Simulation Environment based on SystemC and VEOS for Multi-Core Processors with Virtual AUTOSAR ECUs

Moisés Urbina, Zaher Owda, Roman Obermaisser
University of Siegen, Germany
{moises.urbina, zaher.owda, roman.obermaisser}@uni-siegen.de

***Abstract*—The extension of time-triggered message-based on-chip architectures towards an AUTOSAR MPSoC platform helps to achieve the AUTOSAR goals, in particular with respect to fault isolation and temporal predictability. Simulation environments enable early analysis and performance tests of the software for MPSoC platforms. However, there is no simulation environment that combines on-chip network communication and AUTOSAR-based software. This paper presents a simulation environment for TIme-triggered MEssage-based multi-core platforms based on AUTOSAR (TIMEA). Simulated application cores serve as virtual Electronic Control Units (ECUs), each containing an AUTOSAR operating system and a Run-Time Environment (RTE). The presented simulation environment performs a co-simulation of the AUTOSAR software, the natural environment and the time-triggered NoC. An on-chip simulation model in SystemC is combined with AUTOSAR simulation tools from dSpace. The capability of the simulation environment is evaluated using an antilock braking use case.**

I. Introduction

Multi-Processors System-on-a-Chips (MPSoCs) are becoming widely used for the development of embedded system applications. Several MPSoC architectures have been developed for specific application domains as presented in Sonics [1], AEthereal [2], Nostrum [3]. However, commercial MPSoCs are the cause of major concern to certification authorities [4]. Therefore predictable multi-core platforms (e.g. COMPSOC [5], ACROSS [6], GENESYS MPSoC [7]) provide message-based on-chip networks as a solution.

In this context, simulation environments are significantly important for the development of automotive applications based on MPSoCs, helping system architects in exploring new design decisions. Simulation environments enable early examination on MPSoC platforms, providing environments for performance evaluation before implementing the physical system. Several simulation environments exist for multi-core processors, e.g., based on SystemC or GEM5. In the automotive industry simulation environments play a very important role in the development process of embedded systems.

The AUTOSAR (Automotive Open System Architecture) standard [8] proposed a multi-core version for the AUTOSAR architecture in version 4 [9]. An AUTOSAR multi-core Operating System (OS) manages the parallel execution of AUTOSAR Software Components (SWCs) hosted by an MPSoC with multiple cores interacting through a shared memory.

In previous work [10] we introduced the model of an AUTOSAR MPSoC platform based on Time-Triggered Network-on-a-Chips (TTNoCs), which supports stringent fault isolation and temporal predictability [11]. This model presents autonomous application cores as virtual AUTOSAR Electronic Control Units (ECUs) on a single MPSoC. The Virtual Function Bus (VFB) of AUTOSAR is mapped to NoC messages and is available to the virtual ECUs. Each virtual ECU is an independent unit of abstraction with its own AUTOSAR Basic Software (BSW) and without hidden interactions between the BSW on different virtual ECUs.

However, the simulation of time-triggered message-based multi-core processors hosting AUTOSAR-based software is still an open research problem. The existing AUTOSAR multi-core tools that are available on the market implement a shared memory approach for multi-core simulations (e.g. VECTOR tools [12], ETAS tools [13]). There are no full-system simulation tools that support both AUTOSAR-based software and on-chip network simulation. Such as simulation environment is else required for the validation of the expected advantages provided by the NoC to the AUTOSAR application.

This paper presents as a novel contribution a simulation environment for time-triggered multi-core platforms based on AUTOSAR. We perform a co-simulation of an AUTOSAR-based system simulation tool and a multi-core simulation tool. The dSpace AUTOSAR tools are extended for supporting the simulation of time-triggered networked application cores.

AUTOSAR application cores and their interaction with the natural environment models are simulated by the VEOS simulation framework. A TTNoC model developed in SystemC is used for the simulation of the TTNoC behavior. We propose a communication interface for the synchronization and data exchange between the two simulation systems. Local simulation coordinators are defined, which determine when a co-simulation step can be performed.

The application software is developed using the dSpace tools SystemDesk and TargetLink in combination with Simulink. The communication between the virtual ECUs is performed through the TTNoC.

An anti-lock braking system is considered as an example use case for an evaluation of the developed AUTOSAR multi-core simulation environment.

The remainder of this paper is organized as follows. In Section II we present an overview of related work in the area of simulation environments. The time-triggered MPSoC model for AUTOSAR is presented in Section III. Section IV describes

the developed simulation environment. The explanation of the AUTOSAR development process for simulated multi-core processors is the focus of Section V. Section VI is dedicated to the presentation of an experimental use case. The paper finishes with a discussion and a conclusion in Section VII.

II. Related Work

Several simulation tools support multi-core processors, software architectures, on/off-chip networks and memory system simulations. For example, OPNET [14] and ONNET++ [15] are off-chip network simulation tools that are used for designing and validating networked systems. Furthermore, OPNET provides an interface that is compatible with other simulation tools (e.g. Simulink).

The On-Chip Communication Network (OCCN) [16] proposes a simulation environment for modeling and simulation of on-chip communication architectures. Additionally, cycle-accurate simulation tools such as SystemC and GEM5 are widely used for on-chip network simulations.

On-chip simulation is supported by full system simulators as Simics [17] and Sniper [18]. Also, SystemC and GEM5 support the simulation of system-level architectures.

Simulation tools such as VEOS (dSpace) and CANoe (Vector) provide simulation environments for experimental tests specifically in the automotive domain. These simulation environments support both application simulation as well as simulation of off-chip communication.

However the combination of these simulations tools represents a major scientific challenge. The definition of interfaces for data exchange and synchronization between the simulation tools is required.

In the last years, several co-simulation environments have been presented in research and industry for the co-simulation of off-chip network simulations, on-chip network simulations, and application functionality simulations.

For co-simulations focusing on the communication behavior of the systems, the co-simulation can be abstracted from the system computational functionality. In [19] a time-triggered cyber-physical system simulation is presented performing a co-simulation of SystemC with a hardware-in-the-loop automotive simulation tool. In [20], ONNET++ and SystemC are combined in a co-simulation system for the simulation of distributed systems.

On the other hand, for performing full-system co-simulations, virtual platforms are required to support hardware virtualization (e.g. OPVSim, QEMU). In [21] the so-called transformer is presented. This is a full-system simulator for multi-core simulation based on QEMU. In [22], [23], [24] QEMU and SystemC are combined for the emulation of the CPUs and the communication behavior of a multi-core network on-a-chip respectively. A co-simulation framework between the OVP virtual platform and QEMU is presented in [25]. This co-simulation framework implements a systemC-based interface for integrating systemC components to the overall simulated systems. Moreover, in [26] a design methodology for the integration of heterogeneous SWCs and multi-core architectures is presented. The so-called HeroeS is mapped on a SystemC virtual platform framework for the simulation of embedded real-time architectures, and was evaluated by its integration into an AUTOSAR environment.

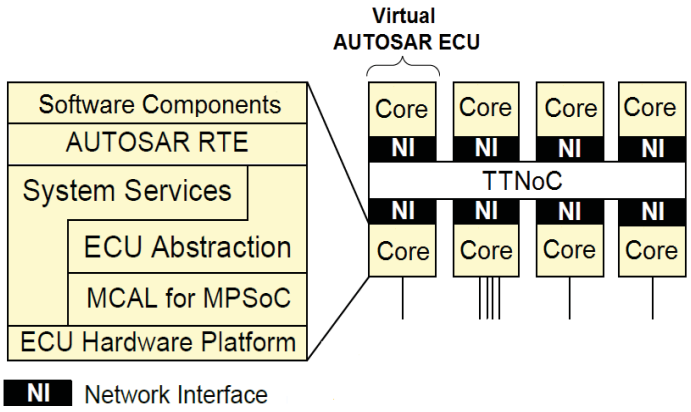

Fig. 1: TIMEA Platform

In [27] a GEM5 full system emulation and a CPU-GPU simulator are combined. This GEM5-GPU simulator implements a shared memory interface for the information exchange between the two simulation tools.

As presented earlier, the simulation of AUTOSAR-based systems implementing on-chip networks for multi-core communication is not supported by any described co-simulation environment. For this reason, this paper presents a simulation environment that combines a tool for AUTOSAR-based software simulation and natural environment simulation (VEOS) with an on-chip network simulation tool (SystemC) with support for time-triggered message-based AUTOSAR multi-core simulations.

III. Message-Based Time-Triggered Multi-Core Platform for AUTOSAR

The AUTOSAR MPSoC platform instantiates the AUTOSAR architecture on top of a time-triggered massage-based on-chip platform as presented in [10]. This TIme-triggered MEssage-based multi-core for AUTOSAR is called the *"TIMEA platform"*. Figure 1 depicts the TIMEA platform. This platform introduces application cores playing the role of virtual ECUs and a TTNoC [10] for the communication between the cores.

A. Virtual ECUs

In the TIMEA platform application cores are required to implement the automotive functionality. These application cores are configured based on the AUTOSAR ECU architecture, playing the role of virtual AUTOSAR ECUs.

Each virtual ECU hosts one or more AUTOSAR SWCs and is provided with a Run Time Environment (RTE) layer. The RTE layer makes the SWCs independent from the underlying platform and from the mapping to a specific virtual ECU [28]. The AUTOSAR BSW on each virtual ECU is extended to support time-triggered message-based NoC communication. Figure 2 shows the TTNoC modules added in the AUTOSAR BSW. New communication modules are integrated in the *"ECU communication layers"* for accessing the TTNoC. A

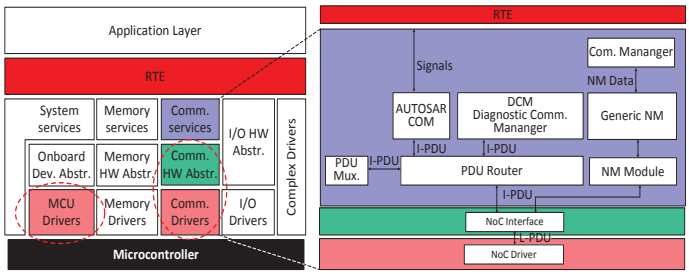

Fig. 2: BSW Modules for TTNoC Communication

NoC driver module for the Microcontroller Abstraction Layer (MCAL), a NoC hardware abstraction interface and extended COM service modules (COM module and PDU router) are integrated. The PDU router is used for routing NoC PDU messages to the COM module and the re-direction of COM PDUs to the NoC interface. The COM module maps PDU messages to RTE signals and vice versa.

The RTE provides signals mapped to the variables handled by the SWC ports. For SWC ports hosting a sender-receiver interface, the data transferred by the interface is mapped to a RTE signal. For client-server interfaces, RTE signals are mapped to the function arguments of the SWC ports.

Depending of the automotive application and its distribution on the TIMEA platform, remaining modules of the AUTOSAR ECU architecture can be configured on a specific virtual ECU (e.g. I/O abstraction modules, off-chip network COM modules). For example, in a virtual ECU hosting sensor/actuator SWCs the AUTOSAR I/O modules for hardware abstraction layer and MCAL must be integrated.

The VFB as defined by AUTOSAR, is available to the SWCs through a hierarchical platform. The communication hierarchy is performed as follows:

- *Communication between SWCs on the same virtual ECU.* Message exchange among SWCs on the same core is performed by the RTE on the specific virtual ECU.
- *Communication between SWCs on different virtual ECU.* Message passing between SWCs on different cores is performed through the extended COM modules and the TTNoC.

IV. Simulation Environment for the TIMEA Platform

The simulation environment for the TIMEA platform was developed performing a co-simulation of an AUTOSAR system and a model of the TTNoC. The AUTOSAR system was modelled and simulated with the dSpace AUTOSAR tools including the VEOS simulation framework, while SystemC is used for the simulation of the TTNoC. Both the on-chip time-triggered communication as well as the communication between SWCs within a virtual ECU are simulated. Developed local coordinators provide basic gateway functionalities between simulation systems. These local coordinators are responsible for the exchange of data and the synchronization of the simulation tools.

A. Simulation Tools

1) *VEOS Simulation Tool*: VEOS is a dSpace software for the simulation of ECUs and environment models on a host PC [29]. The following tools comprise the VEOS software:

- *The VEOS simulator* is the simulation platform for PC-based simulation of simulation systems. The experimental tool ControlDesk can access the VEOS simulator for testing ECU software [30].
- *The VEOS player* is the software tool for the integration of simulated ECUs and environment models into a simulation system and for the execution control of a simulation running in the VEOS simulator.
- *dSpace Target for offline simulation* is a Simulink Coder that allows to build environment models for simulation with VEOS. Environment models serve as virtual representations of an ECU environment during virtual validation scenarios.

2) *SystemC Simulation Tool*: SystemC is a widely used system-level modeling language for event-driven modeling [31], [32]. Timing accuracy in SystemC is ranging from untimed to cycle-accurate where precise temporal specifications and SystemC modules can be simulated to validate the behavior of the platform. Additionally, transaction level models (TLMs) are used in the simulation frameworks to enhance the overall simulation speed. TLMs capabilities such as interface-based communication, blocking and non-blocking process structures, bidirectional and unidirectional transactions inspired researchers to develop simulation frameworks owing to the adaptability and accuracy that SystemC/TLM provides [33].

B. Simulation System for Virtual AUTOSAR ECUs

SystemDesk allows the generation of simulated virtual AUTOSAR ECUs and the integration of them in a simulation system. This simulation system is built with SystemDesk, building an Offline Simulation Application (OSA) file [34]. The OSA file comprises binary files for each virtual ECU. The OSA file is needed to run the simulation system with the VEOS player. In a running simulation the virtual ECUs can be accessed by an experimental tool like ControlDesk for parameter calibration via the Universal Measurement and Calibration Protocol (XCP). XCP is *"a bus-independent, master-slave communication protocol to connect ECUs with calibration systems"* [35]. A2L files are used to access and interpret the data transmitted via XCP. A2L files describe the variables available for measurement and calibration [35].

Environment models representing part or all of an ECU environment in a simulation scenario are integrated to the simulation system. They are developed with Simulink and the dSpace simulation coder TargetLink.

Each virtual ECU and environment model is represented as an independent Virtual Processing Unit (VPU) by the VEOS player.*"VPU is a generic term for a part of a simulation system that can be run in an offline simulation by VEOS"* [34]. Using VEOS player, virtual ECUs and environment models can be interconnected through their VPU ports. VPU ports provide direct communication between VPUs without using

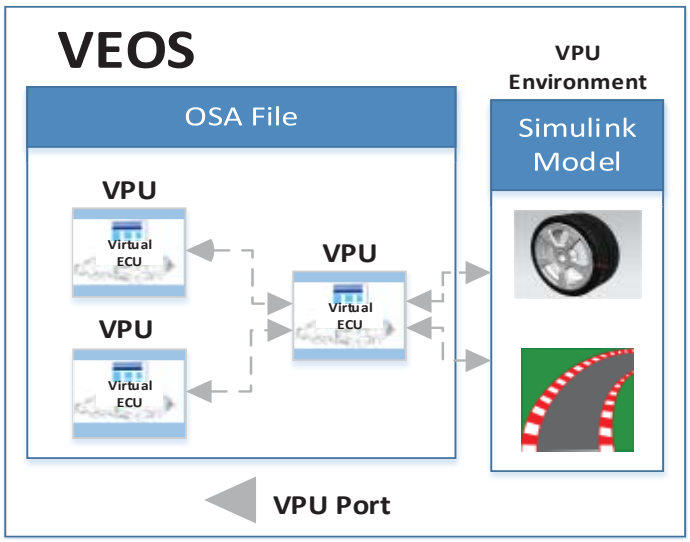

Fig. 3: VEOS environment

off-chip communication [29]. In the VEOS player VPU ports of the virtual ECUs are represented by the sensor/actuator SWC ports used by the I/O hardware abstraction layer, while VPU ports of the environment models are represented by simple Simulink input or output blocks. An architecture picture of the simulation environment and its components is depicted in figure 3.

During the simulation, an emulated AUTOSAR OS is used to run the simulated virtual ECUs on a host PC [29]. The main purpose of the OS is to invoke OS tasks. VEOS simulates correctly the order of OS task and function calls. The simulation time is the same for all the virtual ECUs and environment models in the simulation system. VEOS performs a zero execution time assumption for the simulation. The tasks are executed instantly in the virtual simulation time, which means *"all tasks are assumed to run on the virtual ECUs with an execution time of* $\Delta t_{PC} = 0ms$*"* [29].

Figure 4 depicts a scheduling example with three tasks. Task 1 is triggered periodically every $10ms$ and calls runnable R1. Task 2 is triggered by the OS every $5ms$ and calls runnables R2 and R3. Task 1 is called before task 2 because of its higher priority. Task 3 is an event-triggered task called sporadically by Task 2. Task 3 calls runnable R4. During the simulation of this scenario, VEOS triggers tasks every 5 ms. At $t = 0ms$, Task 1 is triggered and runnable R1 is called. Then Task 2 is triggered and runnables R2 and R3 are called. After runnable R3 returns, VEOS advances the simulated clock to $t = 5ms$ and triggers Task 2 again.

C. SystemC-based TTNoC Simulation

The simulation of the TTNoC is based on the work presented in [33]. TLM/SystemC is used to implement the communication behavior of the TTNoC. Time-triggered channels are expressed using SystemC channels, while message-based communication is managed using interfaces. Interfaces of the TTNoC are inherited from the sc_interface class of SystemC. Time-triggered channels are defined as bidirectional

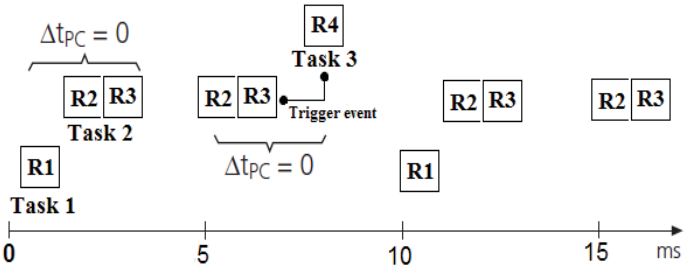

Fig. 4: VEOS scheduling example with three tasks

transactions and process structures are managed as blocking and non-blocking processes.

In the *network interfaces*, a communication interface is required to transmit time-triggered messages by providing procedures to send/receive messages to/from other cores. Moreover, the network interface is responsible for mapping the outgoing messages to the time-triggered communication according to the TDMA schedule.

The transmission of *time-triggered messages* is based on the following parameters: period, phase, message size, sender core ID and receiver core IDs. The formed time-triggered messages in the network interface are periodically transmitted based the time-triggered communication schedule. The phase of the message defines the start time with respect to the start of the period. The message size is determined by the message payload, and the routing requires the source and the destination core IDs of the message.

During the execution of the framework, a cyclic dispatcher method is called in the network interface to determine whether a time-triggered message has to be sent according to the predefined schedule. In this case, the formed time-triggered message is redirected to the corresponding channel port.

To receive a message at the cores, the network interface ports are notified by the time-triggered channels that a message is received. A receive function is triggered in the core that invokes the incoming message.

The TTNoC establishes the time-triggered schedule and the time-triggered communication channels. The schedule is a static configuration parameter and loaded at the beginning of the simulation. The TTNoC uses time-triggered channels that represent the temporal and spatial allocation of physical links of the simulated TTNoC. The time-triggered channels can simulate communication delays that represents the router delays of a real NoC by defining the configuration and schedule of the simulated use case.

D. Co-simulation Realization

Simulation systems presented in sections IV-B and IV-C are combined, resulting in the proposed simulation environment for the TIMEA platform. Figure 5 depicts the AUTOSAR MPSoC simulation system. This approach realizes an extension of VEOS, where simulated virtual ECUs are supported as simulated application cores for the TIMEA platform.

In the co-simulation system, integrated local coordinators implement a socket-based communication for the exchange of data between the virtual ECUs and their corresponding

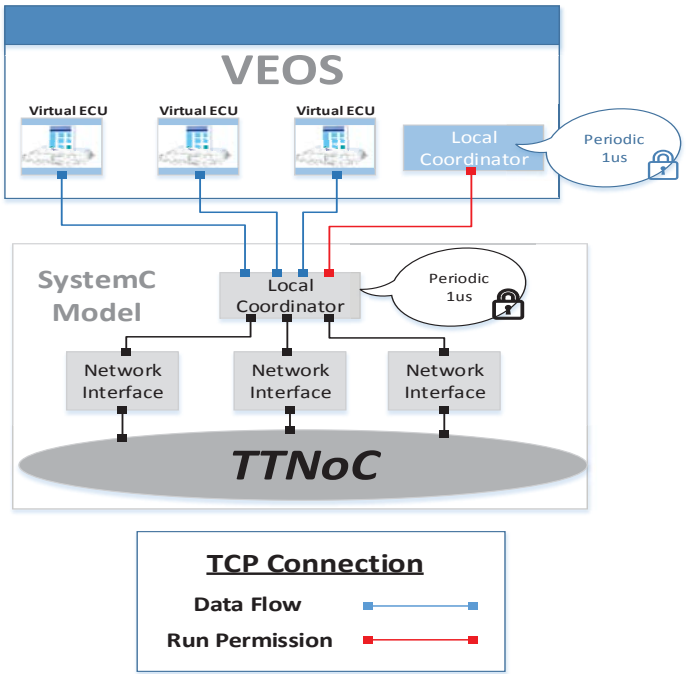

Fig. 5: VEOS/SystemC simulation model

network interfaces, and to synchronize simulation steps on both simulation systems. Virtual ECUs implement TCP clients for the exchange of data with the TTNoC simulation. The socket-based communication makes the co-simulation environment independent from the simulation tools location and their operating system.

A $1\mu s$ time interruption is used for the synchronization of the co-simulation. Since the RTE and the AUTOSAR BSW can only react to an event occurring in SystemC within the interrupt detection latency, we can use the minimum interrupt detection latency for the synchronization of the tasks between the simulation systems. We assume $1\mu s$ as the minimum interrupt detection latency. Also, the data exchange between the virtual ECUs and the TTNoC simulation is performed with a $1\mu s$ periodicity

The following message types are used for the communication between the two simulation systems:

- *D*ata Message: it represents a message sent from the virtual ECUs to the TTNoC local coordinator or vice versa. The structure of the data message contains the following fields:
  1) *M*essage Status: This parameter indicates whether the data message is empty or not.
  2) *S*ender ID: This contains the ID of the virtual ECU sending the message.
  3) *R*eceiver ID: Declares the destination virtual ECU ID.
  4) *S*WC Port: It indicates for which RTE signal the message is sent.
  5) *P*ayload: It contains the user data.

  Fields 4 and 5 represent a PDU message. These concepts will be used in the next section for explaining the BSW configuration of the virtual ECUs.
- *S*ynchronization Message: it represents a run permission sent between simulation systems. It is defined as follows:
  1) *C*reation Time: It represents the creation time of the run permission
  2) *R*un Permission Indication: It indicates that the simulation system can be run for $1\mu s$.

1) *Local Coordinator for VEOS simulation*: In the VEOS simulation system a local coordinator is used for the synchronization of the VEOS simulation system. The local coordinator uses a TCP client for the communication with the TTNoC simulation. This VEOS local coordinator implements a task with a period of $1\mu s$. Whenever a cycle is finished, the VEOS simulation system is paused and a synchronization message is sent to the TTNoC simulation. Thereafter the VEOS local coordinator waits for an incoming synchronization message from the TTNoC simulation in order to run another $1\mu s$ on the VEOS simulation.

2) *Virtual ECUs*: In the VEOS simulation system, virtual ECUs implement TCP clients for the exchange of data with the TTNoC simulation. The virtual ECUs implement a $1\mu s$ periodic task for sending *data messages* to the TTNoC simulation. Outgoing PDUs from the BSW on the virtual ECU are stored in a queue, and are sent to the TTNoC simulation in different *data messages*. A receiving thread is implemented on the virtual ECUs for receiving data messages from the TTNoC simulation. Once a *data message* is received by a virtual ECU, the *message status* is verified. If there is new incoming data from the TTNoC simulation, the *PDU message* is made available to the BSW on the virtual ECU in the next $1\mu s$ VEOS simulation. Empty *data messages* are needed because of the time-triggered behavior for data exchange between simulation systems and the event-triggered execution behavior of the simulation systems.

3) *Local Coordinator for SystemC simulation*: The TTNoC local coordinator implements a TCP server for the exchange of data with the VEOS simulation and for the synchronization of the TTNoC simulation system. The TTNoC local coordinator is used to control the TTNoC systemC model exactly in the same way as the VEOS local coordinator controls the VEOS simulation. Furthermore, the TTNoC local coordinator is used to receive the data from the virtual ECUs, to convert it to the adequate message format for the TTNoC model and to inject it into the respective network interfaces. Also, the TTNoC local coordinator accepts data from the network interfaces, reconverts the message format and sends it to the virtual ECUs.

The co-simulation is started initiating first the TTNoC simulation and then the VEOS simulation. The TTNoC local coordinator waits for the TCP connection of the virtual ECUs and the VEOS local coordinator before triggering the first TTNoC execution. The co-simulation system is controlled using the VEOS interface. It can be paused and resumed at any time and also different simulation steps can be configured.

An example of the synchronization of both simulation systems is shown in Figure 6, which resembles the scenario from Figure 4. Once the TTNoC receives a synchronization message from the VEOS local coordinator and the *data messages* from the virtual ECUs, the TTNoC local coordinator

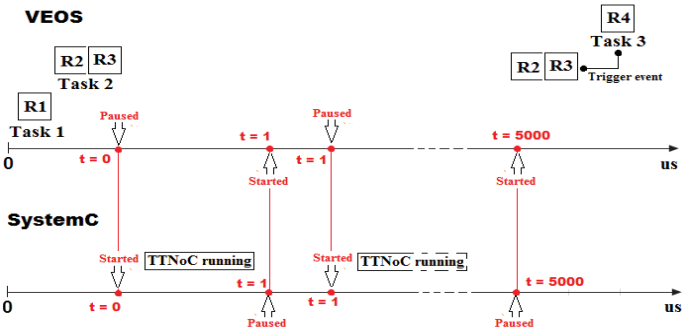

Fig. 6: VEOS/SystemC Co-simulation example with three tasks

verifies whether there is new data coming from the virtual ECUs. In case of new data, the *message payloads* of the respective network interfaces are updated (using *sender ID* and *Receiver ID*) and the TTNoC systemC model is run for $1\mu s$. Thereafter the simulation is paused and, in case of new data in the network interfaces, *message payloads* are packed into *data messages* and sent to the virtual ECUs. Finally a synchronization message is sent to the VEOS local coordinator.

V. Simulation Environment in the Development Process

For the development of the proposed simulation environment for the TIMEA platform, simulated virtual AUTOSAR ECUs are developed using the dSpace AUTOSAR solutions (SystemDesk and TargetLink) [30]. These tools offer a high level of maturity for designing AUTOSAR-based systems and strict compliance to the AUTOSAR workflow [36]. SystemDesk is a software tool that allows to model the software architecture of distributed automotive electronic systems consisting of AUTOSAR SWCs. The SWCs host the application functionality and are developed independently from each other and the specific ECU technology. For the functional behavior modeling the TargetLink AUTOSAR module is available, which allows the import and export of SWC descriptions. It also supports the AUTOSAR-compliant production code generation for the SWCs from the Simulink-Stateflow graphical environment [37]. Furthermore, SystemDesk is used to generate ECU configurations and the generation of simulated virtual ECUs for virtual validation scenarios [30].

A. Software Architecture Modeling

The modeling of the system architecture is performed using the dSpace system architecture tool SystemDesk. The software architecture is designed in terms of SWCs and the interaction between these SWCs.

A set of SWCs is defined, where each one encapsulates a functionality and the integration of all of them represents the specific automotive system. Once the SWCs are identified, input and output ports (called provided and required ports by AUTOSAR) must be added to each one of these SWCs. The AUTOSAR standard defines ports as the interconnection points between SWCs in the AUTOSAR architecture to indicate the data flow between these SWCs [38]. After defining the AUTOSAR SWCs and their ports, specific sender/receiver or client/server interfaces are defined. The interfaces define the information that is transported through the ports. An interface is assigned to every port, describing the data or operations provided or required by a SWC via its ports. Ports, which will be connected to each other, must have compatible interfaces.

Variable data prototypes and argument data prototypes are added to the sender/receiver and client/server interfaces respectively, defining the data transported by theses interfaces. Application data types, computation methods, data constraints and units are defined and assigned to the variable data prototypes and the argument data prototypes to describe the values that theses data types can contain. Application data types represent data types from the application point of view. Computation methods define the scaling conversion between the physical and the internal representation of the data [38]. Data-constraint elements restrict the physical range of values. Units represent the physical dimensions. Constant specifications of the application data types are assigned to the SWC input ports in order to define initial values [38].

An internal behavior is assigned to each SWC. The internal behavior provides means for formally defining the behavior of a SWC [38]. It is characterized by runnables, RTE events, exclusive areas and per instance memories. Exclusive areas represent critical sections that must not be interrupted by other runnables, while per instance memories are used for the exchange of array-based or structure-based variables between two runnables [38]. The initial modeling of the internal behavior of each SWC is performed as follows.

- Several runnable entities are defined for the SWC internal behavior.
- RTE events (e.g. timing event, data received event, etc) are defined, which will trigger the created runnable entities.
- Interrunnable variables are created, which are variables that are just used by the runnables of the specific SWC. A specific variable data type is assigned to each interrunable variable. Also initial constant values for the interrunable variables are defined.
- Data accessed by the defined runnable entities is specified. Interrunnable variables and variable data prototypes defined in the interfaces of the SWC ports are selected.
- A data type mapping set is created, which maps the application data types to implementation data types. Implementation data types specify implementation details such as SW base types or endianness.
- A constant specification mapping set is created, which maps constant specifications that have application data types to constant specifications that have implementation types. For this, constant specifications of the implementation types must be defined.

Once an internal behavior was already assigned to each SWC the descriptions [36] are exported as AUTOSAR xml files for behavior modeling with TargetLink.

B. Internal Behavior implementation for the AUTOSAR SWCs

The behavior modeling of the SWCs is performed using Simulink and the dSpace production code generation tool TargetLink. This software tool provides an AUTOSAR module for modeling, simulating and code generation of AUTOSAR SWCs.

The SWC descriptions of the SystemDesk defined SWCs are imported to the TargetLink Data Dictionary Manager. A Simulink frame model from the imported AUTOSAR SWC files is generated in a new Simulink model environment using the TargetLink frame model feature. The generated frame model works as a starting point for modeling the SWC implementation, having systems representing the SWCs and subsystems representing the runnables within the SWCs. The internal behavior of the SWCs is graphically developed using Simulink blocks and the special TargetLink blocks of the AUTOSAR TargetLink module.

Once the SWC behaviors are modeled, it is possible to perform a Model-in-the-loop (MIL) simulation of the graphical Simulink model to verify the model behavior. An AUTOSAR SWC implementation (C code) is generated for each SWC and a Software-in-the-loop (SIL) simulation can be performed. The new AUTOSAR SWC descriptions with the generated SWC implementations are exported from the TargetLink Data Dictionary Manager for generation of ECU configurations using SystemDesk.

C. Configuration of the Virtual AUTOSAR ECUs

The SWC descriptions provided by TargetLink (with the corresponding SWC implementations) are re-imported to SystemDesk for the completion of the AUTOSAR-based system. SystemDesk allows to define ECU instances that can have communication controllers and connectors for accessing a physical channel of a communication cluster in the context of a system [34]. Depending of the specific automotive functionality, a set of ECU instances is defined. SWC ports are connected each other to indicate the data flow between SWCs through the VFB. Once the ECU instances are identified, the SWCs are mapped to them in order to define the automotive system.

The BSW and RTE of the ECU instances can be configured and generated using SystemDesk. In the proposed simulation environment, ECU instances are configured to simulate the virtual AUTOSAR ECUs in the TIMEA platform. The configuration of a virtual ECU is depicted in figure 7.

The configuration of the BSW and RTE for each ECU instance is performed as follows.

- *ECU Configuration.* A single reduced ECU configuration (without off-chip network communications, external memory access and special complex driver support) is selected for each ECU instance. An I/O hardware abstraction module and a Data Access Point (DAP) module are added to the ECU configurations. The I/O hardware abstraction module is a BSW module for accessing I/O signals [39]. The DAP module provides the needed information to integrate the virtual ECUs in a simulation system [34]. The integration of the I/O abstraction module and DAP module is necessary for virtual ECU hosting sensor/actuator SWCs.
- *Generation of the I/O abstraction layer.* The I/O abstraction layer is configured, selecting the ports (with their corresponding data types) that will be used by the I/O hardware abstraction. A BSW component representation (with its corresponding ports and interfaces) of the I/O hardware abstraction is generated. The BSW component ports are connected to the application SWC ports. The I/O abstraction layer code is generated.
- *Generation of the DAP module.* The DAP module is configured automatically according to the I/O abstraction module configuration. The ECU I/O signals are mapped to DAP signals. A VPU port will be assigned to each DAP signal when a simulation system is run by VEOS player.
- *Configuration of the AUTOSAR operating system.* A reduced configuration of the AUTOSAR OS is used, which just takes care of the tasks to handle application SWC runnables and I/O abstraction runnables. OS tasks are created and the SWC runnables (from the application layer and the BSW abstraction layer) are mapped to them in order to define the execution context of the runnables. The task type, schedule, priority and execution order of the runnables are set for each OS task. A *NI_Task* with $1\mu s$ periodicity is created and the lowest priority is assigned to it. Also, a *COM_Task* with $1\mu s$ periodicity is created and the highest priority is assigned to it. These two tasks will be used later to instantiate COM service functions and NoC interface functions in the RTE. For completing the configuration OS events, OS alarms, OS application modes and OS counter are configured.
- *Generation of the RTE.* Based on the application software and BSW configuration of the ECU instances a RTE implementation is generated (C code) [39]. The generated RTE, besides interconnecting the SWCs, connects the application software to the generated I/O abstraction layer and the operating system.

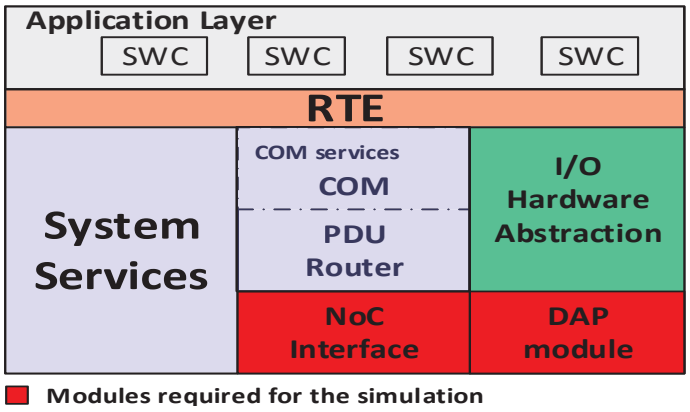

Fig. 7: Virtual ECU Configuration

Before building the ECU configurations and the generation of the virtual ECUs developed BSW modules are integrated to the ECU configurations for the completion of the BSW as shown in Figure 7. Developed NoC interface module and COM

service modules are integrated to each ECU configuration.

For the simulation of the virtual ECUs, the NoC interface module provides a TCP client to connect the virtual ECU with its corresponding network interface of the NoC simulation in SystemC as explained in section IV-D. It is responsible for the construction of the *data messages* sent to the NoC simulation. This is made by integrating incoming PDUs from the PDU router to a *data message* or configuring the *message status* to indicate that the *data message* is empty. The NoC interface module provides a sending queue wherein incoming PDUs from the COM module are stored by the PDU router.

The PDU router re-directs available *PDU messages* in the NoC interface module to the COM module and vice versa. The COM module takes incoming PDUs from the PDU router and puts their *payloads* available to the respective RTE signals. This is made by using the SWC Port identifier provided in the *PDU message*. Furthermore, the COM module takes updated data on the RTE signals and constructs PDU messages. The *PDU messages* are passed to the PDU router, which puts them in the sending queue used by the NoC interface module.

The generated RTE implementation (C code) is modified in order to adapt the integrated COM service modules and the NoC interface module. The TCP/client connection function of the NoC interface module is allocated to the *Rte_Start* task provided by the RTE. This task is in charge of allocating and initializing system resources and communication resources used by the RTE. The NoC interface function for sending *data messages* to the TTNoC local coordinator is allocated to the *NI_Task*. This means, after an execution of $1\mu s$ a *data message* is sent from the virtual ECU to the NoC simulation. Also, a PDU router function is allocated to the *COM_Task*. This function verifies whether there is a new *PDU* available in the NoC interface. Thus, before an execution of $1\mu s$ a new incoming PDU from the NoC simulation is passed to the PDU router. This is possible because of the configured priorities for *NI_Task* and *COM_Task*.

After this, the ECU configurations can be built, generating the virtual ECUs which are integrated to a single simulation system (OSA file) for being run in VEOS.

D. Configuration of the Local Coordinators

An ECU instance is configured to be the local coordinator for the VEOS simulation using SystemDesk. A *SYN_Task* of $1\mu s$ of periodicity is configured in the local coordinator. Before the generation of the VEOS local coordinator, a TCP/-client is integrated. The TCP/client connection function is allocated in the Start function of the VEOS local coordinator. Also, a function for sending *synchronization messages* is allocated to the *SYN_Task* as well as a function that waits for incoming *synchronization messages* from the TTNoC simulation. Thus, after an execution of $1\mu s$ the simulation system is paused till an incoming *synchronization message* arrives from the TTNoC simulation. After this, the VEOS local coordinator can be built and integrated to a simulation system of virtual ECUs.

In the TTNoC simulation system, a main process serves as a TTNoC local coordinator. A TCP/server is integrated. A function that waits for the connection of the virtual ECUs and the VEOS local coordinator is allocated to the TTNoC local coordinator. Thereafter, an infinite loop implements a function that waits for the *synchronization messages* from the VEOS local coordinator and the *data messages* from the virtual ECUs. In case of new *PDU messages*, this function updates the *payloads* of the respective network interfaces. After this function the TTNoC model is run for $1\mu s$ and the function for sending *synchronization messages* is called. In addition, this function also takes new payloads in the network interfaces (in case of new data), constructs the *data messages* and sends them to the virtual ECUs. Thus, after one loop execution the simulation is paused till a new *synchronization message* and new *data messages* arrive from the VEOS simulation.

Period	Phase	SenderCoreID	ReceiverCoreIDs
1ms	0s	0	1
1ms	1us	1	3
1ms	2us	2	1

TABLE I: TTNoC configuration

E. Configuration of the TTNoC Simulation

As discussed in section IV-C, the TTNoC requires configuration parameters to define its time-triggered communication table and the number of the virtual ECUs and their network interfaces. The time-triggered communication table is defined as a Comma-Separated Values (CSV) file that contains the message-based communication configuration.

As shown in the configuration table of the use case in the next section, time-triggered messages are scheduled to be transmitted periodically. The configuration table defines the time-triggered message period, phase, senderCoreID and ReceiverCoreIDs. The message phase parameter defines the start transmission time in relation to the defined period. In addition to that, the static configuration defines the sender virtual ECU ID and the receiving virtual ECUs.

In this paper the configuration of the CSV file is performed manually. The CSV file will be generated from an AUTOSAR data/information exchange format (e.g. FIBEX) to automate its configuration in future work.

VI. Example Use Case and Results

An Anti-lock Braking System (ABS) serves as an example use case to evaluate the developed AUTOSAR multi-core simulation environment. The anti-lock braking system was modelled as an AUTOSAR-based system consisting of five SWCs. The SWCs are distributed on a time-triggered multi-core processor with four virtual ECUs, where one of them hosts two SWCs.

The scheduling of the inter-core communication through the TTNoC is summarized in Table I. A period of $1ms$ is used for each message core and a unique phase is assigned to each message core. One *ms* is a reasonable period in the example, since the minimum task period in the developed ABS application is $5ms$.

A Simulink-based environment model representing the physical wheel, dynamics of the brake system hydraulic component, road characteristics and human braking behavior was

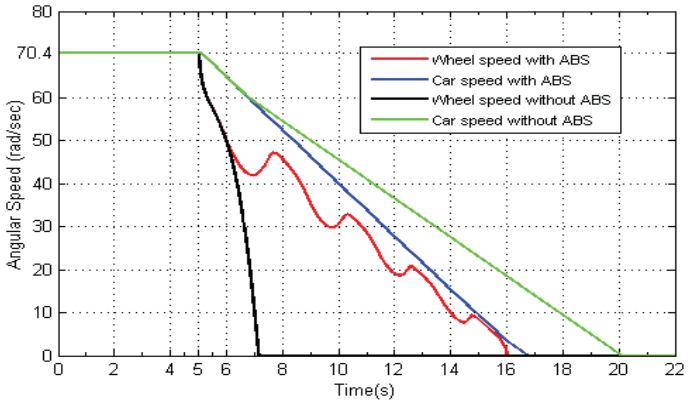

Fig. 8: Car speed and wheel speed

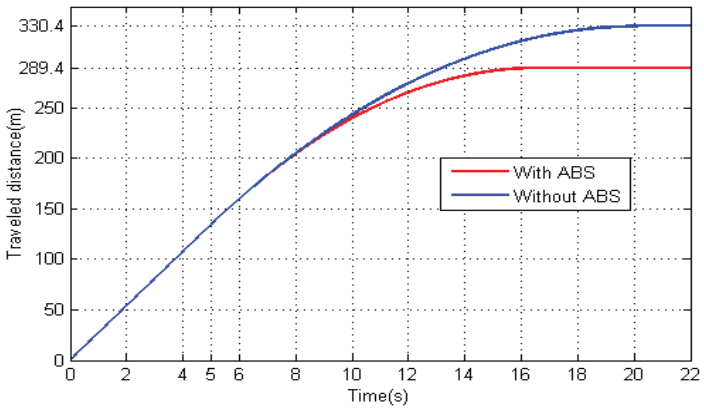

Fig. 9: Distance traveled

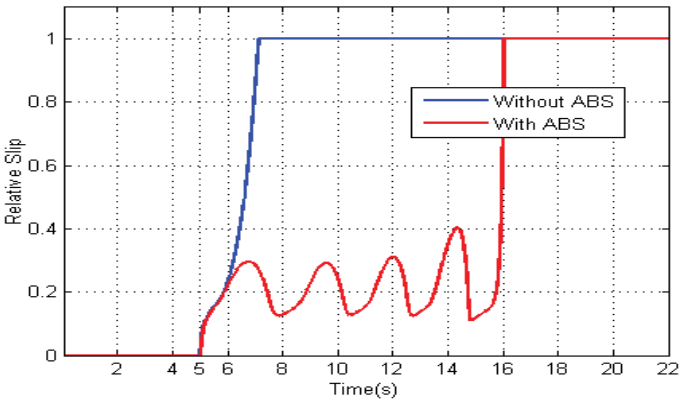

Fig. 10: Wheel Slip

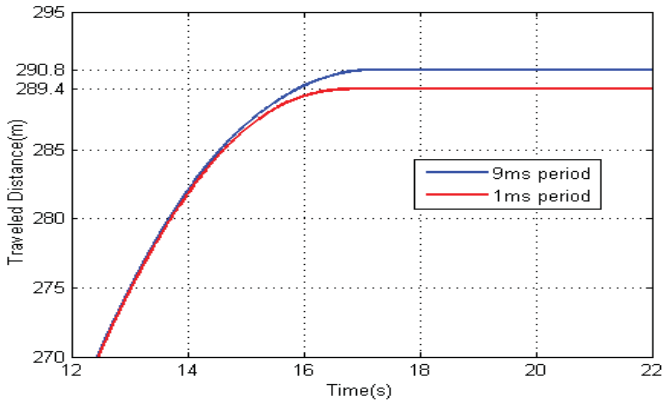

Fig. 11: ABS performance comparison of two different TTNoC configurations

integrated into the simulation system. The wheel has an initial angular speed of $70.4 rad/sec$, which is kept constant for $t = 5s$, when a hard braking is implemented by the driver.

Using the dSpace modular experiment and instrumentation software for ECU development, ControlDeskGeneration, the developed time-triggered multi-core ABS was tested. A $22s$ co-simulation time was performed during an overall of 6 minutes of real time. Figures 8 and 9 show the results with enabled and disabled ABS. Figure 8 compares the angular speed of the wheel with the angular speed of the car.

Figure 9 depicts the distance traveled by the car, showing a reduction of the braking distance of $41m$ when the ABS is enabled. Figure 10 shows the behavior of the relative slip suffered by the wheel without and under the ABS action. As a last result, Figure 11 compares the distance traveled by the car when changing the values of the message periods in Table I to $9ms$. In this scenario the TTNoC communication has a lower frequency than the execution of the software in the virtual ECUs. Thereby, we can observe the influence of the TTNoC on the ABS performance. A reduction of the ABS performance is visible, increasing the braking distance to $1.4m$ compared to the previous configuration of the TTNoC.

VII. Discussion and Conclusion

The integration of AUTOSAR-based system simulators and multi-core simulation tools is significant for the investigation of new emerging AUTOSAR MPSoC platforms. In this paper, a co-simulation environment supporting the integration of the AUTOSAR architecture with a time-triggered MPSoC platform was presented. The dSpace AUTOSAR solutions (SystemDesk and TargetLink) were used for the development of an AUTOSAR system based on application cores as virtual AUTOSAR ECUs in a MPSoC platform. The VEOS simulation framework is extended for the simulation of the multi-core system and its interaction with natural environment models, while SystemC is used for the simulation of a TTNoC model. We provide an interface for the data exchange between both simulation tools. The presented local coordinators control the communication and the synchronization between the AUTOSAR system and the TTNoC, implementing a socket-based communication.

The presented simulation environment allows the connection with the ControlDesk Generation tool for experimental tests. The simulation can be controlled, e.g. resumed and stopped at any time. Specific simulation steps can be configured and also a limited simulation time. Information from the virtual ECUs can be obtained during execution-time. Virtual

ECUs can be turned on/off at any time. Furthermore, the simulation results are obtained graphically and numerically.

The capability of the presented co-simulation environment was evaluated presenting an ABS use case. The influence of the TTNoC on the AUTOSAR-based system was analyzed. For an experimental test, the period of the time-triggered messages was configured with a lower frequency than the AUTOSAR software on the virtual ECUs. The performance of the automotive functionality has decreased as expected.

For future enhancement of the simulation environment, the implementation of the Functional Mock-Interface (FMI) standard [40] will be investigated. Since VEOS supports the integration and co-simulation of Functional Mock-Units (FMUs), it is planned to extend the VEOS local coordinator as an FMU wrapper for co-simulation with different on-chip network simulation tools. Furthermore, the development of a tool for automated configuration of the extended COM service modules and NoC interface module is planned.

ACKNOWLEDGMENTS

This work has been partially supported by the EU project DREAMS (No. 610640). Furthermore, gratitude to the dSpace Company for their software support.

REFERENCES

[1] Sonics, *Sonics u network technical overview*, 2002.

[2] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," *Design Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, Sept 2005.

[3] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone-a communication protocol stack for Networks on Chip," in *VLSI Design, 2004. Proceedings. 17th International Conference on*, 2004, pp. 693–696.

[4] Certification Authorities Software Team (CAST), "Position paper cast-32 multi-core processors," in *Tech*, 2014.

[5] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A Template for Composable and Predictable Multi-processor System on Chips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 2:1–2:24, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1455229.1455231

[6] C. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek, "The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, Sept 2012, pp. 105–113.

[7] R. Obermaisser and H. Kopetz, *GENESYS: A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Sdwestdeutscher Verlag fr Hochschulschriften, 2009.

[8] *AUTOSAR Architecture Overview, AUTOSAR Release 4.1*, AUTOSAR Consortium, 2014.

[9] *AUTOSAR Guide to Multi-Core Systems, AUTOSAR Release 4.1*, AUTOSAR, 2014.

[10] M. Urbina and R. Obermaisser, "Multi-Core Architecture for AUTOSAR based on Virtual Electronic Control Units," in *ETFA 2015. IEEE Conference on*, 2015.

[11] B. Huber and R. Obermaisser, "An ARTEMIS Cross-Domain Embedded System Architecture and Its Instantiation for Real-Time Automotive Applications," in *Proceedings of the 30th IFAC Workshopon Real-Time Programming and 4th International Workshop on Real-Time Software*, Poland, 2009.

[12] "AUTOSAR goes Multi-core the safe way." [Online]. Available: www.vector.com

[13] "Multi-core Automotive ECUs: Software and Hardware Implications." [Online]. Available: www.etas.com

[14] C. Harding, A. Griffiths, and H. Yu, "An interface between matlab and opnet to allow simulation of wncs with manets," in *Networking, Sensing and Control, 2007 IEEE International Conference on*, April 2007, pp. 711–716.

[15] A. Varga, "Using the OMNeT++ discrete event simulation system in education," *Education, IEEE Transactions on*, vol. 42, no. 4, p. 11, Nov 1999.

[16] "OCCN: The On-Chip Communication Network." [Online]. Available: www.occn.sourceforge.net

[17] "SIMICS: The Simics Full-system Simulator." [Online]. Available: www.windriver.com

[18] "SNIPER : The Sniper Multi-Core Simulator." [Online]. Available: www.snipersim.org

[19] Z. Zhang, E. Eyisi, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits, "Co-simulation framework for design of time-triggered cyber physical systems," in *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*, April 2013, pp. 119–128.

[20] B. Muller-Rathgeber and H. Rauchfuss, "A Cosimulation Framework for a Distributed System of Systems," in *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, Sept 2008, pp. 1–5.

[21] Z. Fang, Q. Min, K. Zhou, Y. Lu, Y. Hu, W. Zhang, H. Chen, J. Li, and B. Zang, "Transformer: A functional-driven cycle-accurate multicore simulator," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June 2012, pp. 106–114.

[22] M.-C. Chiang, T.-C. Yeh, and G.-F. Tseng, "A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 593–606, April 2011.

[23] K. Nakajima, T. Hieda, I. Taniguchi, H. Tomiyama, and H. Takada, "A Fast Network-on-Chip Simulator with QEMU and SystemC," in *Networking and Computing (ICNC), 2012 Third International Conference on*, Dec 2012, pp. 298–301.

[24] P. Wehner and D. Gohringer, "Parallel and distributed simulation of networked multi-core systems," in *System-on-Chip (SoC), 2014 International Symposium on*, Oct 2014, pp. 1–5.

[25] F. Cucchetto, A. Lonardi, and G. Pravadelli, "A common architecture for co-simulation of SystemC models in QEMU and OVP virtual platforms," in *Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on*, Oct 2014, pp. 1–6.

[26] M. Becker, U. Kiffmeier, and W. Mueller, "Heroes: Virtual platform driven integration of heterogeneous software components for multi-core real-time architectures," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, June 2013, pp. 1–8.

[27] J. Power, J. Hestness, M. Orr, M. Hill, and D. Wood, "GEM5-GPU: A Heterogeneous CPU-GPU Simulator," *Computer Architecture Letters*, p. 1, 2014.

[28] H. Heinecke, J. Bielefeld, K. Schnelle, M. Maldener, H. Fennel, O. Weis, T. Weber, L. Ruh, J. Lundh, T. Sandn, P. Heitkmper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, T. Scharnhorst, and B. Kunkel, "AUTOSAR Current results and preparations for exploitation," in *EUROFORUM "Software in the vehicle"*, 2006.

[29] *Virtual Validation Overview, Release 2014-B*, dSpace, 2014.

[30] *VEOS Player Document, Release 2014-B*, dSpace, 2014.

[31] R. Obermaisser and P. Gutwenger, "Model-based development of mpsocs with support for early validation," in *Proceedings of Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE*, 2009, pp. 2867 – 2873.

[32] "NOXIM : The NoC Simulator." [Online]. Available: www.noxim.org

[33] Z. Owda and R. Obermaisser, "Trace-based simulation framework combining message-based and shared-memory interactions in a time-triggered platform," in *IEEE First International Conference on Event-Based Control, Communication, and Signal Processing*, ser. EBCCSP '15. Krakow: IEEE, June 2015.

[34] *SystemDesk 4.x Guide, Release 2014-B*, dSpace, 2014.

[35] *ASAM MCD-1 XCP , V.1.2.0*, ASAM, 2010.

[36] *AUTOSAR Specification of Interoperability of AUTOSAR Tools, AUTOSAR Release 4.1*, AUTOSAR, 2014.

[37] *Target Link AUTOSAR Modeling Guide, Release 2014-B*, dSpace, 2014.

[38] *AUTOSAR Software Component Template, AUTOSAR Release 4.1*, AUTOSAR, 2014.

[39] *AUTOSAR Layered Software Architecture, AUTOSAR Release 4.1*, AUTOSAR, 2014.

[40] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clau, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. v. Peetz, S. Wolf, A. S. Gmbh, Q. Berlin, F. Scai, and S. Augustin, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models," in *In Proceedings of the 8th International Modelica Conference*, 2011.