In *Proceedings of the Embedded World Conference*, Nuremberg, Germany, March 03-05, 2009

From Model-based Design to Virtual Prototypes for Automotive Applications

Martin Streubühr, Michael Jäntsch, Christian Haubelt, and Jürgen Teich
Hardware/Software Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
{streubuehr, haubelt, teich}@cs.fau.de, michael.jaentsch@in.tum.de

Abstract

In this paper, we present a new design methodology for automotive applications, combining the strength of model-based design using MATLAB/Simulink and virtual prototyping using SystemC. The design flow starts from an automotive application modeled in MATLAB/Simulink. By including a vehicle model, testing and debugging of the desired application is possible. In a first step, the application model is automatically transformed into SystemC code. In a second step, the ECU (Electronic Control Unit) architecture including control units and communication buses of the vehicle is also modeled in SystemC. While the application model defines the functional property of the envisioned algorithm, the ECU architecture model is responsible for modeling non-functional properties, e.g., time, area and power consumption. Finally, the SystemC application model is related to the ECU architecture model by mapping activations of the application to the modeled control units and buses. In that way, non-functional properties can be evaluated by the help of the SystemC simulation kernel.

Furthermore, the flexibility of the proposed approach allows for assessing the effect of different design decisions in early design phases, as the entire ECU architecture is modeled in SystemC. Hence, partitioning the application model onto many ECUs or multiple processors inside ECUs can be done easily by only changing a single configuration file. We will demonstrate the benefits of the proposed approach using a brake-by-wire application mapped onto an ECU architecture based on a FlexRay bus system.

1 Introduction and Motivation

Model-based Design using MATLAB/Simulink [1] is a de-facto standard for system modeling in the automotive domain. These models serve as high-level reference, executable specification, and allow for functional simulation and even automatic software code generation. However, as the focus is set on functional modeling, important non-functional properties, as timing information, area and power consumption, etc. are hard to evaluate in early design phases.

On the other hand, the SystemC language [2] is gaining a great acceptance in other application domains like, e.g., multi media, communications, and digital signal processing. SystemC is a C++ class library and has recently become an IEEE standard. Its major application domain is at the electronic system level, and it is often used for virtual prototyping. Virtual Prototypes written in SystemC permit modeling and simulating of entire hardware/software systems, the development of firmware before a chip prototype is available, and the early evaluation of non-functional properties.

Figure 1 shows the differences between a functional model in Simulink and different hardware/software implementations where non-functional properties affect the overall execution latency. In Figure 1(a) an exemplary Simulink model is given. Two sensors gain measurements

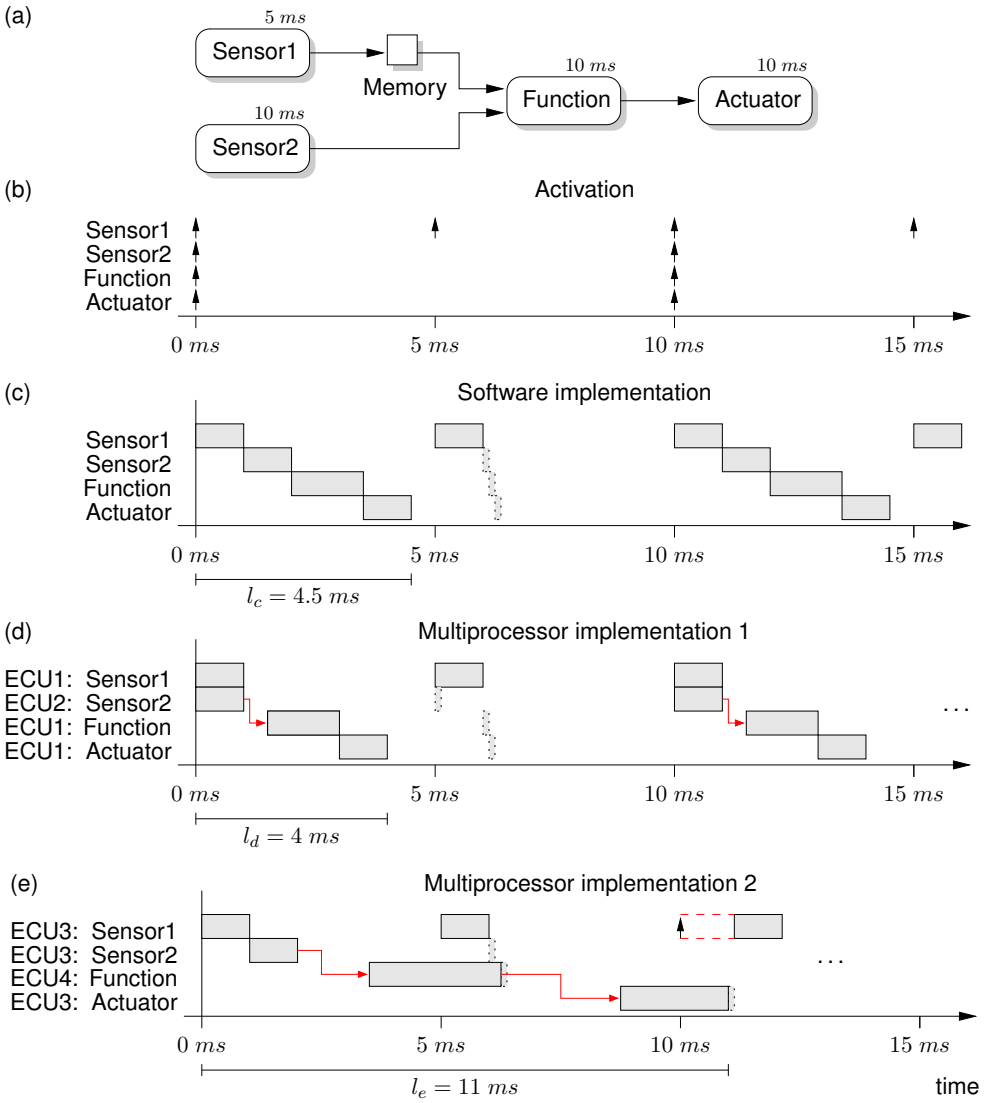


Figure 1: (a) A Simulink model is composed by blocks. (b) Each block has an associated activation period. (c) Software implementation causes execution times for each block. (d) A multiprocessor implementation potentially allows parallel execution of blocks, but introduces communication latencies. (e) Execution conflicts may occur if execution and communication latencies of an implementation became too large.

from the environment and a block (Function) takes the output values from the sensors as input values. A memory block is used as rate transition between different sample times. The output computed by the Function block is used by the Actuator in order to perform some kind of reaction. E.g. the model could represent a simple force-feedback mechanism of an electrical brake pedal in a car: Pedal pressure caused by a driver and brake force measured at the wheels is used to compute feedback forces applied by an electromechanical brake pedal. All blocks in such a model are executed at a certain sample time. Here, block Sensor1 is activated every $5\ ms$, while the remaining blocks are activated at a period of $10\ ms$. The ideal execution of such a reactive Simulink model is depicted in Figure 1(b). Activation of blocks is depicted by events. Execution of blocks is assumed to be infinite fast. Still, the partial ordering of the blocks in the model given by its communication dependencies has to be preserved. This results in some kind of micro-step execution scheme similar to delta-cycles.

Real Time Workshop, a toolbox for Simulink, allows for generation of embedded C code running on embedded processors. For code generation, the partial ordering of a Simulink

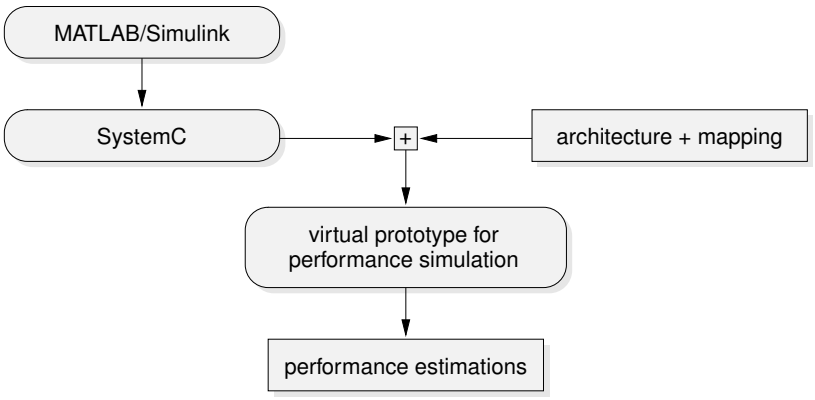


Figure 2: Simulink models are transformed to SystemC models. Architecture and mapping details are added to the SystemC model via configuration files in order to form a virtual prototype. Simulation allows evaluating the non-functional performance parameters.

model is turned into a total execution order. Code for the blocks is generated in individual functions or is in-lined if possible. In a main, function the individual function calls for the blocks are placed according to a total ordering that preserves the semantics of the Simulink model. Also cyclic dependencies that include registers are supported, if the registers allow breaking up the cyclic dependencies. An execution trace for such a generated software code is depicted exemplary in Figure 1(c). There, the execution of blocks running at different sample times can be seen. The entire software program runs at a low sample time. This sample time has to be a common divisor to all sample times occurring in the model. The execution of functions is guarded via modulo counters. In the example in Figure 1(c), the counter for block Sensor2 is tested and incremented at $6\ ms$. As result from this test, the code is not executed, but checking the counter requires some execution time.

Figure 1(d) shows a potential benefit of saving execution latency by a multiprocessor implementation of the Simulink model. While preserving the partial order, the code for blocks Sensor1 and Sensor2 is executed in parallel on different microprocessors ECU1 and ECU2. Also shown is the additional communication latency caused when transmitting the data from block Sensor2 to the Function block. Not in all cases, a multiprocessor implementation guarantees shorter execution latency. E.g. in Figure 1(e) it can be seen that communication latencies and different execution times on different types of microprocessor may cause a higher execution delay. From this example, it can be seen that a good knowledge on possible implementation variants of a model is necessary to draw successful design decisions. Building hardware prototypes is costly and time consuming and thus not appropriate at an early stage of the design. Virtual prototypes can be used to assess the non-functional parameters of implementations of a system with fewer efforts. Still much effort has to be spent for building a virtual prototype from a Simulink model.

In this paper an approach is presented that helps to close the gap. The design flow is given in Figure 2. A modified C code generator transforms a Simulink model into a SystemC model. An architecture model and mapping information is added to this functional SystemC model using XML-formatted configuration files. Performance simulation allows to assess execution latencies and to determine if execution latencies will obtain real-time performance. Different variants in mapping and architecture can be easily evaluated by simply modifying the configuration files. The approach has many similarities to the Y-chart approach [3], as application model and architecture model are separated initially. Mapping both models together forms a virtual prototype at an abstract level and allows for performance simulation.

The rest of the paper is structured as follows. In Section 2, approaches related to our proposed methodology are discussed. Section 3 gives a brief overview on Simulink. Transformation of Simulink models to SystemC models is presented in Section 4. A case study using

an automotive application of a brake-by-wire system is given in Section 5. Finally, Section 6 discusses the presented approach.

2 Related Work

Huang et al. [4] presented a case study of a Simulink-based MPSoC design flow. An initial functional Simulink model is composed to a *combined algorithm and architecture model* (CAAM), which is derived by hand using hierarchical grouping of functional blocks. Here, the hierarchical structure represents the partitioning into CPU subsystems and thread subsystems as well as inter- and intra subsystem communication blocks. Afterwards, the Simulink CAAM model may be implemented at different abstraction levels (Virtual Architecture, Transaction-accurate Model and Virtual Prototype) using a *Multithreaded Code generator*. A very similar approach is reported by Atat and Zergainoh [5]. The functional Simulink model is refined to a Simulink transactional model. Using tool support, the transactional model is transformed to the more detailed *Macroarchitecture Model* or the even more detailed *Microarchitecture Model*. Simulation allows for verification at each abstraction level. In contrast to our work, partitioning into mixed hardware/software systems is done within the Simulink model, by grouping functional blocks and inserting special interface blocks.

Caspi et al. [6] proposed a layered approach quite similar to our own, while translating Simulink models to *SCADE/Lustre* [7], a synchronous language, and afterwards implementing it on a time triggered architecture. Their focus lies on designing safety critical software for the automotive and avionic industry. This is achieved by using *SCADE/Lustre*, which features a level-A certified automatic software code generator.

An approach to the transformation of Simulink models to the *System Property Intervals* language is reported by Jersak et al. [8]. This approach transforms the time-driven model of computation into a data-driven model, by introducing virtual FIFO-queues for synchronization between different sample times.

Stefanov et al. [9] presented a design flow for implementing MATLAB code on a target platform utilizing a microprocessor and an FPGA. A nested loop program specified in MATLAB code is transformed to a *Kahn Process Network*, which can be mapped to the target platform.

3 Simulink

Simulink is a toolbox for MATLAB, developed by The MathWorks [1], which can be used to graphically model and simulate hierarchical systems. Another toolbox for MATLAB is the Real-Time Workshop, which offers a code generator, to generate highly optimized C code from a Simulink model. Simulink provides an interactive, block-diagram based, graphical environment. Libraries offer a broad variety of predefined blocks that can be used along with user-defined functions. These can either be described by embedded MATLAB code or as so called S-Functions [1]. Data-flow between the blocks is realized using directed connections represented by arrows. Blocks are evaluated at a certain time, the sample time, which can be set globally or for each block individually.

Figure 1(a) presents an example of a Simulink model consisting of five blocks. Each block has a certain sample time in this example. Due to different sample times between blocks Sensor1 and Function, a memory (Rate Transition) is used for down sampling the input sensor data. Discrete simulation of a Simulink model runs at steps of a minimum period that is a common divisor to all sample times used in a model. A block is only evaluated at a given time step, if this time stamp is an integer multiple of the blocks sample time. Thereby the partial ordering given by data dependencies is preserved. As can be seen in Figure 1(b), the Simulink simulation would compute the output value of the Sensor1 block in parallel to the Sensor2 block

at time $0\ ms$. Then, output data from block Sensor1 is stored in the Memory block. An old value potentially stored in the Memory is discarded. Afterwards, the Function block and afterwards the Actuator block are simulated. At time stamp $5\ ms$ only the block Sensor1 is executed and its output values is written to the Memory block. The other blocks are not activated at this time stamp. At time stamp $10\ ms$ the execution of the model starts again as in time stamp $0\ ms$.

Simulink is often used to simulate and verify algorithms in early stages of a design, as it allows for rapid design, simulation, and generation of real-time C code. The option of hierarchical model development simplifies the design and reuse of systems.

4 SystemC representation of Simulink

In this section, we give a brief overview on SystemC and present the transformation step from Simulink models to SystemC. Furthermore we clarify how the SystemC model can be configured with architecture and mapping information in order to form a virtual prototype.

4.1 SystemC

The IEEE standard SystemC [2] is a C++-based design language, which is well suited for designing hardware/software systems. SystemC permits modeling at a wide range of abstraction levels. While modeling at register transfer level is possible, SystemC is typically used at higher levels of abstraction like the behavioral level or even the electronic system level.

A SystemC model is composed of modules. Modules may be composed by other modules allowing for hierarchical modeling. SystemC supports different kinds of concurrent processes, namely threads and methods. A module may contain one or more processes, while each process is related to only one module. Communication between modules is restricted to dedicated channels only. The SystemC reference implementation comes with a simulation kernel allowing for discrete event simulation of SystemC models. Execution time delay occurring during execution of processes can be modeled with the SystemC function call `wait`. Passing a delay value to this function (e.g. `wait(20, SC_NS);`) causes the simulation kernel to stop the execution of the calling process until the given time has passed by. Sophisticated tracing mechanisms in SystemC allow for monitoring variable and channel values during simulation.

4.2 Code generation

The straight-forward approach of mapping Simulink to SystemC is as follows: Each Simulink block is transformed into a SystemC module consisting of a thread process. Signals in Simulink are mapped to channels in SystemC. The concurrency in a Simulink model, as well as the execution semantics has to be preserved during the transformation of a Simulink model. To be more precise, we do not transform the Simulink model to SystemC directly, but to a custom modeling library for actor-oriented design [10] that is based on SystemC. By mapping the Simulink model to a synchronous reactive model-of-computation, we preserve the concurrency and semantics of the original Simulink model. A detailed description of this transformation steps can be found in [11].

The proposed method to generate SystemC code from Simulink models is based on the Real-Time Workshop (RTW) [12] toolbox. RTW is able to generate highly optimized C code from Simulink models, while code generation can be customized by the Target Language Compiler (TLC) [12]. By mainly using Target Language Compiler directives, we can use most of the inherent flexibility, e.g., the automatic generation of code for user-specific Simulink blocks.

The SystemC modules created by our customized Target Language Compiler have a static template which is then augmented with functional C code, generated by RTW. Each module declares input and output ports and uses constructor code for initialization according to the given

block's functionality. For discrete Simulink models, blocks may have internal discrete states (DSTATES), e.g., a memory block is able to store the last input. Blocks with discrete states may be used to break combinatorial feedback loops that otherwise could not be supported for C code generation. In this case, two functions `update()` and `output()` are created by the original RTW. Feedback-loops are broken by calling first the `output()` function writing the output, and later, when sufficient data is available, the `update()` function. Obviously, this behavior represents a Moore finite state machine, where outputs only depends on the actual state and may be produced independently from inputs. We cope with discrete state blocks by using a module with two processes. The two processes represent the `output()` and `update()` functions used for writing outputs depending on the internal state and updating the internal state depending on the inputs, respectively. As a result, input and output signals are decoupled and feedback loops are executed in the same manner as in the original Simulink model.

Currently, we only support *discrete* Simulink models. Therefore, only the fixed step, discrete state solver is supported, while Simulink offers a couple of different types of solvers for simulation. In particular, we do not support any *continuous blocks*. Using continuous solvers would make the model-of-computation ambiguous and therefore not appropriate for designing mixed hardware/software systems. Moreover, we do not support multidimensional or complex signals.

All blocks that are stateless, e.g. mathematical operations or any type of source, as well as all blocks from the discrete library, e.g. a *Discrete Transfer Fcn*, are supported for translation to SystemC. We support user-defined functions representing analytical functions by means of the simple *Fcn* block (e.g. $y = x^2 + sin(x)$). Other kinds of user-defined functions are not implemented yet.

In Simulink, it is possible to have blocks running at different sample times in the same model. For code generation using RTW, they must be separated by so called *Rate Transitions*. A *Rate Transition* in Simulink holds the signal from the predecessor block until it changes, similar to a register. So, at whatever sample time the successor is running, it always gets a valid input signal. In order to support different sample times, we assume the entire SystemC model is executed at a minimum period given by the greatest common divisor of all sample times. To achieve the individual sample times of blocks, we use counters to guard the execution of the functionality inside modules.

4.3 Virtual Prototypes

In order to generate a virtual prototype, we need to extend the SystemC model with an appropriate architecture mapping. This step is supported by a custom library for modeling the architecture by the terms of *Virtual Processing Components* (VPC) [13]. The library is implemented in SystemC and allows for performance simulation of SystemC models. It works by creating SystemC modules representing components in the architecture like microprocessors, hardware module, buses, memories, etc. E.g. in Figure 3(a) the three components ECU1, ECU2, and Bus are modeled in the VPC library. The components in the architecture model are configured in the VPC library by an XML formatted configuration file as given in Figure 3(a). Mapping of SystemC modules representing Simulink blocks is twofold. On the one hand the C code in each SystemC module in the application model has to be extended by a function call using the `compute` function from VPC with the module name as parameter. This function call is automatically added to each SystemC module during the transformation from Simulink to SystemC. On the other hand the mapping for each application module to a component has to be modeled in the VPC using the configuration file. The mapping and its XML code for the module Sensor1 to the ECU1 component is given in Figure 3(a). With each mapping a core execution time is associated. In the example Sensor1 has an execution delay of $10\ us$ when mapped to ECU1. In a similar manner, the communication channels are mapped to the architecture. In case of the FlexRay bus, additional parameters are required for the FlexRay schedule and the

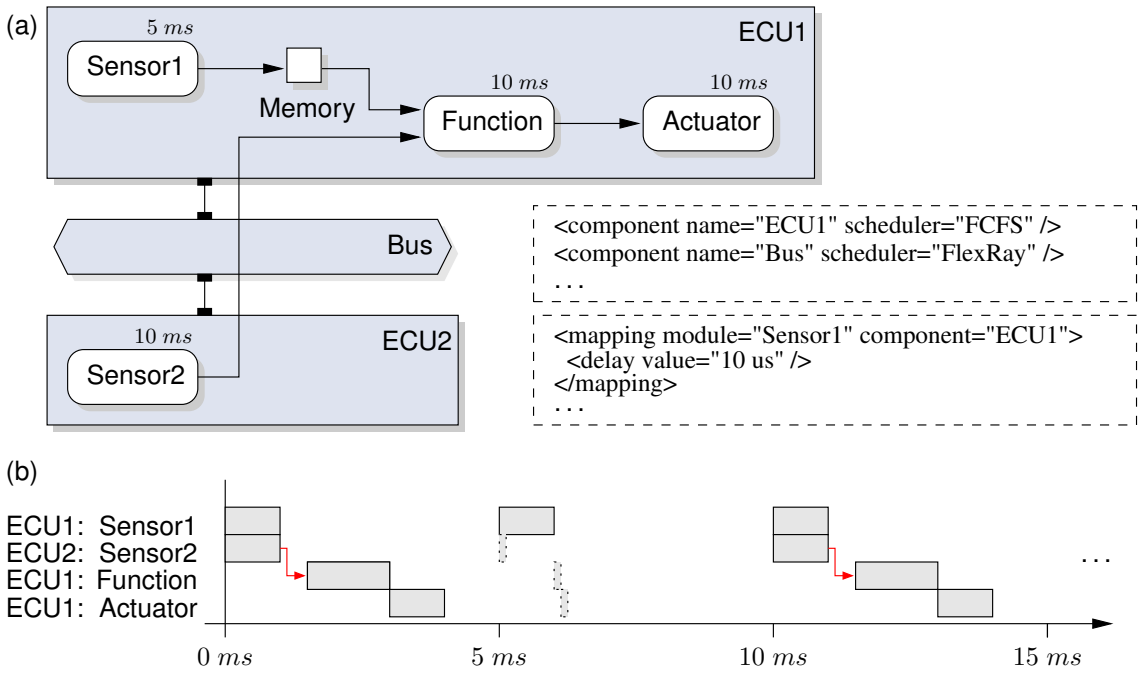


Figure 3: (a) The architecture model and the mapping of SystemC modules to architecture components are configured by an XML file. (b) Performance simulation models delays for execution of modules and communication of data.

mapping of communication messages to FlexRay slots.

If several modules are mapped to the same component, the potentially occurring resource contention is solved by the VPC using sophisticated scheduling policies assigned to each component. Those policies allow modeling the operation system scheduling occurring on microprocessor, as well as arbitration schemes used in shared communication media like buses.

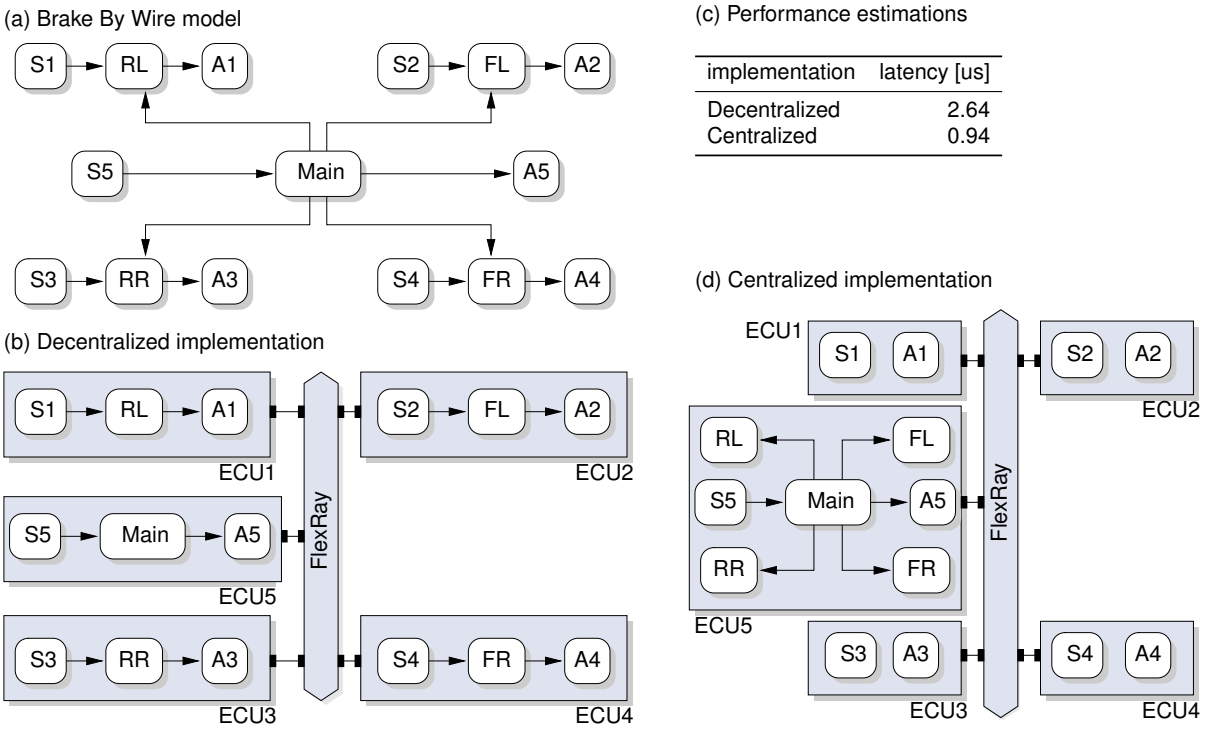
Synchronization between application model and architecture model is implemented in the function `compute`. An exemplary execution scheme for the virtual prototype in Figure 3(a) is given in Figure 3(b). At time $0\ ms$ the module Sensor1 and Sensor2 are activated in the application model as a new activation period is started. Both modules compute their functionality, e.g. reading sensor values from the environment model. Afterwards both modules call the `compute` function where the core execution time and the overhead possibly caused by preemption and scheduling is simulated.[1] The `compute` function returns the process control to the application module when the simulated execution time has passed. Afterwards, the output data is committed to the output channels. After communication delay which is modeled by a `compute` function also, the data is received by the Function module. Thus, the Function module is activated and again functionality and execution time is simulated. Those mechanisms are repeated for all modules and afterwards the very next activation period is started.

Module execution times and communication delays are traced by our VPC library. The trace is post processed after simulation in order to measure the latencies that occur in the activation periods. Also, the trace is analyzed if any execution conflicts between different activation periods occur due to execution delays.

5 Case Study: Brake by Wire

The proposed approach in this paper has been tested with an automotive brake-by-wire application. A brake-by-wire system and a vehicle model have been modeled using Simulink. The

[1] Execution delay on a component is modeled by the `wait` function in SystemC.



implementation	latency [us]
Decentralized	2.64
Centralized	0.94

Figure 4: (a) The brake-by-wire application is modeled in Simulink. After transformation to SystemC, the application is mapped to a decentralized implementation (b) and a centralized implementation (d), each using five ECUs and one bus. (c) Latency values for one execution are recorded during simulation of different architecture variants.

brake-by-wire model is depicted in Figure 4(a). The Main node in the model is responsible for computing break force values for the individual wheels and force feedback values for the break pedal. Each of the four wheel nodes (RL, RR, FL, FR) computes corrected brake forces in order to implement the ABS functionality. Furthermore, the model consists of several blocks responsible for monitoring wheel speeds and applied brake forces to each wheel (S1-S4) and for monitoring force applied to the break pedal and its position (S5). Five actuators apply the brake force to the individual wheels (A1-A4) and a feedback force to the break pedal (A5). All blocks run at the very same sample time of $10\ ms$. During simulation the input values to sensors and output values of actuators are controlled by the vehicle model. The vehicle model is modeled in Simulink and is used to verify the functional model and is used as test-bench for the performance simulation.

We transformed both Simulink models, brake-by-wire and vehicle model into SystemC models using our modified code generator. The SystemC model is linked against the VPC library and configured with architecture and mapping information. Two different mapping variants to an architecture consisting of five ECUs and one FlexRay bus are depicted in Figures 4(b) and 4(d). The centralized mapping in Figure 4(d) performs main node and wheel node computations on ECU5 and sensor and actuator functionality is performed on ECUs 1-4. Another mapping decentralizes the computation as depicted in Figure 4(b). Here, each of the four wheel nodes and the main node are mapped to different ECUs. In both mappings, communication is assumed to take place on internal ECU buffers if possible or on the FlexRay bus for communication between different ECUs.

We assumed artificial execution delays for each block in the brake-by-wire model and no execution delays in the vehicle model. The ECUs are assumed to be identical and thus the execution times are identical for different mappings of individual blocks. Note that, in order to

model heterogeneous ECUs, individual delays need to be modeled for each mapping. During performance simulation the latency for each activation period is measured. The typical latency values observed for both mapping variants are given in the table in Figure 4(c). Sample times of blocks is modeled with $10\ ms$ and thus, the latency needs to be less than $10\ ms$ to guarantee correct functionality of the brake-by-wire system. The centralized and decentralized mappings achieve typical latencies of $2.64\ us$ and $0.94\ us$. Here, both mapping variants are compatible to the sample time of $10\ ms$.

6 Discussion

We have presented an approach for transforming Simulink models to SystemC models. Furthermore, the SystemC models are enriched by architecture models and appropriate application to architecture mappings forming a virtual prototype at a high level of abstraction. The virtual prototype allows to asses the non-functional performance parameters of a candidate architecture variant. By varying the configuration, the design space spanned by the mappings, architecture, ore even scheduling and arbitration parameters can be explored for suitable implementation candidates.

A drawback of our approach lies in the used code generation technique based on the Real-Time Workshop. Optimization steps occurring in the Real-Time Workshop cause in-lining of some Simulink blocks during code generation. Such optimizations may be used to speedup execution on a single processor. Nevertheless those optimizations hide parallelism in the SystemC model that potentially could be used to reduce execution latency in multiprocessor systems. We could overcome the limitation by changing the code generation technique to a separate transformation of each single Simulink block in a SystemC model. A second transformation step is needed to generate the communication dependencies and the instantiation of SystemC modules. In this case, the Real-Time Workshop is bypassed by working directly on the original Simulink model.

References

[1] The MathWorks, Inc.: Simulink - Using Simulink www.mathworks.com., 2007

[2] IEEE: IEEE Standard SystemC Language Reference Manual (IEEE Std 1666-2005), 2006

[3] Kienhuis, B., Deprettere, E., Vissers, K., van der Wolf, P.: An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, Zurich, Switzerland, p. 338–349, July 1997

[4] Huang, K., Han, S.I., Popovici, K., Brisolara, L., Guerin, X., Li, L., Yan, X., Chae, S.I., Carro, L., Jerraya, A.A.: Simulink-based MPSoC design flow: case study of Motion-JPEG and H.264. In: Proceedings of the 44th Design Automation Conference, San Diego, California, p. 39–42 ACM, 2007

[5] Atat, Y., Zergainoh, N.E.: Simulink-based MPSoC Design: New Approach to Bridge the Gap between Algorithm and Architecture Design. In: Proceedings of the IEEE Computer Society Annual Symposium on VLSI, Porto Alegre, Brazil, p. 9–14 IEEE Computer Society, 2007

[6] Caspi, P., Curic, A., Maignan, A., Sofronis, C., Tripakis, S., Niebert, P.: From Simulink to SCADE/Lustre to TTA: a Layered Approach for Distributed Embedded Applications.

Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems, p. 153–162, 2003

[7] Esterel Technologies: SCADE Suite^{TM} www.esterel-technologies.com., 2007

[8] Jersak, M., Cai, Y., Ziegenbein, D., Ernst, R.: A Transformational Approach to Constraint Relaxation of a Time-driven Simulation Model. In: Proceedings of the 13th international symposium on System synthesis, Madrid, Spain, p. 137–142 IEEE Computer Society, 2000

[9] Stefanov, T., Zissulescu, C., Turjan, A., Kienhuis, B., Deprettere, E.F.: System Design Using Kahn Process Networks: The Compaan/Laura Approach. In: Proceedings of Design Automation and Test in Europe, p. 340–345, 2004

[10] Falk, J., Haubelt, C., Teich, J.: Efficient Representation and Simulation of Model-Based Designs in SystemC. In: Proc. FDL'06, Forum on Design Languages 2006, Darmstadt, Germany, p. 129–134, September 2006

[11] Streubühr, M., Jäntsch, M., Haubelt, C., Teich, J., Schneider, A.: Semi-Automatic Generation of mixed Hardware-Software Prototypes from Simulink Models. In: 11. Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Freiburg, Germany, p. 139–148, March 2008

[12] The MathWorks, Inc.: Real-Time Workshop 6: Target Language compiler www.mathworks.com., 2007

[13] Streubühr, M., Falk, J., Haubelt, C., Teich, J., Dorsch, R., Schlipf, T.: Task-Accurate Performance Modeling in SystemC for Real-Time Multi-Processor Architectures. In: Proceedings of Design, Automation and Test in Europe, Munich, Germany, p. 480–481, March 2006