Efficient Multi-Core AUTOSAR-Platform based on an Input/Output Gateway Core

Moisés Urbina and Roman Obermaisser
Institute of Embedded Systems - University of Siegen
Email:{moises.urbina, roman.obermaisser}@uni-siegen.de

***Abstract*—The AUTOSAR standard provides support for multicore systems since version 4. However, this AUTOSAR multicore version focuses on inter-core communication with a shared memory approach. In contrast, the paradigm of message-based network-on-chips provides multiples advantages for real-time embedded systems such as automotive electronics including better temporal predictability, fault containment and energy efficiency. In this paper we propose an efficient multicore architecture for AUTOSAR based on time-triggered network-on-chips and dedicated input/output cores. Additionally, a health monitoring service is integrated into the AUTOSAR ECU architecture in order to provide recovery actions in case of failures of the automotive application or the hardware of a specific core in the multiprocessor.**

The results demonstrate how the operating system overhead decreases considerably when using the defined input/output cores that serve as hardware accelerators for the AUTOSAR software. Also, the reliability of the system is improved significantly due to the implemented health monitoring service.

***Index Terms*—AUTOSAR μECU; Proxy Module; Health Monitoring**

I. Introduction

Multi-Processor System-on-a-Chips (MPSoCs) have become a suitable option for the development of real-time embedded systems. Specifically in the automotive domain the AUTomotive Open System ARchitecture (AUTOSAR) standard describes a multicore version of the AUTOSAR operating system [1] for the implementation of automotive systems.

In previous work a TIme-triggered MEssage-based multicore for AUTOSAR (TIMEA) [2] was presented, which uses a Time-Triggered Network-on-a-Chip (TTNoC) for the communication between the cores. This architecture introduces application cores configured based on the AUTOSAR ECU architecture playing the role of AUTOSAR Micro-Electronic Control Units (μECUs) on the same MPSoC. Thus, the benefits of TTNoC architectures in the context of temporal predictability and fault isolation are merged with the AUTOSAR system.

In this attempt we extend the AUTOSAR Basic Software (BSW) on the μECUs with a proxy module to connect them with a dedicated input/output core that serves as a hardware accelerator to the AUTOSAR application running on the μECUs. Additionally, a health monitoring service is integrated which provides recovery actions to the AUTOSAR software.

A simulation environment for message-based AUTOSAR multicore processors serves for the implementation of the input/output cores and the extended BSW modules. A realistic automotive use case demonstrates how the operating system overhead on the AUTOSAR μECUs decreases significantly when replacing I/O BSW functionalities with dedicated input/output cores. Additionally, the reliability of the AUTOSAR multicore system is improved by the introduction of the health monitoring service.

The remainder of this paper is organized as follows. In Section II we present an overview of related work in the area of I/O multicore solutions. The time-triggered multicore architecture for AUTOSAR is presented in Section III. Section IV describes the extended proxy modules and health monitoring service. The implementation using a simulation environment 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

In the last years I/O management in combination with multicore processors has been investigated. The description and discussion of existing I/O multi-core solutions is presented in this section. Additionally, the ARINC health monitoring service of the avionic domain is summarized.

A. I/O Multi-Core Solutions

A review of different solutions for maintaining coherency between caches and the data generated or consumed by I/O devices is presented in [3]. This work compares different approaches for data and I/O coherence with solutions trading off hardware versus software complexity depending on the application and the system characteristics.

A high performance multicore I/O manager for the Glasgow Haskell Compiler (GHC) is introduced in [4]. The so-called Mio manager eliminates the bottlenecks originating from a typical GHC I/O manager when implemented on a multicore processor. In [5] a virtual regionalized Network-on-Chip (NoC) is used to optimize the performance of the peripheral devices. This work presents an architecture consisting of a NoC with a mesh topology wherein the network is divided into several virtual regions taking advantage of the characteristics of the applications and their communication patterns to adapt to the I/O communication requirements.

A heterogeneous multicore embedded system with virtualization to improve security and isolation among virtualized environments is presented in [6]. This multicore embedded

architecture consists of an I/O management unit that enables the virtualization and provides support for a global coherent address space, flow isolation, security, resource management and runtime monitoring. In [7] a partition scheduler providing conflict-free I/O for multicore avionic systems is introduced. This work proposes a heuristic algorithm that prevents conflicts among I/O transactions from applications running in different cores.

B. ARINC 653 Health Monitoring

ARINC 653 [8] is an avionic standard for Integrated Modular Avionics (IMA) which defines an execution environment based on time partitioning for safety-critical avionics Real-Time Operating Systems (RTOS). It describes a general purpose application interface between the operating system and the application software.

This standard introduces a health monitoring service [9] which provides a framework to raise and handle alarms in a system consisting of three levels in a hierarchical fashion: process level, partition level and module level. A system health monitoring table defines the level of an error (module, partition, process) based on the error and the state of the system. Additionally, fault responses and recovery actions are defined depending on the error level.

Recovery actions for process level errors are defined by the application programmer, while recovery actions for the partition and module level are specified in the health monitoring configuration tables. The partitions can have separated configuration tables or share a common table.

C. Research Gap

The state-of-the-art does not provide I/O management as part of an AUTOSAR message-based multi-core platform. Additionally, the support for remapping between input/output cores and application cores (AUTOSAR μECUs) is required for the integration of the health monitoring service and recovery actions to the AUTOSAR multi-core system.

III. Message-based Multi-Core Architecture for AUTOSAR

The TIMEA platform [2] introduces a multi-core architecture for AUTOSAR based on a TTNoC for the inter-core communication, supporting fault isolation and temporal predictability. Application cores, so-called AUTOSAR μECUs, are in charge of performing the specific automotive application functionality while a TTNoC [10] serves for the exchange of messages.

A TTNoC supports bandwidths of several $Gbps$ and provides communication plans with precise phase positions of messages in the range of a few ns. Since most of the functionalities realized by the AUTOSAR BSW are designed for latency requirements with orders of magnitude no lower than $1ms$, a TTNoC for inter-core communication allows the introduction of dedicated input/output cores for replacing tasks originally handled by the BSW modules in the μECUs. In this paper we present an I/O abstraction core as an input/output core dedicated to the I/O functionalities specified by the AUTOSAR standard (e.g., PWM modulation, ADC functionality, etc). Such an input/output core provides multiple benefits to the AUTOSAR system, e.g., lower latency, higher throughput and less overhead of the AUTOSAR operating system. Using the TTNoC, Software Components (SWCs) that require access to sensors or actuators and that are located in different AUTOSAR μECUs can share the I/O abstraction core for accessing I/O functionalities. In the rest of this paper we use the term sensor/actuator SWC [11] to name these kind of AUTOSAR SWCs.

Figure 1 presents an efficient message-based multi-core architecture for AUTOSAR based on input/output cores. Additionally, complex driver functionalities can also be accelerated using input/output cores. In Figure 1, for instance a Fast-Fourier Transform (FFT) service is available to the μECUs as a dedicated input/output core instead of being present in the BSW of each μECU through the complex driver abstraction layer.

The proposed architecture improves the performance of the AUTOSAR application by reducing the operating system overhead on the μECUs. The architecture leads to a simplified implementation of the AUTOSAR BSW delegating costly I/O functionalities and complex driver functionalities to dedicated input/output cores. Also, the implementation of a complex driver as a dedicated input/output core leads to a significant reduction of the resource requirements in the AUTOSAR layer model which increases the efficiency of the driver functionality.

Based on the defined input/output cores and the extension of new BSW modules, the AUTOSAR application is improved by the following aspects:

- ***Lightweight AUTOSAR operating system.*** Expensive BSW functionalities are delegated to dedicated input/output cores and realized in hardware. Thus, μECU resources can be focused on the AUTOSAR application with a minimum impact on the operating system overhead caused by the proxy service modules.
- ***Efficient implementation of drivers.*** An input/output core dedicated exclusively to a device driver leads to a more efficient implementation of the driver, in particular for complex drivers with stringent timing requirements. The defined I/O abstraction core and an input/output core for a complex device driver serve as hardware accelerators for the μECUs.
- ***Recovery action at the μECU level.*** The integration of a health monitoring service allows to implement recovery actions in case of the failure of a sensor/actuator SWC. The presented health monitoring functionality can resume AUTOSAR runnables that replace a sensor/actuator SWC which accesses any I/O functionality.
- ***Recovery action at the MPSoC level.*** The input/output core remains operational despite the failure of a μECU, which hosts a SWC that uses the service of the input/output core. Furthermore, another μECU hosts a

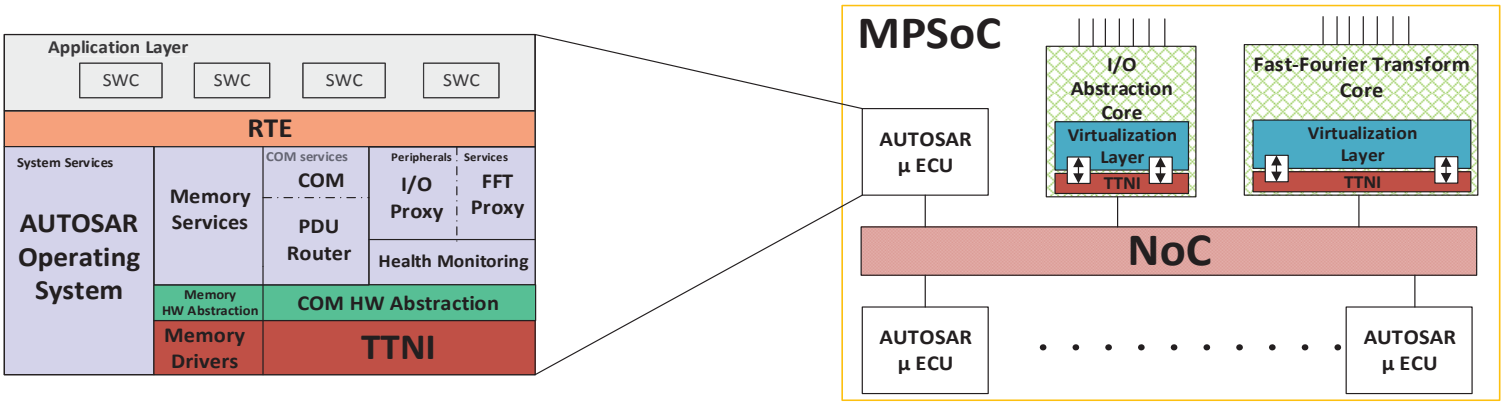

Fig. 1: AUTOSAR Multicore Architecture

standby SWC which takes over by interacting with the input/output core.

A. Time-Triggered Network-on-Chip

In the TIMEA platform we use a time-triggered NoC in the architecture such as AEthereal [12] or TTNoC [10] with inherent fault isolation and temporal predictability. A Time-Triggered Network Interface (TTNI) is available to each μECU and the input/output cores for accessing the TTNoC. The on-chip network gives support for different topologies (e.g., mesh, ring) and offers temporally predictable communication.

The communication between SWCs on different μECUs and ECU signals [13] sent between the dedicated input/output cores and the μECUs are mapped to time-triggered messages by the corresponding TTNIs of the cores. These messages are sent through the NoC according to a pre-configured time-triggered communication schedule, where each message is assigned to a specific time slot with respect to a global time base.

The communication between μECUs and dedicated input/output cores is supported with temporal predictability based on time-triggered channels. These time-triggered channels are used to establish a link between the senders and the receivers cores according to the configuration parameters of the NoC. The configuration parameters depend on the actual NoC. In case of TTNoC, the source-based routing paths need to be defined (e.g., periods, phases, etc), while in case of Aethereal, the routing configuration of the on-chip routers has to be defined.

B. AUTOSAR Micro-ECUs

The μECUs are application cores that are configured according to the AUTOSAR ECU architecture [14]. Each one of them is provided with an application layer consisting of AUTOSAR SWCs and a Run-Time Environment (RTE) to abstract in the automotive application from the BSW functionalities. As defined by AUTOSAR, the RTE serves as a unique interface for the SWCs to make them independent from the underlying hardware platform, in this case, a multi-core platform based on NoCs.

In the TIMEA platform, the BSW of the μECUs is extended with new BSW modules in order to support TTNoC communication. Special communication service modules (COM module, PDU Router) [2] serve as the interface between the RTE and the TTNI for connecting the μECU to the network. Thus, μECUs are able to use the TTNoC for the communication between each other.

In this paper we propose an efficient implementation of the I/O drivers by replacing the original AUTOSAR-defined BSW modules of the I/O abstraction layer by an input/output core (I/O abstraction core in Figure 1) dedicated to these functionalities. Instead of having an implementation of the whole I/O abstraction layer in the BSW of each AUTOSAR μECU, an I/O proxy module is defined. The I/O proxy serves to forward data prototypes [11] (sender/receiver interface) or argument prototypes [11] (client/server interface) sent by the sensor/actuator SWCs to the PDU router and thus, to the input/output core through the TTNoC. Additionally, ECU signals captured by the input/output core can be received by the μECUs and forwarded to the sensor/actuator SWCs using this I/O proxy module.

Moreover, an efficient implementation of a complex driver with a dedicated input/output core realization is possible. External peripherals requiring stringent time constraints that would need a complex driver realization in the BSW of the μECU can be accelerated by delegating the device driver to a dedicated core. In this case, a proxy module for matching ECU signals of the specific input/output core with the data/argument prototypes of their related sensor/actuator SWCs must be integrated into the BSW of the μECUs that require this service. As illustrated in Figure 1, a FFT proxy service module allows the interaction of a μECU with an input/output core dedicated to the FFT functionality. Also, the decision whether a device driver is accelerated by hardware is kept transparent to the RTE and the AUTOSAR application.

Furthermore, a Health Monitoring module is added to the BSW in the μECUs. This new module provides recovery actions to the AUTOSAR BSW by monitoring the data handled

by the sensor/actuator SWCs. In case a sensor/actuator SWC fails, the health monitoring service can activate a replicated SWC located in the same μECU which replaces the failed SWC. If there is no replica available in the same μECU, the health monitoring service makes aware the proxy module to send a notification message to the specific input/output core (e.g., I/O abstraction core) in order to indicate that a replica of the failed SWC located on another μECU must take over the access to the I/O functionality. The realization of the proxy modules and the health monitoring within the AUTOSAR architecture are explained in details in section IV.

C. Input/Output Cores for Hardware Acceleration

These input/output cores replace the functions normally provided by the I/O BSW abstraction layer [13] and the complex driver abstraction layer of AUTOSAR. The input/output cores support bidirectional communication via the TTNI that connects the core to the network-on-chip. Additionally, a virtualization layer abstracts the I/O functionalities from their underlying AUTOSAR μECUs and sensor/actuator SWCs that control the I/O ports. This layer uses the input port and output port provided by the corresponding TTNI for the exchange of data with the network supporting fault isolation.

ECU signals handled by the I/O ports (e.g., ADC converter) are mapped to time-triggered messages as part of the pre-configured time schedule of the network. The virtualization layer injects input signals via time-triggered buffers with a predefined time slot in order to be sent by the TTNI through the TTNoC. Additionally, each message received from the TTNI by the virtualization layer is matched to an output signal of a specific driver device, for example an output pin connected to a light switch.

Additionally, in case multiple μECUs need to access the same service provided by an input/output core dedicated to a complex driver (for example the FFT core of Figure 1), the virtualization layer also supports the sharing of the core among different μECUs based on static priorities defined at compile time. For example, if μECU 1 and μECU 2 want to trigger a FFT of an ECU signal during a certain time period, based on the priorities assigned to the service requests, the virtualization layer will forward them to the FFT service in a specific order. Hence, the FFT of the μECU signal with the highest priority finishes first.

D. Failure Assumptions

We define sensor/actuator SWCs and the μECUs as possible units of failures, i.e. Fault Containment Regions (FCRs). Based on this, we assume the following failure modes:

- ***Omission Failures.*** An omission failure is a transient failure where an FCR fails to send a message to the respective receiver, or where the receiver fails to receive a successfully sent message. This kind of failure can remain undetected by the system, if there is no a priori knowledge about the message periods or the maximum message interarrival times. An example of an omission failure would be a μECU that fails sending a message in its respective time slot through the TTNoC.
- ***Crash failures.*** This kind of failure represents an FCR which does not produce any outputs. An example would be a μECU that stops sending messages through the TTNoC. In contrast to the omission failures, a crash failure is a permanent failure.
- ***Value Failures.*** It represents an arbitrary value message failure which occurs in case the content of a transmitted message does not comply with the specification. An example would be a data prototype sent by a SWC which does not comply with its data constraint element. The data constraint element [11] is an AUTOSAR system specification used to restrict the range of possible values for the data handled by a SWC.

The virtualization layer of the input/output cores allows the use of redundancy for sensor/actuator SWCs at the MPSoC level. A threshold parameter ρ is configured in the virtualization layer, which is used to distinguish between omission failures (transient) and crash failures (permanent). The detection of a crash failure is defined by $N > \rho$, where N represents the actual number of consecutive omission failures detected by the virtualization layer, while ρ is the maximum number of omission failures detected before assuming that a crash failure occurred. Additionally, if a μECU sends a notification message indicating a value failure of a sensor/actuator SWC, the same expression is used by the virtualization layer to determine whether it is a transient failure or a permanent failure. The definition of the threshold parameter ρ depends on the specific design and the technology.

In case a permanent value failure is detected by the virtualization layer, this layer must provide access to another μECU hosting a replica of the sensor/actuator SWC which takes over interacting with the specific I/O functionality.

IV. AUTOSAR BSW Extension

This section provides a detailed description of the proxy service modules and the health monitoring functionality with the integration into the AUTOSAR ECU architecture. Figure 2 illustrates the communication flow between the BSW modules.

A. Proxy Functionality

In the AUTOSAR standard [13] the RTE is in charge of the communication between sensor/actuator SWCs and the BSW modules of the I/O hardware abstraction layer. The I/O abstraction layer consists of BSW SWCs with ports hosting AUTOSAR interfaces (sender/receiver or client/server). Thus, sensor/actuator SWC ports and I/O BSW SWC ports are connected though the RTE layer.

In order to make the application layer and the RTE independent from the acceleration of a device driver, the integration of a proxy module is realized as a BSW SWC, which also implements AUTOSAR interfaces to interact with the application layer. For instance, figure 2 shows an actuator SWC that is connected with a proxy SWC using a client/server interface.

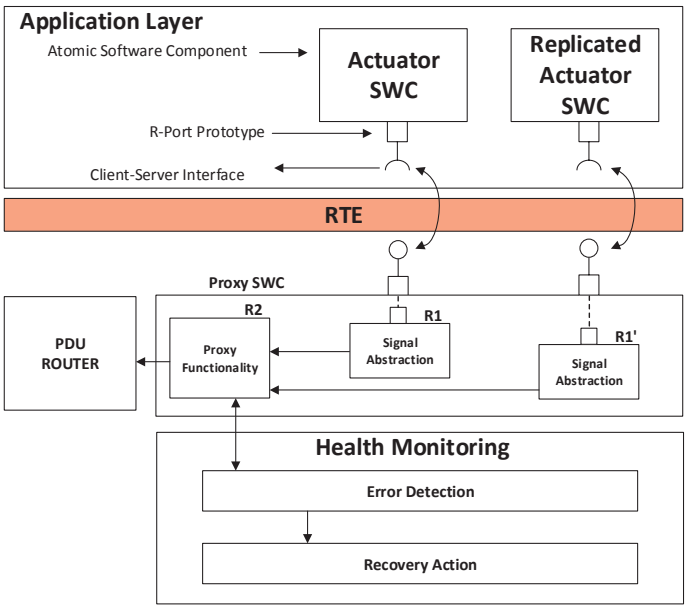

Fig. 2: BSW Modification

In this way, the decision whether a device driver is accelerated or not is kept transparent to the RTE and the application layer.

The internal behavior of the proxy SWC consists of two AUTOSAR runnables. One runnable (R1 in figure 2) is used for the signal abstraction. The aim of this runnable is to abstract in the data handled by the SWC ports from the physical layer values, mapping the proxy SWC ports to ECU signals. Thus, application designers do not have to be aware about implementation details of the device driver APIs and the units of the physical layer values. Thereafter, ECU signals are forwarded to the second runnable of the proxy SWC (R2 in figure 2).

The second runnable matches the ECU signals to Protocol Data Units (PDUs), and invokes the error detection service of the health monitoring module before forwarding the PDU signal to the PDU router. In case this function indicates that a value failure occurred, a recovery function of the health monitoring service is called. Thereafter, the two following scenarios are possible:

- In case a replicated actuator SWC depending on the same I/O functionality is located in the same μECU, the proxy function selects this ECU signal (coming from runnable R1' in figure 2) to be packed into the PDU instead of the original one.
- If no redundancy is available on the same μECU, the proxy functionality configures the PDU with a notification message to indicate to the input/output core that the specific μECU is not able any more to control the I/O functionality.

Additionally, the PDU router defined in [2] is extended with new routing tables to forward PDUs from the proxy module to the TTNI and vice versa.

B. Health Monitoring

The health monitoring module enables recovery actions based on redundancy of actuator SWCs at the μECU level. Figure 2 illustrates how the health monitoring module is integrated into the AUTOSAR architecture and shows its internal structure.

The error detection block provides a detection mechanism based on the data constraint elements associated to the ECU signals in order to detect value failures of the sensor/actuator SWCs. An example is an argument data prototype that exceeds the range of allowed values defined by its data constraint element. This block provides an error detection function, which is used by the proxy functionality as described earlier. In case a failure is detected, the error detection block invokes a recovery action.

The recovery action block hosts health monitoring tables mapping possible software errors to recovery actions. In this work we propose as the only recovery solution the activation of a replicated actuator SWC located in the μECU as mentioned previously. For this, callback functions are used to resume operating system tasks (at run time) that host the runnables which represent the replicated actuator SWC. As defined by AUTOSAR [1], callback functions provide the capability to trigger SWCs that are outside of the AUTOSAR BSW. Thus, a replica of the actuator SWC does not increase the operating system overhead, since the tasks are set up in suspended state [1] as defined at compile time.

V. Implementation Using a Simulation Environment

The realization of the TIMEA platform with the extended BSW modules for the μECUs and the dedicated input/output cores is presented in this section. The co-simulation environment introduced in [15] is used in the implementation. This environment consists of the dSpace VEOS simulator for the AUTOSAR software simulation, and SystemC/TLM for the simulation of the TTNoC [16] and the input/output cores. The coordination technique presented in [17] is used to accurately coordinate both simulation tools. Figure 4 illustrates an architecture picture of the co-simulation environment and its components.

The previously mentioned tools are used for the following reasons:

- ***VEOS Simulator.*** This environment is the dSpace software tool for the simulation of AUTOSAR Electronic Control Units (ECUs) (so-called virtual ECUs) in the automotive industry. Additionally, VEOS allows the simulation of environment models representing the physical environment that interacts with the virtual ECUs during a simulation scenario. Both, virtual ECUs and environment models, are represented by VEOS as Virtual Processing Units (VPUs) [15] which can be connected to each other (see figure 3). During a VEOS simulation, an AUTOSAR Operating System (OS) is emulated for a PC-based simulation, which is in charge of invoking the OS

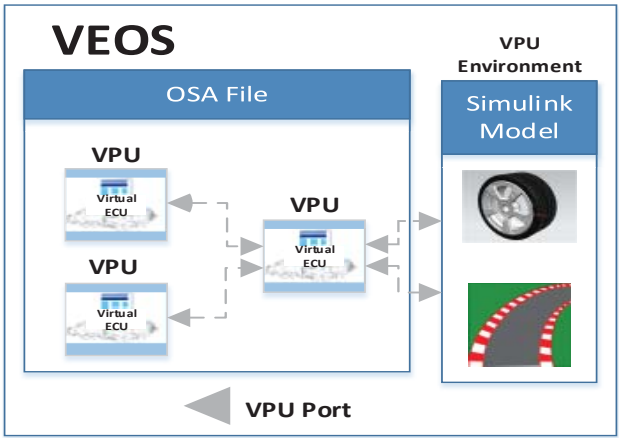

Fig. 3: VEOS Environment

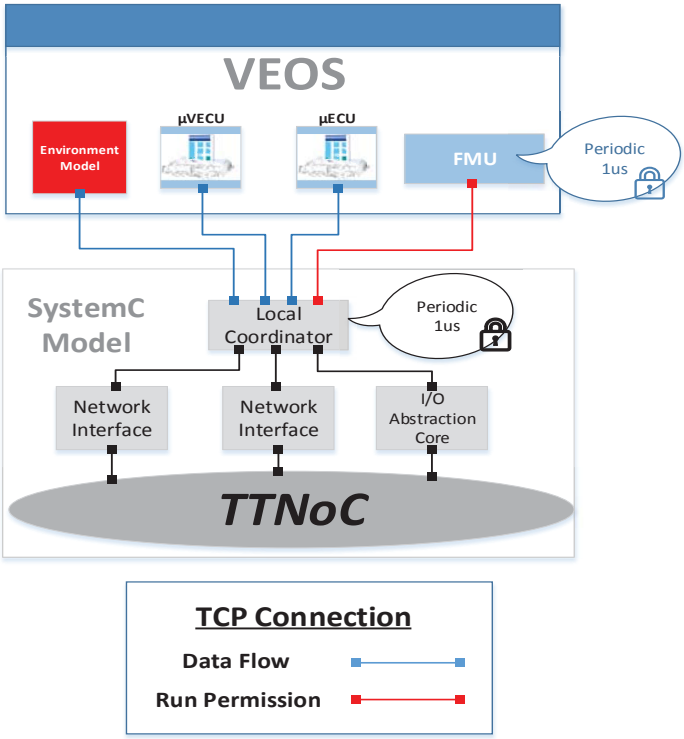

Fig. 4: Co-simulation Environment

tasks and function calls. Using an experimental tool such as ControlDesk the VEOS simulator can be accessed in order to test the virtual ECUs.

- ***SystemC Simulation Tool.*** SystemC is a system-level description language based on C++, which is widely used for event-timing modeling [18] [19]. In systemC the timing accuracy is ranging from untimed to cycle-accurate where precise temporal specifications and SystemC modules can be simulated to validate the behavior of the platform. Furthermore, for the development of simulation frameworks SystemC provides transaction level models (TLMs) to enhance the overall simulation speed. The ability and accuracy that SystemC/TLM provides, have inspired researchers in the development of multiples simulation frameworks in the last years (e.g. [20], [21] and [22]).

The dSpace tools SystemDesk and TargetLink are used for the development of the AUTOSAR application, and the configuration and generation of the μECUs with the integrated proxy module and the health service module. Also, a Functional Mock-up Unit (FMU) [23] and a local coordinator are used for the synchronization and the exchange of data between the two simulation systems.

A. Implementation of the Input/Output Cores

The simulation framework presented in [16] serves for the implementation of the defined input/output cores (i.e., I/O abstraction core and any complex driver dedicated input/output core). Input/output cores are developed as SystemC-based cores that run together with the TTNoC on the same SystemC simulation. For instance figure 4 introduces the I/O abstraction core as a SystemC-based core in the TTNoC simulation.

The framework allows the configuration of the on-chip communication through a pre-defined schedule (CSV file), wherein the period and the phase of each message are set. In this work we extend the on-chip schedule in order to provide information about the priorities of the μECUs that try to access a service of an input/output core (as explained in the last section). Additionally, in order to provide redundancy at the MPSoC level, new time-triggered messages are included in the on-chip schedule to support this specific functionality as well as the configuration of the error threshold parameter ρ defined in the previous section.

The virtualization layer is developed on top of the corresponding TTNI of the input/output core in order to abstract in the driver functionality from the NoC-based multicore platform implementation. Thus, this layer provides variables that map the receiving port of the TTNI directly to ECU signals handled by the input ports of the core (e.g. analog input, ADC, etc), while the data handled by the sending port of the TTNI is mapped to the ECU signals handled by the output ports of the core (e.g. PWM, analog output, etc). Since the extended on-chip schedule provides information about the messages used for supporting redundancy on a different μECU, the virtualization layer allows the mapping of multiple time-triggered messages to a single ECU output-signal. Thus, in case of a crash failure on the μECU responsible of the first assigned time-triggered message, the virtualization layer automatically switches to the second time-triggered message from the μECU wherein redundancy is provided.

Furthermore, the priorities assigned to the μECUs are used by the virtualization layer in order to provide the μECU holding the highest priority a faster access to an input/output core service.

B. Implementation of the Micro-ECUs

The AUTOSAR architecture tool SystemDesk is used for the configuration and the generation of the μECUs in the simulation using the VEOS environment. A complete description

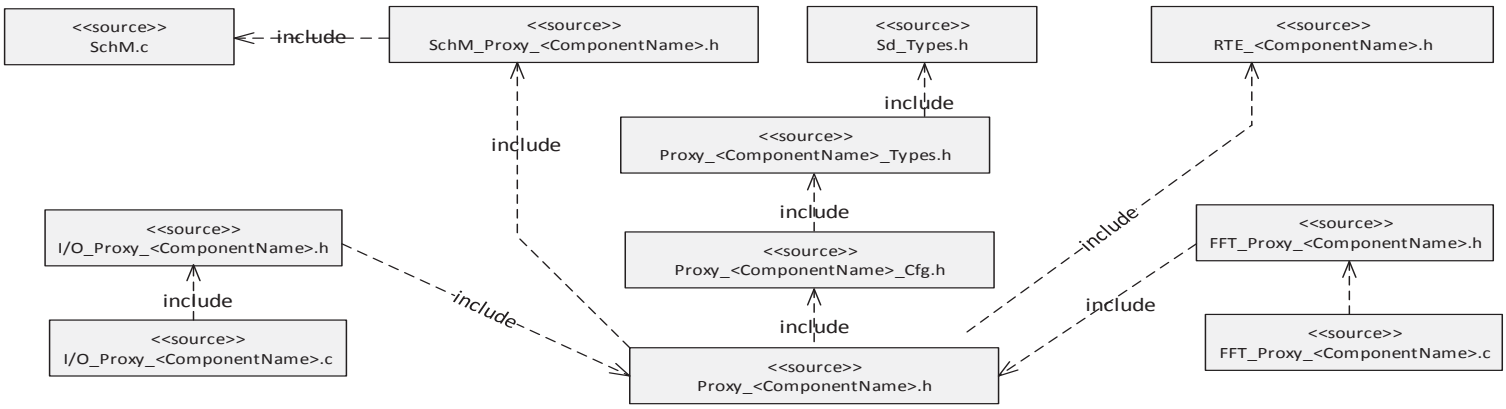

Fig. 5: File Structure of Proxy integration

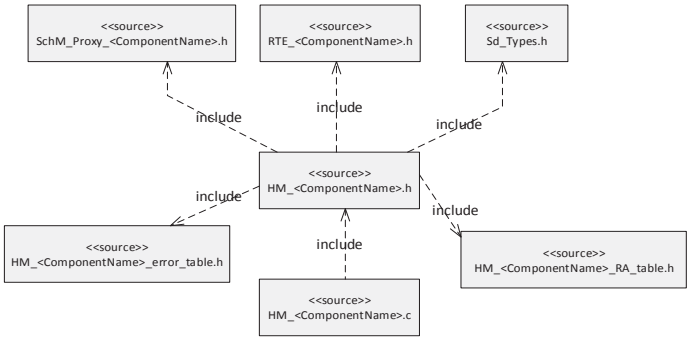

Fig. 6: File Structure for the integration of the Health Monitoring module

of the development process of an AUTOSAR system based on μECUs is presented in [15]. In this section we focus on the integration of the proxy module and health monitoring module into the BSW of the μECUs.

The SystemDesk tool allows the definition of an AUTOSAR system consisting of AUTOSAR SWCs and the interaction between each other implementing AUTOSAR interfaces (i.e., client/server or sender/receiver). Additionally, the internal behavior of these SWCs is modelled using the dSpace code generator tool TargetLink in combination with Simulink.

Additionally, using SystemDesk simulated ECUs can be defined and the AUTOSAR SWCs can be allocated to these simulated ECUs. These simulated ECUs play the role of the μECUs in our simulation of the AUTOSAR MPSoC platform.

The simulated μECUs are configured based on the software architecture picture depicted in figure 1. A single reduced ECU configuration without off-chip network communications, external memory access and special complex driver support, is selected for the configuration of each μECU. Additionally, an empty I/O hardware abstraction layer (without c-code implementation) is added to each ECU configuration. Thus, BSW SWCs are generated automatically together with their interfaces that connect them to the sensor/actuator SWCs at the application layer. This allows us to develop manually the internal behavior (C-code) of these BSW SWCs to implement the proxy modules as defined in the last section.

The OS of each μECU is configured and consists of OS tasks for hosting the SWC runnables that are located in the μECU. Also, OS tasks for hosting the proxy runnables are defined. Moreover, for completing the configuration OS events, OS alarms, OS application modes and OS counters are set up.

Furthermore, based on the application SWCs and the BSW configuration of the μECUs the RTE is automatically generated. This generated RTE interconnects SWCs and connects the application layer with the generated BSW SWCs and the AUTOSAR OS.

Before the generation of the simulated μECUs, the COM service modules (i.e. COM module, PDU router), an NoC interface module, the health monitoring service module and the implementation of the proxy SWCs are integrated into the BSW of the μECUs. The mentioned NoC interface module is a specific simulation module for connecting the μECU with the corresponding TTNIs of the SystemC simulation.

The generated RTE implementation is modified in order to integrate the COM module, PDU router, NoC interface, the proxy modules and the health monitoring service. A detailed description of the integration of the COM service modules and the NoC interface module[1] can be found in [15].

The runnables representing the internal behavior of the proxy SWCs are allocated to their specific tasks (defined previously) in the RTE implementation. Additionally, the initialization functions of the proxy implementations are allocated to the start function of the ECU State Manager [24]. Figure 5 presents the file structure including the dependencies of the proxy implementation with the other BSW modules.

In the implementation of the health monitoring service, the callback functions are used for the recovery actions. These recovery actions are implemented according to the $Rte\_Call\_ < p > \_ < o >$ API [25] of the RTE in order to enable safe configuration of the AUTOSAR services as specified by the standard [13]. The initialization function of the health monitoring module is allocated to the *Rte_Start* task provided by the RTE implementation. This task is in

[1]The NoC interface hosts a TCP client for the exchange of data with the TTNoC simulation

charge of allocating and initializing system resources and communication resources used by the RTE. Figure 6 depicts the file structure presenting the dependencies of the health monitoring implementation with the other BSW modules.

After this, the ECU configurations are built, generating the simulated μECUs, which are integrated to a single simulation system for being run in VEOS.

C. Co-simulation Coordination

As mentioned previously, the simulation technique presented in [17] is used for the simualtion coordination of both simulation systems. This work introduces a local coordinator and an FMU for the co-simulation and synchronization of the simulation steps of the AUTOSAR simulation running in VEOS with the SystemC simulation (see figure 4). TCP/IP serves for the communication between each other. In the VEOS simulation TCP clients of the μECUs, environment models and the FMU serve for the exchange of data with the SystemC simulation wherein the local coordinator provides a TCP server.

Based on the Functional Mock-up Interface (FMI) standard [23], the FMU is configured as an FMU wrapper with a fixed communication step size ($hc_i = \kappa$) according to the minimum interrupt detection latency. Since the VEOS simulation can just react to an event occurring in the TTNoC simulation within the interrupt detection latency, we use the minimum interrupt detection latency as the communication step size parameter defined by the FMI standard. $hc_i = 1\mu s$ is assumed as the minimum interrupt detection latency in the experimental evaluation. In [17] this parameter selection is explained in more detail.

Two types of messages are used for the communciation between the two simulation systems:

- ***Data Message.*** It represents a message sent from the VEOS simulation to the SystemC local coordinator and vice versa. It is used for the exchange of data between the μECUs and their TTNIs in the TTNoC simulation, and the data exchange between the environment models and the input/output cores (e.g. I/O abstraction core).
- ***Synchronization Message.*** It serves for the synchronization of both simulation systems. It is used by the FMU wrapper and the local coordinator to coordinate the simulation steps on both simulation systems according to the fixed communication step size.

VI. Usecase & Evaluation

A. Use Case

For the evaluation, we define a use case consisting of a TIMEA platform of 8 cores as depicted in figure 7. An ABS functionality is distributed over three AUTOSAR $\mu ECUs$, two of them hosting 2 SWCs and one of them hosting just one. The $\mu ECU1$ has a sensor SWC which interacts with the I/O abstraction core receiving an ECU signal from the velocity sensor through an ADC input port. Also, $\mu ECU3$ hosts an actuator SWC responsible for computing the control signal of the ABS functionality (PWM Controller in Figure 7), which is

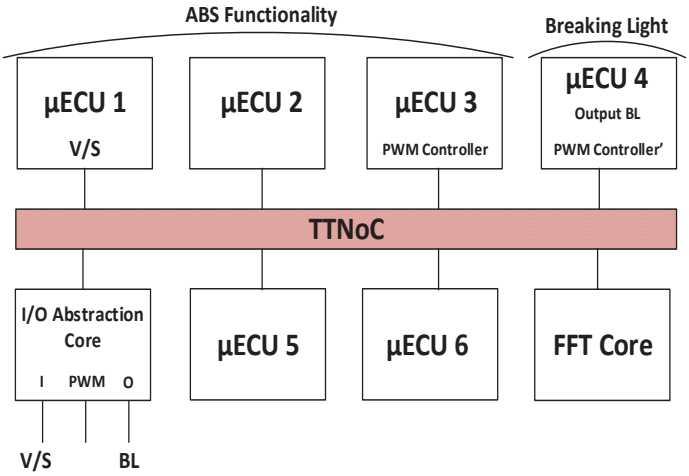

Fig. 7: USE CASE

sent by the NoC to the I/O abstraction core. We inject a fault in the error detection block which consists of an unexpected value for the ABS control signal exceeding the limits assigned by the data constraint element. As a recovery action the health monitoring service in $\mu ECU3$ must resume tasks for activating a replica of the actuator SWC that is located in the same μECU in order to use redundancy at the μECU level. Additionally, a braking light indicator functionality is performed by one SWC hosted by the $\mu ECU4$. In order to provide redundancy at the MPSoC level, a replica of the actuator SWC for the ABS control signal is also allocated to $\mu ECU4$.

Furthermore, $\mu ECUs$ 5 and 6 represent automotive functionalities which require an FFT service each for a certain time, which is made available to the $\mu ECUs$ through a dedicated input/output core (FFT core in figure 7). The FFT application is based on [26], which performs a 16-point FFT computation of the input signal. Different priorities are assigned to each μECU in order to avoid conflicts in accessing the FFT core. $\mu ECU6$ possesses a higher priority than $\mu ECU5$, which is pre-configured at compile time in the virtualization layer of the FFT core as explained in section III. Thus, the values obtained from a specific μECU are queued by the virtualization layer in a 16 point buffer before passing them to the FFT application. The virtualization layer forwards the buffer that corresponds to the higher priority in case two $\mu ECUs$ require the FFT service at the same time.

As shown in figure 7, the I/O abstraction core provides one ADC input pin for receiving the velocity sensor signal, a PWM pin for forwarding the ABS control signal and an output pin connected to the braking light (BL). We configure the parameter ρ (defined in section III) in the virtualization layer of the I/O abstraction core with a granularity of $\rho = 3$. This seems to be a reasonable selection of ρ since the minimum task period in the presented use case is $5ms$.

In our simulation scenario, a hard braking is implemented by the driver with an initial speed of $88km/h$. For this, a Simulink model is used to simulate the human braking behavior, the road characteristics, the physical wheel and

Period	Phase	SenderCoreID	ReceiverCoreIDs	Latency
1ms	2ns	I/O Core	µECU1	50ns
1ms	54ns	µECU1	µECU2	50ns
1ms	106ns	µECU2	µECU3	50ns
1ms	158ns	µECU2	µECU4	50ns
1ms	210ns	µECU3	I/O Core	50ns
1ms	262ns	µECU4	I/O Core	50ns
1ms	314ns	µECU4	I/O Core	50ns
1ms	366ns	µECU5	FFT Core	50ns
1ms	418ns	µECU6	FFT Core	50ns

TABLE I: NoC configuration

the dynamics of the brake system hydraulic component. The simulation time was 18s.

B. Results and Discussion

The NoC configuration implemented for the TIMEA platform is presented in table I. Also, this table provides the resulting latencies for each message sent through the TTNoC. The simulated TTNoC does not accept the sending of simultaneous messages at the same time through the NoC, having a processing time of $50ns$ from sender core to receiver core. For this, phases were selected with a difference of more than $50ns$ to avoid delays because of contention in the network.

RTE interventions serve to access the RTE internal communication of sender receiver and client server interfaces in order to read and to modify the data elements and operation arguments transmitted between the interfaces. Also, the status return values of RTE API functions can be modified using RTE interventions. During a simulation, ports can be stimulated or error states can be injected to test the behavior of the SWCs. We use RTE interventions to perform fault injections in order to change the values of the data prototypes handled by the AUTOSAR SWC ports. Thus, the recovery solutions provided by the health monitoring service can be tested.

Figure 8 compares the behavior of the vehicular speed, while figure 9 illustrates the braking distance behavior. In both figures, curve 1 shows the braking behavior of the car when no fault is injected. Curve 2 shows the braking behavior when using fault injection on the $\mu ECU3$ to test the recovery action, while curve 3 represents the braking behavior when turning off the $\mu ECU3$, which means requesting the virtualization layer of the I/O abstraction core to take the ABS control signal from another core where redundancy is provided ($\mu ECU4$ in this case). As shown in figure 2, the ABS performance in curve 2 and 3 has decreased by 1.3% and 4% respectively compared with curve 1 but still offering a significant difference of the braking distance in comparison with curve 4 when no ABS functionality is provided.

In order to measure the impact of the OS overhead resulting from the implementation of the I/O abstraction core we also run our simulation scenario keeping the I/O functionalities on the BSW of each μECU. Table II compares the number of task invocations realized by the OS with and without a dedi-

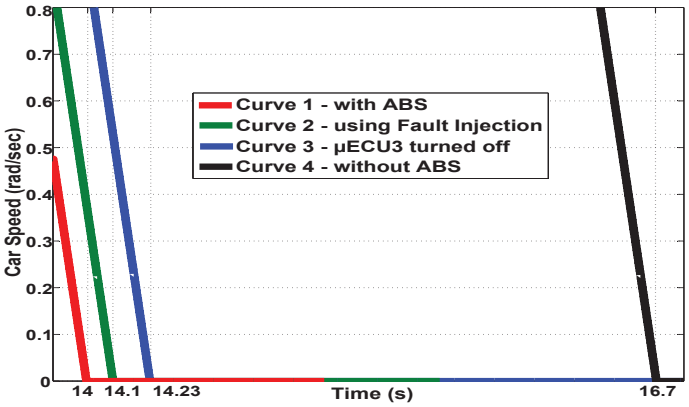

Fig. 8: Car Speed

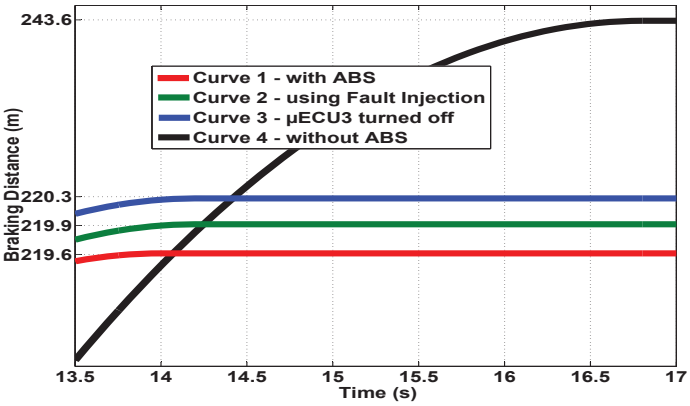

Fig. 9: Braking Distance

cated input/output core. As shown in table II, in $\mu ECU1$ the OS overhead decreases by 27.82% when the I/O abstraction core is implemented, while $\mu ECU3$ and $\mu ECU4$ exhibit an overhead reduction of 18.77% and 41.02% respectively. In $\mu ECU3$ the integrated health monitoring service represents 9.84% of the OS overhead, which can be justified since the reliability of the system has being improved significantly.

Furthermore, $\mu ECU5$ and $\mu ECU6$ require the service provided by the FFT core at three different times during our simulation scenario. Table III presents the latency results obtained by the interaction between the $\mu ECUs$ and the FFT core. Both $\mu ECUs$ access the FFT core using the TTNoC at their pre-defined time slots. The results presented in table III demonstrate how the $\mu ECU6$ with the highest priority always needs the minimum time ($17ms$) for processing the FFT.

	Task Invocations		
Core	With I/O Core	Without I/O Core	Health Monitoring Service
µECU1	4377	7450	----------
µECU3	7576	9327	897
µECU4	7346	12456	----------

TABLE II: Task invocations

Core	FFT initial Request Time (µs)	Proccessing Time (µs)	Ending Time (µs)
µECU5	5000366	32000	5032366
µECU6	5000418	17000	5017418
µECU5	10000366	17000	10017366
µECU6	11000418	17000	11017418
µECU5	15010366	22000	15032366
µECU6	15000418	17000	15017418

TABLE III: FFT timing accesses

VII. Discussion and Conclusion

In this work we presented an efficient time-triggered multi-core architecture for AUTOSAR (called TIMEA) based on input/output cores. Costly I/O functionalities of the AUTOSAR BSW I/O abstraction layer are delegated to a dedicated input/output core (I/O abstraction core) which is made available to the AUTOSAR $\mu ECUs$ through a NoC. Additionally, complex driver functionalities can also be accelerated with a dedicated input/output core. In this work an FFT core was used as an example for a complex driver input/output core.

A new proxy BSW module is introduced in order to make the $\mu ECUs$ able to access the input/output cores. This proxy module maps ECU signals handled by the sensor/actuator SWCs to PDUs, which are sent and received using the TTNoC and the COM modules of the NoC communication. Furthermore, the BSW in the $\mu ECUs$ is also extended with health monitoring which provides recovery actions based on SWC redundancy in case of faults affecting the application core functionality.

A simulation environment for the message-based AUTOSAR multi-core platform was used in the implementation. The results demonstrate how the OS overhead of the $\mu ECUs$ is reduced significantly when the I/O functionalities are delegated to dedicated input/output cores. Additionally, the integrated health monitoring service was tested using an ABS use case. The presented simulation scenario shows how the performance of the ABS functionality is preserved under fault occurrences due to the implemented health monitoring service.

Acknowledgment

This work has been supported in part by the European FP7 project DREAMS under grant agreement 610640. Furthermore, we would like to thank the dSpace Company for their software support.

References

[1] *AUTOSAR Operating System, AUTOSAR Release 4.1*, AUTOSAR, 2014.

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

[3] T. B. Berg, "Maintaining i/o data coherence in embedded multicore systems," *IEEE Micro*, vol. 29, no. 3, pp. 10–19, May 2009.

[4] A. R. Voellmy, J. Wang, P. Hudak, and K. Yamamoto, "Mio: A high-performance multicore io manager for ghc," *SIGPLAN Not.*, vol. 48, no. 12, pp. 129–140, Sep. 2013. [Online]. Available: http://doi.acm.org/10.1145/2578854.2503790

[5] C. Jian, J. Guanjun, L. Jingwei, W. Chao, and C. Tianzhou, "Optimistic peripheral devices performance by virtual regionalized network-on-chip," in *Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on*, Sept 2009, pp. 650–655.

[6] G. Kornaros, M. D. Grammatikakis, and M. Coppola, "Towards full virtualization of heterogeneous noc-based multicore embedded architectures," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, Dec 2012, pp. 345–352.

[7] J. E. Kim, M. K. Yoon, R. Bradford, and L. Sha, "Integrated modular avionics (ima) partition scheduling with conflict-free i/o for multicore avionics systems," in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, July 2014, pp. 321–331.

[8] ARINC, *Specification 651: Design Guide for Integrated Modular Avionics*, Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401, 1991.

[9] H. Zhang, S. Wang *et al.*, "Testing method of integrated modular avionics health monitoring," in *CHEMICAL ENGINEERING TRANSACTIONS CET, AIDIC publication*, 2013.

[10] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz, "The time-triggered system-on-a-chip architecture," in *IEEE Int. Symp. on Industrial Electronics, 2008*, talk: IEEE Int. Symp. on Industrial Electronics.

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

[12] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," vol. 22, no. 5, Sept 2005, pp. 414–421.

[13] *AUTOSAR Specification of I/O Hardware Abstraction, AUTOSAR Release 4.1*, AUTOSAR, 2014.

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

[15] M. Urbina, Z. Owda, and R. Obermaisser, "Simulation environment based on systemc and veos for multi-core processors with virtual autosar ecus," in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*.

[16] 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.

[17] M. Urbina, H. Ahmadian, and R. Obermaisser, "Co-simulation framework for autosar multi-core processors with message-based network-on-chips," in *Industrial Informatics INDIN, 2016 IEEE International Conference on*.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] T. Blochwitz, M. Otter *et al.*, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models," in *In Proceedings of the 8th International Modelica Conference*, 2011.

[24] *AUTOSAR Specification of ECU state manager, AUTOSAR Release 4.1*, AUTOSAR, 2014.

[25] *AUTOSAR Specification of Run Time Environment, AUTOSAR Release 4.1*, AUTOSAR, 2014.

[26] "Fast Fourier Transform based on SystemC." [Online]. Available: https://github.com/systemc/systemc-2.2.0/tree/master/examples/sysc/fft