Timing Simulation of Interconnected AUTOSAR Software-Components

Matthias Krause, Oliver Bringmann
FZI Forschungszentrum Informatik
Haid-und-Neu-Strasse 10-14
76131, Karlsruhe, Germany
mkrause@fzi.de, bringmann@fzi.de

André Hergenhan, Gökhan Tabanoglu
Carmeq GmbH
Carnotstrasse 4
10587, Berlin, Germany
andre.hergenhan@carmeq.com
goekhan.tabanoglu@carmeq.com

Wolfgang Rosentiel
Universität Tübingen
Sand 13
72076, Tübingen, Germany
rosenstiel@fzi.de

Abstract

AUTOSAR is a recent specification initiative which focuses on a model-driven architecture like methodology for automotive applications. However, needed engineering steps, or how-to-come from a logical to a technical architecture respectively implementation, are not well supported by tools, yet. In contrast, SystemC offers a comprehensive way to simulate, analyze, and verify software. Furthermore, it is even able to take the timing behavior of underlying hardware and communication paths into account. Already at a first glance, there are many similarities with respect to the modeling structure between the both concepts. Therefore, this paper discusses approaches on how to use SystemC during the design process of AUTOSAR-conform systems.

1 Introduction

During the last years, automotive software-related systems have been steadily increased with respect to their functionality. Unfortunately, the likewise increased complexity of the systems caused many problems in terms of stability, error-proneness, performance, reusability, modularity, processes, and the like, as well.

Currently, AUTOSAR (automotive open system architecture) [1] tackles the problems. AUTOSAR is an international development partnership consisting of a multitude of car manufacturers, suppliers and tool vendors, defining concepts and workflows, how electronic automotive software-related systems can be formally specified and processed. Lately, first specification results of AUTOSAR have been published. Thus, carmakers, suppliers and tool vendors start to transfer results into practice. However, only few experiences exist handling AUTOSAR-like methodology in software development. In this context, this paper shall examine on how SystemC can support its implementation.

Regardless of the completeness, Section 2 outlines the main concepts and methods of AUTOSAR. Similarly, Section 3 explains the main concepts of SystemC. Following this, Section 4 exposes the obvious affinities of both concepts with respect to the inherent modeling patterns. This leads to studies in Section 5 regarding possible scenarios using SystemC in the development of AUTOSAR systems. Section 6 shows an use case and Section 7 concludes the paper and surveys possible future work. Finally, Section 8 refers to related work.

* This work has partially been supported by the BMBF project VISION under grant number 01M 3078.

2 Concepts of AUTOSAR

2.1 Methodology

AUTOSAR will revolutionize the art of software development in automotive application domains. Instead of the current state-of-the-art ECU-centric development approach, AUTOSAR focuses on the entire system. A fundamental feature is the separation of application and infrastructure which allows for a model-driven architecture like methodology, i.e. a platform independent software development of functionality. Applications can exist and communicate independently of a particular infrastructure and mapping onto ECUs in an environment called Virtual Functional Bus (VFB).

However, AUTOSAR comprises even more: it specifies methodologies and workflows on how to come from the system living in the VFB to software running onto particular ECUs and a three-layer ECU architecture.

The ECU architecture consist of an application layer, a middleware layer, called Run Time Environment (RTE), and the infrastructure layer, called Basic Software (BSW). Assuming that the application elements of the application layer behave exactly the same like in the VFB, then RTE and BSW implement the VFB for an particular ECU.

2.2 Software Component Template

Properties of AUTOSAR applications are described with a specific language, called AUTOSAR Software Component Template (as part of the entire AUTOSAR metamodel). In general, the AUTOSAR Software Component Template is arranged into three parts: regarding the structure, the behavior and the implementation of models.

Referring to the structure, applications encapsulate functionalities within software-components, whereas software-components are available in two flavors: atomic software-components and compositions. Atomic software-components contain single threads of execution, so-called RunnableEntities, and are later-on mapped onto particular ECUs. Compositions are means to structure atomic software-components and can therefore form hierarchies. It is remarkable that the top-most hierarchy level then represent the entire system. Components communicate via ports which are typed by interfaces. General communication paradigms between entities are sender-receiver or client-server communication. In this sense, interfaces are either described for sender-receiver or for client-server communication.

The behavioral section contains the aforementioned RunnableEntities (RE). RunnableEntities can either be triggered by so-called RTEEvents[1] and will just be executed, or they can wait (inside) for an RTEEvent. In the first case, they are referred to as category 1 RunnableEntities; in the latter one as category 2. Common RTEEvents are TimingEvents (a cyclic trigger), DataReceivedEvent (a trigger caused by the reception of data) or OperationInvokedEvent (a trigger caused by an request for an service).

Beyond, there are concepts for implicit reading and writing of data, data consistency mechanisms or mode managements, and others. Please, have a look on the documents published at [1] for further readings. In addition, [14] explains the methodology of the definition and generation of data exchange formats in AUTOSAR.

3 Concepts of SystemC

3.1 SystemC Language

SystemC is a language that bridges hardware and software. Essentially, it is a C++ class library that extends C++ with hardware modeling concepts. This includes hardware related communication, i.e. connecting different modules by signals or complex communication protocols, as well as a timing concept. SystemC also provides a simulation kernel for concurrent hardware simulation.

In SystemC a design is partitioned and encapsulated into modules. Each module can contain other modules and act as a hierarchical element, processes that describe the functionality, and ports through which a module communicates with other modules. A process can be suspended by calling a wait statement with a certain wait condition and is simulated concurrently to other processes by the SystemC simulation kernel. Interfaces contain a set of operations which are accessed by a port and implemented within a channel. These communication operations could be primitive (e.g. signal, fifo) or complex (e.g. specific communication protocol).

3.2 Methodology

Starting at a high level of abstraction, typically at transaction level, is a key feature of the SystemC based design concept. The OSCI Transaction Level Working Group has defined different levels of abstraction for transaction-level modeling. These levels are introduced in [4] and [5]. The highest level is CP - Communicating Processes. At this level, the behavior is partitioned into a network of parallel processes exchanging data through point-to-point connection. By introducing timing annotations (CPT - Communicating Processes with timing), timing behavior can be considered already at this early stage. The next level, PV - Programmers View, is much more architecture specific. Bus or NoC models are instantiated to act as transport mechanisms between the model components. The models are sequenced but untimed. PVT - Programmers View with timing is annotated with estimated multi-cycle timing information. At CC - Cycle Callable level, the system behavior includes cycle-true details and communication models are protocol-true.

[1] The term might be misleading. In fact, VFBEvent would have been a better name for it.

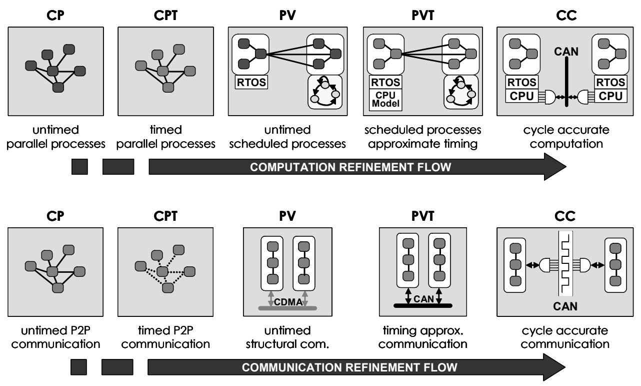

CP
CPT
PV
PVT
CC
RTOS
RTOS
CPU Model
CAN
RTOS CPU
RTOS CPU
untimed parallel processes
timed parallel processes
untimed scheduled processes
scheduled processes approximate timing
cycle accurate computation
COMPUTATION REFINEMENT FLOW
CDMA
CAN
CAN
untimed P2P communication
timed P2P communication
untimed structural com.
timing approx. communication
cycle accurate communication
COMMUNICATION REFINEMENT FLOW

Figure 1. The Communication and computation refinement flow in SystemC steps through the different abstraction levels

In [11] and [12] this concept is picked up and an automated refinement approach is presented that systematically refines the design from CP down to CC level. Within the refinement framework computation and communication are separated from each other. The systematically refinement starts with the communicating processes and than stepwise adds further system information like scheduling behavior and timing behavior of the target architecture. This flow is represented in Figure 1. The refinement requires information about the desired target architecture (RTOS, communication architecture) which is described in XML. The XML description of the communication architecture is conform to IP-XACT [20].

4 Analogies between AUTOSAR and SystemC

There are a lot of affinities between the AUTOSAR Software Component Template and the SystemC language.

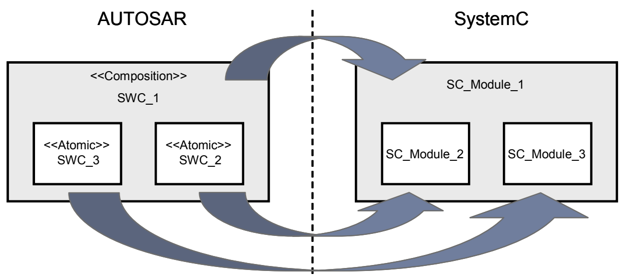


Figure 2. Analogy regarding AUTOSAR software-components or compositions and SystemC modules

In terms of structure, both have entities containing behavioral elements and both can form ordered hierarchies. Therefore, AUTOSAR software-components can generally be represented by SystemC modules (SC_MODULE) as shown in Figure 2. Since there is no extra hierarchical element in SystemC, AUTOSAR compositions map to SC_MODULE, too.

Figure 3 shows analogies regarding communication. Both have the concept of ports: AUTOSAR ports, regardless their direction (provided or required), have their counterparts in sc_port. The same holds slightly true for interfaces which type ports. The marginal difference here is that AUTOSAR explicit states a specific kind of interface: sender-receiver or client-server. SystemC, however, hides this detail within the concept of channels. In both technologies, ports can be exported throughout the entire hierarchy. AUTOSAR realizes this by the concept of delegation connectors, whereas SystemC provides the specific sc_export construct.

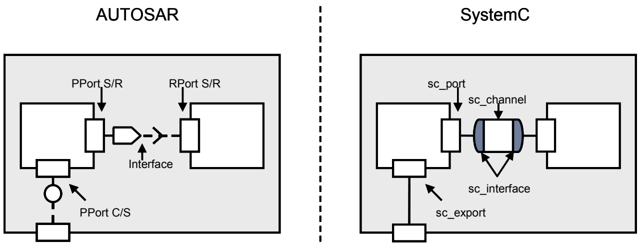


Figure 3. Analogy regarding AUTOSAR ports and interfaces and SystemC ports, interfaces and channels

Figure 4 depicts the context of the schedulable (triggerable) entities. In general, AUTOSAR RunnableEntities can directly be mapped to SystemC processes. Furthermore, since AUTOSAR and SystemC distinguish two types of schedulable entities, a one-to-one mapping of AUTOSAR RunnableEntites of category 1 to SC_METHODs and of AUTOSAR RunnableEntities of category 2 to SC_THREADs is possible. That also means inherently that both offer the opportunity of triggering schedulable entities via events and can wait for an trigger.

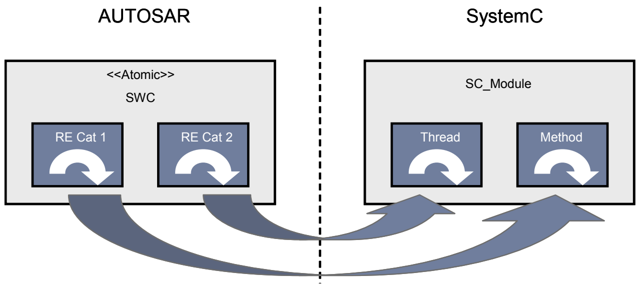


Figure 4. Analogy regarding AUTOSAR RunnableEntities (RE) and SystemC methods and threads

5 SystemC within AUTOSAR design process

5.1 Benefit of methodology

To solve the aforementioned problems, AUTOSAR has specified concepts, infrastructure, and workflows, but does not consider simulation. System simulation is a required step to evaluate the system in an early design phase. Early simulation helps to find errors and bottlenecks within the design resulting in decreasing development time by preventing possible re-designs. Moreover, simulation should consider timing behavior that has a strong impact on the behavior of the entire system with respect to performance and possible errors. This is also not specified by AUTOSAR until now.

It is true that state-of-the-art tools (e.g. MATLAB/Simulink [13]) already support simulation, but they only consider single applications and not the entire interconnected system. Moreover, the timing behavior of the entire infrastructure including communication architecture can not be simulated and evaluated by these tools.

In contrast, SystemC tackles the problems. A particular value of SystemC, with respect to the design at system level, is the ability to design and model the functionality of embedded distributed systems as well as the required target architecture and infrastructure within one design language. This enable simulation and evaluation of a software application on its underlying target architecture and infrastructure respectively, both specified in SystemC.

Additionally, SystemC introduces a simulation concept for the designs and provides a simulation kernel as well. This concept includes timing notations and timing behavior which is not part of AUTOSAR. In brief, SystemC offers those features that are not defined within AUTOSAR or supported by additional simulation tools.

Therefore, sharing both methodologies will lead to an increase in value. The benefit is to enable simulation of interconnected AUTOSAR software-components by integrating timing behavior already at a high level of abstraction to the communication as well as to the application, and hence to detect timing caused errors at an early design time. Figure 5 shows, how the complete flow from the AUTOSAR environment onto a SystemC virtual prototype looks like. As a result, simulation and analysis of the entire system is feasible and the results can influence the AUTOSAR configuration files.

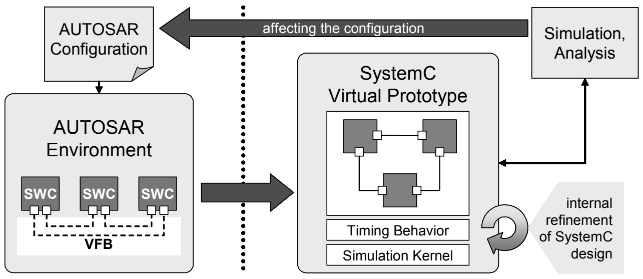


Figure 5. Mapping AUTOSAR software-components onto a SystemC virtual prototype

To this end, mapping of AUTOSAR software-components onto the SystemC simulation environment, as discussed in Section 4, is a necessary step.

Possible scenarios on how SystemC can be used in the development of AUTOSAR systems are discussed below.

5.2 From AUTOSAR VFB view to SystemC CP

Like in the SystemC-based methodology, also the AUTOSAR methodology has different views which can be compared to abstraction levels. Considering applications independent of a particular infrastructure and mapping onto ECUs is the highest level in AUTOSAR.

Similarly, the SystemC based design approach starts at CP. Systems modeled at CP level are still architecture and implementation independent, and there is no arbitration of the data communication. In contrast to the widespread view of SystemC as a hardware modeling language, this level of abstraction is completely independent of the partitioning into hardware and software. Communicating processes describe software processes which are later completely mapped to a target processor.

As a consequence, an AUTOSAR design can be transformed into an equivalent SystemC design at CP level. At SystemC CP, the design can be simulated (still untimed) by the provided simulation kernel, or it can be refined with additional information to allow a timed simulation. This step is discussed in Section 5.4.

Please note that in contrast to CP level, the AUTOSAR VFB view must not contain internal functional behavior. This is not part of the Software Component Template. How to handle this within the SystemC simulation environment is discussed in Section 5.5.

5.3 Transformation after mapping onto ECUs

Mapping AUTOSAR software-components to an ECU architecture is part of the AUTOSAR workflow. Here, the Basic Software is connected via the Run Time Environment to the software-components.

This view is most similar to the SystemC PV level and still untimed. SystemC PV simulates behavior, which is running at one processing element, not in parallel but in a sequenced order by providing scheduling mechanisms. The communicating processes are partitioned to a specified processing element, as within AUTOSAR.

Therefore, an AUTOSAR design can be transformed to SystemC PV. Furthermore, the Basic Software (e.g. RTOS, device drivers, ...) can be implemented in SystemC to provide the same services in SystemC as in AUTOSAR.

This transformation step is particularly recommended, if the implementation of application software already exists. Then, the cycle accurate behavior of the software can be evaluated by refinement onto different virtual prototypes of target architectures, without having built up a real prototyping platform.

5.4 Integrating timing behavior

The integration of timing behavior in SystemC is done by refining the communication architecture as well as the computation as already shown in Section 3.2.

From CP as well as from PV, communication can be refined to PVT or CC respectively, both providing an approximated or accurate timing behavior of a dedicated communication architecture. SystemC encapsulates communication into channels and separates it from the computation. Thus, channels can easily be swapped, e.g. the channels implementing the point-to-point communication (that derives from the VFB) by channels implementing a specific behavior of a communication protocol including its timing behavior. The application can be left untouched if the same interface is used as before.

From CP as well as from PV, computation can be refined to PVT or CC, on condition that functional behavior of the application already exists. From CP, the design firstly has to be mapped to an RTOS model as detailed described in [11]. Then, or by starting at PV, SystemC models of the hardware architecture can be connected to the application. These models interpret the timing behavior of the application with respect to the underlying hardware architecture. Unfortunately, this refinement step is not automated yet.

The required models are already often provided by industry, for example models of microprocessors (e.g. IBM PowerPC [3]) or communication controllers (e.g. Freescale FlexRay Executable Reference Model [2]). This heavily eases the development of executable simulation models if using the provided models.

5.5 Integrating functional behavior

In automotive industry, the functional behavior is in generally developed by the supplier. Functional behavior is often developed with behavior modeling tools and then transformed into source code in a programming language, usually C. AUTOSAR provides infrastructure template for such functional behavior and specifies integration into the infrastructure. Of course it is not possible to consider functional behavior in SystemC, until it is implemented and delivered by the supplier.

If no functional behavior of the runnables exists, communication traffic can be generated by cyclic or random transmitting data of the corresponding data type. Cycle periods must be specified by the designer. This is possible because at VFB level the communication relationships and data types as well as RunnableEntities are already specified. In this case, only refinement of communication is practical, and only timing behavior of the communication architecture is considered.

Otherwise, if internal behavior of the runnables exists, integration into SystemC design is possible. This is supported by C-code generation. The generated C-function can directly be added to the SystemC design because the thread that contains the corresponding function has been generated from the RunnableEntity. In this case, also computation can be refined to a timed model by interfacing RTOS models and processor models.

5.6 Formal transformation rules

The affinities shown in Section 4 are the basis of the transformation from AUTOSAR to SystemC. Basically, the complete semantic of the Software Component Template has to be mapped to SystemC semantic.

This section presents the formal transformation rules for the mapping from AUTOSAR VFB to SystemC CP. Table 1 presents some required (but not all) transformation rules.

By using the UML 2 profile for AUTOSAR [22], the transformation rules are implemented by tool-based mapping of XMI onto SystemC. How to map from UML to SystemC can be checked e.g. in [23].

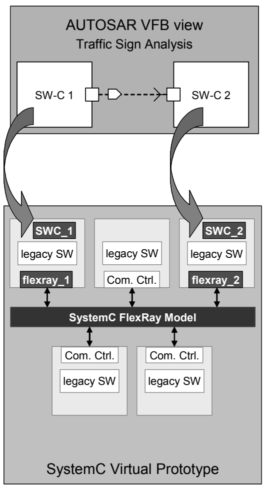
Table 1. Formal transformation rules

AUTOSAR	SystemC
Virtual Functional Bus	
ComponentType	abstract class
AtomicSoftwareComponentType	SC_MODULE
SensorActuatorSoftwareComponent	SC_MODULE
CompositionType	(hierarchical) SC_MODULE
ComponentPrototype	specific instance of an SC_MODULE
PortPrototype	abstract class
PPortPrototype	sc_export if client-server-communication
	sc_port if sender-receiver-communication
RPortPrototype	sc_port
PortInterface	abstract class
SenderReceiverInterface	sc_interface impl. sc_channel
ClientServerInterface	sc_interface impl. sc_channel
Datatype	abstract class
PrimitiveType	abstract class
Range	abstract class
IntegerType / e.g. uint8	sc_int, sc_uint, int, ... / e.g. uint<8>
RealType	float, double
BoolType	bool, sc_bit
OpaqueType	sc_bv
CharType	char
StringType	sc_string
ConnectorPrototype	abstract class
AssemblyConnectorPrototype	sc_export-sc_port binding (client-server)
	sc_port-channel binding(sender-receiver)
DelegationConnectorPrototype	sc_export-sc_export binding
	sc_port-sc_port binding
Run Time Environment Level	
Internal Behavior	In SystemC an AtomicSoftwareComponent must only have one internal behavior.
Runnable Entity - category 1	SC_METHOD
Runnable Entity - category 2	SC_THREAD
RTEEvent	abstract class
TimingEvent	specific instance of sc_event
DataReceivedEvent	specific instance of sc_event
DataReceivedErrorEvent	specific instance of sc_event
DataSendCompletedEvent	specific instance of sc_event
OperationInvokedEvent	specific instance of sc_event
AsynchronousServerCallReturnsEvent	specific instance of sc_event
ModeSwitchEvent	specific instance of sc_event
WaitPoint	wait(specific _sc_event)
DataReadAccess	implicit data access - specific implementation of semantic in SystemC
DataWriteAccess	
DataReceivePoint	explicit data access - via sc_export respectively sc_port
DataSendPoint	

6 Use Case

We used the aforementioned methodology to explore potentials of a given system. The system configuration consist of five ECUs connected via FlexRay [6]. All five ECUs run legacy software. The behavior of the FlexRay bus (static as well as dynamic segment) is already known.

In future, additional functionality, namely a traffic sign analysis, shall be mapped onto the existing ECUs. The application has been modeled using the AUTOSAR methodology and consist of several atomic software-components. It is assumed that communication between the software-components can not be applied to the static segment of the FlexRay cluster, because all static slots are already configured or reserved. However, since the traffic sign analysis application is not safety relevant, it is possible to use the dynamic segment for potential communication. The load of the dynamic segment is already known. Thus, we transferred the AUTOSAR software-components into SystemC. Two communicating software-components are mapped onto different ECUs and their signals are exchanged via a SystemC model of FlexRay. The application itself is well understood in the sense that the provided and required data rates are given. Figure 6 shows the mapping of the software-components onto different ECUs and an excerpt of the corresponding SystemC code of the software-components and instances.



```
SC_MODULE (SWC_1) {
  sc_port<sc_flexray_if> s_port; ....
  SC_CTOR(SWC_1) {
    SC_THREAD(send_data); ....}
  void send_data() { ....
    s_port.write(sdata); ....}
};

SC_MODULE (SWC_2) {
  sc_port<sc_flexray_if> r_port; ....
  SC_CTOR(SWC_2) {
    SC_THREAD(receive_data); ....}
  void receive_data() { ....
    rdata = r_port->read(); ....}
};

int sc_main(int arc, char *argv[]) {
  sc_flexray_bus_model flexray_bus;
  sc_flexray_controller flexray_1;
  flexray_1.port_to_bus(flexray_bus);
  sc_flexray_controller flexray_2;
  flexray_2.port_to_bus(flexray_bus);
  SWC_1 SWC_1_instance;
  SWC_1_instance.s_port(flexray_1);
  SWC_2 SWC_2_instance;
  SWC_2_instance.r_port(flexray_2);
  .....
}
```

Figure 6. Mapping of two SW-Cs to the SystemC virtual prototype & excerpt of corresponding SystemC code

Our simulation environment allows for calibrating the application parameters; data rates as well as bus load on the dynamic segment can be changed. The aim of the simulation is to determine whether some timing requirements for a special component, e.g. latency, will be abide or not.

Figure 7 shows the simulation results. The x-axis represents the sending date [in cycle] of a FlexRay frame. The y-axis represents the delay [in cycles] since the host wanted to send the frame. This figure depicts the results of an ECU, which has been assigned a minislot with low priority. It is possible to analyze the message send delay of the FlexRay frame within the dynamic segment. In this figure, the worst case message send delay is 44 cycles. In summary, the designer can validate the timing requirements, which is not possible in the AUTOSAR environment without SystemC.

Additionally to the timing behavior of the communication, also the timing behavior of the underlying ECU could be taken into consideration. By executing the application on a SystemC model of a possible ECU architecture (e.g. microcontroller), it could be evaluated how fast the application can be executed at the chosen ECU, for example. Of course this could also influence the communication timing behavior.

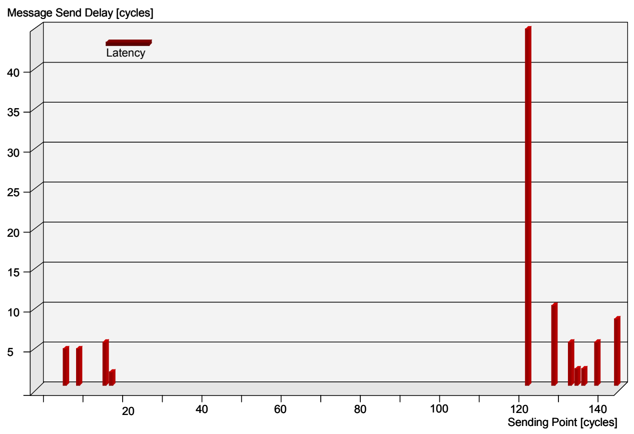


Figure 7. Simulation results: send latency of the message

7 Conclusion

This paper presents an AUTOSAR conform methodology for timing simulation of interconnected AUTOSAR software-components. The use case shows that simulation enables the testing and evaluation of a distributed interconnected system in an early design phase. SystemC offers a comprehensive methodology for the following reasons: On the one hand, SystemC enables to completely describe a system including its application, underlying hardware architecture, and infrastructure. On the other hand, SystemC provides a timing concept as well as a simulation kernel.

This paper points out analogies between SystemC and AUTOSAR but also the fundamental differences. As a consequence, SystemC can be used during the design process of AUTOSAR-conform systems, resulting in a strong increase in value by simulating, analyzing, and evaluating the entire interconnected distributed system at early design time.

Future work will consider the automation of mapping AUTOSAR software-component descriptions onto SystemC. The mapping of functional behavior, e.g. from MATLAB/Simulink, into the SystemC simulation environment is an additional important topic of further activities. Finally, the back-annotation of simulation results and analysis respectively the results of the validation can affect the configuration file. This is also the focus of future work.

8 Related work

This paper takes it for granted that the reader is well acquainted with SystemC and AUTOSAR fundamentals. For detailed information we recommend further readings. For detailed information about the AUTOSAR concept please refer to [1], [8], [9], [14], and [17]. A more detailed examination of the SystemC language and design methodology can be found in [4], [5], [7], [16], and [21]. An extended SystemC based design approach which is briefly introduced in Section 3 is discussed in [11] and [12]. Generating SystemC from UML has been introduced in [23]. The benefit of the methodology that is discussed in this paper is the capability of evaluating interconnected AUTOSAR software-components by using SystemC. The evaluation methods base on SystemC and are not the topic of this paper. In the last years, a lot of work has been spent to these topics of simulating [10] and verifying SystemC based designs e.g. by performance and communication analysis methods [18], [19]. Please refer to these papers for further information.

9 References

[1] AUTOSAR: http://www.autosar.org

[2] M. Baumeister, P. Fuhrmann, F. Armbruster: *Taking Concept Models from Standardization to Silicon;* In FlexRay Special Edition, Hanser Automotive Electronics + Systems, 2005.

[3] R. Bergamaschi: *Transaction-Level Models for PowerPC and CoreConnect;* 11th European SystemC Users Group Meeting, Munich, 2005.

[4] A. Donlin: Transaction Level Modeling: *Flows and Use Models*; Proceedings of the 2nd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis, Stockholm, 2004.

[5] A. Donlin, A. Braun, A. Rose: *SystemC for Design and Modeling of Programmable Systems*; In Proc. of International Forum on Specification and Design Languages, Lille, 2004.

[6] FlexRay: http://www.flexray.com/

[7] T. Grötker, S. Liao, G. Martin, S. Swan: *System Design with SystemC*; Kluwer, 2002.

[8] H. Heinecke, et al.: *AUTOSAR - Current results and preparations for exploitation*; 7th EUROFORUM conference „Software in the vehicle“ Stuttgart, 2006.

[9] H. Heinecke, et al.: *AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures*; Convergence 2004, Detroit, 2004.

[10] J. N. Ip, S. Swan: *A Tutorial Introduction on the New SystemC Verification Standard*; OSCI SCV Working Group, 2003.

[11] M. Krause, O. Bringmann, W. Rosenstiel: *A SystemC-based Software and Communication Refinement Framework for Distributed Embedded Systems*; Proceedings of the 13th Workshop on Synthesis And System Integration of Mixed Information Technologies, Nagoya, 2006.

[12] M. Krause, O. Bringmann, W. Rosenstiel: *Target Software Generation: An Approach for Automatic Mapping of SystemC Specifications onto Real-Time Operating Systems*; Design Automation for Embedded Systems, Volume 10, Issue 4, Springer, 2007.

[13] MATLAB/Simulink: http://www.mathworks.com/

[14] M. Pagel, M. Brörkens: *Definition and Generation of Data Exchange Formats in AUTOSAR;* Proceedings of the Second European Conference ECMDA-FA 2006.

[15] P. Pop, P. Eles, Z. Peng: *Analysis and Synthesis of Distributed Real-Time Embedded Systems*; Kluwer Academic Publishers, Boston, 2004.

[16] A. Rose, S. Swan, J. Pierce, J.-M. Fernandez: *Transaction Level Modeling in SystemC;* OSCI TLM Working Group, 2005.

[17] T. Scharnhorst, et al.: *AUTOSAR - Challenges and Achievements 2005*; Electronic Systems for Vehicles 2005, VDI Congress, Baden-Baden, 2005.

[18] A. Siebenborn, O. Bringmann, W. Rosenstiel: *Communication Analysis for System on Chip Design;* Proc. of the Design Automation and Test in Europe Conference 2004, Paris, 2004.

[19] A. Siebenborn, O. Bringmann, W. Rosenstiel: *Worst-Case Performance Analysis of Parallel, Communicating Software Processes;* Proceedings of the Tenth International Symposium on Hardware/Software Codesign 2002, USA, 2002.

[20] SPIRIT Schema Working Group Membership, IP-XACT Users Guide Version 1.4, 2005.

[21] SystemC: http://www.systemc.org

[22] UML Profile for AUTOSAR Version 1.0.1; AUTOSAR, 2006.

[23] A. Viehl, O. Bringmann, W. Rosenstiel, T. Schönwald: *Formal Performance Analysis and Simulation of UML/SysML Models for ESL Design*; Proceedings of the Design, Automation and Test in Europe Conference, Munich, 2006.