COSMOS: A CoDesign Approach for Communicating Systems

Tarek Ben Ismail, Mohamed Abid, Ahmed Jerraya

TIMA/INPG Laboratory, System-Level Synthesis Group,
46 Av. Félix Viallet, 38031 Grenoble Cédex, FRANCE

Abstract

This paper presents COSMOS, a method for modeling and synthesis of complex communicating systems. COSMOS starts from a system-level specification based on an extended finite state machine model allowing for the specification of complex protocols. System-level synthesis is composed of three tasks: partitioning systems into inter-dependent sub-systems, inter-sub-system communication synthesis and architecture generation. The output is a flexible architecture model which includes both hardware and software components. The overall method will be illustrated through an example.

Keywords: System Design Model, Hardware/Software Codesign, Communication Synthesis

1. Introduction

The complexity of today's designs and in particular heterogeneous systems implies the need for global system approaches that handle both complex behavior and high-level communication. In this paper we are interested in the design of systems composed of a set of parallel processors communicating through well defined protocols. A processor may be a processor running software or an ASIC. The implementation of such a system may be one chip, a board, or a distributed multi-computer system.

1.1. Motivations

The fields of design, specification and synthesis of mixed hardware/software systems are emerging and becoming more and more popular [1, 2, 3, 4].

The main problem in current hardware/software design approaches is the integration of both resulting hardware designs and software designs. The use of a fixed hardware/software communication model restricts the applicability of the approach. The non availability of high-level communication models leads to start the design process with a too detailed description. The key issue that have to be solved is to find a way for abstracting complex communication protocols during the design process. We need to allow handle a communication model at different levels of abstraction in order to delay as much as possible the selection of the communication protocol that will be used. Of course the ultimate goal is to allow the re-use of existing communication models in order to allow hardware/software synthesis and mapping on existing architecture platforms.

The objective of this work is, the definition of a design methodology and tools aimed at the design and synthesis of high-level communication in the case of mixed complex hardware/software systems.

1.2. Related research

Several projects currently in progress (SpecSyn at Irvine [8], CODES at Siemens [10], SDW at Italtel [9], Thomas approach at CMU [12], Gupta and De Micheli approach at Stanford [2], Ptolemy at Berkeley [3, 4]) are trying to integrate both the hardware and the software in the same design process. These approaches differ by (1) the input specification (CDFG model or communicating processes), (2) the synthesis method (automatic, interactive, or manual), and (3) the target architecture considered (monoprocessor, multi-processor).

An input description may be given in a hardware description language (VHDL [11], HardwareC [2]) or a system-level description language (SDL [10], CSP [12], Statemate [10], SpecChart [8]). In the first case, a CDFG intermediate format is generally used by the subsequent synthesis tools. In the second case a system level intermediate format is needed. The application

0-8186-6315-4/94 $04.00

domain considered by each approach is tightly dependent on the description power of the input specification language taken.

It also appears that the synthesis process may be decomposed into two tasks: (1) the partitioning and (2) the communication (or interface) synthesis. The partitioning of a system's specification generates a set of partitions (hardware modules, communication blocks, software modules) which will be mapped on a target architecture. Most of the existing systems that starts from a CDFG, make use of an automatic partitioning. However, due to increased complexity, most systems based on a model of communicating processes make use of either interactive or manual partitioning.

The communication synthesis allows to synthesize interfaces between sub-systems, for example, the definition of the communication protocol or the I/O interface between components. Recent codesign approaches are based on one of three communication models: (1) fixed communication scheme [10] (e.g. point to point), (2) communication with a shared memory model [12], or (3) communication with a protocol [2, 4] (more or less complex).

The target architecture used to implement a design can be classified in one of the following cases: (1) Multi-chip architecture [8], (2) Architecture based on one processor and hardware components (ASICs, FPGAs) [2, 10, 12], and (3) Distributed and flexible architecture [3, 4]. For example, in this latter case, several configurations of processors may be used (monoprocessor, multiprocessor using a shared memory, message passing based architecture, etc.).

The main contribution of this work is the use of a system design model allowing for the re-use of existing communication models. This is obtained thanks to a clean separation between communication and computation during all the synthesis steps. Existing communication models may be abstracted as abstract channel units and used at different levels of abstraction.

The next section describes the hardware/software design models used in COSMOS both at the system-level and the architectural level. Section 3 gives an outline of the COSMOS synthesis environment. In section 4, an application example treated by system-level synthesis tools is presented.

2. Design models for codesign

COSMOS starts the design process with a system-level specification that may be given in an existing language such as SDL, StateCharts, ESTELLE or LOTOS. Our philosophy is to allow the designer to use one or more of these languages and to translate these descriptions into a common intermediate form, called SOLAR [5], capable of modeling the main concepts handled by system-level specification languages (concurrency, high-level communication, synchronization and exceptions). This intermediate form then acts as an input to the system-level synthesis tools. The main steps needed in order to transform a system-level specification into a mixed hardware/software one are system partitioning, communication synthesis and architecture generation. The output of architecture generation is an heterogeneous architecture represented by VHDL for the hardware elements and by C for the software. This description is finally mapped on a multiprocessor architecture. This mapping may be achieved using standard code generators to transform C into assembler code for software parts and synthesis tools in order to translate the VHDL into ASICs.

This section details the design models used by COSMOS. The intermediate form SOLAR will be introduced first, then the target architecture will be explained.

2.1. SOLAR: System-Level modeling for synthesis

SOLAR is a design representation for system-level concepts. SOLAR allows several levels of description, starting from the level of communicating systems which contains a hierarchical structure of processes communicating via channels right down to the register transfer level and basic FSM descriptions. In addition, the communication schemes can be described separately from the rest of the system. SOLAR's basic model is an extended FSM that allows the representation of hierarchy and parallelism. Such a model is shown in figure 1.

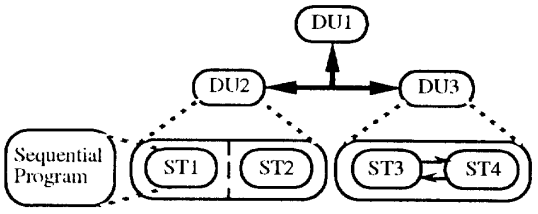


Figure 1: SOLAR: Basic Model.

In SOLAR, a system is structured in terms of communicating DesignUnits (DUs). A DesignUnit can either contain a set of other DesignUnits and communication operators known as ChannelUnits

(CUs), or a set of transition tables modeled by the StateTable (ST) operator. This operator is used to model process-level hierarchy. StateTables can be executed in parallel (as indicated by the dashed line between ST1 and ST2 in figure 1) or serially. They can contain other StateTables, simple leaf states, state transitions and exceptions.

Communication between sub-systems (or DesignUnits) is performed using SOLAR's ChannelUnit. It is possible to model most system-level communication properties such as message passing, shared resources and other more complex protocols. The ChannelUnit combines the principles of monitors and message passing. The model is known as the Remote Procedure Call or RPC. The RPC model is a mechanism that allows processes to communicate across message-carrying networks. The networking services are transparent to the user and communication is invoked using the semantics of a standard procedure call.

Figure 2 shows the basic configuration of the ChannelUnit. Not only does this model enable the user to describe a wide range of communication schemes, it also separates the communication from the rest of the design, thereby allowing the re-use of existing communication schemes.

The ChannelUnit allows communication between any number of DesignUnits (a DesignUnit may be viewed as a system-level process). Access to the ChannelUnit is controlled by a fixed set of procedures known as methods (or services). These methods correspond to the *visible* part of the channel. In order to communicate, a DesignUnit needs to access at least one method. It achieves this through the use of a special procedure call statement known as a CUCall. In other words, the channel acts as a co-processor for the processes using it. The rest of the ChannelUnit is completely transparent to the user and consists of a set of ports linking the methods' parameters to the channel's controller. The controller guards the current state of the channel as well as conflict-resolution functions. The methods interact with the controller which in turn modifies the channel's global state and synchronizes the channel.

During the synthesis process, COSMOS uses an external library of CUs. A CU corresponds to either a standard protocol or a customized protocol described by the user. During partitioning and communication synthesis an abstract model of the channel is used. At the architecture generation step, an implementation of the channel is needed. This implementation may be the result of an early synthesis step using COSMOS or another design method. It may also correspond to an existing architecture.

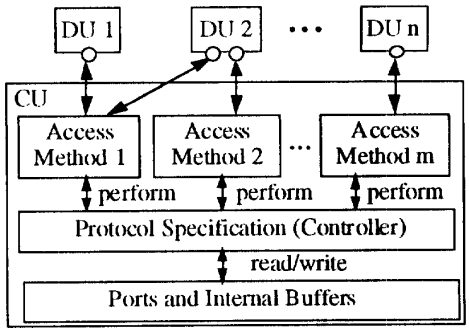


Figure 2: SOLAR's communication model: the ChannelUnit.

2.2. Hardware/Software architectural model

In this approach, a predefined implementation architecture is considered. This architecture serves as a platform onto which a mixed hardware/software system is mapped. The underlying architectural model is general enough to represent a large class of existing hardware/software platforms. As described in figure 3, the architecture is composed of three kinds of components: (1) Software components, (2) Hardware components, and (3) Communication components.

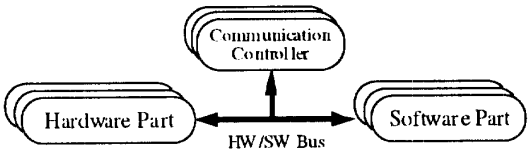


Figure 3: Flexible Architecture Platform.

Communication modules come from a channel unit library, they correspond to existing communication models that may be as simple as a handshake or as complex as a layered network. For example, a communication controller may correspond to an existing interface circuit, an ASIC or some micro-code executing on a dedicated microprocessor. The communication scheme may be one of the three different types: HW↔HW, SW↔SW, or HW↔SW.

The proposed model allows different implementations of mixed hardware/software systems. A typical architecture will be composed of hardware modules, software modules, and communication modules linked with buses. Figure 4 shows an example of an architecture supported by COSMOS.

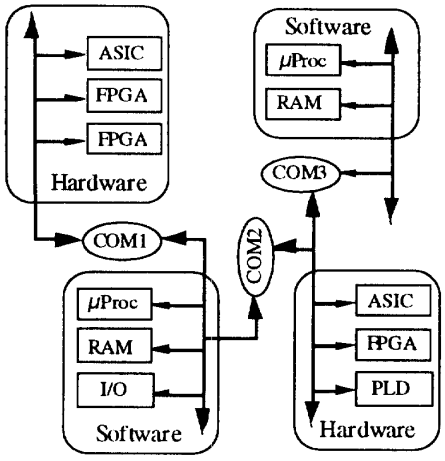


Figure 4: Example of an Architecture Supported by COSMOS.

3. COSMOS: A System-level synthesis framework

This section concentrates on the design flow within the COSMOS environment. Currently, COSMOS starts from the system-level specification language SDL, and produces a heterogeneous architecture including hardware descriptions in VHDL and software descriptions in C. As shown in Figure 5, the COSMOS environment addresses four issues: system-level specification, system-level partitioning, communication synthesis (including channel binding and channel mapping), and architecture generation (including virtual prototyping and architecture mapping). In this chart, the boxes are to be interpreted as activities, whereas the small circles correspond to intermediate models and libraries. These are expanded and represented as system level graphs. These models and libraries are explained in the rest of this section.

In the present version of COSMOS, the design flow starts from an SDL description. However, all the subsequent synthesis steps make use of SOLAR. The translation of SDL communication concepts leads to a reorganization of the description. The SDL to SOLAR translation is performed automatically.

Starting from a SOLAR representation, a system is partitioned by a system-level partitioning tool box called PARTIF [6].

PARTIF starts with a set of communicating processes organized in a hierarchical manner and described in SOLAR. Each process represents an extended FSM. Another input to PARTIF is a library of SOLAR communication models.

The result of system-level partitioning is a set of communicating and heterogeneous processes organized in a graph where the nodes may be either design units or channel units and the edges of the graph may be signals or channel accesses.

In the present version of COSMOS, the partitioning is performed interactively. PARTIF provides five system-level transformation primitives. The two first primitives *MOVE* and *MERGE* allow the reordering of processes hierarchy and merging processes together to form a single process. The two second primitives are *SPLIT* and *CUT*. These allow splitting up one design unit to form inter-dependent design units for distribution purposes. The fifth primitive (called *CLUSTER*) allows to cluster several processes in one design unit. The sequencing of these primitives is decided by the user.

The partitioning step also determines which technology will be used for the implementation of each design unit. For example, a design unit may be implemented in pure hardware, in software running on an operating system or in micro-code adapted for a standard microprocessor. The choice is based on criteria such as execution time, rate of use, reprogrammability, re-use of existing components and technology limitations. This choice is performed manually.

The objective of communication synthesis [7] is to transform a system containing a set of processes communicating via high-level primitives through channels into a set of interconnected processes communicating via signals and having the control of this communication distributed among the processes. Communication synthesis starts with two inputs: A heterogeneous system graph and a library of SOLAR communication models. As stated above, this activity may be decomposed into two tasks: channel binding and channel mapping [7].

The channel binding algorithm is assumed to choose the appropriate set of channel units from the library of communication models to carry out the desired communication. This is similar to the binding/allocation of functional units in classic high-level synthesis tools. The communication between the sub-systems may be executed by one of the schemes (synchronous, asynchronous, serial, parallel, etc....) described in the library. The choice of a given channel unit will not only depend on the communication to be executed but also on the performances required and the implementation technology of the communicating design units. In the present version of COSMOS, this task is done manually.

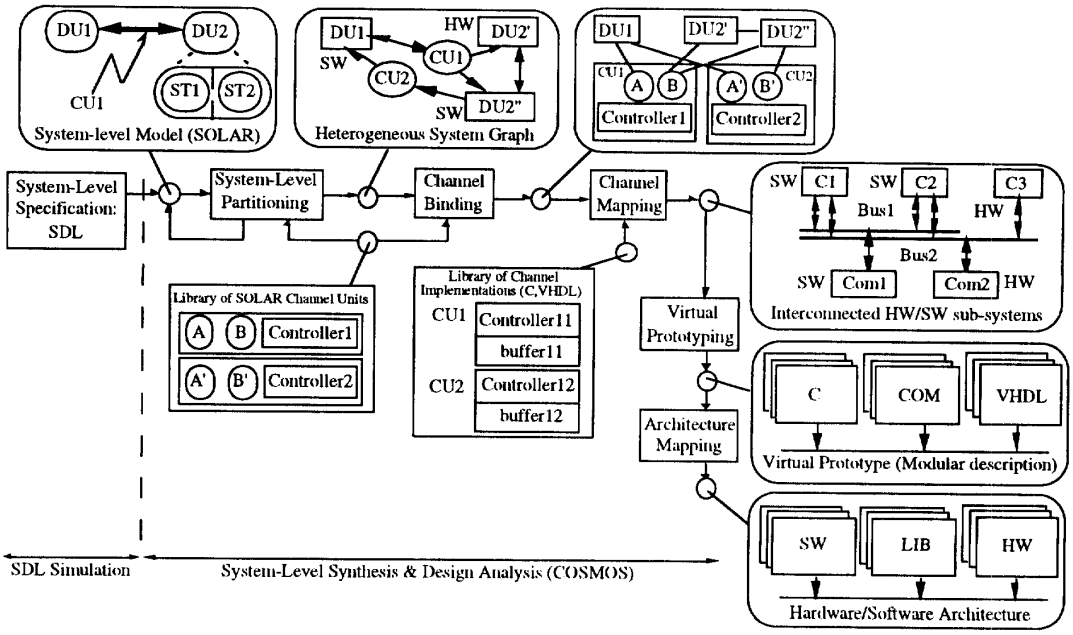


Figure 5: System-Level Synthesis: Design Flow.

The channel mapping task replaces all channel units by distributing all of the information contained therein among the design units and specific communication controllers. These controllers are selected from a library of channel implementations. A channel may be a software component described in C code or a hardware component described in VHDL. All processes that execute a channel unit call to a particular method, will have this call replaced by a call to a local procedure implementing that method. The variables and signals used by the methods become ports in the modified design unit. In other words, the channel accesses are expanded into bundles of signals (ports) according to the methods used by the design units. The implementation of this local procedure call will depend on the implementation of the corresponding design unit. If the design unit is entirely software executing on a given operating system, method calls are expanded into system calls, making use of existing communication mechanisms within the system. If the design unit is to be executed on a standard micro-processor, the method becomes an access to a bus routine written in assembler. These two options are more software-oriented and require the user to partition his system into software and hardware elements before executing the communication synthesis. The design unit can also be executed as embedded software on a hardware datapath controlled by a micro-coded controller. In this case, the method call will become a call to a standard micro-code routine. Finally, the design unit may be implemented entirely in hardware.

The architecture generation step starts with a set of interconnected hardware/software sub-systems (output of communication synthesis). As indicated above, this activity is decomposed into two tasks: virtual prototyping and architecture mapping.

During the first task, a translation of SOLAR into executable code (VHDL and/or C) is performed. Each sub-system is translated independently. The output of virtual prototyping is a heterogeneous architecture represented by VHDL for the hardware elements and C for the software elements (see figure 5).

During the architecture mapping task, this mixed description is mapped on a flexible architecture including hardware sub-systems, software sub-systems, and communication sub-systems. This may be achieved using standard code generators to transform C into assembler code for software parts and synthesis tools in order to translate the VHDL into ASICs. Channel units correspond to library components. The separation between communication and computation (hardware and software) allows the re-use of existing communication models. In fact a channel may be the result of an early synthesis step using COSMOS or another design method. It may also correspond to an existing architecture (see section 4).

4. An Example

To illustrate the hardware/software codesign approach, an example of a Real-Time Acquisition and Storage system (RTAS) will be presented (see figure 6). In this example, the RTAS system performs the following tasks:

- Receipt of an analog signal from 8-bit multiplexed wires,
- Conversion of each analog signal into a digital signal,
- Storage of the result into disk.

The RTAS system is mainly used for the acquisition of biological signals. The signals frequency may reach 4 Khz. In order to satisfy Shannon theorem, a minimal sampling frequency must be 64 Khz. Consequently, the time separating two samples should not exceed 12.5 ms.

Thus, the time of reading, converting and storing signals has to be less than 12,5 ms (this time corresponds to the selection of the sampling frequency).

As shown in figure 6, the RTAS system is composed of two sub-systems communicating via a communication channel. The first is the Acquisition sub-system which receives analog signals, selects and converts the appropriate signal and sends it over the communication channel. The second is the Storage sub-system which receives and stores the information.

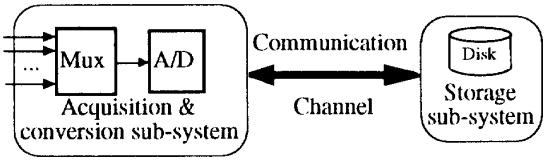


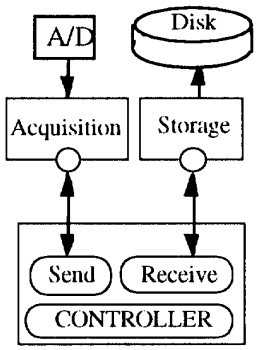
Figure 6: Real-Time Acquisition and Storage System.

4.1. System modeling

For this example we will start the design just after process-level partitioning. We will then start with a communicating systems graph.

The system is composed of two communicating sub-systems. A brief outline of the SOLAR description of this system is presented in figure 7(b). The first DesignUnit, RTAS is a structural view of the system. The two constituent DesignUnits, Acquisition and Storage, are instantiated and a netlist of the connections between these two DesignUnits is given. One of the nets connecting the two DesignUnits is a ChannelUnit.

This description is independent of the communication protocol. At this level, the communication channel is a ChannelUnit that allows communication between the Acquisition sub-system and the Storage sub-system known as Acquisition DesignUnit (ADU) and Storage DesignUnit (SDU) (figure 7(a)). The access to the channel is accomplished by "methods" (services). In order to communicate, the ADU and the SDU should invoke the appropriate method. In this example, the channel contains two methods accessed by the DesignUnits. The Send method is called by the ADU process to send digital signals and the Receive method is called to read the information.




```
(SOLAR RTAS-system
 (DesignUnit RTAS
  (Interface...)
  (Contents
   (Instance Acquisition...)
   (Instance Storage...)
   (Net (Joined...) ..)
  )
  (DesignUnit Acquisition
  /* Receives analog signals */
  ...
  )
  (DesignUnit Storage
  /* Stores digital signals */
  ...
  )
```

(b)

Figure 7: (a) Communicating Sub-systems,
(b) Overview of SOLAR structure.

4.2. Hardware/Software partitioning

In this example, the only partitioning needed is hardware/software partitioning. The SDU is mapped to software, the ADU and the communication controller are mapped to hardware in order to meet the real-time constraint(12.5 ms).

Only the Acquisition DesignUnit, the Storage DesingUnit and the communication channel will be realized. We assume that a multiplexer, an A/D converter and a Storage disk are chosen among already existing devices.

4.3. Communication synthesis

The objective of communication synthesis is to transform the RTAS system containing a communicating ADU and SDU into interconnected ADU and SDU communicating via a channel unit from the library. The two steps of communication synthesis will be detailed.

In this case we started the channel binding step with a library including two protocols* having the same interface (communication primitives). The first is a simple synchronous protocol. This protocol assumes that only one converted signal at a time is loaded in the system memory. In this case, only the system bus is used for data transfer. However, transfer, control and storage programs bring the execution time to more than 12,5 ms. Thus, this scheme does not meet real-time requirements. The second protocol is asynchronous. The channel is used for governing access to a dual port RAM (two memories). The ADU and the SDU may work asynchronously. For example, while the ADU is writing to the first memory, the SDU may be reading from the second memory.

An analysis indicated that only the second solution meets real-time requirements. Then, the channel is bound to the second CU.

During the channel mapping step, the information contained in the ChannelUnit is completely distributed among the communicating processes. The resulting system is a set of interconnected processes each of which can be synthesized independently using the appropriate tools (see figure 8). This model is represented as a set of interconnected SOLAR DesignUnits. In this case we obtain a hardware module, a software module and a communication module from the library. One can note that this module is implemented in hardware.

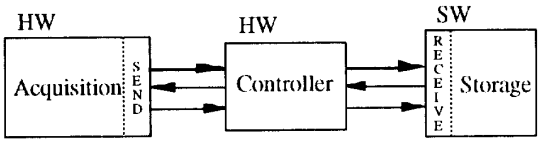


Figure 8: Communication Synthesis Output for the RTAS.

4.4. Architecture generation

The first task was to convert the SOLAR description in order to generate C code for the Storage module and VHDL code for the Acquisition and controller components. This model (also called virtual prototype) can be validated using classic VHDL-C simulation environments.

The final step of codesign is to produce a prototype of the system. The prototyping method is used for accurate performance evaluation using realistic implementation time. Figure 9 shows a global view of the RTAS system prototype.

In this case we used an Intel 286-based PC AT as a development platform. The hardware is implemented on an FPGA based extension board. The communication module is composed of an interrupt-controller, a DMA, and a dual port memory. The protocol is executed as follows: Converted signals are transferred into the source memory using the Send method. When this memory is full, the SDU is interrupted through an interrupt-controller to transfer acquired information into the system memory using a DMA. The acquisition is carried on with the second memory. A communication controller is used to control transfer between the ADU and the SDU and to resolve memory access conflicts.

An analysis of the prototype system indicates that this solution correctly implements the system functionality while meeting real-time requirements. This system has been experimented with real-biological signals. The result is satisfactory.

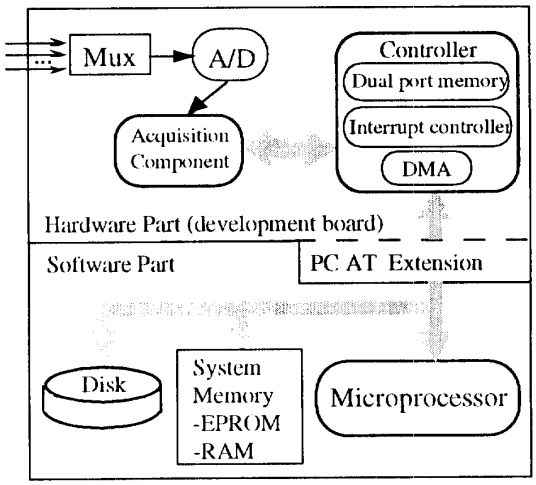


Figure 9: The RTAS system prototype.

5. Conclusions

In this paper a full methodology for hardware/software codesign called COSMOS has been presented. COSMOS is based on a system level intermediate format called SOLAR that allows system specification at different levels. This model allows for the re-use of existing communication models thanks to a clean separation between communication and computation during all the synthesis steps. Existing communication models may be abstracted as abstract channel units and used at different levels of abstraction.

* SOLAR allows many different protocols.

References

[1] G. Boriello, K. Buchenrieder, R. Camposano, E. Lee, R. Waxman, W. Wolf, "Hardware/Software Codesign", IEEE Design and Test of Computers , pp. 83-91, March 1993.

[2] R.Gupta, G. DeMicheli, "Hardware-Software Cosynthesis for Digital Systems", IEEE Design and Test of Computers, pp. 29-41, September 1993.

[3] M.Chiodo, P.Giusto, A.Jurecska, L.Lavagno, H.Hsieh, A.Sangiovanni-Vincentelli, "A Formal Specification Model for Hardware/Software Codesign", Handouts of Int'l Wshp on Hardware-Software Co-design, Cambridge, Massachusetts, October 1993.

[4] A. Kalavade, E. A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications", IEEE Design and Test of Computers, pp. 16-28, September 1993.

[5] A. A. Jerraya, K. O'Brien, "SOLAR: An Intermediate Format for System-Level Modeling and Synthesis", in "Computer Aided Software/Hardware Engineering", J.Rozenblit, K.Buchenrieder (eds), IEEE Press 1994.

[6] T. Ben Ismail, K. O'Brien, A. A. Jerraya, "Interactive System-Level Partitioning with PARTIF", Proc. EDAC'94, Paris, France, February 1994.

[7] K. O'Brien, T. Ben Ismail, A. A. Jerraya, "A Flexible Communication Modelling Paradigm For System-Level Synthesis", Handouts of Int'l Wshp on Hardware-Software Co-Design, Cambridge, Massachusetts, October 1993.

[8] D. Gajski, F. Vahid, S. Narayan, "A Design Methodology for System Specification Refinement", Proc. EDAC'94, Paris, France, February 1994.

[9] S.Antoniazzi, M.Mastretti, "An Interactive Environment For Hardware/Software System Design at the Specification Level", Microprocessing & Microprogramming, Vol. 30, pp. 545-554, 1990.

[10] K. Buchenrieder, A. Sedlmeier, C. Veith, "HW/SW Co-Design With PRAMs Using CODES", Proc. CHDL '93, Ottawa, Canada, April 1993.

[11] N.L.Rethman, P.A.Wilsey, "RAPID: A Tool for Hardware/Software Tradeoff Analysis", Proc. CHDL '93, Ottawa, Canada, April 1993.

[12] D.E.Thomas, J.K.Adams, H.Schmitt, "A Model and Methodology for Hardware-Software Codesign", IEEE Design and Test of Computers, pp. 6-15, September 1993.