# HLA-based Simulation Environment for distributed SystemC Simulation

Christoph Roth, Oliver Sander, Matthias Kühnle, Jürgen Becker
Karlsruhe Institute of Technology (KIT)
Institute for Information Processing Technologies (ITIV)
Vincenz-Prießnitz-Straße 1
76131 Karlsruhe
{christoph.roth, oliver.sander, matthias.kuehnle, becker}@kit.edu

## ABSTRACT

We present a new approach of interconnecting diverse SystemC simulations using the High Level Architecture (HLA) as simulation backbone. The presented simulation environment is characterized by its generality and extendability. It basically allows different kinds of execution like distributed simulation of a single SystemC model as well as co-simulation with other arbitrary simulators. The emphasis within this work is on the synchronization and time flow mechanisms that need to be applied when executing a single SystemC model in parallel. A case study is performed by means of a loosely-timed SystemC transaction level model of a homogenous Multi-Processor System-on-Chip. The SystemC model exploits temporal decoupling which allows adjusting different computation to synchronization ratios, serving as basis for performance evaluation.

## Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Hardware description languages, Simulation*; I.6.8 [**Simulation and Modelling**]: Types of Simulation—*Discrete event, Distributed, Parallel*

## General Terms

Design, Languages

## Keywords

HLA, SystemC, TLM, System-on-Chip

## 1. INTRODUCTION

Especially in recent years a rapid evolution of embedded systems took place thus allowing for integration of more and more functionality into a single device. Within this development two facts can be observed: (1) Shrinking VLSI structure sizes in combination with the system-on-chip paradigm are one of the key enabler for power efficient single chip integration. (2) Many of the new functions we see today rely on communication between diverse embedded devices. One classic example are sensor actuator networks comprising a huge amount of embedded sensor nodes. Each of the nodes can be a system-on-chip device of intrinsic complexity. When simulating such communication-centric systems the limiting factor of simulation speed is mostly given by the communication and synchronization overhead. An approach to reduce this overhead is using more abstract transaction level models (TLM) [5],[13] which increase the ratio between computation and synchronization while still allowing for quite accurate evaluations of single devices.

However, we believe the simulation of a single device exclusively on the transaction level is not enough to exploit the impact of distributed sensor network applications on the single device system-on-chip architecture. In our work a more holistic approach is targeted, allowing for a scalable detailed simulation of several nodes or sub-modules on different abstraction levels like transaction level, cycle-accurate level (CAL) or register transfer level (RTL) [13] concurrently. This demands for an underlying simulation environment supporting scalability, adaptability and modularity. Therefore, within our concept each node or sub-module can be placed into its own SystemC [4] simulation. In a next step the SystemC simulations are interconnected by a generic simulation backbone based on the High Level Architecture (HLA) [6], inherently supporting parallel/distributed discrete event simulation (PDES or DDES) [12] [19] on diverse platforms as well as the connection of other arbitrary simulators like network simulators for more comprehensive evaluations. Communication of several SystemC simulations is thereby generally performed using higher abstraction levels in order to increase the computation to synchronization ratio.

Within this contribution we present the new approach of interconnecting diverse SystemC simulations using the HLA. We discuss important aspects of time synchronization between different SystemC simulations in detail and evaluate synchronization performance of the HLA backbone by means of a scalable transaction level model of a homogenous Multi-Processor SoC (MPSoC). Based on the results we draw first conclusions regarding distributed SystemC simulations using our approach.

The remainder of this paper is organized as follows: In section 2 we summarize a selection of related work. SystemC, the TLM 2.0 design methodology as well as concepts of the HLA are shortly depicted in section 3. Afterwards, in


SIMUTOOLS 2011, March 21-25, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245520

sections 4 and 5 the simulation environment is introduced and the experimental setup is described. Performance analysis results are presented in section 6. Section 7 concludes and points to further research.

## 2. RELATED WORK

The parallel simulation of SystemC models gains more and more interest in the research community. Within the last years diverse approaches into this direction have been developed. One of the first works explicity refering to the application of PDES is presented in [9] and [10]. The approach is based on kernel integration of remote functionality and allows for synchronization between several distributed SystemC kernels on delta cycle level via MPI. It is therefore basically applicable for RTL simulation. In [14] a similar approach is described that accesses the SystemC kernel via the SystemC user level functions. Due to the high delta cycle synchronization overhead both approaches only gain leverage for models that provide for high computation to synchronization ratios like TLMs. To further speed up pure TLM simulations several specific TLM engines have been developed. In [23] a light-weight simulation kernel is presented that is optimized for parallel simulation of *adaptive TLM models*. In [17] also a specialized simulation engine for TLMs and a modelling methodology called *TLM with distributed time* is introduced to simulate an MPSoC in parallel. Other concepts instead of PDES are used e.g. in [22] and [20]. In [22] the SystemC kernel is improved by parallel programming techniques, leveraging the parallel execution capabilities of multi-core machines. The authors of [20] use general purpose graphics processors (GPGPUs) for kernel parallelization. Finally an example having not performance but distributed IP core verification as focus is described in [16] and [11].

To the best of our knowledge none of these approaches uses the HLA as communication backbone. They all have the drawback of being limited to SystemC parallelization/ distribution whereas our approach basically allows for parallel/distributed simulation as well as co-simulation of SystemC models with other simulators. Further more, many approaches in the transaction level domain often depend on purpose-built simulation engines, making them not suitable for concurrent simulation on different abstraction levels like TLM and RTL, however, being a main demand of our research in the context of wireless sensor networks.

## 3. FUNDAMENTALS

### 3.1 SystemC

There is a set of design languages that can be chosen for hardware modeling during design space exploration (DSE). Most important characteristics of such a language are that it must support

- HW/SW co design
- reuse of existing IP (still C/C++ is clearly the most used high level programming language [18])
- system level IP integration
- different levels of abstraction
- creation of executable platforms in a straight forward manner

Among all languages that fulfill these properties, SystemC [4] (others exist like SystemVerilog, raw C++, UML, Vera, etc.) is probably of greatest significance, supported and widespread both by industry and academia.

Transaction level modeling (TLM) [5] [13] is an important methodology enhancing the SystemC standard. To speed up simulations it allows for abstraction of communication by defining specific base protocols and interfaces and provides for special modelling techniques like *temporal decoupling* (within its *loosely-timed* specification) and *pass by reference*. Furthermore, it permits fast design changes through guaranteed interoperability of TLM modules.

### 3.2 The High Level Architecture

The HLA was originally defined by the Defense Modeling and Simulation Office (DMSO) for the U.S. Department of Defense. Its original field of application are military training simulations in which thousands of military participants interact within a shared training exercise. The HLA is a generic software architecture combining all the components necessary for PDES. It determines the functional entities, design rules and interfaces for computer-based simulation systems and specifies the communication between the single components of an entire simulation. Communication between different simulators is independent of the underlying computing platforms. Further advantages of the HLA are the support of easy interoperability, reusability and adaptability. In 2000 the HLA became an international standard (IEEE 1516.x) [6].

The general structure of the HLA is shown in fig. 1. The logical representation of a interconnection of different simulators is called a *Federation* and includes multiple modules (*Federates*) which communicate via a *Runtime Infrastructure (RTI)*. The RTI provides for a number of services like Federation Management, Time Management or Object Management that are relevant for simulation control, synchronization and data exchange. Communication from the RTI to a federate and vice versa is established via the RTI-Ambassador and the Federate-Ambassador modules which cause a strict segregation of simulation and communication functionality.

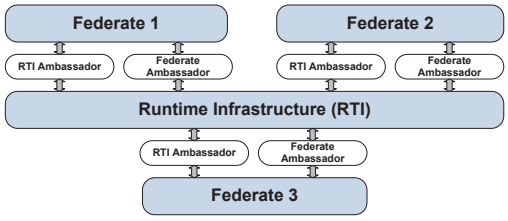


**Figure 1: The High Level Architecture (HLA)**

The HLA specifies a software architecture and not an implementation. There exist several commercial as well as open-source RTI implementations. The task of the simulation designer is to select an appropriate RTI and to develop the federates which finally access the RTI via the ambassador modules.

# 4. THE SIMULATION ENVIRONMENT

To the best of our knowledge this work is the first one considering the HLA for distributed simulation of SystemC models. Since we plan to apply it in the context of a complete wireless sensor network simulation, we are reliant upon sufficient performance, flexibility and scalability. These requirements are the justification for the HLA to form the core for managing communication and synchronization. The simulation environment shall support two types of execution:

1. Distributed simulation of several nodes together with other simulators like e.g. network simulators [2]

2. Distributed simulation of a single sensor node by dividing it into sub-modules which can be described on different abstraction levels like TLM, CAL or RTL concurrently

Thereby, each node/sub-module can be simulated by a separate SystemC kernel. In general, execution performance of a distributed simulation is greatly influenced by the synchronization overhead which increases with increasing communication effort (this issue is also illustrated by the later described experiments in section 5). For case one, synchronization is not that problematic since wireless communication latencies are much lower than intra-chip communication latencies. The bottleneck within the simulation environment will most likely be located in the distributed simulation of a sensor node itself. To relax the synchronization effort, data exchange between sub-modules (case two) is generally performed on higher levels of abstraction. In order to investigate the synchronization effort of intra-chip communication, within this paper we evaluate the distributed simulation of a single scalable transaction level model of a system-on-chip.

## 4.1 Synchronization and Time Flow

A distributed SystemC simulation may execute orders of magnitude slower than its non-distributed counterpart if synchronization becomes the dominant factor. The literature on PDES distinguishes between two types of synchronization algorithms *conservative* and *optimistic* [12]. In short, conservative synchronization algorithms avoid violating the causality relationships between the logical processes that are to be synchronized. They always guarantee events to be delivered in the correct time order. In contrast to that, optimistic approaches allow violating the causality relationships but provide for mechanisms like *timewarp* [15] to restore already past points in time. The HLA time management provides both types. Due to implementation complexity and enormous memory requirements of optimistic synchronization [12], we use the conservative synchronization interface of the HLA.

A second technique for which the HLA provides freedom of choice is the time flow mechanism. It can be *time-stepped* or *event-driven*. In a time-stepped approach simulation time is subdivided into a sequence of equal-sized time steps. The time globally advances from one time step to the next. In contrast to that, in an event-driven simulation the simulation state is only updated in case of an occuring event. Simulation time does not advance from one time step to the next but advances from the time stamp of one event to the next. Since the SystemC kernel itself is an event-driven simulation kernel and the time of the next event to be processed can always vary, the event-driven time flow mechanism is prefered.

In the following a simulation library is described, integrating an HLA interface with SystemC and implementing the described techniques.

## 4.2 The SystemC Federate

The SystemC federate library is the basic component for distributed SystemC execution. It combines the OSCI SystemC 2.2.0 kernel together with the HLA interfaces. In order to switch the RTI the federate library only has to be recompiled with a different RTI implementation.

Within each federate a separate SystemC simulation kernel is executed. To provide for maximum flexibility a modular structure similar to the HLA structure itself has been chosen (fig. 2). The functional components RTI ambassador, federate ambassador, adaptor, controller and object database are interconnected by the mediator which forwards communication requests. The federate library is equipped with an XML parser making it parameterizable also during runtime.

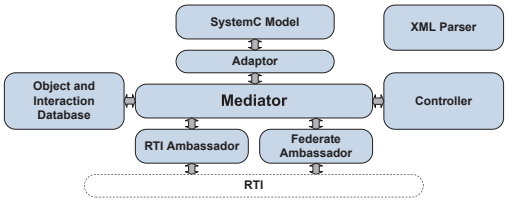


**Figure 2: SystemC Federate Library**

### *4.2.1 Controller*

The controller directs the local simulation which means initializing, executing and shutting down the federate. It contains the simulation loop. Since its implementation is model dependent, the controller is provided as an abstract base class. In fig. 3 the simulation loop is exemplarily illustrated for the transaction level model of section 5 by means of a state chart.

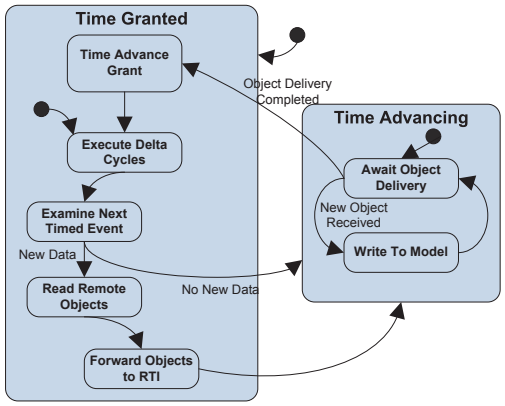


**Figure 3: Simulation Loop**

Conservative synchronization combined with an event - driven time flow works as follows: After having executed all

delta cycles at the actual point in time (*Execute Delta Cycles* state) by calling the $sc\_start(SC\_ZERO\_TIME)$ function repeatedly, each federate forwards remote data to the RTI and switches from *Time Granted* to *Time Advancing* by accessing the appropriate ambassador functions. Thereby, they request the RTI for a time advance to the time stamp of their next local timed event. After data delivery and determination of the lower bound on the time stamps (LBTS) of all global timed events by the RTI the federates switch back to *Time Granted*. Simulation time is only advanced if data is delivered (*Await Object Delivery* state) or the LBTS is reached (*Time Advance Grant* state) by calling $sc\_start(deltaTime)$. In order to access the SystemC kernel like in [14] the SystemC user-level functions are used. In contrast to [14], using our approach synchronization is generally not performed on delta cycle level but only on the level of timed events which is sufficient for our transaction level model.

#### 4.2.2 Adaptor

Models are integrated into the federate via an adaptor module. The adaptor performs the translation of local data into remote RTI objects or interactions. Its implementation can vary depending on the type of the model to be integrated (e.g. sub-module or complete node). Because of that, it is also provided as an abstract base class which has to be specialized for the connection of a designated SystemC model.

#### 4.2.3 Object and Interaction Database

Each federate must define a Simulation Object Model (SOM) [6] which determines the data (objects and interactions) that it is able to exchange. During the initialization phase the SOM is stored in the object and interaction database of a federate and that way accessible for all federate components.

## 5. EXPERIMENTAL SETUP

As a case study and in order to evaluate applicability and performance of the simulation environment, a transaction level model of a homogenous MPSoC has been implemented and prepared for distribution. In the following, the functionality of the model is shortly depicted.

### 5.1 TLM-based Reference Model

The overall structure of the MPSoC model is shown in fig. 4. It consists of several processing nodes being interconnected by a network on chip with mesh topology. The main feature is the scalability of the computation to synchronization ratio of each processing node as well as the quantity of nodes per row and column. The processing nodes are described in a loosely-timed coding style using the OSCI TLM 2.0 blocking transport interface [5].

A more detailed view of a single processing node is given in fig. 5. Its structure is generally characteristic for MPSoC nodes and consists of a processor connected to timer, router and memory via a local bus. The model applies temporal decoupling [5]. In general, temporal decoupling allows processes to run ahead of simulation time (instead of forcing them to synchronize with the SystemC kernel) by retarding calls to the $wait(time)$ function. This reduces costly context switches and increases the computation to synchronization ratio. To avoid processes to run ahead with no limit

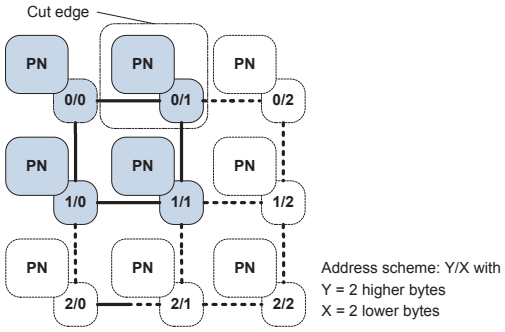


**Figure 4: MPSoC Structure**

the OSCI TLM 2.0 standard defines the global quantum $q$ which is an upper synchronization border. Synchronization is forced if the global quantum is exceeded. The concrete application of temporal decoupling within the sub-modules of the processing node is described below.

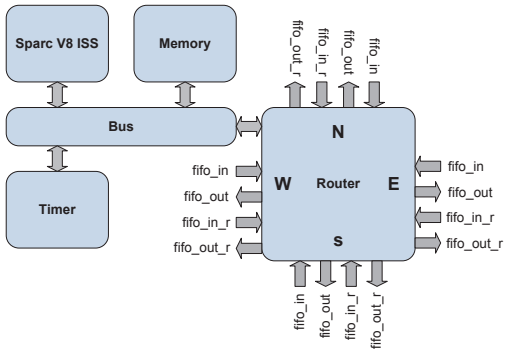


**Figure 5: Processing Node**

#### 5.1.1 Processor

In respect of future investigations we selected an instruction set simulator (ISS) as processor model, simulating an arbitrarily chosen architecture (SPARC V8) [7]. The ISS basically allows for simulations of instruction level time granularity. Temporal decoupling makes time granularity of communication coarser by aggregating the execution time of several instructions into the global quantum $q$. For this purpose, we define $q$ as the product of a number of instructions $n_i$ and the cycle time $c$ (assuming one instruction to be executed per cycle). This means that only after execution of $n_i$ instructions the $wait(q)$ function is called and control is returned back from the ISS thread to the SystemC kernel which then is able to schedule other threads. In general, $q$ is chosen several orders of magnitudes higher than $c$ in order to mimic long software execution times before interaction with the external network via the router can be performed.

#### 5.1.2 Router

The router is equipped with four fifos for each of the cardinal directions north, east, south and west (see fig. 5). The host fifos are used for packets to be transmitted to direct neighbours. Routing fifos are used in case of packet forwarding between nodes having greater distance than one

single hop. Each packet is divided into header and data field. Table 1 details the packet header. The allocation of the fields is given below.

| Header | | | | DATA |
|---|---|---|---|---|
| Word0 | Word1 | Word2 | Word3 | |
| 31..0 | 31..0 | 31..0 | 31..0 | |
| SRC | DST | LEN | TYPE | |

**Table 1: Fifo packet header structure**

**SRC:** Source address, the packets origin

**DST:** Destination address

**LEN:** Data length field in 32 bit words

**TYPE:** Packet type, set by application

Packet transmission and reception is performed by the *routing_thread* which is invoked in case of a new packet available either in one of the input fifos or in the internal packet buffer. In the reception case the *routing_thread* compares the destination address of the packet to the own address. If a match is detected, the packet is forwarded to the host, otherwise it is written in one of the outgoing routing fifos using a simple x-y-routing-scheme which first performs routing in horizontal and then vertical direction. In the transmission case the destination address is also analyzed and compared to the addresses of the neighbouring nodes. If a match is detected, the packet is written into the appropriate host fifo, otherwise it is written into one of the routing fifos. Basis for address comparison is the address assignment shown in fig. 4. Transmissions via fifos are afflicted with a delay $d_f$ ranging in the order of magnitude of several clock cycles which is modelled by a $wait(d_f)$ call. This results in a creation of events having much finer time granularity compared to the events created by the processor. The different granularity was chosen for the sake of a more realistic timing of simulation since in reality hardware routing occurs much faster than software controlled packet transmission.

## 5.2 Model Partitioning

There exist several possibilities for cutting the MPSoC model into parallelizable sub-modules e.g. partitioning along architectural boundaries or logical partitioning. We decided to use the former (being the most obvious variant) and simulated one processing node per federate. Fig. 4 exemplarily shows the cut edge for node 0/1. That way, each federate is always charged with similar workload. For the chosen partitioning the fifos that interconnect the processing nodes need to be cut into two parts. Each counterpart communicates with an adaptor (see section 4.2.2) which handles remote transmission and reception.

Remote fifo packets transmitted by one federate are broadcasted to all others. This makes the utilization of a unique ID necessary. Data that was read from a fifo is written by the adaptor into an HLA object called *hla_packet* (see table 2). Beside the already mentioned fifo packet fields of table 1 there exist two additional ones namely $fifoID$ and $rFlag$ for unique identification of the counterpart fifo at the receiver node. The $fifoID$ is generated by concatenation of sender address and receiver address of the next hop node. The order of concatenation determines the direction of communication. The $rFlag$ determines whether the next hop is the final one or not.

| | |
|---|---|
| fifoID | Unique ID for fifo identification |
| rFlag | Packet is intended for routing fifo |
| source | Source address |
| dest | Destination address |
| length | Data length |
| type | Packet type |
| data | payload |

**Table 2:** *hla_packet* **Structure**

## 5.3 ISS Software

Two types of software are executed on the processing nodes making them either a *producer* or a *consumer*. Consumers request the producers one after another by transmitting request-packets. After having forwarded a request-packet a consumer waits in a loop for an acknowledge-packet from the addressed producer. As soon as the acknowledge-packet is received the consumer transmits the next request-packet. Conversely producers permanently wait in a loop for request reception. As soon as a request-packet is received an appropriate acknowledge-packet is generated and forwarded.

Temporal decoupling affects the behaviour of consumer and producer by increasing the number of loop iterations until synchronization with the kernel and therefore with the appropriate router as well as other processing nodes is done. That way, different algorithm execution times are modelled.

## 5.4 Functional Verification

The global timing behaviour was verified to be correct by comparing the time stamps of transmitted packets while using either distribution or no distribution. Fig. 6 shows an example of the emerging synchronization pattern in the case of simulating four processing nodes in parallel. The global quantum $q$ sets an upper border for global synchronization. However, additional shorter synchronization cycles are automatically inserted in case of low latency packet transmission and direct routing due to the lower chosen delay $d_f$ of a fifo compared to the global quantum $q$.

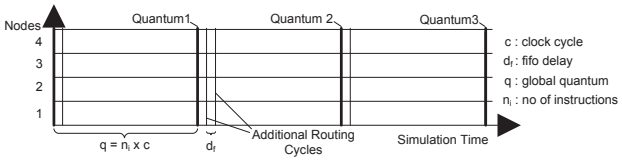


**Figure 6: Emerging Sychnronization Pattern**

# 6. PERFORMANCE ANALYSIS

To evaluate performance we utilized a network (Ethernet, 100 MBit/s) consisting of up to nine linux workstations with equal configuration namely a 2 GHz DualCore CPU as well as 2 GB RAM as hardware platform. We compiled the federate library with CERTI [24][21][1] since it is free open source software. Unfortunately we were not able to use Portico [3] because it had a bug in the time management services which was discovered during implementation. However, we plan to do so as soon as the bug is fixed. Looking at CERTI, the implementation of the RTI is done in a centralized manner. Fig. 7 shows the resulting layered implementation when using CERTI. The lower layers consist of two types of processes, local ones called *RTI Ambassadors* (RTIA) and a central one called *RTI Gateway* (RTIG). These processes as

well as the SystemC federate library are linked with each other using Unix and TCP sockets. Thereby the RTIG is of predominant importance since any form of communication between federates, be it for data exchange or be it for synchronization purposes, is done via the RTIG.

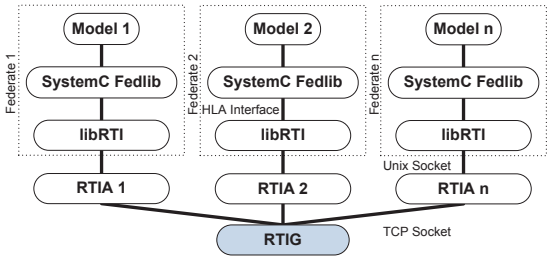


**Figure 7: Simulation Implementation using CERTI**

## 6.1 Synchronization Algorithms

In its actual implementation CERTI can be compiled with two types of time synchronization algorithms, the so called *Chandy/Misra/Bryant Null Message Algorithm* (NMA) [8] or the *Null Prime Message Algorithm* (NPMA) which was designed by the CERTI developers. The former one has the drawback of not avoiding the *time creep* problem, occuring in case of low *lookahead* values [12]. The *time creep* problem has strong effect in our case since we are forced to chose the fifo delay $d_f$ as *lookahead* which consists of only a few clock cycles. Applying the NMA we generally got execution times that always lay several orders of magnitude beyond the sequential case. The NPMA avoids the *time creep* problem by including the time stamp of the next unprocessed event (so called *conditional information* [12]) in computing LBTS values. Because of that, the measurements presented in the following are all based on the utilization of the NPMA.

## 6.2 Experiment 1: Varying Node Number

To evaluate performance when varying the node number, the reference model was configured with 2, 4 and 9 nodes as shown in table 3. In each row we put one producer. The clock frequency of the MPSoC model was set to 100 MHz. The overall simulation was executed for one second of simulation time. The experiment was carried out for the parallel as well as the sequential case. Looking at parallel execution, each federate process was executed together with its respective RTIA process on a separate cpu core. When starting a federate process, its RTIA process is automatically initialized as background process, having the same cpu affinity as the federate process itself. Within experiment one also the RTIG process was executed on a separate core. For the sequential case the whole simulation was perfomed as pure SystemC simulation without HLA extension on one single core.

| Case | Per row | Per column | Producers | Consumers |
|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 6 |

**Table 3: Model Configuration (Experiment 1)**

Fig. 8 points out the achieved speedup of the parallel case in relation to the sequential case depending on the global quantum which is indicated in nanoseconds. For all three cases the speedup increases with increasing global quantum which corresponds to an increasing number of simulated instructions per synchronization step. The speedup approximates to an area of saturation which is the maximum theoretically achievable speedup. Since we used one core for each node, this value corresponds to the number of cores in each case. Additionally it can be stated: When considering for example nine nodes, a parallel simulation is profitable for the given kind of distribution, if the global quantum exceeds about $3,3 * 10^4$ nanoseconds. At 100 MHz simulated clock frequency and one instruction per clock cycle this corresponds to $3,3 * 10^3$ instructions which each ISS must be allowed to run ahead before synchronizing again with the remaining simulation.

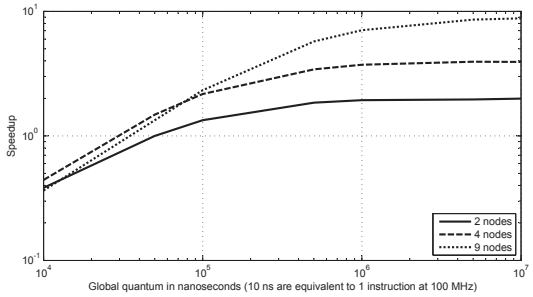


**Figure 8: Results (varying node number)**

## 6.3 Experiment 2: Varying Distribution

Finally, we evaluated the impact of different kinds of process distributions. For that reason, we used an MPSoC comprising only two nodes, one executing the consumer application and the other executing the producer application. The clock frequency of the MPSoC model as well as the maximum simulation time were the same as in experiment one. As shown in table 4, we distributed the federates F1 and F2 (including their particular RTIA processes) and the RTIG on the four cores of two workstations (WS1 and WS2) in different ways.

| | Dist1 | | | Dist2 | | | Dist3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | F1 | F2 | RTIG | F1 | F2 | RTIG | F1 | F2 | RTIG |
| WS1 core1 | X | | | X | | | X | | X |
| WS1 core2 | | X | | | | X | | X | |
| WS2 core1 | | | X | | X | | | | |
| WS2 core2 | | | | | | | | | |

**Table 4: Process Distribution (Experiment 2)**

For each of the three distributions we again measured the execution time that was needed to carry out the simulation for different global quanta. Results of the measurement are shown in fig. 9. For all three cases the speedup again increases with increasing global quantum, until reaching the maximum achievable speedup, which is two. As can be seen, the centralized implementation of CERTI has great influence on simulation performance. The smallest global quantum wherefrom parallel simulation is profitable depends on the kind of distribution. The earliest profitable variant is distribution three, since the speedup is greater than one already for a global quantum of about $2,6 * 10^4$ ns.

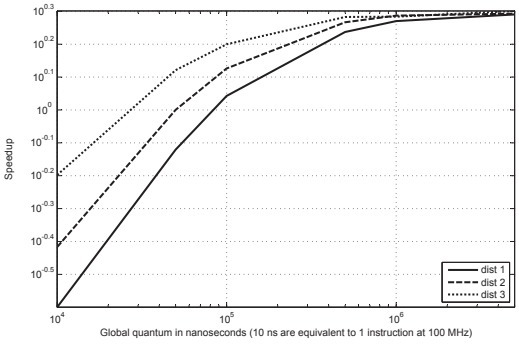


**Figure 9: Results (varying distribution)**

Despite the fact that in case of local execution on a single workstation F1 and the RTIG are executed on the same core, local execution is faster than any distribution on two workstations which adds the network latency of the Ethernet connection. For that reason, distribution one is the slowest variant since both federates have to communicate with the RTIG via Ethernet.

# 7. CONCLUSION AND OUTLOOK

We presented a simulation environment for parallel and distributed SystemC simulation. The approach is the first one combining SystemC with the HLA. Using the HLA a modular structure is obtained which is characterized by its scalability and extendability also for different simulators. We verified the correct functionality and evaluated the performance of the simulation environment by means of a scalable loosely-timed transaction level model of an MPSoC. We showed that for a loosely-timed transaction level model we are able to receive a maximum speedup that is almost proportional to the degree of parallelization if the ratio of computation to synchronization gains a certain level. We also showed that the underlying RTI implementation has great influence on execution performance.

Further research includes the evaluation of other possibilities for model partitioning as well as the impact on performance when describing selected nodes internally on lower abstraction levels like register transfer level. Moreover, the environment shall be extended regarding co-simulation capabilities with other domains like network simulators [2] which forms the basis for building up a simulation framework for SoC based wireless sensor nodes.

# 8. REFERENCES


[1] Certi. *http://www.cert.fr/CERTI.*

[2] The ns-3 Network Simulator. *http://www.nsnam.org/.*

[3] The Portico Project. *http://www.porticoproject.org.*

[4] IEEE Standard System C Language Reference Manual. *IEEE Std 1666-2005)*, pages 1 –423, aug. 2006.

[5] Open SystemC Initiative (OSCI) TLM Working Group. Transaction Level Modelling Standard 2 (OSCI TLM 2). june 2008.

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

[7] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, and E. Barros. The ArchC architecture description language and tools. *Int. J. Parallel Program.*, 33(5):453–484, 2005.

[8] M. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of distributed Programs. In *IEEE Transactions on Software Engineering SE-5, (5)*, pages 440–452, 1979.

[9] B. Chopard, P. Combes, and J. Zory. A Conservative Approach to SystemC Parallelization. In *Computational Science ICCS 2006*, volume 3994 of *Lecture Notes in Computer Science*, pages 653–660. Springer Berlin / Heidelberg, 2006.

[10] P. Combes, E. Caron, F. Desprez, B. Chopard, and J. Zory. Relaxing Synchronization in a Parallel SystemC Kernel. pages 180 –187, 2008.

[11] A. Fin, F. Fummi, and D. Signoretto. The use of SystemC for design verification and integration test of IP-cores. pages 76 –80, 2001.

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

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

[14] K. Huang, I. Bacivarov, F. Hugelshofer, and L. Thiele. Scalably distributed SystemC simulation for embedded applications. pages 271 –274, jun. 2008.

[15] D. Jefferson, B. Beckman, F. Wieland, L. Blume, and M. Diloreto. Time warp operating system. In *SOSP '87*, pages 77–93, New York, NY, USA, 1987. ACM.

[16] S. Meftali, A. Dziri, L. Charest, P. Marquet, and J. luc Dekeyser. SOAP Based Distributed Simulation Environment for System-on-Chip (SoC) Design. 2005.

[17] A. Mello, I. Maia, A. Greiner, and F. Pecheux. Parallel simulation of systemC TLM 2.0 compliant MPSoC on SMP workstations. pages 606 –609, mar. 2010.

[18] R. Merritt. Parallel software plays catch up with multicore. *available at http://www.eetimes.com*.

[19] J. Misra. Distributed discrete-event simulation. *ACM Comput. Surv.*, 18(1):39–65, 1986.

[20] M. Nanjundappa, H. Patel, B. Jose, and S. Shukla. SCGPSim: A fast SystemC simulator on GPUs. pages 149 –154, jan. 2010.

[21] E. Noulard, J.-Y. Rousselot, and P. Siron. CERTI, an Open Source RTI, why and how. In *Proceedings of the Spring Simulation Interoperability Workshop, San Diego, USA*, mar. 2009.

[22] E. P, P. Chandran, J. Chandra, B. P. Simon, and D. Ravi. Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines. In *PADS '09: ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 80–87, Washington, DC, USA, 2009. IEEE Computer Society.

[23] R. Salimi Khaligh and M. Radetzki. Modeling constructs and kernel for parallel simulation of accuracy adaptive tlms. pages 1183 –1188, mar. 2010.

[24] P. Siron. Design and Implementation of a HLA RTI Prototype at ONERA. In *Proceedings of the 1998 Fall Simulation Interoperability Workshop, Orlando, 1998.*