Holistic Simulation of FlexRay Networks by Using Run-Time Model Switching

Michael Karner∗, Eric Armengaud†, Christian Steger∗ and Reinhold Weiss∗

∗Institute for Technical Informatics, Graz University of Technology, Austria
†Virtual Vehicle Competence Center, Austria

{michael.karner, steger, rweiss}@tugraz.at, eric.armengaud@v2c2.at

Abstract—Automotive network technologies such as FlexRay present a cost-optimized structure in order to tailor the system to the required functionalities and to the environment. The space exploration for optimization of single components (cable, transceiver, communication controller, middleware, application) as well as the integration of these components (e.g. selection of the topology) are complex activities that can be efficiently supported by means of simulation. The main challenge while simulating communication architectures is to efficiently integrate the heterogeneous models in order to obtain accurate results for a relevant operation time of the system. In this work, a run-time model switching method is introduced for the holistic simulation of FlexRay networks. Based on a complete modeling of the main network components, the simulation performance increase is analyzed and the new test and diagnosis possibilities resulting from this holistic approach are discussed.

I. INTRODUCTION

The FlexRay technology [1] is being introduced as a new wired network for automotive high-speed control applications. This communication protocol provides features like increased data rates (a factor of 10 faster than the CAN protocol) and the simultaneously support of time-triggered and event-triggered communication schemes for both deterministic network behavior and the efficient transmission of single events. This protocol, in comparison to predecessors (e.g. CAN), presents a very large flexibility that results in a huge number of implementation variants. Parameters such as topology (passive line, active star, hybrid topologies), bus schedule (e.g. slot / cycle length) and communication matrix (mapping between ECUs, slots and frames) are as much variables that can influence the quality of the communication.

One important aim of the TEODACS project is to understand the interactions between the layers building a dependable network and to evaluate the different effects influencing the communication (e.g. topology, EMC). The approach is based on the development of a heterogeneous co-simulation model of a FlexRay network tightly interfaced to a realistic FlexRay prototype. This combines the good observability and diagnosability provided by the simulation environment with the realistic system behavior provided by the hardware prototype.

The holistic simulation of a FlexRay network represents a challenge due to the heterogeneous nature of the system. Hence, a typical network consists of analog components (physical layer, transceivers), digital components (data link layer, communication controllers), basic software components (middleware, AUTOSAR) and software components (application). An important problem is to obtain a long simulated time (time interval under observation) for the analysis of the entire system, while at the same time keeping a reasonable simulation time (computation speed of the simulation) and a reasonable simulation accuracy for each component. This is especially difficult for lower layers that require more performance for the simulation of complex analog components.

Within this work, the run-time model switching method introduced in [2] is integrated within our FlexRay co-simulation environment. The main concept is to provide different simulation models of the same component (e.g. one for high speed, another one for high accuracy) that can be switched during the simulation run. This approach provides the test designer with a new degree of freedom for defining for each component the time intervals for which a computational expensive high-detail simulation model is required and when a simpler and faster model suffices. This method can be compared with an oscilloscope having the capability to zoom into significant parts of the simulation, while performing a fast (less accurate) run for the less interesting part of the simulation.

This document presents two main contributions. First, the performance increase resulting from the run-time model switching method is analyzed (both theoretically and experimentally). Second, the resulting heterogeneous co-simulation model of the FlexRay network is presented, and the resulting cross-layer analysis possibilities (e.g. analysis of the influence between the physical and data link layer) are discussed. The document is organized as follows: Section II provides an overview of the state of the art regarding simulation methods of automotive networks. In Section III, the concepts of the run-time model switching method are illustrated and the rationale for the performance analysis presented. The focus of Section IV is set to the experimental validation of our approach and finally Section V concludes this work.

II. SIMULATION OF FLEXRAY COMMUNICATION NETWORKS: STATE OF THE ART

For the simulation of different parts of a (FlexRay) communication network several solutions and simulation models have been developed. For example, in [3] a SystemC-based FlexRay communication controller is presented for the timing analysis of interconnected AUTOSAR components. Furthermore, a communication controller model developed using Verilog is proposed in [4] for the assessment of message missing failures. In [5], another implementation of a communication controller
using SystemC is described. Yet another simulation model of a FlexRay communication controller is demonstrated in [6]. Here, the controller is implemented using standard C language and simulated on a PC used to interact with real hardware components like engines.

Moreover, different FlexRay transceiver models are available: A generic model implemented using VHDL-AMS and including physical effects like thermal power is presented in [7], while the model presented in [8] is a specific behavioral Saber model of an NXP transceiver. The simulation setup described in [9] incorporates several simulation models for the FlexRay physical layer including cable, passive star and transceiver simulation models using Synopsis Saber. All of these approaches are focused on one single component or layer, thus the interactions with the other components and protocol layers involved are difficult to analyze (e.g. interaction between physical layer and data link layer).

In [10], the simulation of large heterogeneous systems including several ECUs communicating via a network is implemented by using transaction based modeling. The authors use a hardware abstraction layer for high level access to the hardware components and simulate at a very high abstraction layer, hence achieving a high simulation speed but losing many important details about the communication.

A further approach for analyzing dependable communication networks is residual bus simulation which enables the emulation of an entire sub-network. Industrial solutions from e.g. Elektrobit, dSPACE and Vector are available. All of these examples are a combination of hardware and software components: The software emulates the functionalities of the missing ECUs and the application messages are transmitted to a real network using dedicated hardware components. Hence, the advantages of simulation (observability, traceability, flexibility,...) are lost for many parts of the system.

It can be seen that there exist several solutions for the simulation of dependable communication networks, especially of FlexRay networks. However, these simulations usually cover only a specific part of the network and/or are working in a very abstract way. No holistic approach covering all parts of the network (from physical layer up to the application layer) with sufficient accuracy is available, making a comprehensive in-depth analysis nearly impossible to achieve by using existing setups.

III. RUN-TIME CO-SIMULATION MODEL SWITCHING

A. Run-time switching

Co-simulation is a well known approach for the integral simulation and analysis of different simulation models implemented at different abstraction levels and/or by using different hardware description languages (HDL). However, when combining highly heterogeneous simulation models, standard co-simulation presents limitations: the overall simulation performance depends on the slowest simulator involved, slowing down the other simulators to guarantee synchronization and data exchange. The drop in simulation performance is especially a problem when mixing accurate models (e.g. physical/analog level) with system level simulation in a common co-simulation. At system level, the focus is set to short simulation time (computation speed of the simulation) in order to obtain longer simulated time (time interval under observation). At physical level the simulation times are typically much higher because of the simulation of complex analog components. Hence, relatively short simulated times are explored here ($\mu s, ms$). The guidelines of the different abstraction level models are quite opposite: For system level models, the simulated time of several seconds can not be handled by the low level physical simulation model. This would lead to enormous simulation times for the low level model that may not be acceptable or even accomplishable within a reasonable amount of time. On the other hand, the very short simulated times for the low level physical simulation model are too short for the abstracted system level model. During this very short time interval, nothing of interest may happen at system level, thus rendering the system level simulation futile.

To overcome this problem, the methodology of dynamically switching the co-simulation models used is proposed in [2]. The basic idea is to change at run-time the simulation models that are used for a specific part of the co-simulation and for given components. For example, while most of the time a fast less-detailed SystemC simulation model is used for the cable, for special intervals a more-detailed low level VHDL-AMS model is utilized. This speeds up the co-simulation (compared to low level simulation for the whole simulation) while at the same time providing the high accuracy when required. The principle is shown in Figure 1. Splitters and mergers (switches) are used to perform the co-simulation model switching and to ensure a correct data flow within the system. For more details about the basic methodology and the implementation, see [2].

![Fig. 1. Methodology of Run-Time Co-Simulation Model Switching](image-url)
switching occurs. In the case of FlexRay, its periodic and a-priori known behavior can be used to find synchronization points where the system is in a known state (e.g. at cycle, frame or even bit borders). Hence, switching can take place at such points without problems. The definition of switching points is a rather system specific task with the basic goal of determining either periodic or beneficial points during system execution.

B. Performance analysis

In the run-time simulation model switching approach, the developer has the possibility to simply adjust the trade-off between the simulation performance and the model accuracy achieved. This is of great advantage, as in fact the developer can specify the simulation performance in advance, hence adjusting the simulation time needed. By specifying the simulated time (including parallel processing) for each abstraction level of the switched component the developer can select the relation between simulation performance and accuracy according to the requested needs. This is shown in Figure 2. Thus, the simulation performance improvement factor in comparison with standard co-simulation methodologies is more or less freely selectable by the developer. This is of great advantage when simulating complex networks.

![Fig. 2. Shifting the Complexity of Co-Simulation Models](image)

To estimate the achieved simulation performance by applying the run-time co-simulation model switching to our FlexRay network simulation we extended the theory presented in [2]. The total simulated time \( T \) in seconds (s) for each switched co-simulation model consisting of \( M \) different abstraction levels is written as

\[
T[s] = \sum_{n=1}^{M} T_n[s]
\]

This time already includes the defined amount of parallel processing. The total simulated time spent on parallel processing per model abstraction level \( n \) is

\[
T_{P_n}[s] = \sum_{i=1}^{X} t_{P_{ni}}
\]

with \( t_{P_{ni}}[s] \) as the parallel processing time for this abstraction level \( n \) before switch number \( i \) is activated. Following this, the simulated time where the abstraction level \( n \) is running stand-alone because of switch \( i \) is stated as \( t_{S_{ni}}[s] \) and the total simulated time where the simulation model for abstraction level \( n \) is running stand-alone is

\[
T_{S_n}[s] = \sum_{i=1}^{X} t_{S_{ni}}
\]

with \( X \) as the total number of switches including this abstraction level. Summing up, the total simulated time for all abstraction models including parallel processing is

\[
T[s] = \sum_{n=1}^{M} T_n = \sum_{n=1}^{M} (T_{P_n} + T_{S_n})
\]

Out of this, a factor \( w \) for the estimated simulation performance achieved by applying the run-time co-simulation model switching approach can be derived.

\[
w = \frac{1}{\sum_{n=1}^{M} T_n \cdot C_n}
\]

Here, \( C_n[\frac{1}{s}] \) are the implementation dependent simulation costs compared to real-time. This factor has to be derived out of experiments or experience. Typical factors for \( C_n \) are e.g. \( 10^9 - 10^9 \) for VHDL-AMS models or \( 10^2 - 10^4 \) for SystemC models. Following this, the simulation performance depending on the switching configuration and the abstraction levels involved can be estimated as

\[
w = \frac{1}{\sum_{n=1}^{M} \sum_{i=1}^{X} (t_{P_{ni}} + t_{S_{ni}})} \cdot C_n
\]

This allows the designer to compare the estimated performance of different switching configurations. This factor is no absolute value; it only shows the relation between different switching configurations. For example, it can be calculated how a lengthening of the parallel processing time or of the simulated time for one abstraction level affects the total simulation performance of the run-time switched co-simulation. An example calculation will be shown in the next section.

IV. ADVANCED CO-SIMULATION OF DEPENDABLE COMMUNICATION ARCHITECTURES: THE FLEXRAY EXAMPLE

A. FlexRay co-simulation environment

The co-simulation environment developed within the TEDAC project implements a co-simulation of a FlexRay communication framework from the physical layer up to the application, see Figure 3. As co-simulation framework we use the commercially available co-design tool CISC SyAD [11]. Additionally, we use the car simulator CarMaker / AVL InMotion T\( T^M \) [12] to generate realistic data to be transferred via the FlexRay network.

Regarding the simulation models of the network components, we combine models at different abstraction levels and implemented using different HDLs. The physical layer (cables, active/passive star, transceiver) is implemented both in more-detailed VHDL-AMS models and also using fast but less-detailed SystemC simulation models. This provides us with the ability to apply the run-time co-simulation model switching approach to the most computational intensive layer: the physical layer. The FlexRay communication controller (data link layer) is implemented using SystemC.

The middleware (selected AUTOSAR [13] concepts) and the software application are also implemented using SystemC.
and C/C++. Hence, the same application code can be re-used for real hardware FlexRay controllers. By using the run-time co-simulation switching this setup allows us to analyze the complete network behavior and especially all the interactions between the different layers of the network – from the physical layer up to the application – within a reasonable amount of time.

B. Experimental setup

For our experiments we created the sample FlexRay setup shown in Figure 4. The network is consisting of 5 FlexRay nodes including transceivers, communication controllers and software. The topology with different cable lengths and the FlexRay specific line termination is also shown in Figure 4. We applied the run-time co-simulation model switching by inserting SystemC-based splitters/mergers (switches; see Section III) in the co-simulation to handle the switching process. We switched between fast but less-detailed SystemC simulation models for transceivers and cables (only including length-based delay and attenuation) and very slow but more-detailed VHDL-AMS models (including all cable effects like reflections, termination etc.) for these components. The signals to be switched are at the interface between the communication controller and the transceiver: receive data (RxD) and transmit data (TxD). This allows the investigation of effects occurring at physical layer on higher layers like data link layer (e.g. shortening/lengthening of bits, misinterpretations because of low signal integrity) and application (e.g. missing data because of frame corruption, problems due to startup/synchronization errors etc.). Also the other way is possible: effects of the protocol configuration or the bit structure on the physical waveforms can be analyzed.

An important topic when applying the run-time co-simulation model switching is the choice of switching point and parallel processing duration. As FlexRay is a time-triggered communication protocol there exist pre-defined points where the state of the system is completely known.

Hence, it is the most advantageous solution to select the switching points there with a parallel processing time large enough to allow proper initialization of the co-simulation model to be switched to. In our experiments, we defined that most of the time the fast SystemC model for topology and transceivers is used. However, in every frame the header is transmitted via the more-detailed but slow VHDL-AMS models.

In the experiments the following FlexRay-related configuration is used: A total of 16 FlexRay communication cycles is simulated, including startup and integration of the nodes. Each cycle lasts 5ms and, if all nodes are integrated, contains 12 FlexRay frames. Each frame is 26µs long and transmitted at 10MBit/s. In each frame, the first 6µs ($t_{S_1}$) are transmitted via the more-detailed VHDL-AMS ($n = 1$) models of topology and transceiver. The remaining part ($t_{S_2} = 20µs$) of the frame is transmitted using the fast SystemC ($n = 2$) models, see Figure 4. We assume that the attributes of the signal integrity will not change within one frame and therefore the detailed analysis of the first 6µs will provide relevant results for the entire frame. In this experiment, up to 12 switches occur per cycle. Based on the FlexRay schedule and experiments, the parallel processing time is set to $t_{P_1} = t_{P_2} = 2µs$.

C. Results

Four different experiments have been performed based on the setup described in Section IV-B. Experiment one was the setup shown in Figure 4 but using SystemC simulation models only, hence, without switching. The second experiment was performed using VHDL-AMS only for the models of transceiver and cables. Here, also no switching was used. The third experiment applied the run-time simulation model switching approach to the co-simulation. The switching configuration as described in Section IV-B was used and switching
occurred after the FlexRay header (6µs) has been transmitted. For these first three experiments, a valid FlexRay network was simulated (correct cable length and termination). In the fourth experiment, the network topology was modified. The line from the bus to node 3 has been altered from 0.8m to 7.0m. The faulty termination concept led to several physical effects (e.g., reflections) that have degraded the signal integrity within the FlexRay network. For this experiment, the same switching setup as in experiment three was used.

In the first three experiments, the switching within the frames worked as expected and the simulation performance was improved compared to pure VHDL-AMS simulation. A comparison is shown in Table I. As the network topology was designed according to the FlexRay specification, the frames were transmitted correctly and the upper layers worked without problems.

However, the situation is a bit different for the fourth example: here, the setup can definitely be regarded as borderline case because of the 7.0m long, unterminated line to node 3, leading to reflections. The simulation demonstrated that, depending on the transmitting node, different nodes have problems in receiving correct frames. For example, the data node 2 is sending can be received without any problems by nodes 0, 1 and 4, while node 3 reports a header error check failure. However, if node 1 is transmitting the nodes 0, 2 and 3 receive the data correctly, while node 4 reports an error in the frame header. These effects lead to reception asymmetry that can move the system into inconsistent states and thus should be avoided.

In Figure 6 it can be seen that for a frame transmitted by node 4 (TxD4), node 3 using the VHDL-AMS model is not able to receive any meaningful data on RxD3 because of reflections etc. Hence, the communication controller very early reports an error that it is not able to find a byte start sequence (BSS), see Figure 5. The problems can be easily detected by looking at the resulting differential voltage waveform at node 3. It is far away from any meaningful state. In Figure 6, the SystemC topology model starts parallel processing at around 285.749ms before switching at 285.751ms. During parallel processing the output of the SystemC model is not used. It can be seen that the SystemC model does not show any distortion (because it only includes length-based delay and attenuation); hence, no error would be detected during the whole simulation if the SystemC model would be the primary model all the time. The situation is a bit different for nodes 0, 1 and 2. In fact, using the VHDL-AMS model they are able to receive the header data (transmitted by node 4) correctly most of the time. But at around 285.750ms, one bit sequence gets altered by a reflection. The high bit sequence is shortened and the following low bit sequence is stretched, leading to a header checksum error detected by the controllers, see Figure 5. By looking at the SystemC data it is obvious that this error does not happen there. These errors definitely influence the application as, depending on the sender node, a node may or may not receive frames correctly. At the worst, the FlexRay network may lose its synchronization and get split apart in different clusters of nodes that are not able to communicate with each other (cliques). The switched simulation detected all of the higher level errors like shown in Figure 5 that were also detected during the full VHDL-AMS only simulation run.

<table>
<thead>
<tr>
<th>Seconds</th>
<th>100% VHDL-AMS</th>
<th>Switched</th>
<th>100% SystemC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor</td>
<td>287</td>
<td>104</td>
<td>1</td>
</tr>
</tbody>
</table>

Table I: Example Simulation Time Comparison for one FlexRay Cycle with 12 Frames

With these experimental results we are able to verify the simulation performance estimation proposed in Section III. We disregard the negligible simulated times with no frame transmission (for easier determination of $C_{AMS}$ and $C_{SYSC}$) and use the switching configuration described in Section IV-B. Based on experiments we determine $C_{AMS} = C_1 = 10^6$, $C_{SYSC} = C_2 = 3 \cdot 10^3$ and $i = 12$ (as 12 frames are transmitted in one cycle). By using Formula 5 it is calculated that the simulation performance difference between pure VHDL-AMS simulation and the switched simulation should be about 3.22. Via Table I it is ascertainable that the actual difference is 2.79, a discrepancy of about 15% to the estimation. Considering that the values of $C_{AMS}$ and $C_{SYSC}$ are basic approximations to the reality this is a rather good result. The simulation performance of the switched co-simulation compared to the SystemC only simulation is estimated by Formula 5 as 103 while the results of Table I show a simulation performance difference of 104, hence nearly identical results.

V. CONCLUSION

Holistic simulation of automotive communication architectures is strongly required for an early system integration as well as for an efficient cross-domain design exploration and optimization. The highly heterogeneous nature of the system under consideration strongly limits the simulation (and therefore the analysis) capabilities. In this work, we have integrated a run-time model switching method within our heterogeneous co-simulation model of a FlexRay network. The rationale for the expected performance increase has been presented and further experimentally validated. Another important result of this work is the fault propagation analysis within the communication architecture. Hence, we have pointed out that the same faulty topology might present very different symptoms and lead to different reactions of the system.
**ACKNOWLEDGMENT**

The authors wish to thank the "COMET K2 Forschungsförderungs-Programm" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economics and Labour (BMWA), Österreichische Forschungsförderungsgesellschaft mbH (FFG), Das Land Steiermark and Steirische Wirtschaftsförderung (SGF) for their financial support. Additionally we would like to thank the supporting companies and project partners austriamicrosystems, AVL List and CISC Semiconductor as well as Graz University of Technology and the University of Applied Sciences FH Joanneum. Further information about the TEODACS project can be found on www.teodacs.com.

**REFERENCES**


Fig. 6. TxD/RxD Excerpt of a Switched FlexRay Frame