Heterogeneous Co-Simulation of Networked Embedded Systems

Franco Fummi† Stefano Martini ‡ Giovanni Perbellini ‡
Massimo Poncino † Fabio Ricciato * Maura Turolla *

† Università di Verona Verona, ITALY
‡ Embedded Systems Design Center Verona, ITALY
*Telecom Italia Lab Torino, ITALY

Abstract

Networked embedded systems pose several challenges in the modeling, simulation, and design domains. The presence of the network, in particular, makes an already critical task such as HW/SW co-simulation even more complex, since a three-way (HW/SW/network) co-simulation and co-design capability is required. Modeling of networks and their interaction with hardware and software is thus key for an effective design methodology at early stages of the design flow.

In this work, we present a HW/SW/network co-simulation and co-design methodology, based on the integration of heterogeneous simulation environments such as SystemC and NS (Network Simulator). This methodology has been successfully applied to the design of a system-on-chip performing the fast path of IPv4 routing, allowing to explore different HW/SW allocation for different network configurations.

1. Introduction

The always increasing complexity in embedded systems makes the design issues more and more challenging. In fact, in addition to the complexity of designing the hardware and the software components, the embedded system engineer is now faced with the problem of interfacing the embedded system with a networked world in which the design it will operate.

In other terms, embedded systems are no more viewed as isolated monads; rather, they interact among them and with the environment, exchanging data and messages in such a way to set up a more or less ubiquitous network. Accounting for the presence of a network is essential: First, it can be used as an additional "variable" to allows for further optimizations of the design. Second, it allows to verify the functionality of the embedded system within its actual context. For efficiency reasons, the design of the embedded system, of the network protocols, and of the interacting elements should be carried out concurrently, starting from the system level specification.

This scenario turns the conventional hardware-software co-design problem into a more complex one, where the network is an explicit dimension, that we call "hardware/software/network co-design"; the network becomes then one variable in the design of an embedded system as hardware and software are. Figure 1 shows what are the steps of the "network synthesis", and how they match their corresponding tasks in the HW and SW domain.

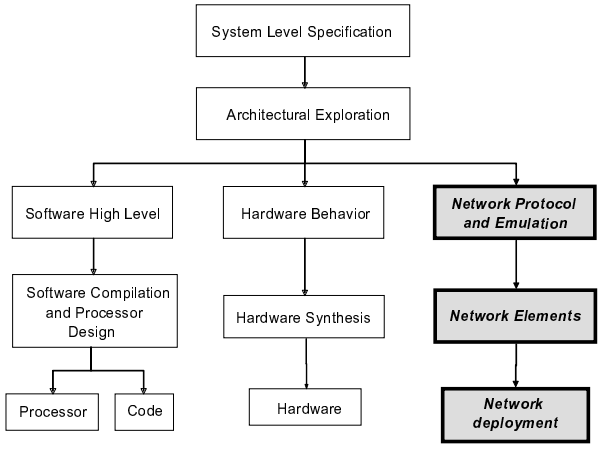

Figure 1. Network Synthesis vs. HW and SW Synthesis.

While HW/SW co-design and its many issues (partitioning choices, communication issues and synthesis, co-modeling, co-simulation, and co-verification) has been thoroughly studied [1], the HW/SW/network co-design is quite a new topic. Although it has problems simi-

1530-1591/04 $20.00 (c) 2004 IEEE

lar to HW/SW, it also has to cope with the difficulty coming from the different "cultures" of the designers coming from the HW/SW and network domain.
The network dimension gives to the the embedded system designer an additional degree of freedom in choosing how to implement a certain action/function. Given the different abstraction level of the HW/SW and network domains, we envision a *hierarchical* partitioning step in the HW/SW/network co-design; that is, there is a first-level partitioning between HW/SW and network, and an eventual, lower-level conventional HW/SW partitioning.
In other terms, the HW/SW/network partitioning answers the question whether the function should be realized by a board containing the embedded system, or it can be delegated to networks elements; HW/SW partitioning, conversely, decides whether the function should be performed by a processor or by dedicated hardware. Any a priori definition of partitions can lead to sub-optimal designs.
The architectural exploration involves thus also the network elements and the topology, with an increase of complexity in the solution space. This complexity is reflected into the modeling and simulation steps that should help to find the optimal partition among hardware or software blocks and network elements.
On the other hand, the great complexity of networking software and hardware, added with the variability of the network workloads, make the overall verification quite complex. Around the embedded system we need a refined model of the network environment in order to make the software and hardware to act on that network to verify and, possibly, validate the involved algorithms and architectures. The idea is to model these networked embedded systems in a hybrid style: part of the system is modeled in a conventional, system-level, hardware-oriented environment, while the network part is described with a proper network modeling tool. The former part has a straightforward path to HW/SW implementation, while the latter offers a great availability of protocols and network structure models.
In this paper, we discuss a possible implementation of HW/SW/network co-design, that uses a HW/SW/network co-simulation environment that integrates HW (specifically, SystemC), SW (specifically, Instruction Set Simulators), and network (specifically, NS-2) simulation in a way that allows for timing-accurate simulation (between HW and network), and allows to quickly evaluate alternative partitioning solutions. The methodology has been applied for the design of a system-on-chip implementing a embedded packet engine.

2. Enabling Technologies

The heterogeneous co-simulation of networked embedded systems is an essential feature for HW/SW/network co-design, since it represents the base mechanism that allows to do simulation, validation and profiling. Although homogeneous co-simulation is more efficient, in this context heterogeneity is mandatory and not really a choice. So far, the literature about heterogeneous co-simulation, has not addressed the issue of integrating the simulation of HW, SW and a network. Many two-tier co-simulation approaches do exist, however, such as solutions that integrate network and HW simulation or HW and SW simulation.
The methodological effort of this work has been that of taking such two-tier approaches and integrating them into a three-tier scheme. This integrated co-simulation environment has been realized by integrating well-known simulation engines (see Figure 2):

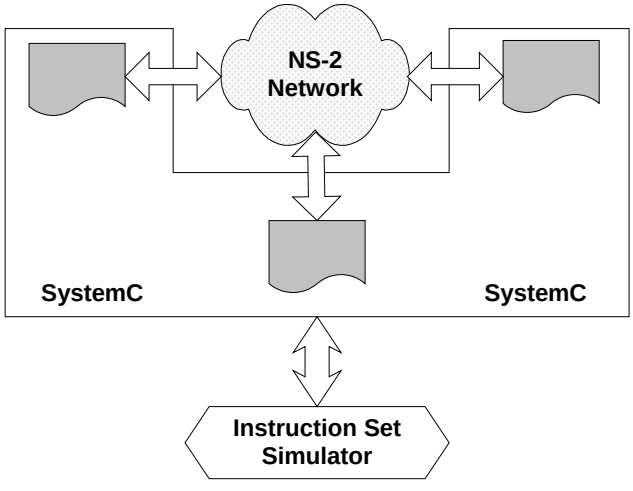

Figure 2. Integration of Simulation Environments.

the network side is modeled by a Network Simulator called NS-2 [3]; NS-2 is a discrete event simulator for network protocols and algorithm analysis. NS-2 is based on two programming languages: C++ and OTcl; the former is used to create simulation models of network components, while the latter is used to describe the simulation scenarios, choosing the components to simulate and setting their parameters. NS-2 comes with a number of available protocol models, and its open-source model guarantees the early availability also of even the latest protocols and/or algorithms.
The HW/SW domain can be modeled by C/C++ code, modeling the behavior of the device, properly wrapped with SystemC [9] code. In this approach the overall device (HW and SW) is modeled as a set of SystemC modules linked together. SystemC is a C++ class library that can be used to create models of a system at different abstraction levels, from the system-level to the cycle-accurate one. The library provides the constructs required to model system architectures including hardware timing, concurrency and re-

active behavior that are missing in standard C++. Therefore, the hardware part is modeled with SystemC (at any level of abstraction) and the software is a C/C++ program executed by a real CPU on a board, or by an Instruction Set Simulator (ISS), which represents a model of the processor.
The three simulation environments (NS-2, SystemC and ISS/board) are integrated in a unique heterogeneous co-simulation framework. Timing-accurate co-simulation is realized by using two existing methodologies:

- NS-2/SystemC integration is achieved by adopting the solution proposed in [2]. It consists of an efficient co-simulation scheme based on messages exchanged between the two simulators, so as to establish a timing-accurate communication.
- ISS/SystemC integration has been implemented by using two methodologies presented in [10]; the first approach provides a co-simulation scheme where the communication between the simulators is embedded into the SystemC kernel; in the second scheme "calls" to the SystemC hardware functionalities are mapped to device drivers calls of the operating system running on the ISS.

3. Case Study

The integrated, three-tier schemes has been applied to the design of a system-on-chip that implements the fast path of IPv4 routing. The chip should support the checking and the classification of the packets, as well as support quality of service. The conceptual scheme of the router is shown in Figure 3. The model follows a modular approach [5], by isolating the different functionalities and connecting them in a data-path oriented paradigm. The networking application was split into independent modules that, through specific adaptation libraries, can be mapped on different platforms or processor architectures.
That allows, on one hand, to have flexible software that can be easily tuned to fit the different kinds of configurations that the router application can have; on the other side, it allows to follow the required changes of the programming model and of the architecture during the development of the design.
The ingress part is formed by a line card with two interfaces connected to a switch fabric, the egress forwards packet to two output interfaces. Figure 3 shows a configuration of the router fast-path ingress side with Virtual Output Queuing. It is composed by:

- *CheckIPv4*: module to validate the packet and modify the time-to-live field [7];
- *LookupIPv4*: this module performs the lookup of the IP address defining the interface which the packet should be forwarded to. It is based on a LCtrie tree structure with the implementation given in [6];

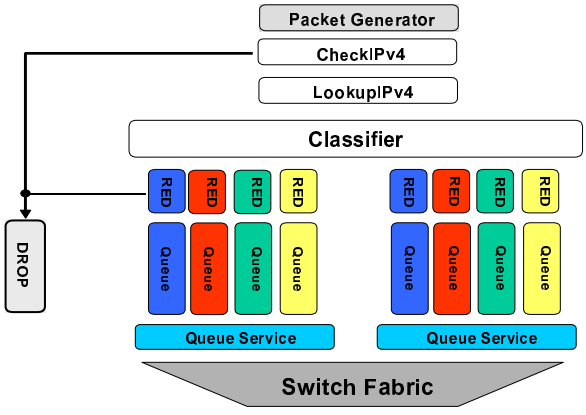

Figure 3. Case study: IPv4 router.

- *Classifier*: this module classifies the packet in order to perform a QoS based queuing. It can perform different algorithms like linear, hierarchical trie, RFC [8]. In the considered test case it looks up just for the ToS field;
- *Random Early Discard - RED*: it performs the RED algorithm to prevent the queue saturation [6];
- *Queue*: it manages the queuing of the packets and the management of the packet memory;
- *Queue Service*: it can serve the queue using different algorithm like Round Robin, WRR, MWDRR, and WDRR. In the considered case the WRR was used.

The choice of modularizing the application was suggested by the need of changing the partitioning of the application. The decisions space includes the following options:

- What algorithms and protocols should be used, considering the impact at network level;
- What parts of the design should be implemented with dedicated hardware;
- What parts of the design should be implemented in software;
- What parts of the design should be included into the system-on-chip and what should be outside.

In addition, we should decide which kind of instruction set, (and thus, of processor), should be used. Due to the complexity of the design, the verification phase was also a key factor in the design success.
The partitioning problem has two dimensions:

- A *model point of view*: We need to decide the most appropriate environment to be used for each functionality of the router at the different abstraction layers;

- An *implementation point of view*: We need to split the functionality among HW, SW and network.

The methodology for the partitioning is carried out in two phases, corresponding to the two levels of the hierarchy mentioned in Section 1. The first one implies both the model and the implementation point of view, the second one only the implementation point of view.

3.1. First-Level Partitioning

As already mentioned, the availability of a co-design framework can be used to describe the different parts of the router with the most appropriate environment. The first partitioning should be done during the modeling phase. The partition was driven by the following factors:

- *Level of abstraction at which the module will be described at the end of the modeling phase*: in other words, we need to split the functionalities that should not be refined from the functionalities that can lead to HW or SW implementation;
- *Availability of the identified functionalities in the tools*: try to reuse the functionalities already present in the tool libraries;
- *Communication pattern between the various functionalities*: avoid, at least in a first stage, to have functionalities with high interaction in two different tools.

Therefore, the functionalities described in NS-2 are defined by the subset given by the intersection of (i) the set of functionality present in NS-2 and (ii) those that do not require refinement at lower levels; The functionalities simulated with GDB will be determined at lower abstraction levels by the usual HW-SW partition. The functionalities modeled in SystemC will be those not present in NS-2 library (at the system level), and and those to be refined till arriving to HW synthesis (at the lower level).
It should be noticed that such partitioning can be carried out in several phases. With the refinements of the design we can move the some functionalities from one domain to another domain.
In our case study, the starting point was a simple router described in NS-2. After the first functional definition and the module division, the innovative router algorithms, such as the lookup algorithm, were described in ANSI C. Than, we inserted the LookupIPv4 ANSI C code in SystemC module, as shown in Figure 4.
This module implements an algorithm not available in NS-2 router. It should be noticed that in our case it was quite intuitive what functionalities should map onto hardware and which ones mapped on the the network, so the choice between a SystemC or NS-2 description is driven merely by the modeling opportunity.

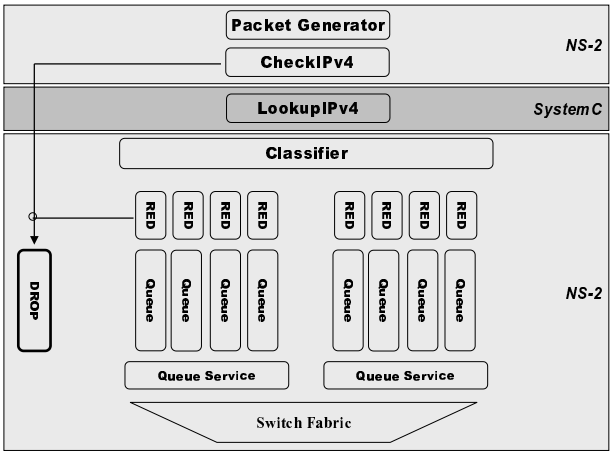

Figure 4. First Partitioning.

The first algorithmic exploration was done by connecting the SystemC module with NS-2 structure, using the paradigm described in Section 2.
A first system-level analysis was done by gathering information on the behavior of the algorithms: either those present in NS-2 either the new ones. This first model allows the tuning of the algorithm parameters with respect to the network. For examples, the characteristic of the RED algorithm, the length of the queues, the lookup search algorithm, the weights of the WRR algorithm and data structures were determined in this way.
In this first stage there is no notion of architecture. The only partitioning regards the functionalities that are described in SystemC and those described in NS-2, and corresponds to the first level of the partitioning hierarchy (HW/SW vs. network). We then refined the model by porting the CheckIP module into SystemC, as shown in Figure 5.

3.2. Second-Level Partitioning

The second partitioning phase concerns the allocation of tasks to either HW (i.e., synthesizable SystemC) or SW (i.e., C++ programs).
This phase is driven by the profiling information extracted by simulation; profiling gives a quantitative evaluation of the computational weight of the different blocks, and has been done as follows. The ANSI C blocks were first compiled on different instruction set. Then, using the described methodology, the code was simulated with the debugger using NS-2 and SystemC as testbench for the software.

4. Partitioning Evaluation

The partitioning evaluation, for the case study described in Section 3, is based on the profiling results shown in Figure 6.

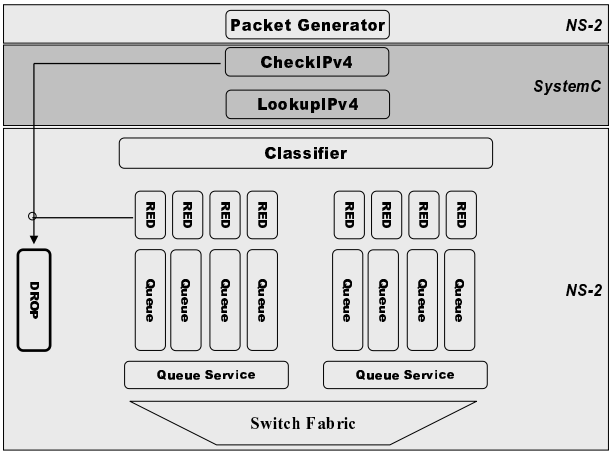

Figure 5. Second Partitioning.

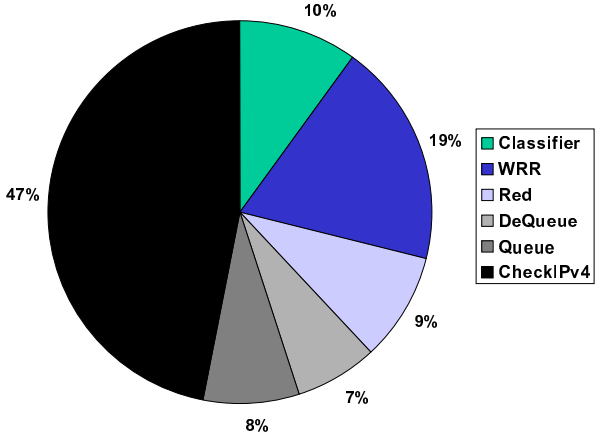

Figure 6. Profiling Breakdown.

After profiling, the most computationally expensive parts (namely, the checkup block) were refined in SystemC in order to arrive to a synthesizable description. Modules that are left implemented in software are simulated using the debugger. The cross-target feature of the GNU debugger allowed us to prove different instruction sets for the addressed algorithms.

After the refinement and optimization steps, the whole system was tested with different network configurations. The tool allows to easily change the characteristic of the traffic, the number of the involved nodes, the link bandwidth and the topology, thanks to the features of NS-2. After the choice of the processor, the last step was to connect the debugger to a real board in order to have more detailed feedbacks in terms of the execution speed and of the synchronization with the rest of the design.

Thanks to the described integrated tool, the refinement step of the hardware, the optimization of the code and the network statistic analysis could all be performed in parallel. The integrated environment allowed us to avoid the overhead of the test benches and of the simulation patterns using. The results of the profiling leads to the partitioning scenario shown in Figure 7.

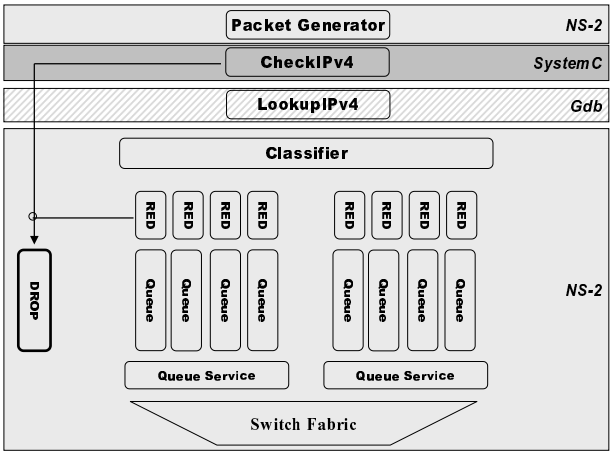

Figure 7. Partitioning for the Validation of the Case Study.

The router, modeled with SystemC and an ANSI C program, receives packets from the network, modeled with NS-2. The packets are then routed to the proper destination.

The co-simulation performances for this scenario are shown in Table 1. Column *Sim. Time* reports the simulated time, in milliseconds, while the remaining three columns show the times for the SystemC/ISS simulation, the NS simulation, and the total simulation time. We can notice how the simulation is quite efficient (the overhead is about 13%), thus allowing fast exploration of design alternatives.

Sim. time [ms]	*SystemC/ISS [ms]*	*NS-2 [ms]*	*Total [ms]*
1000	1093	43	1136
10000	10925	429	11354
100000	109208	4288	113496

Table 1. CPU Times for Different Simulated Times of the Final Partitioning.

5. Conclusions

In this paper we presented the hardware/software/network co-design paradigm as the methodology to be applied during the project of an embedded system with high networking capabilities. The methodology implies the introduction

of additional partition steps in the usual HW-SW design approach. The partition is carried out in a hierarchical fashion, by first allocating the functionalities of the application between network elements and embedded system, and eventually assigning the latter functionalities to either HW or SW. The methodology was applied to the design of the fast-path of a router, allowing to explore different HW/SW allocation for different network configurations. Performance results clearly demonstrate the feasibility of the proposed approach to the design of networked embedded systems.

References

[1] *Hardware/Software Co-design*, Kluwer Academic Publishers, 1996.

[2] F. Fummi, P. Gallo, S. Martini, G. Perbellini, M. Poncino, F. Ricciato, "A Timing Accurate Modeling and Simulation Environment for Networked Systems". *DAC-40: 40th Design Automation Conference*, Anaheim, CA, June 2003.

[3] L. Breslau em et al. "Advances in Network Simulation", *IEEE Computer*, pp. 59–67, May 2000.

[4] Synopsys Inc., *SystemC Users Guide*, version 2.01, 2003.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, "The Click Modular Router", *ACM Transactions on Computer Systems*, Vol. 18, No. 3, pp. 263–277, August 2000.

[6] S. Nilsson, G. Karlsson, "Fast Address Look-Up for Internet Routers," *ICBC'97: International Conference of Broadband Communications*, 1997.

[7] G. W. Wrigth, W. R. Stevens, *TCP/IP Illustrated, Vol. 2*, Addison-Wesley, 1995.

[8] P. Gupta, N. McKeown, "Algorithms for packet classification," *IEEE Network*, Vol. 15, No. 2, pp. 24-32, 2001.

[9] Synopsys Inc., *SystemC User's Guide*, version 2.1, 2002.

[10] F. Fummi, S. Martini, G. Perbellini, M. Poncino, "Native ISS-SystemC Integration for the Co-Simulation of Multi-Processor SoC". *Technical Report*, University of Verona, Verona, June 2003.