TIMMO-2-USE
Timing Model – Tools, algorithms, languages, methodology, USE cases

Report type: Internal report
Report name: Brake-By-Wire Validator
Report status: Consortium Confidential
Version number: 1.0
Date of preparation: 2012-09-21
TIMMO-2-USE Partners

AbsInt Angewandte Informatik GmbH
Arcticus Systems AB
Chalmers University of Technology
Continental Automotive GmbH
Delphi France SAS
dSPACE GmbH
INCHRON GmbH
Institute National de Recherche en Informatique et Automatique INRIA
Mälardalen University
Rapita Systems Ltd, UK
RealTime-at-Work
Robert Bosch GmbH
Symtavision GmbH
Technische Universität Braunschweig
Time Critical Networks (TCN) AB, Sweden
University of Paderborn
Volvo Technology AB

Project Coordinator

Dr. Daniel Karlsson
Volvo Technology AB
Götaverksgatan 10
417 66 Göteborg
Tel.: +46 31 332 9949
Email: Daniel.B.Karlsson@volvo.com

© Copyright 2010: The TIMMO-2-USE Consortium
Authors

Simon Schliecker, Symtavision
Mesut Özhan, INCHRON
Cecilia Ekelin, VTEC
Henrik Lönn, VTEC
Ubaldo Tiberi, VTEC
Lei Feng, VTEC
Reinhold Heckmann, AbsInt
Jonas Lext, Time Critical Networks
Ulrich Kiffmeier, dSPACE
Hamza Qadir Raja, dSPACE
## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2012-09-21</td>
<td>Final version of report</td>
</tr>
</tbody>
</table>
# Table of contents

1 Introduction................................................................................................................. 6
   1.1 Validator description................................................................................................. 6
   1.2 Validator development ............................................................................................. 7
2 System modelling tools .................................................................................................. 8
   2.1 Papyrus .................................................................................................................... 8
      2.1.1 Vehicle Level....................................................................................................... 8
      2.1.2 Analysis Level.................................................................................................... 9
      2.1.3 Design Level .................................................................................................. 13
   2.2 SystemDesk ........................................................................................................... 16
3 Implementation tools ...................................................................................................... 21
   3.1 Simulink ................................................................................................................ 21
   3.2 TargetLink ............................................................................................................. 26
   3.3 ArcticStudio .......................................................................................................... 27
4 Timing analysis tools ..................................................................................................... 30
   4.1 aiT WCET Analyzer ................................................................................................. 30
   4.2 TCNAnalyzer ........................................................................................................ 34
   4.3 SymTA/S ............................................................................................................... 35
   4.4 INCHRON Tools .................................................................................................... 40
5 Tool chains .................................................................................................................... 45
   5.1 Implementation tool chain (Simulink-TargetLink-ArcticStudio)................................. 45
   5.2 Timing analysis tool chain (SystemDesk-TargetLink-aiT-SymTA/S) ......................... 45
   5.3 XML based tool chain ............................................................................................ 47
6 Summary ......................................................................................................................... 49
7 References ..................................................................................................................... 50
1 Introduction

This report contains a more detailed description of the Brake-By-Wire (BBW) validator developed as part of the TIMMO-2-USE project. The report extends the text in D14 [1] by providing more elaborate descriptions and pictures of the models and tools that have been used to perform the validation activity.

The objectives of the validation were:

- To develop the BBW system using a model-based approach.
- To apply the methodology provided by TIMMO-2-USE.
- To express timing constraints using TADL2.
- To perform timing analysis of the modeled BBW system.
- To investigate how the modeling and analysis tools can be used in a tool chain.

This report documents how these objectives have been addressed.

1.1 Validator description

The validator implements a brake-by-wire (BBW) application with ABS, where no mechanical connection exists between the brake pedal and the brakes applied to the wheels. A sensor attached to the brake pedal reads the pedal angle, which is used to compute the desired brake torque. The ABS algorithm uses this brake torque request together with the vehicle speed and the wheel speeds to compute the actual brake torque to be applied to the wheels. The target platform comprises five ECUs which are connected using Ethernet in a ring topology. (See also deliverable D6.) The topology and main functional blocks of the BBW application are illustrated in Figure 1.

Figure 1 BBW hardware topology and functional components

The BBW application has been developed in Simulink. The Simulink model has then been migrated to TargetLink from which AUTOSAR SWC C code has been generated. This code has in turn been packaged into an AUTOSAR software architecture using ArcticStudio. This tool has also been used for the configuration of the AUTOSAR BSW and for the building of executables for the ECUs. Thus the ECUs run the AUTOSAR platform from ArcCore. The BBW
application is distributed over a network of ECUs, where each wheel node manages the functionality of a particular wheel (i.e. SpeedXX, ABSatXX and BrakeXX) and the pedal node manages the overall brake control (i.e. Pedal, BrakeTorqueCalc and GlobalBrakeController). The pedal node also hosts the simulation of the vehicle dynamics.

### 1.2 Validator development

The workflow used for developing the BBW application is illustrated in Figure 2. In the figure, the bubbles represent the different tools. The black bubbles to the right are the development tools while the red bubbles to the left are the analysis tools. The artefacts used or produced by the tools are listed in green in the middle. For each tool it is also mentioned which aspects the tool was used for. The tool chain is in effect as the tools operate on the same set of artefacts and/or import/export artefacts from/to the other tools. The workflow follows more or less a clockwise motion through the tool bubbles.

Figure 2 Tools and artifacts used in development of the BBW validator

In the following sections the description of the development activities will be organized based on the individual tools and how they have contributed to the overall development and TIMMO-2-USE validation. There is also a section describing the particular tool chains in more detail. The tool descriptions are organized into the sections “System modelling”, “Implementation” and “Timing analysis”.

---

**Please note:**

- The text is extracted and reformatted into a natural representation.
- The diagram is preserved as a visual aid.
- All content is maintained as accurately as possible without introducing new facts or interpretations.

---

**Author:**

**Title:**

**Page:**

**Version:**

**Internal Report – Brake-By-Wire Validator**

**1.0**

**7**
2 System modelling tools

These tools are used for modelling the structure of the system and to provide annotations of timing constraints.

2.1 Papyrus

Papyrus version 0.8.1 with a UML profile for EAST-ADL 2.1.9 is used to make an EAST-ADL model of the BBW system. This allows to model timing constraints using TADL concepts included in the EAST-ADL profile. The BBW system model consists of two parts: i) structural modelling of the BBW system and ii) modelling of timing constraints using TADL2. Due to the lack of an EAST-ADL profile for the Implementation and Operational level, the Papyrus model contains structure and timing models only for Vehicle, Analysis and Design level. Finally, since in this small example all the requirements are considered to be relevant for the next phase, Step (1) “Refine, introduce and validate requirements” and Step (7) “Specify and validate timing requirements” of the GMP are not conducted in none abstraction level.

Validation goals

The purpose of the Papyrus model was to capture the structure of the BBW system and to express the timing constraints imposed on the system using TADL2. The model is used as input for the analysis tools. As part of this modelling activity, the methodology steps have also been exercised, and the following use cases have been considered:

UC#0001 – Specify timing budgets
UC#0002 – Specify mode-dependent timing information
UC#0005 – Develop control applications
UC#0007 – Develop application and infrastructure
UC#0008 – Exchange models
UC#0010 – Specify synchronization timing constraints

2.1.1 Vehicle Level

Methodology

At this abstraction level, the following steps of the GMP have been addressed:

- Create solution
- Attach timing requirements to solution
- Create timing model

The tasks “Analyze timing model” and “Verify solution” have not been addressed since there is not an analyzable solution yet, but the specification is provided as a high level requirements and features.
Figure 3 Vehicle Level timing model

Modeling
As shown in Figure 3, the structural model is not yet concrete, but only a specification at a higher level of abstraction is provided. However, a timing model is present.

TADL2
The timing constraints imposed on the system include a reaction constraint between the brake pedal input event and the actuators output events. Such reaction constraint reads: “The vehicle shall start to brake within 5 meters after the brake pedal is pressed.” This means that the effect experienced on the actuator output due to a braking action must occur within a certain time bound. Notice how such a bound is expressed in meters, which is a multiform time expression. Therefore, at this level, the TADL2 timing constraints used are

- Reaction constraint
- Multiform timing constraint

2.1.2 Analysis Level

Methodology
At this abstraction level, the following steps of the GMP have been addressed:

- Create solution
- Attach timing requirements to solution
- Create timing model

The Task (5) “Analyze timing model” and the Task (6) “Verify solution against timing requirements” have not been addressed since timing analysis support is more challenging on more concrete abstraction levels.

Figure 4 Analysis Level - Functional Analysis Architecture (FAA)

**Modeling**

Illustrated in Figure 4, the system is composed by four wheel speed sensors, which measure the wheels speed, one brake pedal sensor, which measures the braking pedal position and four actuators connected to the wheels of the vehicle which generate the physical braking force. The system also includes a vehicle speed estimator represented by the “VehicleSpeedSensor” component, four ABS components, and two controller components denoted as “BrakeTorqCalc” and “BrakeController”.

Internal Report – Brake-By-Wire Validator 1.0
The components are connected as follows: the wheel speed sensors are connected to the near environment, together with the brake pedal sensor. Information provided by the wheels speed sensors serve as input to both the vehicle speed estimator and the ABS components. The ABS components take the vehicle speed estimate and the desired torque for each wheel as additional input. Such additional input is provided by the “BrakeController” component. The ABS components generate a torque which is the command input to the actuators. The actuators apply the requested torque to the near environment. As we described, the “BrakeController” component generates individual torques that are fed into each ABS. Such individually required torques are generated by partitioning a global requested torque (input to the “BrakeController” component) which depends on the pedal position. The component that has the task of converting the braking pedal position (expressed in percentage) to a desired torque is the “BrakeTorquCalc” component. Such “BrakeTorquCalc” component is connected to the braking pedal position sensor, which in turn is connected to the near environment.

Figure 5 Analysis Level - Multiform time reaction constraint
Given the current solution, the timing constraints from the vehicle level are refined, and new timing constraints are added. For instance, Figure 5 illustrates the reaction constraint between the braking pedal sensor input and the actuators output. At this level the constraint is split by considering four different reaction constraints. In these four constraints the stimulus is common and it is represented by the input reading event of the braking pedal, while the responses are represented by the output event of each actuator. Additional reaction constraints are imposed between the wheel sensors input reading events and the actuators output events. Figure 6 illustrates one of two synchronization constraints introduced at this level. One synchronization constraint is applied to the input event of the wheel sensors, since the data from such sensors should reflect the near environment at the same point in time. The second synchronization constraint is imposed on the actuator output since we want that the effect of braking occur at the same time on all the wheels. The reaction constraints are still expressed in multiform time,
while the synchronization constraints are expressed in physical time. The TADL2 guide used here are the following:

- Reaction constraint
- Multiform timing constraint
- Synchronization constraint

### 2.1.3 Design Level

**Methodology**

At this abstraction level, the following steps of the GMP have been addressed:

- Create solution
- Attach timing requirements to solution
- Create timing model
- Analyze timing model
- Verify solution against the requirements

For the design level, schedulability analysis support has been provided by Symtavision and INCHRON.

**Modeling**

The structural modelling includes the hardware architecture, the functional architecture and the mapping of functions into the hardware.

The solution obtained at the analysis level is refined here. First of all, **Local Device Managers (LDMs)** are included in the design. They represent the software part of the sensor and are needed to interface the sensors and the actuators with the rest of the system. Moreover, the “VehicleSpeedSensor” and the “BrakeController” components are merged into the “GlobalBrakeController” component.

The hardware design architecture (HDA), as in Figure 7, is also considered at this level of abstraction. The system includes 5 ECUs connected in a ring topology according to an Ethernet communication protocol. Four ECUs are respectively associated to the wheels, and there is a central ECU which acts as a main controller and to which the braking pedal is attached.

The mapping of the functional design architecture (FDA) into the hardware is performed at this abstraction level. For instance, the ABS, the “WhlSpdSensorLDM” and the “BrakeActuatorLDM” components are mapped into the four ECUs associated to the four wheels. The “GlobalBrakeController”, the “BrakeTorqCalc” and the “BrakePedalLDM” are mapped into the central ECU.

Finally, Design level and Implementation level models are also modelled in Rubus Designer and SystemDesk, respectively.

**TADL2**

The timing constraints imported from the analysis level are refined according to the generated solution. First of all, the reaction constraints between the braking pedal sensor input and the actuator output, expressed in multiform time, has been split into a mode dependent timing constraint. For the sake of simplicity, we have defined three different modes describes as follows

- Mode 1: $0 \leq v < 30$ Km/h,
- Mode 2: $30 \leq v < 90$ Km/h,
- Mode 3: $90 \leq v < 130$ Km/h,

where $v$ represents the vehicle speed. By using the well-known relation $s=vt$, where $s$ is the distance traveled and $t$ is the time, and

\[
\begin{align*}
\text{Mode 1:} & \quad 600 \text{ ms}, \\
\text{Mode 2:} & \quad 200 \text{ ms}, \\
\text{Mode 3:} & \quad 138 \text{ ms}.
\end{align*}
\]

The four reaction constraints between the wheel speed sensor inputs and the actuator outputs are split into four age constraints and four reaction constraints. The age constraints are placed between the sensors input reading event and the ABS input reading event. Such age constraints mean that the torque computed by the ABS is valid only if the information from the sensors is fresh enough. Once the desired torques are computed, we also wish that they are applied to the wheels as soon as they are computed. Therefore, we placed four reaction constraints between the ABS and the actuators components. The synchronization constraints are the same as in the Analysis Level.

At this abstraction level, we also annotated time periods and execution times of each component (see Figure 9). The values of time periods and execution times are derived from Simulink and through application of aiT on the BBW binary files.

The TADL2 guides that have been included are the following:

- Execution time constraints
- Reaction time constraints
- Age timing constraints
- Synchronization constraints
- Periodic timing constraints
- Mode dependent constraints

Limitations & Assumptions
Since the UML-profile has not implemented TADL2 yet, the timing constraints have been modelled using the existing constructs in EAST-ADL/TADL. The semantics is however to be interpreted as that of TADL2.

Only the functions of the BBW application have been modelled, i.e. AUTOSAR BSW functions are not included. In practice this is needed in order to annotate and analyse the timing properties also of the platform to get a complete understanding of the timing behaviour.

Validation results
The use cases have been adequately addressed by TADL2 and the methodology. The development of the BBW system has however followed a bottom-up approach rather than top-down which makes it hard to validate an improvement in the early phases. For example, analysis tools are typically targeting the implementation level which means that there is no direct support for early phase analysis of timing behaviour.

2.2 SystemDesk

SystemDesk was used to create the AUTOSAR model of Volvos BBW application at the implementation level. In SystemDesk, the developed BBW model was simulated using a 6-Ecu and a 5-Ecu based system architecture. To specify AUTOSAR specific system timing information an add-on to SystemDesk was developed, called SystemDesk Timing Extension Editor (SDTEE).

Validation goals
The following use cases and features were investigated:
- UC#0001 - Specify Time Budgets
- UC#0002 - Specify Mode Dependent Timing Information
- UC#0010 - Specify Synchronization Timing Constraints
- Specify system timing information at various abstraction layers according to AUTOSAR 4.0.3 standard
- Interoperability of multiple tools, i.e., TargetLink, SystemDesk, AbsInt aiT, SymTA/S, INCHRON, using AUTOSAR schema

Methodology
The following GMP steps were addressed on implementation level:
• Create Solution
• Attach Timing Requirements To Solution
• Create Timing Model
• [Analyze Timing Model (through tool coupling with SymTA/S)]

Modeling

At first, the Brake-By-Wire behavioral model was defined in TargetLink by people at Volvo Group truck technology, Sweden. Figure 10 shows the BBW model in TargetLink. It includes a Vehicle_Body_Wheel component, a global brake controller (GlbBrakeCtrl) component, a Brake torque calculator (BrakeTorqCal) component, and four anti-lock braking system (ABS at XXWheel) components. In the model, the BrakeTorqCal component periodically samples the signals from the vehicle brake pedal that also serves as the stimulus in the model, to compute the required brake torque on the four wheels. The req. brake torque was then used by GlbBrakeCtrl to compute the actual brake torque along with the vehicle speed estimate from the four wheels. The vehicle speed estimate was obtained at each wheel from its rotations per minute (rpm). Here it is important to note that the Vehicle_Body_Wheel component was only included to verify/check the system behavior in a closed loop simulation. It represents the environment model and does not belong to the anti-lock braking system controller.

Figure 10: Brake-by-Wire model in TargetLink

TargetLink was then used to generate the AUTOSAR implementation (code) of the components present in the BBW TargetLink model after simulating the model behavior. The generated code files along with
the AUTOSAR software component descriptions were then imported in the BBW *SystemDesk* model.

In the second step, a complete BBW software model was developed in *SystemDesk* according to AUTOSAR specifications. Figure 11 shows the BBW software architecture in *SystemDesk*. It consists of variety of software components and compositions. The model includes a *VehicleBody*, and a *WheelAssembly composition*. The composition *VehicleBody* includes a software component i.e., *Swc_Vehicle*, and a composition i.e., *BrakePedalSystem*. The *BrakePedalSystem* composition further includes three software components, namely, a *Swc_BrakePedalInputHandler*, a *Swc_BrakeTorqCal*, and a *Swc_GlbBrakeCtrl*. And, the composition *WheelAssembly* comprises on two software components, namely a *Swc_ABS_at_Wheel* and a *Swc_WheelBrakeActuatorHandler*. Again, the environment model *VehicleBody* is only included to perform offline closed-loop simulations.

Figure 11: Brake-by-Wire software architecture in SystemDesk
As briefly described above, in the Brake-by-Wire model the stimulus originates from the input $RP_{\text{RawPedPos}}$, i.e., Brake Pedal Position, present on the $Swc_{\text{BrakePedalInputHandler}}$ software component. The response of the particular stimulus after getting processed through different software components appears on the port, i.e., $PP_{\text{actBrkTorq_Nm}}$ of $Swc_{\text{ABS_at_Wheel}}$.

To simulate the BBW system model, the software architecture shown in Figure 11, was at first mapped on a 6-ECU based system architecture and later it was mapped on a 5-ECU based system architecture in SystemDesk. The 6-ECU model consists of a separate Vehicle Model, a Brake Input Handler ECU, and four wheel ECUs. And, in the 5-ECU based system architecture the vehicle software component and the brake input handler composition were combined in one ECU, called as the Vehicle ECU.

Both BBW system models in SystemDesk were then simulated separately using dSPACE Offline Simulator (VEOS) to compare the system behaviour with the change in system architecture and with the TargetLink BBW model. Figure 12 shows the simulation plots of the BBW system in VEOS.

![Simulation plots of BBW system in VEOS](image)

Figure 12: Simulation plots of BBW system in VEOS

To specify the BBW system timing information as per AUTOSAR 4.0.3 specifications, a prototype version of the SystemDesk Timing Extensions Editor (SDTEE) has been developed. Figure 13 shows a screenshot of the Timing Extensions Editor integrated in SystemDesk. SDTEE enables the user to specify the system timing information in terms of events, event chains, and constraints on them at various levels of detail according to the AUTOSAR 4.0.3 standard. It allows the user to specify the system timing information at virtual functional bus (VFB), software components (SWC), System, basic software (BSW) module, and electronic control unit (ECU) level. The values of the high-level timing requirements were obtained from the design level (EAST-ADL, Papyrus) and manually entered in the AUTOSAR timing model. In the time-budgeting step, the given end-to-end latency requirements were split into smaller parts along the respective event chains.
TADL2

The SystemDesk Timing Extension Editor (SDTEE) was developed to allow the users to specify the system timing information according to the AUTOSAR 4.0.3 Timing Extensions. SDTEE does not yet support all the enhancements proposed in TADL2 even though it includes most of the new features proposed/addressed in TADL2.

Limitations & Assumptions

The BBW system composition was manually converted from the Papyrus model and the implementation of its software components were taken from the TargetLink BBW model.

Considering the limitations of the modelled system, the SystemDesk model only includes a very simple model of the basic software (BSW) load. Another limitation of SystemDesk appears in communication topology, i.e. SystemDesk does not support Ethernet communication drivers. Therefore, a CAN bus was used instead. Note, that in the AUTOSAR ECU architecture the upper levels of the COM-Stack (Com, PduR) are identical for CAN and Ethernet.

Validation results

SystemDesk with SDTEE supports the following use cases:

• UC#0001, UC#0002, UC#0010

SDTEE fulfills all standard specifications of AUTOSAR 4.0.3

The following tool couplings have been established and successfully applied in the BBW validator:

o TargetLink ↔ aiT

o SystemDesk ↔ SymTA/S

o INCHRON was able to import the AUTOSAR BBW model exported by SystemDesk.
3 Implementation tools

These tools are used for developing the BBW application and to make it execute on the selected hardware topology and software platform (AUTOSAR).

3.1 Simulink

Simulink has been used to model and test (by simulations) the behavior of the brake-by-wire system at the Design Level. The FDA is taken from the respective Papyrus model, and it has been implemented.

Validation goals

We aim at validating some use cases defined in the project. For instance, the following use-cases have been addressed:

- UC#0001 Specify time budgets
- UC#0002 Specify mode dependent timing information
- UC#0003 Revise erroneous timing information
- UC#0005 Develop control applications
- UC#0008 Exchange models

Once the model has been created with UC#0005, several simulations have been iterated by changing the solution and by deriving new timing properties with UC#0003. This task has been performed in order to determine reasonable time requirements for the next level of abstraction. Such timing requirements are specified with UC#0001 and UC#0002. Finally, the model is used as input to TargetLink through UC#0008.

Methodology

The following GMP steps have been used for any addressed use-cases:

- Create Solution (behavior model)
- Create timing model (periods added)
- Analyze timing model (preliminary delays established)

Modeling

In Figure 14, the longitudinal vehicle dynamics have been modeled according to the Newton Law. A four wheels vehicle has been considered. The vehicle dynamics are controlled by a control system composed by the following elements:

- One brake pedal sensor;
- Four wheel speed sensors;
- Four brake actuators;
- An ABS module for each wheel;
Figure 14 - Simulink model of the brake-by-wire
- A brake torque calculation;
- A global torque controller.

The pedal position is measured by the brake pedal sensor. Information about the pedal position is sent to the Brake_Torq_Calculation which converts the pedal position, expressed in percentage, into a desired torque, expressed in Nm. The value of the required torque enters to the block Global Brake Controller which distributes the overall torque to each single wheel and it estimates the vehicle speed. The individual torque provided by such a block, serves as input to the ABS together with the wheel speed measurements and the vehicle speed estimator. Given these information, the ABS decides if lock or unlock the wheel based on the computed slip rate. The output of the ABS, which is still a torque sent to the braking actuator.

**Limitations & Assumptions**

In the modeling of the vehicle dynamics some forces, e.g. air drag force, have been neglected, and there is no coupling among the wheels. Moreover, the continuous-time model of the vehicle dynamics is converted into a discrete-time because the vehicle dynamics will be simulated with an ECU. No real wheel speed sensor and brake actuator has been modeled, and it is assumed that they operate in a continuous-time fashion, i.e., the measurements are continuously picked by the sensors and the actuators are continuously updated. The ABS module works in an on/off fashion, which means that if the slip rate is less or equal than 0.2, then the ABS delivers the brake torque to the wheel, and if the slip rate is greater than 0.2 (i.e. the wheel is locked), then the ABS release the brake on the wheel. The ABS is active only if the vehicle speed is greater than 10 Km/h. The global torque is distributed to each wheel with a fixed distributor. The estimate of the vehicle speed does not consider if the vehicle is operating in sliding, skidding or pure rolling mode, but just performs an average between the individual wheels speed to estimate the vehicle speed (it assumes it works in pure rolling mode). Finally, a dummy accelerator signal is added to the vehicle dynamics model.
Validation results

The simulation results are depicted in Figures 15-18. The periods are set to 10 ms for the blocks brake_Torq_Calculation and ABS. The global brake controller operates periodically with 20 ms period, and, finally, the discrete-time model of the vehicle dynamics operates with a period of 5 ms. It is interesting to see how the ABS acts on two wheels for which it has been detected a slip rate of 0.2. However, since the speed rate value is computed through an estimate of the vehicle speed, the ABS actually acts on a threshold that is different than 0.2. The deviation of the threshold used by the ABS and the desired value of 0.2 depends on the quality of the vehicle speed estimate. However, the vehicle correctly performs a braking action with the current setup. The timing properties of this model are used as requirements for the next abstraction level.
Figure 17 - Applied braking torque

Figure 18 - Wheels speed
3.2 TargetLink

TargetLink enhances Simulink model for the automatic code generation of efficient C code of the major Simulink function blocks shown in Figure 14. In the development of the BBW use case, TargetLink is employed to (1) generate AUTOSAR compliant C code for the final implementation on the ECUs and network, (2) define the interface ports and corresponding data types of the AUTOSAR software components, (3) derive the scaling factors of the fixed-point data types in the software, and (4) generate the AUTOSAR configuration arxml file.

Validation goals

The TargetLink model is a refinement of the Simulink model aiming for generating efficient and AUTOSAR-compliant C code of the Simulink functions. Therefore, it validates the same goals as does Simulink.

- UC#0001 Specify time budgets
- UC#0002 Specify mode dependent timing information
- UC#0003 Revise erroneous timing information
- UC#0005 Develop control applications
- UC#0008 Exchange models

The exchange of models includes both the exchange between Simulink and TargetLink and the exchange between normal TargetLink model and AUTOSAR-compliant TargetLink model.

Methodology

Identical to the use of Simulink, the following GMP steps have been used during the development of the BBW use case:

- Create Solution (behavior model)
- Create timing model (periods added)
- Analyze timing model (preliminary delays established)

Modeling

The TargetLink model is converted from the Simulink model in 4 using the Model Conversion tool provided by TargetLink. The conversion applies to the following Simulink function blocks: Brake_Torq_Calculation, Global Brake Controller, ABS_RR_Wheel, Vehicle Body Wheels, and Vehicle_Speed_Estimator. Since the ABS functions at the four wheels are identical, only one function block is prepared for code generation with TargetLink.

Many production processors and network buses do not support floating point data type. In the BBW application, we limit ourselves to fixed point data type. The determination of the scaling factors is derived with the help of the scaling tool of TargetLink. The user specifies the bit-length of the input data and the range and then TargetLink can automatically calculate the suitable scaling of all variables and parameters.

To generate AUTOSAR configuration files and AUTOSAR-compliant code, we further update the TargetLink model with AUTOSAR
features with the free TargetLink AUTOSAR Migration tool. The tool can automatically enhance the TargetLink function blocks with AUTOSAR interfaces and data ports.

After all these preparations, the layout of the TargetLink model is still the same as Figure 14, whereas most of the Simulink function blocks with the double data type are replaced by TargetLink function blocks with fixed-point data types.

**Limitations & Assumptions**

The TargetLink model inherits all limitations and assumptions of the Simulink model. To generate efficient C code, a few Simulink computation blocks are simplified to avoid 64 bit word size or small denominator. These simplifications, together with the tiny rounding error caused by the fixed-point data type, slightly vary the simulation outputs of the Simulink model. Fine tuning on the TargetLink model is necessary to have close resemblance to the output of the Simulink model. The simulation result with the TargetLink model is closer to the actual outcome of the final implementation.

After obtaining the working TargetLink model, all further improvements and upgrades on the BBW application are made directly on the TargetLink model. Benefiting from the automatic code generation function of TargetLink, new versions of the software can be produced immediately from the updated TargetLink model. Model-based analysis methods should be applied to the TargetLink model instead of the Simulink model.

**Validation results**

Given the same input signals as in Figure 16, the TargetLink model produces similar response as in Figure 15, Figure 17 and Figure 18. The difference between the responses of the Simulink model and the TargetLink model is ignorable.

### 3.3 ArcticStudio

**Validation goals**

The role of Arctic Studio is to support the implementation of the Brake-by-wire function on ArcCore’s AUTOSAR platform ArcticCore, running on FlexECU-II ECU hardware connected via Ethernet.

**Methodology**

ArcticStudio was used in the Create Solution step of the Implementation Phase.

**Modeling**

Models corresponding to the SWC template, System Template and ECU resource template were created in ArcticStudio.
The models included runnable to task mapping, setting priorities and periods for the tasks. The table below illustrates the runnables and their parameters.

<table>
<thead>
<tr>
<th>Runnables</th>
<th>ECU</th>
<th>Task</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Veh_Body_Wheels</td>
<td>BrakeControlECU</td>
<td>SWC_TASK3</td>
<td>5 ms</td>
</tr>
<tr>
<td>BrakeTorque_Cal</td>
<td>BrakeControlECU</td>
<td>SWC_TASK2</td>
<td>10 ms</td>
</tr>
<tr>
<td>BrakeControl</td>
<td>BrakeControlECU</td>
<td>SWC_TASK1</td>
<td>20 ms</td>
</tr>
<tr>
<td>FrontLeftWheel</td>
<td>BrakeECU_FL</td>
<td>SWC_TASK1</td>
<td>10 ms</td>
</tr>
<tr>
<td>FrontRightWheel</td>
<td>BrakeECU_FR</td>
<td>SWC_TASK1</td>
<td>10 ms</td>
</tr>
<tr>
<td>RearLeftWheel</td>
<td>BrakeECU_RL</td>
<td>SWC_TASK1</td>
<td>10 ms</td>
</tr>
<tr>
<td>RearRightWheel</td>
<td>BrakeECU_RR</td>
<td>SWC_TASK1</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

All tasks were given the same priority and the scheduler was fully preemptive. The extended task model was used (i.e. tasks may self-suspend) although the basic task model could have been selected for this application.

**Limitations & Assumptions**

In the implementation several design function prototypes are combined into a fewer number of runnables. E.g. the LDM functions are “removed”. The reason is that the code (and its structure) is based on the TargetLink model which did not contain the LDM functions.

**Validation results**

Application code generated from Simulink models in Target link was integrated in SWC stubs generated by ArcCore SWC builder.
ArcCore RTE builder, BSW builder and ECU extract builder was used to generate code and configure AUTOSAR basic software. The resulting software was downloaded and executed on the 5 ECU Ethernet-based BBW cluster.
4 Timing analysis tools

These tools are used to analyze different timing properties of the BBW system.

4.1 aiT WCET Analyzer

AbsInt's aiT tool determines safe and precise upper bounds for the worst-case execution times of tasks in real-time systems. In the validation, aiT was called directly from the TargetLink tool using the integrated AbsInt toolbox. The resulting WCET information was imported into the SystemDesk BBW model. The WCET results have also been used as input for the simulations in the INCHRON Tool-Suite and in the worst-case response time analysis using SymTA/S.

Validation goals

The purpose of the validation was to find out to what extent the tasks of the BBW validator can be analyzed for WCET, and to test the communication with the other tools, in particular the connection with the TargetLink tool. The following use cases and tasks were considered:

- UC#0001 - Specify Time Budgets
- UC#0002 - Specify Mode Dependent Timing Information
- Capture, Analyze, and Utilize Worst Case Timing Information
- Perform timing analysis on code level

Methodology

Since the aiT tool works on binary executables, its usage is mostly confined to the implementation level. The related TimingExplorer tool can also be applied at design level after having generated and compiled some prototypical code, typically by employing a model-based code generator.

Being an analysis tool, aiT is typically applied in the GMP step “Analyze Timing Model”. Its results are usually fed into other tools such as SymTA/S, SystemDesk, or the INCHRON tool suite.

TADL2

As explained above, aiT is employed in the validator as an auxiliary tool that provides input for other tools. Therefore, it is not directly concerned with the TADL2 language.

Modeling

As a code-level analysis tool, aiT does not itself model the application, but relies on the binary code generated by the modelling tools. In the case of the BBW validator, this code is given by five executables, one for the control unit (BrakeControlECU.elf) and one for each of the ECUs at the four wheels (BrakeECU_FL.elf, BrakeECU_FR.elf, BrakeECU_RL.elf, and BrakeECU_RR.elf). Two sets of executables have been used in the validation, one compiled by Volvo with a GCC compiler and one compiled by dSPACE from its SystemDesk model.
On the other hand, aiT relies on a hardware model to perform its analyses. The tool contains parameterizable hardware models for all supported architectures. In case of the BBW validator, the following hardware configuration was specified by Volvo:

- Target processor MPC5567, rev. 416;
- Clock rate 133 MHz;
- External bus division factor 2;
- Flash module with address pipelining disabled and 7 read wait state cycles;
- Unified 2-way cache with store buffer and streaming enabled;
- Two memory banks:
  - Bank 0 at 0xA0000000..0xA000FFFF
  - Bank 2 at 0x20000000..0x2000FFFF
  - For both: port size 16, cycle length 2, burst length 0;
- Stack address: 0x4000f8e0;
- Memory area 0x40000000..0x40014000 is read-write;
- Memory area 0x00000000..0x00200000 is read-only.

**Algorithms**

aiT performs a static WCET analysis that is able to cover all possible program executions to obtain safe upper bounds of the WCET. In order to avoid too conservative WCET estimations, variable ranges and loop bounds can be constrained. For TargetLink software components, this information is automatically obtained from the Data Dictionary.

**Validation results**

The integration of aiT in TargetLink of dSPACE is well established. aiT can be called from TargetLink via the integrated AbsInt toolbox. Upon invocation from TargetLink it gets automatically the input it requires, and automatically communicates the generated results back to TargetLink for inclusion into the SystemDesk BBW model.

aiT can be successfully applied to all relevant tasks in both sets of executables. It is able to produce results sufficient for verifying all given timing constraints by the tools that use aiT results.

Concerning their analyzability, the routines in the executables provided by Volvo fall into three classes.

The first class consists of routines corresponding to TargetLink nodes (listed in the table below). aiT can analyze these entries without further annotational efforts by only using information from the dSPACE data dictionary. The analyzed worst-case execution times are as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>WCET (cycles)</th>
<th>WCET (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrakeControl</td>
<td>2021</td>
<td>15.196 µs</td>
</tr>
<tr>
<td>BrakeTorque_Cal</td>
<td>533</td>
<td>4.008 µs</td>
</tr>
<tr>
<td>Veh_Body_Wheels</td>
<td>11328</td>
<td>85.173 µs</td>
</tr>
<tr>
<td>Wheel Type</td>
<td>Cycles</td>
<td>µs</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>FrontLeft_Wheel</td>
<td>1390</td>
<td>10.452</td>
</tr>
<tr>
<td>FrontRight_Wheel</td>
<td>1390</td>
<td>10.452</td>
</tr>
<tr>
<td>RearLeft_Wheel</td>
<td>1391</td>
<td>10.459</td>
</tr>
<tr>
<td>RearRight_Wheel</td>
<td>1368</td>
<td>10.286</td>
</tr>
</tbody>
</table>

Further information on the execution paths (in the form of aiT annotations) might lead to lower bounds.

The second class consists of the corresponding runtime environment entries Rte_BrakeControl etc. All these routines have the same structure: they consist of a pre-phase where input signals are read, the execution of the TargetLink node code, and a post-phase where output signals are sent. The pre- and post-phases use functionality provided by the AUTOSAR framework. Some of the called AUTOSAR functions contain loops that depend on the size of the signals used within the model. Therefore, the loop iteration counts cannot be automatically derived and need manual annotations.

Information on the type of the signals was delivered by dSPACE: all signals have type UINT16. This could be translated into information on the size of the signals by inspecting the source code of the open AUTOSAR implementation of ArcCore, which has been used in the BBW validator. The result is that the size of individual signals is 16 bits or 2 bytes. Thus, the unknown loop bounds have been set to 2 for each loop. This led to the following AIS annotation file:

```plaintext
## Do not limit the number of loop contexts
interproc flexible;

### memcpy and memset are called in Rte_*
### with parameter3 == destSize,
### which is the byte length of a signal.
### Signals are UINTs of size 2 bytes.
instruction "memcpy" is entered with r5 = (1..2);
instruction "memset" is entered with r5 = (1..2);

### The size limit is also valid for called functions,
### and the signal bitsize is limited to 16 bits.
instruction "Com_ReadDataSegment"
   is entered with r5 = (1..2), r7 = (0..16);

### The signal width limit of 2 bytes
### leads to the following loop bounds:
loop "Com_ReadSignalDataFromPduBuffer" + 2 loops max 2;
loop "Com_ReadSignalDataFromPduBuffer" + 3 loops max 2;
loop "Com_ReadDataSegment" + 1 loops max 2;
loop "Com_ReadDataSegment" + 2 loops max 2;
loop "Com_WriteSignalDataToPduBuffer" + 2 loops max 2;
loop "Com_WriteSignalDataToPduBuffer" + 3 loops max 2;
loop "Com_WriteSignalDataToPduBuffer" + 4 loops max 2;
loop "Com_WriteDataSegment" + 1 loops max 2;

### We also assume that the error routine
### assert_func is never called:
snippet routine "__assert_func" is never executed;
```
Using these annotations, the following worst-case execution times are obtained:

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Cycles</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rte_BrakeControl</td>
<td>96648</td>
<td>0.727 ms</td>
</tr>
<tr>
<td>Rte_BrakeTorque_Cal</td>
<td>20443</td>
<td>0.154 ms</td>
</tr>
<tr>
<td>Rte_Veh_Body_Wheels</td>
<td>99501</td>
<td>0.749 ms</td>
</tr>
<tr>
<td>Rte_FrontLeft_Wheel</td>
<td>85761</td>
<td>0.645 ms</td>
</tr>
<tr>
<td>Rte_FrontRight_Wheel</td>
<td>85761</td>
<td>0.645 ms</td>
</tr>
<tr>
<td>Rte_RearLeft_Wheel</td>
<td>85762</td>
<td>0.645 ms</td>
</tr>
<tr>
<td>Rte_RearRight_Wheel</td>
<td>85739</td>
<td>0.645 ms</td>
</tr>
</tbody>
</table>

As explained above, the actual signal lengths have been used in deriving these execution times. The times might be reduced with additional information on execution path restrictions.

The third class of routines consists of those named SWC_TaskN (with N = 1, 2, 3). They mainly consist of infinite loops with 2 (or more) back-edges. The overall structure is like this:

```
LOOP:
1. WAIT FOR EVENT
2. GET RESOURCE
3. GET EVENT
4. CLEAR EVENT
5. RELEASE RESOURCE
6. IF SOME CONDITION
   6.1. GOTO LOOP
7. ELSE
   7.1. CALL RTE-FUNCTION
   7.2. GOTO LOOP
```

The code handles asynchronous events. This is not in the scope of aiT, which focuses on code level, but must be handled by system-level analysis.

**Limitations & Assumptions**

aiT can compute execution times of software on a processor. Other times like blocking times or message transfer times are beyond the scope of the code-level analysis by aiT. They are typically handled by the tools that take aiT’s results as input.

**Sensitivity of results**

The results surely depend on the parameters of the hardware configuration, but the amount of this dependence was not checked.

On the other hand, we observed a major effect of the signal size on the calculated WCET. Initially, we assumed a signal size of 8 bytes for all signals, which is the maximum possible in the ArcCore.
AUTOSAR implementation. When we added the information that all signals have 2 bytes, the calculated WCET dropped by about 50%.

In any case, we assume a strong dependence on the selected AUTOSAR implementation. There is a huge difference between the overall WCET and the net WCET, i.e. the WCET without considering RTE calls that reflects the pure implementation disregarding the platform. The application code itself is quite short, but when it is packed into AUTOSAR, many costly RTE calls are added at the beginning and the end. A reason might be that the ArcCore implementation is not that efficient and other implementations might produce more efficient code. Maybe some RTE calls can be avoided or replaced by more efficient ones.

**AUTOSAR BSW considerations**

Generally, it has been proved difficult to find bounds for the iteration numbers of the loops in the BSW functions, and information about the values of function pointers used there for indirect procedure calls. In addition, many BSW functions are recursive, with problems to find bounds for the recursion depth – even given that the source code of the ArcCore implementation was available. For the future, one may hope that AUTOSAR implementers provide some information about these topics, e.g. by means of code annotations.

## 4.2 TCNAnalyzer

**Validation goals**

In the BBW-validator, five three-port Ethernet switches connected in a ring topology are used to interconnect the five ECUs. Accordingly, ECU signals are transmitted in broadcasted UDP frames. TCN TimeAnalyzer was used to compute upper and lower bounds on the forwarding delays of all the different UDP frames. These constraints can then be used to verify the time assigned during the time budgeting to the network transmission part, i.e., of a signal end-to-end delay.

(UC#0001 - Specify Time Budget).

**Methodology**

During the TIMMO-2-USE project, TCN TimeAnalyzer has been extended with functionality to interpret the constructs in the EAST-ADL and TADL languages needed to perform an analysis, for example, the topology of the electrical architecture, timing constraints etc. However, the analysis still needed to be complemented with some information from the implementation level to compute timing bounds, for example, the switch frame buffer sizes. Therefore, one can argue that the analysis was performed partly on design level and partly on implementation level.

On these levels, the GMP steps in which TCN TimeAnalyzer might be used should include
- Create timing model
- Analyze timing model
- Verify solution against timing requirements
Modeling

The Ethernet frames forwarded through the switched Ethernet network in the BBW-validator were given in the form of a communication matrix showing all signals, i.e., their sender and receiver ECUs, periods, data sizes etc. Together with the network topology these data were fed into TCN TimeAnalyzer network model.

Limitations & Assumptions

The Ethernet switches used should currently be of store-and-forward type with shared memory architecture. Preferably, the switches should be tested by TCN to ensure good coincidence with the behavior of the switch timing model in the TCN TimeAnalyzer analysis engine. However, as long as the switch fabric utilization is low to moderate it should be safe to assume that most switches show the same behavior.

TADL2

The computations of the upper bound correspond to the TADL2 Reaction Constraint. However, as TCN TimeAnalyzer currently cannot manage holistic end-to-end latency requirements, the computed frame delay bounds should be fed into another tool that can analyze task scheduling and perform computation of holistic end-to-end timing requirements.

Validation results

The best and worst case UDP frame forwarding delays in the switched Ethernet network were successfully computed and exported to file. This file was placed in the project SVN database as a means to distribute it to other project partners for potential import of the results into models generated by tools from other vendors. As the results file is written in HTML format, automated import and parsing of the results file by other tools should be straightforward.

4.3 SymTA/S

In the scope of this validator, SymTA/S was used for model-based performance analysis of the break-by-wire system both statistically and with respect to worst-case behaviour.

Validation goals

The following goals have been pursued with the SymTA/S timing model:

- **Early assessment of the system’s timing behavior** (acknowledging limitations of design level information) in order to identify hotspots.
- **Sensitivity analysis** in order to identify available slack with respect to deviation in the implementation.
- **Performance analysis of final design** in order to investigate adherence to given timing constraints (provided by TADL constructs)
Methodology

All GMP steps on design and implementation level are addressed by Symtavision tooling and approach.

Modelling

In this validator, SymTA/S models were created in two phases.

The first model ("early phase") was generated through an AUTOSAR import of design level file (exported from Arctic Studio). This model corresponds to a design level model. It does not specify any tasks or specific scheduling of the function blocks, therefore SymTA/S was used to auto-generate the missing information (i.e. tasks were created based on runnable periods, task priorities were assigned according to rate monotonic priority assignment).

The second SymTA/S model ("implementation phase") was generated through the AUTOSAR import (from dSPACE SystemDesk) and the import of the task-level timing information derived with AbstInt aiT. The model corresponds to the implementation level and contains the involved software components, runnables, tasks, as well their communication.

The SymTA/S analysis model comprises 5 ECUs, the central brake controller and 4 wheel controllers, each running an AUTOSAR OS with preemptive scheduling that are connected via a 1Mbit/s CAN bus (the speed is to emulate Ethernet timing behavior, see below).

In order to achieve the goal of investigating different implementation alternatives, the model was also modified directly in SymTA/S.

Basic Software (BSW) Functions

Except for the basic software modules involved in the communication, the basic software was not detailed in the model. Instead non-specific main functions have been included in the model. Different execution times of these main functions have been assumed in order to assess the impact of different BSW implementations.

TADL2

The latency constraints concerning the event chains between pedal actuation and BrakePedalActuatorFL_EventChain have been imported into the SymTA/S model based on the SystemDesk output. In ad-
dition, several generic constraints have been introduced (e.g., load < 100%, task deadlines = periods) in order to assess schedulability.

The constraints in this validator are based on TADL constructs, but due to the early stage of this format, the actual exchange of the constraint information in this validator is done via AUTOSAR 4.0.3 timing extensions as well as manual remodelling using constraints in SymTA/S.

Limitations & Assumptions

In SymTA/S the Ethernet bus in the validator has been represented by a CAN network with similar timing properties.

Validation results

SymTA/S was used to apply the following algorithms for timing analysis:

- Worst-case response time analysis
- Worst-case end-to-end latency analysis
- Distribution analysis for end-to-end analysis

In particular, the following basic analyses were performed in this validator.

Analysis of application load

The worst-case analysis provides an initial overview over the system load. The central body ECU services the software components “Swc_BrakeTorqCal”, “Swc_GlbBrakeCtrl”, and “Swc_Vehicle” and thus is in the focus of the analysis.

Based on the worst-case execution time analysis of the involved runnables (as provided by Absint tooling), the load of these application functions is 24.472% (excluding any asynchronous basic software calls). The load is a function of the task execution time and its period or trigger interval.

The following table shows the top 5 runnables with respect to imposed processor load on the body ECU:

<table>
<thead>
<tr>
<th>Runnable</th>
<th>Period</th>
<th>Order</th>
<th>Core Execution Time</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Vehicle.Body.Wheels</td>
<td>Swc.BrakeTorqCal</td>
<td>0.205 s</td>
<td>1</td>
<td>0.205 s</td>
</tr>
<tr>
<td>2 BrakesPedL쟝</td>
<td>Swc.BrakeTorqCal</td>
<td>0.21 s</td>
<td>1</td>
<td>0.436 s</td>
</tr>
<tr>
<td>3 ABS.L谴责</td>
<td>Swc.BrakeTorqCal</td>
<td>0.31 s</td>
<td>0</td>
<td>0.31 s</td>
</tr>
<tr>
<td>4 BrakeTorqCal</td>
<td>Swc.BrakeTorqCal</td>
<td>0.31 s</td>
<td>0</td>
<td>0.31 s</td>
</tr>
<tr>
<td>5 Airpipe_AutoPss</td>
<td>Swc.BrakeTorqCal</td>
<td>0.30 s</td>
<td>0</td>
<td>0.30 s</td>
</tr>
</tbody>
</table>

This gives an indication of the potential hot spots of the design in order to focus design effort.

Analysis of Event Chain Latency

A better indication of real-time schedulability is given by analysis of the constrained event chain. For example, the event chain BrakePedalActuatorFL_EventChain was evaluated to have a total response time of 23.46ms, counting network delays and synchronization effects in the worst case.
When the frame is triggered periodically, then a possible synchronization delay can be observed, leading to 23ms end-to-end latency on the worst-case. This delay lies well within the bounds of all associated age constraints for this chain, which is 600ms in Mode1, 200ms in Mode2, and 138 in Mode3.

**Distribution Analysis**

The synchronization effects can also be observed in typical cases, which is shown by the results of the distribution analysis for this event chain. On average, the latency of this event chain is about 10.9ms.

**Sensitivity analysis**

As the analysis is based on an early phase model (design level), the final implementation is still subject to some variability.

- variations in execution time of application software
- changes in the specification, such as additional functions
- variation in the complexity of the utilized OS and BSW

In order to assess the robustness of the design with respect to such changes, one may be interested in the “timing reserves” that exist. To assess the timing reserves, sensitivity analysis was used. The
following diagram, exported from SymTA/S, shows the maximum possible execution time that additional functions or tasks added to the Body ECU may have without endangering any timing constraints. The reserves depend on the period and priority of the added time consumer.

In the design level analysis, the implementation is only "predicted". The existing design freedom can lead to deviating implementations.

In particular, the following model properties may affect the processor load and lead to deviating task response times:

- Execution times of actual implementation may be different
- Additional functions may be integrated

These properties do not have an impact on the processor load, but impact the data handover and may thus lead to additional (or less) synchronization delays:

- Mapping of functions to runnables may be different
- Mapping of runnables to tasks may be different
- Other implementation details (priorities, offsets, preemptiveness, ...).

Such changes need to be considered by adequate reserves (as identified through sensitivity analysis). Furthermore, the synthesis rules (that are used to derive non-specified implementation details) need to be tailored to the company guidelines.

**Optimization**

The model-based performance analysis allows metric-based optimization of configuration parameters (such as tasks and frame offsets).

In the present example, triggering the ActBrkTorq frames directly (instead of periodic transmission every 10ms) will reduce the worst-case end-to-end latency to 13.46ms (from 23.46ms).
Figure 21: Frame triggered directly by task

When the frame is triggered directly by the update of the data, then the synchronization delay is avoided, reducing the worst-case end-to-end latency of this event chain by 10ms.

4.4 INCHRON Tools

The INCHRON Tool-Suite was used to create a C code based simulation model of the Volvo Brake-by-Wire application in order to perform various timing simulations.

Validation goals

The simulation of the Volvo Brake-by-Wire application was performed with focus on the new TADL concepts as well as the consistency of the proposed methodology. In detail, the following aspects were evaluated:

- Modeling of event chains on implementation level
- Formalized representation and automatic verification of TADL requirements
- Support of probabilistic timing information
- Support of mode / scenario dependent timing information
- Asynchronous time bases

Methodology

The INCHRON approach basically covers all GMP steps at design and implementation level.

Modeling

The INCHRON simulation model comprises 5 ECUs, the central brake controller and 4 wheel controllers, each running an AUTOSAR OS with preemptive scheduling. A convenient way to create the skeleton of the model was by reusing the AUTOSAR configuration information provided by SystemDesk and to some extent by reusing the C code generated with TargetLink.
All ECUs, they are connected to the same 1 Mbit/s high-speed CAN bus, have their own time base (clock). In order to simulate the effects of asynchronicity, all clocks were given different drift factors.

For the simulation the following processes were modeled:

<table>
<thead>
<tr>
<th>Name</th>
<th>SchM</th>
<th>OsTask_5ms</th>
<th>OsTask_10ms</th>
<th>OsTask_20ms</th>
<th>CanRtxtr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Preemptable</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Activation</td>
<td>1ms</td>
<td>5ms</td>
<td>10ms</td>
<td>20ms</td>
<td>Interrupt</td>
</tr>
</tbody>
</table>

The process behavior (function calls, data accesses etc.) was implemented in C using the chronSIM model libraries. Algorithmic details of application functions were mostly replaced by simple stub code with configurable delays (via C macros), e.g. like this:

```c
void Run_WheelBrakeActuatorHandler(void) {
    inchronDelay(ET_Run_WheelBrakeActuatorHandler);
}
```

Hereby, different execution time scenarios (e.g. worst-case, uniform distribution between best-case and worst-case etc.) were evaluated in the simulation. Also for sensibility analysis different scaling factors were applied to the application and the RTE functions.
For the basic software, only a single basic software task \( \text{SchM} \) with varying execution time was modeled.

Furthermore, an event chain was modeled in the C code on function-level (rather than process-level) granularity for the end-to-end analysis of the brake control, altogether covering five measurement steps, that reach from the point in time when a new brake pedal position is acquired to the point in time when a new braking torque is sent to the brake actuator.

**AUTOSAR BSW considerations**

For the simulation, only the communication functions were explicitly modelled in the C code. Other basic software functions have been summarized abstractly in one basic software task with varying execution time.

For all simulation runs the basic software load was assumed to be varying between 30% and 90% (Gaussian distribution with mean 60%).

**Limitations & Assumptions**

In total eight simulation runs with different setups – each covering 20 seconds of target time – were performed:

1. **Synchronization**
   a. ECU clocks are synchronized
   b. ECU clocks are not synchronized (drifting)

2. **Sensitivity analysis (application software)**
   a. Execution times scaled to 100%
   b. Execution times scaled to 133%
   c. Execution times scaled to 166%
   d. Execution times scaled to 200%
At this the following limitations and assumptions were valid:

**Communication architecture**

Due to the lack of support for Ethernet in the current version of the INCHRON Tool-Suite, it was necessary to change Volvo's specification by replacing the Ethernet bus with a CAN bus.

**Execution times**

Basically, the execution times that were used for the simulation are based on the estimations done by AbsInt with aIT. However, in order to increase the dependability of the simulation results and to explore the sensitivity of the system, the execution times of application functions and RTE functions were considerably increased.

**TADL2**

For each signal path from the brake pedal sensor to the four wheel controllers an event chain and end-to-end delay constraints were defined. Additionally, for all tasks a response time constraint equaling the task period was defined.

**Validation results**

The overall results of the different simulation runs are summarized in the following table (there are also detailed HTML reports available):

<table>
<thead>
<tr>
<th>Simulation scenario</th>
<th>Synchronized clocks</th>
<th>Execution time scaling factor</th>
<th>Response time violation</th>
<th>End-to-end violation</th>
<th>End-to-end delay (worst seen case)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes</td>
<td>100%</td>
<td>no</td>
<td>no</td>
<td>10.45 ms</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>133%</td>
<td>no</td>
<td>no</td>
<td>10.91 ms</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>166%</td>
<td>no</td>
<td>no</td>
<td>11.17 ms</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>200%</td>
<td>yes</td>
<td>no</td>
<td>17.78 ms</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>100%</td>
<td>no</td>
<td>no</td>
<td>11.31 ms</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>133%</td>
<td>no</td>
<td>no</td>
<td>11.85 ms</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>166%</td>
<td>no</td>
<td>no</td>
<td>15.35 ms</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>200%</td>
<td>yes</td>
<td>no</td>
<td>25.81 ms</td>
</tr>
</tbody>
</table>

As a general observation it is apparent, that clock synchronization has no major effect on the reliability of the Brake-by-Wire application, as in all cases the even chain finishes in time – far below the required end-to-end delay.

It is also noticeable, that response time violations only occur with scaling factors greater than 166%. At this, one should also note that the execution times for the 100% simulation runs had already been scaled up compared to the original values computed by AbsInt with aIT.
However, due to the different execution rates of the brake pedal sensors, the brake control function, and the brake actuators a typical under-/over-sampling pattern can be observed, where on an average only one out of four sensor values is processed by the brake control.

Finally, the event chain is sensible to toggling effects caused by varying execution times and drifting clocks.
5 Tool chains

This chapter describes in more detail the different tool chains that have been developed and used. To “count” as a tool chain the tools should be able to automatically transfer data among them. That is, tools which require that information is re-entered are not mentioned in this chapter. Since the tools in most cases are already described in previous chapters, the focus on the tool chain descriptions is on the interaction between the tools.

5.1 Implementation tool chain (Simulink-TargetLink-ArcticStudio)

This tool chain describes the “pure” development, i.e. how to produce code and an executable system.

Purpose

The purpose of the tool chain is to provide support for programming and system configuration.

Work flow

The investigated work flow is:

1. Develop BBW application in Simulink
2. Perform simulation to assess behaviour
3. Migrate the Simulink model to a TargetLink model
4. Perform simulation to confirm behaviour
5. Migrate the TargetLink model to AUTOSAR
6. Generate C code for AUTOSAR
7. Package C code for BBW hardware topology
8. Provide AUTOSAR configuration parameters
9. Build executable(s)

Result

Since TargetLink is implemented as a Simulink Toolbox, the interaction between Simulink and TargetLink is quite seamless. As for the C code generation, TargetLink automatically provides the structure and interfaces necessary to make the code fit the AUTOSAR software template.

5.2 Timing analysis tool chain (SystemDesk-TargetLink-aiT-SymTA/S)

Purpose

The motivation behind such a timing analysis tool chain is the seamless timing model/information exchange between the system design and analysis tools. In particular, it should fully support the timing modelling and analysis according to the AUTOSAR timing extensions.
Work flow

The proposed tool chain firstly analyses the system temporal behavioural from worst case execution time perspective. The worst case execution time of a computational task is the maximum time duration it may take to execute on a specific target platform. Its analysis assumes the task execution in a non-pre-emptive mode i.e., non-blocking and un-interruptible.

For a code based system temporal analysis an interconnection between TargetLink (TL) and AbsInt aiT was developed by implementing a toolbox in TargetLink. Figure 22 highlights the tool coupling between TargetLink, AbsInt aiT and SystemDesk.

For this purpose, firstly, a behavioral model of the BBW software components was defined in dSPACE TargetLink tool. In the next step, TargetLink was used to generate the implementation (code files) of the software components along with the data dictionary. The data dictionary contains the description of the variables used in the generated code files. It includes variable ranges, loop bounds, and code patterns applied by TL. Furthermore aiT receives information about lookup table functions, which often determine the execution time of an ECU software. The generated code files – *.c, *.h, were then compiled in TL for a specific target ECU to generate the binary files, i.e., *.elf, *.out. In the next step, the implemented AbsInt toolbox in TargetLink automatically configures the aiT tool to process the generated binaries along with the data dictionary for a static WCET analysis. aiT processes the binary containing the task to be analyzed with the description of the hardware including memories and buses. Along with the WCET analysis, aiT provides several reports and views for in-depth analysis of execution times up to the instruction level. Finally the analyzed WCET results are exported back to the data dictionary, from where they can be imported in SystemDesk together with AUTOSAR software component description and its implementation for further system development.

Once worst-case execution times have been derived and the scheduling policy and timing constraints have been specified,
scheduling analysis can be used to derive the worst-case response times of the involved tasks and runnables. In this tool chain, SymTAS/S is used for this purpose. It allows importing software and operating system models via the AUTOSAR ECU-Configuration format. By combining software models with network configuration, complete end-to-end analyses can be performed. Timing evaluation of the system can then be the basis for optimization steps that improve real-time performance without impact on the processor load, for example optimization of runnable order or task offsets.

Figure 23: AUTOSAR model exchange between dSPACE SystemDesk and SymTA/S

The model-based approach enables an early assessment and optimization of the configuration, as well as guidance throughout the implementation process. Deviations from the original estimates can be identified early, either leading to updated specifications or focused performance improvements.

Figure 23 highlights the information exchange between SystemDesk and SymTA/S. In the first step, a complete system software architecture model that may include the hardware topology, mapping of software components on ECUs, was exported in AUTOSAR format from SystemDesk. In the next step, the exported AUTOSAR file(s) was imported in SymTA/S. SymTA/S then performs local scheduling analysis per processor and both worst-case and distribution analysis for end-to-end event chains.

5.3 XML based tool chain

This tool chain describes the enabling technology implied by having a common XML format which can facilitate model exchange among tools. The tool chain is therefore not fixed but can be continuously extended.

Purpose
The purpose of the tool-chain-enabling technology is to allow smooth exchange of data contained in system models and analysis models. This means easy-to-use export/import of model data to/from a common file format. Within TIMMO-2-USE the file formats ARXML and EAXML have been used. These are XML formats given by the XML schema for the respective meta-model (AUTOSAR and EAST-ADL).

**Work flow**

The investigated work flow is:

1. Prepare AUTOSAR/EAST-ADL model (including TADL2)
2. Export/Save model as ARXML/EAXML
3. Import ARXML/EAXML file into analysis tool
4. Perform analysis

Since AUTOSAR is more commonly used, this workflow is more frequently supported by tools. However, it has been the intention in TIMMO-2-USE to improve this workflow also for EAST-ADL.

**Result**

The different tools investigated in the project supported the work flow as follows:

*Papyrus*: No XML export for EAST-ADL implemented yet.

*VSA*: VSA was used in TIMMO as basis for development of a timing editor for EAST-ADL. It supports export to EAXML.

*Rubus Designer*: This tool was used by Arcticus as a basis for developing an EAST-ADL timing editor. The intention was to support export to EAXML but this functionality has not been fully implemented due to lack of resources.

*TCN Editor*: Within the project, TCN put some efforts into developing an EAST-ADL editor. The editor is able to partially import EAXML files but export is not yet supported.

*ArcticStudio*: Since it is an AUTOSAR tool suite, it is capable of both importing and exporting ARXML.

*SystemDesk*: SystemDesk is capable of both importing and exporting ARXML.

*TCNAnalyzer*: The tool is capable of importing the hardware architecture described in an EAXML file. For implementation details a proprietary XML formal is used.

*SymTA/S*: It would be rather straight-forward to make the tool import EAXML but this has not been done due to lack of resources. The tool can import ARXML and export XML in a proprietary format.

*INCHRON Tool Suite*: The tool is capable of importing ARXML.
6 Summary

This report contains a more elaborate description of the validation efforts performed in the context of the Brake-By-Wire Validator. The aim of this document is to provide more helpful details for anyone trying to perform similar development activities and to benefit from TADL2, TIMMO-2-USE tools and methodology. Therefore the focus is on what has been done and how rather than coverage of TIMMO-2-USE requirements. For the latter, see deliverable D14 [1].
7 References

[1] TIMMO-2-USE Deliverable D14 Validator documentation
http://www.timmo-2-use.org