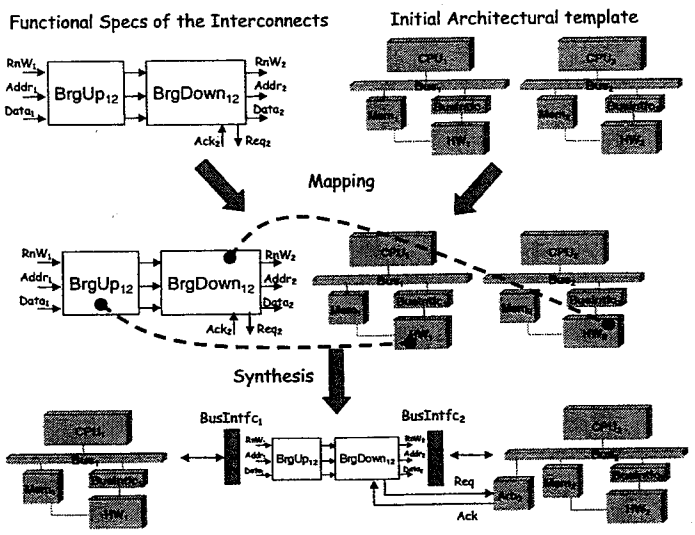


Fig. 2. An example of bridge design.

In the following phase, we perform the mapping of the functional specification of the interconnects onto the unconnected dual-processor template. This phase is depicted in the middle section of Figure 2 where $BrgUp_{12}$ has been mapped as

hardware connected to Bus_1, while $BrgDown_{12}$ has been mapped as hardware connected to Bus_2.

At this point, the final synthesis step can start. In this phase, the new architectural platform is automatically generated using hardware/software partitioning and interface synthesis support provided by ACES. The resulting outcome is the fully connected architectural template shown in the deployment diagram at the bottom of Figure 2. In this diagram, modules represented in white are the class instances present in the initial object model diagram. These components have been synthesized by the codesign tool depending on the mapping specified during the hardware/software partitioning phase. In this case, $BrgUp_{12}$ and $BrgDown_{12}$ have been implemented in hardware. Their port-to-port connections specified at the functional level remain port-to-port connections in the final architectural diagram. On the other hand, their connections to an abstract bus have been routed, through dedicated bus interfaces ($BusIntfc_1$ and $BusIntfc_2$), to the specific architectural bus at which they have been connected.

These interfaces are also instanciated automatically by the ACES interface synthesis tool for a set of supported bus architectures. For $BrgDown_{12}$, since it is a bus master, additional request and acknowledge lines need to be connected to the bus arbiter. In the current version of our platform design tool, we provide the possibility to customize the arbitration policy by choosing between a basic round robin (no priority) and a static priority algorithm. The bus arbiter is then automatically configured during the synthesis phase in order to accomodate the required number of masters and the selected priority. An alternative is to model the bus arbitration algorithm completely at the functional level and avoid to use the default one.

The architectural bus at which a functional component has to be connected is identified during the mapping phase by the fact that both CPUs and hardware elements in the initial architectural platform are connected to a unique bus. In this way, when we specify a hardware mapping of a new functional component, we know the bus at which we have to connect the bus interface ports that might appear in its description. In the current version of our tool, we do not allow to connect one functional component to multiple buses. In order to do this, it is necessary to partition the functionality of the component into smaller functional units, all connected to one single bus. This is what has been done in this simple bridge design example in which the bridge has been partitioned into two subcomponents connected to different buses.

4 Conclusions

The complexity of current embedded systems requires the adoption of programmable solutions. Platform based design concepts can allow to achieve this goal, but in order to be effective, they require efficient techniques for modeling new architectural specifications and maximize the opportunities for design reuse. In this paper we have presented a methodology that allows for the design

of reusable platforms with programmable interconnects by leveraging on UML specifications and an existing hardware/software codesign technology.

References

1. A. Basu, M. Lajolo, and M. Prevostini, "UML in an Electronic System Level Design Methodology," in *UML-SOC'04 (DAC Workshop), San Diego, California*, pp. 47–52, 2004.