Uninterpreted Co-Simulation for Performance Evaluation of Hw/Sw Systems

J.P. Calvez, D. Heller, O. Pasquier

IRESTE University of NANTES, La Chantrerie CP 3003 44087 NANTES Cedex 03
FRANCE, Email: jcalvez@ireste.fr

Abstract

Performance modeling and evaluation of embedded hardware/software systems is important to help the CoDesign process. The hardware/software partitioning needs to be evaluated before synthesizing the solution. This paper presents a co-simulation technique based on the use of an uninterpreted model able to accurately represent the behavior of the whole system. The performance model includes two complementary viewpoints: the structural viewpoint which describes the functional structure, the hardware structure, the functional to hardware mapping, and the behavioral viewpoint which specifies the temporal evolution of each function or process. Attributes are added to the graphical model to specify the local properties of all components.

The performance properties of the solution are obtained by simulation with VHDL. Software functions are executed according to the availability of an execution resource which simulates a microprocessor. This technique leads to rapidly obtain a lot of results by modifying appropriate parameters of the model, and so to easily scan the CoDesign space to decide on the best implementation. This modeling and estimation technique is fully integrated in a whole development process based on the MCSE methodology.

1: Introduction

In CoDesign, one major problem concerns the performance evaluation during the design step. Indeed, designers first have to define the appropriate functional architecture and then to find the partitioning and the allocation on the selected hardware. This means that the solution is deduced from the required performance constraints.

First of all, in order to answer correctly the design objective, one needs to consider the whole development life cycle and to base system developments on a complete design model and methodology. The work presented here is based on the use of the MCSE methodology [4] and specifically on the benefits of the functional model. Then, all along the design process, the selected solution has to be verified and evaluated in accordance to functional and non-functional requirements.

In order to avoid the late discovering of performances not met, the objective of the CoDesign method is to establish and maintain a strong link between the two concurrent developments: hardware and software. The two development branches result from the Hw/Sw partitioning. Deciding on an appropriate partition is therefore essential.

In this paper, we describe an efficient technique to evaluate performance properties of embedded Hw/Sw systems in order to correctly decide on partitioning and allocation according to performance constraints. Section 2 presents an important goal the designer is faced with. Section 3 describes the proposed CoDesign process. Section 4 briefly presents the uninterpreted performance model and the meaning of some attributes. Section 5 describes the co-simulation technique we are developing. Through an example, Section 6 explains the use of this model to extract performance properties and decide on the partitioning. Conclusions are drawn in the last section.

2: The partitioning goal in CoDesign

The final quality of systems that designers develop is mostly dependent on the development process. The first step is concerned with customer requirements which are then translated into functional and non-functional specifications. Performance constraints are one important category of non-functional specifications for Hw/Sw systems. From the specifications, designers have to decide on an architecture able to satisfy the application functionalities and performance constraints. The partitioning and the allocation of functionalities onto components are decided on during the CoDesign step [11],[16]. The last steps concern the implementation, the unit tests, the integration of all the parts, the tests and certification of the whole system with its environment.

Partitioning and allocation are strongly dependent on non-functional constraints: performances, timing constraints, cost, time-to-market, etc. One problem is to correctly elicit these requirements with the customer. Another problem we consider here is how to decide on the

0-8186-7243-9/96 $05.00

partitioning. As a matter of fact, designers have to estimate and predict the performances of selected architecture(s) and compare them with the requirements. Later, during the synthesis of the solution which leads to the implementation step, more accurate information is available to refine the performance estimation and, if necessary, correct the design.

Performances qualify the behavior of the system relatively to observation criteria which may be external to the system (response time, throughput, etc.) or internal (utilization of a resource, bus throughput, etc.) [2],[7],[13]. Each kind of performance is called a performance index. Here we are concerned with the dynamic performances of real-time systems which are the most difficult to estimate and satisfy.

The estimation of system performances is usually done by analytical methods or simulation techniques [13]. In order to select the simulation technique for its capability to model transients, two types of models are possible: interpreted and uninterpreted. An *uninterpreted model* is a model for which the behavior is not dependent on the data values. It is the contrary for an interpreted model as it is the case for an algorithm or a state-based diagram. Therefore, an uninterpreted model is a more abstract model or is an abstraction of an interpreted model obtained by removing the data or information values. The effect of these data values are abstracted and replaced by attributes. For example, the attribute *Execution Time* replaces the execution duration of a sequence of statements on a processor, the attributes *Size* and *Id* replace the content of a message. Few performance models and tools exist to evaluate the dynamic performances of any kind of systems [1],[14].

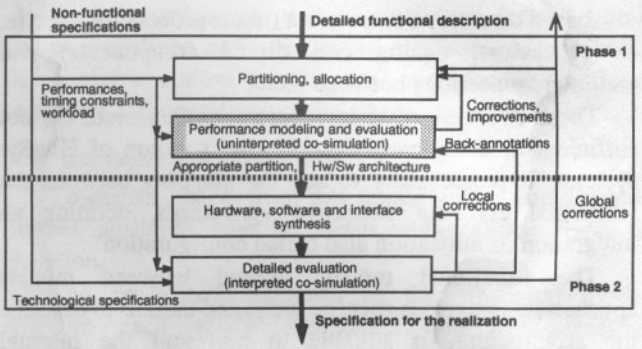
In CoDesign, an accurate estimation of the temporal properties of a solution needs to simulate the hardware part and the software part together. Since a microprocessor is used to run several processes or tasks, its properties and the task-scheduling policy mainly define the global system behavior. The time scale is not the same either: > 1 µs for the software, < 100 ns for the hardware. Therefore a *co-simulation* is necessary.

To help designers during the partitioning phase, we propose to use a performance model which is an uninterpreted model to represent the hardware and software organization and behavior of the solution. Properties are extracted by co-simulation of this model, which means the simulation of the software and the hardware together. This technique is much faster than using an interpreted model and easily allows to study the influence of some specific parameters. Generic architectures may also be studied.

3: Presentation of the method

So as to correctly master the partitioning and allocation in order to find the most appropriate mapping of the functional description onto a hardware architecture, the CoDesign process we propose is depicted in Figure 1. For more details on the global design process, the reader can refer to [4], [9]. In our approach, before partitioning, the designer needs to correctly delimitate the critical parts of the project, to design a functional solution, to specify the performance requirements and the system workload conditions, to define a detailed functional solution including the geographic partitioning constraints and the physical interfaces.

The CoDesign stage is decomposed into two phases, in each one a verification by co-simulation enables to decide on corrections or to continue.



-Figure 1 - The CoDesign with performance estimation.

The partitioning and allocation can be based on various methods: automatic, semi-automatic, interactive [16]. Since the input functional description is conform to the MCSE methodology, in [5],[9] we suggested to follow an interactive coarse-gain partitioning procedure driven by the designer who can easily decide on an appropriate choice for each function.

The uninterpreted performance model presented in the next section is then easy to obtain. The structural model results from the composition of the functional structure and the hardware architecture according to the mapping. The behavioral model of each function is an abstraction of the algorithmic behavior. Attributes and parameters specify the properties of all components. The workload of the system is used to define the context of the simulation. The performance indexes are used for selecting the results to observe.

In the second phase, when an appropriate partition is reached, the functional description, the hardware architecture and the mapping are used to obtain the hardware and software descriptions by synthesis [12]. Both descriptions are used for a final verification by a detailed interpreted co-simulation. A back-annotation of execution times is also possible to enhance the performance model.

This process allows to follow a smooth incremental design path with a better integration of performance mastering. In this way the correction or improvement feedback loop is shorter. As a matter of fact, without the

uninterpreted performance modeling and co-simulation phase, the verification of performance satisfaction is possible only after the complete synthesis of the solution and a detailed co-simulation which needs more time.

4: Presentation of the model

The conceptual model of MCSE [4] includes two views, each corresponding to a specific aspect of the solution:

- the *functional model* (hierarchical and graphical model) describes a system by a set of interacting functional elements (organizational dimension or functional structure) and the behavior of each of them.
- the *executive model* describes the architectural structure based on active components (microprocessors, specific processors, analog and digital components) and interconnections between them.

These two views, when separately considered, are not sufficient to completely describe the solution of Hw/Sw systems. It is necessary to add the *mapping* between the functional and the executive viewpoints, defining an integration or allocation also called configuration.

The functional model, located between models appropriate to express specifications and models to describe the architecture, is suitable to represent the internal organization of a system by explaining all necessary functions and couplings between them according to the problem viewpoint. Designing with this method leads to an internal technology-independent solution. All or part of the description may be implemented either in software or in hardware. Therefore, this model is interesting as a specification input for a Hw/Sw CoDesign method based on a coarse-grain partitioning

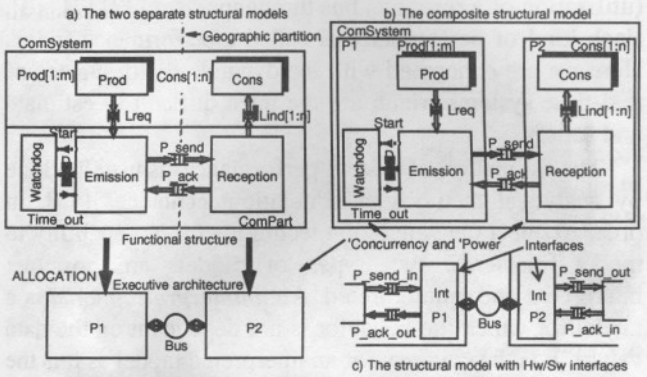
We enhanced this model according to two complementary and orthogonal viewpoints to extend its usefulness to performance modeling:

- the *organizational viewpoint* (structural model) which describes the system by a hierarchical structure including the above functional and executive structures.
- the *behavioral viewpoint* for each function or component, which specifies the set of operations and their total or partial time ordering. This is an uninterpreted model of the function.

In the next sections we briefly introduce the two viewpoints. More information on the performance model and its use can be found in [6], [10]. In [6] this model was used to analyze the performance properties of a real-time video server.

4.1: The structural model

The meaning of the structural model is extended by considering both the functional meaning (function, event, shared variable, port) and the executive meaning (processor, signal, shared memory, communication node). Thus, it is possible to represent both structures - functional and executive - with the same graphical model, and so to describe the complete architectural solution with the partitioning and allocation (functional to executive binding). Figure 2 illustrates the concept. On the left hand side, two structures and the partitioning and allocation are depicted. On the right hand side, only one structure represents the same solution.



-Figure 2 - Structural model for performance modeling.

The example considered here is a simplified communication system *ComSystem* for message transfer between producers *Prod[1:m]* and Consumers *Cons[1:n]*. The function *Emission* has to send each message of *LReq* to its corresponding function *Reception* through the port *P_Send*. To guarantee a correct transfer, each message has to be acknowledged via the port *P_Ack*. *Emission* uses a watchdog function to limit the waiting duration of the acknowledgement.

The executive structure is composed of two processors linked by a node representing a bus. Each processor can be characterized by two attributes: 'Concurrency (the number of functions it can execute simultaneously) and 'Power (relative CPU speed value). The bus can be specified by its concurrency (number of simultaneous accesses), its send and receive times for each message. Here the chosen allocation is simple to understand since it is based on the geographic partition constraint.

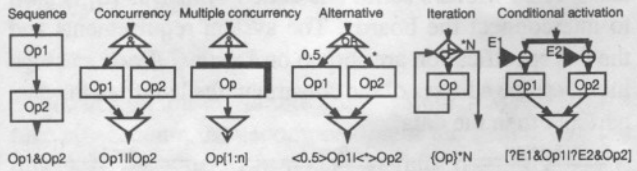
The objective of the performance model (structural viewpoint) on the right hand side is to represent the two structures and the allocation with only one model. Figure 2-b depicts such a composite or combined structural model. Starting from the functional structure, each processor *P1* and *P2* is added as an encapsulation of the set of functions that each processor has to execute. This operation corresponds to a graph restructuration which is called folding: a group of nodes in a graph is selected to form a new node composed of the subnodes previously selected. In this way, *P1* and *P2* have in fact the meaning of a function with nevertheless the two specific properties of a processor:

'Concurrency and 'Power. In this structure the inter-processor link through the node *Bus* is not considered for simplification. The temporal properties of the link are abstracted and integrated into the 2 relations by *P_Send* and *P_Ack*. If the above structural model is considered too abstract, it is possible to keep the *Bus* link. In that case, interfaces (which are functions) between functions and processors must necessarily be added (Figure 2-c).

4.2: The behavioral model

The behavioral viewpoint of an active component is orthogonal to the structural one. A behavioral model, which is hierarchical, graphical and uninterpreted, is described according to the vertical axis representing the temporal evolution and the horizontal axis describing the data or information flow (transactions).

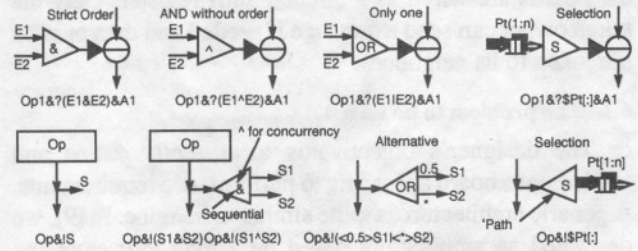
The temporal description is based on five kinds of composition: sequence (&), alternative (|), concurrency (||), iteration ({-}), conditional (guarded) activation ([?-|?-]). Figure 3 gives the graphical notation for each of them (vertical axis). Exclusive or concurrent evolutions are drawn as parallel branches. Complex internal behaviors result from dynamic instanciations of activities and activity refinements. An activity considered elementary is called an operation.



-Figure 3 - Representation of composition rules.

The modeling of performance implies the notion of execution time for operations and exchanges in order to extract the global temporal properties of a system from its local temporal properties. Therefore, an execution time (attribute *'Time*) is added to each operation.

To express interaction rules between vertical branches Temporal dependencies, (i.e. synchronizations, communications) and inputs and outputs are represented horizontally. An activation condition is elaborated from available inputs. The generation of outputs or internal data are actions executed after operations. Composition operators for interactions are represented in Figure 4.



-Figure 4 - Notation for function or activity interdependencies.

The actions concern the generation of an information item or of an event through the outputs of the component including the activity or towards other internal activities. For a shared variable, the action concerns a reading or a writing operation. The Alternative symbol leads to produce only one output. In this case, a rule (attribute) must be specified to define the output concerned: determinism, random. The Sequence symbol defines the order of reception or generation. The Selection symbol allows to specify which input or which output in a set or in a vector is concerned. The attribute *'Path* is used for that purpose. The behavioral model is illustrated by the example given further in Section 6.

An information item or a transaction is defined with a set of predefined and user carried attributes. All attributes are useful to control the temporal evolution of functions, activities and operations of the whole structure.

4.3: Attributes and parameters

To extract results from this uninterpreted performance model, attributes must be added to the above graphical notation.

The predefined attributes of the structural model concern each active component and each relation component. For a function, a processor and even a system, (i.e. an active structural component), we have selected the following attributes:

- *'Power*: floating-point value, (1 by default)
- *'Concurrency*: a positive integer, (1)
- *'Policy*: (PSP, PSD, NPS, TS), (PSP means Preemptive Scheduling based on Priority)
- *'Overhead*: a time, (0)
- *'Priority*: an integer, (1 for the lowest priority)
- *'Deadline*: a time, (0).

These attributes are useful to evaluate the properties of an architecture. The attribute 'Power simulates the power of a computer unit; execution times of all included active elements are scaled by this coefficient. It simulates the clock speed of the computer. The attribute 'Concurrency is useful to simulate a component having a limitation of running resources. With it, it is easy to simulate a monoprocessor or multiprocessor type of component. By changing the attribute 'Policy, it is also easy to experiment with the influence of different policies and compare them. The attribute 'Overhead is useful to simulate the time needed for task context switching. The last two attributes 'Priority and 'Deadline are used to select the most urgent task to run (only one of the two values is used according to 'Policy).

Each kind of relation components is also specified by a set of attributes. Here, due to lack of space, we only give the selected attributes of a port:

- *'Policy*: (Fifo, Priority), (Fifo)
- *'Concurrency*: a positive integer, (1)

- *'Capacity*: max number of messages, ≥0, (1)
- *'Write*: a time, (0)
- *'Read*: a time, (0).

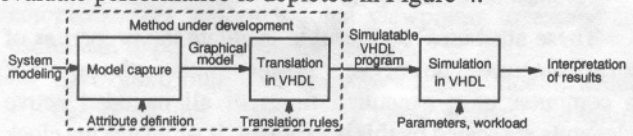
The predefined attributes of the behavioral model are: *'Time* for operation durations, *'Size* for the size of data or information items, *'Path* to specify a path through a selection operator, *'Cond* for a conditional loop, *'Id* for the identification of a function or an activity.

In general, the value of an attribute is dynamic and is defined by any mathematical expression including constant values, parameters, other attributes, the current time, mathematical and probabilistic functions.

The resulting model is an uninterpreted one. Notice also that several models may specify an active component at the same time. This means that during the top-down design process the behavioral model is a specification from which it is easy to deduce an equivalent structural solution.

5: Co-simulation and result extraction

Our performance modeling technique and the corresponding simulation method are an integral part of a set of tools we have been developing as a help to the MCSE methodology. The performance model has to be simulated to be usable for CoDesign as a help to decide on partitioning and allocation. Two techniques are possible: use of a specific simulator developed for the proposed model, translation of the model into a language for which a simulator already exists. In this latter case, the model is translated into an executable description. We are currently considering two techniques: translation into VHDL and then simulation to extract appropriate characteristics, translation into C++ and execution. The process under development to evaluate performance is depicted in Figure 4.



-Figure 5 - Process for performance evaluation.

Designers first have to define the appropriate performance model according to what they want to evaluate. The model is captured with graphical tools and the attributes of all the elements are added. The graphical model is then automatically translated into a simulatable VHDL program according to translation rules. The simulation VHDL model with defined parameters and an appropriate simulation of the workload of the system generates events and data which are interpreted to obtain the results.

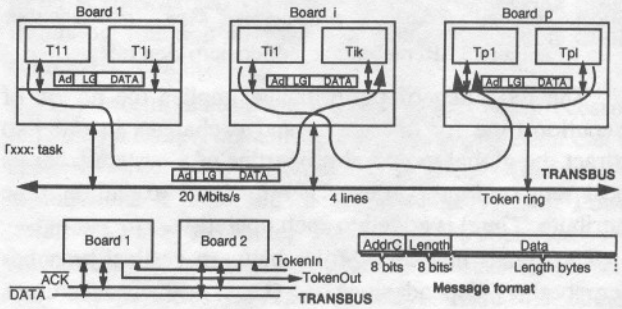
The performance analysis of the event trace leads to estimate the properties of the solution during the design step and to select the best solution and parameters [7].

VHDL is very efficient to describe and simulate concurrent functions and multiple instanciations with generic parameters. The simulation allows to extract various characteristics of architectures to evaluate their costs and performances. High-level descriptions are also very easy to describe and test in the form of uninterpreted models. Generic parameters are an efficient way to specify the behavior of all types of components of the model.

But for hardware/software co-simulation, we have observed some limitations due to the fact that VHDL was conceived to describe circuits rather than systems. Probably the main limitation of VHDL for our goal is the lack of an external process suspension including freezing the process time to simulate a multiple function processor sharing. Further details on the translation rules into VHDL are described in [6], more specifically the execution of several functions onto a limited processing resource.

6: An illustrative example

The case study we have chosen to illustrate our approach is described in [5],[9]. The required goal is to design and prototype a distributed communication system obtained by assembling many similar boards. On each board, producers have to send short messages or packets (256 bytes max.) to consumers located on the same board or on other boards. Producers and consumers are software tasks. A 20 Mbits/s serial bus called TransBus [3] is used to interconnect the boards. The system requirements and the bus specification are shown on Figure 7. Each message includes: the address of the consumer, the length of the data part and then the data.



-Figure 6 - Requirements of the communication system.

The bus access management is based on a hardware token ring. At any moment, only one board must own the token. The token is implemented as a boolean signal and all the boards are wired as a circular shift register. Only the token owner can send a message if needed and then pass on the token to its neighbor.

6.1: The problem to be solved

The designer's objective is to correctly define and implement a board according to performance requirements. A generic architecture is quite simple to imagine. In [9], we described an architecture based on a microprocessor, an FPGA and a shared memory. It is easy here to find that a

strict minimum of hardware is necessary to satisfy the 20 Mbits/s transmission rate and the bit protocol imposed by the Transbus. In the rest of the paper, we suppose the existence of this hardware part to implement the bus interface. It includes a parallel to serial convertor to transmit each byte and the reverse for the reception of each byte.

The problem here is to determine the remainder of the solution. The first step consists in defining the functional solution. This means identifying all processes and relations between them. The next step consists in defining the partitioning and the allocation. But to do so, it is necessary to have quantitative information on the required performances and on the performances estimated according to the selected functional design and generic hardware architecture. To obtain this information, a model of the solution is needed.

Rather than developing a complete interpreted model for both the software part and the hardware part and co-simulate it, we show in this example that it is relatively easy to estimate various performance indexes on different implementations with a generic uninterpreted model and a co-simulation of it. In the next sections, we describe the functional model, the behavioral model, the various results obtained by co-simulating the hardware and the software at a macroscopic level but sufficiently detailed to rapidly observe interesting results.

6.2: The functional model

To design this communicating system, it is necessary to take into account the decomposition of the system into a set of boards and the interconnection bus (the geographical distribution of the application). This task is well done by applying the specification and functional design steps of the MCSE methodology. The result of geographic partitioning and introducing the physical interface is described in Figure 7-a which presents the complete detailed functional solution of each board which satisfies these technological constraints. The transbus is here modelled (abstracted) in Figure 7-b by a vector of events *Token[1:k]* to represent the token ring and a vector of ports *TB[1:k]* to describe the behavior of the message transfer between each pair of boards.

Each message produced is sent by a producer *Prod[i]* to the function *Routing* through its port *Treq[i]*. The address field is used by *Routing* to determine if the designate consumer is local (same board) or distant. For each distant communication, the function *EmissionMess* sends each message from *Lreq* to the addressee board through the port *TB[addressee]*. Since only one board at one and the same time must access the TransBus, *EmissionMess* first has to request the token (event *EmisReq*) and wait for it (event *TokenOk*). When the message sending is finished, the event *EmisEnd* releases the token which is then sent to its neighbor (*Token[i+1 mod K]*). The function *ReceptionMess* receives each message which concerns the board and sends it to the function *Dmux* through the port *Lind*. *Dmux* sends each message to the addressee consumer.

6.3: The behavioral model

The behavior of each function in an uninterpreted form is given in Figure 7-c according to the notation described in Section 4. Attributes are added to the graphical model to specify the complete behavior. To understand the notations,

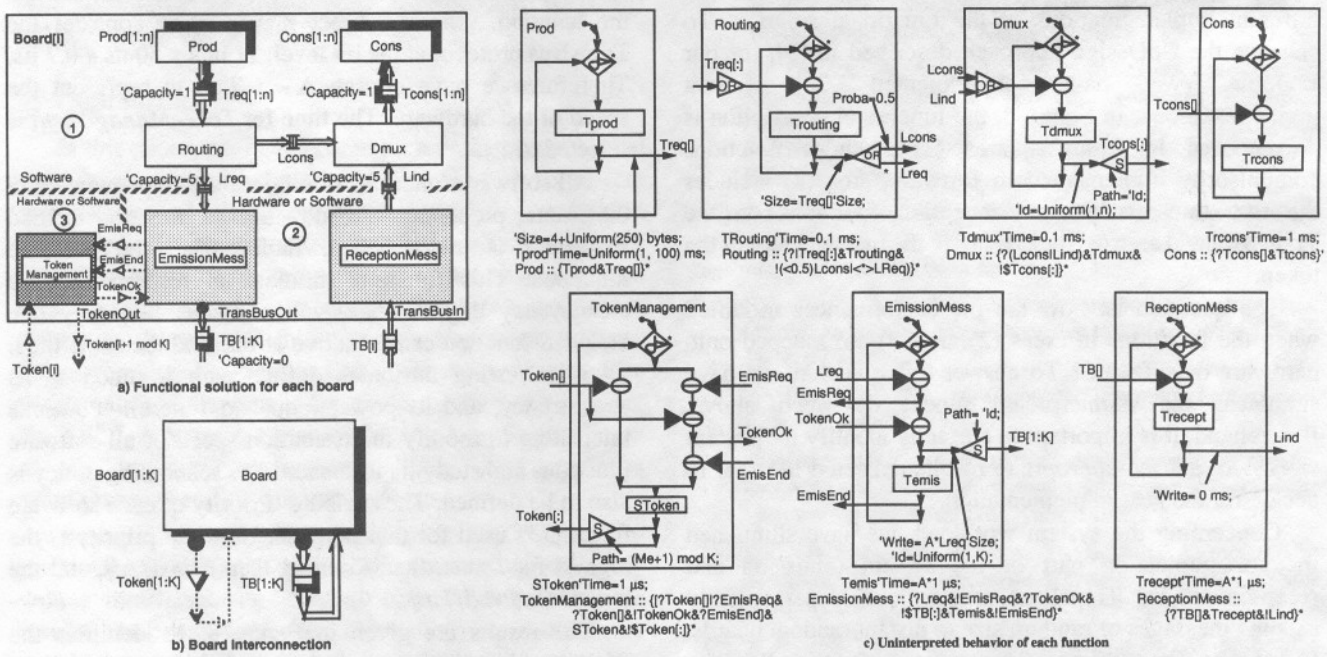


Figure 7- Functional structure for one board and the uninterpreted behavior of each function.

Treq[] means the port of the same index as the producer in the vector *Treq[1:n]*. *Treq[:]* designates the complete vector. This notation is interesting for generic components.

Producers and Consumers are very simple cyclic processes. The size of each produced message is a random value (function *Uniform*). The time interval between two successive messages is defined by the execution time of the operation *Tprod* (attribute 'Time). This is an easy way to specify the system workload for this example.

The function *Routing* receives messages from all ports in *Treq[1:n]* and simulates the routage of a part of them to the port *Lcons* (local messages). The attributes *Proba* and * (which means Else) are used to specify the selected output. Each generated message includes the attribute 'Size of the input message. *Dmux* receives messages from the ports *Lcons* and *Lind* and routes them to the consumer whose identifier is defined by the included attribute 'Id.

The function *TokenManagement* is in charge of the bus allocation toward only one board. *EmissionMess* receives a message from *Lreq* and then asks for the token. The transmission of each message on Transbus is modelled by sending it in the port of *TB[:]* whose index is equal to the attribute 'Id defined as a random index of a board. The transmission time is specified by the attribute 'Write which uses the size coming from the received message and the parameter *A* defining the time to transmit each byte. The function *ReceptionMess* is a cyclic process waiting for each message in its port *TB[]* and then sending it to the port *Lind*.

6.4: Co-simulation and results

The objective of this example is to show that the model described above leads to evaluate the main performances of different implementations of the functional solution. To enhance the CoDesign approach described in [5], for our example, we have experimented 3 different implementations. In Figure 8, the functional description is decomposed in 3 areas: area (1) includes functions compulsorily implemented in software, area (2) includes the two transmission and reception functions on the TransBus, and area (3) includes only the management of the token.

The question is: how are the performances modified when the functions in areas (2) and (3) are mapped onto hardware or software? To answer this question, we have simulated the uninterpreted model described above. Beforehand, it is important to correctly identify the system workload and the appropriate results expected in order to decide for the best implementation.

Concerning the system workload, we have stimulated the communication part of the system (emission and reception on the TransBus) with producers permanently sending messages of random size to distant random boards. In this case, *Tprod'Time* = 0 ms and in the function *Routing*, *Proba* = 0 (no local transmission). A consumer is also supposed to spend at least 1 ms to exploit each input message. The servo-control of producers to consumers is obtained by the capacity of each port (attribute 'Capacity). We have chosen: capacity of *Treq[i]* and of *Tcons[i]* = 1, capacity of *Lreq* and of *Lind* = 5. The correct simulation of TransBus is obtained with *TB[:]'Capacity* = 0, which means a rendez-vous between the sender *EmissionMess* and the receiver *ReceptionMess* connected to the port used.

Concerning the results to evaluate, we consider the following as representative of the efficiency of the communication system:

- the latency of a message from the producer to the consumer,
- the throughput on the TransBus,
- the utilization ratio of the processor running the software on each board.

Because of the random character of behavior of the model, the 3 results are evaluated as the average of all boards and all producers and when the steady state is reached (# 0.1 s observed by simulation).

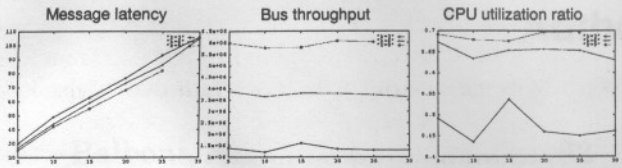
The interest of the co-simulation is to study the influence of different generic parameters of the system. Therefore, we have varied the number of boards (3, 6, 9) and the number of producers and consumers (generic parameter n).

-A- Maximum hardware (areas (2) and (3) in hardware)

To obtain appropriate results, it is necessary to know the time needed to send each byte on TransBus when *EmissionMess* and *ReceptionMess* are implemented in hardware. The value is taken from [5] where we described the solution. Another direct means is to consider the TransBus protocol at the bit level: 11 bits x 50 ns # 0.7 µs. Therefore we have chosen A = 0.7 µs to represent the speed of the hardware. The time for *TokenManagement* is selected to 1 µs.

All software functions of a board are implemented on the same processor. To do that, a function named *Processor* is added which includes all the software functions. This function simulates a resource with a concurrency degree of 1, which means that only one included function can be active at one and the same time. Two interesting attributes define such a function: its concurrency, and its power (equal to 1 here). 'Power is interesting to modify the execution speed of all software functions and study its influence. The scheduling policy is also to be defined. The attribute 'Priority of each software function is used for that purpose. Here the priority is the highest for *Dmux*, then *Routing*, then *Cons[1:n]*, and the lowest for *Prod[1:n]*.

The results are given in Figure 8. K identifies the number of boards.

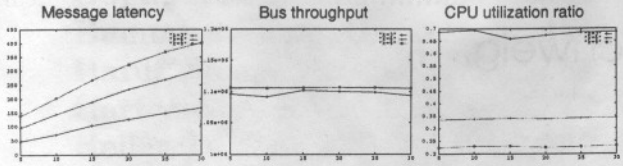


-Figure 8 - Results for the maximum in hardware.

The message throughput on each board is constant and equal to the size average of all the produced messages (129), each being consumed every 1 ms by a consumer. The bus throughput is not dependent on n but on K. The message latency increases with the number of producers and consumers because each CPU is shared by all of them. The CPU utilization rate is relatively low for K=3, because the bus is not properly used.

-B- All functions in software

All the functions are added inside the function *Processor*. The time needed to send each byte on TransBus is now A=7µs. The execution time chosen for the operation *SToken* is 20 µs which is the time needed for a CPU on receiving an interrupt. The results are given in Figure 9.



-Figure 9 - Results for all functions in software.

The bus throughput is now constant and a little lower than the previous case because 20 µs are added between two successive messages. The latency and the CPU utilization rate are similar.

7: Conclusion

In this paper, we have described a performance model and a co-simulation technique to help designers for system partitioning and allocation while developing embedded hardware/software systems. The model is of uninterpreted type, this means that it represents the whole solution at an abstract level but accurate enough to evaluate the system properties. Because of this type, the simulation is faster than a complete hardware/software interpreted model. The performance evaluation is based on a VHDL simulation; the VHDL program is obtained by a systematic translation of the graphical performance model and attributes according to specific translation rules.

A graphical tool and the automatic VHDL program generator is under development. We are also experimenting with a translation into C++ to obtain the performance results. The resulting method and the tool we are currently developing are fully integrated in the complete MCSE system-level methodology.

References

[1] J. Aylor, R. Waxman, B.W. Johnson, R.D. Williams, The integration of performance and functional modeling in VHDL, in "Performance and Fault Modeling with VHDL", J.M. Schoen, Editor, Prentice-Hall, New Jersey, 1992, pp 22-145

[2] R. Bordewisch, W. Föckeler, B. Schwärmer, F-J. Stewing, Non-Fonctional Aspects: System Performance Evaluation, In "Systems Engineering. Principles and Practice of Computer-Based Systems Engineering", Editor B. Thome, John Wiley, 1993, pp 223-271

[3] J.P. Calvez, O. Pasquier, A TRANSputer interconnection BUS for hard real-time systems, Transputer'92 Besançon France IOS Press, May 20-23, 1990, pp 273-283

[4] J.P. Calvez, Embedded Real-time Systems. A specification and Design Methodology, John Wiley, 1993

[5] J.P. Calvez, D. Isidoro, A CoDesign experience with the MCSE methodology, Proceedings of the Third International Workshop on Hardware/Software CoDesign, Grenoble, France, Sept 22-24, 1994, pp 140-147

[6] J.P. Calvez, D. Heller, O. Pasquier, System performance modeling and analysis with VHDL: Benefits and limitations, VHDL-FORUM EUROPE Conference, IRESTE, Nantes, France, April 24-27, 1995

[7] J.P. Calvez, O. Pasquier, Performance Assessment of Embedded Hw/Sw Systems, ICCD'95, International Conference on Computer Design, Austin, Texas, October 2-4 1995

[8] J.P. Calvez, A System Specification Model and Method, Current Issues In Electronic Modeling, Issue #4: Modeling of System Abstraction, Kluwer Academic Publishers, 1996

[9] J.P. Calvez, A CoDesign Case Study with the MCSE Methodology, To appear in the journal "Design Automation of Embedded Systems", Special issue on "Embedded Systems Case Studies", Kluwer Publisher, 1996

[10] J.P. Calvez, A System-level performance model and method, Submitted to "Current Issues In Electronic Modeling", Issue #6: Meta-modeling: Performance, Software and Information Modeling, Kluwer Academic Publishers, 1996

[11] D.D. Gajski, F. Vahid, S. Narayan, J. Gong, Specification and Design of Embedded Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1994

[12] R.K. Gupta, G. De Micheli, Hardware-Software cosynthesis for digital systems, IEEE Design & test of computers, Sept. 1993, pp 29-41

[13] R. Jain, The Art of Computer Systems Performance Analysis, John Wiley, 1991

[14] SES/workbench: a multilevel design environment for modeling and evaluation of complex systems, Scientific and Engineering Software, Inc., August 1989

[15] D.E. Thomas, J.K. Adams, H. Schmit, A model and methodology for Hardware-Software Codesign, IEEE Design & test of computers, Sept. 1993, pp 6-15

[16] W.H. Wolf, Hardware-software Co-Design of embedded systems, Proceedings of the IEEE, Vol 82, No 7, July 1994, pp 967-989