A Model for System-Level Timed Analysis and Profiling

A Allara (1) W. Fornaciari (2,3) F. Salice (3) D. Sciuto (2)
(1) ITALTEL, Central Research Labs, CLTE, 20019 Castelletto di Settimo m.se (MI), Italy.
(2) Politecnico di Milano, Dip. di Elettronica e Informazione, P.zza L. Da Vinci 32, 20133 Milano, Italy
E-mail: {fornacia, sciuto}@elet.polimi.it
(3) CEFRIEL, via Emanueli 15, 20126 Milano (MI), Italy, E-mail: salice@mailer.cefriel.it

Abstract

Fast evaluation of functional and timing properties is becoming a key factor to enable cost-effective exploration of mixed hw/sw design alternatives for embedded applications. The goal of this paper is to present a modeling strategy to specify functionality and timing properties of uncommitted mixed hw/sw systems. In addition, the paper proposes a simulation algorithm able to perform fast high-level simulation of the system by taking into account the initial hw vs sw allocation of system modules. The related CAD simulation environment allows the designer to access profiling information which can be useful to remodel the system to meet the functional/timing goals as well as to drive the following hw vs sw partitioning activity. Experimental data obtained by reengineering an industrial design are also included in the paper.

1. Introduction

The rise in global competition has dramatically shortened the available product development time [1]. Mixed hw/sw architectures are becoming common in order to tradeoff performance and cost and the borders among the typical design flow activities are vanishing, frequently partitioned in architecture definition, functional design, independent hw and sw development, system integration and global debug [3]. Unfortunately, this classical approach suffers of a number of drawbacks, the most relevant one being the shifting of a significant part of the system tuning and debug at prototyping time.

Under the EDA point of view, the problems in designing embedded systems cover the capturing of constraints both on behavior and timing-related issues, scheduling, hw vs sw partitioning, synthesis of CPU code, interfaces and hardware parts. All these aspects, should be supported by analysis, simulation and debugging activities, possibly through methodologies easy to be integrated in industrial CAD environments and in particular exploitable during the early stages of the design.

At this time, for a variety of reasons (both technological and economical), it is hard to find full functional hardware emulators operating efficiently to simulate cooperation between the hw and sw domains [2] [4]. Current VHDL simulation environments are good candidate to replace dedicated hardware emulators and the leading edge of the research is proposing attractive solutions for efficient low-level simulation of embedded software [5] [17]. However, as emerging by a market survey we carried out within the ESPRIT- ESD project SEED (Software-hardware Exploration, a European Demonstration Project) [11], the most important *value added* from the designer's perspective is the possibility to roughly estimate at system-level, before committing to a specific hw-sw implementation, the satisfaction of functional and timing requirements and to evaluate the impact of moving pieces of specs from hw to sw and *viceversa*.

The goal of this paper is twofold: first to define a specification technique allowing the specification of both functional and timing requirements of embedded architectures. The second goal is to define a verification strategy based on a high-level simulation of such specification, while allowing execution profiling.

The impact of the methodology onto a real test case has been carried out during the SEED project, where an industrial design provided by Italtel (the main Italian Telecom industry) has been redesigned.

The embedded architecture considered is constituted by one CPU core cell surrounded by some hw modules acting as peripheral co-processors. The starting point of the system analysis is a description captured by using the TOSCA editor [6] producing a OccamII-based description of the behavior, where the original specification formalism has been extended to cover also the representation of timing properties. The simulation of the system proceeds by taking into account the hw vs sw allocation of design subparts by estimating their execution times according to the strategies described in [7] [12] and by considering also the overhead due to the bussed hw-sw communication and the presence of only one executor (the CPU) for the sw-bound modules. System evolution can be monitored by querying the simulation database through SimView™, a graphical waveforms generator provided by Mentor Graphics. As a result, system profiling information is obtained through ASCII reports on the activities and constraint violations of the system

processes. A backannotation of such measures in the graphical specification is also provided. Such information will constitute the starting point to drive the design space exploration where the system is *reshaped*, by exploiting formal properties of the process algebra [13] [14] along the guidelines recalled in [6].

The paper is organized as follows. The next section presents the modeling strategy adopted to represent the system behavior, with particular emphasis on the timing extensions, while giving appropriate references for readers more interested in syntax details. Section three discusses how the high-level timing characterizations of the various *actors* of system -hardware software and communication- have been considered. The simulation algorithm on which the system-level analysis is based, is presented in section five, which shows how the hw-sw communication, and CPU process scheduling have been taken into account. Section six contains a brief presentation of the related CAD simulation environment, where some results extracted from an actual industrial design of Italtel used as benchmark for the SEED ESPRIT project, are shown. Some conclusions will be drawn in section seven, together with an outline of the ongoing research activity.

2. The timed system specification

A valuable design environment for real-time systems, should provide the user a modeling strategy able to capture the functionality and the timing constraints both on

- *performance*, to specify bounds on response time;
- *behavior*, to specify the rate at which the external stimuli have to be applied.

In current practice, global timing constraints are known *a priori*, since usually they represent a significant part of the design requirements and have to be carefully considered during the hw vs sw partitioning task. By considering an *event* as a external stimulus, as well as an observable response of the system, according to [8] three different timing restrictions can be envisioned between events: *maximum, minimum* and *duration* (of a given event). Hence, the possible combinations of max/min timing constraints are:

- *stimulus-stimulus*: max/min admissible delay between two stimuli;
- *stimulus-reaction*: max/min delay between the stimulus and the corresponding system reaction;
- *reaction-stimulus*: max/min insensitivity time among a system reaction and the following stimulus;
- *reaction-reaction*: max/min time between two system reactions.

Other approaches, such as [9] [3], address the definition of timing properties by considering rates and min/max constraints *tied* to the *operations* characterizing the events, instead of them. We followed a similar approach, by extending the expressiveness design representation of TOSCA (based on a Occam II graphical editor) through the possibility to anchor *tags* (labels) to processes and to specify the following constraints:

```
MAXDELAY OF <tag> IS <value>
MINDELAY OF <tag> IS <value>
MAXDELAY FROM <tag1> TO <tag2> IS <value>
MINDELAY FROM <tag1> TO <tag2> IS <value>
MAXRATE OF <tag> IS <value>
MINRATE OF <tag> IS <value>
```

As shown in the example of figure 1, all the constraints are gathered within a *constraints definition section* grouping all the properties defined over the labels.

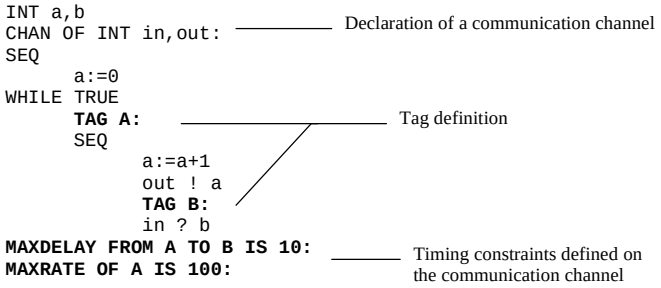

Figure 1. Definition of timing constraints within a TOSCA system description.

The Occam II description is composed of simple processes (e.g., a simple addition) as well as processes acting as a "host" for others (see SEQ statements in figure 1), more details on the Occam II system representation can be found in [6]. Apart the above timing extensions, the system description contains information on the hw or sw allocation of processes, which can be defined at different granularity levels: simple process, *host* process, procedure. To reduce the complexity of exploring alternative allocation schemes, and according to the feedback from the interviews performed to industrial designers within the framework of the SEED ESPRIT project, it has been decided to consider the procedure as a "monolithic block" not to be further decomposed. Moreover, it has been introduced the possibility do define vector of procedures (whose number is statically defined) and the restriction to avoid global variables. The latter assumption, imposes the procedures to access external variables through the parameters interface; in such a way, it is easier to identify the effects of moving procedures from hw to sw (and viceversa).

3. The hardware-software model

The crucial issue in designing a mixed hw-sw system is the allocation of functionalities to the hw or sw domain according to a set of user criteria. Possible strategies may consider the quality of a solution by moving down to the synthesis [16] [15] and then backannotating the result at the higher levels of the design flow. Other approaches, aiming at deeply and quickly spanning the space of possible solutions while accepting less precise results, found the decision on some estimations to be computed either by inspecting properties of the system description or by analyzing the system behavior through simulators working at the higher levels of abstraction (e.g. queue models).

Our strategy is a *meeting in the middle* approach. In fact, we developed a simulation strategy allowing the designer to:

1. debug the system behavior at the system level without the necessity to pass through the implementation steps;
2. take into account characteristics of the final implementation concerning the hw or software implementation domain of the submodules composing the system, and the *media* which will be actually used for communication (system bus, memory variables, dedicated hw lines);
3. verify the meeting of timing requirements when the specification is mapped onto the target architecture, with a sufficient confidence to make possible tuning and comparisons among possible alternative implementations.

The strategies adopted to estimate the execution time and the cost of each module composing the system are out of the scope of this paper, more details can be found in [5] [7]. The rest of this paper is devoted to the description of the simulation algorithm built on top of such estimations, and how those basic results have been mapped onto the OccamII operators (and the extensions we introduced), available in TOSCA to describe the system.

Delays and latencies of the specification take into account the same basic timing composition operations defined in [9], e.g., sequential, parallel and mutual exclusive. In addition, our proposal solves the problem (at simulation time) of considering the side-effects on the execution times of the software-bound operations due to the split of the microprocessor computational power among the running sw processes. In fact, a basic assumption in evaluating the actual effects of the target architecture is related to consider the availability of a single CPU shared by all the sw-bound processes, and the presence of multiple hw co-processors, each implementing a hw-bound module (the extension to cover different architectures is anyhow possible). Under these assumptions, the presence of a scheduler for the sw-processes is simulated, assuring *fairness* in distribution of the CPU time. The extension to consider different scheduling policies is straightforward, requiring only to modify the formulas calculating the number of active processes running on the CPU (see below).

The *intrinsic time* necessary to carry out each basic operation according to the hw or sw partition it belongs, can be gathered from a *project technology file* which has been automatically compiled by the TOSCA environment: for the sw operations the analysis considers the amount of instructions of a machine-independent assembler-like format, called VIS (Virtual Instruction Set) [5] [10], necessary to implement each Occam II construct [6]. The computation of the execution times for the hw-bound operations is based on the strategies presented in [7]. It has to be pointed out that the information on the hw or sw partition is crucial, even for computing the timing properties of a simple assignment; in fact the *right-value* of a compound statement, such as `a:=(b*c)+d/f`, can be treated by simply adding the *timing cost* of each single operation in case of sw-bound, while for hw-bound partitions the total execution time has to be computed by recursively evaluating the cost of each single operation of the expression. In general, for the hw-bound statements, we consider the possibility of evaluating two *boundary* cases: strictly sequential operations and maximum parallelism. In the former case the execution time is determined by simply adding the contribution of each single operation:

$$t_{execution} = t_{op} + t_{ex.right} + t_{ex.left}$$

where $t_{ex.rigt}$ and $t_{ex.left}$ are the computation time for the right and left operands of the *op* operation. In the case of fully parallelism, it becomes:

$$t_{execution} = t_{op} + max\,(t_{ex.right}, t_{ex.left})$$

For the remaining operators, the most important being: assignment, parameter passing, input, arithmetic/logic operations, conditioned branch, `ALT` construct, `SKIP` construct (see also [6] [11] [13] for more details), the computation is performed in a similar manner:

$$t_{execution}\ \textbf{\textit{IF}} = t_{op.if} + max\,(t_{ex.condition});$$

$$t_{execution}\ \textbf{\textit{WHILE}} = t_{op.while} + max\,(t_{ex.condition})$$

$$t_{execution}\ \textbf{\textit{ALT}} = t_{op.alt} + \sum_{i=0}^{n_{cond}-1} max\,(t_{ex.condition} + t_{val.input})$$

where $t_{val.input}$ is the time necessary to verify the presence of a datum ready on one of the input channels. Concerning the procedure call, by calling $t_{op.call}$ the time necessary to transfer a single bit, dim_i the size of the array of parameters, n_{bit} the size of each single location, it is:

$$t_{execution}\ \textbf{\textit{CALL}} = t_{op.call} + \sum_{i=0}^{n_{parameters}-1} max\,(\dim_i\ n_{bit}\ K)$$

where *K* is a factor transforming in seconds the elements of the sum.

The channeled communication, due to the blocking nature of the rendez-vous mechanism, has been modeled by using three different processing corresponding to the different phases of the communication: *input* process, *communication dummy* process and *output* process, since the datum producer has to wait until the consumer will unlock it by performing a *read* operation on the channel. The *input* and *output* processes have an execution time ($t_{instrinsic}$) modeling the transfer of the datum from the memory to the channel plus the actual transferring delay through the channel. The time necessary to move a single bit is assumed to be the same for the input and output processes, $t_{instrinsic}$ is evaluated on the basis of the type of datum carried by the channel, by multiplying its size for the number of bits constituting a given type (*type*, evaluating 1 for bits, 8 for bytes and 16 for integers) and for the time necessary to transfer a single bit:

$$t_{instrinsic} = n_{vector.size}\ type\ t_{bit}$$

Of course, both input and output processes can belong to different hw or sw partitions, in the latter case the process

becomes part of the CPU load, so that its execution time is dependent on the activity of the other active sw-bound processes.

As stated above, the actual communication is modeled through a *dummy* process. Three different cases can be conceived according to the domains to be connected:

- *Sw-Sw*: this kind of communication is performed through a direct call to the operating system, usually such a time in not relevant and it can be neglected.
- *Hw-Hw*: if the processes are located on the same ASIC, the transferring time can be neglected under the assumption to use dedicated lines, i.e. not engaging the system data bus.
- *Hw-Sw*: in this case the data exchange takes place between different partitions via the system bus, both operating system routines and dedicated hw of the co-processors are involved to carry out such a task. This circumstance is modeled by a dummy process which is aggregated to the CPU workload.

In addition to the process-to-process communication, another type of interaction exists, at a lower level of abstraction, between the process and the procedures it calls. Such kind of interaction is very sensitive to number and size of the passed parameters, and it is modeled similarly to the interprocess communication:

- *Hw-Hw*: a dummy process is introduced to represent the communication among hw partitions whose execution time is:

$$t_{HW.comm} = t_{bit.hw} + \sum_{i=0}^{n_{parameters}-1} (vector_i.dimension\ \ type_i)$$

- *Sw-Sw* this is the classical case of a software procedure call, the modeling of the context saving is pertaining the CALL process, which, running on the CPU affects all the active processes. Hence, the call/return procedure overhead is represented by a dummy sw-sw communication process whose execution time is:

$$t_{SW.comm} = t_{bit.sw} + \sum_{i=0}^{n_{parameters}-1} (vector_i.dimension\ \ type_i)$$

Hw-Hw: this case is modeled in the same way of the hw-sw communication among processes, a dummy process insisting onto the sw partition is created to model the context saving/restoring.

4. Putting it all together: the *time stretching* algorithm

In order to speedup the simulation/verification of the system behavior, an event-driven simulation strategy has been develped. The kernel algorithm is based on a *elastic* model of the time between events. The time proceeds by considering a discrete time model, where an event can be:

- start or end of a process execution;
- start or end of one of the following activities: loading of a datum on a channel, pick-up of a datum by a receiver process, datum transfer on a channel;
- procedure call;
- start or end of the computation of a test condition.

The simulation algorithm is able to *stretch* the intrinsic execution time of processes according to the CPU workload. The algorithm goes on by applying two different transformations on time (*expansion* and *contraction*) to incrementally find out the system latency and any violation of the timing constraints:

- The *expansion* is applied to a process due to the presence of more processes executed in parallel.
- The *contraction* performs the opposite action: giving an expanded timing axis (originated for example by an hypothetical parallel execution of sw-bound processes), the time is condensed so that each process has an elapsed time corresponding to the case where each one of them is the only CPU owner.

Let us in introduce some preliminary notation that will be used to describe how the simulation algorithm works.

- T_{abs}: time instant corresponding to the actual behavior of the system under simulation;
- $t_{intrinsic}$: execution time of a process by considering unlimited resources in case of hw-bound and a dedicated CPU in case of sw-bound;
- $t_{remaining}$: time to complete the execution in the case it is the only active process;
- n_a: number of sw-bound processes simultaneously active;
- τ_c: context switching time;
- T_c: commutation period;
- n_{Hsw}: number of sw-bound idle processes;
- T_{iH}: spooling period;
- τ_i: duration of the spooling process;
- M: time interval between the events *i* and *i+1*.

The reference point of the algorithm is the *absolute* time, which is the value visible to the user. The simulation proceeds until the first event, after that it is incremented by *M* according to the hw and sw processing being executed. M is computed as the minimum interval before the arrival of a sw or hw process with an expanded time. Sw processes are considered expanded to mimic the effect of the CPU scheduling algorithm, in such a way it is possible to estimate in a realistic manner which is really the first event among all the ones triggered by the active system processes. The *remaining* time is the time necessary to complete an instruction, for a sw process it is determined by contracting the time already expanded, i.e. it is the difference between the previous remaining time and the ratio between the increment M of the absolute time and a coefficient of expansion. In case of hw processes it is the difference between the previous remaining time and the interval M, without any necessity of contraction. This mode of operation is applied again after introducing new processes to be executed in parallel, and by computing new values of M (based upon the remaining times of the processes

partially executed and on the execution time of the new processes).

The execution time of a sw process is determined on the basis of three main factors:

1. The number n_a of active processes, which allows the estimation of the *virtual* workload of the CPU where the sw processes are being executed.
2. The context switching time τ_c, modeling the saving of variables and memory configuration of the process to be interrupted. The *swap* among processes is assumed to occur at a given rate whose period is T_c ; under these assumptions $(n_a - 1)\frac{\tau_c}{T_c}$ is the overhead due to the context switch of the n_a active sw processes (which in many cases can be irrelevant in comparison to T_c).
3. The number of processes (input or output) waiting to communicate because the datum they are waiting for is not available yet. These processes are managed by a *spooling* operating system demon which periodically (rate T_{iH}) verifies (consuming τ_i seconds) the idle communication processes, so that $n_{Hsw}\frac{\tau_i}{T_{iH}}$ is the time component necessary to test the n_{Hsw} idle processes.

In summary, the *expansion* of each sw process is determined as follows, where t_{sw} is the intrinsic time to execute the given statements (in clock cycles) on the CPU.

$$t_{sw.expanded} = d\ t_{sw} \quad \text{being} \quad d = n_a + (n_a - 1)\frac{\tau_c}{T_c} + n_{Hsw}\frac{\tau_i}{T_{iH}}$$

For instance, the assignment of a byte takes one clock cycle, while two clock cycles are required in case of assignment of an entry in a vector. As it can be understood by analyzing the above formula, it has been implicitly assumed a software scheduling algorithm granting the same CPU sharing to all processes. To cover the case of different policy, the algorithm requires some minor changes in the formula computing the factor of expansion *d*. A pseudo-code description of the algorithm is reported in figure 2.

$t_{remaining}(0) = t_{intrinsic}$;
loop:
  if $\nexists$ $proc_{hw}$ then $t_{min.HW} = \infty$;
  M = min (t_{min} of the expanded sw processes, t_{min} of the hw processes)
/ modeling of the execution activities (corresponding to M) */*
  $T_{absolute} = T_{absolute} + M$;
  $\forall$ active $proc_{sw}$, it is $t_{remaining}(i+1) = t_{remaining}(i) - \frac{M}{n_a + (n_a - 1)\frac{\tau_c}{T_c} + n_{Hsw}\frac{\tau_i}{T_{iH}}}$;
  $\forall$ active $proc_{hw}$, it is $t_{remaining}(i+1) = t_{remaining}(i) - M$;
  activation of new processes (if any);
endloop;

Figure 2. Pseudo-code of the Time Stretching (TS) algorithm.

5. CAD environment and experimental results

The presented algorithm has been implemented in C++ and it is now linked to the TOSCA TCL/TK-based user interface [11]. The analysis of the system behavior can be carried out in a twofold manner:

- by inspecting the *textual* information concerning the evolution of all the processes, the violation of the timing constraints, variable monitoring, etc, contained in the simulation reports produced by the tool;
- by using the *graphical* user interface, which performs queries on the simulation database through SimView™, a waveform tracer by Mentor Graphics

The user can ensure the *semantic* correctness of the specification under the functional point of view by using both interfaces, the first being particularly useful to verify the meeting of the real-time design constraints. In any case, apart from the purely functional debug of the system, a valuable source of information is represented by the simulation report which, among the other data, provides:

- process duration and execution frequencies (or latency); for each value the *max*, *min*, *mean* and *variance* over the simulation period is provided;
- violations of the timing constraints (possibly ordered by severity);
- profiling of process execution;
- dead-code and variables never accessed;
- list of processes remaining idle at the end of simulation, which may be *suspected* to be involved in deadlocks;

This allows the designer a better awareness of the critical sections of the code under the timing and computational points of view.

Concerning the performance of the simulator, we extensively stressed the system by modeling toy benchmarks and a real component (called ILC16), commercialized by Italtel, used as test vehicle for the SEED Esprit project. By using a SPARCstation 20 running at 85MHz, we observed an average *simulation ratio* around 16-18. This means that, to simulate a system characterized by 24K events/s, the simulator is able to process 1.5K events per CPU second. The first implementation delivers a throughput already good enough to enable its effective use for developing real-size designs. In particular, such a type of analysis seems to be two orders of magnitude faster than the low-level strategy [18], presented in [5], based on the use of VHDL models for both the hw and the sw.

The ILC16 component, is a data-link controller for sixteen asynchronous/synchronous data streams based on the HDLC protocol. The original design allocates the managing of the channels to a RISC CPU core cells, while the HDLC protocol processing is hw-bound; both section are embedded in the same integrated circuit.

The system has been reverse engineered by using the TOSCA environment and validated through the simulator here presented, the specification is composed of more than 4K OccamII lines of code. Each of the sixteen links is composed of:

- one in/out HDLC module, implementing the ISO/OSI level 2 functionality;
- one FIFO buffering the incoming/outcoming data streams;
- a DMA controller to transfer the data on the local bus interface.

In the OccamII specification, the FIFO is in between a producer and consumer processes (see figure 3).

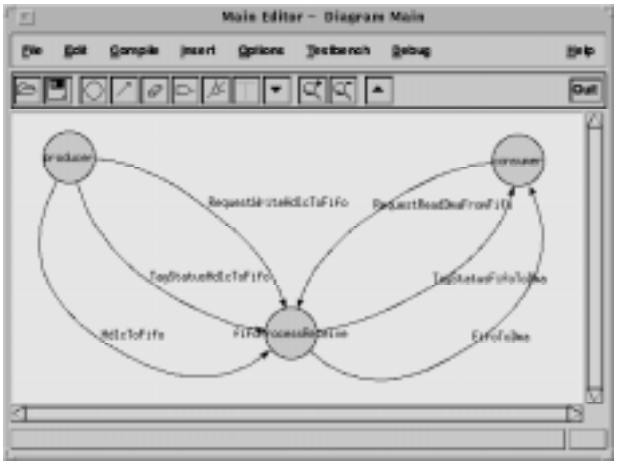

Figure 3. Graphical OccamII specs of the FIFO subsystem.

One of the bottlenecks of the architecture is represented by the transferring rate of the data from the FIFO to the consumer process, a low value with respect to the rate of the producer process will result in a FIFO overrun error. To model such a constraint, the statement `MINRATE OF Write IS 3` has been added to the Occam specification, where `Write` is a *tag* located on the process moving data from the FIFO to the consumer, and 3 is a frequency expressed in Hertz.

Other serious problems may arise from the latency of the process devoted to manage the case of FIFO overrun. In fact, the stall has to be removed (by a dedicated process) as soon as possible. In this case, the constraint is represented by `MAXRATE OF Full IS 4 (KHz)`, where `Full` is a *tag* anchored to the process of data refusing, and 4 is the max frequency expressed in KHz. Finally, a `MINRATE` constraint has been added to supervise a ALT process modeling the spooling of the processes communicating with the FIFO.

The various components of the system have been initially allocated to either hw or sw domains and then simulated to verify the functionality and the fulfillment of the timing constraints. The trend of some of the timing constraints versus the rate of the producer process is reported in figure 4.

Different scenarios have been considered before committing to the final hw/sw implementation, ranging from the fully hw and fully sw solution. Apart from the above considerations on the functional/timing properties, the system specifications have been modified trough a set of transformations according to the computation of some metrics evaluating the *quality* of the system [6], but this aspect is out of the scope of this paper. Different alternatives in terms also of microprocessor clock frequency have been taken into account.

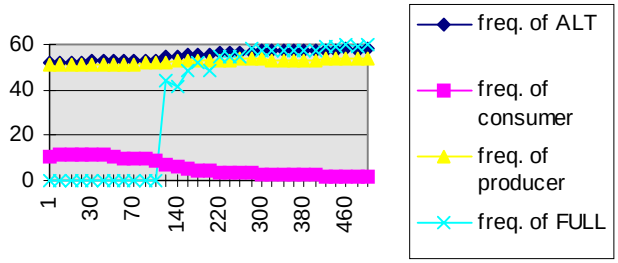

Figure 4. Trend of some timing constraints versus the data production rate of the consumer process feeding the FIFO.

The total manpower (excluding the synthesis stage) has been allocated on the different design stages as follows:

Occam Language (tool learning)	4 weeks
ExplorationManager (tool learning)	1 week
Design Specification	8 weeks
Functional Debug	16 weeks
Design Space Exploration	2 weeks
Total week/man	31 weeks

This results are important in order to get a feedback of the effectiveness of the proposed methodology for design entry and design space exploration. We can identify two different type of designer according to the level of confidence with the tool, whose temporal characterization of the activities is depicted in figure 5 and figure 6.

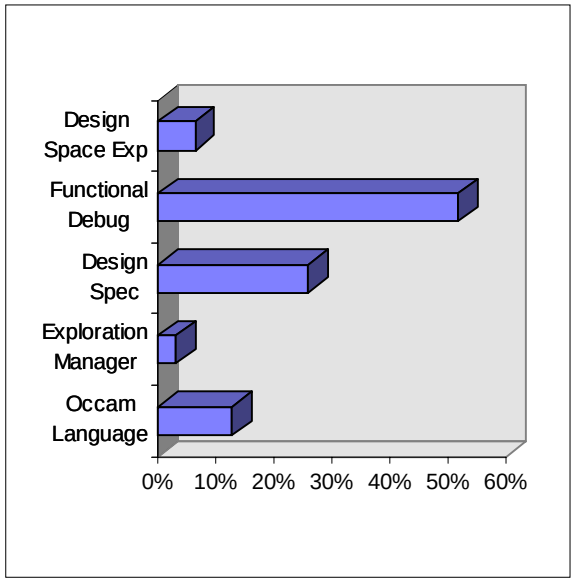

Figure 5. Manpower partitioning for a *starting from scratch* designer.

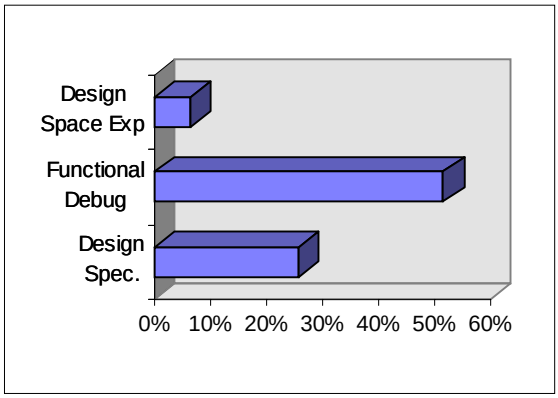

Figure 6. Manpower partitioning for a trained designer.

6. Concluding remarks

In many practical cases a significant value added is the possibility to roughly estimate if a given mixed hw/sw implementation fulfills both functional and timing requirements, during the early stages of the design flow. In such a way, once the primary goal of realizing a system not violating the specifications is met, if is possible to pay more attention in the design space exploration and the time-consuming finer grain optimization steps.

The aim of the presented investigation has been to fill the existing gap between the system-level uncommitted functional analysis (necessary to verify/debug the system) and low level simulation used to fine tune the architectural characteristics. The paper presented a simulation strategy based on a system level timing extension of a hw/sw modeling environment based on OccamII graphical system descriptions. Such information, together with an initial hw/sw pre-allocation of the modules composing the system, constitutes the substrate on which an high-level simulation algorithm is based. The resulting speed-up in simulation time, while the monitoring of the meeting of users defined timing constraints, allows the designer both to validate the design behavior on a given target architecture and to compare alternative solution in order to meet other figures of merit. Benchmarking of the simulation strategy onto a real-size industrial example is carried out as part of the activity of the SEED Esprit project.

The simulation algorithm has been integrated within the TOSCA codesign framework, and is now available to drive the design space exploration task in addition to the evaluation metrics and formal transformations presented in [6] [7]. Current effort in devoted to better integrate the evaluation metrics, the presented high-level co-simulation tool and the application of the available formal transformations under the *umbrella* of the TOSCA Exploration Manager.

7. References

[1] D. D. Gajski, L. Ramachandran, P. Fung et al., 100-hour Design] *Cycle: A Test Case*, Proceedings of the 31st ACM/IEEE Design Automation Conference, 1994.

[2] P.G.Paulin, C.Liem, T.C.May, S.Sutarwala, *DSP Design Tool Requirements for the Nineties: An Industrial Perspective*, Journal of VLSI Signal Processing, special issue on "Synthesis for DSP", vol.9, n.1-2, January, 1995.

[3] De Micheli G., Sami M. G. editors, *Hardware/Software Co-Design*,: NATO ASI Series, Series E: Applied Sciences - vol.310, Kluwer Academic Publishers, The Netherlands, 1996.

[4] Gil Lauder, *Bridge The Emulator Gap For High-Performance Debugging*, Electronic Design, vol. 44, n.24, November 18, 1996, pp 130-140.

[5] A.Balboni, W.Fornaciari, D.Sciuto, *Co-synthesis and Co-simulation of Control-Dominated Embedded Systems*, Int. Journal Design Automation for Embedded Systems, vol.1, n.3, July 1996, Kluwer Academic Publisher, Norwell, MA, USA.

[6] Balboni A., Fornaciari W., Sciuto D., *TOSCA: a Pragmatic Approach to Co-Design Automation of Control Dominated Systems*, Hardware/Software Co-design, NATO ASI Series, Series E: Applied Sciences - vol.310, pp.265-294, Kluwer Academic Publisher, 1996.

[7] A.Balboni, W.Fornaciari, D.Sciuto, *Partitioning of Hw-Sw Embedded Systems: a Metrics-Based Approach*, Integrated Computer-Aided Engineering, to appear, John Wiley, 1997.

[8] B.Dasarahy, *Timing contrstaints of real-time systems: Construct for expressing them, methods of validating them,* in IEEE Transaction on Software Engineering, vol.11, Jan 1985.

[9] R.K.Gupta, G.De Micheli, *System Synthesis via Hardware-Software Codesign,* Technical Report CSL-TR-92-548, Stanford University, October, 1992.

[10] A.Balboni, W.Fornaciari, D.Sciuto, M.Vincenzi, *The use of a Virtual Instruction Set for the Software Synthesis of Hw/Sw Embedded Systems*, In proc. of IEEE ISSS'96, 9th International Symposium on System Synthesis, La Jolla, California, USA, November 6-8, 1996.

[11] SEED ESPRIT-ESD project n.22133, *Reference manual* of the Occam Graphical Editor, http://www.cefriel.it/eda/projects/seed/mainmenu.htm.

[12] D.Gajsky, F.Vahid, S.Narayan, J.Gong, *Specification And Design of Embedded Systems,* Prentice Hall, New Jersey, 1994.

[13] Jifeng H., Page I., Bowen J. *Towards a Provably Correct Hardware Implementation of Occam.* Technical Report, Oxford University Computing Laboratory, 1994.

[14] Hoare C. A. *Communicanting Sequential Processes*, Prentice Hall, Englewood Cliffs, NJ, 1985.

[15] Gupta R.K., De Micheli G., *A Co-Synthesis Approach to Embedded System Design Automation*, *Design Automation for Embedded Systems*, Kluwer Academic Publisher, 1996, 1(1-2): 69-120.

[16] Benner T, Ernst R., Henkel J. *Hardware-Software Cosynthesis for Microcontrollers*, IEEE Design&Test, 1993, 10(4): 64-75.

[17] Paulin P.G., Liem C., May T.C., Sutarwala S. *Codesyn: A retargetable Code Synthesis Systems*, IEEE Proc. of Int. Symposium on High Level Synthesis, 1994.

[18] W.Fornaciari, F.Salice, D.Sciuto, *A two-level Cosimulation Environment*, IEEE Computer, June 1997, pp. 109-111.