A Flexible Model for Evaluating the Behavior of Hardware/Software Systems

A. Allara (a), S. Filipponi (b), W. Fornaciari (b, c), F. Salice (b), D. Sciuto (c)

(a) ITALTEL-SIT, Central Research Labs, CLTE, 20019 Castelletto di Settimo m.se (MI), Italy,
(b) CEFRIEL, via Emanueli 15, 20126 Milano (MI), Italy, E-mail: salice@mailer.cefriel.it
(c) Politecnico di Milano, Dip. Elettronica e Inform., P.zza L. Da Vinci 32, 20133 Milano, Italy,
E-mail: {fornacia, sciuto}@elet.polimi.it

Abstract

Hardware-software co-design in becoming a "must" for many embedded applications requiring to *tradeoff* a number of constraints such as size, cost, performance, real-time requirements, design flexibility, etc. . Even if, according to purpose of the digital system, the range of possible architectures is rather wide, for our field of interest (telecom embedded systems) the target architecture can be roughly described as composed of a microprocessor surrounded by some hardware modules connected through buses.

The aim of this paper is to present a model (and the related CAD environment) supporting the simultaneous analysis of functionality, timing performance (in terms of execution time of hw and sw modules and bus use), and execution *profile* of the system specification assuming the given target architecture. The goal of the proposed approach has been to define a simulation algorithm able to consider the partition of each section of the specification and the consequent bus traffic at the system level, in order to enable the designer to efficiently debug and evaluate the specification while considering the timing issues of a mixed hw-sw architecture very close to the final one. The paper gives also the *flavor* of the CAD environment built around the presented simulation strategy.

1. Introduction and motivation

The current market trend for the embedded applications is moving toward the realization of digital system under an increasing number of constraints (silicon, size, performance, power, ...), possibly with the goal of limiting the design cycle in 100 hours [1]. This complex scenario is forcing both the technology and the design methodology to tightly cooperate to satisfy the time to market requirements. Under the "silicon" viewpoint, for many applications the solution based on a *system-on-chip* implementation is becoming affordable, while the CAD environments are becoming more oriented toward the use of hardware description languages (such as VHDL) as the basis on which the entire design flow should be founded [2]. Nevertheless, most of the existing systems do not consist only of hardware components (e.g., ASICs), but a widely adopted strategy is to leverage the high cost of a fully silicon implementation with the shifting of some functionality towards a microcontroller where a dedicated (small size) software is running on it.

A widespread investigation we conducted by considering several companies producing embedded systems[1], revealed that an ideal design methodology supporting the developing of such systems, has to allow the designer to evaluate alternative hw-sw architectures during the early stages of the design. In particular, for the field of interest of our industrial sponsor (Italtel, which is the main Italian telecom company) whose core business is producing equipment for public switching, it is crucial to split the partitioning into two phases: the first one, according to a manually driven coarse grain estimation of the functionality is devoted to fix specification errors and to meet the real-time constraints, then, once the functionality is established, the task of the designer is to map it onto the hardware and software modules while fine tuning cost and performance tradeoffs.

A key issue of such a process is to quickly estimate the main timing characteristics of the specification before moving down in the detailed analysis of each solution. The potential benefit is twofold: during the early stages of the design flow it is possible to discard the architectural alternatives poor in performance or not satisfying the timing requirements, the latter is the possibility to *profile* the system specification which is very valuable to drive the hw vs sw partitioning task.

The TOSCA [3] project is based on a design flow encompassing all the stages of the development of a hw-sw system. The top of the design flow is the capture of both the system level specs and the goals/constraints of the project. The functional validation can be performed at two different levels of detail:

- *system-level*: where the functional bugs can be fixed and high-level profiling and structural analysis of the code can be performed to establish a near-optimal hw-sw partitioning;

[1] This market survey is part of the ESPRIT- ESD project n. 22133: SEED (Software hardware Exploration, a European Demonstration project).

0-8186-7895-X/97 $10.00 © 1997 IEEE

- *architectural*: where the system has been synthesized in terms of VHDL for the hw and assembly code for the software. In [4] such a low-level co-simulation strategy has been described. It is fully carried out within a VHDL environment, since it is based on the use of a generic representation of the software (VIS, Virtual Instruction Set) for which an instruction-level execution engine written in VHDL has been developed.

The focus of this paper is on the system-level simulation of the system, aiming to *booster* both the system validation and partitioning tasks. The system specification is based on a Occam II description [5] [6], which allows the representation of both parallel and sequential execution of processes at any abstraction level. Communication between concurrent processes is obtained through *rendez-vous* channels [5] [6]. An important advantage of using channels to represent communication is the possibility to apply the same compact model of data exchange between heterogeneous processes (and consequently interfaces). The analysis is performed by considering the Occam II system specs captured via the graphical environment of TOSCA, which extends the language to cover the definition of maximum, minimum and duration constraints among events, which can be both stimuli internal to the system and external.

The system is intended to be implemented onto a single chip where a microprocessor core cell interacts with other peripheral hardware modules called co-processors. Hardware to hardware communication is performed through the system bus while hw to hw data exchange is carried out by using dedicated lines. The top level assumptions on which the simulation is based are derived from an initial pre-allocation of the modules in hw or sw; according to this information and the computational requirements of each module, a simulation algorithm has been defined considering the following aspects to comply with the behavior of the final system:

- bandwidth requirement for the bussed hw-sw communication;
- sw to sw and hw to hw communication;
- computational workload of the microprocessor;
- execution time of the hw bound modules of the specification.

The paper is organized as follows, the next section presents the hardware/software model on which the simulation strategy is based, while section three introduces the timing analysis onto which the simulation kernel has been built. Section four, for lack of space, briefly summarizing benefit of the proposed approach to the designer and related CAD environment. The future directions of our research are finally discussed in section five.

2. The hardware-software timing model

To enable the use of Occam II to describe mixed hw/sw systems, apart from improving the design entry task through the definition of a mixed textual/graphical CAD environment, the expressiveness of the language has been extended to cover the following combinations of timing constraints [7]:

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

For such a purpose, the internal Occam II representation stored within the design database has been customized to link the functional specification to the timing constraints (**mindelay, maxdelay, minrate, maxrate**). The Occam II code contains some *labels* (**tags**) anchored to specific points within the specification and a *constraint definition section* grouping all the constraints defined over the labels, as shown in figure 1.

```
INT a,b
CHAN OF INT in,out:          ———— Declaration of a communication channel
SEQ
      a:=0
WHILE TRUE
      TAG A:                 ———— Tag definition
      SEQ
            a:=a+1
            out ! a
            TAG B:
            in ? b
MAXDELAY FROM A TO B IS 10:  ———— Timing constraints defined on
MAXRATE OF A IS 100:                the communication channel
```

Figure 1. Definition of timing constraints within a Occam II specification.

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). The hw or sw pre-allocation can be defined at different granularity levels: simple process, *host* process, procedure. To reduce the complexity of exploring alternative

allocation schema, and according to the feedback from the Italtel designers we interviewed, it has been decided to consider the procedure as a "monolithic block" not to be further decomposed. More details on the Occam II system representation can be found in [4].

The proposed timing analysis strategy to find out delays and latencies of the specification takes into account the same basic timing composition operations defined in [7]: 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.

The analysis begins by considering the *intrinsic time* necessary to carry out each basic operation according to the hw or sw partition it belongs to. These data 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 VIS instructions necessary to implement each Occam II construct [3], while the computation of the execution times for the hw-bound operations is based on the approach presented in [8]. The basic operations considered are the ones defined in [3] [4] [8], the most important are: assignment, parameter passing, input, arithmetic/logic operations, conditioned branch, **ALT** construct, **SKIP** construct. It has to be pointed out that the information on the hw or sw partition is mandatory, 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 conside the possibility of evaluating two *boundary* cases: strictly sequentially 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 computation is performed in a similar manner:

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

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

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

where $t_{val.input}$ is the time necessary to verify the presence of a datum ready on one of the input channels.

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

where $t_{op.call}$ is the time necessary to transfer a single bit, dim_i is the size of the array of parameters, n_{bit} the size of each single location and K is a factor transforming in seconds the elements of the sum.

Concerning the communication among channel, due to the blocking nature of rendez-vous, three different phases have been modeled - *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 moving 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 both for all the input and output processes, $t_{instrinsic}$ is evaluated on the basis of the type of the 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 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:

- *Sw-Sw communication*: 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 communication*: 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 communication*: 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.

3. The *time-stretching* algorithm

Despite the system specification allows the designer to represent parallel processes execution, it is an hard task for the designer to consider the actual presence of only one sw executor (i.e. the CPU). In order to verify the timing correctness of the specification and to drive the hw vs sw allocation we developed a simulation algorithm 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 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 of them is the only CPU owner.

Before presenting the algorithm, let us in introduce the following notations, whose meaning will be presented step by step later:

- 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}: the spooling period;
- τ_i: duration of the spooling process;
- M: time interval between the events *i* and *i+1*.

The algorithm is *event-driven*, 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 transferring on a channel;
- procedure call;
- start or end of the computation of a test condition.

The *absolute* time is the reference value that is visible to the user, it goes on 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 very precise 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).
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 $t_{remaining}(i+1) = t_{remaining}(i) - \dfrac{M}{n_a * (1+\frac{\tau_c}{T_c}) + n_{Hsw} * \frac{\tau_i}{T_{iH}}}$;
  $\forall$ active $proc_{hw}$, it is $t_{remaining}(i+1) = t_{remaining}(i) - M$;
  activation of new processes (if any);
endloop;

Figure 2. The Time Stretching (TS) algorithm for high-level simulation of hw-sw systems.

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 $n_a \frac{\tau_c}{T_c}$ is the overhead due to the context switch of the 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}) verify (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.

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

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.

4. System simulation

As recalled in the introduction, the purpose of this top-level simulation is to speed-up the functional validation of the system specification while introducing new analysis concerning the hw vs sw allocation task. A software implementation has been developed and integrated within the TOSCA hw/sw codesign framework [4]. To enhance the user friendliness two different type of user interaction have been provided:

- *textual*: by producing the display of textual information concerning the evolution of all the processes, the violation of the timing constraints, variable monitoring, ...
- *graphical*: by querying the simulation database with a commercial waveform tracer (SimView™ 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 general, since the processes can be executed more times, the simulator makes available to the user the min/max values, average and variance, for each monitored rate or latency. 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, commercialized by Italtel, consisting in a data-link controller managing sixteen

asynchronous/synchronous data streams. By using a SPARCstation 20 running at 85MHz, we observed an average *simulation ratio* around 18, i.e. it is necessary to spend 18ms of CPU time to simulate 1ms of the system time. Even if the first implementation delivers a throughput already good enough to enable its effective use for developing real-size project, we expect to improve the performance by a factor of two by re-engineering the code. The CAD environment is also able to compute some metrics to evaluate the *quality* of the mixed hw/sw implementation; due to the lack of space this aspect cannot be discussed here, more details on the considered metrics can be found in [4] [8].

5. Conclusions

The paper introduces a modeling strategy and a simulation algorithm to represent the behavior of mixed hw-sw architecture starting by system-level specifications. The approach is particularly valuable since it allows the designer to maintain the analysis at very abstract level, while gathering significant information on hypothetical hw vs sw allocations of the system processes.

The simulation kernel has been encapsulated within the TOSCA codesign environment and interfaced with the software suite computing the evaluation metrics driving the user during the partitioning task. The proposed approach, through the algorithm presented in figure 2, allows simulation of a system behavior to be mapped onto a hw-sw architecture with one CPU, with a characterization very close to the final one obtained performing low-level simulation. Despite this apparent specificity, the approach can be easily extended to cover different target architectures, i.e. multiprocessors systems. In fact, it is simply necessary to modify the determination of the computational overhead both for the sw-bound and communication processes, while the general simulation kernel and graphical interface still remain valid.

Currently our effort is in the direction to make available the backannotation of the results coming from the low-level VHDL-based co-simulation to the upper level simulator here presented. Moreover, a *co-design advisor* it is under develop to implements different hw-sw partitioning algorithms based on the metrics developed.

6. 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] 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.

[3] Balboni A., Fornaciari W., Sciuto D., Co-synthesis and Co-simulation of Control-Dominated Embedded Systems, Journal Design Automation for Embedded Systems, Kluwer Academic Publisher, Norwell, MA, USA, vol.1 n.3, pp.257-289, July, 1996.

[4] 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.

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

[6] Hoare C. A. (1985) "Communicanting Sequential Processes", Prentice Hall, Englewood Cliffs, NJ, 1985.

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

[8] 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.