HW/SW Codesign for Embedded Telecom Systems

S. Antoniazzi (a,b), A. Balboni (b), W. Fornaciari (a), D. Sciuto (c)

(a) CEFRIEL, via Emanueli 15, 20126 Milano (MI), Italy, E-mail: fornacia@mailer.cefriel.it
(b) ITALTEL-SIT, Central Research Labs, CLTE, 20019 Castelletto di Settimo m.se (MI), Italy
(c) Politecnico di Milano, Dip. Elettronica e Inform., P.zza L. Da Vinci 32, Milano, Italy, E-mail: sciuto@elet.polimi.it

Abstract

The aim of this paper is to define an approach, tailored for control-oriented applications, to manage system cospecification, high-level partitioning, hw/sw tradeoffs and cosynthesis. Our research effort focuses on fulfilling the goal of linking high-level specifications to efficient and cost-effective hw/sw implementations, by investigating techniques such as synchronous cospecification styles, direct machine code generation as well as exploiting the capability of commercial VHDL synthesis tools.

1. Introduction

Heterogeneous hardware/software architectures, for many application fields requiring an ASIC approach, may provide a more effective design solution for some target performance/cost figures with respect to fully dedicated hardware implementations. Therefore, new design automation methodologies should be placed on top of current ASIC design flows in order to integrate dedicated logic obtained by register-transfer level synthesis, with CPU core cells and the related software (firmware).

Although hardware/software codesign goals and strategies will not probably converge to a single common interpretation, due to the wide spectrum of application fields and design requirements, the potential value-added provided by the automation of codesign tasks has been shown by a number of recent research works ([1], [2], [3], [4], [5], [6]).

The aim of this paper is to introduce a novel methodology to manage the codesign process for a specific application field, namely control-dominated ASICs such as those embedded into telecom digital switching subsystems. The development of such methodology is currently in progress within a research project called TOSCA (TOols for System Codesign Automation), in which one of the main activities is the definition of a support environment by integrating commercial EDA software with new experimental tools.

After a general overview of the proposed codesign methodology, the paper will discuss the main phases allowing hw/sw tradeoff exploration and cosynthesis starting from high-level cospecification. The prototype software environment, supporting the envisaged codesign flow, will also be addressed.

2. The codesign flow

The aim of the proposed methodology is to allow the designer to experiment alternative system designs in order to balance hardware costs and software performances. Such a goal can be achieved through a design flow, able to capture initial specification avoiding as much as possible any implementation bias but, at the same time, suitable to make possible fast architectural exploration and direct integration within existing commercial synthesis environments.

In order to give effectiveness and validate the proposed approach, a prototype software environment, supporting the codesign flow depicted in fig.1, is currently under development. The target is to cover the following issues:

- acquisition of behavioral specifications, suited to the application field of interest, while maintaining full independence of any particular hw/sw implementation;
- analysis/validation at specification-level;
- tool-directed restructuring and hw/sw binding of specifications;
- concurrent synthesis of sw-bound, hw-bound parts and related interfaces;
- cost/performance analysis of the alternative hw/sw architectures;
- integration with commercial RTL synthesis and optimization tools, as well as software/firmware development tools.

The codesign process starts from a system model captured via a mixed graphical/textual formalism, based on concurrent and hierarchical finite-state machines (FSMs). After a preliminary analysis/validation activity, an internal system representation (TOSCA DB) tailored to support high-level architectural exploration, is obtained. The main activities involving the design database, are the *restructuring* of the initial system modularization to produce a new set of system partitions and their association (*binding*) either with software or with dedicated hardware; the HW-SW interface generation; the cosynthesis stage. A set of strategies and basic transformations can be iterated onto the system representation, until the design constraints are satisfied. The user, as well as some heuristic strategies, can organize these actions along predefined schedules called *recipes*. Both software and hardware synthesis exploit technology-dependent parameters, enabling a realistic cost/performance estimation of each proposed architectural solution.

According to the chosen level of accuracy, the information used for hardware characterization range between purely estimation and data obtained through an actual synthesis process. Due to the impossibility of carefully controlling time

1063-6404/94 $4.00 © 1994 IEEE

delay, code size and low level interfacing schemas, the software parts need to be considered at a lower level with respect to C-language based solutions, in particular if processor retargeting capabilities are envisaged. Our solution is to consider the software description at the level of a virtual assembly whose structure can be mapped onto different CPUs core with fully predictable translation rules and, consequently, reliable performance estimation.

Finally, back-end tools produce a design representation of the selected hw/sw solution (assembly code, VHDL) acceptable for the target commercial implementation environment.

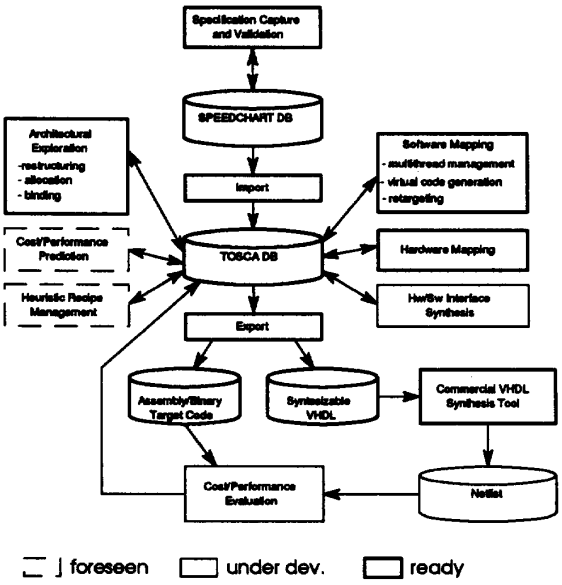

Figure 1: The codesign activities supported by the TOSCA environment.

3. Specification and system-level exploration

As mentioned in the introduction, the TOSCA approach focuses on control-dominated designs. Therefore, a particular specification style has been adopted, based on a preliminary analysis of the selected target field. Such analysis stage has shown that the following aspects should be carefully taken into account:

a) a concurrent model is required, based on multiple interacting processes;
b) each process may be properly modeled via a synchronous state/transition description;
c) high-level concepts/constructs such as timeouts, behavioral hierarchies and symbolic data may be very valuable in order to cope with specification complexity and implementation independence;
d) logic functions and bit vector data types dominate descriptions while arithmetic data processing plays a secondary role;
e) multiple clocks may drive processes.

It should be pointed out that the synchronous paradigm is not intended to force any hardware bias but to model the intrinsic nature of the application itself. For a discussion of synchronous approaches to reactive system modeling see ([7], [8]).

A commercial environment (SPeeDCHART by Speed Electronic) has been adopted for both specification and validation purposes. The formalism provided by SPeeDCHART belongs to the Statecharts family ([9], [10], [11]), coupling graphics with textual descriptions written in a VHDL-like script language. Such environment also allows system validation by simulation, including visual animation capabilities.

The system is composed of multiple processes communicating via shared signals. Special states labeled *entry* always allow identification of the initial state of a diagram at any hierarchical level. Actions and priorities (represented via *circled numbers* on edges) can be associated with transition edges. State nodes can also have behavioral scripts in terms of entry/exit actions: entry actions are triggered when the related state is reached and are semantically equivalent to actions associated with all the incoming transitions; exit actions are triggered when the related state is exited and are semantically equivalent to actions associated with all the outcoming transitions. There are also some useful additional features such as timeouts (captured by the *settimer* statement) and the symbolic representation of communication messages.

Since the subsequent codesign stages have been implemented on top of a different software environment (the O2 object-oriented DBMS) a preliminary step (Import) has to be performed in order to translate SPeeDCHART design data into the TOSCA internal model. Such core representation preserves the concurrent/hierarchical structure of the original specification. The main difference is in the management of textual scripts: the Import procedure starts from ASCII files produced by SPeeDCHART and carries out a parsing process building symbol tables for declarations and data flow graphs for conditions and actions. Enumerative types are also encoded in this step.

After the Import procedure has been completed, architectural tradeoffs may be carried out by iterated manipulation of the core model. This stage manages two classes of objects (*processes*, namely the modules in the specification, and *architectural units*, i.e. the modules in the target implementation) and involves three tightly related activities:

- *restructuring*: transformations local to a single process modifying the cardinality of the whole process set (acts at process-level only);
- *allocation*: clustering of processes by assigning an architectural unit identifier to each process (operates on both processes and architectural units);
- *hw/sw binding:* marking of each architectural unit either with a software or a hardware constraint (operates on architectural units only).

In the following, a software-bound architectural unit will be indicated with the term *thread* while a hardware-bound unit will be called *coprocessor*.

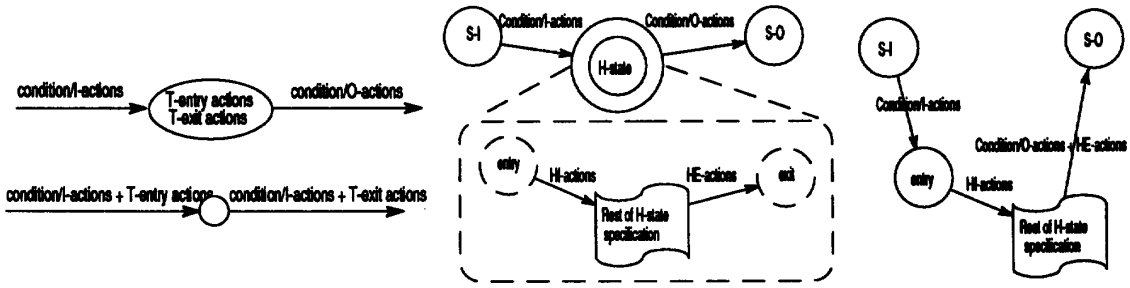

Figure 2: State actions unfolding (left); the simplest case of hierarchy removing (right).

Concerning restructuring, a set of basic transformation algorithms has been made available:

a) unfolding entry/exit actions;
b) unfolding FSM enabling conditions;
c) flatten hierarchies;
d) mapping timers onto counters;
e) collapsing a set of processes.

As mentioned above, entry/exit actions may be possibly associated with states, thus allowing a more concise representation. Users may preserve such a model structure or apply the algorithm a), obtaining the representation depicted in fig.2 (left), by migrating entry and exit actions respectively toward incoming and outcoming transitions.

SPeeDCHART also allows to specify a general condition associated with a whole FSM: the FSM is enabled if and only if such a condition is true. The transformation b) moves the condition from the FSM level downto each transition: such condition is merged with the original transition predicate by using an AND operator.

In order to make easier the hardware/software mapping step, hierarchical descriptions can be expanded by the application of algorithm c). Referring to the case reported in fig.2 (right), all the incoming edges will be linked to the *entry* state together with their condition/actions pairs.

Concerning the outcoming arcs, if no *exit* condition is present, is sufficient to create an edge connecting the *exit* state with the target state at the upper level. In the general case of edges having their own target state and condition/actions pairs, for each state belonging to the child it is necessary to introduce an edge connected to the upper level state. Actions can still remain joined to the original condition or added to the target state action list. Zero-flagged counter variables with decrement control, introduced by transformation d), are employed to act as special objects modelling timeout conditions.

Finally, the task of collapsing multiple machines on a common final target machine (e) is implemented by a technique based on symbolic execution. A similar approach, but assuming an asynchronous semantics, is discussed in [6]. This capability can be used to collapse machines belonging to a common partition (architectural unit) to obtain a coarse grain description useful for the subsequent hw/sw binding and cosynthesis. The algorithm for building a set of architectural units it is based upon a basic procedure, MERGE(proc., proc.), able to collapse two FSMs at once, and performs a pairwise merging of the processes. First actions of the MERGE(MA, MB) procedure are the creation of the interface specification for the merged machine and the removal of the unnecessary inter-module links by transforming them into internal variables. MA and MB, beginning from their first state, are then symbolically executed in parallel. States in the target machine are obtained from each explored state pair (MA,MB). Their DFGs (if any entry/exit action is present) correspond to the merging of those from MA and MB states. The transitions of the merged-machine are obtained by considering all possible combinations of those present in MA and MB. For each resulting transition the following rules are applied:

- actions are merged;
- conditions are combined by an AND operator;
- priorities are managed by computing a function of the original priorities.

Users may define their own custom flows (*recipes*) based upon the above kit of basic transformation algorithms. Recipes may also contain special report actions, describing characteristics and statistics about intermediate and final results. The output of this process is a set of monolithic architectural units with a binding establishing either a hardware or software implementation. Each architectural unit is then passed as input to the subsequent cosynthesis stages, as described in the next section.

Additional transformation algorithms are under development, such as moving arithmetic operators across clock steps and splitting a process into multiple subprocesses.

4. Hardware/software mapping

The target hw/sw architecture is realized on a single chip. The most general case includes one off-the-shelf CPU core cell and a collection of synthesized *coprocessors*. After restructuring, allocation and binding, each resulting hardware-bound architectural unit is mapped onto its own coprocessor. In this discussion the term coprocessor includes also arithmetic/logic operations and the possible private storage capability, while high-level synthesis tools typically separate controllers from data-paths.

If a coprocessor requires interfacing to/from software-bound elements, then it is connected to the CPU shared data bus (and related address/control lines) and to the interrupt lines. All hardware-to-hardware interfaces are managed by customized local interconnection lines. The RAM memory required for

program/data storage shares with coprocessors the main data bus but can be accessed only by the CPU core.

Concerning the *hardware mapping* strategy adopted in TOSCA, it should be pointed out that control-oriented specifications cannot be easily managed by classical high-level synthesis approaches involving operators scheduling.

In fact, circuit speed estimation is very difficult when dealing with descriptions dominated by logic functions, where arithmetic operations are typically restricted to a few additions, subtractions and comparisons (if any of them is present at all). During the next stage involving VHDL translation into a generic netlist, technology mapping and logic implementation, any direct relationship between functional specification and synthesized implementation is lost. Estimating area is also a very hard task. As a consequence, scheduling operators according to estimated propagation delays cannot be considered a realistic approach. In the TOSCA module devoted to hardware mapping, each hardware-bound architectural unit (possibly obtained from multiple merged processes) is implemented by generating a finite state machine VHDL description. Since the starting point is a synchronous model, no additional scheduling step is needed. The VHDL code generator translates the internal representation of each FSM into a VHDL template complying to the guidelines for synthesizability enforced by commercial tools such as MGC Autologic and Synopsys. The data flow graphs modeling conditions and actions are translated into VHDL statements included in the related template. The algorithm adopted is able to produce a very readable description by building expressions whenever possible instead of basic assignments for each DFG node. Parameters such as the logic types to be used (e.g. BIT-VECTOR vs IEEE standard packages) or modeling style (structural vs behavioral) can be customized by the user.

In particular cases, such as for instance counters, predefined library components may be preferred to RTL synthesis in order to guarantee an efficient implementation.

The application field requirements have led to discard a C-language based approach for the automated implementation of *software-bound* elements. In fact an high-level language such as C does not allows an accurate control of time delays nor the code size as well as the low level characterization of the I/O interface. Therefore, a lower level of abstraction has been introduced with the concept of *Virtual Instruction Set* (VIS), comparable to the one provided by a RISC assembly language while maintaining independence from the target CPU core. VIS is defined in terms of a register-oriented machine supporting unsigned/signed integer data types (8, 16 and 32 bits) as well as all typical arithmetic/logic operations.

At present, a code generation prototype tool has been developed supporting a single software-bound FSM (anyway, multiple machines may be collapsed before software synthesis). Such a tool provides register usage optimization and automated packing of single-bit variables. Data transfer from software to hardware and viceversa is modeled via memory-mapped coprocessor registers.

VIS code is finally translated into the target assembly language (or binary image) by first generating an ASCII representation and invoking a rule-based program implemented by means of a PERL (a text processing language for UNIX platforms) script. A retargeting tool has been implemented for a Motorola 68000 core. The approach can be easily extended to most popular CPU cores.

Work is in progress in order to manage multiple concurrent software threads with a minimum overhead, adopting a static scheduling strategy.

5. Conclusions and future developments

An application-oriented hw/sw codesign methodology has been presented. A prototype toolset covering cospecification, hw/sw exploration and cosynthesis has been also developed. Work is in progress aiming at introducing more sophisticated algorithms and features on top of such a basic framework. Currently most of the implementation effort is devoted to the transformation algorithms and to the cost/performance evaluation, while restructuring and hw/sw binding can be performed only manually (the choice concerning the strategy to be adopted at each iteration of the exploration cycle is left to the user).

Global cosimulation is one of the issues that will be addressed in the future work. Both the specification level and the implementation levels will be developed.

References

[1] Gupta R.K., and De Micheli G., Hardware-Software Cosynthesis for Digital Systems, IEEE Design&Test, September 1993.

[2] Barros E., Rosenstiel W., Xiong X., Hardware/Software Partitioning with UNITY, Proc. of 2nd Workshop on HW/SW Co-Design, Cambridge, Massachussetts, October, 1993.

[3] Benner T, Ernst R., Henkel J., Hardware-Software Cosynthesis for Microcontrollers, IEEE Design&Test, Vol.10, No.4, December 1993.

[4] M.Chiodo, P.Giusto, A.Jurecska, L.Lavagno, H.Hsieh, A.Sangiovanni-Vincentelli, Synthesis of Mixed Software-Hardware Implementations from CFSM Specifications, Proc. of 2nd Workshop on HW/SW Co-Design, Cambridge, Massachussetts, October 1993.

[5] Steinhausen U., Camposano R., Gunther H., Ploger P., Theibinger M., Veit H., Vierhaus H.T., Westerholz U., Wilberg J., System-Synthesis using Hardware/Software Codesign, Proc. of 2nd Workshop on HW/SW Co-Design, Cambridge, Massachussetts, October, 1993.

[6] Wolf W., Takach A., Huang C., Manno R., The Princeton University Behavioral Synthesis System, 29th DAC, 1992.

[7] Benveniste A. and Berry G., The Synchronous Approach to Reactive and Real-Time Systems, in Proc. of the IEEE, Vo.79, No.9, September 1991.

[8] Clarke E., Long D. and McMillan K., A Language for Compositional Specification and Verification of Finite State Hardware Controllers, in Proc. of the IEEE, Vo.79, No.9, September 1991.

[9] Harel D. et al., STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Trans. on Software Engineering, Vol.16, No.4, April 1990.

[10] Narayan S., Vahid F., Gajski D.D., System Specification with the SpecCharts Language, IEEE Design & Test, Vol.9, No.4, December 1992.

[11] Vahid F., Gajski D., Specification Partitioning for System Design, Proc. of the 29th Design Automation Conf., 1992.