HW/SW Co-Simulator for Embedded Heterogeneous Parallel Systems

D. Ciambrone[1], V. Muttillo[1], G. Valente[1], L. Pomante[1]
[1]Università Degli Studi Dell'Aquila - Center of Excellence DEWS, L'Aquila, Italy
{daniele.ciambrone}@student.univaq.it, {vittoriano.muttillo, giacomo.valente}@graduate.univaq.it, {luigi.pomante}@univaq.it

I. Introduction

The growing complexity of nowadays embedded digital systems, especially if based on modern *System-on-Chip* (SoC) adopting explicit heterogeneous parallel architectures [7], and their reduced time-to-market has radically changed the common design methodologies. Traditional design techniques, based on independent design of HW/SW components are no longer sufficient to support the integration of subparts of such SoCs. Here, HW/SW co-design methodologies, where designers can easily check system-level constraints satisfaction and evaluate cost/performance trade-off for different architectural solutions, are of renovated relevance.

These kinds of methodologies are able to lead the system-level analysis by means of several models, metrics and tools, supporting the designer in all those activities that are normally entrusted only to his experience. In particular, HW/SW co-simulation tools cover a very important role in a HW/SW co-design flow, because they allow a fast and correct analysis of the system properties, and to realize a virtual system prototype. In such a context, this work presents a HW/SW co-simulator to be integrated into an ESL HW/SW co-design methodology targeting embedded heterogeneous parallel systems [8].

II. State of the art

In the recent years, with the advent of *Electronic Design Automation (EDA)*, we have seen a push towards the development of the so-called *Electronic System-Level (ESL)* tools, able to span the complete design space across hardware and software boundaries.

CoFluent Studio [1] by Intel is a modeling and simulation environment for early high-level design space exploration. As a graphical frontend for SystemC, it allows capturing of application functionality, system architecture and their mapping. Application models are specified as networks of timed processes, communicating through high-level message-passing channels, queues, events and shared variables. CoFluent studio can generate a SystemC TLM of the resulting architecture for simulation and virtual prototyping.

Among academic simulators, it is possible to find *eSSYn* [2], developed at University of Cantabria. eSSYn is a software synthesis tool for embedded systems. The system model is made of three sub-models, following the *Y* structure, quite common in current design methodologies. The two branches of the *Y* are the separate *Platform Independent Model* (PIM) for SW application on one side and the *Platform Description Model* (PDM) on the other. Both models are connected by the *Platform Specific Model* (PSM) that defines the mapping of SW into HW. In order to use eSSYn, system designers need to provide a software component-based model of the application, a model for the hardware platform to specify the available resources, and a mapping of software components and cores. Then, eSSYn will generate all required code and system calls implementing communications among software components, all required *makefiles* for compilation and executable files ready to be uploaded to the HW platform. Furthermore, it provides a simulation environment called VIPPE.

All these simulators are placed within a framework of HW/SW co-design to perform functional and timing simulations by using SystemC as HW/SW description language. The proposed co-simulator, also if still based on SystemC, presents some important differences. First of all, the system behavior modeling is based on a CSP-like (*Communicating Sequential Processes* [3]) *Model of Computation* (MoC), from which is then generated the simulated SystemC code. This allows the designer to perform the modeling activity focusing on a straightforward and well-known MoC that can be later further exploited also to perform further analysis on the model. Second, all commercial and academic simulators are based on a HW architecture, bounded to the designers choices in the initial steps of the co-design flow, while the proposed simulator is designed to interact (during *Design Space Exploration* step) with another tool that is able to automatically define a HW architecture and a mapping. In other words, the whole approach does not follow the classical *Y* structure, but a linear structure where the HW platform is automatically defined depending on behavioral, timing and architectural constraints.

III. Reference ESL HW/SW co-design flow

The reference ESL HW/SW co-design flow is described in [4]. The entry point of the flow consists of a *System Behavior Model (SBM)* based on a CSP-like MoC and supported by the HW/SW description language *SystemC*. Passing from a computational model to a functional model, there is the first step of the level flow, called *Functional Simulation*. This step takes in input all processes and channels composing the system, to verify the correctness of SBM through functional simulation.

In the following steps, the flow is supported by a *Technology Library*, which can be considered a generic "database" that provides the characterization of all the technologies (i.e. processors, memories, interconnections) available to build the target HW system.

The next step is the *Co-Analysis&Co-Estimation*. During Co-Analysis, the proposed co-simulator is exploited to analyze the system and to evaluate different metrics: *affinity* and *concurrency*. The first represents how much a process is suitable to be executed on a specific class of processor [5]. The

second one concerns both processes and channels to define how much each one could be concurrent respectively with the others. Co-Estimation is in charge to estimate *load*, *size* and *bandwidth*. *Load* represents processor utilization percentage that each process would impose to each processor when implemented in SW. *Size* represents the number of bytes in RAM and ROM needed to store data and instructions for each process implemented in SW processor. For hardware implementations, it is the number of mm^2 (or *Geq*, *LUT*, *LB*, etc.) needed to realize processing, memory and connection elements. *Bandwidth* represents the number of bits sent/received over each channel in a time specified by the designer (i.e. *Time to Completion*). After that, it is possible to enter into the *Design Space Exploration* step, which is characterized by 2 activities: *HW/SW partitioning, mapping and architecture definition* and *Timing Co-simulation*. The first one is responsible to define the HW architecture of the target system, partitioning and mapping of processes on available processors, and mapping of links. The first activity defines all necessary inputs needed to the timing co-simulator to check if timing constraints are satisfied. The next section will provide more details about the proposed co-simulator.

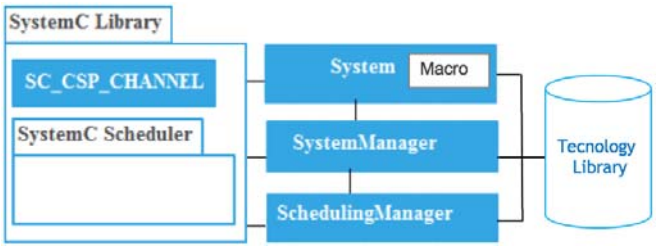

Fig. 1. SystemC-based Co-Simulator Architecture

IV. HW/SW Co-Simulator

The main goal of the present work concerns the design and the implementation of a HW/SW co-simulator. The first step has been to define the SW architecture, as represented in the Fig. 1. The large component on the left is the SystemC [6] Library, which embeds a *SystemC Scheduler* and has been extended with a new SystemC channel to model also CSP channel semantic. In fact, SC_CSP_CHANNEL has been developed according to properties of CSP model and SystemC. It inherits from a SystemC primitive channel and implements a full-handshake policy with blocking *read()* and *write()* methods.

This component works with *System*, *SystemManager* and *SchedulingManager* components, supported by a *Technology Library*. System represents the SBM. It is instrumented by means of some macros in order to take into account timing and scheduling effects. SystemManager defines details and features of the target system that needs to be simulated. It is a C++ class, responsible for generating instances of processors, processes and interconnection links.

The class SchedulingManager probably represents the central element in the simulator. This block is responsible to realize processes scheduling and implement different scheduling policies. So, the key elements of the proposed co-simulator are the instrumentation of code by macros and scheduling policies. The first are used to support a mechanism of full handshake among the SchedulerManager and the processes to allow the desired scheduling of processes. This mechanism represents an additional level of scheduling to the one of SystemC Kernel, which is execution-driven without preemption (cooperative multi-tasking simulation environment). On the base of such a mechanism, it has been possible to implement different scheduling policies (e.g. *Round Robin*, *Fixed Priority*, *etc.*).

The simultaneous use of macro and scheduling policies ensures the possibility to simulate all the possible process-processor mapping combinations. Moreover, to take into account the scheduling overhead, simulation time is charged by a fixed constant time related to the use of the scheduler and an additional fixed one (defined by the designers for each possible processor) in case of context switch. It is worth noting that processes implemented directly in HW are not subject to scheduling issues and overhead. In the end, it is possible to execute HW/SW co-simulations to evaluate timing constraints satisfactions for different reference applications.

Conclusions

This work has presented a SystemC-based HW/SW co-simulator for Embedded Heterogeneous Parallel Systems. The next steps will be to fully integrate and validate it in the context of a full working ESL HW/SW co-design flow [8].

Acknowledgments

This work has been partially supported by the ECSEL RIA 2016 MegaM@Rt2 and AQUAS projects.

References

[1] CoFluent Design & CoFluent Studio, http://www.cofluentdesign.com/, Accessed 2017-04-29.

[2] H. Posadas, P. Penil, A. Nicol as, and E. Villar. Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics. Journal of Systems Architecture, 61(8):341360, 2015.

[3] C. A. R. Hoare, Communicating sequential processes, Springer New York, pp. 413-443, 1978.

[4] L. Pomante, P. Serri. “SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems”. International Journal of Information Systems, 2014.

[5] D. Sciuto, F. Salice, W. Fornaciari, C. Brandolese, Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC. IEEE Transactions on Computers, vol. 55, no. 5, May 2006.

[6] SystemC, http://www.systemc.org, Accessed 2017-04-29.

[7] https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[8] https://www.hepsycode.com