ulm university universität
uulm

Case Study:
Engine Control Application

Patrick Frey

Ulmer Informatik-Berichte

Nr. 2010-03
April 2010

Case Study: Engine Control Application

Patrick Frey
patrick.frey@etas.com

April 6, 2010

Abstract

This report presents the case study of an engine control application.

The objective of the case study is to demonstrate the applicability of the concepts from our Timing Model for AUTOSAR and the RTE Tracing approach [4] on an integrated, rigorous and close to real-world example. This includes the modeling of the relevant signal paths within the application software of the corresponding AUTOSAR system and their association with application-specific timing requirements. Furthermore, suitable timing properties are determined with the help of an RTE Tracing experiment such that the degree of fulfillment of the timing requirements can be evaluated by means of Timing Oscilloscope Diagrams.

Contents

List of Figures

1 Introduction

In order to demonstrate the applicability of the developed concepts of our Timing Model for AUTOSAR [4] for describing signal, expressing timing requirements and determining timing properties by means of the RTE Tracing approach, a case study has been conducted. The case study, an engine control application, stems from a legacy demonstration project conducted at ETAS (ETAS DemoCar project [8], [9]) where the functionalities of an engine control application have been developed with the help of ETAS tools. In the course of our case study, it has been re-engineered to an AUTOSAR-compliant ECU system for our purposes.

The engine control application contains several functionalities for which timing requirements can be specified. These must be satisfied by an AUTOSAR-compliant realization. The objective of the case study is to describe these timing requirements by means of the concepts of the Timing Model for AUTOSAR and to evaluate their degree of fulfillment by means of an RTE Tracing experiment.

1.1 Outline

The rest of this report is structured as follows:

Section 2 gives a brief introduction into the widely employed working principle of the four-stroke cycle for internal combustion engines. Furthermore, the tasks of an engine control application to control the combustion processes in an engine are described. The consequences from engines being operated at different engine speeds on the determination of important operating parameters are also discussed.

Section 3 then gives a more detailed overview of the basic functionalities of the engine control application under consideration as case study. The relevant input and output signals between the engine control application and its environment are explained. This also includes a description of the most relevant signal paths for the different functionalities as well as the timing requirements which can be associated with the signal paths.

Our case study object is based on a legacy engine control application from a previous demonstration project at ETAS (ETAS DemoCar project [8], [9]). For our purposes, we have reengineered the engine control application to an AUTOSAR-compliant ECU system. Section 4 describes the AUTOSAR compliant ECU software architecture of the reengineered engine control application and the technical realization as an AUTOSAR compliant single ECU system.

Section 5 then describes the application of the concepts of the Timing

Model for AUTOSAR to the AUTOSAR-compliant engine control application. The relevant signal paths of the basic functionalities that have been identified are modeled by means of hierarchical event chains that denote path specifications. The latter are associated with application-specific timing requirements. The objective is then to conduct an RTE Tracing experiment to determine the timing properties and to evaluate the degree of fulfillment of the timing requirements.

The technical setup used for RTE Tracing experiments with the AUTOSAR-compliant engine control application is described in section 6. A hardware-in-the-loop (HIL) setup is designed where the realized AUTOSAR ECU system is coupled to a simulation hardware running a simulation model for the physical processes in an engine, the vehicle it is installed in and a virtual driver who drives the vehicle. From the development of the input and output signals of the engine control application it is shown that the implemented AUTOSAR-compliant engine control application operates as intended. The HiL setup is thus viable for performing RTE Tracing experiments.

Section 7 discusses the results from a conducted RTE Tracing experiment. The determined timing properties of the AUTOSAR-compliant engine control application at a specific engine speed (2000rpm) are presented, and the degree of fulfillment of the timing requirements is discussed with the help of the Timing Oscilloscope Diagrams.

Section 8 provides a summary of the results of this report.

2 Internal Combustion Engines

2.1 Introduction

In internal combustion engines ([10], [13]), chemical energy provided in the form of liquid fuel is transformed into heat and mechanical energy by means of combustion. The combustion is termed internal as it takes place within a combustion chamber. The mechanical energy is produced at the crankshaft of the engine and then transferred via the drivetrain to the wheels where it drives the vehicle.

In the following, the four-stroke cycle which is widely employed in internal combustion engines in automotive vehicles is explained. The engine control application of our case study controls such kind of combustion processes in an eight-cylinder gasoline engine with intake-manifold fuel injection[1]. To

[1]Note that as a large variety of other types of engines exists, with respect to our case study, we restrict our considerations to this specific type of engines where the air/fuel mixture is prepared outside the combustion chamber in the intake manifold.

understand the required functionalities of an engine control application, and thus the purpose of our case study object, it is helpful to understand the basic working principle of such engines.

2.2 The Four-Stroke Cycle

Combustion process in a single cylinder The four stroke cycle is an operating cycle of internal combustion engines such as gasoline engines widely used in automotive vehicles. Figure 1 depicts the four different strokes of a combustion process in a single cylinder.

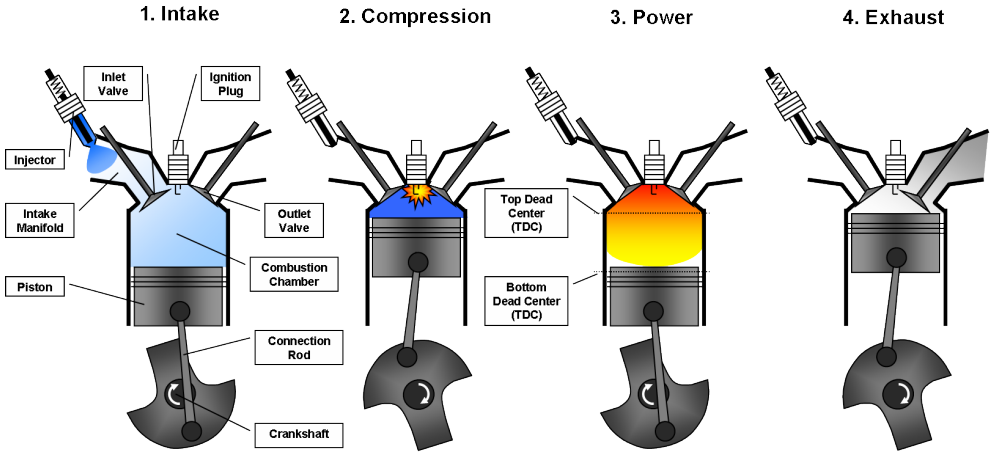


Figure 1: Combustion process in a single cylinder

The four strokes are:

1. **Intake:** Fuel is injected into the intake manifold by an injector where it is mixed with air from the inlet. The air/fuel mixture is then soaked into the combustion chamber through the downwards movement of the piston and the consequent lower pressure in the combustion chamber. When the piston reaches the bottom dead center (BDC) and when the inlet valve is closed, the air/fuel mixture is uniformly distributed in the combustion chamber.

2. **Compression:** The air/fuel mixture in the combustion chamber is compressed until the piston reaches the top dead center (TDC). Before the piston reaches the TDC, an ignition spark must be produced by the ignition plug in order to ignite the air/fuel mixture.

3. **Power (Combustion):** While the air/fuel mixture is burned, the piston is pushed downwards through the expansion of the combustion gases.

This movement is translated to a torque on the crankshaft which drives the drivetrain. In this phase, the chemical energy of the air/fuel mixture is transformed into heat and mechanical energy.

4. **Exhaust:** The outlet valve opens and the exhaust gases stream out of the combustion chamber. Additionally, the piston pushes the exhaust gases out of the combustion chamber during its upwards movement.

In order to continuously deliver mechanical energy, the four-stroke combustion cycle is repeated over and over.

Coordination of combustion processes in multiple cylinders Gasoline engines installed in automotive vehicles in general have multiple cylinders where individual such combustion processes take place. Each combustion process in the cylinders makes a contribution to the overall driving torque. In such multi-cylinder engines, the single combustion processes need to be coordinated according to a specific pattern. This is required to minimize vibrations caused by the inertia forces of the moving masses in the engine. In the eight-cylinder engine for which the engine control application of our case study has originally been developed, for example, the combustion processes are arranged as shown in figure 2.

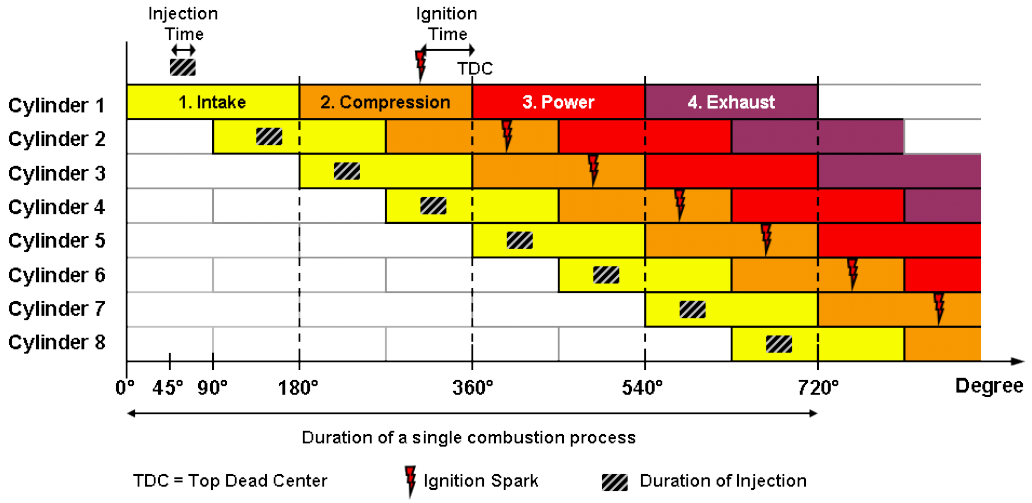


Figure 2: Arrangement of combustion processes in the eight-cylinder engine under consideration

The figure shows that the single combustion processes have a relative offset of 90°. The combustion processes are independent of each other, meaning that no two combustion processes in any two cylinders happen at the same

time. This has the effect that the inertia forces are leveled out such that the engine runs smoothly with a minimum of vibrations.

The figure also shows that the four stroke phases each take 180° measured in terms of an angle relative to the crankshaft, the so-called crankshaft angle (CA). One combustion process takes two crankshaft revolutions, i.e., 720°CA. Fuel is injected in the intake cycle and ignited just before the end of the compression cycle, before the piston reaches the TDC. The power and exhaust strokes then complete the combustion cycle.

2.3 Tasks of an Engine Control Application

In order to keep the engine running, two parameters need to be determined for each combustion process: the injection time and the ignition time. This is the main task of an engine control application. In the following, the basics for the determination of these parameters are explained.

Injection Time Parameter The driving torque delivered by an engine mainly depends on the load of fuel that is combusted in the cylinders. A higher air/fuel load in a combustion chamber leads higher forces through a greater volume expansion of the gas in the cylinder. This consequently leads to a higher driving torque. For an optimal combustion[2], the air/fuel ratio should be close to the theoretical ideal ratio of 14.7:1 (stoichiometric mixture). I.e., in order to optimally combust 1kg of fuel, 14.7 kg air are required. In an air-flow controlled engine as considered in our case study, the fuel mass to be injected is determined based on the current air flow such that the optimal air/fuel ratio is maintained. This is the basis for an optimal combustion and efficient usage of the chemical energy provided by the fuel. The mass of fuel that is injected into a combustion chamber by an injector is proportional to the time that the injector opens, thus the term injection time. The current mass air flow on which the fuel mass depends is measured by a mass air-flow sensor which is installed in the intake system right after the throttle (see figure 3). The mass air flow sensor measures the current air-flow in kg/h. From that, the fuel mass per stroke can be determined. The air-flow is dictated by the throttle which is governed by the accelerator pedal. The latter is naturally under the control of the driver who, by this mechanism, can request a specific driving torque in order to accelerate the vehicle.

Ignition Time Parameter After the production of an ignition spark, the combustion of the air/fuel mixture in a cylinder takes approximately 2ms. The ignition time must be determined in such a way that the maximum

[2]optimal in this context means that no fuel is left unburnt and that the exhaust gases contain as few polluting substances as possible

pressure from the combustion is achieved shortly after the top-dead center. Thus, the ignition angle needs to be adjusted to earlier points in time at higher engine speeds, i.e., longer before the piston reaches the top dead center ([10], [13]). The ignition time is in general determined as an angle that is relative to the crankshaft. It determines the point in time when the ignition spark must be produced before the cylinder reaches the TDC. The ignition time depends on the speed of the engine and the load of the air/fuel mixture for the combustion process. As the air/fuel mixture depends on the mass air flow sensed by the mass air flow sensor, the ignition time also indirectly depends upon the latter.

Figure 3 depicts a schematic overview of an engine with the most important sensors and actuators for the basic operation of the engine through an engine control application.

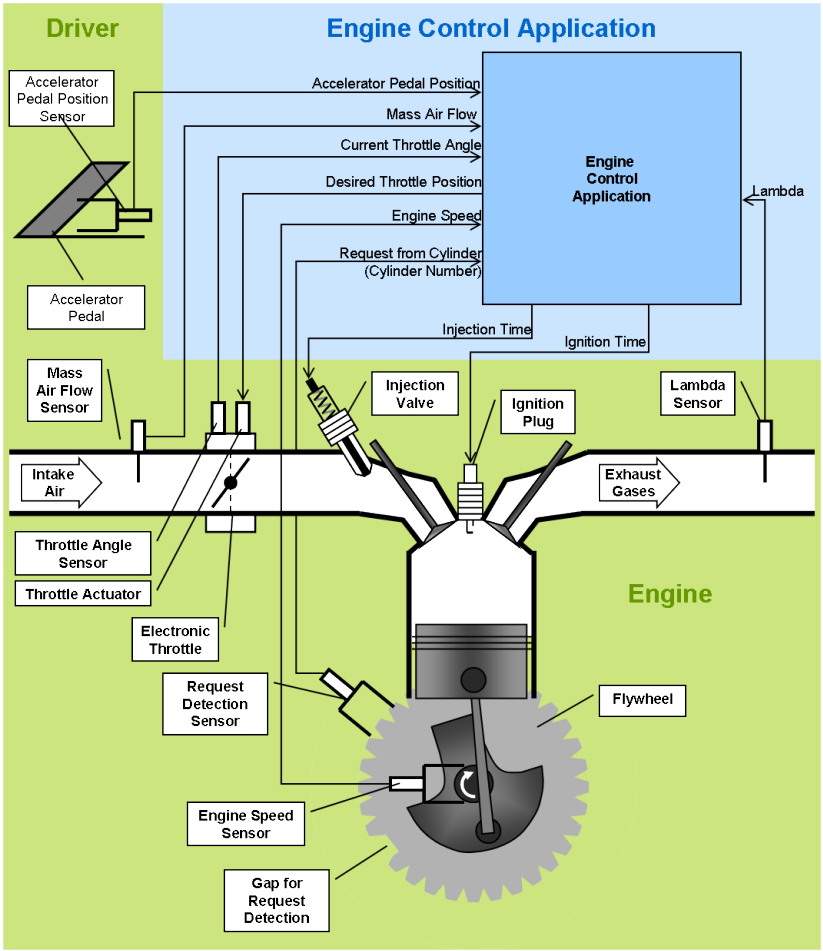


Figure 3: Schematic overview of an engine with engine control application

Consequences from combustion process in one cylinder To operate the engine at different engine speeds, the injection time and ignition time parameters need to be provided in synchrony with the speed of the engine. To achieve this synchronization, the injection time and ignition time parameters are requested on a cylinder-specific basis from the engine control application. This translates into engine speed dependent requests for the injection time and ignition time parameters for the combustion process.

Figure 4 depicts the cylinder-specific requests from a single cylinder at different engine speeds.

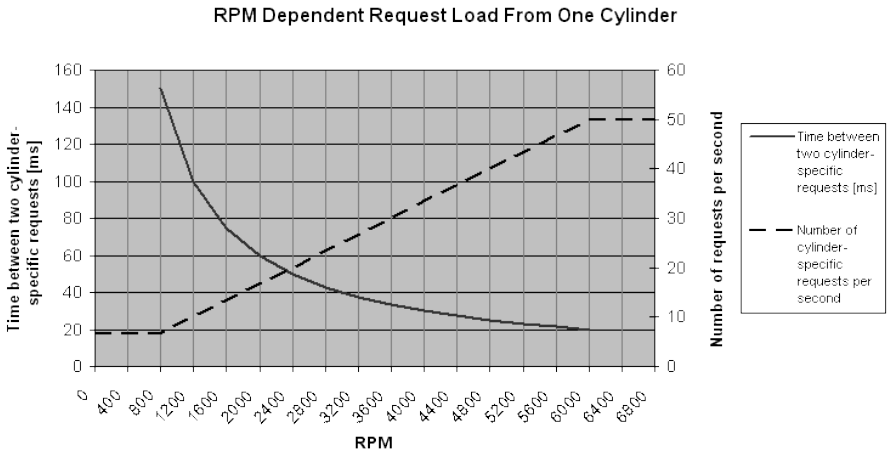


Figure 4: Number of cylinder-specific requests from a single cylinder at different engine speeds

The considered engine speed range between 800 rpm and 6000 rpm is the range where the engine is operated after being started. The lower range value is the engine speed where the engine is idling to prevent it from stalling. The upper range value is a defined limit to prevent the engine from mechanical damages due to uncontrollable combustion processes ("knocking") at higher engine speeds. As can be seen from the figure, the number of cylinder-specific requests is linear dependent on the speed of the engine. The time between two cylinder-specific requests also depends on the speed of the engine and decreases with increasing speed of the engine. At the highest speed of the engine, i.e., at 6000 rpm, only 20ms pass between two consecutive combustion processes in an individual cylinder. At the lowest speed of the engine, i.e., at 800rpm, 150ms pass between two consecutive combustion processes in an individual cylinder. Between two such cylinder-specific combustion processes, the ignition time and injection time values need to be determined based on

the latest inputs from the sensors (i.e., the mass air flow sensor).

From the above considerations, timing requirements for the correct operation of a single combustion process can be derived:

- Between each two cylinder-specific combustion processes, the injection time and ignition time parameters need to be determined based on the current state of the engine (inputs from sensors)
- The most stringent requirements on the calculation of new values for the injection time and ignition time parameters apply at the highest speed of the engine. In our case study, this is at 6000rpm.
- At the highest speed of the engine, the injection time and ignition time parameters need to be determined at least every 20ms based on newly acquired inputs from the respective sensors.

By satisfying the timing requirements towards the determination of the injection time and ignition time parameters at the highest speed of the engine it is guaranteed that the less stringent timing requirements that apply at lower engine speeds are also satisfied.

Consequences from combustion processes in multiple cylinders Figure 5 shows a similar diagram as shown in figure 4, this time, the number of cylinder-specific requests from eight cylinders are considered (as in our case study). In the considered engine, the combustion processes of the single cylinders are arranged according to the coordination pattern as shown in figure 2.

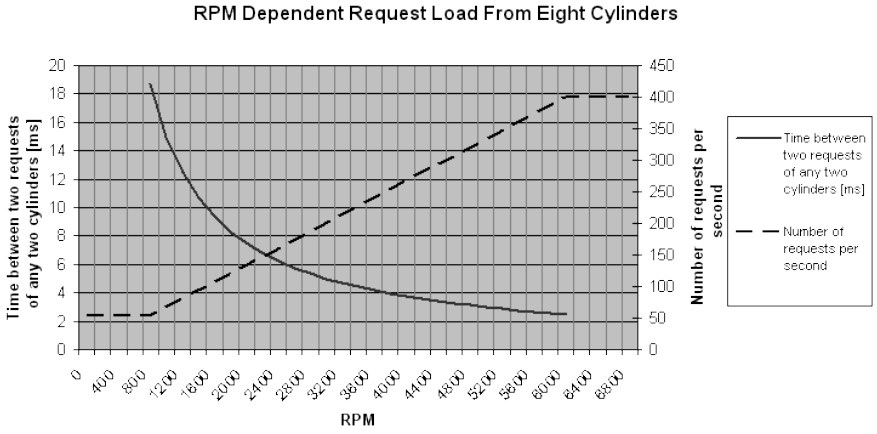


Figure 5: Number of cylinder-specific requests from eight cylinders at different engine speeds

Due to the coordination pattern of the combustion processes with their 90°CA offset to each other, injection time and ignition time parameters are requested in short intervals from the engine control application. When the engine is continuously operated, every 90°CA a different cylinder requests its parameters. At the highest speed of the engine, i.e. at 6000rpm, every $\frac{20ms}{8} = 2.5ms$ a new request is made from one of the cylinders. The engine control application must be capable to process these requests and to deliver the requested injection time and ignition time parameters to the respective actuators.

From the latter considerations, requirements for a complete engine control application for an eight cylinder engine can be derived. In principle, it must be capable to determine the parameters for the individual combustion processes that are independent of each other at any speed of the engine.

2.4 Summary and Conclusion

In this section, the working principle of the four-stroke cycle that is widely employed in gasoline engines that power automotive vehicles has been explained. The main task of an engine control application is to determine the parameters to operate the combustion processes in the cylinders of an engine: the injection time and the ignition time. The injection time determines the amount of fuel that is injected into the intake manifold to be combusted in a cylinder. It thus decides upon the driving torque that is delivered by the engine. The ignition time determines the point in time when the air/fuel mixture is ignited. It is important for a complete and optimal combustion in order to achieve a maximum conversion of the chemical energy to mechanical energy. To influence the amount of fuel that is injected for a combustion process, the amount of air that is available for the combustion is regulated. This is performed by means of a throttle that is installed in the intake system. It is governed by the accelerator pedal which is under the control of the driver. In order to deliver the driving torque requested by the driver in different driving situations, gasoline engines powering automotive vehicles are operated at different engine speeds. The most stringent timing requirements apply at the highest speed of the engine. Due to this fact, certain requirements towards the timely delivery of the parameters arise. These must be satisfied by an engine control application. For the combustion process in a single cylinder it is important that the injection time and ignition time parameters are up-to-date and provided in time. This translates into certain timing requirements on the functionality of an engine control application. For the overall operation of the engine, it is important that the parameters for the combustion processes in all cylinders are determined and provided in

time according to the coordination scheme of the combustion processes employed by the engine. This translates into the requirement that the engine control application must be capable to adequately operate the combustion processes in all cylinders.

In the following section, the structure of the basic functionalities of the engine control application under consideration, the signal paths they contain and the timing requirements that can be formulated towards these are outlined.

3 Engine Control Application: Basic Functionalities, Signal Paths and Timing Requirements

3.1 Introduction

The objective of the engine control application under consideration is to control the combustion processes in the single cylinders of an air-flow controlled, intake manifold fuel injected eight-cylinder engine in order to deliver a desired driving torque according to the drivers wish. To control the combustion processes, a multitude of functions are required which need to be implemented in software. For our case study, we focus on the basic functionalities. These are the control of the air-flow via the throttle, the calculation of injection times and the ignition times, and the timely delivery of the latter upon cylinder-specific requests.

Section 3.2 gives a coarse grain overview of the functionalities of our engine control application. This is followed by a more detailed description of the individual basic functionalities and the elementary functions they are composed of in section 3.3. Furthermore, the chains of cause-and-effect and related signal paths for the basic functionalities under consideration are described as well as the timing requirements that must be satisfied and which are associated with the signal paths. When mechanical parts are replaced by electronics and software in automotive applications, specific redundancy concepts need to be applied to guarantee the safety of the functionality. This is also the case for our engine control application: the accelerator pedal and the throttle are coupled electronically to the electronic control unit with the engine control application. In our case study, this leads to the introduction of redundant accelerator pedal sensors and redundant throttle sensors. This, however, introduces additional functions as input signals must be acquired redundantly and merged before any calculations based on these signals can be

made. This also introduces additional timing requirements on the synchronized processing of certain signals. These issues are also discussed further in section 3.3. Section 3.4 provides a short summary of the basic functionalities of our engine control application, the signal paths and the timing requirements.

3.2 Overview of the Basic Functionalities

Figure 6 depicts an overview the engine control application, the physical processes it is embedded in and the signals exchanged between the engine control application and the physical processes.

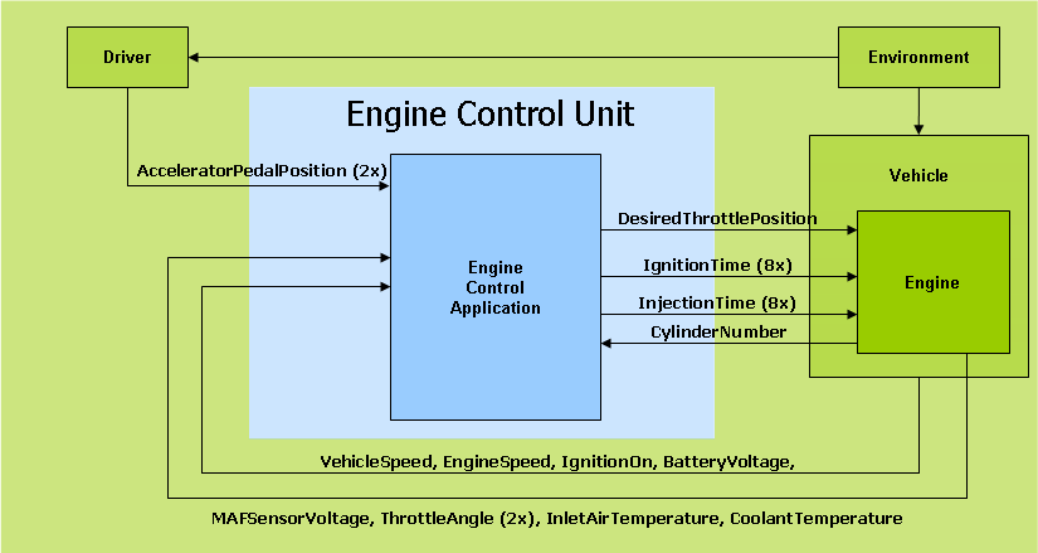


Figure 6: Overview of the engine control application, its environment and the exchanged signals

The engine control application reads input values from various sensors (i.e., mass air flow, current throttle angle[3]) and the set-point device that is under the control of the driver (i.e., accelerator pedal position). Based on those inputs, appropriate outputs for the actuators which influence the combustion processes (i.e., fuel injectors and ignition plugs) are calculated. Furthermore, the throttle position is controlled which dictates the air flow

[3]Note that also other input values from other sensors are read by the engine control application, e.g., the engine speed, vehicle speed, coolant and inlet air temperatures, battery voltage, lambda value etc. these, however, are not directly relevant for the signal paths considered in our case study.

in the intake. The latter influences the air/fuel ratio in the combustion processes and thus the driving torque.

Figure 7 shows an overview of the internal structure and basic functionalities of the engine control application.

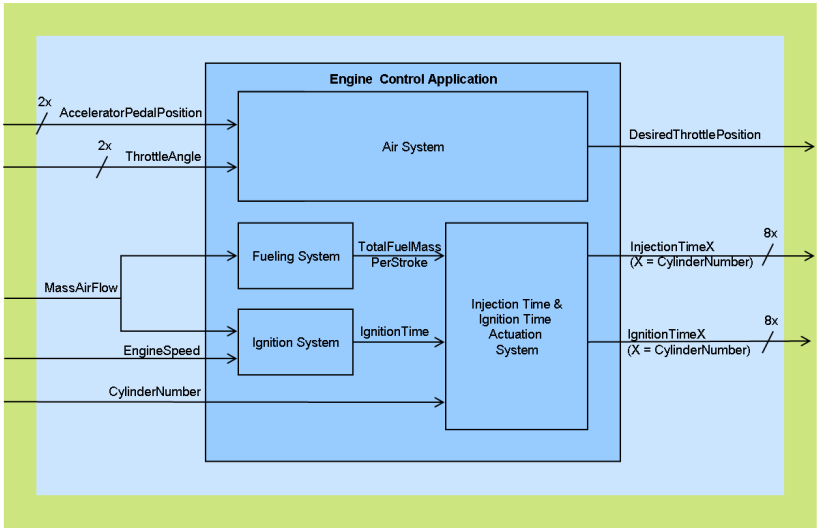


Figure 7: Overview of the internal structure and basic functionalities

The overall functionality of the engine control application is divided into four subsystems:

Air system The air system calculates a new desired throttle position based on the current throttle position and the accelerator pedal position. This influences the air flow and subsequently the injected fuel mass and ignition timing for a combustion process.

Fueling system The fueling system calculates the fuel mass to be injected in the intakte manifold for a combustion process in a cylinder. This is based on the current mass air flow sensed in the intake system.

Ignition system The ignition system calculates the ignition angle that determines the point in time when an ignition spark is produced to ignite the air/fuel mixture in the combustion chamber. This is based on the mass air flow and the current engine speed.

Injection time and ignition time actuation system The injection time and ignition time actuation system delivers the latest values of the two parameters upon a cylinder specific request. The calculation of the two parameters is decoupled from the actuation as the parameters

are pre-calculated for all cylinders (sequential injection). This allows a fast response to a cylinder-specific request.

In our case study, specific functionalities that were traditionally realized as purely mechanical systems are realized as mechatronical systems, i.e., as mechanical system additionally comprising electronics and software. The legislative demands of the E-Gas concept [1] require the introduction of certain redundancy concepts due to safety considerations when mechanical components are replaced by electronics-based solutions. Such concepts include to redundantly acquire inputs by multiple sensors and to compute a voted signal for further processing from the redundantly acquired sensor values. In figure 7, the input signals for the accelerator pedal position and the throttle position are thus duplicated.

In the following, the basic functionalities of the engine control application are described in more detail.

3.3 Detailed Description of Basic Functionalities, Signal Paths and Timing Requirements

In the following, the individual functionalities of the engine control application under consideration are described in more detail. This includes descriptions of

- the structure of the functionalities by means of the elementary functions they are composed of,
- the important signal paths, and
- the timing requirements that are specified towards the functionalities and which can be associated with their signal paths.

3.3.1 Air system

Overview

The air system calculates a new desired throttle position based on the current throttle position and the accelerator pedal position. This influences the air flow and subsequently the injected fuel mass (and also the ignition timing) for a combustion process.

The throttle installed in the engine is a dynamic system that needs to be controlled. Together with the throttle installed in the engine, the air system thus forms a closed-loop control application.

The air system is subdivided into several elementary functions that

- capture input values from the accelerator pedal and throttle sensors (functions AcceleratorPedalSensor and ThrottleSensor),
- analyze redundantly captured input values from the sensors and compute a merged input signal (functions AcceleratorPedalVoter and ThrottleSensor)
- analyze the merged accelerator pedal position input signal and compute a desired throttle position set-point signal (function PedalFeel)
- compute a control signal for the adjustment of the current throttle position based on the desired throttle position (function ThrottleController)
- bring the computed desired throttle position into effect (function ThrottleActuator)

Figure 8 depicts an overview of the air system.

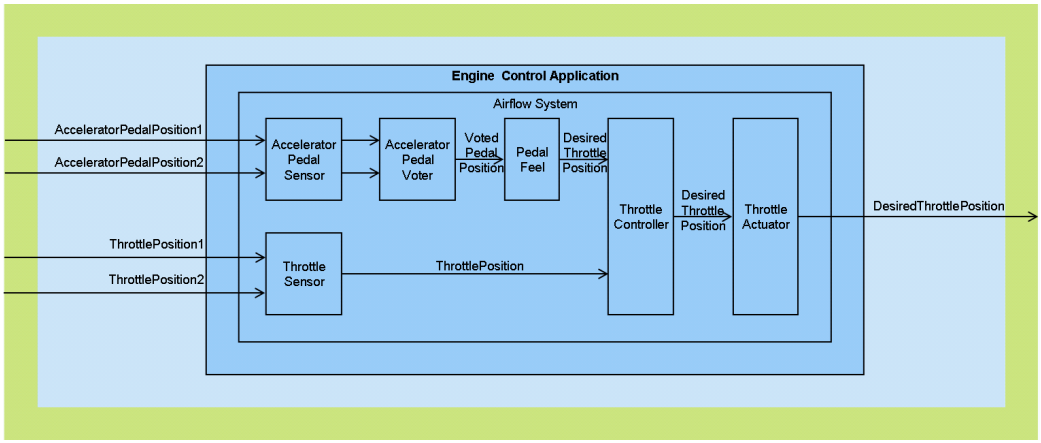


Figure 8: Overview of the air system

Signal Paths

The air system contains several signal paths between its input signals and output signals. In principle, two conceptual signal paths can be identified:

- The first conceptual signal path is from the accelerator pedal position to the desired throttle position. This signal path describes the chain of cause-and-effect on how a change of the accelerator pedal position influences the new throttle position.
- The second conceptual signal path is from the current throttle position to the desired throttle position. This signal path describes the chain of

cause-and-effect that corresponds to the feedback path of the throttle control application.

Due to the required redundancy concepts, the accelerator pedal position and the current throttle position are sensed by duplicate sensors and delivered as separate signals. There are thus four concrete signal paths where each two signal paths correspond to one conceptual signal path.

Figure 9 depicts an overview of the air system including identified signal paths.

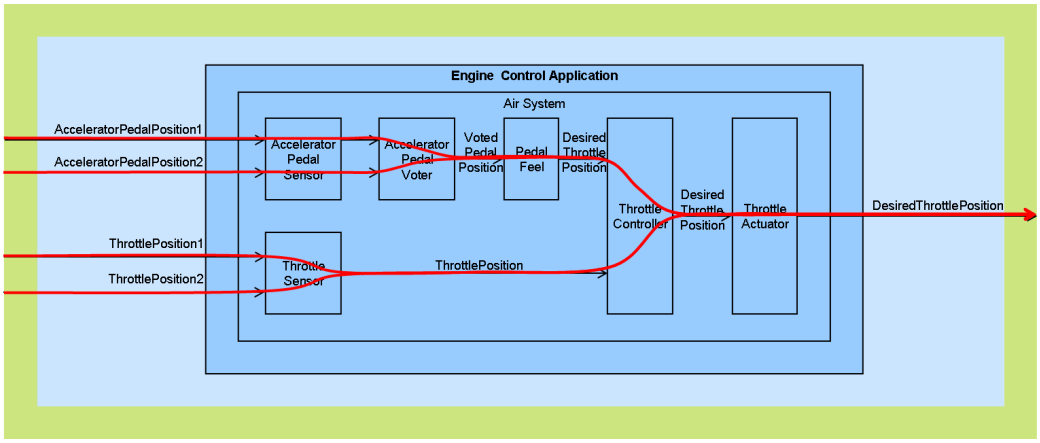


Figure 9: Overview of the air system, including identified signal paths

In the following, the timing requirements that can be associated with the signal paths are described.

Timing Requirements

Each of the identified signal paths of the air system is associated with three different timing requirements. These originate from the fact that the throttle control application in the air system is a continuous synchronous real-time application. The timing requirements can be formulated as follows:

1. The first timing requirement is a timing requirement on the latency of the signal transformation along the feedback path of the throttle control application. A minimization of the path delay τ is required such that these can be considered as being negligible. For this, the path delay must be less than or equal to 1ms. This is expressed by

$$\tau_{\mathsf{ThrottlePosition1} \rightarrow \mathsf{DesiredThrottlePosition}} \leq 1ms$$

and

$$\tau_{\mathsf{ThrottlePosition2} \rightarrow \mathsf{DesiredThrottlePosition}} \leq 1ms$$

2.+3. The second and third timing requirement are timing requirements on the latency between consecutive effective sampling and actuation actions. The throttle control application requires the maintenance of a nominal effective sampling interval of 10ms. A deviation of 1ms is acceptable (acceptable effective sampling jitter). The same also holds for the effective actuation interval (acceptable effective actuation jitter). These timing requirements are expressed by

$$\mathsf{h}^{\mathsf{sampling}}_{\mathsf{ThrottlePosition1} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

and

$$\mathsf{h}^{\mathsf{sampling}}_{\mathsf{ThrottlePosition2} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

for the nominal sampling intervals, and

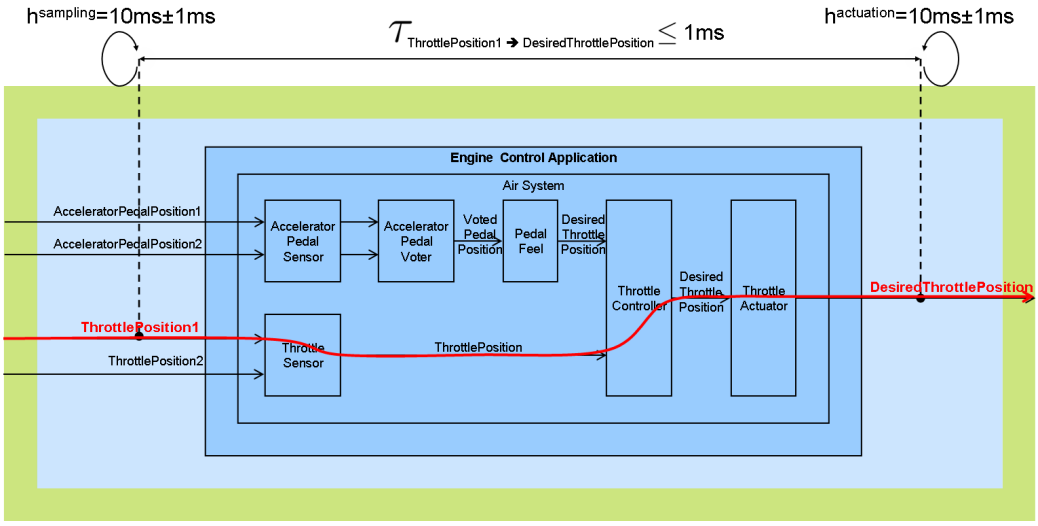
$$\mathsf{h}^{\mathsf{actuation}}_{\mathsf{ThrottlePosition1} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

and

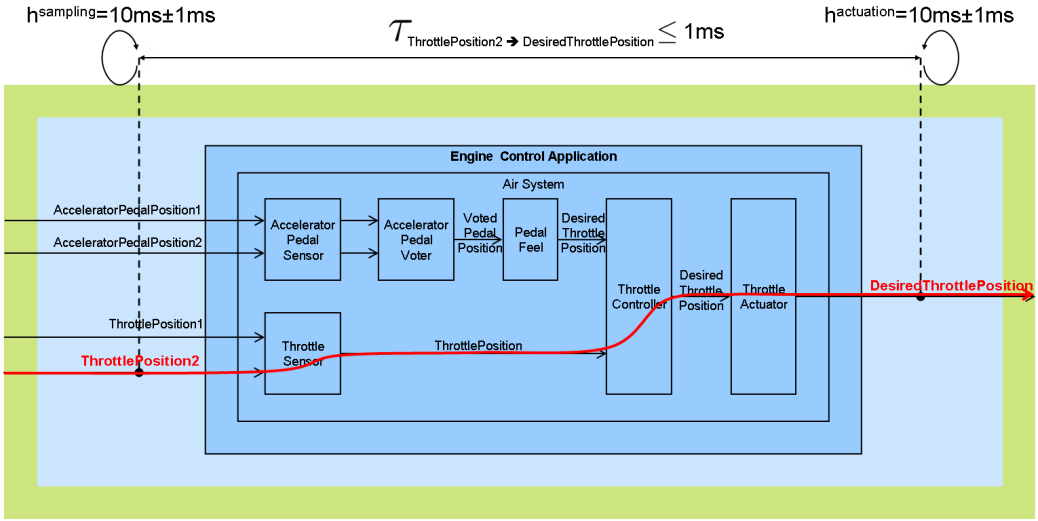
$$\mathsf{h}^{\mathsf{actuation}}_{\mathsf{ThrottlePosition2} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

for the nominal effective actuation intervals.

Figures 10(a) and 10(b) show the signal paths from the input signals ThrottlePosition1 and ThrottlePosition2, respectively, to the output signal DesiredThrottlePosition and the associated timing requirements.



(a) Signal path from input signal ThrottlePosition1 to output signal DesiredThrottlePosition and associated timing requirements



(b) Signal path from input signal ThrottlePosition2 to output signal DesiredThrottlePosition and associated timing requirements

Figure 10: Signal paths from input signals ThrottlePosition1/2 to output signal DesiredThrottlePosition and associated timing requirements

In the air system, the set-point signal for the throttle controller is computed based on the accelerator pedal position. Three functions are involved in this process. For the signal paths between the two accelerator pedal position signals and the desired throttle position signal, similar timing requirements can be formulated as for the feedback path:

1. The first timing requirement is a timing requirement on the latency of the signal processing along the path from one of the accelerator pedal position signals to the desired throttle position. As for the feedback path of the throttle control application, the minimization of the path delay τ is desirable. For this, the delay must be less than or equal to 1ms. This is expressed by

$$\tau_{\mathsf{AcceleratorPedalPosition1} \rightarrow \mathsf{DesiredThrottlePosition}} \leq 1ms$$

and

$$\tau_{\mathsf{AcceleratorPedalPosition2} \rightarrow \mathsf{DesiredThrottlePosition}} \leq 1ms$$

2.+3. The second and third timing requirement are timing requirements on the latency between consecutive effective sampling and actuation actions. The throttle control application requires the maintenance of a nominal effective sampling interval of 10ms. A deviation of 1ms is acceptable. The same also holds for the effective actuation interval. These timing requirements are expressed by

$$\mathsf{h}^{\mathsf{sampling}}_{\mathsf{AcceleratorPedalPosition1} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

and

$$\mathsf{h}^{\mathsf{sampling}}_{\mathsf{AcceleratorPedalPosition2} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

for the nominal sampling intervals, and

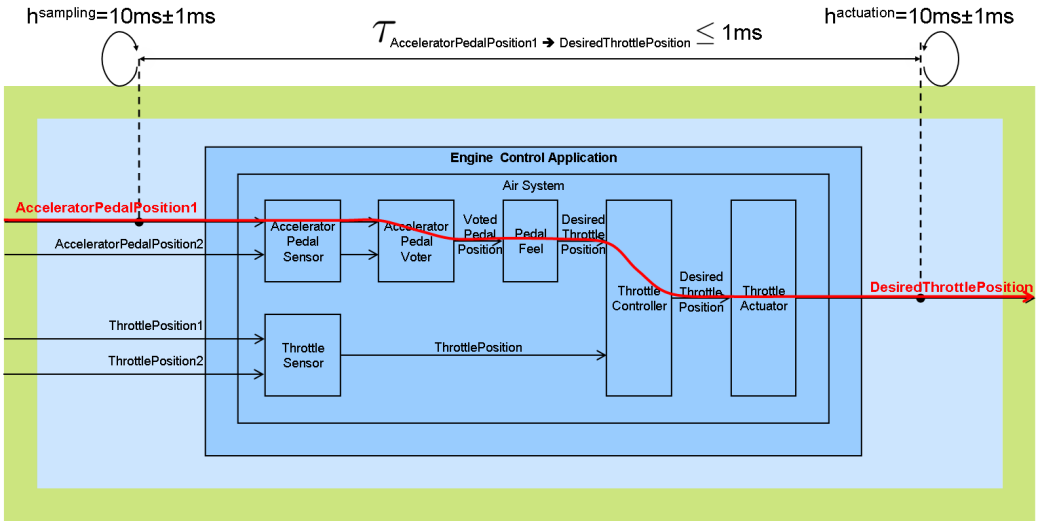
$$\mathsf{h}^{\mathsf{actuation}}_{\mathsf{AcceleratorPedalPosition1} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

and

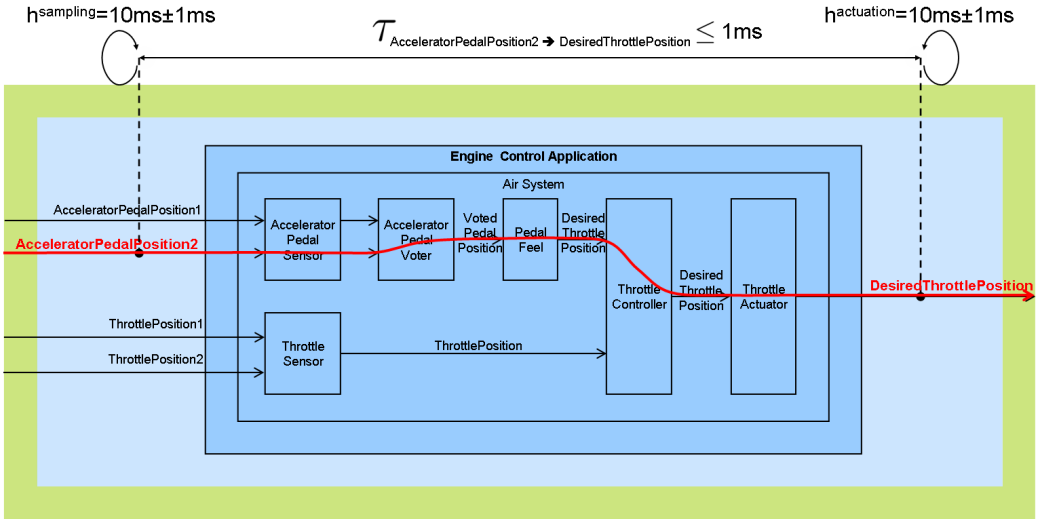
$$\mathsf{h}^{\mathsf{actuation}}_{\mathsf{AcceleratorPedalPosition2} \rightarrow \mathsf{DesiredThrottlePosition}} = 10ms \pm 1ms$$

for the nominal effective actuation intervals.

Figures 11(a) and 11(b) show the signal paths from the input signals AcceleratorPedalPosition1 and AcceleratorPedalPosition2, respectively, to the output signal DesiredThrottlePosition.



(a) Signal path from AcceleratorPedalPosition1 to DesiredThrottlePosition and associated timing requirements



(b) Signal path from AcceleratorPedalPosition2 to DesiredThrottlePosition and associated timing requirements

Figure 11: Signal paths from AcceleratorPedalPosition1/2 to DesiredThrottlePosition and associated timing requirements

Furthermore, due to the employed redundancy concept, timing requirements on the synchronization of related input signals can be formulated.

The accelerator pedal position is provided as two signals from the two different accelerator pedal sensors. After a conversion of the voltage values delivered by the sensors (function AcceleratorPedalSensor), a voted accelerator pedal position signal is determined (function AcceleratorPedalVoter). In order to produce such a voted accelerator pedal position signal, it is required that the values of the original voltage signals are temporally consistent. In other words, the two input signals must be synchronized within an interval of 1ms with respect to the voted accelerator pedal position signal. This is expressed by

$$\mathsf{d}_{\mathsf{AcceleratorPedalPosition1} \rightarrow \mathsf{VotedPedalPosition}} \leq 1ms \tag{1}$$

and

$$\mathsf{d}_{\mathsf{AcceleratorPedalPosition2} \rightarrow \mathsf{VotedPedalPosition}} \leq 1ms \tag{2}$$

Figure 12 depicts the relevant segments of the signal paths from the input signals AcceleratorPedalPosition1 and AcceleratorPedalPosition2 to the common intermediate signal VotedPedalPosition where both signal paths join. Furthermore, the timing requirement that is associated with the two path segments is shown.

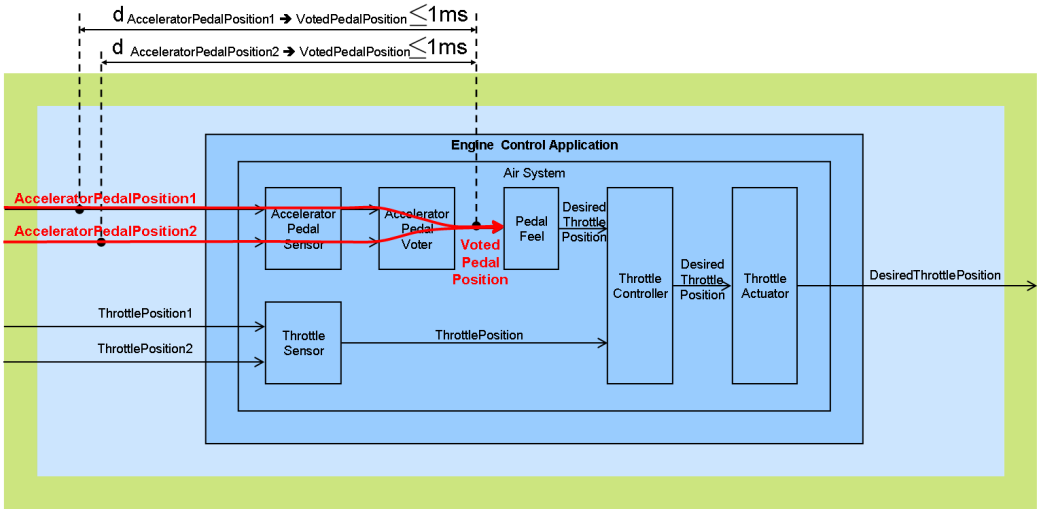


Figure 12: Signal path segments from input signals AcceleratorPedalPosition1 and AcceleratorPedalPosition2 to common intermediate signal VotedPedalPosition and associated timing requirements

Similar to the accelerator pedal position, the throttle position is also provided as two signals from the two redundant throttle sensors. The conversion of the voltage values delivered by the sensors to a percentage value and the determination of a voted throttle position signal is performed by a single function (function ThrottleSensor). Again, in order to produce such a voted signal, the two input signals must be synchronized within an interval of 1ms with respect to the voted signal. This is expressed by

$$\mathsf{d}_{\mathsf{ThrottlePosition1} \rightarrow \mathsf{ThrottlePosition}} \leq 1ms \tag{3}$$

and

$$\mathsf{d}_{\mathsf{ThrottlePosition2} \rightarrow \mathsf{ThrottlePosition}} \leq 1ms \tag{4}$$

Figure 13 depicts the relevant segments of the signal paths from input signals ThrottlePosition1 and ThrottlePosition2 to the common intermediate signal ThrottlePosition where both signal paths join. The timing requirement that is associated with these is also shown.

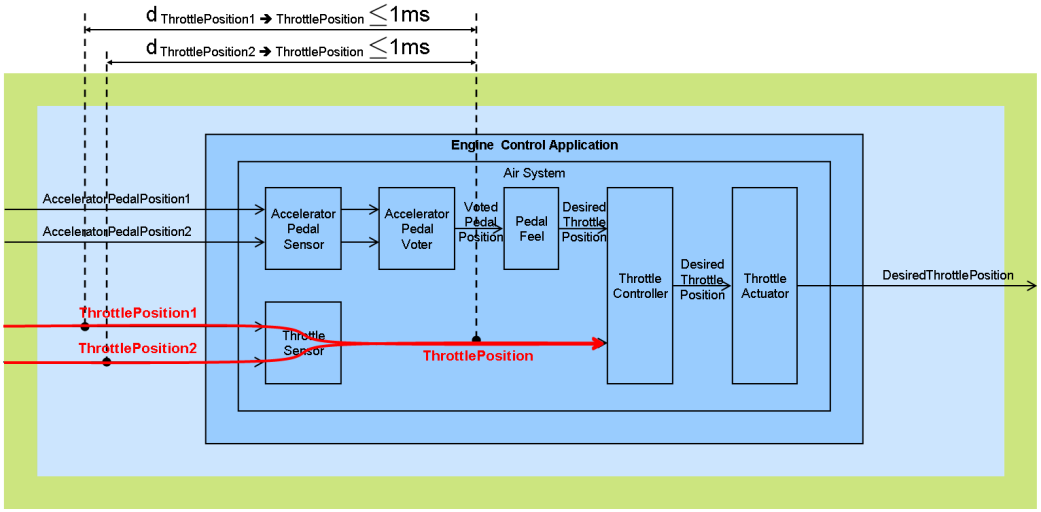


Figure 13: Signal paths segments from ThrottlePosition1 and ThrottlePosition2 to ThrottlePosition and associated timing requirements

3.3.2 Fueling System

Overview

The fueling system calculates the fuel mass to be injected in the intakte manifold for a combustion process in a cylinder. The total fuel mass per stroke is determined based on the current mass air flow in the intake system.

The fueling system is subdivided into several elementary functions that

- capture input values from the mass air flow sensor (function MassAirFlowSensor),
- determine the mass air flow per stroke (function AirMassFlow)
- determine the base fuel mass per stroke based on the mass air flow per stroke (module BaseFuelMass)
- determine adjustments of the fuel mass due to specific wall wetting effects in the intake system of the engine (function TransientFuelingCompensation)
- determine the total fuel mass per stroke based on the transient fuel mass per stroke (function TotalFuelMassPerStroke).

The determined total fuel mass per stroke applies for the combustion processes in all cylinders (i.e., it is not determined on a cylinder-specific basis in our engine control application). The actuation of the calculated total fuel mass per stroke is performed by the injection time and ignition time actuation system upon cylinder-specific requests.

Figure 14 depicts an overview of the fueling system.

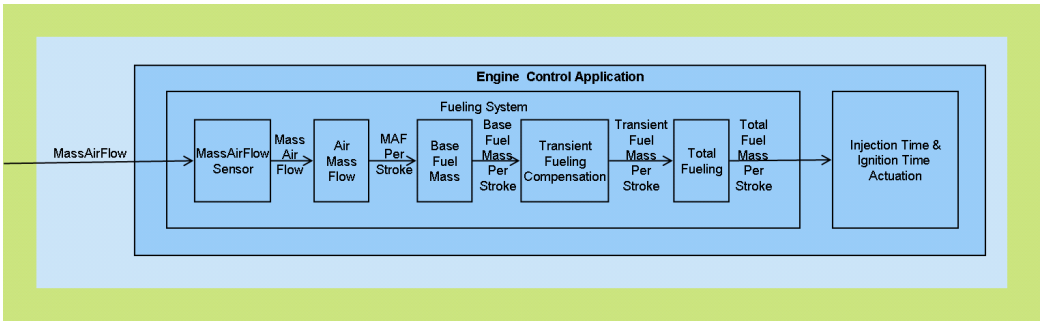


Figure 14: Overview of the fueling system

In the following, the signal path and the associated timing requirements are described.

Signal Paths and Timing Requirements

Figure 15 depicts the relevant signal path that can be identified in the fueling system for the calculation of the total fuel mass per stroke based on the mass air flow.

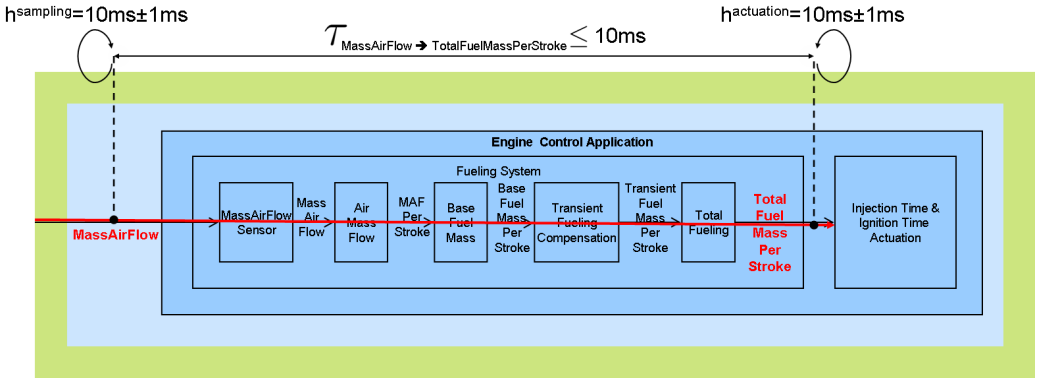


Figure 15: Signal path from input signal MassAirFlow to intermediate signal TotalFuelMassPerStroke and associated timing requirements

The signal path is associated with timing requirements that are derived from the operation of the engine at its highest speed, i.e., at 6000rpm. As described in section 2.3, the time between two consecutive combustion processes in a single cylinder is 20ms at the highest speed of the engine. Between each such two combustion processes, the injection time and ignition time parameters need to be determined for the next combustion process.

To achieve the latter in all cases, the following timing requirements are formulated:

1. The calculation of a new value for the total fuel mass per stroke (basis for injection time) shall take at maximum 10ms. This is expressed by

$$\tau_{\mathsf{MassAirFlow} \rightarrow \mathsf{TotalFuelMassPerStroke}} \leq 10ms$$

2.+3. A new value for the total fuel mass per stroke shall be provided every 10ms. This translates into an effective actuation interval of 10ms. A deviation of 1ms is acceptable. This also requires that the current mass air flow is sampled every 10ms. Here, a deviation of 1ms is also acceptable. These timing requirements are expressed by

$$h^{\mathsf{sampling}}_{\mathsf{MassAirFlow} \rightarrow \mathsf{TotalFuelMassPerStroke}} = 10ms \pm 1ms$$

for the nominal sampling interval, and

$$h^{\mathsf{actuation}}_{\mathsf{MassAirFlow} \rightarrow \mathsf{TotalFuelMassPerStroke}} = 10ms \pm 1ms$$

for the nominal effective actuation interval.

Note that the timing requirements are more strict than what would be required: effective sampling and actuation intervals of 20ms would be sufficient. The more strict timing requirements, however, guarantee that an up-to-date value for the injection time actuation value is always available.

3.3.3 Ignition System

Overview

The ignition system calculates the ignition angle that determines the point in time when an ignition spark is produced to ignite the air/fuel mixture in the combustion chamber. This is based on the mass air flow and the current speed of the engine.

The ignition system is subdivided into several elementary functions that capture input values from the mass air flow sensor (function MassAirFlowSensor), determine the mass air flow rate (function AirMassFlow), and determine the ignition time based on the mass air flow rate and the engine speed (function IgnitionTiming).

The determined ignition time applies for the combustion processes in all cylinders (i.e., as the injection time, it is not determined on a cylinder-specific basis in our engine control application). The actuation of the calculated ignition time is performed by the injection time and ignition time actuation system upon cylinder-specific requests.

Figure 16 depicts an overview of the ignition system.

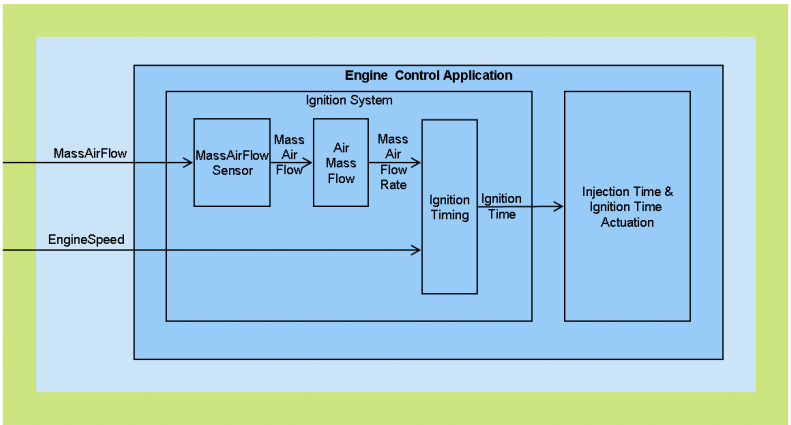


Figure 16: Overview of the ignition system

Signal Paths and Timing Requirements

Figure 17 depicts the signal path that can be identified in the ignition system for the calculation of the ignition time based on the mass air flow.

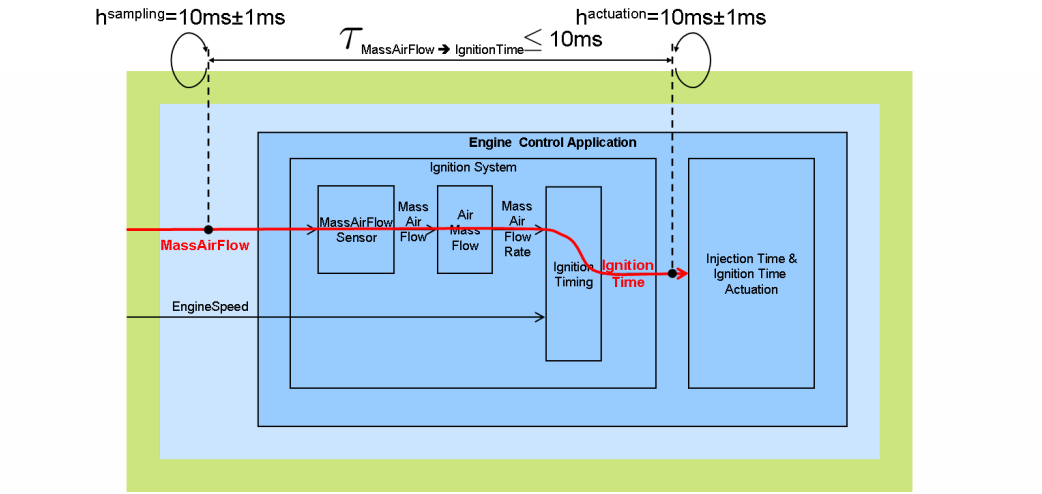


Figure 17: Signal path from input signal MassAirFlow to intermediate signal IgnitionTime and associated timing requirements

As with the signal path in the fueling system, the signal path in the ignition system is associated with timing requirements that are derived from the operation of the engine at its highest speed. The timing requirements are the same as for the fueling system, however, they refer to a different signal path:

1. The calculation of a new value for the ignition time shall take at maximum 10ms. This is expressed by

$$\tau_{\mathsf{MassAirFlow} \rightarrow \mathsf{IgnitionTime}} \leq 10ms$$

2.+3. A new value for the ignition time shall be provided every 10ms. This translates into an effective actuation interval of 10ms. A deviation of 1ms is acceptable. This also requires that the mass air flow is sampled every 10ms. Here, a deviation of 1ms is also acceptable. These timing requirements are expressed by

$$h^{\mathsf{sampling}}_{\mathsf{MassAirFlow} \rightarrow \mathsf{IgnitionTime}} = 10ms \pm 1ms$$

for the nominal sampling interval, and

$$h^{\mathsf{actuation}}_{\mathsf{MassAirFlow} \rightarrow \mathsf{IgnitionTime}} = 10ms \pm 1ms$$

for the nominal effective actuation interval.

3.3.4 Injection Time and Ignition Time Actuation System

Overview

The injection time and ignition time actuation system delivers the latest values of the two parameters to the respective actuators upon a cylinder specific request. The calculation of the two parameters is decoupled from the actuation as the parameters are pre-calculated for all cylinders (sequential injection).

The injection time and ignition time actuation system is subdivided into two elementary functions that

- each evaluate the cylinder number for which the injection time or ignition time parameter is requested,
- update the output value for the respective actuator with the latest value that has been determined for the injection time (function InjectionTimeActuation),
- update the output value for the respective actuator with the latest value that has been determined for the ignition time (function IgnitionTimeActuation).

Figure 18 depicts an overview of the injection time and ignition time actuation system.

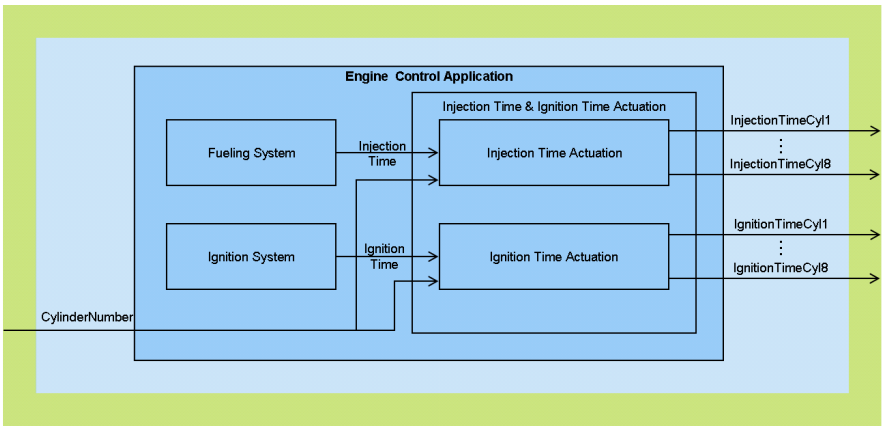


Figure 18: Overview of the injection time and ignition time actuation system

Signal Paths and Timing Requirements

Figure 19 depicts the chains of cause-and-effect from the input signal CylinderNumber, determining the cylinder for which the injection time and ignition time parameter are requested, to the cylinder-specific output signals InjectionTime and IgnitionTime. As the engine control application under consideration controls the combustion processes in an eight-cylinder engine, there are eight chains of cause-and-effect each from the signal CylinderNumber to the signals InjectionTime[1..8] and IgnitionTime[1..8]. Furthermore, the timing requirements that are associated with the chains of cause-and-effect are shown.

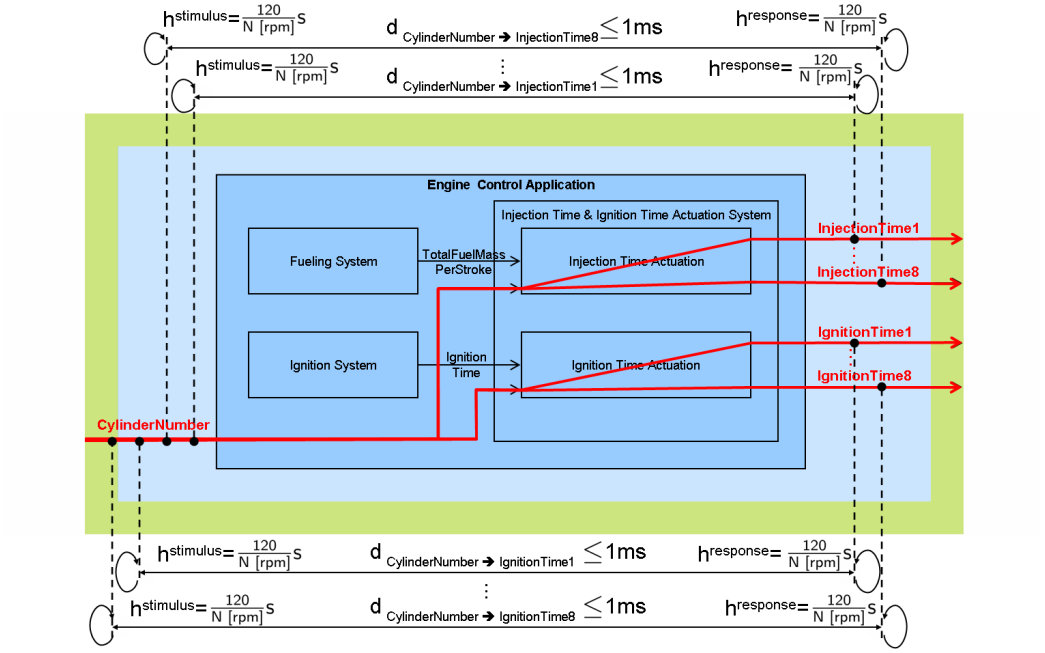


Figure 19: Chain of cause-and-effect from stimulus signal CylinderNumber to cylinder-specific response signals InjectionTime[1..8] / IgnitionTime[1..8] and associated timing requirements

For the injection time and ignition time actuation, the following timing requirements apply:

1. The latency between a cylinder-specific request and the update of the injection time and ignition time values for the respective actuators must be less than 1ms. This is expressed by

$$\mathsf{d}_{\mathsf{CylinderNumber} \rightarrow \mathsf{InjectionTime[1..8]}} \leq 1ms$$

and

$$\mathsf{d}_{\mathsf{CylinderNumber} \rightarrow \mathsf{IgnitionTime[1..8]}} \leq 1ms$$

As the latency between two cylinder-specific requests for the injection time and ignition time parameters depends on the current speed of the engine (see section 2.3), the following engine-speed dependent timing requirements apply:

2.+3. A combustion process performed according to the four-stroke cycle takes 720°CA, i.e., two engine revolutions (2 [rev/min]). At a specific engine speed N [rev/min], $\frac{\mathsf{N}\ \text{[rev/min]}}{2\ \text{[rev]}}$ combustion processes take place per minute. This translates to $\frac{\mathsf{N}\ \text{[rev/min]}}{2\ \text{[rev]}} \frac{1\ \text{[min]}}{60\ \text{[s]}} = \frac{\mathsf{N}}{120\ \text{[s]}}$ combustion processes taking place per second. The reciprocal $\frac{120\ \text{[s]}}{\mathsf{N}}$ determines the time between two consecutive combustion processes. It is thus required that at a specific engine speed N [rev/min], the latency between two consecutive cylinder-specific requests for the injection time and ignition time parameters is exactly $\frac{120}{\mathsf{N}}[s]$. Consequently, the latency between two consecutive updates of the injection time or ignition time parameter for a specific cylinder must also be exactly $\frac{120}{\mathsf{N}}[s]$. This is expressed by

$$\mathsf{h}^{\mathsf{stimulus}}_{\mathsf{CylinderNumber} \rightarrow \mathsf{InjectionTime}} = \frac{120}{\mathsf{N}} s$$

for the nominal interval between consecutive stimuli, and

$$\mathsf{h}^{\mathsf{response}}_{\mathsf{CylinderNumber} \rightarrow \mathsf{IgnitionTime}} = \frac{120}{\mathsf{N}} s$$

for the nominal interval between consecutive responses.

For example, at a specific engine speed of 2000rpm, the time between two cylinder-specific requests is $\frac{120}{2000}\mathsf{[s]} = \mathsf{0.06s} = \mathsf{60ms}$.

Figure 19 shows the signal paths from the input signals CylinderNumber to the output signals InjectionTime[1..8] and IgnitionTime[1..8], respectively. Furthermore, the timing requirements which are associated with these signal paths are shown.

3.4 Summary and Conclusion

The objective of the engine control application under consideration is to control the combustion processes in the single cylinders of an air-flow controlled, intake manifold fuel injected eight-cylinder engine in order to deliver a driving torque that corresponds to the drivers wish. At first, an overview on the functionalities of our engine control application has been given (section 3.2). For the purposes of our case study, we focus on its basic functionalities. Four different functionalities have been distinguished. These are the control of the air-flow via the throttle (air system), the calculation of injection times and the ignition times (fueling system and ignition system), and the timely delivery of the latter upon cylinder-specific requests (injection time and ignition time actuation system). Section 3.3 has then given more detailed descriptions of the elementary functions they are composed of, the important signal paths and chains of cause-and-effect that can be identified and the timing requirements that can be associated with these.

In the following section, the AUTOSAR-compliant software architecture that has been developed for the engine control application is described.

4 AUTOSAR Software Architecture and ECU System

4.1 Introduction

In order to demonstrate the applicability of the concepts of the Timing Model for AUTOSAR and the RTE Tracing approach, at first, an AUTOSAR-compliant software architecture needs to be developed for the engine control application.

The engine control application of the ETAS DemoCar project ([8], [9]) has already been re-engineered towards an AUTOSAR-compliant ECU system in the course of several previously conducted works ([3], [5], [6], [11], [12]). The AUTOSAR software architecture described in the following is an advancement of the previous works. Several modifications have been made as improvements (e.g., renaming of signals, restructuring of functions, etc.) in order to adequately apply the developed concepts of the Timing Model for AUTOSAR and the RTE Tracing approach.

In the following, the AUTOSAR software architecture is described. At first, the design decisions that have been taken for which AUTOSAR concepts have been applied in the development of an AUTOSAR-compliant software architecture are described (section 4.2). Section 4.3 then gives an overview

of the AUTOSAR-compliant software architecture for the engine control application. In section 4.4, excerpts of the software architecture are discussed which correspond to the individual functionalities of the engine control application. Furthermore, it is explained how the signal paths that have been described in section 3.3 for the basic functionalities of the engine control application are represented in the AUTOSAR-compliant software architecture. In order to realize the engine control application as single ECU system, several steps need to be taken according to the AUTOSAR methodology. These are the steps of the system configuration and the subsequent ECU configuration. These are described in section 4.5.

4.2 Description of Employed of AUTOSAR Concepts

The following design decisions have been taken with respect to the application of the AUTOSAR concepts for the specification of the application software architecture of the engine control application:

Single RunnableEntity per AtomicSoftwareComponentType: Each AtomicSoftwareComponentType is specified such that there is only a single RunnableEntity in its InternalBehavior. When being triggered, the RunnableEntity reads the values of the DataElementPrototypes, performs a data transformation of this input data to output data, and writes the output values onto the DataElementPrototypes in the PPortPrototypes.

No Client/Server communication Client/Server communication is not employed as the engine control application does not contain service-oriented parts.

Employment of Sender/Receiver communication: In order to avoid any data consistency problems due to (quasi-)parallel execution of RunnableEntities, implicit Sender/Receiver communication is employed for the continuous synchronous real-time functionalities. In those parts where the focus is on the reactivity to an external event (injection time and ignition time actuation), explicit Sender/Receiver communication is employed (reactive real-time functionalities).

Flat component hierarchy: To ease readability, only a single component hierarchy level is used. I.e., only one CompositionType is employed in which all AtomicSoftwareComponentTypes of the different functionalities are instantiated to ComponentPrototypes and where the data flow

between the components is established through AssemblyConnector-Prototypes. This CompositionType is the top-level composition representing the overall engine control application.

Single instantiation of AtomicSoftwareComponentTypes: The different AtomicSoftwareComponentTypes are each instantiated only once in the overall application software. In fact, the engine control application contains no parts where multiple instantiation could be directly applied without further modifications of the original algorithms.

Furthermore, the following specifics apply for the engine control application: In the technical setup that is used for RTE Tracing experiments with the AUTOSAR-compliant engine control application, no real engine is employed, and also no hardware for the sensors and actuators is employed (see section 6). To stimulate the inputs of the engine control application and provide adequate input values, and to also process its computed output values adequately, a simulation model is used that simulates the processes in an engine and also includes a virtual driver. The simulation model is executed on a real-time simulation hardware that is coupled to the target hardware with the AUTOSAR-compliant engine control application. The coupling is established via a CAN bus.

In contrast to that, in a real AUTOSAR-compliant ECU, sensor and actuator hardware devices are connected via the microcontroller peripherals. The latter are accessed in software through drivers that belong to the platform software (basic software modules of the microcontroller abstraction layer (MCAL), ECU abstraction layer and services layer as well as complex device drivers). The basic software modules in general provide access to the sensors and actuators by means of Client/Server communication, i.e., in the form of a service that can be called by the sensor or actuator software components.

In our case study, however, system input signals and system output signals are communicated by means of Sender/Receiver communication. Sensor software components have an RPortPrototype from which the input signal of a sensor is read; analogously, actuator software components have a PPort-Prototype onto which the output signal for an actuator is writte. This eases the definition of remote communication via the employed CAN bus.

4.3 Overview of AUTOSAR Software Architecture

Figure 20 (see next page) depicts an overview of the developed AUTOSAR software architecture of the engine control application.

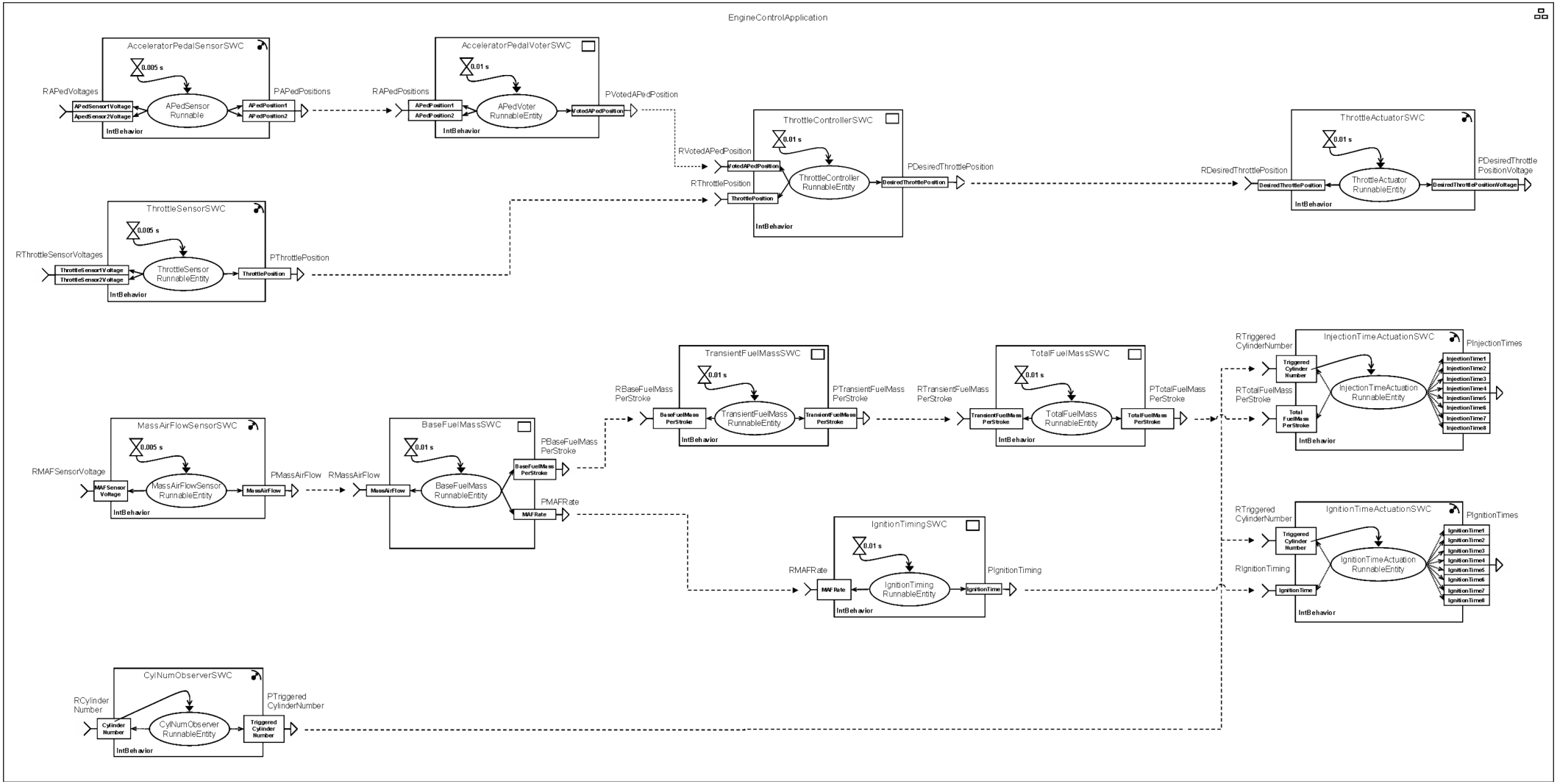


Figure 20: Overview of the AUTOSAR software architecture

The AUTOSAR software architecture of the engine control application is represented by the CompositionType EngineControlApplication that is referred to as top-level composition in the AUTOSAR system specification. It contains 13 ComponentPrototypes that are all instances of a distinct AtomicSoftwareComponentType or SensorActuatorSoftwareComponentTypes. The data-flow from system inputs to system outputs is established by means of AssemblyConnectorPrototypes.

Note that

- only those software components are shown that belong to one of the basic functionalities that are considered in our case study; the engine control application consists of several more software components that are not in the focus of our case study.

- only those PortPrototypes are shown that are relevant for later describing signal paths and timing requirements by means of the concepts of the Timing Model for AUTOSAR; the software components have several more PortPrototypes, however, these are not in the focus of our case study.

In the following, excerpts of the AUTOSAR software architecture are presented and discussed that correspond to the basic functionalities of the engine control application.

4.4 Detailed Description of Basic Functionalities

4.4.1 Air System

The functionality of the air system is realized by five software components. Each software component realizes one or more functions that have been described in section 3.3.1.

Figure 21 shows the excerpt from the AUTOSAR software architecture for the air system of the engine control application.

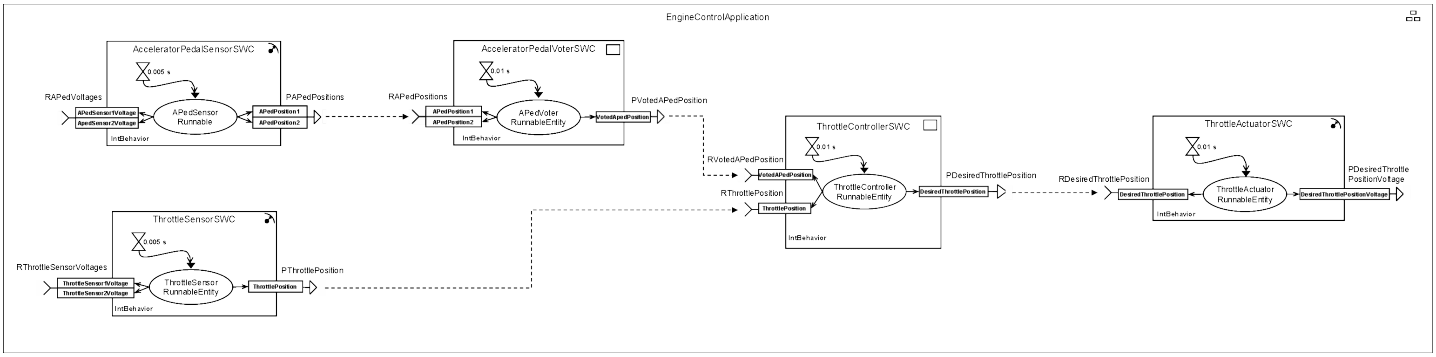


Figure 21: Excerpt from the AUTOSAR software architecture for the air system

In the following, the software components are described:

AcceleratorPedalSensorSWC This software component is a SensorActuatorSoftwareComponentType that realizes the function AcceleratorPedalSensor. The APedSensorRunnableEntity captures the current accelerator pedal position in terms of a voltage value delivered by the sensor and translates it to the corresponding accelerator pedal position in terms of a percentage value. For this, it reads the input values from the respective DataElementPrototypes (APedSensor1Voltage and APedSensor2Voltage), performs the voltage-to-percentage transformation, and writes the output values on the DataElementPrototypes of the RPortPrototype PAPedPosition (APedPosition1 and APedPosition2). As the E-GAS concept prescribes to employ redundant sensors, the voltage values of the two distinct accelerator pedal sensors are captured and translated to respective percentage values one by one.

Figure 22 depicts the graphical representation for AcceleratorPedalSensorSWC.

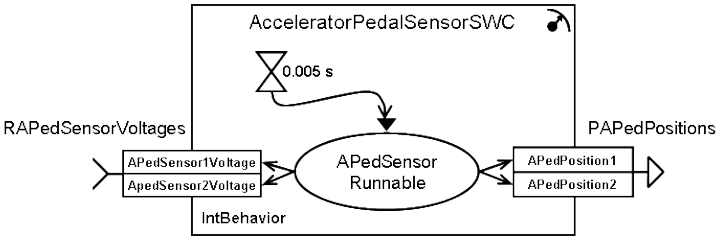


Figure 22: AcceleratorPedalSensorSWC

Note that

- the APedSensorRunnableEntity is triggered by a TimingEvent at a 5ms rate
- the accelerator pedal sensor voltage signals are clustered to a single Sender/Receiver interface which types the RPortPrototype RAPedSensorVoltages
- the APedSensorRunnableEntity can perform read actions on the accelerator pedal sensor voltage signals by means of the implicit Sender/Receiver communication pattern
- the accelerator pedal position signals are also clustered to a single Sender/Receiver interface; this is used to type the PPortPrototype PAPedPositions
- the APedSensorRunnableEntity can perform write actions on the accelerator pedal position signals by means of the implicit Sender/Receiver communication pattern.

AcceleratorPedalVoterSWC This software component is an AtomicSoftwareComponentType that realizes the function AcceleratorPedalVoter. The APedVoterRunnableEntity reads the accelerator pedal positions delivered by the AcceleratorPedalSensorSWC and computes a voted accelerator pedal position.

Figure 23 depicts the graphical representation for AcceleratorPedalVoterSWC.

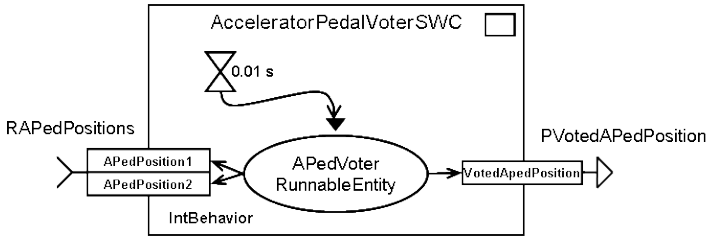


Figure 23: AcceleratorPedalVoterSWC

Note that

- the APedVoterRunnableEntity is triggered by a TimingRequirement at a 10ms rate
- the APedVoterRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

ThrottleSensorSWC This software component is a SensorActuatorSoftwareComponentType that realizes the function ThrottleSensor. The ThrottleSensorRunnableEntity captures the current throttle position from the two redundant sensors by means of voltage values, transforms them to percentage values and computes a voted throttle position value.

Figure 24 depicts the graphical representation for ThrottleSensorSWC.

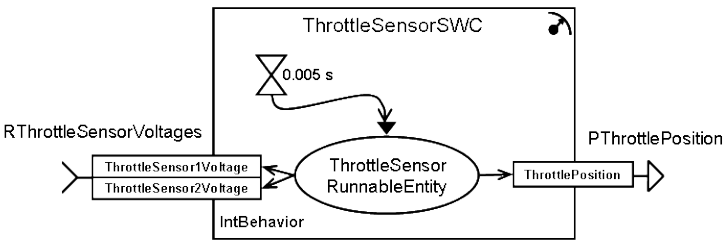


Figure 24: ThrottleSensorSWC

Note that

- the ThrottleSensorRunnableEntity is triggered by a TimingRequirement at a 5ms rate
- the ThrottleSensorRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

ThrottleControllerSWC This software component is an AtomicSoftwareComponentType that realizes the functions PedalFeel and ThrottleController. In a first step, the ThrottleControllerRunnableEntity determines the size of the desired throttle position based on the voted pedal position (function PedalFeel). It then determines the size of the new throttle position to be set based on the desired throttle position and the current throttle position (function ThrottleController). The throttle controller is realized as a PIDT1 controller without taking a potential time delay into account.

Figure 25 depicts the graphical representation for ThrottleControllerSWC.

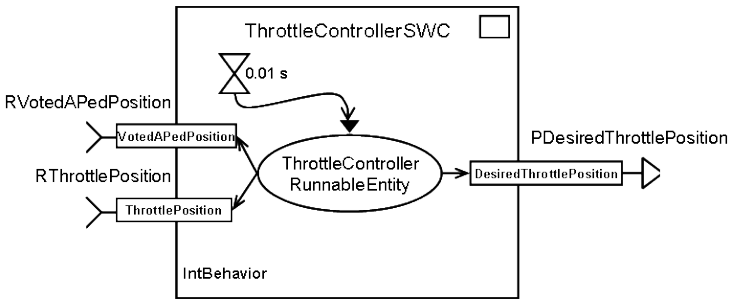


Figure 25: ThrottleControllerSWC

Note that

- the ThrottleControllerRunnableEntity is triggered by a TimingRequirement at a 10ms rate
- the ThrottleControllerRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

ThrottleActuatorSWC This software component is a SensorActuatorSoftwareComponentType that realizes the function ThrottleActuator. The ThrottleActuatorRunnableEntity takes the determined new throttle position as a percentage value and transforms it to a voltage value.

Figure 26 depicts the graphical representation for ThrottleActuatorSWC.

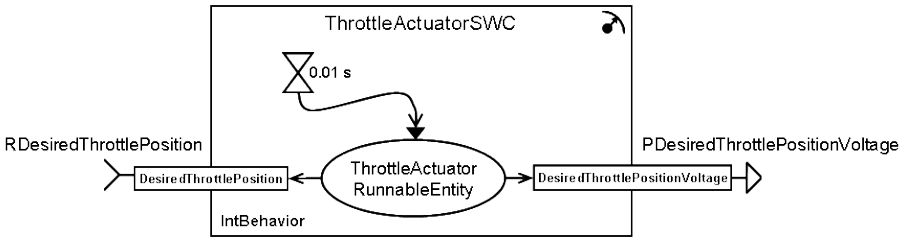


Figure 26: ThrottleActuatorSWC

Note that

- the ThrottleActuatorRunnableEntity is triggered by a TimingRequirement at a 10ms rate
- the ThrottleActuatorRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

4.4.2 Fueling System

Figure 27 shows the excerpt from the AUTOSAR software architecture for the fueling system of the engine control application.

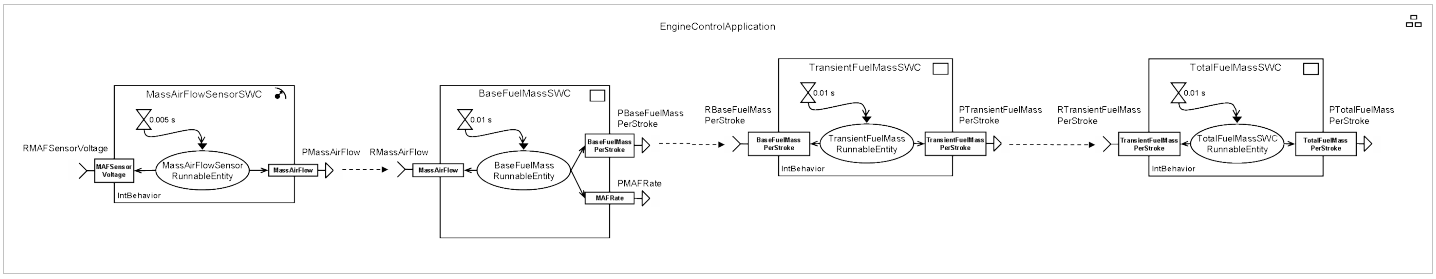


Figure 27: Excerpt from the AUTOSAR software architecture for the fueling system

In the following, the software components are described:

MassAirFlowSensorSWC This software component is a SensorActuatorSoftwareComponentType that realizes the function MassAirFlowSensor. The MassAirFlowSensorRunnableEntity captures the current mass air flow in terms of a voltage value delivered by the sensor and translates it to the model value (unit kg/h).

Figure 28 depicts the graphical representation for MassAirFlowSensorSWC.

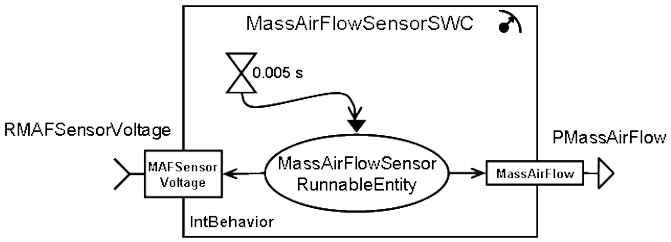


Figure 28: MassAirFlowSensorSWC

Note that

- the MassAirFlowSensorRunnableEntity is triggered by a TimingEvent at a 5ms rate

- the MassAirFlowSensorRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

BaseFuelMassSWC This software component is an AtomicSoftwareComponentType that realizes the functions AirMassFlow and BaseFuelMass. In a first step, the BaseFuelMassRunnableEntity reads the current mass air flow provided by the MassAirFlowSensorSWC and determines the mass air flow per stroke. In a second step, the base fuel mass per stroke is determined based on the mass air flow per stroke by the BaseFuelMassRunnableEntity.

Figure 33 depicts the graphical representation for BaseFuelMassSWC.

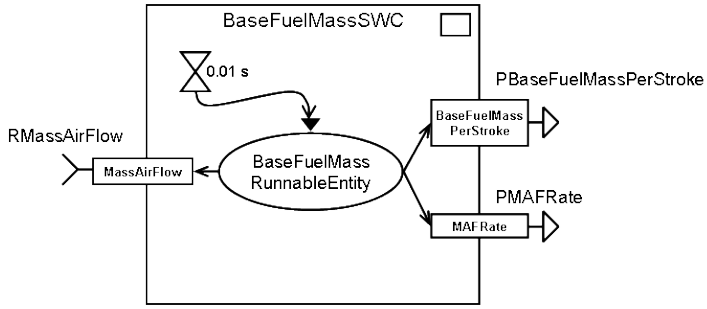


Figure 29: BaseFuelMassSWC

Note that

- the BaseFuelMassRunnableEntity is triggered by a TimingEvent at a 10ms rate
- the BaseFuelMassRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

TransientFuelMassSWC This software component is an AtomicSoftwareComponentType that realizes the function TransientFuelingCompensation. The TransientFuelMassRunnableEntity reads the determined base fuel mass per stroke provided by the BaseFuelMassSWC and adds an additional mass of fuel to compensate for specific wall-wetting effects in the intake system. It then writes the so-called transient fuel mass per stroke that the TransientFuelMassSWC provides on a PPortPrototype.

Figure 30 depicts the graphical representation for TransientFuelMassSWC.

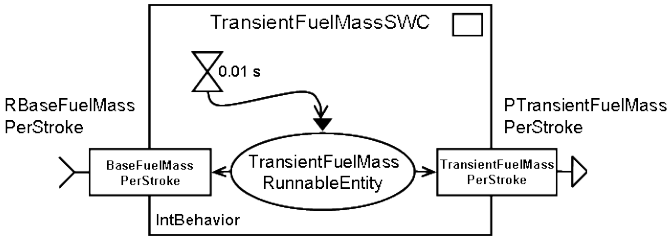


Figure 30: TransientFuelMassSWC

Note that

- the TransientFuelMassRunnableEntity is triggered by a TimingEvent at a 10ms rate
- the TransientFuelMassRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

TotalFuelMassSWC This software component is an AtomicSoftwareComponentType that realizes the function TotalFueling. The TotalFuelMassRunnableEntity reads the determined transient fuel mass per stroke provided by the TransientFuelMassSWC and determines the total fuel mass per stroke.

Figure 31 depicts the graphical representation for TotalFuelMassSWC.

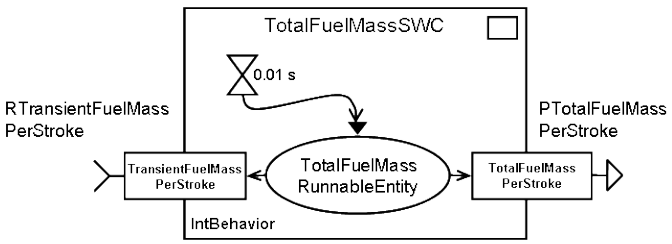


Figure 31: TotalFuelMassSWC

Note that

- the TotalFuelMassRunnableEntity is triggered by a TimingEvent at a 10ms rate
- the TotalFuelMassRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

4.4.3 Ignition System

Figure 32 shows the excerpt from the AUTOSAR software architecture for the ignition system of the engine control application.

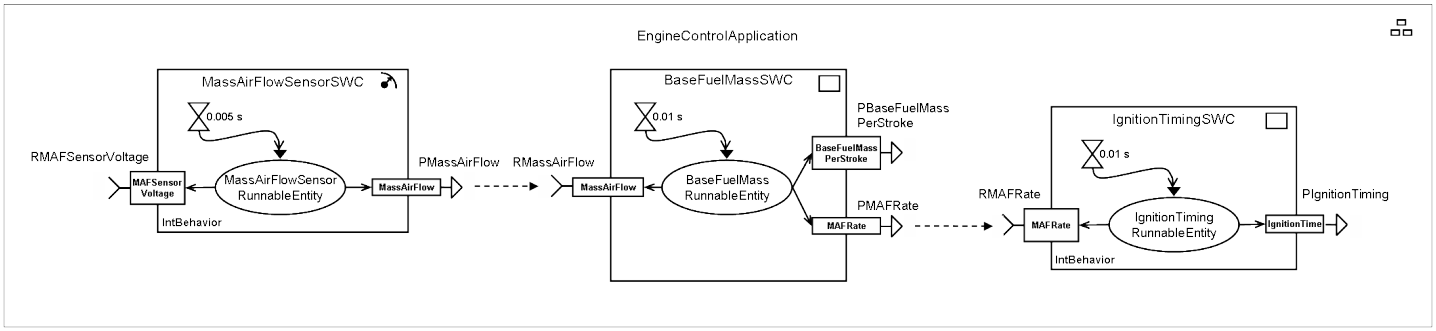


Figure 32: Excerpt from the AUTOSAR software architecture for the ignition system

In the following, the software components are described:

MassAirFlowSensorSWC This software component is a SensorActuatorSoftwareComponentType that realizes the function MassAirFlowSensor. It is the same software component that is also part of the fueling system where it has been already described.

BaseFuelMassSWC This software component is an AtomicSoftwareComponentType that realizes the functions AirMassFlow and BaseFuelMass. It is the same software component that is also part of the fueling system, however, for the ignition time computation, a different output signal is relevant: the mass air flow rate. In a first step, the BaseFuelMassRunnableEntity reads the current mass air flow provided by the MassAirFlowSensorSWC and determines the mass air flow per stroke. In a second step, the mass air flow rate is determined based on the mass air flow per stroke by the BaseFuelMassRunnableEntity.

Figure 33 depicts the graphical representation for BaseFuelMassSWC.

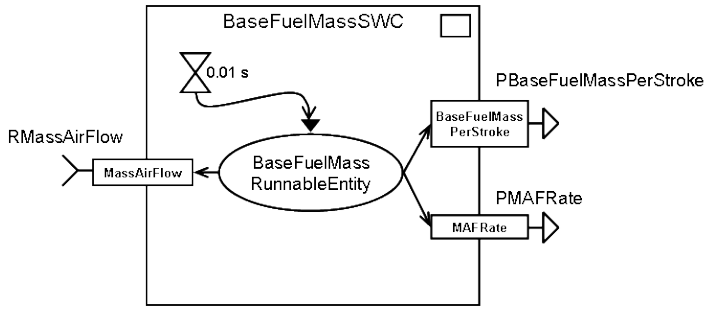


Figure 33: BaseFuelMassSWC

Note that

- this software component is also part of the fueling system
- the BaseFuelMassRunnableEntity is triggered by a TimingEvent at a 10ms rate
- the BaseFuelMassRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

IgnitionTimingSWC This software component is an AtomicSoftwareComponentType that realizes the function IgnitionTiming. The IgnitionTimingRunnableEntity reads the determined mass air flow rate provided by the BaseFuelMassSWC and determines the optimal ignition time for a combustion process. It then writes the ignition time value onto the DataElementPrototype IgnitionTime that the IgnitionTimingSWC provides on a PPortPrototype.

Figure 34 depicts the graphical representation for IgnitionTimingSWC.

Note that

- the IgnitionTimingRunnableEntity is triggered by a TimingEvent at a 10ms rate
- the IgnitionTimingRunnableEntity can access the required and provided DataElementPrototypes by means of implicit Sender/Receiver communication

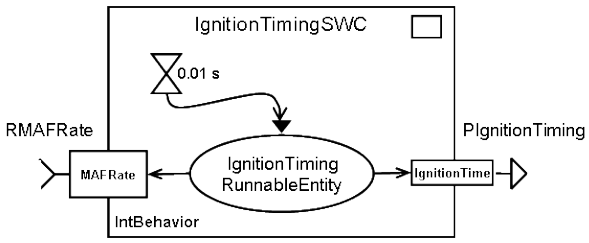


Figure 34: IgnitionTimingSWC

4.4.4 Injection Time and Ignition Time Actuation System

Figure 35 shows the excerpt from the AUTOSAR software architecture for the injection time and ignition time actuation system of the engine control application.

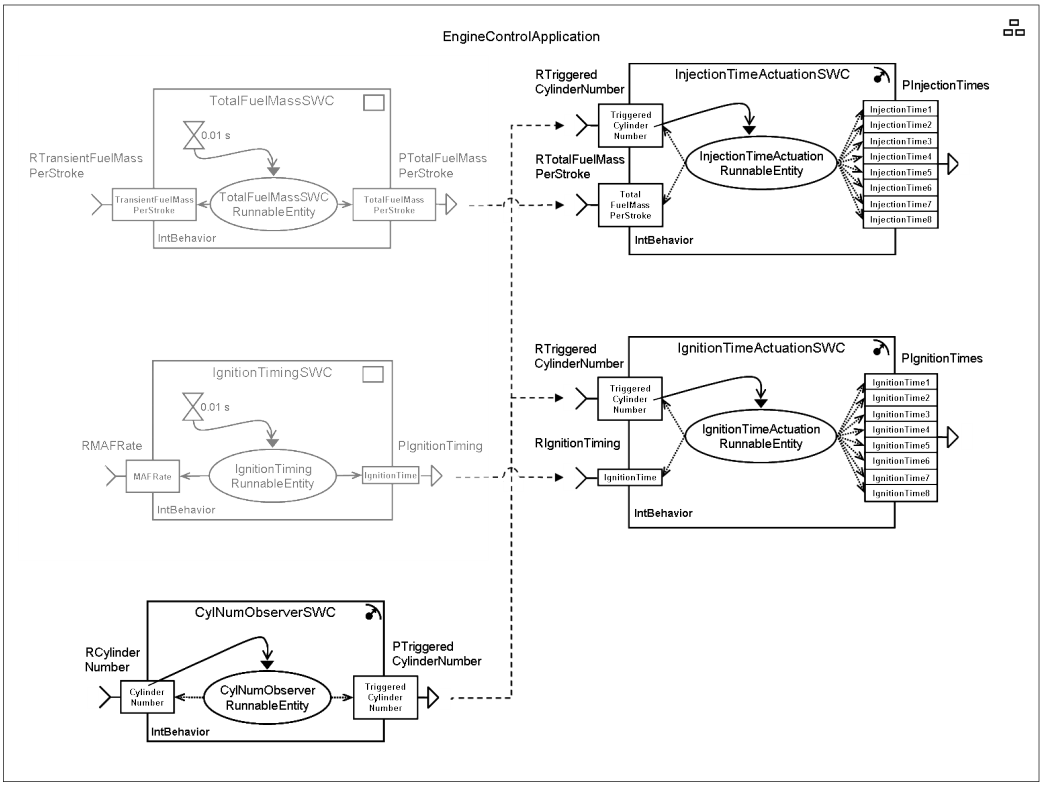


Figure 35: Excerpt for injection time and ignition time actuation system from AUTOSAR software architecture of the engine control application

The injection time and ignition time actuation is decoupled from the

calculation of the two parameters; this is performed by the fueling system and the ignition system. Actuation is performed upon cylinder-specific requests that depend on the speed of the engine. In order to detect if a new request for a cylinder-specific actuation is made, a specific mechanism is employed that is realized in software in our case study. The mechanism works as follows:

The input signal CylinderNumber is provided from a simulation model of the engine which is outside the engine control application. The value of the signal CylinderNumber determines the cylinder for which the injection time and ignition time parameters are requested.

Whenever the parameters for a combustion process are requested, the cylinder number changes. The change of the CylinderNumber is detected by the software component CylNumObserverSWC. The contained RunnableEntity is triggered upon the reception of a new CylinderNumber signal. It analyzes the value of the CylinderNumber signal and compares it with the value that has previously been sent. If a change is detected, then a request is being made, and the value of the CylinderNumber is written to the output signal TriggeredCylinderNumber. This will then trigger the RunnableEntities of the InjectionTimeActuationSWC and IgnitionTimeActuationSWC.
In the following, the software components are described:

CylNumObserverSWC This software component is an AtomicSoftwareComponentType. It evaluates a received signal CylinderNumber and determines if a request for the injection time and ignition time parameters are made by a specific cylinder. If this is the case, it writes the cylinder number onto the DataElementPrototype TriggeredCylinderNumber that is provided by the CylNumObserverSWC.

Figure 36 depicts the graphical representation for CylNumObserverSWC.

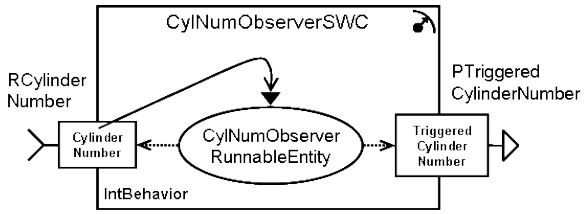


Figure 36: CylNumObserverSWC

Note that

- the CylNumObserverRunnableEntity is triggered by a DataReceivedEvent from the DataElementPrototype CylinderNumber in RPortPrototype RCylinderNumber
- the CylNumObserverRunnableEntity can access the required and provided DataElementPrototypes by means of explicit Sender/Receiver communication

InjectionTimeActuationSWC This software component is an AtomicSoftwareComponentType that realizes the function InjectionTimeActuation. When being triggered, the InjectionTimeActuationRunnableEntity reads the pre-calculated total fuel mass per stroke provided by the TotalFuelMassSWC, transforms it to the corresponding injection time value and writes the latter onto a DataElementPrototype InjectionTime[1..8] that the InjectionTimeActuationSWC provides on a PPortPrototype. The DataElementPrototype InjectionTime[1..8] is determined based on the received DataElementPrototype TriggeredCylinderNumber.

Figure 37 depicts the graphical representation for InjectionTimeActuationSWC.

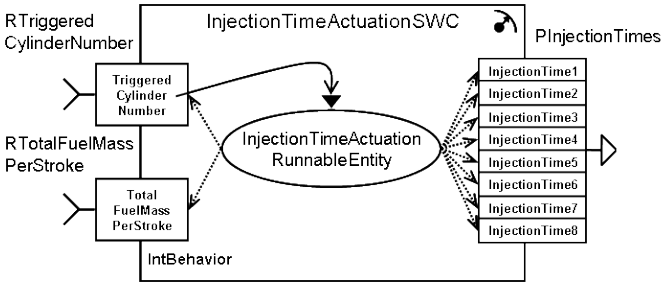


Figure 37: InjectionTimeActuationSWC

Note that

- the InjectionTimeActuationRunnableEntity is triggered by a DataReceivedEvent of the DataElementPrototype TriggeredCylinderNumber in RPortPrototype RTriggeredCylinderNumber
- the InjectionTimeActuationRunnableEntity can access the required and provided DataElementPrototypes by means of explicit Sender/Receiver communication

IgnitionTimeActuationSWC This software component is an AtomicSoftwareComponentType that realizes the function IgnitionTimeActuation. When being triggered, the IgnitionTimeActuationRunnableEntity reads the pre-calculated ignition time value provided by the IgnitionTimingSWC and writes it onto a DataElementPrototype IgnitionTime[1..8] that the IgnitionTimingSWC provides on a PPortPrototype. The DataElementPrototype IgnitionTime[1..8] is determined based on the TriggeredCylinderNumber.

Figure 38 depicts the graphical representation for IgnitionTimeActuationSWC.

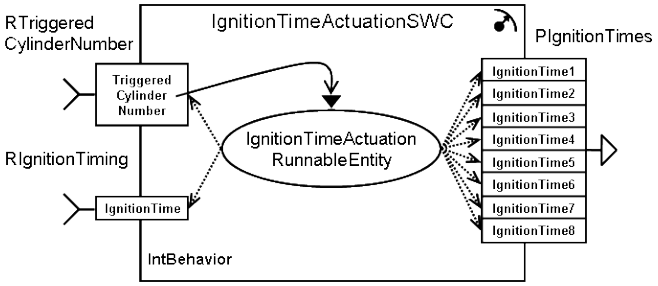


Figure 38: IgnitionTimeActuationSWC

Note that

- the IgnitionTimeActuationRunnableEntity is triggered by a DataReceivedEvent of the DataElementPrototype TriggeredCylinderNumber in RPortPrototype RTriggeredCylinderNumber
- the IgnitionTimeActuationRunnableEntity can access the required and provided DataElementPrototypes by means of explicit Sender/Receiver communication

4.5 Description of System Configuration and ECU Basic Software Configuration

4.5.1 System Configuration

The engine control application is realized as a single ECU system. This means that the system topology consists of exactly one ECU instance. All software components that together constitute the engine control application are deployed onto that ECU instance.

Figure 39 depicts an overview of the AUTOSAR system after the system configuration.

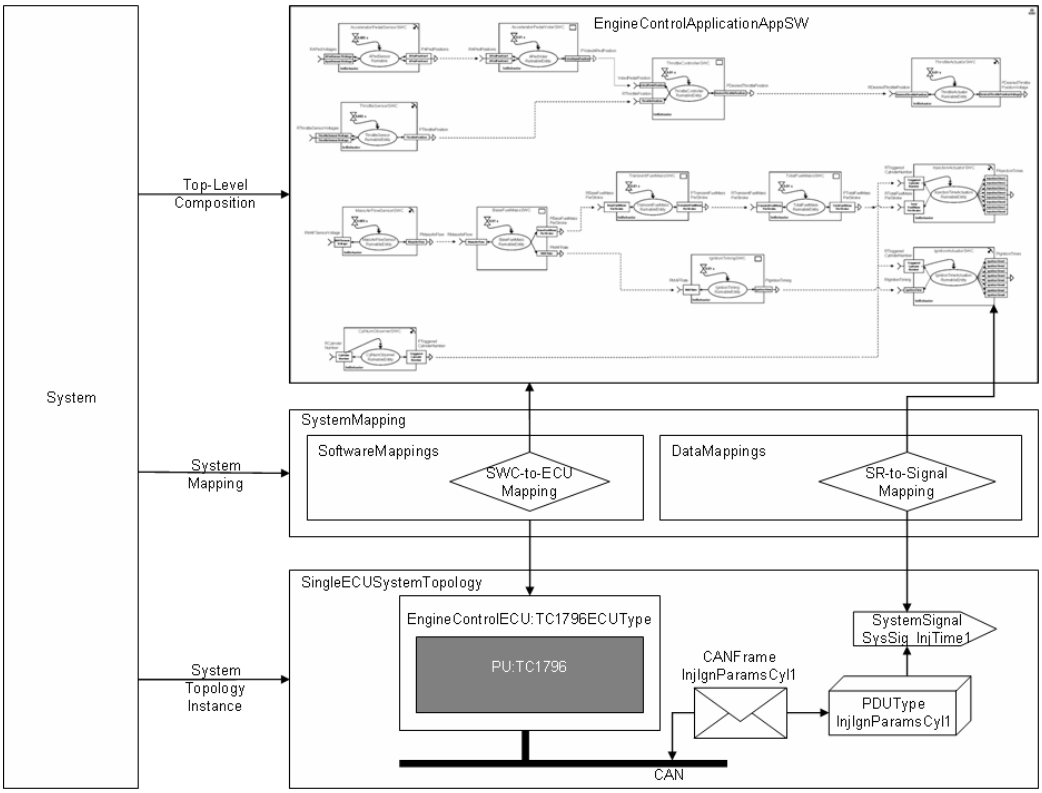


Figure 39: Overview of the AUTOSAR system after system configuration (excerpt)

The system configuration is established in the following steps:

- The CompositionType EngineControlApplicationAppSW is referred as the top-level composition by the AUTOSAR system. It thus represents the software architecture of the engine control application.
- The SystemTopologyType SingleECUSystemTopology is referred as the system topology instance by the AUTOSAR system. It comprises a single ECU instance EngineControlECU that is of type TC1796ECUType. A CANBus PowerTrainCanBus is specified over which CAN frames with input and output signals are sent and received by the engine control application.
- The link between the logical software architecture and the system topology instance is established by the system mappings. This comprises the

software-to-ECU mapping of the top-level composition to the single ECU instance of the system topology and the entailed data-mappings. The latter are required in order to send and receive the system output and input signals, i.e. the values of the DataElementPrototypes of the unconnected PortPrototypes, over the CAN bus.

With respect to the data-mappings, the following configuration decisions are of importance:

- Each DataElementPrototype of a PortPrototype that is not connected internally within the application software of the engine control application is a system input or a system output signal. In our technical setup for RTE Tracing experiments (see section 6), these need to be sent and received via the CAN bus to the real-time simulation hardware on which the simulation model of the engine is executed.

- The injection time and ignition time parameters that are requested by the engine and determined by the engine control application are grouped to a single PDUType on a cylinder-specific basis. This PDU-Type is assigned 1:1 to a CANFrame. I.e., for each cylinder, a request for the two parameters results in the transmission of a single CAN frame.

The result of the system configuration is the ECU extract for the one ECU of our AUTOSAR system. In order to build the ECU software of that ECU, the basic software needs to be configured and generated. This is described in the next section.

4.5.2 ECU Basic Software Configuration

The ECU basic software configuration of our AUTOSAR-compliant engine control application comprises the configuration of the operating system and the COM stack. In the following, we focus on the configuration decisions of the operating system as this has main influence on the execution of the engine control application. For the COM stack, it can be assumed that the basic software modules COM, PDU-R, CAN-IF and CAN-DRV are adequately configured.

Configuration of Operating System

Four OS-tasks are defined in order to execute the RunnableEntities of the application software:

Task5ms This OS-task is responsible for executing the RunnableEntities that belong to software components which read inputs from the sensors. These are the ThrottleSensorSWC, the AcceleratorPedalSensorSWC and the MassAirFlowSensorSWC. The respective RunnableEntities are assigned to this OS-task in the following order: 1. APedSensorRunnable 2. ThrottleSensorRunnableEntity 3. MassAirFlowSensorRunnableEntity.

Task10ms This OS-task is responsible for executing the RunnableEntities that belong to software components of the air system, the fueling system and the ignition system. These are the AcceleratorPedalVoterSWC, the ThrottleControllerSWC and the ThrottleActuatorSWC for the air system; the BaseFuelMassSWC, TransientFuelMassSWC and the TotalFuelMassSWC for the fueling system; and the BaseFuelMassSWC and the IgnitionTimingSWC for the ignition system. The respective RunnableEntities are assigned to this OS-task in the following order: 1. APedVoterRunnableEntity 2. ThrottleControllerRunnableEntity 3. ThrottleActuatorRunnableEntity 4. BaseFuelMassRunnableEntity 5. TransientFuelMassRunnableEntity 6. TotalFuelMassRunnableEntity 7. IgnitionTimingRunnableEntity.

TaskCylNum This OS-task is responsible for executing the RunnableEntity that belongs to the CylNumObserverSWC of the injection time and ignition time actuation system. I.e., only one RunnableEntity, CylNumObserverRunnableEntity, is assigned to this OS-task. As the CylNumObserverRunnableEntity is triggered upon the reception of a new CylinderNumber signal via COM, it makes sense to assign it to an own OS-task.

TaskInjIgnActuation This OS-task is responsible for executing the RunnableEntities that belongs to the the InjectionTimeActuationSWC and the IgnitionTimeActuationSWC of the injection time and ignition time actuation system. The respective RunnableEntities are assigned to this OS-task in the following order: 1. InjectionTimeActuationRunnableEntity 2. IgnitionTimeActuationRunnableEntity.

In order to execute the OS-tasks by the operating system, configuration parameters for these must be provided.

Figure 40 (see next page) depicts the overview of the AUTOSAR software of the engine control application. In the figure, the assignment of the RunnableEntities to the OS-tasks is shown, including the sequence in which the RunnableEntities are executed within the OS-tasks.

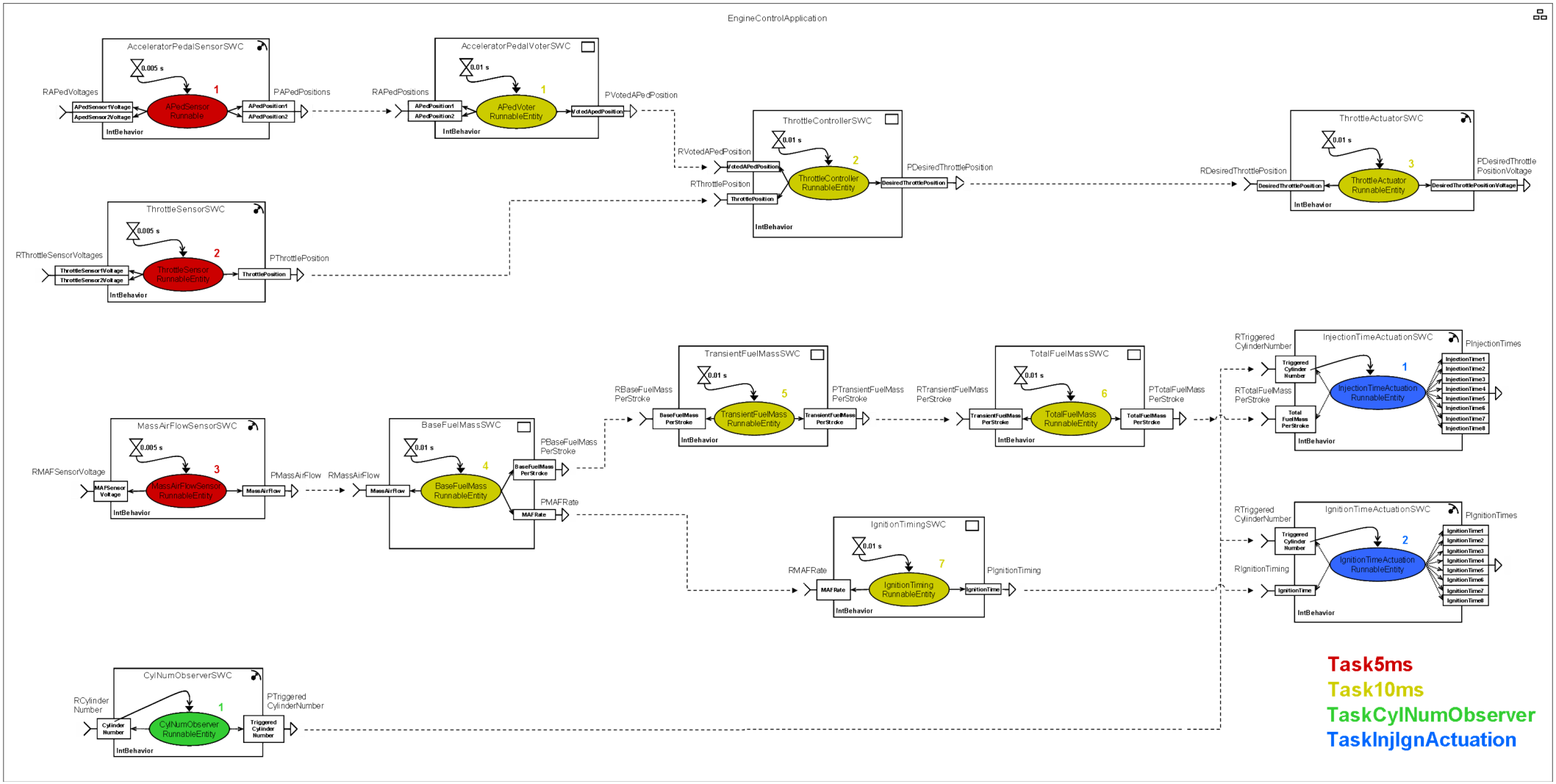


Figure 40: Overview of the AUTOSAR software architecture, including details from OS configuration

4.6 Summary

In this section, the AUTOSAR-compliant software architecture that has been developed from the legacy engine control application of the DemoCar project ([8], [9]) has been presented.

At first, the design decisions that were taken for which AUTOSAR concepts are to be applied in the development of an AUTOSAR-compliant software architecture have been described (section 4.2). Section 4.3 then gave an overview of the AUTOSAR-compliant software architecture for the engine control application. In section 4.4, excerpts of the software architecture have been discussed which correspond to the individual functionalities of the engine control application.

In order to realize the engine control application as single ECU system, several steps needed to be taken according to the AUTOSAR methodology. These are the steps of the system configuration and the subsequent ECU configuration. These have been described in section 4.5.

In the next section, it is described how the concepts of the Timing Model for AUTOSAR are applied for (i) the specification of the signal paths of the engine control application in the AUTOSAR-compliant software architecture, and (ii) the assignment of the latter with application-specific timing-requirements.

5 Application of Concepts from Timing Model for AUTOSAR

5.1 Introduction

In order to specify the signal paths of the different functionalities of the engine control application and to describe the timing requirements we employ the concepts of the Timing Model for AUTOSAR [4].

For this the following modeling steps are performed:

- In a first step, the relevant ObservableEvents within the context of the individual AtomicSoftwareComponentTypes are identified and described by means of AUTOSAR-specific ObservableEvents (RTEAPIEvents). These are then concatenated to AtomicEventChainTypes in the context of the AtomicSoftwareComponentTypes.

- In a second step, the AtomicEventChainTypes are instantiated and aggregated to CompositeEventChainTypes in the context of the CompositionType that represents the software architecture of the engine control application. These CompositeEventChainTypes are then the PathSpecifications for the functionalities of the engine control application which are associated with the application-specific timing requirements. The latter have been described in section 3.3.

In order to perform RTE Tracing experiments based on the PathSpecifications, the latter need to be first flattened and then augmented with additional ObservableEvents (OSTaskEvents) based on information stemming from the basic software configuration of the involved ECUs.

In the following, the specification of the signal paths for the different functionalities of the engine control application are described (section 5.2). This is followed by a description of the preparations for the RTE Tracing experiments in section 5.3. The latter includes the description of the augmented PathSpecifications with additional OSTaskEvents.

5.2 Specification of Signal Paths for Basic Functionalities and Association with Timing Requirements

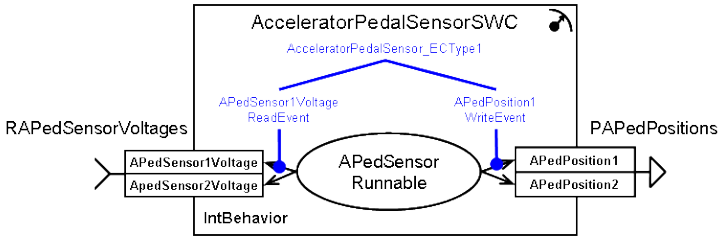
5.2.1 Air System

As described in section 4.4.1, the AUTOSAR software architecture for the engine control application comprises five AtomicSoftwareComponentTypes which together constitute the functionality of the air system. The signal

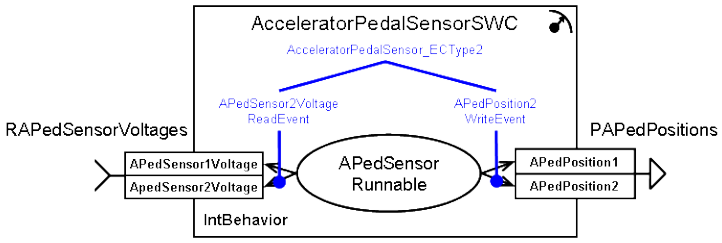
paths that are conceptually contained in the air system have been described in section 3.3.1. In the following, these signal paths are described in the AUTOSAR software architecture of the engine control application by means of the concepts of the Timing Model for AUTOSAR.

ObservableEvents and AtomicEventChainTypes

AcceleratorPedalSensorSWC Figure 41 depicts the AtomicSoftwareComponentType AcceleratorPedalSensorSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).



(a) RTEAPIEvents and AtomicEventChainType for first signal path



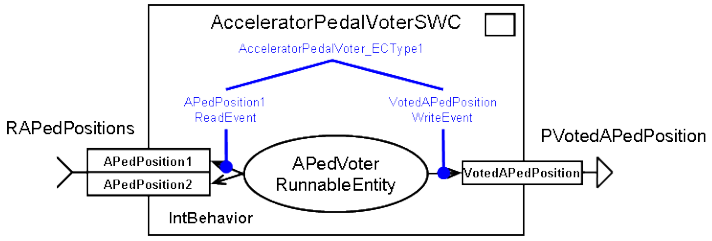
(b) RTEAPIEvents and AtomicEventChainType for second signal path

Figure 41: AcceleratorPedalSensorSWC with RTEAPIEvents and AtomicEventChainTypes

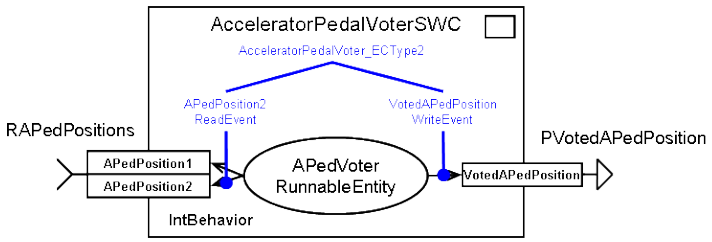
Two AtomicEventChainTypes are specified for the AcceleratorPedalSensorSWC (AcceleratorPedalSensorSWC_ECType1 and AcceleratorPedalSensorSWC_ECType2). Each AtomicEventChainType specifies that the ReceiveDataImplicitEvent must occur before the SendDataImplicitEvent such that a data transformation of the accelerator pedal sensor voltage value read by the APedSensorRunnableEntity into an accelerator pedal position percentage value that is written by the APedSensorRunnableEntity can be observed.

Note that the difference between the two AtomicEventChainTypes is that they refer to different ReceiveDataImplicitEvents.

AcceleratorPedalVoterSWC Figure 42 depicts the AtomicSoftwareComponentType AcceleratorPedalVoterSWC where the implicit read and write actions to the DataElementPrototypes have been marked.



(a) RTEAPIEvents and AtomicEventChainType for first signal path



(b) RTEAPIEvents and AtomicEventChainType for second signal path

Figure 42: AcceleratorPedalVoterSWC with RTEAPIEvents and AtomicEventChainTypes

Two AtomicEventChainTypes are specified for the AcceleratorPedalVoterSWC (AcceleratorPedalVoterSWC_ECType1 and AcceleratorPedalVoterSWC_ECType2).

Note that the difference between the two AtomicEventChainTypes is that they refer to different ReceiveDataImplicitEvents. Furthermore, the SendDataImplicitEvent VotedApedPositionWriteEvent is specified once and used twice in the two distinct AtomicEventChainTypes; this shows that the description of an ObservableEvent can be reused for the specification of different event chains.

ThrottleSensorSWC Figure 43 depicts the AtomicSoftwareComponentType ThrottleSensorSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (a ReceiveDataImplicitEvent and a SendDataImplicitEvent).

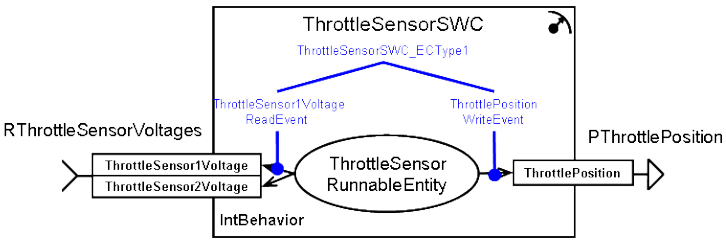


Figure 43: ThrottleSensorSWC with RTEAPIEvents and AtomicEventChainType (first signal path)

The second signal path is modeled analogously, however, the ReadDataImplicitEvent refers to the read action to the second DataElementPrototype (ThrottleSensor2Voltage).

ThrottleActuatorSWC Figure 44 depicts the AtomicSoftwareComponentType ThrottleActuatorSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

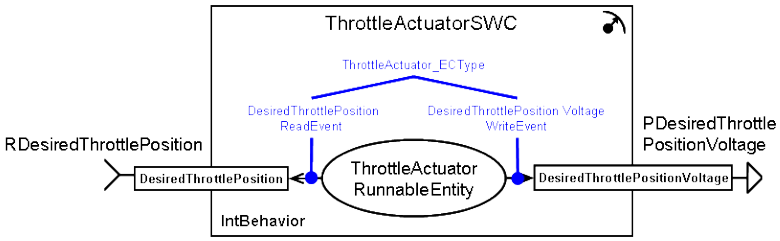
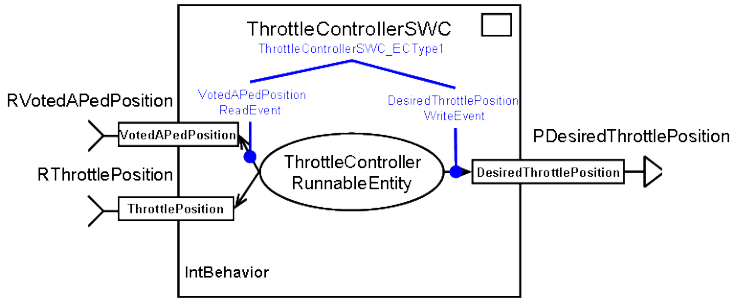
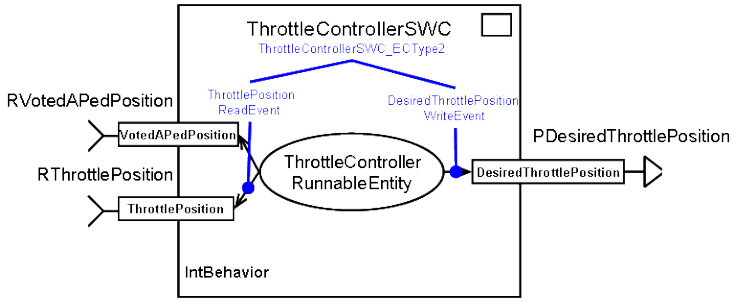


Figure 44: ThrottleActuatorSWC with RTEAPIEvents and AtomicEventChainType

ThrottleControllerSWC Figure 45 depicts the AtomicSoftwareComponentType ThrottleControllerSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (a ReceiveDataImplicitEvent and a SendDataImplicitEvent).



(a) RTEAPIEvents and AtomicEventChainType for first signal path



(b) RTEAPIEvents and AtomicEventChainType for second signal path

Figure 45: ThrottleControllerSWC with RTEAPIEvents and AtomicEventChainTypes

CompositeEventChainTypes and TimingRequirements

Figure 46 depicts the relevant excerpt of the AUTOSAR software architecture for the air system of the engine control application. Furthermore, the PathSpecification for the signal path from the input signal ThrottleSensor1Voltage to the output signal DesiredThrottlePositionVoltage is shown, including its associated timing requirements.

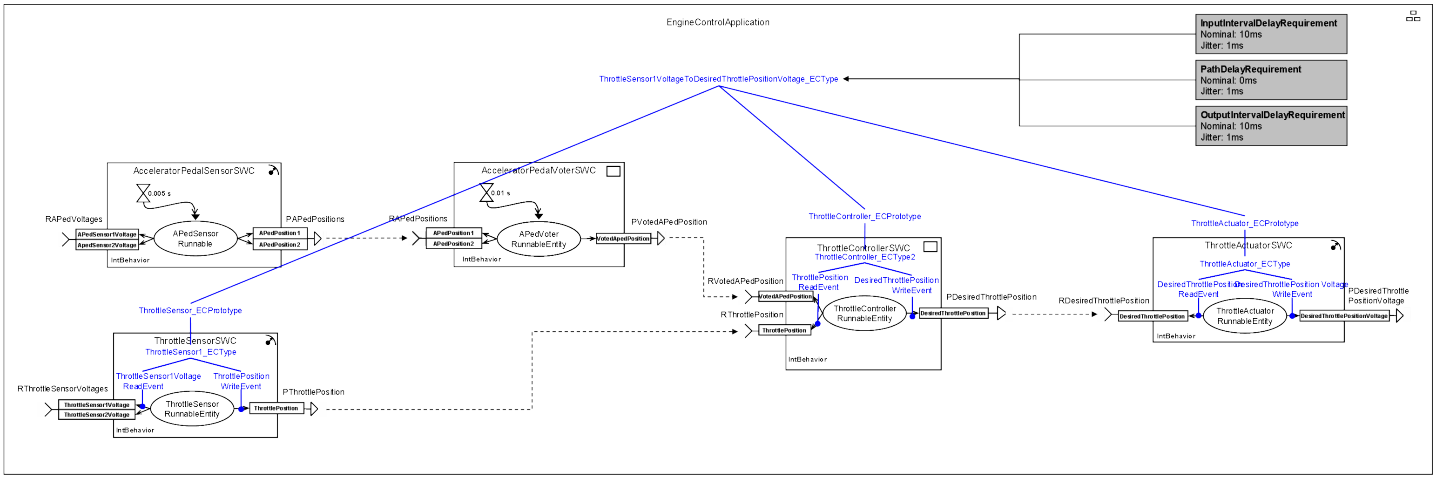


Figure 46: Excerpt for the air system with PathSpecification and associated timing requirements

The nominal feedback path delay is specified by means of a PathDelayRequirement. The nominal effective sampling and actuation rates are specified by means of an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement.

Figure 47 depicts the flattened PathSpecification.

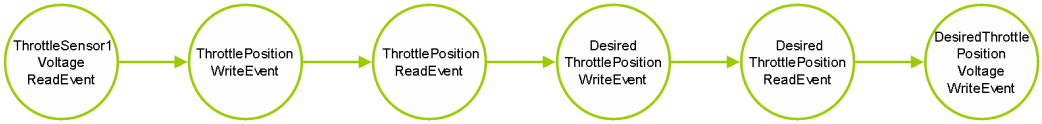


Figure 47: Flattened PathSpecification for the signal path from ThrottleSensor1Voltage to DesiredThrottlePositionVoltage

The description of the signal path from ThrottleSensor2Voltage to DesiredThrottlePositionVoltage is analogous.

Figure 48 depicts the relevant excerpt for the air system of the AUTOSAR software architecture of the engine control application, including the PathSpecifiation for the signal path from APedSensor1Voltage to DesiredThrottlePositionVoltage and its associated timing requirements.

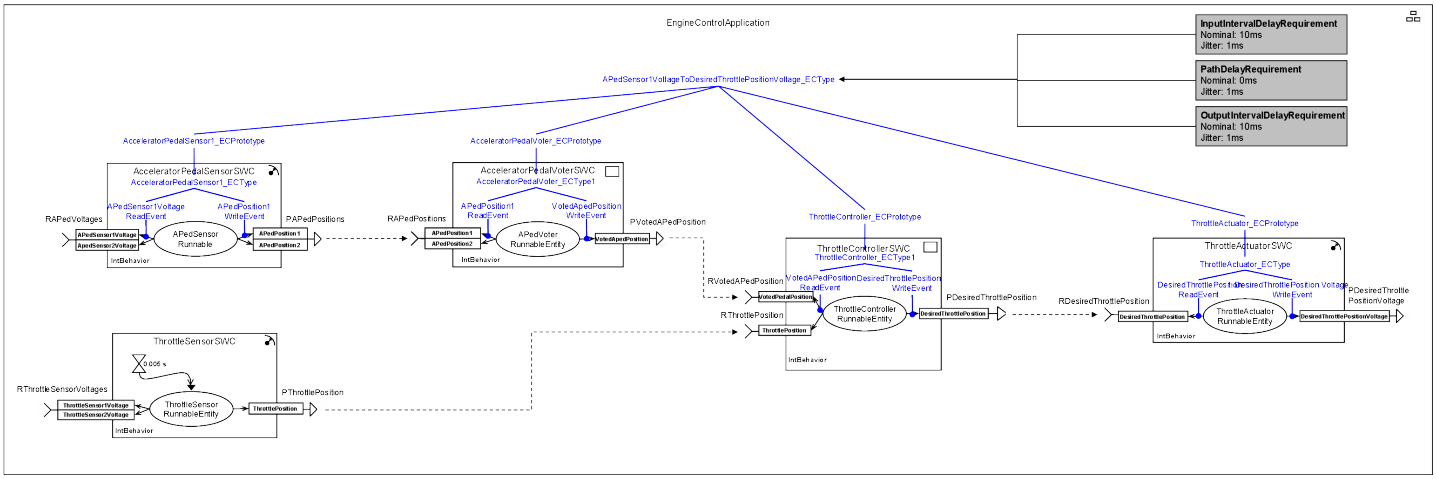


Figure 48: Excerpt for the air system with PathSpecification and associated timing requirements

The latency for the influence of the first accelerator pedal sensor on the actuated desired throttle position is specified by means of a PathDelayRequirement. The nominal effective sampling and actuation intervals are specified by means of an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement.

Figure 49 depicts the flattened PathSpecification.

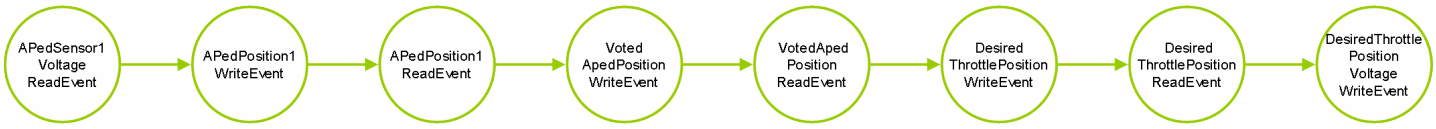


Figure 49: Flattened PathSpecification for the signal path from APedSensor1Voltage to DesiredThrottlePositionVoltage

The description of the signal path from APedSensor2Voltage to DesiredThrottlePositionVoltage is analogous.

Figure 50 depicts the relevant excerpt for the air system of the AUTOSAR software architecture. Furthermore, the two event chains that specify the signal paths from ThrottleSensor1Voltage and ThrottleSensor2Voltage to DesiredThrottlePositionVoltage are shown.

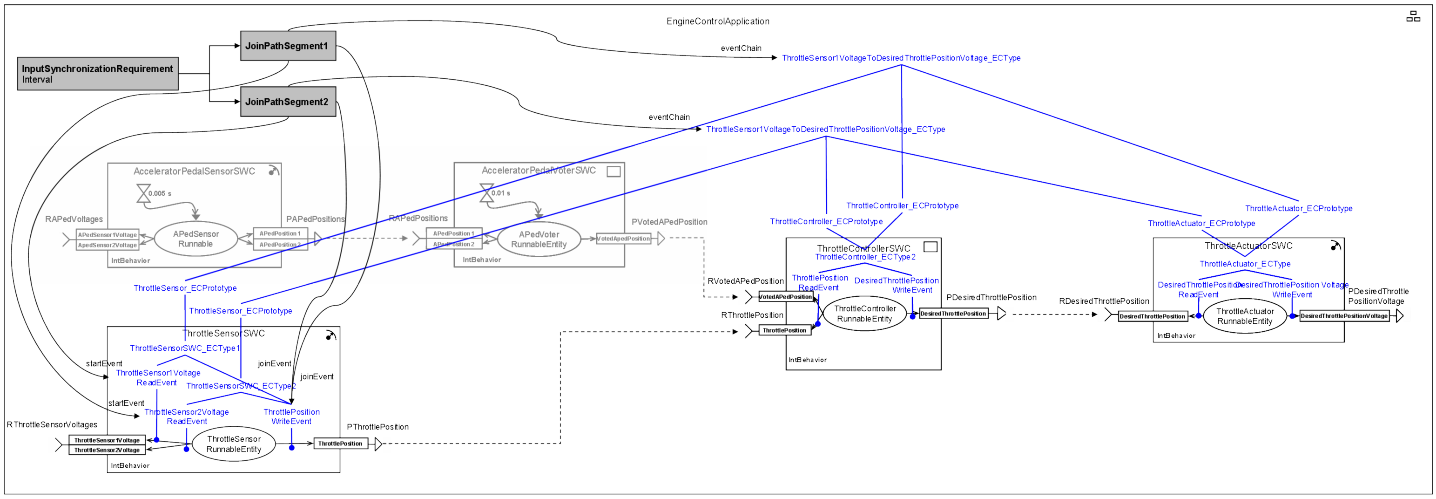


Figure 50: Excerpt for the air system with PathSpecification and associated timing requirements

The common JoinEvent is the ThrottlePositionWriteEvent which marks the implicit write action of the ThrottleSensorRunnableEntity to the merged signal ThrottlePosition. For each the two PathSpecifications, the respective JoinPathSegments that contain the ObservableEvents to be synchronized (StartEvent) and the common ObservableEvent with respect to which the synchronization is measured (JoinEvent) is described. The InputSynchronizationTimingRequirement refers to the two JoinPathSegments such that the synchronization of the input signals ThrottleSensor1Voltage and ThrottleSensor2Voltage can be specified.

Figure 51 depicts the flattened PathSpecification.

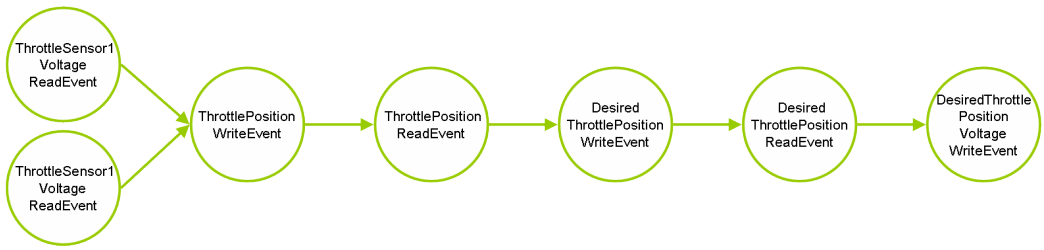


Figure 51: Flattened PathSpecification for the signal paths from APedSensor1Voltage and APedSensor2Voltage to DesiredThrottlePositionVoltage

The description of the InputSynchronizationTimingRequirement for the two input signals APedSensor1Voltage and APedSensor2Voltage is analogous.

5.2.2 Fueling System

As described in section 4.4.2, the AUTOSAR software architecture for the engine control application comprises four AtomicSoftwareComponentTypes which together constitute the functionality of the fueling system. The signal path that is conceptually contained in the fueling system has been described in section 3.3.2. In the following, this signal path is described and associated with timing requirements.

ObservableEvents and AtomicEventChainTypes

MassAirFlowSensorSWC Figure 58 depicts the AtomicSoftwareComponentType MassAirFlowSensorSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

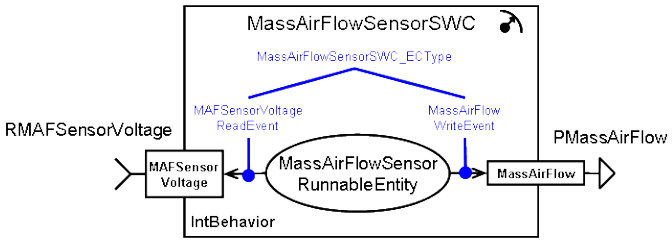


Figure 52: MassAirFlowSensorSWC with RTEAPIEvents and AtomicEventChainTypes

BaseFuelMassSWC Figure 59 depicts the AtomicSoftwareComponentType BaseFuelMassSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

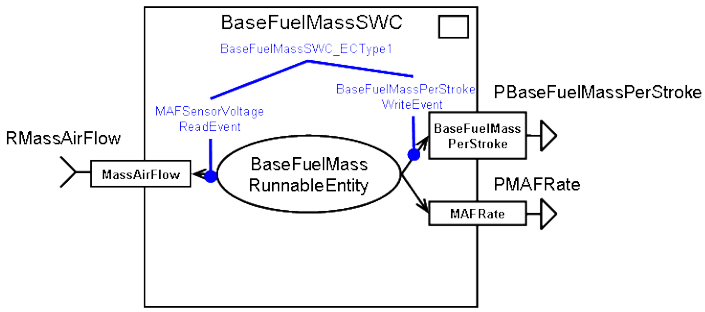


Figure 53: BaseFuelMassSWC with RTEAPIEvents and AtomicEventChainTypes

TransientFuelMassSWC Figure 54 depicts the AtomicSoftwareComponentType TransientFuelMassSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

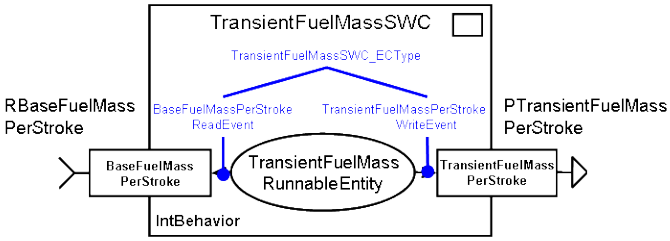


Figure 54: TransientFuelMassSWC with RTEAPIEvents and AtomicEventChainTypes

TotalFuelMassSWC Figure 55 depicts the AtomicSoftwareComponentType TotalFuelMassSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

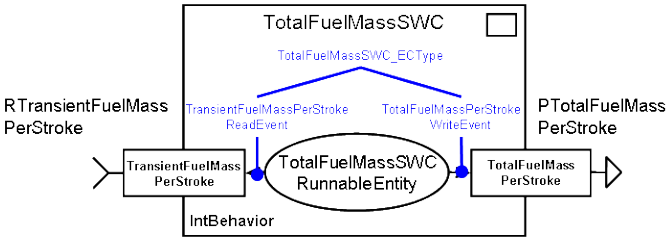


Figure 55: TotalFuelMassSWC with RTEAPIEvents and AtomicEventChainTypes

CompositeEventChainTypes and TimingRequirements

Figure 56 depicts the relevant excerpt of the AUTOSAR software architecture for the fueling system of the engine control application.

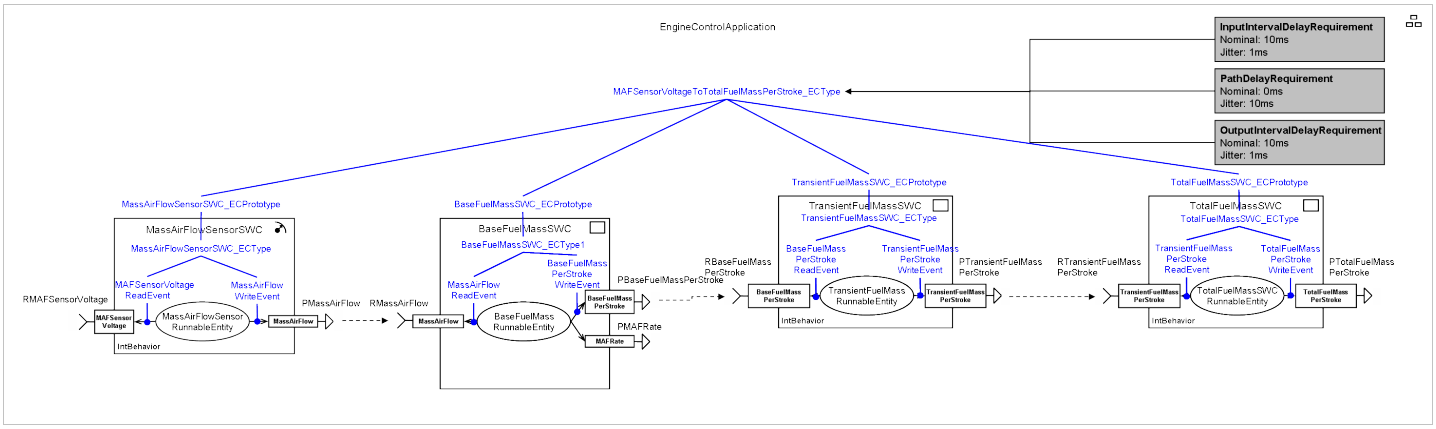


Figure 56: Excerpt for the fueling system with PathSpecification and associated timing requirements

Figure 57 depicts the flattened PathSpecification.

Figure 57: Flattened PathSpecification for the signal path from MAFSensorVoltage to TotalFuelMassPerStroke

The nominal path delay is specified by means of a PathDelayRequirement. The nominal effective sampling and actuation rates are specified by means of an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement.

5.2.3 Ignition System

As described in section 4.4.3, the AUTOSAR software architecture for the engine control application comprises three AtomicSoftwareComponentTypes which together constitute the functionality of the ignition system. The signal path that is conceptually contained in the ignition system has been described in section 3.3.3. In the following, this signal path is described and associated with timing requirements.

ObservableEvents and AtomicEventChainTypes

MassAirFlowSensorSWC Figure 58 depicts the AtomicSoftwareComponentType MassAirFlowSensorSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

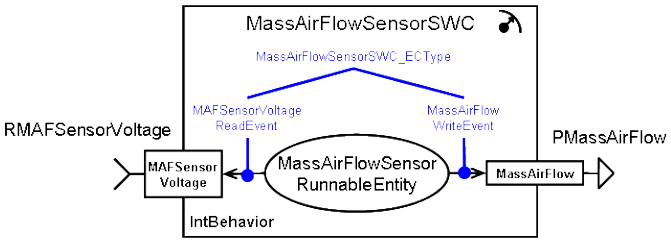


Figure 58: MassAirFlowSensorSWC with RTEAPIEvents and AtomicEventChainTypes

BaseFuelMassSWC Figure 59 depicts the AtomicSoftwareComponentType BaseFuelMassSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

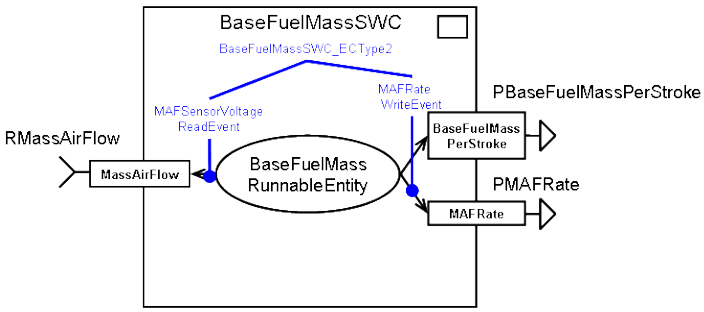


Figure 59: BaseFuelMassSWC with RTEAPIEvents and AtomicEventChainTypes

IgnitionTimingSWC Figure 60 depicts the AtomicSoftwareComponentType IgnitionTimingSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataImplicitEvent and SendDataImplicitEvent).

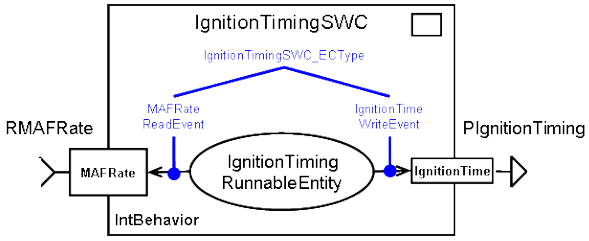


Figure 60: IgnitionTimingSWC with RTEAPIEvents and AtomicEventChainTypes

CompositeEventChainTypes and TimingRequirements

Figure 61 depicts the relevant excerpt of the AUTOSAR software architecture for the fueling system of the engine control application.

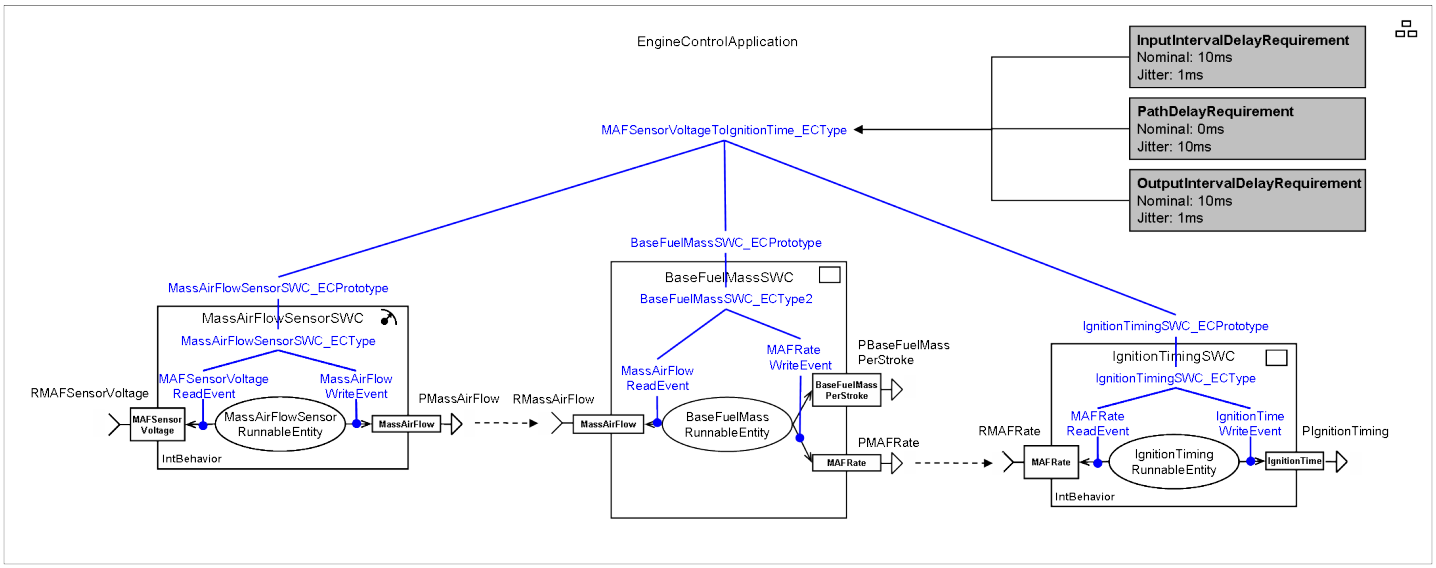


Figure 61: Excerpt for the ignition system PathSpecification and associated timing requirements

The nominal path delay is specified by means of a PathDelayRequirement. The nominal effective sampling and actuation rates are specified by

means of an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement.

Figure 62 depicts the flattened PathSpecification.

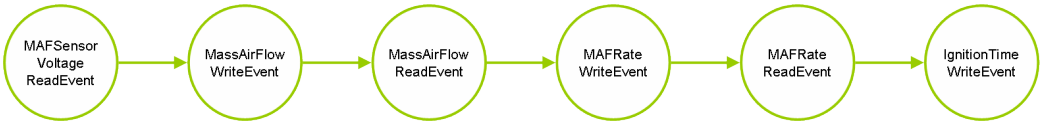


Figure 62: Flattened PathSpecification for the signal path from MAFSensor-Voltage to IgnitionTiming

5.2.4 Injection Time and Ignition Time Actuation System

As described in section 4.4.4, the AUTOSAR software architecture for the engine control application comprises three AtomicSoftwareComponentTypes which together constitute the functionality of the injection time and ignition time actuation system. The signal paths that are conceptually contained in the injection time and ignition time actuation system have been described in section 3.3.4. In the following, these signal paths are described by means of event chains.

ObservableEvents and AtomicEventChainTypes

CylNumObserverSWC Figure 63 depicts the AtomicSoftwareComponentType CylNumObserverSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataExplicitEvent and SendDataExplicitEvent).

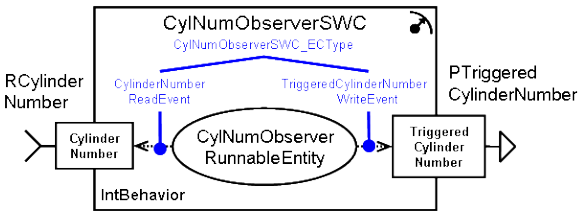


Figure 63: CylNumObserverSWC with RTEAPIEvents and AtomicEventChainTypes

InjectionTimeActuationSWC Figure 64 depicts the AtomicSoftwareComponentType InjectionTimeActuationSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataExplicitEvent and SendDataExplicitEvent).

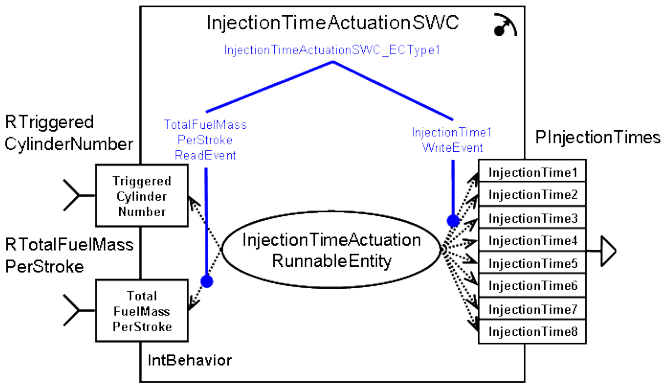


Figure 64: InjectionTimeActuationSWC with RTEAPIEvents and AtomicEventChainTypes

IgnitionTimeActuationSWC Figure 65 depicts the AtomicSoftwareComponentType IgnitionTimeActuationSWC where the implicit read and write actions to the DataElementPrototypes have been marked by RTEAPIEvents (ReceiveDataExplicitEvent and SendDataExplicitEvent).

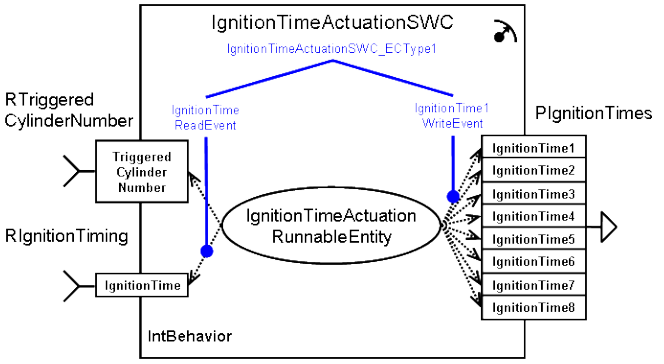


Figure 65: IgnitionTimeActuationSWC with RTEAPIEvents and AtomicEventChainTypes

CompositeEventChainTypes and TimingRequirements

Figure 66 depicts the relevant excerpt of the AUTOSAR software architecture for the fueling system of the engine control application.

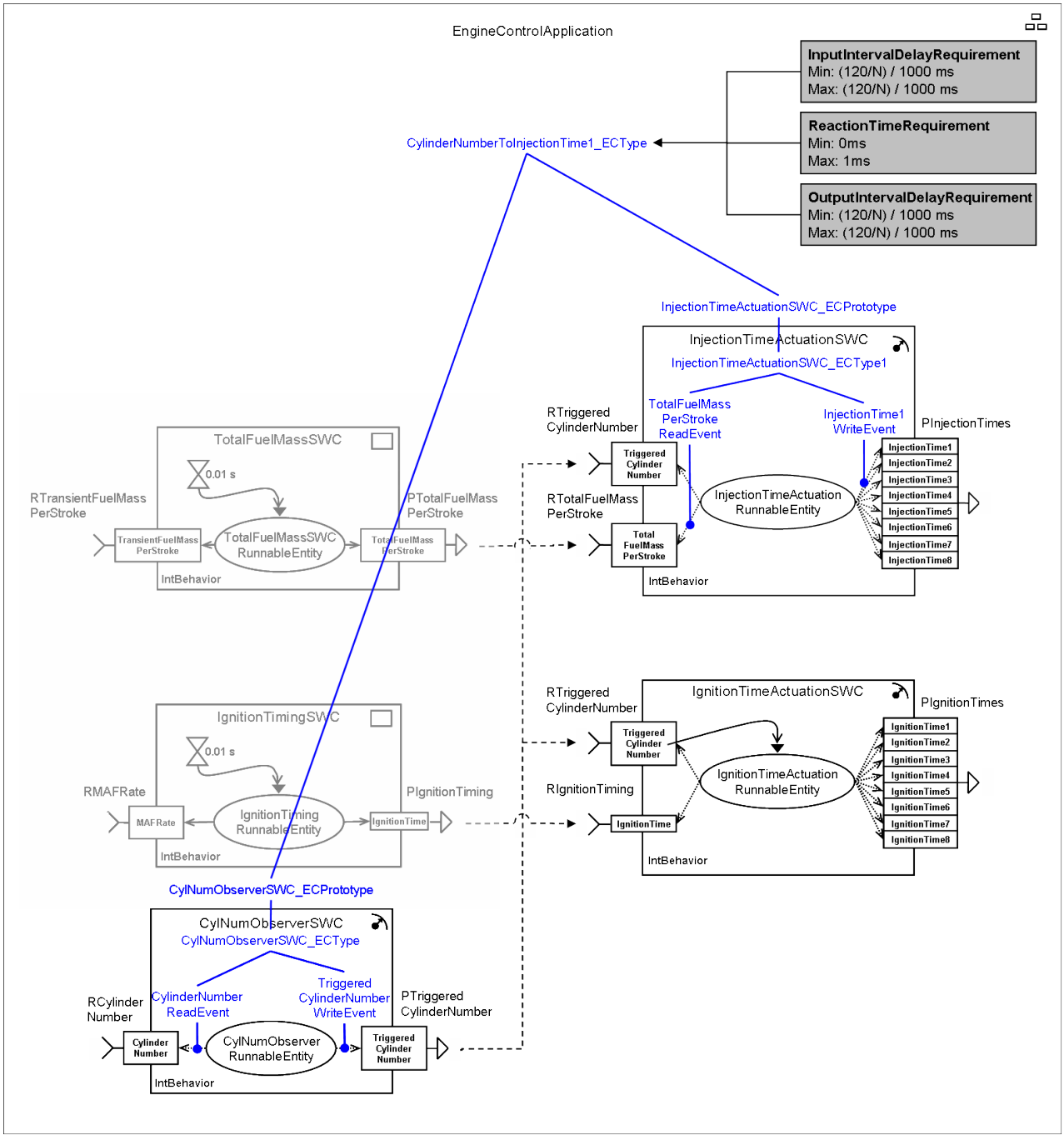


Figure 66: Excerpt for the injection time and ignition time actuation system with PathSpecification and associated timing requirements

Figure 67 depicts the flattened PathSpecification.

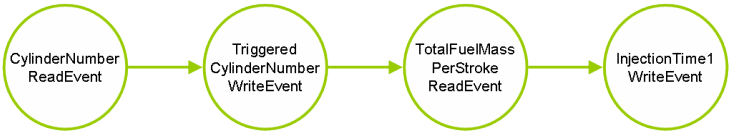


Figure 67: Flattened PathSpecification for the signal path from CylinderNumber to InjectionTime1

The nominal path delay is specified by means of a PathDelayRequirement. The nominal effective sampling and actuation rates are specified by means of an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement.

The PathSpecifications for the other InjectionTime actuations are analogous.

Figure 68 depicts the relevant excerpt of the AUTOSAR software architecture for the fueling system of the engine control application.

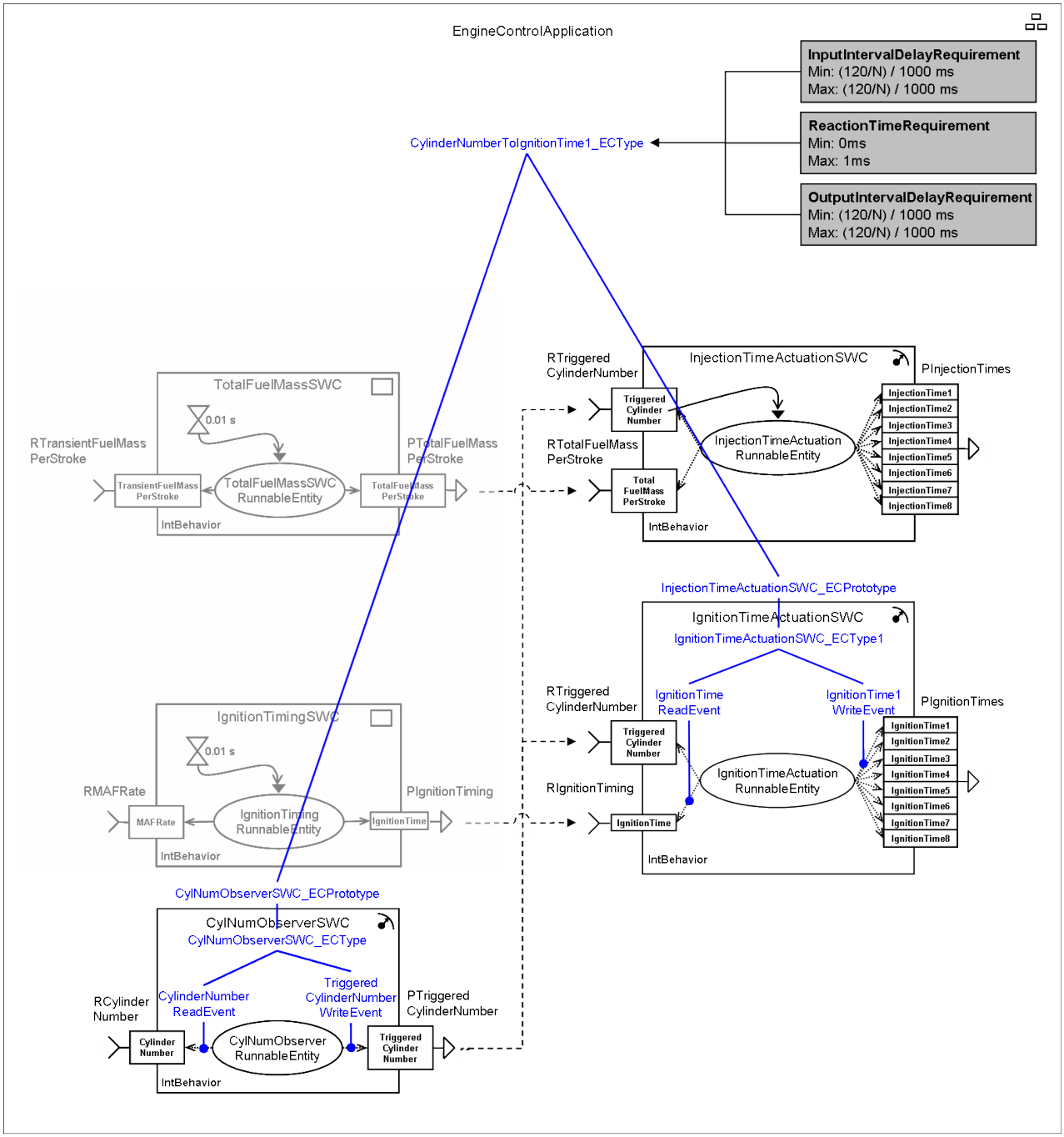


Figure 68: Excerpt for the injection time and ignition time actuation system with PathSpecification and associated timing requirements

Figure 69 depicts the flattened PathSpecification.

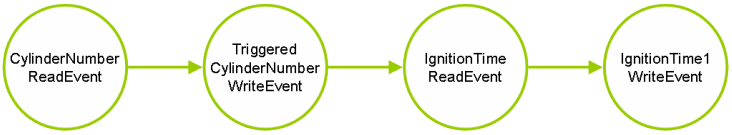


Figure 69: Flattened PathSpecification for the signal path from CylinderNumber to IgnitionTime1

The nominal path delay is specified by means of a PathDelayRequirement. The nominal effective sampling and actuation rates are specified by means of an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement.

The PathSpecifications for the other IgnitionTime actuations are analogous.

5.3 Preparations for RTE Tracing Experiments

In order to conduct an RTE Tracing experiment with the AUTOSAR-compliant engine control application, an instrumentation for the RTE of the single-ECU system is required. This can be obtained either manually or automatically by implementing the VFB Trace hook functions with functions to log the occurrences of ObservableEvents with a current time stamp.

For our case study, the RTE Tracing instrumentation has automatically been generated with the help of a prototype tool. The prototype tool builds upon the textual description language for AUTOSAR (ARDSL) that has been developed to model AUTOSAR systems.

From the textual ARDSL specifications for the AUTOSAR compliant engine control application and the PathSpecifications for the different functionalities, an RTE Tracing instrumentation is generated. To log the occurrences of the ObservableEvents, trace-points are generated for the logic analysis tool RTA-TRACE.

Figure 70 depicts an overview of the layered ECU software architecture for the engine control application. The target platform is an evaluation board with Infineon TriCore1796 microcontroller. The RTE Tracing instrumentation is also shown.

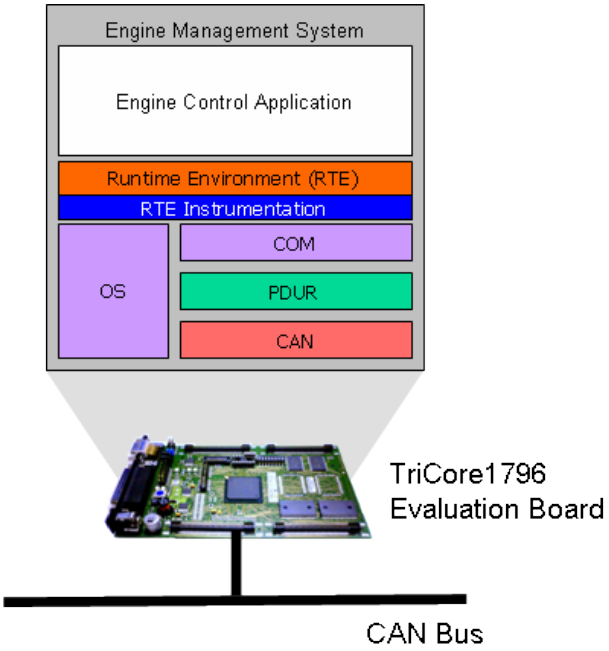


Figure 70: Overview of AUTOSAR compliant ECU software architecture, including RTE Tracing instrumentation

6 Technical Setup for RTE Tracing Experiments

6.1 Introduction

In order to conduct an RTE Tracing experiment with the engine control application, a technical setup is required where the application is executed on a platform that is close to the real-world target, where the inputs are adequately provided or stimulated, and where the outputs are adequately processed. At the same time, timing data in the form of the occurrences of relevant ObservableEvents need to be captured in order to determine the timing properties.

For this purpose, we have designed a hardware-in-the-loop (HiL) test setup where the AUTOSAR-compliant engine control application is realized as a single ECU system and operated against a simulated plant model.

Figure 71 depicts an overview of the technical setup employed to perform RTE Tracing experiments in order to obtain meaningful timing data.
In the following, the different parts of the setup are explained:

- The engine control application is realized as a single ECU system with

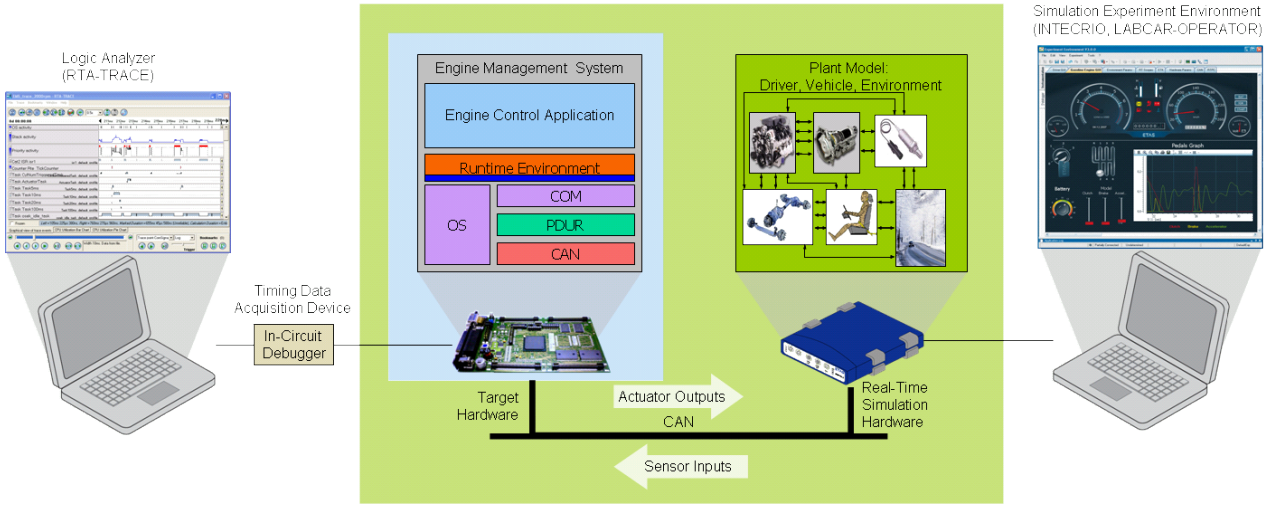


Figure 71: Overview of the technical setup for RTE Tracing experiments: the AUTOSAR-compliant engine control application is operated in-the-loop with a simulated model of the engine, driver and environment

an AUTOSAR-compliant ECU software architecture. The platform software is constituted by the mandatory operating system (OS) module and modules that form a communication stack with a CAN driver (COM, PDUR, CAN). The application software of the engine control application is executed on top of a runtime-environment (RTE).

- The inputs and outputs of the engine control application are sent over a CAN bus to the plant model that is executed on a real-time simulation hardware.

- To conduct RTE Tracing experiments it is necessary to instrument the RTE. For this purpose, an RTE Tracing instrumentation has automatically been generated from the ARDSL description of the engine control application. For our experiments, the RTE Tracing instrumentation is based on all PathSpecifications that have been defined for the different functionalities. I.e., timing data for all PathSpecifications for which timing requirements have been defined is captured at the same time. Alternatively, we could also focus on an individual PathSpecification or only a subset of PathSpecifications.

- The plant model is executed on a real-time simulation hardware (ES900 from ETAS). The latter provides excessive computing power (800MHz

floating-point CPU) and memory (512 MB Flash) such that the complex plant model can be executed in real-time.

- The plant model simulation is operated from an operational PC with a real-time simulation experimentation environment. The real-time simulation hardware is under control of the operational PC. The software allows the configuration of the parameters of the plant model for the simulation experiment and to display the values of measurement data (value domain) by means of a set of configurable instruments.

- The RTE of the AUTOSAR-compliant engine control application is equipped with an instrumentation for RTE Tracing. During runtime, timing data in the form of occurrences of ObservableEvents is produced and captured. This timing data needs to be collected and transferred to a PC in order to be analyzed. For this purpose, a state-of-the-art logic analyzer (RTA-TRACE [2] from ETAS) is employed. The RTA-TRACE application is executed on a PC which is connected to the target hardware through a in-circuit debugger (ICD). The ICD serves as timing data acquisition device. During runtime, the timing data that is produced by the RTE Tracing instrumentation is collected and uploaded to the PC where it is processed and displayed by the logic analyzer application (time-domain).

6.2 Description of the Plant Model

The gasoline engine vehicle model (GEVM) [7] from ETAS is used in HiL environments to test the functionality of engine control units. It is a Matlab/Simulink based model containing subsystems mainly to simulate the physical processes in a gasoline engine (combustion processes, air-flow related processes in the intake system, ignition system, etc.). Furthermore, it contains the model of a vehicle and its driving environment in order to simulate different kinds of driving scenarios. To perform automatic tests, it also contains the model of a virtual test driver. Through this, automatic tests can be performed by selecting one of the provided standard drive cycles ([7]).

6.3 Results from In-The-Loop Experiments

In the following, the results from an example closed-loop operation of our AUTOSAR-compliant engine control application against the simulated environment are discussed.

It will be shown that the engine control application operates as desired in the value domain, i.e., it performs all its operations as desired and delivers adequate outputs to provided inputs. This is then the basis for the conduction of RTE Tracing experiments in order to obtain meaningful timing data.

6.3.1 Development of Input Signals for Engine Control Application

The following figures depict the development of certain signals as they are delivered by the simulated plant as inputs to the engine control application.

Figure 72 depicts the development of the accelerator pedal position as it is provided by the two simulated accelerator pedal sensors in the plant model.

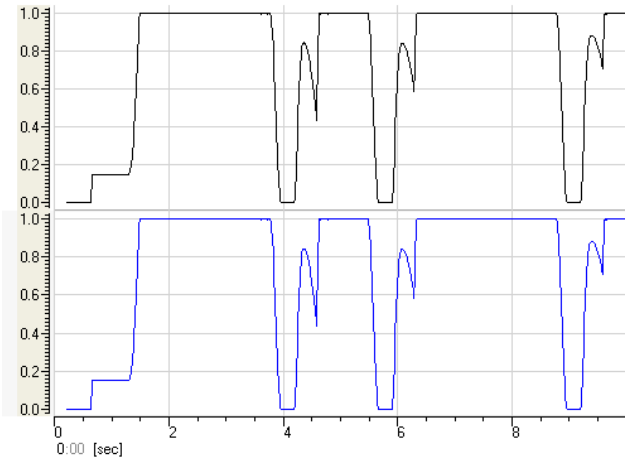


Figure 72: Accelerator pedal position

Figure 73 depicts the development of the accelerator pedal positions[4] in relation to the the development of the clutch position and the selected gear at the given point in time. The clutch and gear selection are also operated by the virtual driver.

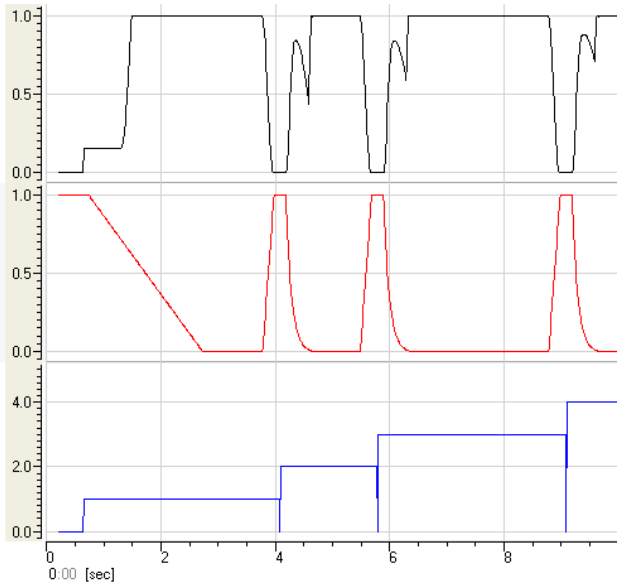


Figure 73: Accelerator pedal position in relation to clutch position and selected gear

Note that the clutch position and the selected gear are not input signals for our engine control application. They are provided here for better understandability of the other input signals and their development during acceleration.

The accelerator pedal position shows how the virtual driver accelerates in the course of the drive cycle. When the engine speed reaches approximately 4500rpm, the virtual driver shifts one gear up. From the relation between the accelerator pedal and the clutch positions it can be seen how the virtual driver performs a shifting process: the accelerator pedal is released, the clutch is pushed, the gear is changed, and the clutch is released again. After that, the accelerator pedal is pushed again, the engine speed increases and the vehicle accelerates. Note that right after the start, the clutch is released rather smoothly; during the following shifting processes, the clutch is pressed and released faster.

[4]the accelerator pedal positions overlap

Figure 74 depicts the speed of the simulated vehicle as provided by the simulated plant.

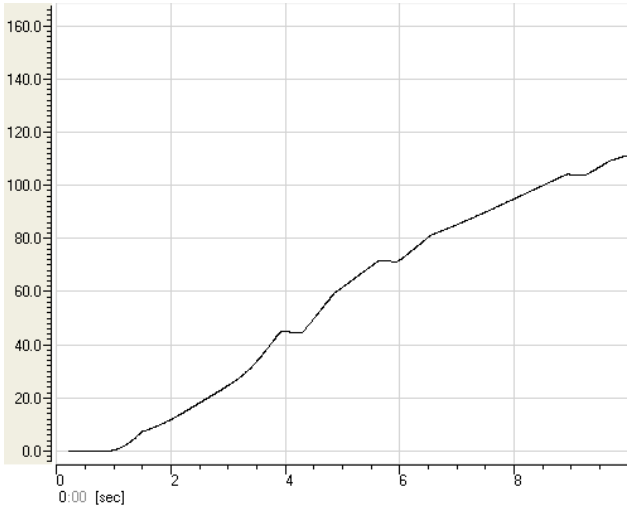


Figure 74: Vehicle speed

Figure 75 depicts the development of the engine speed as provided by the engine speed sensor in the plant model.

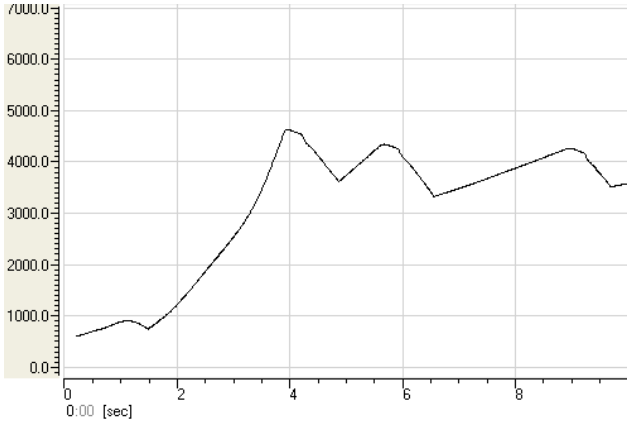


Figure 75: Engine speed

The development of the engine speed shows how the engine behaves during acceleration. The gear shifting processes can be easily identified as the effect is a decrease of the engine speed. The development of the vehicle speed shows that the vehicle is accelerating from zero to over 100 km/h in the measured time frame. Here, the gear shifting processes can also be identified as irregularities in the development of the vehicle speed signal.

Figure 76 depicts the development of the throttle angle as it is provided by the two simulated throttle sensors in the plant model. Figure 77 depicts the development of the mass air flow as provided by the simulated mass air flow sensor in the plant model.

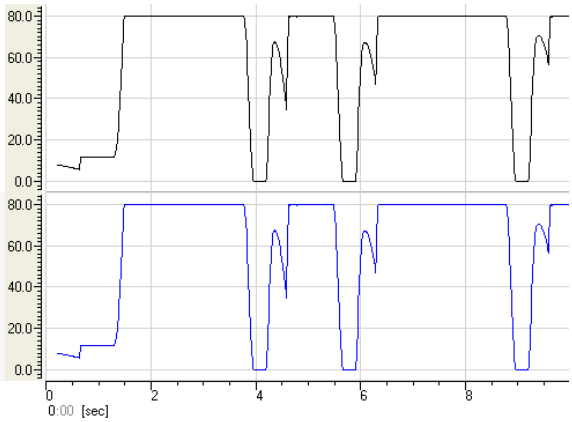


Figure 76: Throttle angle

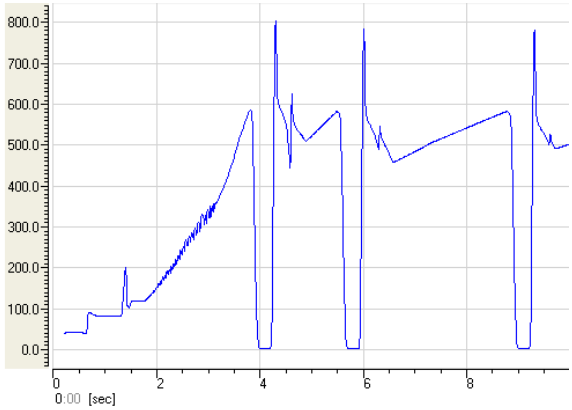


Figure 77: Mass air flow

The curves of the throttle angle look very akin to the curves of the accelerator pedal position. This is due to the fact that the throttle position is directly influenced by the latter. The value range of the throttle angle is between 0° and 80°. The mass air flow depends on the position of the throttle and is measured in kg/h. When the throttle has a wider opening angle, more air can flow through the intake system and thus the measured mass air flow is higher. Due to the dynamics of the air in the intake system, the air flow increases slower than the throttle opens.

Figure 78 depicts the sequence in which the engine requests injection time and ignition time parameters for the combustion processes in the single cylinders.

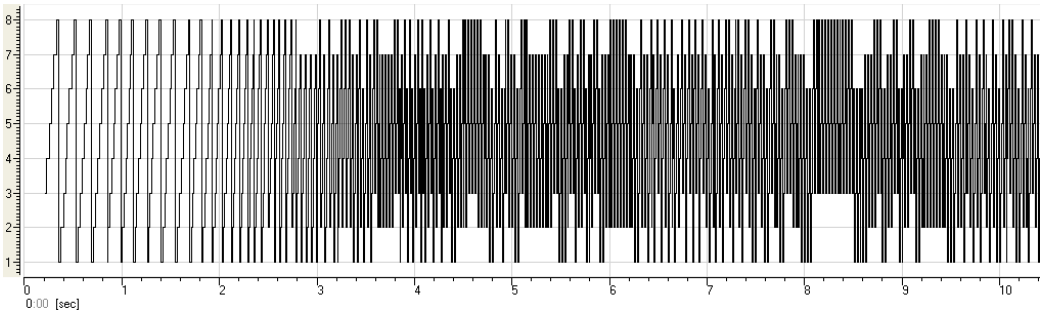


Figure 78: Cylinder-specific requests for injection time and ignition time parameters (cylinder number)

As can be seen in the figure, the eight cylinders of the engine perform requests in a sequential order, i.e., starting from the first cylinder to the 8th cylinder and then starting over again. The combustion processes are thus arranged as described in section 2.2 (see figure 2).

With increasing engine speed, the temporal distance between two cylinder-specific requests decreases. The relation between the engine speed and the temporal distance between two consecutive combustion processes has been explained in section 2.2. This, however, can only be seen from the density of the curve.

Figure 79 shows a magnified excerpt from the previous figure (time between 0 and 2s). The sequence of cylinder-specific requests can clearly be identified.

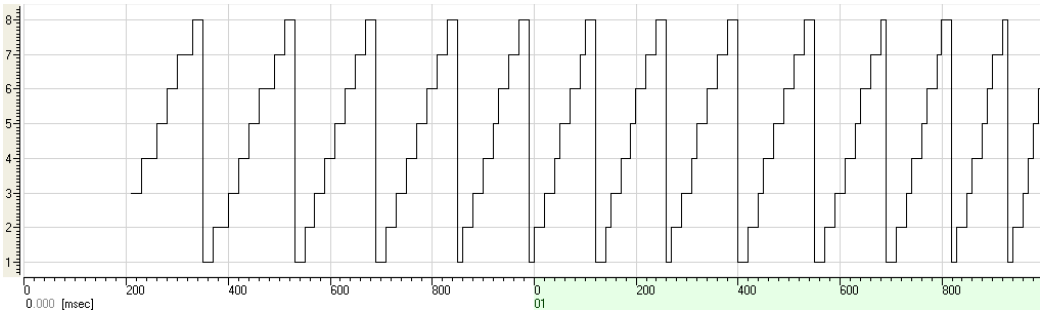
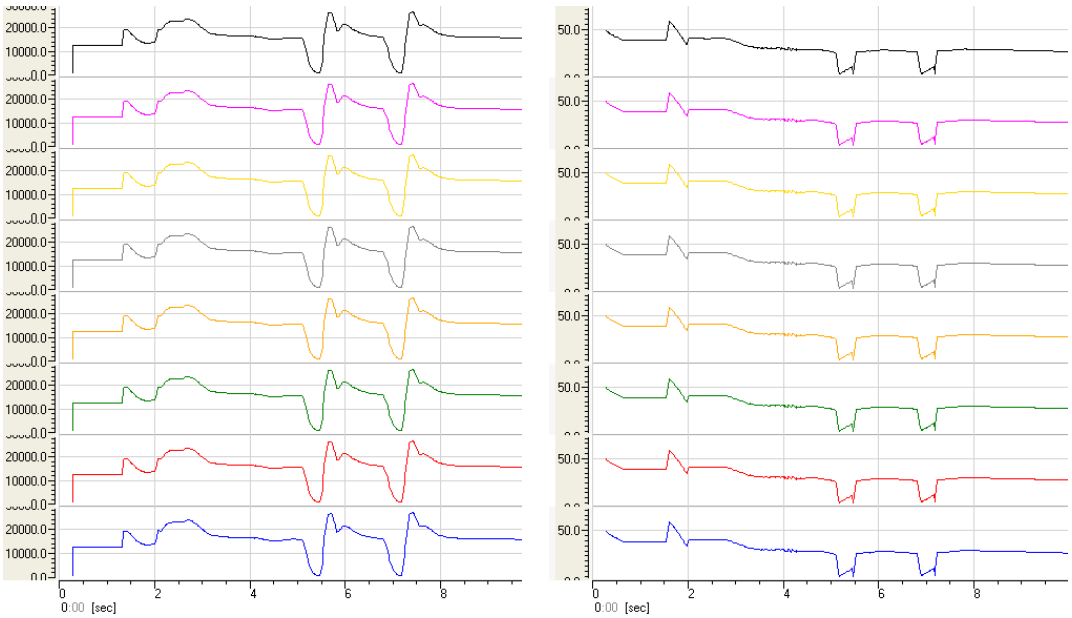


Figure 79: Cylinder-specific requests for injection time and ignition time parameters (cylinder number) (magnified excerpt)

6.3.2 Development of Output Signals from Engine Control Application

The following figures depict the development of the output signals calculated by the engine control application as they are delivered to the simulated plant. The output signals are based on the input signals that were delivered to the engine control application.

Figure 80(a) and 80(b) show the development of the injection times and ignition times of the distinct cylinders.



(a) Injection times (in μs)

(b) Ignition times (in °crankshaft (°CA) relative to basic angle of 54°CA from TDC)

Figure 80: Injection times and ignition times

The calculated injection time values are within a plausible value range between 3.5ms and 20ms [13, page 153]. The peak value is reached shortly after gear shift process.

Figure 81 shows the development of the desired throttle position that is determined by the engine control application. It is based on the current throttle position and the accelerator pedal position.

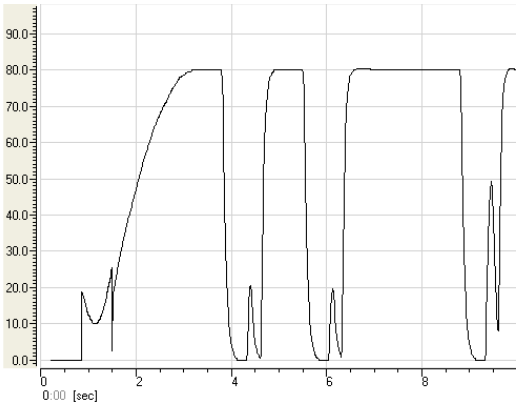


Figure 81: Desired throttle position

6.3.3 Summary

As can be seen from the figures with the development of the input and output signals, the engine control application operates as intended (i.e., the values are all plausible). The engine control application is capable of controlling the combustion processes in the cylinders as desired. The HiL setup can thus be used to perform RTE Tracing experiments and to obtain meaningful timing data.

7 Results from RTE Tracing Experiments

7.1 Introduction

In order to evaluate the degree of fulfillment of the timing requirements, an RTE Tracing experiment has been conducted based on the setup described in section 6.

In the RTE Tracing experiment, the AUTOSAR-compliant engine control application is operated in closed-loop against the simulated engine model on the real-time simulation hardware. A multitude of ObservableEventInstances have been captured through the RTE Tracing instrumentation and uploaded with the help of the logic analyzer that is connected to the target hardware of the AUTOSAR-compliant engine control application. These ObservableEventOccurrences have then been analyzed with the help of the

path specifications, and the timing properties of the different paths have been determined.

7.2 Limitations of the RTE Tracing experiment

There are two limitations that should be mentioned with respect to the conducted RTE Tracing experiment.

- In order to ease the interpretation of the timing properties that depend on the speed of the engine, a constant engine speed of 2000rpm has been chosen. This has the effect that the inter-arrival rate of cylinder-specific requests for injection time and ignition time parameters is fixed. This allows the determination of concrete values for the engine-speed dependent timing requirements.

- Due to the limited amount of memory reserved for capturing timing data on the target hardware, and due to the fact that the software execution on the target hardware must be stopped for the upload of the timing data to the host PC, consistent timing data can only be captured for a time frame of approximately 800ms for our AUTOSAR-compliant engine control application. Timing data captured over longer time frames result from multiple independent uploads where the target hardware has been stopped and restarted in between. This has lead to discontinuities in the captured timing data which makes the data inconsistent for our analyses. For our analyses, we thus focus on consistent sets of data only. This has lead to a considered time frame of approximately 800ms.

In the following, the timing properties that have been determined based on the captured ObservableEventInstances for the signal paths of the different functionalities are presented. Furthermore, the degree of fulfillment of the timing properties is discussed.

7.3 Timing Properties of the Air System

As described in section 3.3.1, the air system contains four signal paths that are of interest.

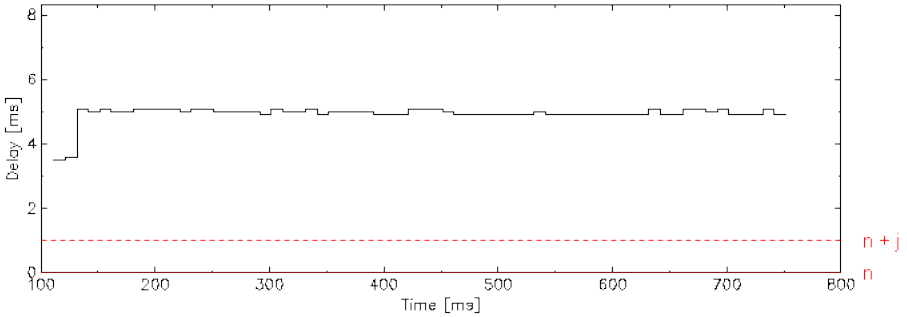
These are the signal paths from the input signals delivered by the throttle sensors and the accelerator pedal sensors, respectively, to the output signal representing the desired throttle position for the throttle actuator.

7.3.1 Timing Properties for Signal Paths from AcceleratorPedalPosition[1/2] to DesiredThrottlePosition

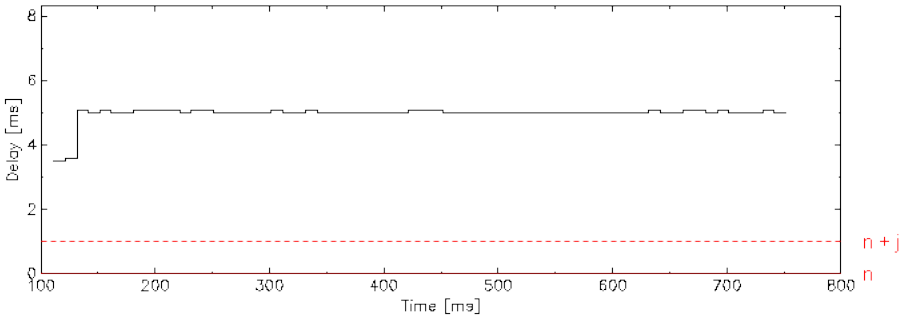
There are two signal paths from the input signals delivered by the accelerator pedal sensors to the output signal representing the calculated desired throttle position for the throttle actuator.

Path Delays

Figure 82 depicts the Timing Oscilloscope Diagrams for the PathDelays from AcceleratorPedalPosition1 and AcceleratorPedalPosition1, respectively, to DesiredThrottlePosition.



(a) PathDelays from AcceleratorPedalPosition1 to DesiredThrottlePosition



(b) PathDelays from AcceleratorPedalPosition2 to DesiredThrottlePosition

Figure 82: Timing Oscilloscope Diagrams for PathDelays from AcceleratorPedalPosition1 and AcceleratorPedalPosition2 to DesiredThrottlePosition

The timing requirements that are formulated towards the signal paths express that the path delay should be minimized to 0ms whereby a deviation of 1ms is acceptable (PathDelayRequirement with nominal value of 0ms and jitter value of 1ms).

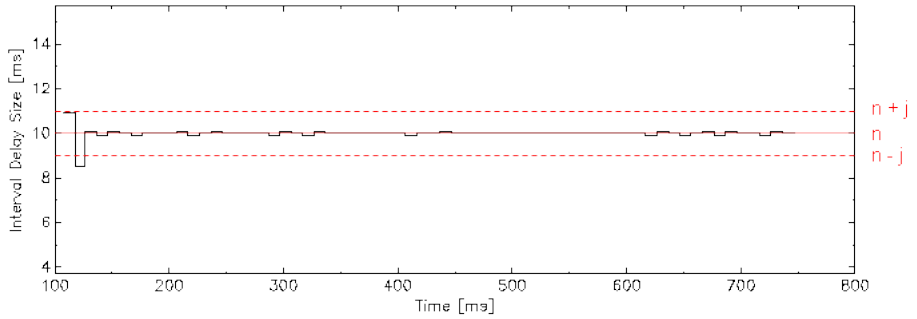
The following observations can be made:

- The PathDelays vary around a mean value of 5ms.
- There is a sudden rise of the PathDelays from 3.5ms to 5ms at the beginning.
- The PathDelayRequirements are **not** satisfied.

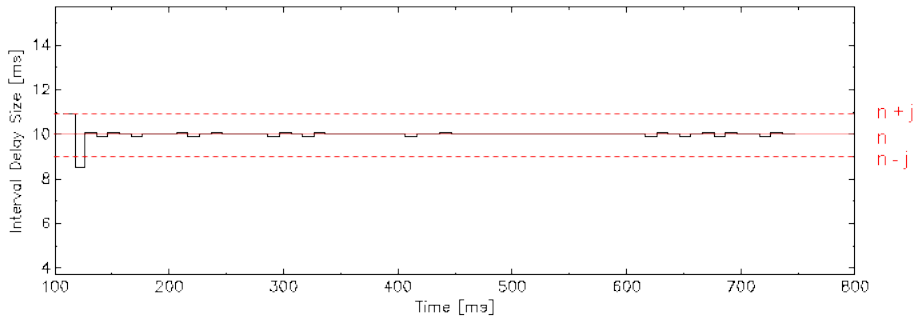
The rationale for the PathDelayRequirements not being satisfied cannot be seen from the Timing Oscilloscope Diagrams.

Input Interval Delays

Figure 83 depicts the Timing Oscilloscope Diagrams with the InputIntervalDelays for the signal paths from AcceleratorPedalPosition1 and AcceleratorPedalPosition2 to DesiredThrottlePosition.



(a) InputIntervalDelays for signal path from AcceleratorPedalPosition1 to DesiredThrottlePosition



(b) InputIntervalDelays for signal path from AcceleratorPedalPosition2 to DesiredThrottlePosition

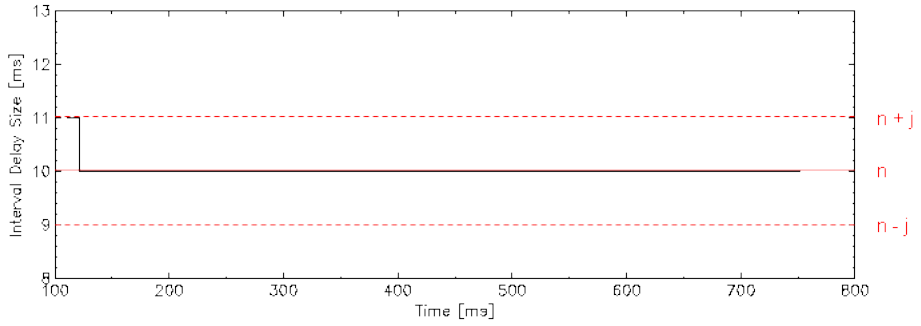
Figure 83: Timing Oscilloscope Diagrams for InputIntervalDelays

The following observations can be made:

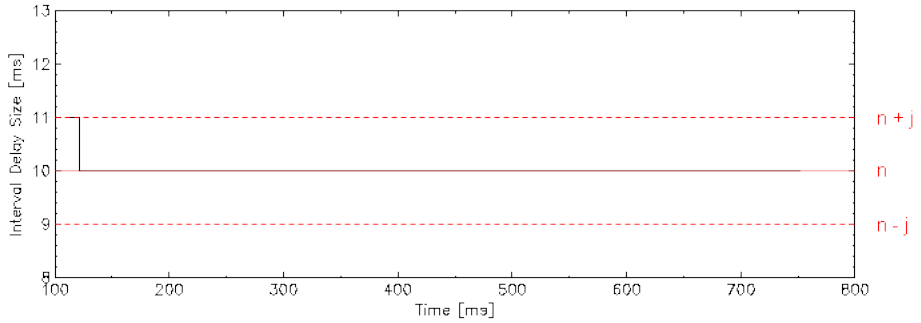
- The InputIntervalDelays vary around a mean value of 10ms, meaning that the effective sampling rate is 10ms.
- There is a temporary deviation of the InputIntervalDelays from the mean at the beginning. The InputIntervalDelays start at 10.9ms, then fall to 8.5ms until they settle at around 10ms. Although the InputIntervalDelayRequirement is formally violated, the deviation is tolerable as it only spans over two effective samples.

Output Interval Delays

Figure 84 depicts the Timing Oscilloscope Diagrams with the OutputIntervalDelays for the signal paths from AcceleratorPedalPosition1 and AcceleratorPedalPosition2 to DesiredThrottlePosition.



(a) InputIntervalDelays for signal path from AcceleratorPedalPosition1 to DesiredThrottlePosition



(b) InputIntervalDelays for signal path from AcceleratorPedalPosition2 to DesiredThrottlePosition

Figure 84: Timing Oscilloscope Diagrams for OutputIntervalDelays

The following observations can be made:

- The OutputIntervalDelays are constant at 10ms.
- There is a temporary deviation of the OutputIntervalDelays at the beginning of the measurement. The OutputIntervalDelays start at 11ms and then settle at 10ms. The temporary deviation is within the tolerance range of the OutputIntervalDelayRequirement.
- The OutputIntervalDelayRequirements are satisfied.

Input Synchronization Intervals

Figure 85 depicts the Timing Oscilloscope Diagram for the InputSynchronizationIntervals for the two join path segments.

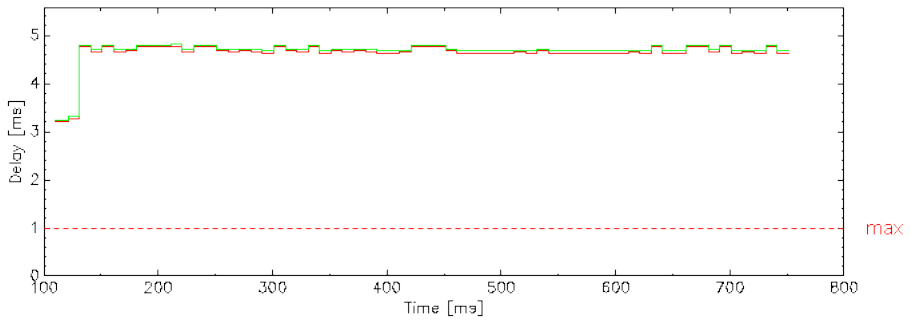


Figure 85: Latency of join path segments from input signals AcceleratorPedalPosition[1/2] to common intermediate signal VotedPedalPosition

The following observations can be made:

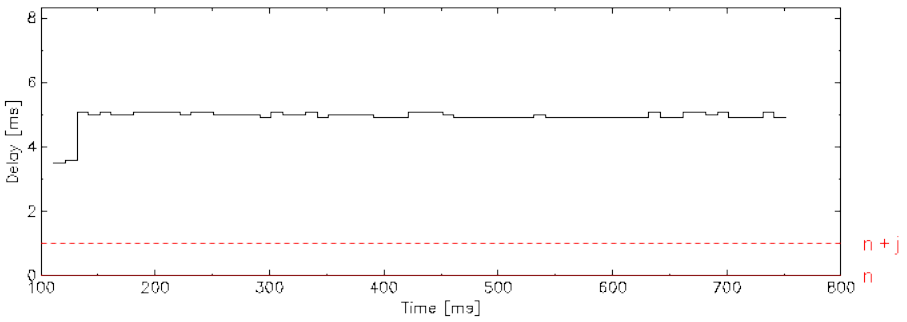
- The input signals AcceleratorPedalPosition1 and AcceleratorPedalPosition2 are closely synchronized as the curves for the path delays of the respective join path segments in the Timing Oscilloscope Diagram are close to each other.
- The path delays of the join path segments are constantly over the demanded size for the input synchronization interval. Thus, the timing requirement is formally violated.

Der Grund für die Verletzung des Timing Requirements liegt daran, dass die Latenz entlang der einzelnen Join-Path-Segments zu lang ist und nicht dass die Signale unsynchron verarbeitet werden.

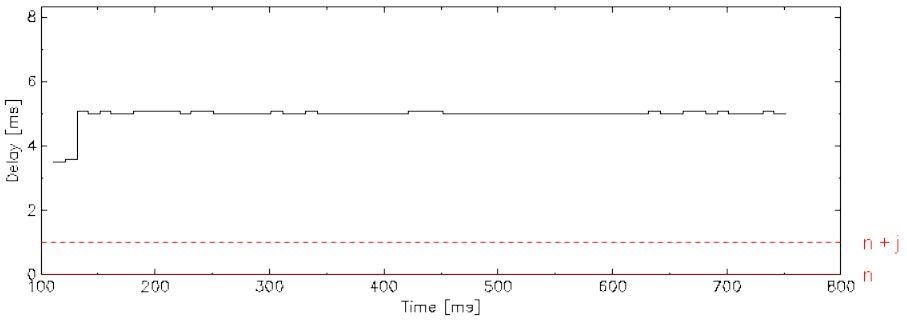
7.3.2 Timing Properties for Signal Path from ThrottlePosition[1/2] to DesiredThrottlePosition

Path Delays

Figure 86 depicts the Timing Oscilloscope Diagrams for the PathDelays from ThrottlePosition1 and ThrottlePosition2, respectively, to DesiredThrottlePosition.



(a) PathDelays from ThrottlePosition1 to DesiredThrottlePosition

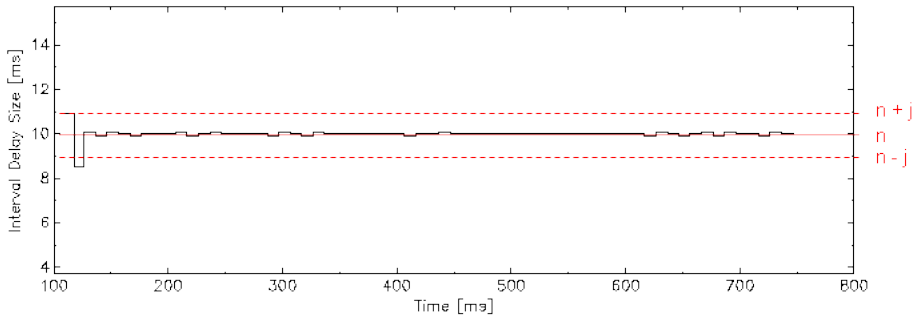


(b) PathDelays from ThrottlePosition2 to DesiredThrottlePosition

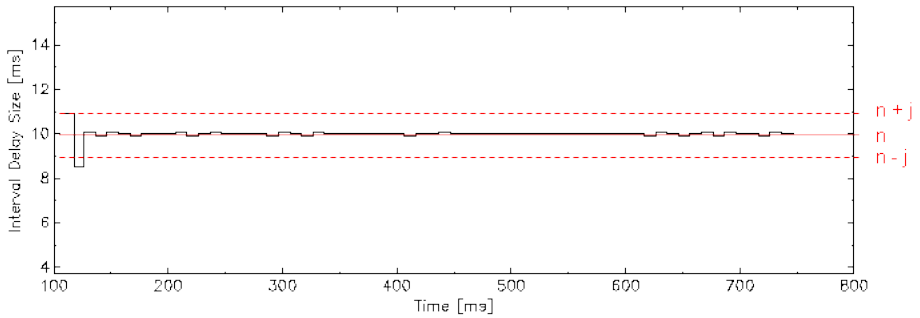
Figure 86: Timing Oscilloscope Diagrams for PathDelays from ThrottlePosition1 and ThrottlePosition2 to DesiredThrottlePosition

Input Interval Delays

Figure 87 depicts the Timing Oscilloscope Diagrams with the InputIntervalDelays for the signal paths from ThrottlePosition1 and ThrottlePosition2 to DesiredThrottlePosition.



(a) InputIntervalDelays for signal path from ThrottlePosition1 to DesiredThrottlePosition

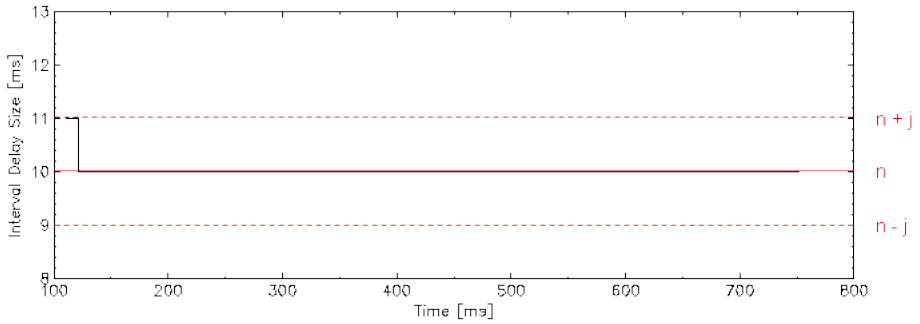


(b) InputIntervalDelays for signal path from ThrottlePosition2 to DesiredThrottlePosition

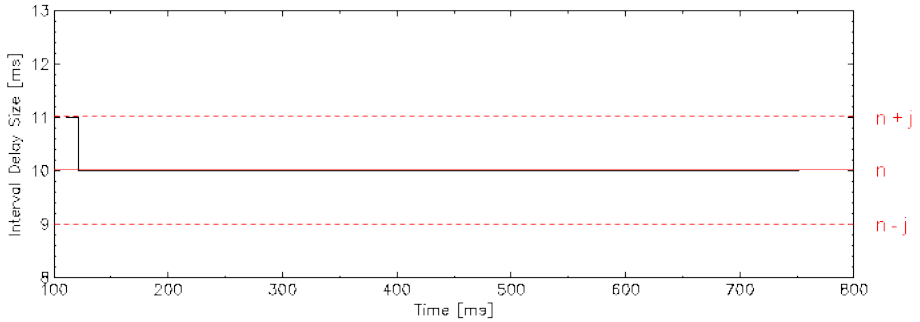
Figure 87: Timing Oscilloscope Diagrams for InputIntervalDelays for the signal paths from ThrottlePosition1 and ThrottlePosition2 to DesiredThrottlePosition

Output Interval Delays

Figure 88 depicts the Timing Oscilloscope Diagrams with the OutputIntervalDelays for the signal paths from ThrottlePosition1 and ThrottlePosition2 to DesiredThrottlePosition.



(a) InputIntervalDelays for signal path from AcceleratorPedalPosition1 to DesiredThrottlePosition



(b) InputIntervalDelays for signal path from AcceleratorPedalPosition2 to DesiredThrottlePosition

Figure 88: Timing Oscilloscope Diagrams for OutputIntervalDelays for the signal paths from ThrottlePosition1 and ThrottlePosition2 to DesiredThrottlePosition

Input Synchronization Intervals Delays

Figure 89 depicts the Timing Oscilloscope Diagram for the InputSynchronizationIntervals for the two join path segments.

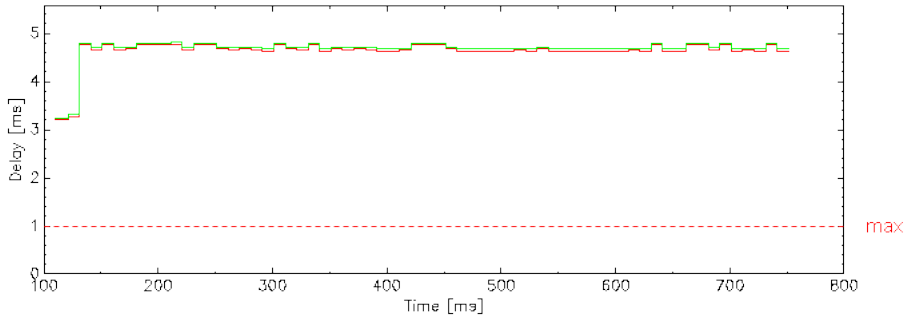


Figure 89: Latency of join path segments from input signals ThrottlePosition[1/2] to common intermediate signal VotedPedalPosition

7.4 Timing Properties of Fueling System

For the injection time (or fuel mass per stroke) calculation, timing requirements in the form of a PathDelayRequirement, an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement have been formulated. In the following, the corresponding timing properties, i.e., PathDelays, InputIntervalDelays and OutputIntervalDelays, that have been determined are described.

7.4.1 Path Delays

Figure 90 depicts the Timing Oscilloscope Diagrams with the PathDelays of the injection time calculation.
The following observations can be made:

- The PathDelays vary irregularly between a minimum and a maximum value.
- The PathDelayRequirement is satisfied

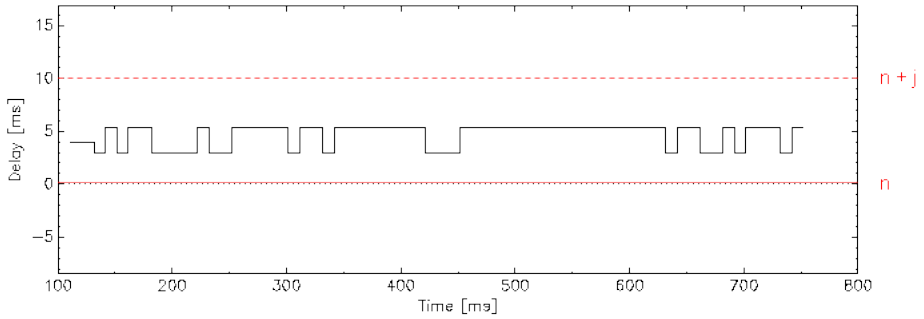


Figure 90: Timing Oscilloscope Diagram for PathDelays of injection time calculation

7.4.2 Input Interval Delays

Figure 91 depicts the Timing Oscilloscope Diagrams with the InputIntervalDelays of the injection time calculation.

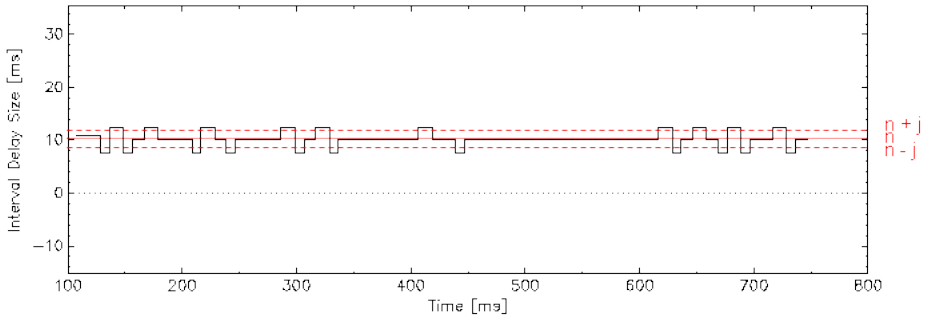


Figure 91: Timing Oscilloscope Diagram for InputIntervalDelays

The following observations can be made:

- The InputIntervalDelays vary irregularly between a minimum and a maximum value and around a mean value.
- The InputIntervalDelayRequirement is violated.

7.4.3 Output Interval Delays

Figure 92 depicts the Timing Oscilloscope Diagrams with the OutputIntervalDelays of the injection time calculation.

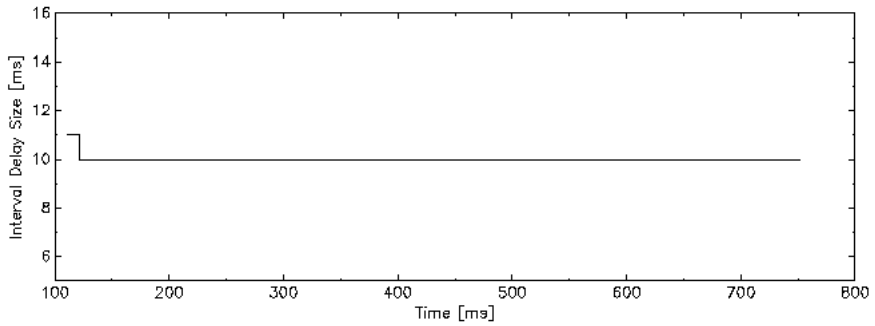


Figure 92: Timing Oscilloscope Diagram for OutputIntervalDelays

The following observations can be made:

- The OutputIntervalDelays are constant at 10ms after a temporary deviation at the beginning.
- The OutputIntervalDelayRequirement is satisfied

7.5 Timing Properties of Ignition System

For the ignition time calculation, timing requirements in the form of a PathDelayRequirement, an InputIntervalDelayRequirement and an OutputIntervalDelayRequirement have been formulated. In the following, the corresponding timing properties, i.e., PathDelays, InputIntervalDelays and OutputIntervalDelays, that have been determined are described.

7.5.1 Path Delays

Figure 93 depicts the Timing Oscilloscope Diagrams with the PathDelays of the ignition time calculation.
The following observations can be made:

- The PathDelays vary irregularly between a minimum and a maximum value.
- The PathDelayRequirement is satisfied

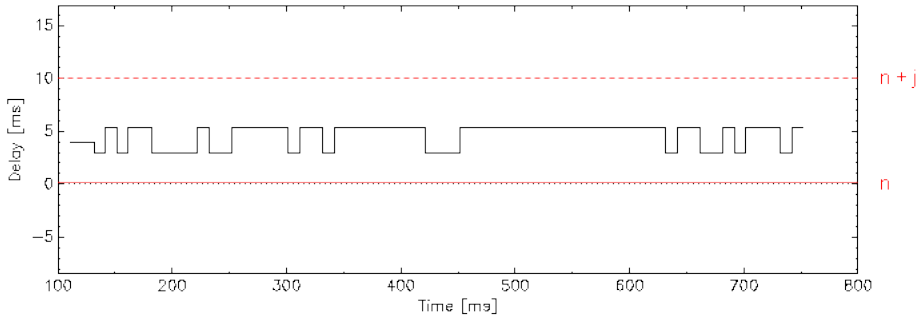


Figure 93: Timing Oscilloscope Diagram for PathDelays

7.5.2 Input Interval Delays

Figure 94 depicts the Timing Oscilloscope Diagrams with the InputIntervalDelays of the ignition time calculation.

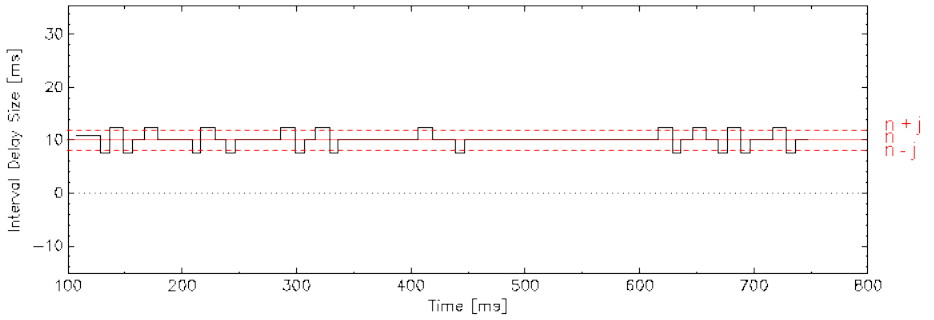


Figure 94: Timing Oscilloscope Diagram for InputIntervalDelays

The following observations can be made:

- The InputIntervalDelays vary irregularly between a minimum and a maximum value and around a mean value.
- The InputIntervalDelayRequirement is satisfied

7.5.3 Output Interval Delays

Figure 95 depicts the Timing Oscilloscope Diagrams with the OutputIntervalDelays of the ignition time calculation.

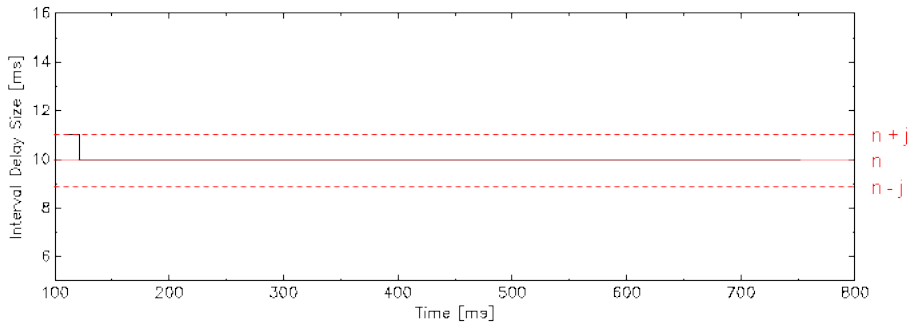


Figure 95: Timing Oscilloscope Diagram for OutputIntervalDelays

The following observations can be made:

- The OutputIntervalDelays are constant at 10ms after a temporary deviation at the beginning.
- The OutputIntervalDelayRequirement is satisfied

7.6 Timing Properties of Injection Time and Ignition Time Actuation System

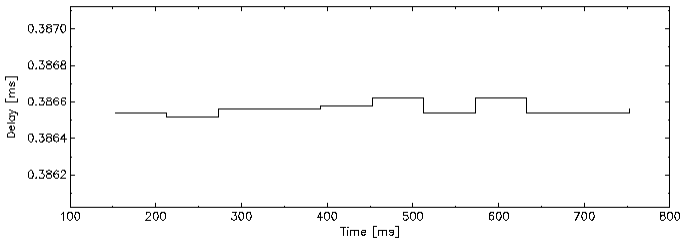
For the actuation of the calculated injection time and ignition time parameters upon a cylinder-specific request, timing requirements in the form of ReactionTimeRequirements have been formulated. In the following, the corresponding timing properties, i.e., the ReactionTimes, that have been determined are described.

Furthermore, InputIntervalDelays and OutputIntervalDelays have also been determined. These are interpreted as latencies between consecutive effective injection time/ignition time requests (consecutive effective stimuli) and consecutive effective injection time/ignition time actuations (consecutive effective responses), respectively.

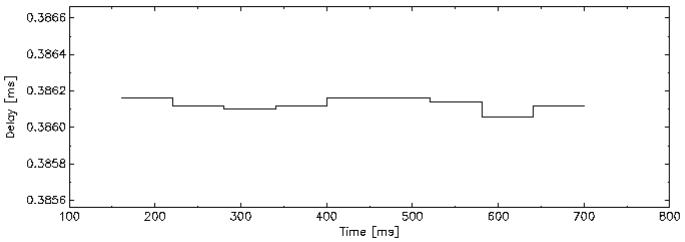
7.6.1 Injection Time Actuation

Reaction times for injection time actuations

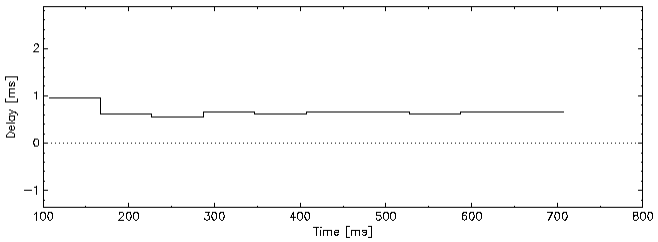
Figure 96 depicts the ReactionTimes for the cylinder-specific injection time actuations.



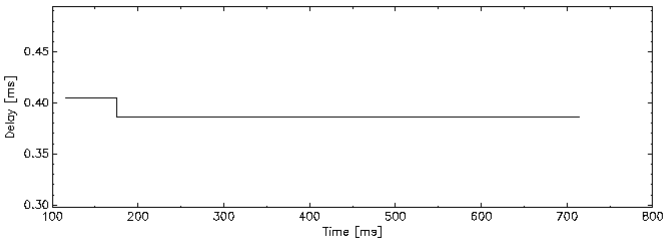
(a) Cylinder number 1



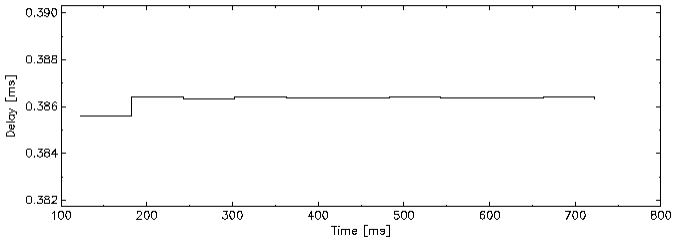
(b) Cylinder number 2



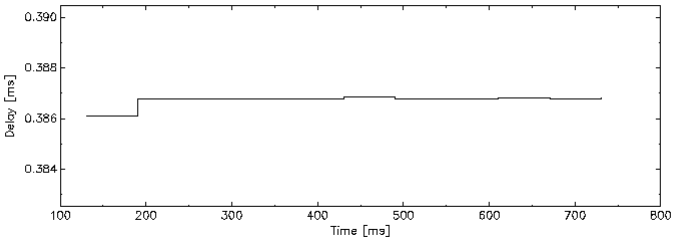
(c) Cylinder number 3



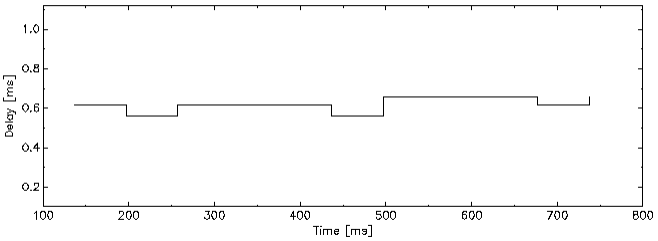
(d) Cylinder number 4



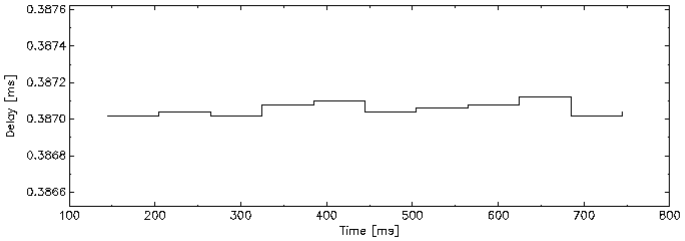
(e) Cylinder number 5



(f) Cylinder number 6



(g) Cylinder number 7

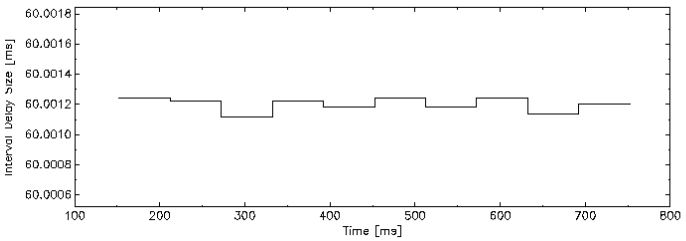


(h) Cylinder number 8

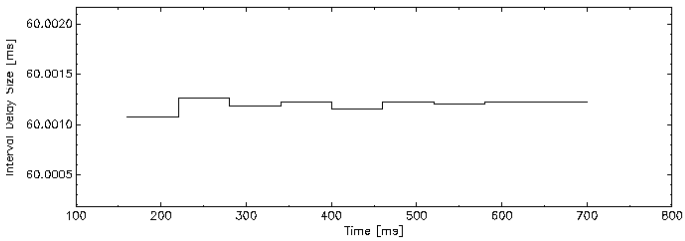
Figure 96: Timing Oscilloscope Diagrams for ReactionTimes of cylinder-specific injection time actuations

Latencies between consecutive effective injection time actuation requests

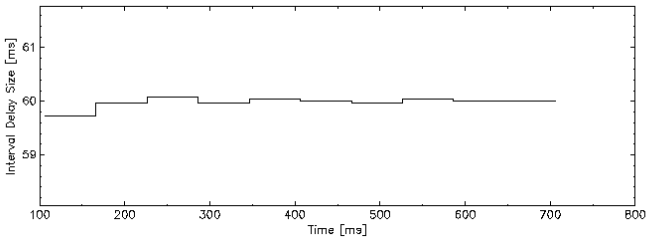
Figure 97 depicts the latencies between consecutive effective injection time requests (consecutive effective stimuli).



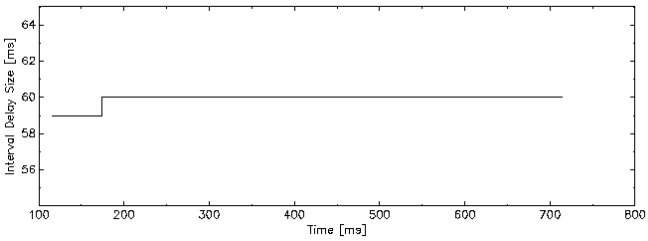
(a) Cylinder number 1



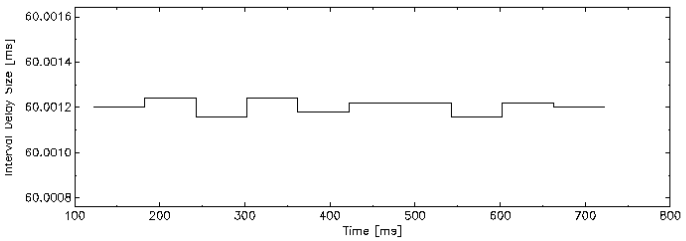
(b) Cylinder number 2



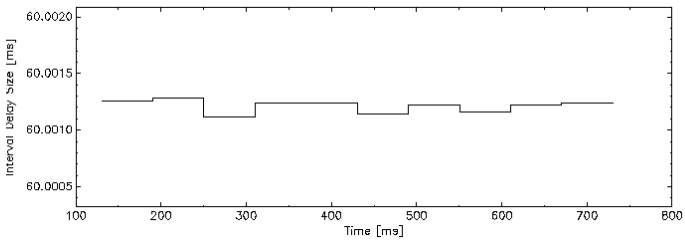
(c) Cylinder number 3



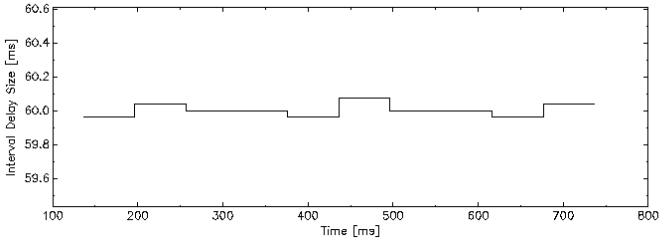
(d) Cylinder number 4



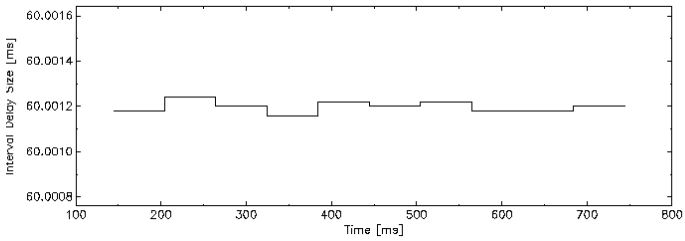
(e) Cylinder number 5



(f) Cylinder number 6



(g) Cylinder number 7

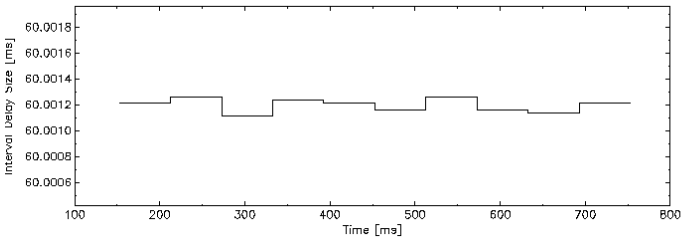


(h) Cylinder number 8

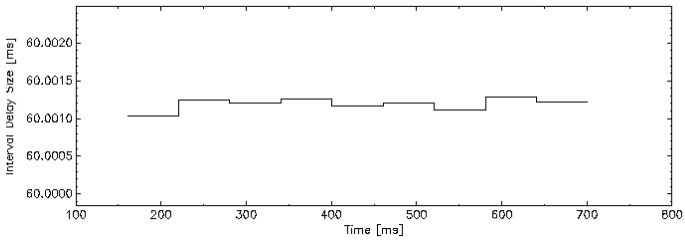
Figure 97: Timing Oscilloscope Diagrams for latencies between consecutive effective injection time requests

Latencies between consecutive effective injection time actuations

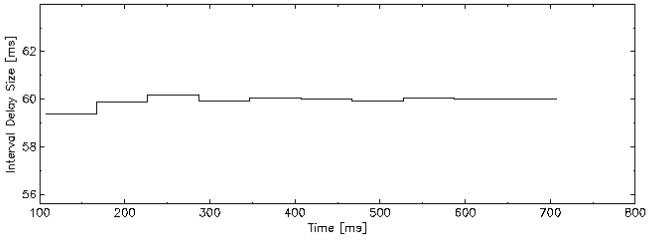
Figure 98 depicts the latencies between consecutive effective injection time actuations (consecutive effective responses).



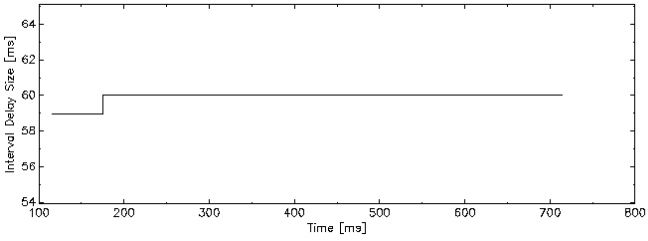
(a) Cylinder number 1



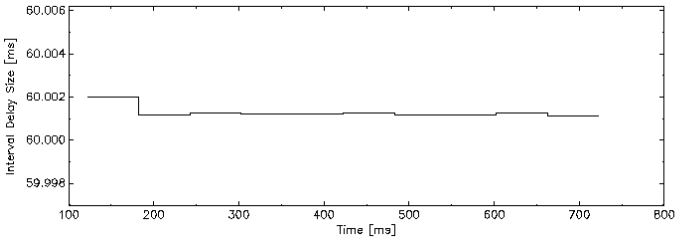
(b) Cylinder number 2



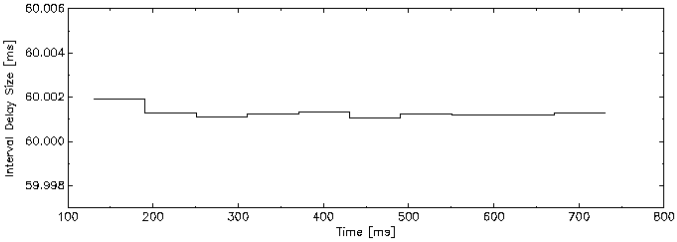
(c) Cylinder number 3



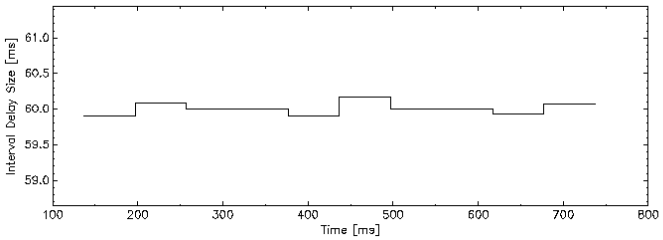
(d) Cylinder number 4



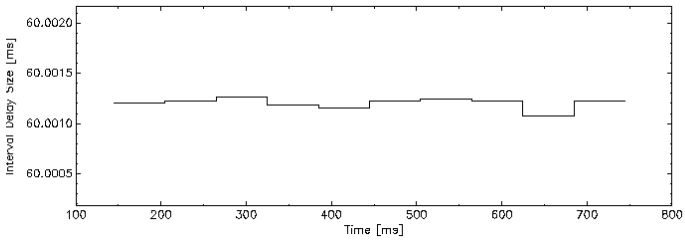
(e) Cylinder number 5



(f) Cylinder number 6



(g) Cylinder number 7



(h) Cylinder number 8

Figure 98: Timing Oscilloscope Diagrams for latencies between consecutive effective injection time actuations

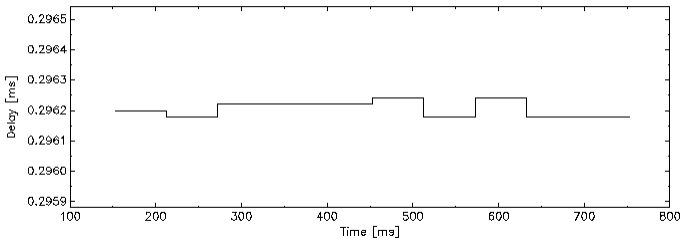
The following observations can be made and conclusions can be drawn:

- The latencies between consecutive effective injection time requests are approximately 60ms for all cylinders. This corresponds to an engine speed of 2000ms. This is the engine speed at which the RTE Tracing experiments have been conducted. It can be concluded that there are no misses in the injection time calculations.
- The latencies between consecutive effective injection time actuations are approximately 60ms for all cylinders. This again corresponds to an engine speed of 2000ms, the engine speed at which the RTE Tracing experiments have been conducted. From the plots it can be concluded that there are no misses in the injection time actuations.

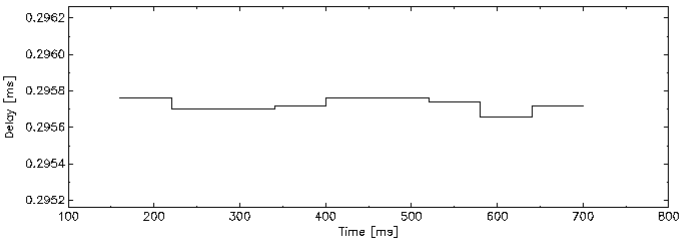
7.6.2 Ignition Time Actuation

Reaction times for ignition time actuations

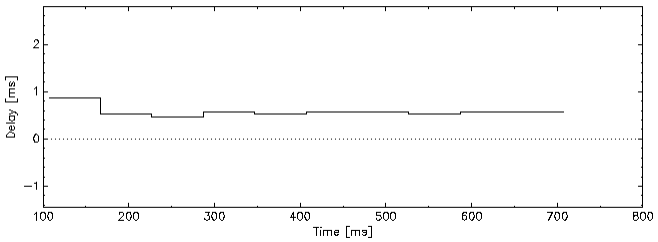
Figure 99 depicts the ReactionTimes for the cylinder-specific ignition time actuations.



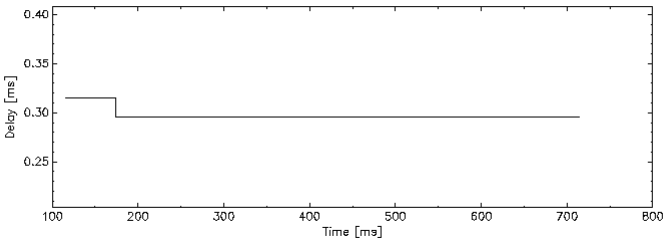
(a) Cylinder number 1



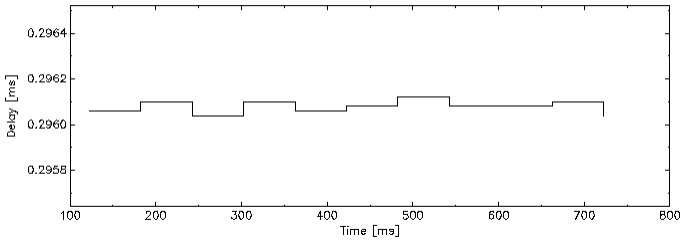
(b) Cylinder number 2



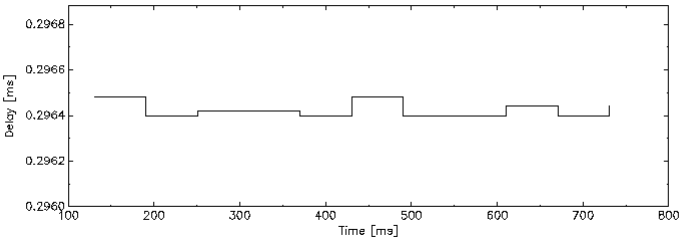
(c) Cylinder number 3



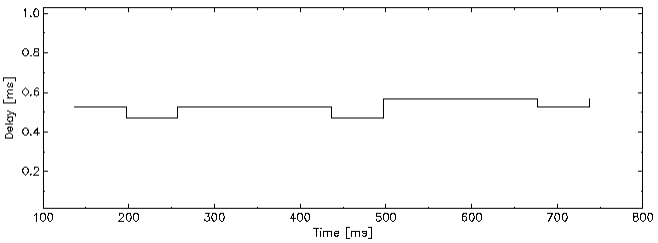
(d) Cylinder number 4



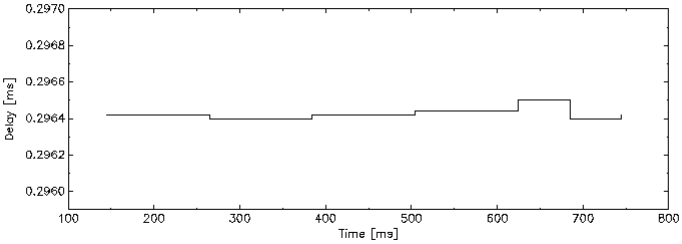
(e) Cylinder number 5



(f) Cylinder number 6



(g) Cylinder number 7

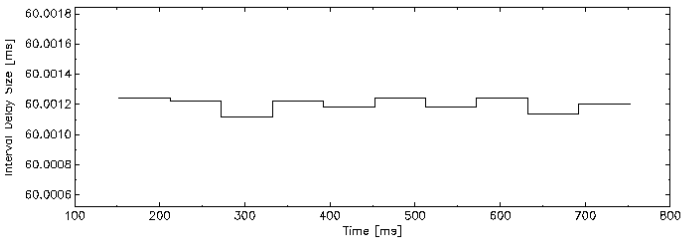


(h) Cylinder number 8

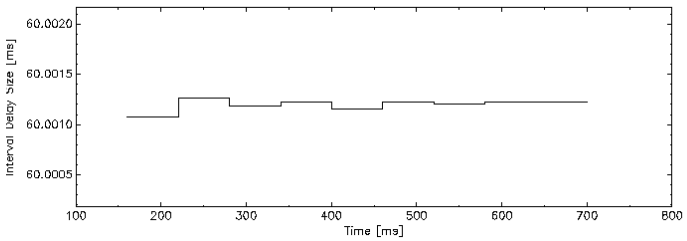
Figure 99: Timing Oscilloscope Diagrams for ReactionTimes of cylinder-specific ignition time actuations

Latencies between consecutive effective ignition time actuation requests

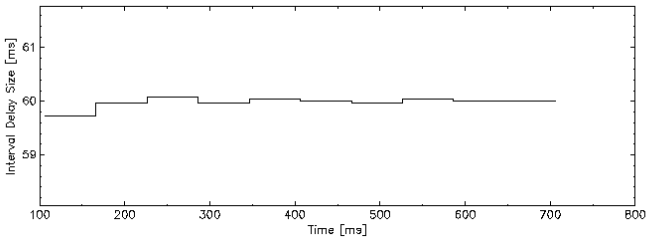
Figure 100 depicts the latencies between consecutive effective ignition time requests (consecutive effective stimuli).



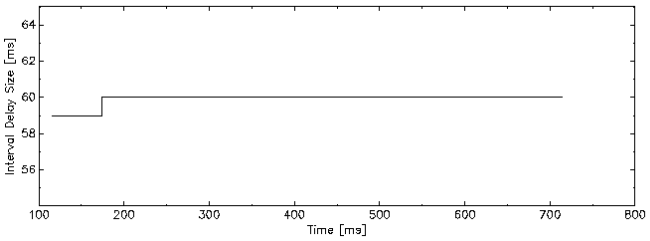
(a) Cylinder number 1



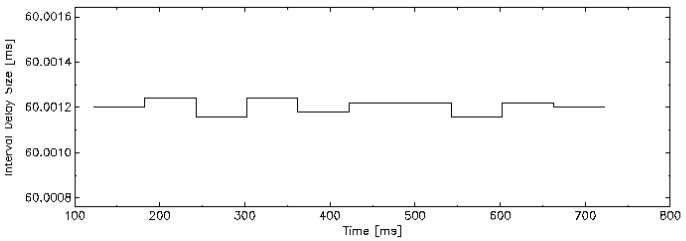
(b) Cylinder number 2



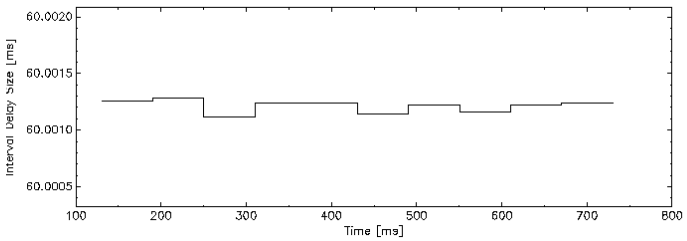
(c) Cylinder number 3



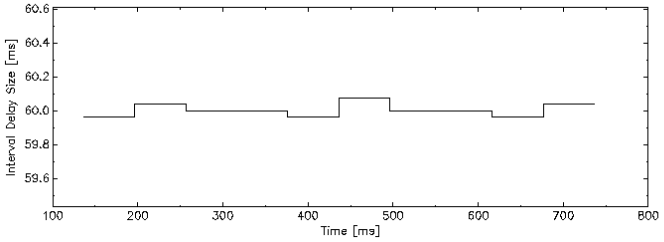
(d) Cylinder number 4



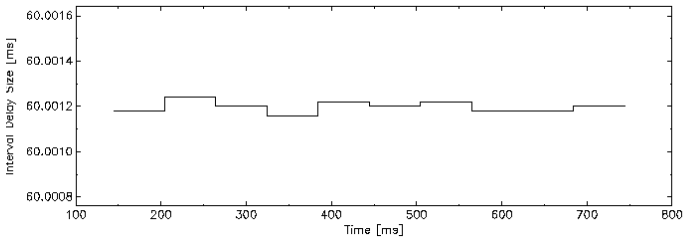
(e) Cylinder number 5



(f) Cylinder number 6



(g) Cylinder number 7

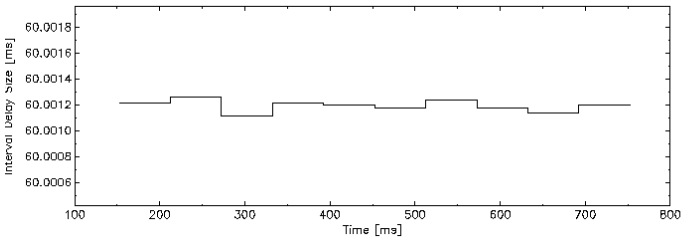


(h) Cylinder number 8

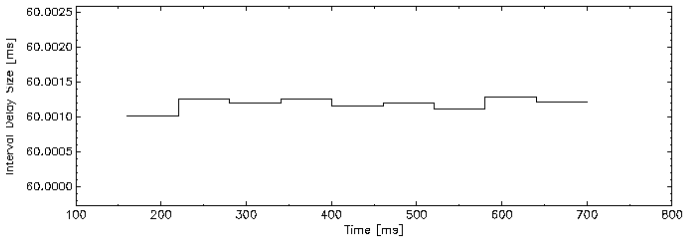
Figure 100: Timing Oscilloscope Diagrams for latencies between consecutive effective ignition time requests

Latencies between consecutive effective ignition time actuations

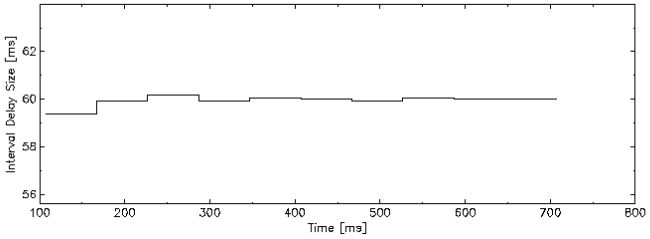
Figure 101 depicts the latencies between consecutive effective ignition time actuations (consecutive effective responses).



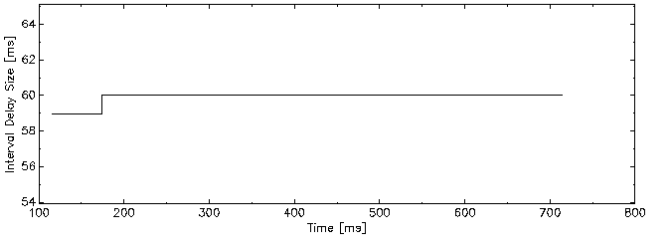
(a) Cylinder number 1



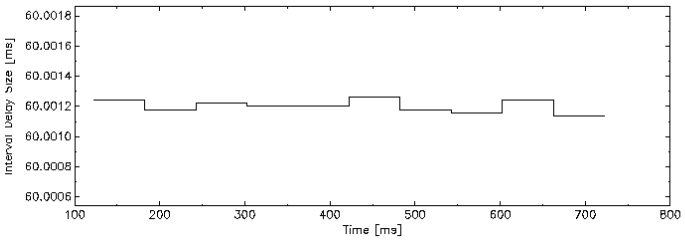
(b) Cylinder number 2



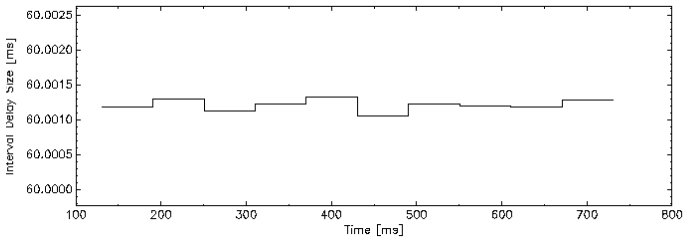
(c) Cylinder number 3



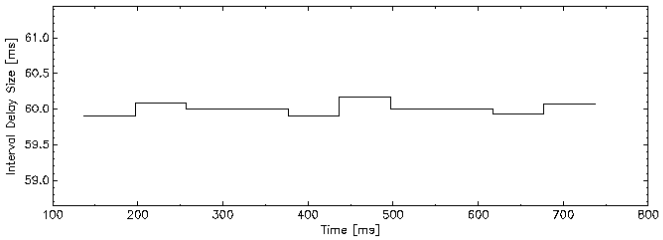
(d) Cylinder number 4



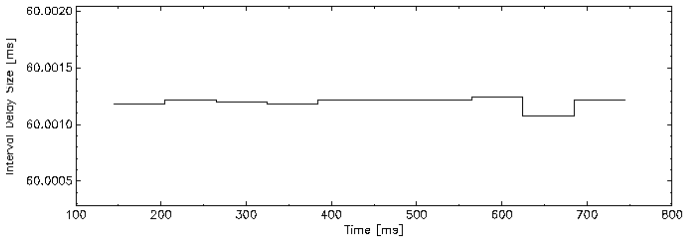
(e) Cylinder number 5



(f) Cylinder number 6



(g) Cylinder number 7



(h) Cylinder number 8

Figure 101: Timing Oscilloscope Diagrams for latencies between consecutive effective ignition time actuations

7.7 Summary and Conclusion

In this section, the results from the RTE Tracing experiments with the AUTOSAR-compliant engine control application have been presented. The determined timing properties for the signal paths of the air system, fueling system, ignition system and the injection time and ignition time actuation system have been shown by means of Timing Oscilloscope Diagrams. It has been shown that the timing requirements that have been formulated towards these functionalities of the engine control application are only partly satisfied. The reason for non-satisfied timing requirements lies in the coupling of the AUTOSAR-compliant engine control application and the simulation hardware with the engine simulation via a CAN bus. The CAN bus is inadequate for a constant provision of input data with low delays.

The RTE Tracing experiments have been conducted for a scenario with a constant engine speed of 2000rpm. This has allowed to evaluate the correctness of consecutive effective injection time and ignition time actuations. For other engine speeds, similar results can be obtained.

8 Summary

This report has presented the case study of an engine control application to which the concepts of our Timing Model for AUTOSAR and the RTE Tracing approach have been applied [4]. Section 2 gave a brief introduction into the working principles of internal combustion engines and the requirements towards an engine control application for controlling the combustion processes in the cylinders of an engine.

Section 3 then gave a more detailed overview of the basic functionalities of the engine control application under consideration as case study object. This also included a description of the most relevant signal paths for the different functionalities as well as the timing requirements which can be associated with the signal paths.

As our case study object is based on a legacy, non-AUTOSAR-compliant engine control application to which the concepts of our Timing Model for AUTOSAR could not be directly applied, the engine control application was first reengineered to an AUTOSAR-compliant system. Section 4 described the AUTOSAR compliant software architecture of the reengineered engine control application and the important configurations for its realization as AUTOSAR-compliant single-ECU system.

Section 5 then described the application of the concepts of our Timing Model for AUTOSAR. The signal paths of the different functionalities that

were identified have been modeled by means of PathSpecifications and associated with application-specific timing requirements. The objective was then to conduct RTE Tracing experiments to determine the timing properties and to evaluate the degree of fulfillment of the timing requirements. For this, an RTE Tracing instrumentation was generated and integrated with the ECU software.

The technical setup used for RTE Tracing experiments was described in section 6. A hardware-in-the-loop (HIL) setup was designed where the relevant signals between the AUTOSAR-compliant engine control application being realized as single-ECU system and a simulated environment are exchanged over a CAN bus. During an RTE Tracing experiment, timing data (i.e., time-stamps for the occurrences of AUTOSAR-specific ObservableEvents) was captured by means of a state-of-the-art logic analyzer. The timing data was then analyzed in order to obtain the timing properties results.

Section 7 discussed the results from the conducted RTE Tracing experiments. The determined timing properties of the AUTOSAR-compliant engine control application were presented, and the degree of fulfillment of the timing requirements was discussed based on the introduced visualization means, i.e., Timing Oscilloscopes Diagrams. It could be shown that the timing requirements are only partly satisfied in the analyzed scenario from the RTE Tracing experiment. Some timing requirements could not be satisfied. This is mainly due to the asynchronous delivery of input signals via the CAN bus.

The case study shows that the concepts of our Timing Model for AUTOSAR are applicable to real-time applications realized in terms of an AUTOSAR system. Together with the AUTOSAR-specific ObservableEvents, the concepts for hierarchical event chain provide the necessary means for the description of signal paths in AUTOSAR systems such that application-specific timing requirements can adequately be expressed. By means of RTE Tracing, it is possible to determine timing properties that correspond to the application-specific timing requirements such that the degree of fulfillment of the timing requirements can be evaluate.

References

[1] Arbeitskreis E-Gas. Standardisiertes Überwachungskonzept für Motorsteuerungen von Otto- und Dieselmotoren, Version 2.0, Mai 2003.

[2] ETAS GmbH. RTA-TRACE User Guide. http://www.etas.de/, 2006.

[3] Ulrich Freund, Patrick Frey, Stefan Schimpf, and Dirk Ziegenbein. Model-based AUTOSAR Integration of an Engine Management System. *Proceedings of the AUTOREG 2008 - Steuerung und Regelung von Fahrzeugen und Motoren*, February 2008.

[4] Patrick Frey. *A Generic Timing Model for Real-Time Applications and its Application to AUTOSAR.* Dissertation, Universität Ulm, (to appear).

[5] Patrick Frey and Ulrich Freund. AUTOSAR compliant reengineering of an Engine Management System. In *Proceedings of the 4th Workshop on Object-Oriented Modeling of Embedded Real-Time Systems (OMER4)*, October 2007.

[6] Patrick Frey and Ulrich Freund. Model-Based AUTOSAR Integration of an Engine Management System. In *Proceedings of the 8th Stuttgart International Symposium "Automotive and Engine Technology"*, March 2008.

[7] Gasoline Engine Vehicle Model (GEVM) V 5.0. ETAS GmbH. http://www.etas.de/, 2005.

[8] Markus Gebhardt, G. Stier, Andy Beaumont, and A. Noble. Automation of Ecu Software Development: From Concept to Production Level Code. *SAE Technical Papers*, 1999. SAE-Paper 1999-01-1174.

[9] ETAS GmbH. Toolkette im Einsatz. *ETAS RealTimes Magazine*, 1:6–9, 1999.

[10] Hans-Herrmann Braess (Hrsg.) and Ulrich Seiffert (Hrsg.). *Handbuch Kraftfahrzeugtechnik*, volume 2. Vieweg Verlag, April 2001.

[11] Fabian May. Spezifikation von Zeitverhalten und Zeitanalyse in AUTOSAR basierten Regelungssystemen. Diplomarbeit, Institute of Computer and Communication Engineering, Department of Electrical Engineering and Information Technology, Technical University Carolina-Wilhelmina of Braunschweig, December 2008.

[12] Daniel Munzinger. AUTOSAR Softwareintegration einer Motorsteuerung. Diplomarbeit, Hochschule der Medien, Stuttgart, September 2007.

[13] Robert Bosch GmbH. *Ottomotor-Management - Systeme und Komponenten*, volume 2. Vieweg Verlag, May 2003.

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*

Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2010-01 *Hariolf Beth, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer
Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm