A Co-Design Methodology Based on Formal Specification and High-level Estimation

C. Carreras†, J. C. López†, M. L. López†, C. Delgado-Kloos†, N. Martínez*, L. Sánchez†

† Dep. Ing. Electrónica
ETSIT, Univ. Polit. Madrid
28040 Madrid, Spain

‡ Dep. Ing. Sistemas Telemáticos
ETSIT, Univ. Polit. Madrid
28040 Madrid, Spain

* Dep. Ingeniería
Univ. Carlos III Madrid
28911 Leganés, Spain

Abstract

This paper presents a methodology for hardware-software co-design. It is based on the formal description technique LOTOS in the specification phase, and on estimation methods at different levels of abstraction in the partitioning phase. The LOTOS specification describes the system as a set of interacting communicating processes. Our HW-SW partitioning algorithm is guided by communications, performance and area estimates and by the suitability of each process for implementation in hardware or software. A partition is evaluated against the design goals and constraints, first using high-level estimates and then, if requirements are met, computing estimates at lower levels of abstraction. If the partition fails, the partitioning model is updated with the new low-level estimates and a new partition is generated. If it succeeds, the resulting hardware and software specifications are synthesized using existing high-level synthesis tools and compilers.

1 Introduction

Hardware-Software co-design is viewed as a promising cost-effective approach to build complex systems. Cost-effectiveness derives from an appropriate partitioning of the system tasks among hardware components (application specific ASICs), which are fast but expensive, and software components (executing in one or more programmable devices, usually processors), which are slower but inexpensive. Besides, certain behaviors are better suited to HW or SW implementations.

The two main tasks in co-design are (1) specification and formal transformation, and (2) partitioning and translation to synthesizable HW and SW descriptions. In the co-design environment presented here, the system is specified using LOTOS as description language, and it is partitioned according to high-level cost and performance estimates. LOTOS is most appropriate to describe systems with control-dominated interaction between processes. Partitioning is guided by the system goals and constraints. The standard procedure for evaluating a specific partition against these requirements has usually been to co-simulate the synthesized HW and SW. However, this is a costly process that, ideally, should be used only to verify the final design. Instead, we follow an alternative approach where the design is refined as much as possible using approximate cost and performance estimates before actually reaching the synthesis and simulation phase [5, 3, 4, 9].

This paper is structured as follows. Section 2 introduces the co-design methodology. Section 3 describes the specification phase based on LOTOS. Section 4 presents the partitioning phase. The evaluation methods that provide estimates to guide partitioning are described in Section 5. Finally, Section 6 presents some concluding remarks and future work.

2 Methodology

A basic methodology for HW-SW co-design is represented in Figure 1. In the specification phase, the system is described using the formal language LOTOS. LOTOS has the characteristics required for system level specification. The goals and constraints that guide the co-design process are part of the description. Formal transformations are applied to optimize it.

After the specification phase, syntax and semantics are analyzed to build an intermediate form over which HW-SW partitioning is performed. A HW-SW target architecture is defined consisting of one processor running the SW, one ASIC implementing the HW and a shared memory accessed through a common bus. Interface modules are used to connect the processor and the ASIC to the bus, allowing for communications and data transfers between them. This architecture is scalable...
with respect to the number of processors and/or ASICs. It also permits the study of nondeterminism at the system level caused by interference at the shared bus. In this methodology, partitioning pursues the global minimization of a cost function based on communications, performance and area estimates and the suitability of HW or SW components to implement specific functions. This optimization is based on modeling and evaluating an initial partition against goals and constraints. If they are not met, the evaluation model provides more accurate high-level estimates and a new partition is generated, modeled and evaluated. This process is repeated until the system requirements are met.

Partitioning provides two LOTOS descriptions for implementation as HW and SW respectively. A compiler from LOTOS to VHDL is used to translate the HW system level description so it can be synthesized by a high-level synthesis tool for a specific target technology. Similarly, another compiler translates the SW description from LOTOS to C which is then compiled for the target processor, thus providing the executable machine code. The resulting synthesized HW and machine code are verified together in a co-simulation environment against non-functional and functional constraints.

The methodology handles specifications at four levels of abstraction (Figure 1). In the partitioning phase, evaluation of a partition against temporal and non-functional constraints can be performed at any of these levels: (1) system specification, (2) HW and SW specification, (3) synthesizable HW and SW, and (4) synthesized HW and SW. A partition can be evaluated at a level if there are estimation and/or simulation models available for that level. Otherwise, evaluation must be carried out at a lower level, thus requiring a design step to be performed. In the worst case from the point of view of the computation time spent in the partitioning phase, each partition must be synthesized and cosimulated (level 4). The methodology presented here reduces this computation time by using evaluation methods at higher levels. In particular, it defines a system estimation model (level 1) based on simulation results from level 1, HW estimates from level 3, and SW estimates from level 4.

### 3 Specification Phase

A specification language for codesign should encourage fully realization independent descriptions. The specification method should neither direct to a definite system realization nor support only one target architecture. The use of implementation oriented languages (like VHDL or C) for cospecification is completely inappropriate because the specification would bias the final realization, not allowing for a fair assignment of subcomponents to software and to hardware. This requirement enforces a high degree of abstraction.

Systems are usually composed of concurrent behaviors that cooperate with each other. The concurrent components interact to perform two tasks. One is the exchanging of data and, in addition, they need to synchronize with one another to coordinate their activities.

LOTOS [1] is an ISO standard language that fulfills all these requirements. It is a system level specification language that supports concurrency, synchronization, composition of processes and nondeterminism. The language supports a wide range of abstraction levels, from algebraic specifications to algorithmic-style versions. The standard includes also its formal semantics, and there-
where the processes synchronizing in one gate negotiate events, or it is an estimated/measured value. By comparing required versus estimated annotations, we can check feasibility and find bottlenecks in the design.

Following the data provided by [18], Table 1 shows a comparison between different specification languages according to some important features. Letters N, P and S mean that the corresponding feature is not, partially or fully supported, respectively.

The qualitative feature measures are:

- **Formal semantics**: Formal description of the language which allows for formal verification and transformation of specifications.
- **State transitions**: Support for describing state transitions for changing between modes in the model.
- **Behavioral hierarchy**: Ability to hierarchically decompose functionality into simpler pieces, both sequential and concurrent.
- **Concurrency**: Support for concurrent processes or procedures.
- **Program constructs**: Support for sequential algorithm description, similar to high-level software language constructs.
- **Exceptions**: Support for describing immediate reactions of the model to external events, like interrupt or reset.
- **Behavioral completion**: Explicit indication of completion of a behavior (e.g. signaling the ending of VHDL process).

The language describes systems by defining the interactions that constitute the externally observable behaviors and the temporal relationships between them. It is based on process algebraic methods and abstract data types. It consists of two parts: one, based on the calculi CCS [13] and CSP [6], for describing behaviors, and the other, based on the abstract data type language ACT-ONE, for describing data. The semantics of the behavior part is given in operational style as a labelled transition system.

A system and its components are represented in LOTOS by *processes*. A process displays an observable behavior to its environment in terms of permitted sequences of observable actions. A process interacts with its environment via *gates*. A gate models the logical or physical attachment points between a system and its environment. The interaction units between a LOTOS process and its environment are represented by *events*, where the processes synchronizing in one gate negotiate the value interchange.

There already are a number of tools for LOTOS. To mention a few, the following have been developed at our department: LOLA [16], a transformation and validation tool; TOPO [10], a translation tool to C; HARPO [2], a translation tool to VHDL; several editors for the textual and graphical versions of LOTOS, etc.

Design languages have tended to concentrate on the behavioral and structural aspects of systems rather than on non-functional requirements (such as space, performance, etc.). If non-functional requirements can be formally expressed in the early phases of system definition and carried over through the whole design cycle, a better design exploration could be carried out and decisions that otherwise would have to be made too early can be left for later, when more information is present.

LOTOS has been extended with quantitative temporal annotations and non-functional requirements to increase the amount of information available in the specification. The extended language, called X-LOTOS [17], includes non-functional annotations to deal with hardware, software and communications aspects of the design. Together with the initial requirements, the measures and estimates obtained during the evaluation of the constraints can be "back-annotated" to the specification. Functions have been formally defined to check their feasibility and consistency. This provides assistance at an early time to the designer, who will then be able to detect the bottlenecks in the design.

The non-functional annotations in X-LOTOS cover hardware, software and communications aspects. Hardware annotations relate to hardware characteristics. Some are related to physical properties of the ASICs (area, power, maximum number of pins, and maximum number of ASICS), and others are related to the mapping of the operations inside the X-LOTOS specifications to physical devices (mapping of operations to logical devices, mapping of logical to physical devices, and maximum number of functional units per ASIC). Software annotations relate to software characteristics (memory, CPU time, and maximum number of CPUs allowed). Communication annotations relate to communication among the partitions. There are two main kinds of annotations: those concerned with the amount of communications at a given point of the system (traffic and throughput), and those that deal with the implementation of the logical channels (declaration of physical channels and mapping of logical to physical channels).

Finally, other annotations show whether a process belongs to the software or hardware partition, distinguish several instantiations of the same process or give the bit width of a given sort. Apart from the name and parameters (when necessary), annotations may also have a field that indicates whether it corresponds to a user requirement, or it is an estimated/measured value. By comparing required versus estimated annotations, we can check feasibility and find bottlenecks in the design.
### 4 Partitioning Phase

In order to perform the partitioning task, the initial LOTOS specification is translated into a graph-based intermediate form called Process Communications Graph (PCG). In this graph, nodes represent processes which are connected by edges according to the communications between them. Processes can be assigned to HW or SW, being the base object to be considered during the partitioning task. Therefore, our approach is based on coarse granularity. There are several reasons to choose coarse granularity:

1. It is reasonable to maintain the processes as defined by the designer in the system-level specification as partitioning objects.
2. The complexity of the problem is smaller because the search space is reduced.
3. Granularity at the level of processes is more suitable for the translation procedures from LOTOS to VHDL and to C ([2], [10]), which also use processes as translation units.
4. Coarse granularity maintains the original structure of the specification and, therefore, it supports interactive design (the designer can identify the different behaviors)

Inputs to the partitioning phase are the PCG, cost and performance estimates, and the design goals and constraints. For every process $i$ the following estimates are required:

- Hardware area ($h_{ai}$) and execution times from any input gate $m$ to any output gate $n$ ($ht_{i}^{m,n}$).
- Software size ($ss_{i}$) and execution times ($st_{i}^{m,n}$).
- Average number of times the process is executed ($n_{i}$).

For the communications produced at every PCG edge $k$ the required estimates are:

- Synchronization times ($t_{sync,k}$).
- Average number of times the synchronization takes place ($n_{k}$).

Finally, the design constraints to be evaluated are:

- Total hardware area ($A_{HW}$).
- System response time/throughput ($T$).
- Software memory size ($S_{SW}$).

It is considered that maximum, minimum and average values are obtained for the all previous estimates. The units being used are clock cycles for times, equivalent gates for hardware areas, and bytes for software sizes. High-level estimation methods are described in the next section. At lower levels, several methods for obtaining estimates from synthesizable descriptions can be found in the literature ([11], [7], [4]).

Hardware-software partitioning is performed in two steps. First, a constructive method is employed to build the initial partition. Second, this initial partition is refined by means of an iterative algorithm. This method reduces the total computation time in comparison to an iterative procedure starting from a random partition.

The constructive step is implemented using a classical clustering algorithm [8]. This algorithm builds a cluster tree using a closeness function between objects. The processes in this tree are assigned to HW and the remaining processes are assigned to SW. The clustering proceeds until the HW area constraint is violated, assuming that the timing constraints are still not satisfied. Otherwise, the partitioning phase is finished. The closeness function takes into account the estimates given as inputs. In particular, it groups those processes that have characteristics more suited to hardware implementation and that communicate frequently with each other. This reduces the communications overhead in the interface between hardware and software. The closeness function can be described in the following way:

$$C_{ij} = \frac{n_{i} \times \Delta t_{i}^{m,n} + n_{j} \times \Delta t_{j}^{m,n} + f_{comm}(i,j)}{ht_{i}^{m,n} + ht_{j}^{m,n}}$$

where $\Delta t_{i}^{m,n} = st_{i}^{m,n} - ht_{i}^{m,n}$ represents the performance improvement obtained if the process $i$ is moved from software to hardware, and $f_{comm}(i,j)$ is a function that is proportional to the average number and cost of communications between processes $i$ and $j$.

<table>
<thead>
<tr>
<th>Language Features</th>
<th>VHDL</th>
<th>CSP</th>
<th>StateCharts</th>
<th>SDL</th>
<th>SpecCharts</th>
<th>LOTOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal semantics</td>
<td>N</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>State transitions</td>
<td>N</td>
<td>N</td>
<td>S</td>
<td>P</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Behavioral hierarchy</td>
<td>P</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Concurrency</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Program constructs</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Exceptions</td>
<td>N</td>
<td>N</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Behavioral completion</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

Table 1: Specification Language Features
The second partitioning step is only performed when the initial partition does not meet the timing requirements. Therefore, it must be refined. Refinement is performed by a group migration algorithm that works with the last objects added to the hardware cluster and ends when the time requirement is satisfied.

5 Estimation for HW-SW Codesign

Evaluation can be performed through simulation or estimation. Simulation, as defined here, obtains quality measures executing a design model, while estimation provides quality measures solving an approximate model of the design. It is usually the case that simulation models are interpreted, that is, they require the definition of input data sets. On the other hand, estimation models tend to be uninterpreted, based on average values or other approximations. Therefore, in general, simulation is more accurate but requires more computation time than estimation. For this reason, both approaches are complementary and can be used in the design process. Since computation time is usually a limiting factor, simulation is preferred at high levels of abstraction and estimation is used at lower levels.

In this methodology, the LOTOS simulation tool TOPOSIM [12] provides measures about the frequencies of synchronizations and process executions. The other values used to evaluate a PCG partitioning are obtained from uninterpreted estimation models at lower levels.

The times $T_{\text{DFG}}$, $T_{\text{COND}}$, $T_{\text{CALL}}$, $T_{\text{LOOP}}$ associated to the whole CDFG, performance related metrics (like response times) depend on the path followed through the CDFG during execution. The response time model of the whole CDFG is based on the addition of response times associated to the control nodes in each execution path. Since each control operation can include a local DFG, response times for procedural calls, conditionals and loops can be computed as:

$$T_{\text{CALL}} = T_{\text{DFG}}$$

$$T_{\text{COND}} = \sum_i p_i T_{\text{DFG},i}$$

$$T_{\text{LOOP}} = n T_{\text{DFG}}$$

where $n$ (iterations in the loop) and the probabilities $p_i$ (of each outcome of the conditional statement) are obtained from simple probabilistic models or from abstract interpretation. By following the hierarchy in the description, any $T_{\text{DFG}}$, including the one for the whole CDFG, can be expressed in terms of the response times of the DFGs at its leaves:

$$T_{\text{DFG}} = F(T_{\text{DFG,1}}, \ldots, T_{\text{DFG,J}})$$

The times $T_{\text{DFG}}$ depend on the operator response times which are fixed for a given operator library and on the data-path structure resulting from synthesis. The impact of the times $T_{\text{DFG}}$ in $T_{\text{DFG}}$ is described by the function $F$ which associates each $T_{\text{DFG}}$ to the frequency of execution of the DFG, $f_{\text{DFG}}$. Therefore, it only remains to be determined how structural components are combined to execute a DFG in order to obtain the DFG
estimates. This is done by modeling the high-level synthesis tasks.

It is assumed that a data-path architecture has two levels of multiplexers for register access and functional unit access. All transfer times are characterized. Data can be fetched from memory. Since the memory response time is variable (memory is shared with the SW component) the ASIC remains idle while memory reads are performed. The control block receives status lines from the data-path and generates the control signals required to perform each DFG in the global CDFG.

High-level synthesis performs the following tasks: mapping, clocking, scheduling, allocation and binding. Mapping is estimated approximately considering that each operator type is mapped to only one component type (isomorphic relationship). The assignment is driven by the overall goals and constraints. Clocking is estimated using the clock slack minimization method [15]. After this step, all component delays are rounded to clock cycles. Scheduling and allocation are estimated from the results of scheduling the CDFG twice for two particular cases of allocated resources. These tasks are modeled in the following paragraphs. Binding is approximated by simple assignment rules in the scheduling/allocation model.

The estimation of scheduling and allocation provides the timing and the resources required to execute the CDFG. Assuming an ASAP algorithm (As Soon As Possible) for scheduling, it is applied twice for two particular sets of allocated resources: no resource constraints and a single component of each type. Unconstrained number of resources does not mean one component per operator, since components are reutilized in different time steps (shared). Scheduling for no constraints provides the maximum number of required components of each type, $C_i$, the number of operations performed by the $j$th component of type $i$, $N_{i,j}$, and the (minimum) time to execute the CDFG, $T_{\text{min}}$. The execution time, $T_{\text{min}}$, is modeled with the following expression:

$$T_{\text{min}} = \alpha[t_{\text{mem}} N_{\text{mem}} + \sum_i t_i N_{i,1}]$$  \hspace{1cm} (2)

This model assumes that $T_{\text{min}}$ is proportional to the sum of products of component delays, $t_i$, multiplied by the number of operations performed by the first component of each type. Equivalent terms are defined for memory accesses, $t_{\text{mem}}$ and $N_{\text{mem}}$, although they are maintained apart since there is only one data memory. The approximated model only considers the first component of each type since all other components operate in parallel. Since $T_{\text{min}}$ has been determined, the value of the constant $\alpha$ can be computed from the model for later use during HW-SW co-estimation.

Scheduling the CDFG for a single component of each type is the most restrictive case. It provides the maximum time to execute the CDFG, $T_{\text{max}}$. A new model is developed for this time, given by the expression:

$$T_{\text{max}} = T_{\text{min}} + \beta \sum_i t_i \sum_{j=2}^{C_i} N_{i,j}$$  \hspace{1cm} (3)

This model adds to the unconstrained response time $T_{\text{min}}$ the time to execute all the operations previously executed in parallel that now must be performed sequentially multiplied by a constant $\beta$. This constant can be computed since $T_{\text{max}}$ is obtained from the second scheduling and the other values in the expression are known.

In this situation, a general model which relates the response time of a CDFG with a generic set of allocated resources is developed. In this set, $X_i$ is the number of allocated components of type $i$. The model is expressed as:

$$T_{\text{CDFG}} = T_{\text{min}} + \beta \sum_i \frac{t_i}{X_i} \sum_{j=1}^{X_i} N_{i,j}$$  \hspace{1cm} (4)

For each component type, it computes the additional time resulting from operations that could be performed in parallel with $C_i$ components but must be done sequentially with $X_i$ components. These operations are assigned to the available $X_i$ components assuming equal probabilities, so the operations performed by the first one in the list are obtained simply dividing by $X_i$. The sum of the additional times for all component types is multiplied by $\beta$ and added to $T_{\text{min}}$. The model provides $T_{\text{CDFG}}$ once resources are allocated (area constraint).

If $T_{\text{CDFG}}$ is known (timing constraint), resources can be allocated maintaining the proportion of resources of each type obtained in the unconstrained case.

Estimation of the number of registers requires the CDFG to be scheduled a third time, now considering the $X_i$ components that have been allocated. In this situation, the number of registers is determined through live variable analysis [14]. Multiplexers are assumed to have as many inputs as types of components are required ($i \times 1$ MUXs). It is estimated that there are as many MUXs between registers and functional components as the total number of scheduled functional components, and as many MUXs between components and registers as registers. This assumption models the case where at least there is a path between each register and each component type viceversa.

The control block is synthesized as a state machine implemented with a state register and a ROM for the control logic. The size of the state register is obtained considering that the total number of states is the number of time steps required to execute the whole CDFG. With respect to the ROM, it is assumed that there is one control line for each register and each functional component.
and \( \log_2 i \) control lines for each MUX. In this situation, the size of the ROM is estimated as:

\[
(S \times [(1 + \log_2 i)(R + \sum X_i) + \log_2 S]) \text{bits}
\]

where \( S \) is the number of states and \( R \) the number of registers. The word size depends on the total control lines and the bits to represent the next state.

### 5.2 Software Estimation

The methodologies for HW and SW estimation have several points in common. In both cases, they take DFGs as inputs and there is a target technology or a target processor that is characterized through technology files. However, there are differences too. The DFG operations of the HW description can be executed in parallel, while SW operations must be executed sequentially. Besides, the VHDL description of the HW is expressed in terms of a modeled VHDL library while the C code describing the SW is modeled at a lower level (assembly code). This is due to the fact that an off-the-shelf processor (and C compiler) is used to run the SW, so unknown (proprietary) optimizations are applied in the compiler and, therefore, they are difficult to model. This does not occur in the HW because the synthesis tool provides the mechanisms to force the synthesis using the modeled library.

Once the assembly code is obtained, the response time of the complete CDFG can be estimated from a technology file. The technology file describes the instruction set of the target processor, including the instruction execution time for each instruction type of the set (ALU, move, load/store, conditional branch, unconditional jump).

Estimates of the required memory space are obtained separately for instructions and data. The memory space for instructions can be computed from the previous technology file of the instruction set assuming that it also includes the size of each instruction. All instructions in the code must be considered to compute the total required memory. Space for data is estimated from the declared data types and structures in the original C code and a corresponding technology file which contains the sizes associated to each basic data type.

### 5.3 Hardware-Software Co-estimation

Hardware-Software co-estimation is performed to model the interference that occurs between HW and SW accesses to the shared data memory and its impact on performance. The response times provided by the independent HW and SW estimation approaches can be expressed as:

\[
T_{\text{HW}} = T_{\text{CDFG,HW}} + a t_q N_{\text{mem,HW}}
\]

\[
T_{\text{SW}} = T_{\text{CDFG,SW}} + t_q N_{\text{mem,SW}}
\]

These equations add the waiting time due to interference when accessing the shared memory to the response time of the corresponding CDFG. The waiting time is modeled in terms of the average waiting time per access to the shared memory, \( t_q \), and the number of memory accesses, \( N_{\text{mem}} \), in each case. The equation for the HW includes the constant \( a \) that was computed during HW estimation so that the models with and without interference are consistent.

When evaluating the timing of a part of the PCG that crosses HW-SW boundaries, actions in HW and SW are represented by their corresponding DFGs, whose response time ultimately depends on their DFGs and the memory queuing time, \( t_q \). This value must be determined to estimate the synchronization times at the boundary. Therefore, the response time \( T_{\text{HW-SW}} \) is expressed in terms of the corresponding \( T_{\text{HW}} \) and \( T_{\text{SW}} \) and relates to \( t_q \). Equivalently, such expression relates the average flow of memory requests \( F_{\text{mem}} \), obtained from \( N_{\text{mem}} \), to \( t_q \).

In this situation, a queueing model of the system architecture is introduced. This model contains a service center, the data memory being accessed through a common shared bus, and two queues that receive the flows of memory requests from HW and SW. The average waiting time in the queues are obtained from approximate analytic queueing models. The solution provides the value of \( t_q \) in terms of the average arrival rates, that is, the flow of memory requests \( F_{\text{mem}} \). This solution can be combined with the expression of \( F_{\text{mem}} \) derived from \( T_{\text{HW-SW}} \) in the system of equations:

\[
F_{\text{mem}} = \Omega(t_q)
\]

\[
t_q = \Psi(F_{\text{mem}})
\]

where functions \( \Omega \) and \( \Psi \) can be quite complex. One approach is to use approximate methods to find the solutions. In particular, iterative methods, like binary division of an initial interval of possible solutions, are usually fast. Once \( t_q \) and \( F_{\text{mem}} \) are obtained, estimates of \( T_{\text{HW-SW}} \), the objective of co-estimation, are computed.

### 6 Concluding Remarks

A methodology for HW-SW co-design has been introduced. Its key elements are the use of a formal description technique as the basis of the specification and formal refinement of system descriptions, and the use
of a partitioning strategy based on approximate estimates at several levels of abstraction. Special emphasis has been placed in developing high-level cost and performance predictors based on a library of structural components and high-level synthesis and co-estimation models. Future work will cover some of the issues left open in the methodology, including the evaluation of results for a set of benchmarks of system descriptions. It is also desirable to extend the methodology to other execution models and architectural features like pipelining, so predictors of other quality metrics like throughput can also be obtained. Finally, the accuracy of the estimation models should be improved modeling some secondary aspects not considered here, like the wiring area in the HW estimation. At present, we are working on several cases studies in order to validate and improve our methodology.

Acknowledgements

We would like to thank Andrés Marín (Dpto. de Ingeniería de Sistemas Telemáticos) from the Universidad Politécnica de Madrid, for his insights into LOTOS to VHDL translation.

This work has been partially supported by ESPRIT project No. 8135 COBRA (Hardware/Software Codesign Basic Research Action) and by CICYT projects TIC93-0515-C02 and TIC94-0627-CE.

References