Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 729786, 13 pages
doi:10.1155/2012/729786

Research Article

Efficient Execution of Networked MPSoC Models by Exploiting Multiple Platform Levels

Christoph Roth, Joachim Meyer, Michael Rückauer, Oliver Sander, and Jürgen Becker

Karlsruhe Institute of Technology (KIT), Institute for Information Processing Technology (ITIV), 76131 Karlsruhe, Germany

Correspondence should be addressed to Christoph Roth, christoph.roth@kit.edu

Received 21 February 2012; Revised 3 June 2012; Accepted 21 July 2012

Academic Editor: Massimo Conti

Copyright © 2012 Christoph Roth et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Novel embedded applications are characterized by increasing requirements on processing performance as well as the demand for communication between several or many devices. Networked Multiprocessor System-on-Chips (MPSoCs) are a possible solution to cope with this increasing complexity. Such systems require a detailed exploration on both architectures and system design. An approach that allows investigating interdependencies between system and network domain is the cooperative execution of system design tools with a network simulator. Within previous work, synchronization mechanisms have been developed for parallel system simulation and system/network co-simulation using the high level architecture (HLA). Within this contribution, a methodology is presented that extends previous work with further building blocks towards a construction kit for system/network co-simulation. The methodology facilitates flexible assembly of components and adaptation to the specific needs of use cases in terms of performance and accuracy. Underlying concepts and made extensions are discussed in detail. Benefits are substantiated by means of various benchmarks.

1. Introduction

Today, two major trends can be observed in the embedded domain. (1) Multiprocessor System-on-Chips (MPSoCs) become more and more popular for embedded systems since functionality is evermore integrated directly into a single device. Besides that, this trend towards multicore is driven by the need to optimize performance per watt which results in an increase of parallelism instead of the clock frequency. (2) Applications of embedded systems evermore rely on communication between several or many devices. When considering, for example, the vision of the so-called Cyber-Physical Systems (CPS), this trend will continue or rather become much more intense in future [1]. CPS are characterized by their high adaptability and the capability of very tight coupled interaction among each other as well as the environment. In many imaginable scenarios, such networked embedded devices must meet hard requirements like high performance and low power consumption due to their autonomy in the field of application.

Due to the high degree of parallelism provided by networked MPSoCs system design, verification and validation become increasingly complex. Usually, system development starts on higher abstraction levels. The system is then iteratively refined until reaching the register transfer level (RTL) implementation. During each refinement step, often simulation makes a significant contribution to verify and validate the current system architecture. With increasing model fidelity such simulations can become an extremely time-consuming task.

We believe, simulation of multiple networked devices on multiple abstraction levels is essential for a comprehensive evaluation of networked MPSoCs since (1) the dynamic impact of network communication on alternative system configurations and vice versa needs to be considered, (2) a network-wide optimization of single nodes regarding size, power consumption, performance, and so forth demands for detailed simulations of several nodes in parallel, (3) a step by step refinement process including reuse of intellectual property (IP) cores directly induces simulation at various

abstraction levels, and (4) verification and validation of the system with respect to possible network and traffic conditions is important through all steps of system design, even in late phases, when the system model approaches the final implementation. In order to meet these requirements, new simulation tools and methodologies are necessary which support the process of system design by flexibly combining advantages of both system and network domain and by having special regard on simulation performance at the same time.

In this work, a methodology is presented which facilitates efficient execution of system/network co-simulation on different abstraction levels. Main part of the methodology is a construction-kit-like simulation platform that is divided into several platform levels. This approach allows flexibly interconnecting system domain tools (e.g., OSCI SystemC kernel or Modelsim [2]) with network domain tools (e.g., OMNET++ [3] or ns-3 [4]) and exploiting different hardware components like workstations and FPGA-based rapid prototyping systems for efficient co-simulation and co-emulation. In previous work, parallel simulation of a single MPSoC [5] as well as synchronization algorithms for system/network co-simulation [6] has been considered for applicability with a high-level-architecture- (HLA-) based simulation backbone. In contrast, key contributions of this work are as follows.

(i) Introduction of the co-simulation methodology.

(ii) Embedding of a realistic MPSoC model called HeMPS [7] into the methodology by extending it with wireless networking capabilities through a virtual network interface.

(iii) Extension of the platform of [6] towards a multilevel construction kit by equipping it with a co-emulation interface for fast execution of detailed RTL-based system models.

(iv) Benchmarking and functional verification of the simulation platform by means of the newly integrated HeMPS system using different abstraction levels.

The remainder of this paper is organized as follows. Section 2 summarizes fundamentals and presents related works in the fields of parallel discrete event simulation and network/system modeling and design as well as combined approaches. The proposed simulation methodology is presented in Section 3. Extensions on targeted simulation models as well as the implementation of the simulation platform are described in Sections 4 and 5. In Section 6 performance of different platform configurations is evaluated followed by a demonstration of applicability on a realistic scenario in Section 7. Finally, in Section 8 the work is concluded and an outlook is given to possible further work.

2. Fundamentals and Related Work

2.1. Parallel and Distributed Discrete Event Simulation. Parallel and distributed discrete event simulation (PDES and DDES) is a research topic for more than three decades now. It denotes a technology that enables execution of discrete event simulations (DES) on parallel and/or distributed platforms. Major objectives are the following [8]:

(i) reduction of execution time,

(ii) enabling larger simulations by incorporating distributed resources,

(iii) reuse in terms of integration of possibly spatially distributed simulators of different types.

An integral part of PDES research deals with the so-called *synchronization problem* [9] since the chosen synchronization mechanism in combination with the underlying hw/sw platform fundamentally influences performance. Basically, a PDES is made up of several *logical processes* (LP) that maintain separate event queues. Within each LP time progress is controlled by a scheduler that always selects the smallest time stamp event for execution. Whenever causal dependence between local events in the queue and events generated by other LPs cannot be excluded, the LPs must also consider these remote events during event scheduling. Various algorithms have been proposed for this issue in the literature. They can be divided into *conservative* and *optimistic* approaches. In short, conservative synchronization algorithms avoid violating the causality relationships between LPs by always guaranteeing event delivery and execution in the correct time order. The approaches in [10–12] are examples of conservative algorithms. Also the method we proposed in [6] for system/network co-simulation is a conservative approach. In contrast, optimistic approaches allow violating the causality relationships but provide for mechanisms to restore already past points in time (e.g., [13] or [14]).

Simulator interoperability is another major research topic that directly derives from the afore-mentioned "reuse" objective. Up to now, standards like HLA [15] have mainly addressed the syntatic interoperability between different simulation systems. One major reason why PDES/DDES is still not really established for common use is the lack of adequate tools that automate the process of configuring and setting up a distributed simulation for efficient execution. In this context, Strassburger et al. mention in [16] as one of the main research challenges and compelling problems in the field of distributed simulation systems the need of a technical approach for *true plug-and-play capability* between simulation packages of different domains.

2.2. Network Modeling and Design. For modeling and design of computer networks, there exists a wide range of so-called network simulation tools. Well-known examples are OMNET++ [3], ns-2 [17], ns-3 [4], or JiST [18]. These tools generally facilitate specification and simulation of the behavior of complete protocol stacks referring to the ISO/OSI reference model. A network model typically consists of several or many devices being interconnected by virtual wired or wireless links. Each device implements the functional description of the targeted network protocol including protocol state machines, packet structure, and data manipulations. Beyond that, possible environmental influences

on communication behaviour like mobility or obstacles can also be defined. During simulation execution, device models interact with each other through virtual network channels. Typical observable parameters are network traffic, transmission latencies, packet loss, or link failures. The so-called *network emulation* introduced by Fall [19] tries to increase realism and precision of generated network traffic by integrating physical systems with a network simulation. Since the approach of Fall demands for real-time execution, recent work [20] introduced an approach for synchronized network emulation using virtual machines.

Generally, the main drawback of network simulation tools is their ineptitude for detailed system modeling, since they do not reproduce behavior and interaction of underlying hardware and software components accurately.

2.3. Simulation Platforms for Embedded Systems. Today, simulation plays a major role during both design space exploration (DSE) of embedded systems as well as software development for such systems. OVP [21] and GEM-5 [22] are two well-known simulation platforms in the embedded domain. OVP is optimized for very fast execution and targets the provision of virtual prototypes for software development. Due to the lack of enough accuracy, models provided by OVP are typically not suitable for a DSE. GEM-5 is a simulator for computer architecture research with which microarchitectural characteristics like pipelines, caches, or interconnects can be modeled and simulated, making it suitable for a DSE.

In a typical DSE process, a basic architecture platform consisting of hardware and software is assumed, that is, parameterizable to a certain extend. Starting from a given target application, the architecture platform is iteratively refined towards the final implementation. In the domain of MPSoCs, recent approaches employ multiaccuracy simulation models that allow choosing between faster or more accurate simulation. In this context, Moreno et al. [23] propose an MPSoC design flow that integrates NoC models of different accuracy. Indrusiak et al. [24] propose, a layered approach that allows obtaining accurate figures for communication latency from an abstract application model. In [25], a model-based methodology and supporting toolset is presented that lets designers estimate application-specific network on chip power dissipation at early stages of the design flow. Authors employ a unified model and define different application-mapping platform layers. Last but not least, an example of a simulation platform that can accelerate DSE by exploiting parallelism provided by SMP workstations is described in [26].

The presented frameworks and methodologies for DSE all target evaluation and analysis of embedded system architectures. However, they focus on optimization of single systems without considering interaction with an outer network.

2.4. Network Centric Development of Embedded Systems. Pioneering work in the area of system/network co-simulation has been done by Fummi et al., for example, in [27, 28], reporting about a co-simulation environment for SystemC and ns-2. In [28], the authors show by means of several application examples that using system/network co-simulation reduces the modeling effort of networked devices and allows easy design verification and validation. In [27], they focus on timing accurate integration and synchronization and describe necessary kernel extensions for a tightly simulator coupling by linking the kernels together. Recent work from the same authors [29] introduces a TLM-based methodology for system/network design-space exploration of networked embedded systems which extends the traditional transaction level modeling [30] refinement process with a new dimension to represent network configuration alternatives. They derive a general criterion to map functionalities to system and network models and apply the proposed methodology to the design of a Voice-over-IP client which is partially refined down to RTL level. An example for a co-simulation framework based on a loosely coupling of SystemC and OMNET++ is given in [31] which is motivated by the evaluation of automotive IT architectures and the fact that in the automotive industries predominantly so-called "rest bus" simulations are state of the art which rely on real working hardware, and therefore, can only take place in late design phases. Finally, the approaches in [32, 33] examples for co-simulation platforms in the domain of wireless sensor networks. The former focusses on the evaluation of hardware/network interactions in WSNs in general, whereas the approaches in [33] focusses on investigating the application of reconfigurable hardware in the WSN context. Thereby, the authors explicitly model on cycle accurate level in order to reproduce hardware behaviour in detail.

Except this work, there exists no research paper targeting a system/network co-simulation environment that can be specialized towards different use cases and allows integration of simulation tools in a construction kit-like manner. Furthermore, none of the related work supports co-simulation, co-emulation, and parallel/distributed execution at the same time.

3. Simulation Methodology

The proposed simulation methodology describes the way of proceeding for a network centric evaluation of embedded MPSoCs using various abstraction levels. It is illustrated in Figure 1.

The methodology is composed of three steps.

(1) *Scenario Definition.* A *scenario* is defined for which a system analysis by means of simulation should be performed. The scenario definition is given by a specification of the communication protocol including the target application and the environmental context (e.g., node position, distance, mobility, etc.) in which the application will be executed. The scenario definition is done in the network domain using a respective network simulation tool, since such tools inherently provide support for defining a scenario in a formal way using a specification file.

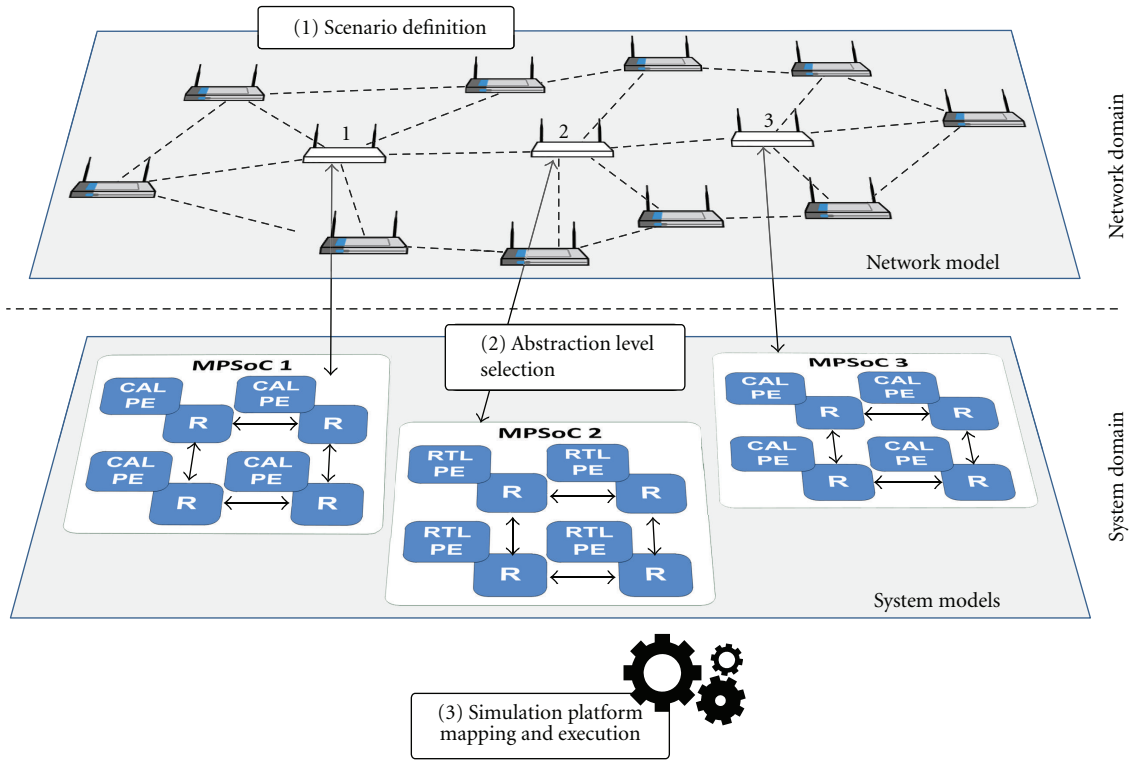


Figure 1: Simulation methodology.

(2) *Abstraction Level Selection.* A *cross-domain simulation model* (CSM) is generated that links the network domain to the system domain. Therefore, it is defined, which of the nodes of the scenario definition remain an abstract artifact in the network domain model and which of them are modeled in more detail. In the latter case, an abstract artifact in the network domain needs to be linked with an associated detailed artifact in the system domain (*system domain model*). In Figure 1, only cycle accurate and register transfer level are illustrated. Other abstraction levels or even mixed abstraction levels within a single node are also imaginable. In [6], the foundation for dividing a node into network and system domain artifacts is described. From the communication protocol perspective nodes can be either *local nodes* or *remote nodes*. Local nodes implement the complete protocol stack of the network protocol that is simulated within the network domain. Remote nodes only implement lower protocol layers within the network domain. Upper protocol layers are implemented in the system domain. The position of the cut between protocol layers determines the data that needs to be exchanged between network and system domain artifacts of a remote node. The decision whether a node is extended to a remote node depends on the targeted design or verification *use case* (e.g., network-wide optimization of several node architectures or verification of a single node). Basically a nodes internal hardware/software interactions need to be reproduced in detail if there is the necessity of gaining information about hw/sw-system characteristics, for example, in terms of latency and throughput.

(3) *Simulation Platform Mapping and Execution.* The CSM is mapped and executed on an appropriate instance of the *Simulation Platform* (SP) (see Figure 2). The SP is made up of two *platform levels* (not to be confused with *abstraction levels* of step two) called *simulator interoperability level* (SIL) and *execution platform Level* (EPL). Each platform level is assigned a distinct task. The SIL provides the *simulation infrastructure* for the CSM in terms of interconnected and correctly synchronized domain specific simulators. The EPL is the *execution infrastructure* in terms of different workstations, rapid prototyping systems, and local area network (LAN). Different manifestations of the platform levels can be combined in a construction kit-like manner. A concrete manifestation again depends on the targeted *use case*, characteristics of the CSM like number of system domain models and execution performance requirements.

4. Target Cross-Domain Simulation Models

The targeted cross-domain simulation models consist of instances of the HeMPS MPSoC [7] in the system domain and a 802.11 wireless network model in the network domain.

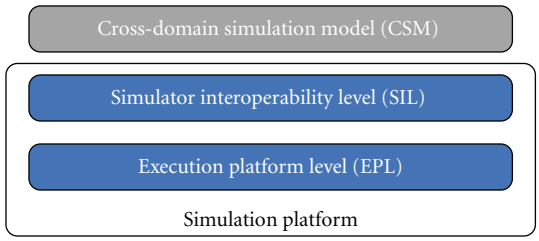


Figure 2: Simulation platform.

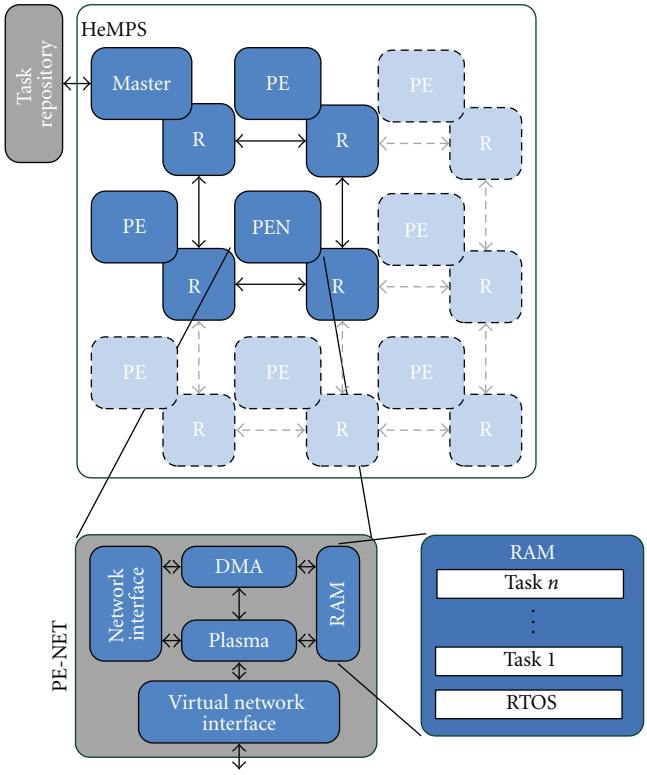


Figure 3: HeMPS with virtual network interface extension.

In the following, HeMPS itself as well as newly made interface extensions for integration of the model into a CSM are described. Besides that, details of interface extensions of the network model that have not been treated in previous work [6] are also shortly explained.

4.1. System Domain Model. HeMPS is a homogeneous MPSoC that consists of a configurable number of processing elements (PE) being interconnected by an NoC with mesh structure. It is completely available in SystemC and can be executed with the OSCI system kernel or modelsim. In contrast to the models used in previous work [6], HeMPS is much more realistic. Figure 3 gives an overview of the HeMPS architecture.

Main hardware components of HeMPS are the HERMES NoC [34] and the MIPS processor plasma [35]. The plasma cores execute a multithreaded real-time operating system (RTOS). Applications running on HeMPS are modeled using

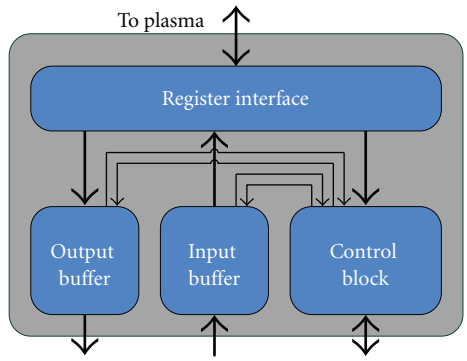


Figure 4: Virtual network interface.

task graphs. They are stored in an external memory named *task repository* and are allocated by a *master* to the remaining PEs (also called *slaves*). HeMPS has been chosen since it also fulfills the following requirements:

(i) scalability of the NoC size,

(ii) availability of RTL models and faster models on cycle accurate level (CAL) of the processing elements,

(iii) easy configurability (size, task mapping, routing strategy, etc.) and model generation,

(iv) HeMPS is open source.

The model has been extended with a virtual network interface (VNI) that can be connected to any of the PEs within the whole HeMPS array. The VNI is accessed from the SIL via a model adaptor (see Section 5.1) and forms the basis for data exchange with a network domain artifact on MAC layer. In Figure 4, the inner structure of the VNI is depicted. By means of a register interface, the peripheral is integrated into the address space of the plasma core. To exchange data with the SIL, input and output buffers are present. Both can hold 1024 bytes of data. Access to the buffers is directed by the control block which gathers the buffer states for generation of respective status signals and accepts read/write commands from the remaining model and the SIL.

4.2. Network Domain Model. Typically, nodes in network simulators are represented by hierarchically nested modules that communicate by passing messages to each other. Module nesting allows users to reflect the logical structure of network protocols. The modular approach of network simulation tools is exploited for instantiation of an invisible dummy module called VNI_Master which can be accessed by the model adaptor of the SIL. It is the counterpart to the VNI in the system domain. The VNI_Master acts as a kind of gateway and forwards network packets from the SIL to network domain artifacts (remote nodes). In turn, it also forwards network packets from network domain artifacts to the SIL. The approach of using a dummy module can be applied in any available network simulator since basic concepts are equivalent. However, due to its good extensibility, OMNET++ together with the MiXiM framework has been used.

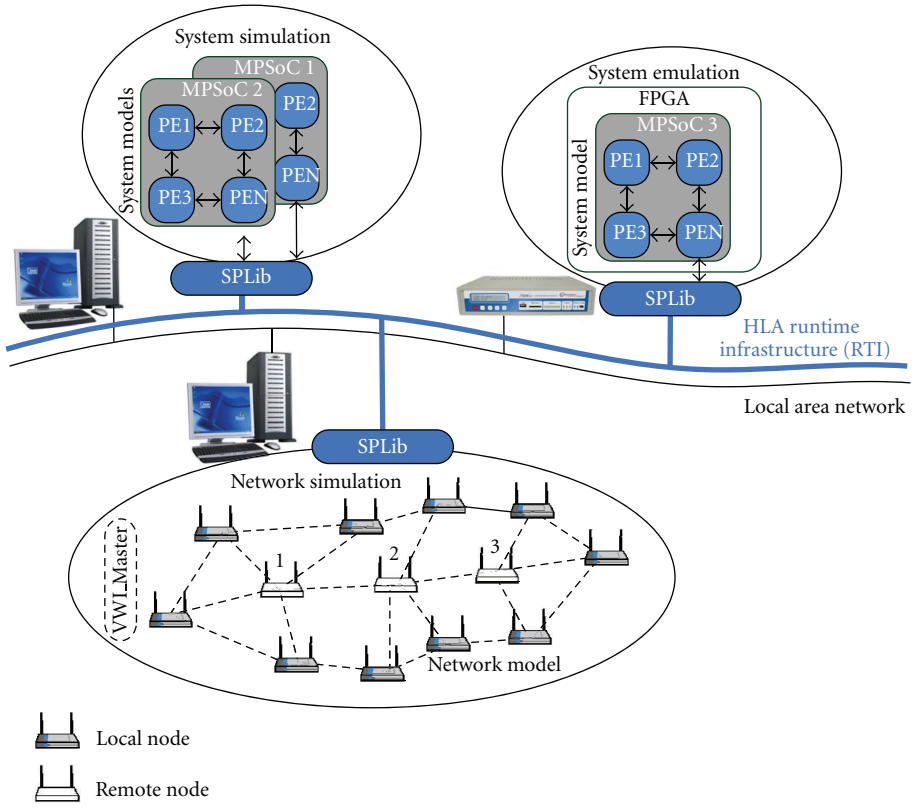


Figure 5: Simulation model and platform instance.

5. Simulation Platform

The construction kit-like design of the simulation platform facilitates

(i) assembly of different CSM, SIL, and EPL components,

(ii) reuse of domain specific simulation tools and models,

(iii) adaptation to specific use cases.

Figure 5 provides an overall view of an instantiation of the simulation platform that is feasible with the currently available components on the EPL and the SIL. Regarding the EPL, execution of either network or system simulators is done on Linux workstations. Emulation of system models is done on FPGA-based rapid prototyping systems called *Hardware-Prototyping and Emulation System—Industrial Reference Platform* (IRP) [36]. Since co-emulation has not yet been considered in previous work [6], SIL and EPL are described with a special regard on emulation and the difference to simulation. For the sake of completeness, available synchronization mechanisms on the SIL, as they were already described in [6], are shortly depicted.

5.1. Simulation Interoperability Level. The implementation of the SIL is given by the *Simulation Platform Library* (SPLib). The SPLib encapsulates an HLA simulation backbone called CERTI [37]. The SPLib complements the HLA with domain specific interfaces that facilitate controlled data exchange between domain specific models. The link that is formed by the SPLib between HLA, simulators, emulators, and models is basically fixed but parameterizable within certain degrees of freedom, that is,

(i) type of network simulator,

(ii) number and type of system simulators,

(iii) number of system emulators,

(iv) number of models per system simulator or emulator,

(v) synchronization method used for controlling data flow between models.

The library structure is illustrated in Figure 6.
It consists of the following:

(i) different types of adaptors for model connection,

(ii) different types of controllers implementing different synchronization methods,

(iii) a database for configuring the library regarding different use cases,

(iv) ambassadors that access the HLA services.

A model adaptor interacts via function calls with the connected model. However, while the OMNET++ and SystemC model adaptors directly access the VNI_Master respectively, VNI (see Section 4), the functions of the emulator

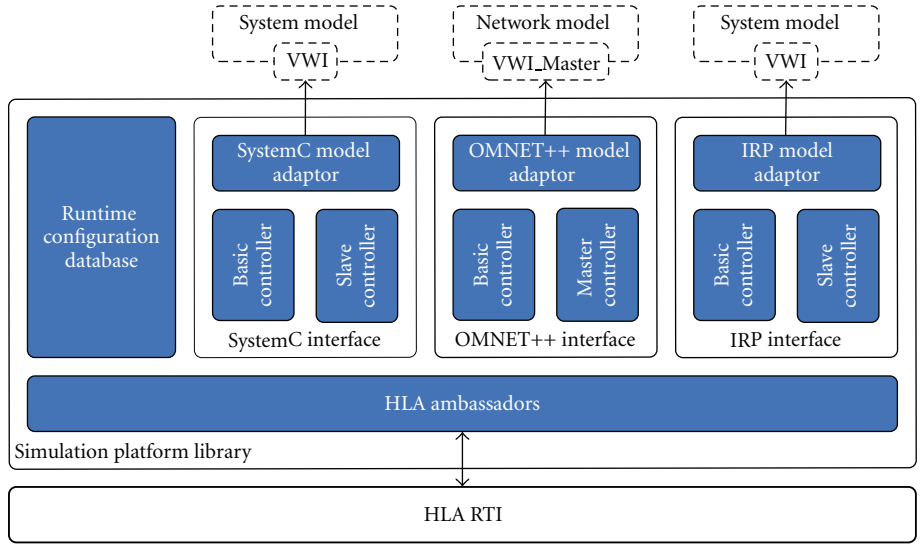


Figure 6: Simulation platform library.

adaptor need to be able to access the emulation hardware. As will be described in more detail in Section 5.2, the IRP emulation system consists of an Atom processor that connects to a FPGA via PCIe. The SPLib runs on the Atom while the model runs on the FPGA. Because of that, the adaptor functions have been equipped with routines that allow reading and writing from/to the PCIe bus by means of a PCIe driver and thus accessing the model indirectly.

Different synchronization methods are implemented by means of different controller instances. In case of SystemC or OMNET++ simulation, a controller implementation performs time advancement of the respective kernel by directly interacting with its scheduler. In contrast, in case of emulation, time advancement of the model that is running on the FPGA is controlled by accessing a special cycle register via PCIe that allows model execution for a configurable amount of cycles. Currently, basic *symmetric* as well as master/slave based *asymmetric* controllers are available for simulation and emulation. For the sake of completeness, their implementation is shortly summarized in the following.

5.1.1. Symmetric Conservative Synchronization. Symmetric conservative synchronization is already inherently supported by the HLA. CERTI provides two conservative algorithms for this purpose, the *Null Message Algorithm* (NMA) [10] and the *Null Prime Message Algorithm* (NPMA) [11], an improved version of the NMA. Regardless of whether the NMA or the NPMA is configured in CERTI, the HLA API remains the same. As has been shown in our previous work [6], when relying on system models of a very fine time granularity of events, symmetric synchronization can become a strong bottleneck in co-simulation since it incorporates all events that are generated in the system and the network domain. Most of the generated synchronization cycles are superfluous since no causal dependencies between simulators are generated.

5.1.2. Asymmetric Conservative Synchronization. The asymmetric synchronization method was proposed in [6]. It takes advantage of communication delays between simulated devices as guarantee for model independence. Time advancement is exclusively controlled by the network simulator which acts as master. It specifies how far system simulators (slaves) are allowed to run ahead. Hence, the amount of synchronization no longer depends on the fine time granularity of cycle accurate system models, but on network characteristics like message rate and transmission delays being of much higher scale. The only disadvantage of the asymmetric approach is that "node internal" events are possibly delayed. Internal events are events that are transmitted from one domain to another and that effect an immediate event in the opposite direction. For instance, a parameter request from an upper protocol layer simulated in the system domain to lower protocol layers simulated in the network domain is such a internal event.

5.2. Execution Platform Level. The execution platform level (EPL) is the lower level of the simulation platform. In the current implementation, two types of components are available, workstations with Linux OS as well as the newly supported IRP-based emulation systems which can be used for direct FPGA-based execution of RTL system models without the need of a simulator within the SIL. Basically, the choice between system simulation and emulation depends on both, use case and performance requirements. Exploiting IRPs for emulation is much more performant compared to workstation-based simulation (see Section 6); however, emulation is only possible if the system model is present as synthesizable RTL description and its resource requirements do not exceed available resources on the FPGA. Besides that, it is basically limited to use cases in the context of functional verification and application development or in which realistic traffic needs to be generated efficiently

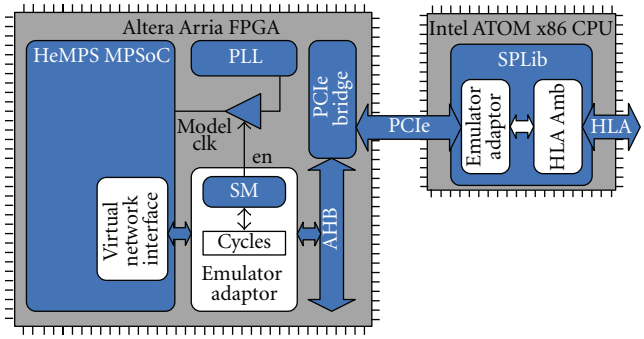


Figure 7: Emulator setup.

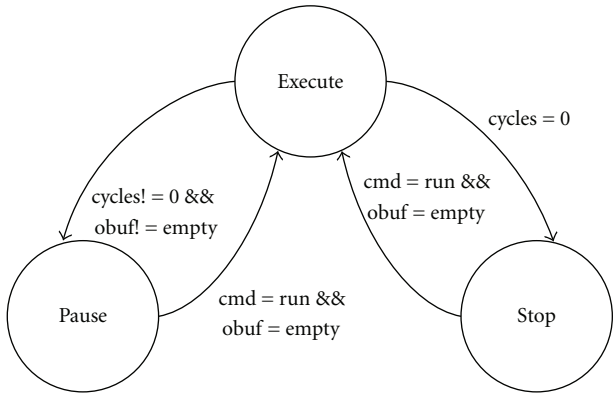


Figure 8: Hardware adaptor state machine.

(keyword *synchronized network emulation*). If the design use case demands, for example, for determination of the power consumption, then simulation on higher abstraction levels including a power estimation model is probably the best choice in order to do this.

The structure of the IRP together with the extensions towards emulation is illustrated in Figure 7. The IRP consists of a 1.6 GHz Intel Atom processor and an Altera Arria FPGA. Both are interconnected via a PCIe interface. The Atom runs the SPLib on top of a Gentoo Linux distribution and serves as mediator between FPGA, and HLA. On the FPGA the synthesized system model as well as some additional hardware components are executed. The model adaptor of the SPLib is complemented with a hardware counterpart that is connected to PCIe via AHB bus and accesses the model via the VNI.

In order to enable synchronized model execution and data exchange with the HLA, time advancement must be performed in a well-controlled manner which means regular start and automatic suspension of execution after a configurable number of clock cycles. For that reason, the hardware part of the model adaptor is equipped with a cycle register that allows specifying the number of clock cycles the model is allowed to run ahead. Activation and deactivation of the model clock is subject to the state machine illustrated in Figure 8.

During the *execute* state, the PLL clock is interconnected to the model and the cycle register is counted down with the processing frequency f_p until either reaching zero or a message is available in the output buffer. In the first case, the state machine switches to *stop* and waits for a new clock cycle value as well as a *run* command. In the second case, the output buffer must first be read out before the state machine can switch back to *execute*. The processing frequency f_p must not be confused with the model frequency f_m. The model frequency f_m expresses the "virtual" clock frequency of the system in relation to simulation time instead of the wall-clock time. When neglecting the communication and synchronization overhead between FPGA and Atom/HLA, the overall wallclock time $T(\Delta t)$ needed for executing a RTL model for a simulation time delta of Δt can be expressed as a relation of f_m and f_p times Δt

$$T(\Delta t) = \frac{f_m}{f_p} \times \Delta t. \quad (1)$$

The maximum value of f_p depends on factors like FPGA type, synthesis tool, and size of the model. Its value typically ranges in the order of several tens/hundreds of MHz.

Table 1: Variable benchmark parameters.

Parameter	Location
HeMPS abstraction level	CSM
Clock frequency	CSM
NoC size	CSM
Remote node fraction	CSM/SIL
Synchronization method	SIL
Execution platform	EPL/SIL
Workstation cores	EPL

6. Performance Evaluation

In the following, performance characteristics of the platform are evaluated. While focus in previous work [6] was on the comparison of different synchronization methods, the following evaluations target a more comprehensive consideration and evaluation of performance benefits of the overall simulation platform when using different

(i) abstraction levels on the CSM,

(ii) configurations on the SIL and EPL.

Hence, advantages of the construction kit-like design are emphasized. Starting point is a wireless communication scenario. From this, three different benchmarks are derived. Each benchmark covers one or several realistic use cases. Adjustable parameters that are applied in the benchmarks are summarized in Table 1.

6.1. Scenario Specification. Within the considered scenario, 64 nodes communicate via a 11 Mbps 802.11b wireless channel. They are arranged in a mesh structure. Neighbouring nodes have a distance of 20 m to each other. Each node transmits broadcast packets with a frequency of 100 Hz. Pure local nodes implement the broadcast application by means of a simple traffic generator that is triggered every millisecond.

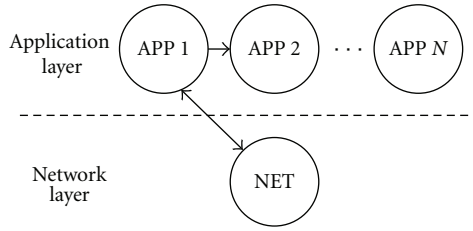


Figure 9: HeMPS software structure.

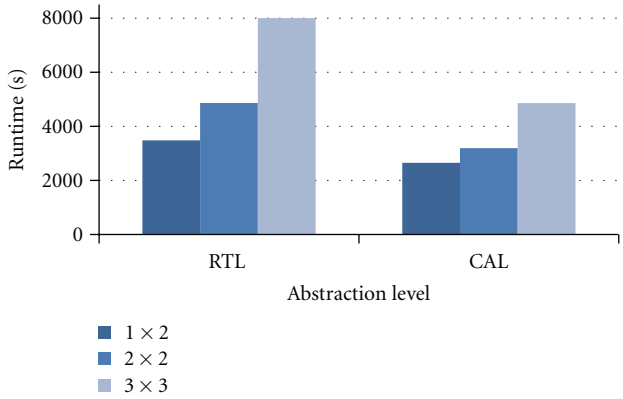


Figure 10: Results abstraction benchmark.

6.2. Additional Requirements for the CSM. Since the cut between network and system domain artifacts is done on MAC layer, the HeMPS systems need to implement application and network layer. Therefore, the MPSoCs run a network layer task that is regularly triggered by an application task pipeline in order to transmit the broadcast packets (see Figure 9). The application tasks in the pipeline simply generate workload in an endless loop.

Measurements have been performed with the OSCI SystemC kernel V2.2.0 and OMNET++ 4.1. Beside that, the CERTI null prime message algorithm (NPMA) is taken as basis for all measurements. The EPL was composed of a linux workstation (2.0 GHz QuadCore CPU, 8 GB RAM), an IRP system and a 100 MBit/s local area network.

6.3. Abstraction Benchmark. In this benchmark, a single remote node is co-simulated with 63 local nodes. This co-simulation use case is motivated by the need to analyse hw/sw characteristics of a single node (e.g., number of processing elements, task distribution, etc.) with respect to certain network traffic with the remaining nodes. RTL processing elements are needed if, for example, a detailed exploration of different types of MPSoC processing elements should be done. CAL processing elements can be used if only processing performance of the whole MPSoC with regard to network traffic shall be evaluated in detail. Symmetric synchronization is used for maintaining full accuracy also for node internal events. Sequential execution is necessary in the case that only a single workstation core is available. Parameters are summarized in Table 2.

Table 2: Abstraction benchmark configuration.

Parameter	Range
HeMPS abstraction level	RTL, CAL, NET
Clock frequency	100 MHz
NoC size	1×2, 2×2, 3×3
Remote node fraction	1/64
Synchronization method	Symmetric
Execution platform	1 workstation
Workstation cores	1

Results are illustrated in Figure 10. The simulation has been executed for 20 ms. As can be seen, in both cases, RTL and CAL, the runtime of co-simulation increases with the model size. However, when raising the abstraction level and using CAL instead of RTL, speedups of 1.3 (for the 1×2 system) to 1.65 (for the 3×3 system) are gained. The value of the speedup increases with the model size, since with increasing model size the computational overhead prevails the communicational overhead, introduced by the symmetric synchronization method. As a comparison, when simulating all 64 nodes as local nodes (pure network simulation, the most abstract case), the simulation only lasts three seconds instead of 2600 to 8000 seconds that were measured in case of co-simulation.

6.4. Parallelization Benchmark. In this benchmark, several remote nodes are co-simulated with several local nodes. This co-simulation use case is motivated by the need to analyse performance of a distributed application that is mapped onto different processing elements of distributed MPSoCs. The focus thereby would be on the mutual impact of selected MPSoC systems. CAL is used, since performance of each MPSoC as a whole is of interest. Parallel execution is possible in the case that several workstation cores are available. Parameters are summarized in Table 3.

Measured speedups are illustrated in Figure 11. In case of two, three, and four workstation cores, the network model was always executed separately on a single workstation core. As can be seen, except the 1×2 case, the speedup generally increases with increasing the degree of parallelization from two to three or four workstation cores. In case of a model size of 1×2 and four workstation cores, the computational overhead is obviously too low in order to provide an additional gain compared to three workstation cores. Beside that, the speedup provided by two workstation cores compared to a single one is marginal and partially even smaller than one. This is due to the fact that the network simulation always has been executed on a separate core and the bulk of computational overhead is generated by the system simulation. Thus, the network simulation is idle most of the time. However, for example, in case of a 1×2 model and three workstation cores (one running the network model exclusively), a larger speedup than two is measured. This in turn can be traced back to cache effects within the memory hierarchy of the used processors. When using a single workstation core for execution, the amount

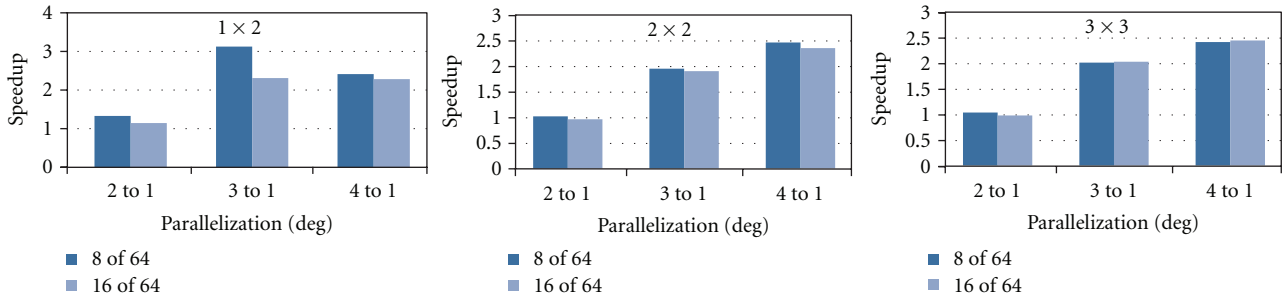


Figure 11: Results parallelization benchmark.

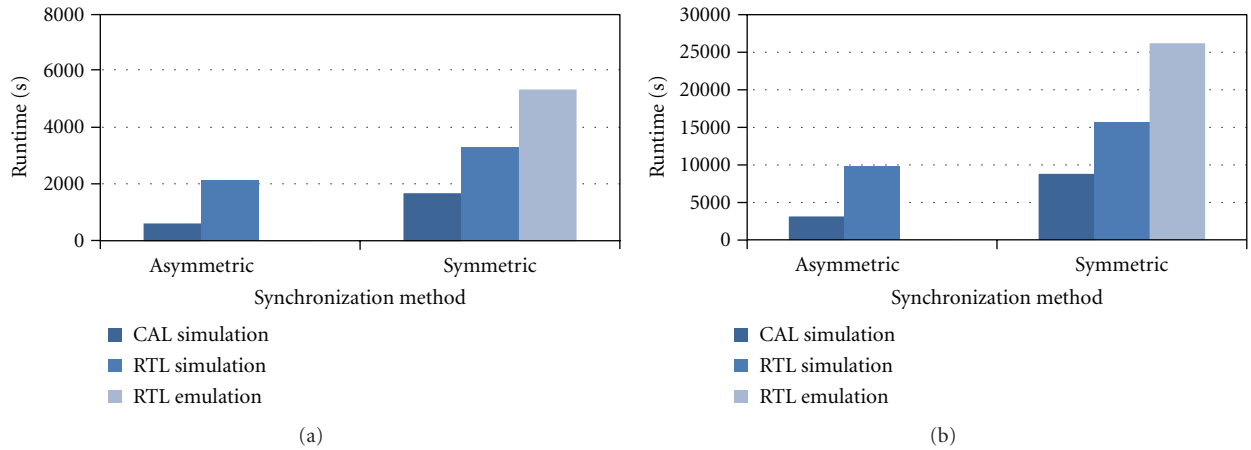


Figure 12: Results emulation benchmark (100 MHz (a) and 500 MHz (b)).

Table 3: Parallelization benchmark configuration.

Parameter	Range
HeMPS abstraction level	CAL
Clock frequency	100 MHz
NoC size	1×2, 2×2, 3×3
Remote node fraction	8, 16 of 64
Synchronization method	Symmetric
Execution platform	1 Workstation
Workstation cores	1–4

Table 4: Emulation benchmark configuration.

Parameter	Range
HeMPS abstraction level	RTL
Clock frequency	100 MHz, 500 MHz
NoC size	2×2
Remote node fraction	1/64
Synchronization method	Symmetric, asymmetric
Execution platform	1 workstation, 1 IRP
Workstation cores	1

of data that needs to be processed obviously outvalues the amount of fast cache memory. If increasing the number of workstation cores, more fast cache memory is available for holding data. This reduces the number of slow RAM accesses and provides the performance gain.

6.5. Emulation Benchmark. In this benchmark, a single remote node is co-emulated with 63 local nodes. This co-emulation use case is motivated by the need to speedup execution of detailed RTL models and to evaluate a hardware prototype of a single node with respect to its behaviour under realistic network traffic conditions. Asymmetric synchronization can be used without a loss of accuracy if no node internal events occur. Parameters are again summarized in Table 4. For the purpose of comparison, the same measurements are also done using co-simulation.

Execution times for 20 ms of simulation time are shown in Figure 12. The following characteristic can be observed in case of simulation. Raising the simulated clock frequency from 100 MHz to 500 MHz goes along with an increase of the execution time by a factor of about five. This characteristic is observed for both RTL and CAL simulation. Also the synchronization method has no significant influence on this characteristic. Both RTL- and CAL-based processing elements obviously provide enough computational overhead to hide the synchronizational overhead.

In case of emulation, the following can be observed. When using asymmetric synchronization co-emulation

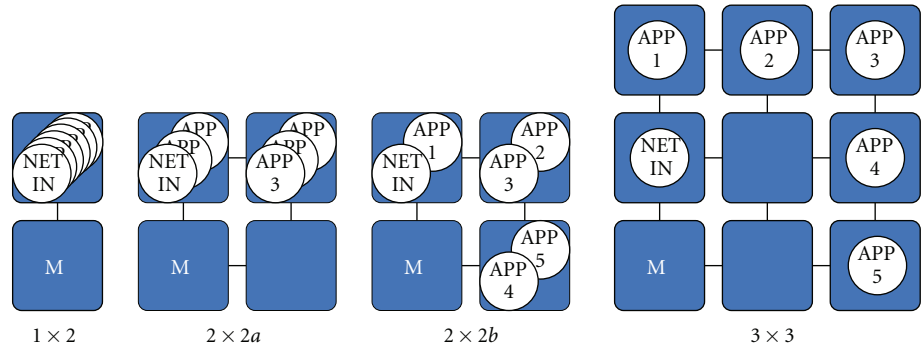


Figure 13: MPSoC configurations.

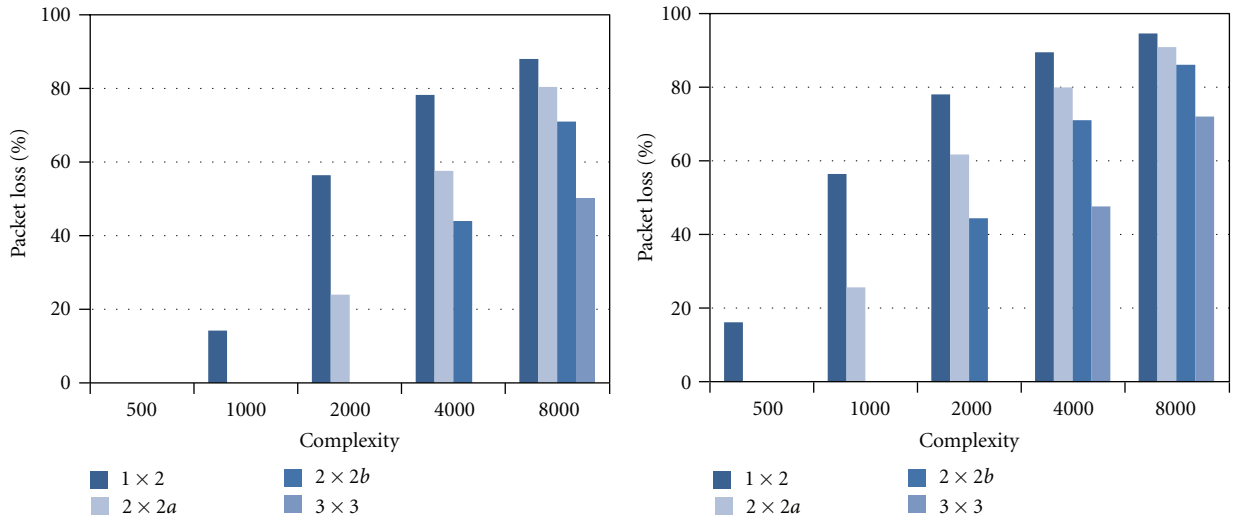


Figure 14: Dummy application: 500 Pkt/s (a) and 1000 Pkt/s (b).

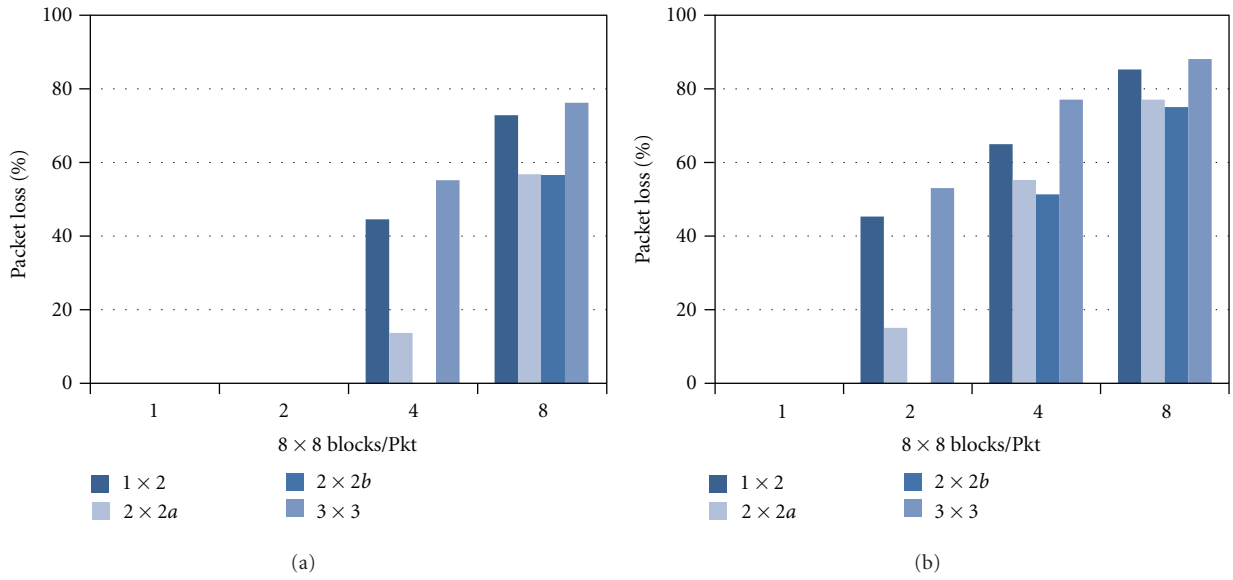


Figure 15: MPEG decoder: 500 Pkt/s (a) and 1000 Pkt/s (b).

strongly outperforms co-simulation for both simulation clock frequencies. Reasons are the independence of asymmetric synchronization of the time granularity [6] and the reduced computational overhead through FPGA execution (after logic synthesis the 2×2 RTL model runs with a processing clock frequency f_p of 25 MHz on the FPGA). In contrast, symmetric co-emulation generates even worse results than any co-simulation benchmark. This can be traced back to the additional delay of the LAN that connects the IRP system to the workstation.

7. Functional Simulator Verification

In order to verify that the simulation platform is working correct and appropriate for the intended objectives, a further example scenario has been set up. By the help of this, the effect of different network packet rates and different MPSoC configurations on the packet loss within the virtual network interface is exemplarily investigated.

7.1. Scenario Specification. A number of nodes are arranged in a circle having a diameter of 20 m. In the middle of the circle is an additional local node that stimulates the others via a 11 Mbps 802.11b channel at different packet rates.

7.2. Additional Requirements for the CSM. Each node around the circle is a remote node. Used MPSoC configurations are characterized by a certain number of available processing elements in combination with a certain kind of task mapping which results in varying throughput constraints. Measurements are performed by means of two types of applications, a five task dummy application, and a realistic five task MPEG decoder, both based on the pipelined software structure of Figure 9. Applied configurations and task mappings are illustrated in Figure 13.

Beside the packet loss of 1% that is induced by the wireless channel model, each MPSoC configurations induces an additional packet loss within the virtual network interface. This packet loss is measured.

7.3. Dummy Application. In case of the dummy application, each received network packet is assigned a certain amount of workload w in terms of integer multiply operations. Each application task processes 1/5 of the overall workload w by executing $1/5 \times w$ integer multiply operations. Benchmarking results are illustrated in Figure 14.

As can be seen, the hw/sw configuration strongly influences throughput and therewith the packet loss. In case of 500 pkt/s, the 1×2 system is able to cope with a packet complexity of 500 integer multiply operations, whereas the 3×3 system can process complexities of 4000 operations without packet loss. Doubling the packet rate to 1000 pkt/s expectably results in a left shift of the diagram.

7.4. MPEG Decoder. The same analysis has been performed by means of a realistic MPEG decoder consisting of an initiator task (INIT) that divides network packets into 8×8 MPEG blocks, a variable length decoder task (VLC), an inverse quantization task (IQUANT), an inverse discrete cosine transformation task (IDCT), and an output task (OUT) for data printing. Each received network packet contains a configurable number of 8×8 blocks of an encoded MPEG stream which is forwarded into the pipeline. As measurement results in Figure 15 reveal, variant $2 \times 2b$ is the most appropriate since it can cope with 4 blocks/pkt at 500 pkt/s and 2 blocks/pkt at 1000 pkt/s. The additionally emerging communication overhead of the 3×3 system is obviously too large to further speedup MPEG processing. This results in an increase of the packet loss compared to the $2 \times 2b$ system of 21%, respectively, 13% for 500 pkt/s, respectively, 1000 pkt/s.

8. Conclusion

We presented a methodology that enables efficient execution of cross-domain simulation models on different abstraction levels. Basis of the methodology is a construction kit-like simulation platform that allows assembly of different components and eases adaptation to different design and verification use cases. The usefulness of the methodology and its underlying simulation platform have been demonstrated by performance benchmarks that included consideration of different abstraction levels, parallelization and emulation. Beside that, the platform has been functionally verified by a realistic use case. We are sure, the methodology can support productivity and efficiency during development of networked embedded systems like future Cyber-Physical Systems.

Further research will focus on automatization of the simulation platform mapping step that is part of the methodology. Therefore, the development of an online profiling tool is planned which allows discovering bottlenecks during simulation execution. Results shall serve as basis for the development of a formal model of the platform that allows estimating performance. In this context, system/network co-simulation shall also be combined with distributed simulation of single systems. In addition, further abstraction levels are planned to be integrated into the HeMPS MPSoC model.

References

[1] P. Tabuada, *Cyber-Physical Systems: Position Paper*, 2006.

[2] "Modelsim," http://www.model.com/.

[3] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM '01)*, June 2001.

[4] "ns-3," http://www.nsnam.org/.

[5] C. Roth, G. Almeida, O. Sander et al., "Modular framework for multilevel multi-device MPSoC simulation," in *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symbposium*, 2011.

[6] C. Roth, O. Sander, and J. Becker, "Flexible and efficient cosimulation of networked embedded devices," in *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design (SBCCI '11)*, pp. 61–66, ACM, New York, NY, USA, 2011.

[7] E. A. Carara, R. P. de Oliveira, N. L. V. Calazans, and F. G. Moraes, "HeMPS—a framework for NoC-based MPSoC generation," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '09)*, pp. 1345–1348, May 2009.

[8] R. M. Fujimoto, "Parallel simulation: distributed simulation systems," in *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation (WSC '03)*, Winter Simulation Conference, 2003.

[9] R. M. Fujimoto, *Parallel and Distribution Simulation Systems*, John Wiley & Sons, New York, NY, USA, 1999.

[10] M. Chandy and J. Misra, "Distributed simulation: a case study in design and verification of distributed programs," *IEEE Transactions on Software Engineering SE-51979*, no. 5, pp. 440–452.

[11] J. Chaudron, E. Noulard, and P. Siron, "Design and modeling techniques for real-time RTI time management," in *Proceedings of the Spring Simulation Interoperability Workshop*, 2011.

[12] B. D. Lubachevsky, "Efficient distributed event-driven simulations of multiple-loop networks," *Communications of the ACM*, vol. 32, no. 1, pp. 111–131, 1989.

[13] D. Jefferson, B. Beckman, F. Wieland, L. Blume, and M. Diloreto, "Time warp operating system," in *Proceedings of the eleventh ACM Symposium on Operating systems principles (SOSP '87)*, vol. 21, no. 5, pp. 77–93, ACM, 1987.

[14] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[15] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)," IEEE Std 1516.x-2010, 2010.

[16] S. Strassburger, T. Schulze, and R. Fujimoto, "Future trends in distributed simulation and distributed virtual environments: results of a peer study," in *Proceedings of the Winter Simulation Conference (WSC '08)*, pp. 777–785, December 2008.

[17] "ns-2," http://nsnam.isi.edu/nsnam/.

[18] R. Barr, Z. J. Haas, and R. van Renesse, "JiST: an efficient approach to simulation using virtual machines: research articles," *Software-Practice & Experience*, vol. 35, no. 6, Article ID 106016, pp. 539–576, 2005.

[19] "Network emulation in the vint/ns simulator," in *Proceedings of the 4th IEEE Symposium on Computers and Communications*, p. 244, IEEE Computer Society, 1999, http://dl.acm.org/citation.cfm?id=876890.880459.

[20] E. Weingärtner, F. Schmidt, T. Heer, and K. Wehrle, "Synchronized network emulation: matching prototypes with complex simulations," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 2, pp. 58–63, 2008.

[21] "Open Virtual Platform," http://www.ovpworld.org/.

[22] "The gem5 simulator system," http://www.m5sim.org.

[23] E. I. Moreno, K. M. Popovici, N. L. V. Calazans, and A. A. Jerraya, "Integrating abstract NoC models within MPSoC design," in *Proceedings of the 19th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP '08)*, pp. 65–71, June 2008.

[24] L. S. Indrusiak, L. C. Ost, F. G. Moraes et al., "Evaluating the impact of communication latency on applications running over on-chip multiprocessing platforms: a layered approach," in *Proceedings of the 8th IEEE International Conference on Industrial Informatics (INDIN '10)*, pp. 148–153, July 2010.

[25] L. Ost, L. S. Indrusiak, G. M. Guindani, F. G. Moraes, and S. Määttä, "Exploring NoC-based MPSoC design space with power estimation models," *IEEE Design and Test of Computers*, vol. 28, no. 2, pp. 16–28, 2011.

[26] I. M. Pessoa, A. Mello, A. Greiner, and F. Pêcheux, "Parallel TLM simulation of MPSoC on SMP workstations: influence of communication locality," in *Proceedings of the International Conference on Microelectronics (ICM '10)*, pp. 359–362, December 2010.

[27] F. Fummi, P. Gallo, S. Martini, G. Perbellini, M. Poncino, and F. Ricciato, "A timing-accurate modeling and simulation environment for networked embedded systems," in *Proceedings of the 40th Design Automation Conference*, pp. 42–47, June 2003.

[28] N. Drago, F. Fummi, and M. Poncino, "Modeling network embedded systems with ns-2 and systemc," in *Proceedings of the 1st IEEE on Circuits and Systems for Communications (ICCSC '02)*, pp. 240–2245, 2002.

[29] N. Bombieri, F. Fummi, and D. Quaglia, "System/network design-space exploration based on TLM for networked embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 9, no. 4, pp. 1–37, 2010.

[30] F. Ghenassia, *Transaction-Level Modeling with Systemc: Tlm Concepts and Applications for Embedded Systems*, Springer, Secaucus, NJ, USA, 2006.

[31] B. Müller-Rathgeber and H. Rauchfuss, "A cosimulation framework for a distributed system of systems," in *Proceedings of the IEEE 68th Vehicular Technology Conference (VTC-Fall '08)*, pp. 1–5, September 2008.

[32] X. Deng, Y. Yang, and S. Hong, "A flexible platform for hardware-aware network experiments and a case study on wireless network coding," in *Proceedings of the 29th conference on Information communications*, March 2010.

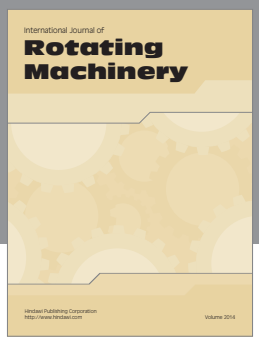
[33] J. Zhang, Y. Tang, S. Hirve, S. Iyer, P. Schaumont, and Y. Yang, "A software-hardware emulator for sensor networks," in *Proceedings of the 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '11)*, pp. 440–448, June 2011.

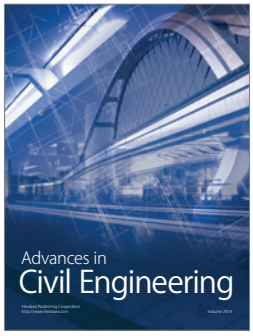
[34] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.

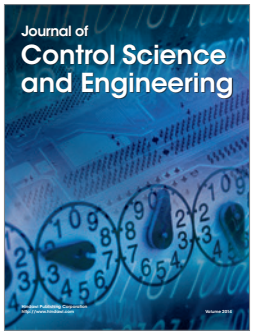
[35] "OpenCores," http://www.opencores.org/.

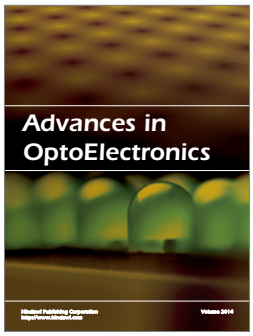
[36] G. E. Research, *Industrial Reference Platform*, Whitepaper, 2009.

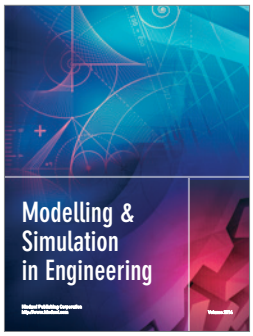
[37] E. Noulard, J.-Y. Rousselot, and P. Siron, "CERTI, an Open Source RTI, why and how," in *Proceedings of the Spring Simulation Interoperability Workshop*, 2009.

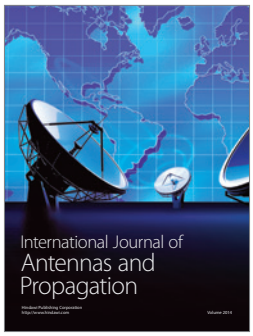


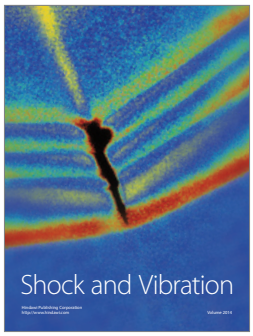


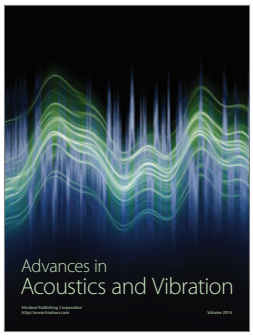












Hindawi

Submit your manuscripts at
http://www.hindawi.com