Testing and Monitoring of FlexRay Based Applications

Roman Pallierer, Thomas M. Galla

EB (formerly DECOMSYS)

roman.pallierer@elektrobit.com, thomas.galla@elektrobit.com

October 11, 2007

1 Introduction to FlexRay Based Applications

FlexRay has been developed for future in-car control applications demanding high data rates, deterministic behavior and the support of fault tolerance. Application domains of FlexRay-based systems include power train, chassis and body control. Furthermore, FlexRay is considered as backbone network interconnecting several main electronic control units (ECUs) with each other. This chapter is focused on testing and monitoring concepts for such applications interconnected via FlexRay.

1.1 System Architecture

When describing the system architecture a distinction between the hardware and the software aspects of the system architecture can be made. The hardware architecture shown in this chapter reflects the state of the art in today's automotive systems. The software architecture presented in this chapter conforms to the AUTOSAR standard [1] (see Chapter SV). However some simplifications are made for the sake of brevity, clarity, and simplicity (i.e., unnecessary details are sometimes omitted).

1.1.1 Hardware Architecture

The hardware architecture of automotive systems can be viewed at different levels of abstraction.

On the highest level of abstraction, the *system level*, an automotive system consists of a number of networks interconnected via gateways (see Figure 1). In general these networks correspond to the different functional domains that can be found in today's cars (i.e., chassis domain, power train domain, body domain).

The networks themselves comprise a number of electronic control units (ECUs) which are interconnected via a communication media (see zoom-in on network A and D in Figure 1). – The physical topology used for the interconnection is basically arbitrary; however bus, star,

and ring topologies are the most common topologies in today's cars. This level, named *network level,* represents the medium level of abstraction.

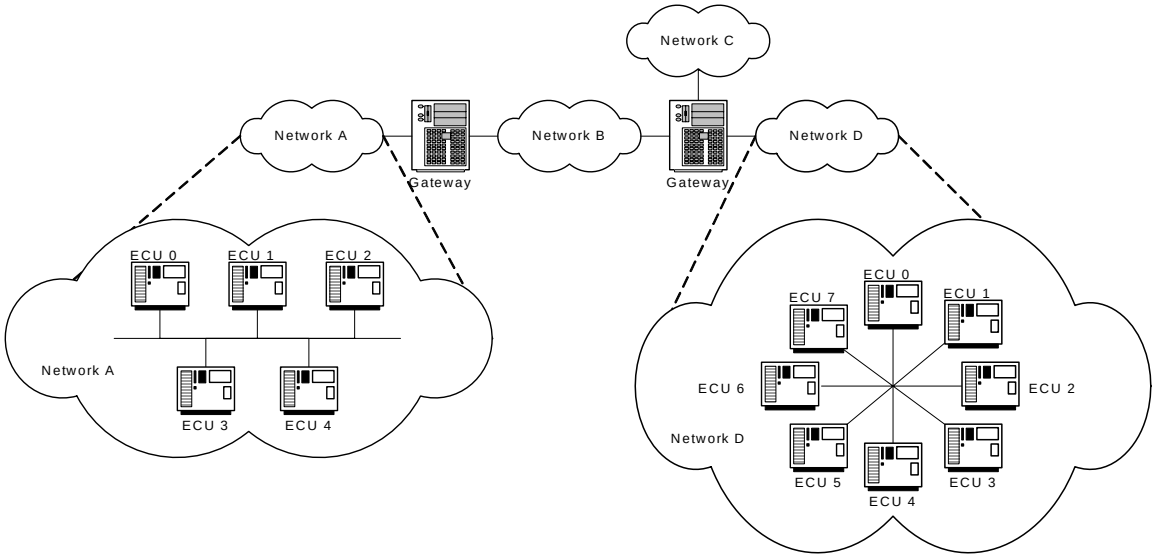

Figure 1: Hardware Architecture – System and Network Level

Note that conceptually speaking, a gateway is a special ECU that actually is a member of all networks that are interconnected by this gateway.

On the lowest level of abstraction, the *ECU level* (Figure 2), the major parts of an ECU are of interest. An ECU comprises one or more micro controller units (MCUs) as well as one or more communication controllers (CCs). In most cases, exactly one MCU and one CC are used to build up an ECU.

In order to be able to control physical processes in the car (e.g., control the injection pump of an engine) the ECU's MCU is connected to actuators via the MCU analog or digital output ports. To provide means to obtain environmental information, sensors are connected to the MCUs analog or digital input ports. We call this interface the ECU's environmental interface.

The CC (s) facilitate(s) the physical connectivity of the ECU to the respective network(s). We call this interface of an ECU the ECU's network interface. The number of CCs hosted by

gateway ECUs thus usually equals the number of networks interconnected by the respective gateway.

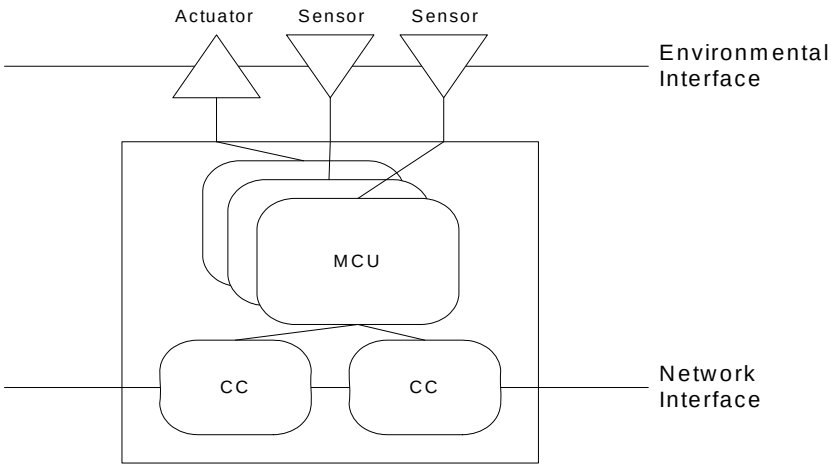

Figure 2: Hardware Architecture – ECU Level

1.1.2 Software Architecture

The AUTOSAR software architecture makes a rather strict distinction between application software and basic or system software. While the basic (or system) software provides functionality like communication protocol stacks for automotive communication protocols (e.g., FlexRay), an operating system and diagnostic modules, the application software comprises all application specific software items (i.e., control loops, interaction with sensors and actuators etc.). This way, the basic or system software provides the fundament, the application software is built upon.

1.1.2.1 Application Software

Application software in AUTOSAR consists of application software components, which are ECU and location independent and sensor-actuator components that are dependent on ECU hardware and therefore location dependent. Whereas instances of application software components can easily be deployed to and relocated among different ECUs, instances of sensor-actuator components must be deployed to a specific ECU for performance/efficiency

reasons. Deploying multiple instances of the same component to a single ECU is supported by the AUTOSAR component standard. A simple example for the deployment of multiple instances of the same component is an ECU with two redundant sensors. In that scenario two instances of the respective sensor components would be deployed to the ECU, each instance servicing exactly one of the two sensors.

Application software components as well as sensor-actuator components are interconnected via so-called connectors. These connectors represent the exchange of data, usually called "signals" in automotive domain, among the connected components. The characterics, requirements and the constraints on such a signal exchange are specified as attributes of the respective connector. – Hereby the following classes of characteristics, requirements and constraints have to be considered:

Timing characteristics and requirements: This class defines the temporal properties of the signal exchange, namely the properties occurrence, period, latency, and jitter. As far as the *occurrence* of a signal exchange is concerned, a distinction between periodic exchange, sporadic exchange and aperiodic exchange can be made. While for periodic and sporadic exchanges, constraints on the temporal distance between two consecutive signal exchanges can be specified, no such constraints can be given for aperiodic exchanges. The *period* of the signal exchange is hereby the temporal distance between two consecutive signal exchanges in case of periodic signals (P in Figure 3). In case of sporadic signals the period defines the minimum temporal distance between any two consecutive signal exchanges (sometimes also called minimum interarrival time). For aperiodic signals, the period property is not used. The *latency* (L_i in Figure 3) of a signal exchange is defined as the temporal distance between the initiation of the signal transmission at the sender (i.e., the point in time the sending application software component calls the sending API service) and the

signal reception at the receiver (i.e., the point in time the received signal is available at the receiving application software component). Given the characteristics of periodic/sporadic signals exchanged through a given network, it is possible to evaluate *a priori* the worst latency for each signal; for example, in [2] the authors have shown how to evaluate this worst case for a FlexRay network while in Chapter (see HH) in this book, is presented the evaluation method of the same characteristic for a CAN network is presented. Properties can be required on the latency of signals such as *the maximum allowable latency* (for example, L_i has to be always lower than $L_{\max}$ in Figure 3) or an imposed mean value M (for example, $\frac{\sum_i L_i}{i}$ has to be equal to M in Figure 3). The deviation of the actual observed latency of the exchange of a specific signal from the mean latency is termed *jitter*. Since minimizing the jitter is of utmost importance to ensure high quality distributed control loops, the *maximum allowable jitter* is another important attribute of the connector (for example, $abs(L_i - M)$ has to be less than a given value $J_{\max}$ in Figure 3). Note that transmission guarantee requirements (e.g., guaranteed vs. best effort transmission) can easily be expressed by means of the mean latency and the maximum allowable jitter parameter. In particular, setting the required mean latency of a connector for example to a value different from infinity and requiring that the maximum jitter is smaller than a defined value formulates a requirement for a guaranteed transmission with bounded jitter. Setting the required mean latency to infinity however and requiring a maximum jitter that is smaller than a defined value formulates a requirement for a non guaranteed transmission which however in case the transmission takes place has a bounded jitter.

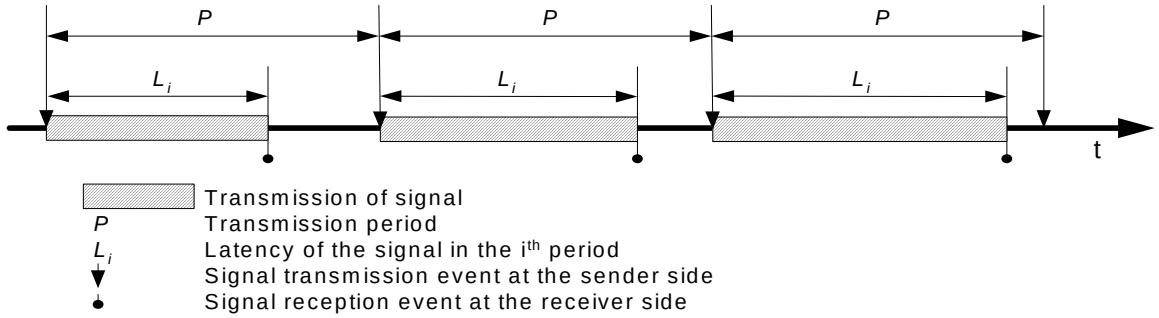

Figure 3: Characteristics for a Periodic Signal Exchange

Fault-tolerance requirements: This class defines the fault-tolerance properties of the signal exchange, namely the properties redundancy type, redundancy degree, and additional parameters for a certain redundancy type. As far as the *redundancy type* is concerned a distinction between spatial redundancy (i.e., signal exchange via multiple physical communication channels) and temporal redundancy (i.e., performing the signal exchange multiple times with the same signal value within a given interval) can be made. The number of different physical communication channels or the number of time-redundant signal exchanges within a specific interval is defined by the property *redundancy degree*. Since both types of redundancy (namely spatial and temporal) can be combined for a single signal exchange, a separate instance of the redundancy property is required for spatial and temporal redundancy. For the temporal redundancy an additional attribute is required to specify the minimum temporal distance between two consecutive replicas of a signal exchange. The rationale behind this attribute is the requirement that for example disturbance bursts with a maximum duration of ε have to be tolerated. In that case, the minimum temporal distance between two consecutive replicas of the signal exchange has to be larger than the maximum disturbance burst duration ε in order to be able to tolerate such kind of burst. For further information on how to determine the distribution of the replica for a TDMA based protocol, you can refer to [3]. Explicitly specifying the

redundancy type is not required, since this information is implicitly defined via the spatial and the temporal redundancy degree.

1.1.2.2 Basic or System Software

In addition to the application software components, AUTOSAR also defines a layered architecture of basic (or system) software modules, which provide a basic platform for the execution of the application software components.

The AUTOSAR basic software is horizontally subdivided into different types of services, namely:

- input/output services, which provide standardized access to sensors, actuators and ECU onboard peripherals,
- memory services, which facilitate the access to internal and external (mainly non-volatile) memory,
- system services, which contain modules like operating system, ECU state management a.s.o.,
- and last but not least the communication services, which provide a communication stack used for access to the different vehicle networks (i.e., LIN, CAN, and FlexRay).

Communication Services

The communication services are a group of modules for vehicle communication (CAN, LIN, and FlexRay). The communication stack built up by the modules of the communication services is depicted in Figure 4. – Hereby the striped boxes indicate communication protocol specific modules. The "XXX" is thereby a placeholder for the respective communication protocol (i.e., CAN, LIN, FlexRay). Thus the AUTOSAR communication services contain

communication protocol specific instances of the transport protocol (TP) and network management (NM).

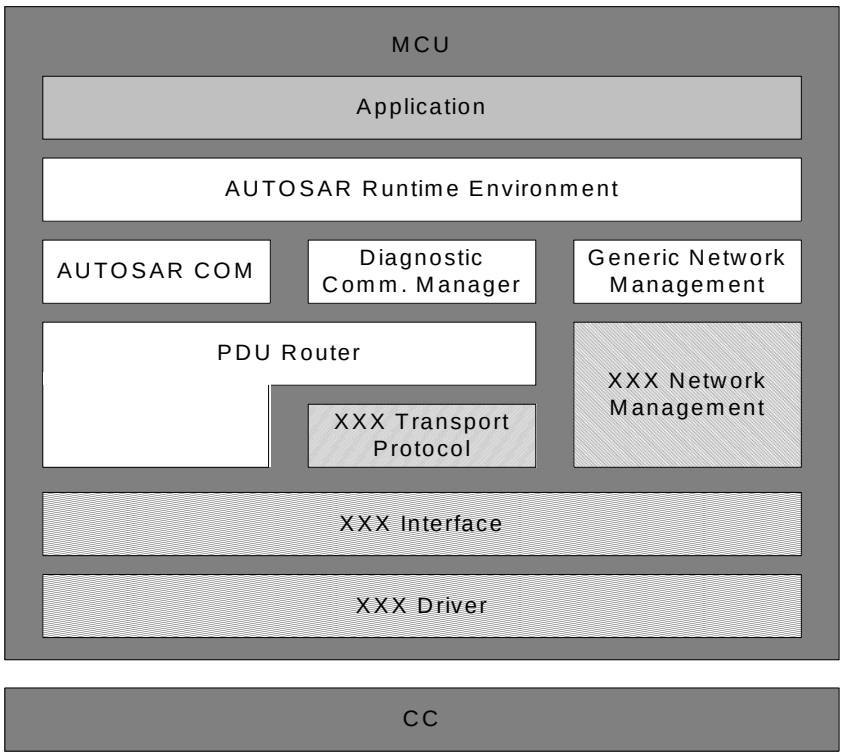

Figure 4: AUTOSAR Communication Services

XXX Transport Protocol (TP): In AUTOSAR the transport protocol is used to perform segmentation and reassembly of large PDUs (called messages) transmitted and received by the diagnostic communication manager. In AUTOSAR a dedicated TP is used for each communication protocol (CAN, LIN, FlexRay). These protocols are rather similar or even compatible (in certain configuration settings) to the ISO TP for CAN [6] specified in ISO/DIS 15765-2.2.

PDU Router: The PDU router module provides two major services. On the one hand it dispatches PDUs received via the underlying interfaces (e.g., FlexRay interface) to the different higher layers (COM, diagnostic communication manager). On the other hand the PDU router performs gateway functionalities between different

communication networks by forwarding PDUs from one interface to another of either the same (e.g., FlexRay ➔ FlexRay) or of different type (e.g., CAN ➔ FlexRay).

COM: The COM module provides signal-based communication to the higher layers (RTE). The signal-based communication service of COM can be used for intra-ECU communication as well as for inter-ECU communication. In the former case, COM mainly uses shared memory for this intra-ECU communication, whereas for the latter case at the sender side COM packs multiple signals into a PDU and forwards this PDU to the PDU router in order to issue the PDU's transmission via the respective interface. On the receiver side, COM obtains a PDU from the PDU router, extracts the signals contained in the PDU and forwards the extracted signals to the higher software layers.

Diagnostic Communication Manager (DCM): The diagnostic communication manager provides services which allow a tester device to control diagnostic functions in an ECU via the communication network (i.e., CAN, LIN, FlexRay). Hereby the DCM supports KWP2000 [5] standardized in ISO/DIS 14230-3 and the unified diagnostic services (UDS) protocol [2] standardized in ISO/DIS 14229-1.

Network Management (NM): Network management provides means for the coordinated transition of the ECUs in a network into and out of a low-power (or even power down) sleep mode. AUTOSAR NM is hereby divided into two modules: a communication protocol independent module (generic NM) and a communication protocol dependent module (CAN NM, LIN NM, and FlexRay NM).

XXX Interface: The Interface module is protocol specific, meaning that dedicated interfaces for the different communication protocols (i.e., FlexRay, CAN, and LIN)

do exist. Based on the frame-based services provided by the respective drivers (see below) the Interface modules facilitate the sending and the reception of *protocol data units (PDUs)*, where multiple PDUs can be packed into a single frame at the sending ECU and have to be extracted again at the receiving ECU. – In FlexRay the point in time when this packing and extracting of PDU takes place as well as the point in time, when the frames containing the packed PDUs are handed over to the respective driver for transmission or retrived from the driver upon reception is governed by the temporal scheduling of so-called *communication jobs* of the FlexRay Interface. Hereby each communication job can consist of one or more *communication operations*, each of these communication operations handling exactly one communication frame (including the PDUs contained in this frame).

XXX Driver: Just like the Interface module, the Driver module is protocol specific as well. The Driver module provides the basis for the Interface module, by facilitating the transmission and the reception of frames via the respective communication controller.

Runtime Environment (RTE): The AUTOSAR RTE provides the interface between application software components and the basic software modules as well as the infrastructure services that enables communication to occur between application software components.

Application Layer: Actually this layer is not part of the AUTOSAR basic software modules layered architecture, since this layer contains the AUTOSAR application software components described in Section 4.

When looking at FlexRay Driver, FlexRay Interface, FlexRay Transport Protocol, and COM, communication at different levels of abstraction and granularity is facilitated, namely

frame level, PDU level, message level and signal level. – Note that all of the previously listed requirements can be applied to any of these different levels of abstraction.

1.2 The FlexRay Protocol

In 2000 BMW, DaimlerChrysler, Philips and Freescale (Motorola) founded the FlexRay consortium [6, 7] with the objective to develop a new communication protocol for high-speed control applications in vehicles to increase safety, reliability and comfort. Since then the Consortium has grown to a number of more than 100 members including some of the automotive industry's largest and most influential players, such as General Motors, Ford and Bosch among others. In 2006, the BMW Group implemented the first FlexRay-based series application in the X5 family [8] demonstrating the performance of this new communication technology on the road.

The FlexRay protocol provides fast, deterministic and fault-tolerant communication to overcome the performance limits of previously established protocols in the automotive domain, e.g. CAN. Therefore, FlexRay supports two communication channels, each operating at a data rate of up to 10 Mbit/s. The FlexRay communication scheme includes a static and dynamic segment. Data transmission in the static segment is fully deterministic with guaranteed frame latency and jitter, whereas the dynamic segment provides a flexible bandwidth allocation for asynchronous data transmission. For the deployment of the FlexRay protocol all parameters of the communication scheme, such as the length and properties of the static and dynamic segment, have to be configured statically. These parameters highly depend on the requirements of the application.

Testing and monitoring approaches have to consider the detailed configuration parameters of FlexRay. Furthermore, the deterministic timing of the static and dynamic segment can be used to establish an efficient simulation environment.

For a detailed description of the static and dynamic segment, and the timing in FlexRay please refer to chapter FlexRay.

2 Objectives for Testing and Monitoring

Automotive systems often have to meet dependability requirements, due to the inherently safety critical nature of these kinds of systems (especially as far as the chassis domain is concerned). According to Laprie et al. [7] testing is one mean to establish the desired amount of dependability. Through testing the following goals can be achieved:

Fault removal: In the development phase of a system, design faults can be detected by means of testing and can be removed from the system, thus resulting in a higher dependability of the system.

Fault forecasting: When exposed to realistic load scenarios and when supplied with input that is close to real life, the frequency and the severity of faults can be assessed prior to system deployment (i.e., prior to start of production of the car). Based on this data forecasts regarding faults occurring in the field can be made.

Basically a distinction between static testing and dynamic testing can be made. Static testing comprises practices to verify the system without actual execution. Practices like static analysis (e.g., inspections, walk-throughs, data flow analysis, complexity analysis, static source code checks by compilers or dedicated source code checkers) or theorem proving by means of prover engines fall into this category. Since static testing of FlexRay-based systems is not fundamentally different from static testing of non FlexRay-based system, we will not address the practices of static testing any further.

In dynamic testing the system is exercised with a defined set of stimuli (the so-called test vectors) in order to judge – based on the responses of the system to these test stimuli – whether the system behaves according to its specification or whether the systems' responses deviate from this specification. Such a deviation from the system's specification is termed a failure of the system.

When conducting dynamic tests on a system however it is important to have proper means to monitor and record the response of the device under test to the test stimuli. In networked systems, the test stimuli can to a large degree be provided via the communication media. Similarly the responses of the device under test are to a large degree visible on the communication media as well. Therefore for properly testing (parts of) networked systems, some kind of monitoring device to record the network traffic as well as some device capable of providing the proper stimuli via the network are required.

2.1 "Criteria" to Test and Monitor

When testing a system (or a part of a system), the main interest lies in finding out whether the system behaves according to its specification or whether the observed behavior of the system deviates from the system's specification. Such a deviation can take place either in the time, in the value domain, or in the code domain.

2.1.1 Deviations in the Time Domain

When looking for deviations in the temporal domain, all timing-related requirements listed in Section 1.1.2.1 have to be taken into consideration and have to be applied at the different levels of granularity (i.e., frame level, PDU level, message level, and signal level).

As far as the period requirement of a connector is concerned, tests have to be conducted to validate that for periodic or sporadic information exchanges, the observed period matches the required period. In FlexRay, for frames scheduled in the static segment, this period is guaranteed by the FlexRay protocol in fault-free scenarios. For frames scheduled in the guaranteed part of the dynamic segment, the observed period may deviate by a maximum of almost the length of the dynamic segment (see Figure 5).

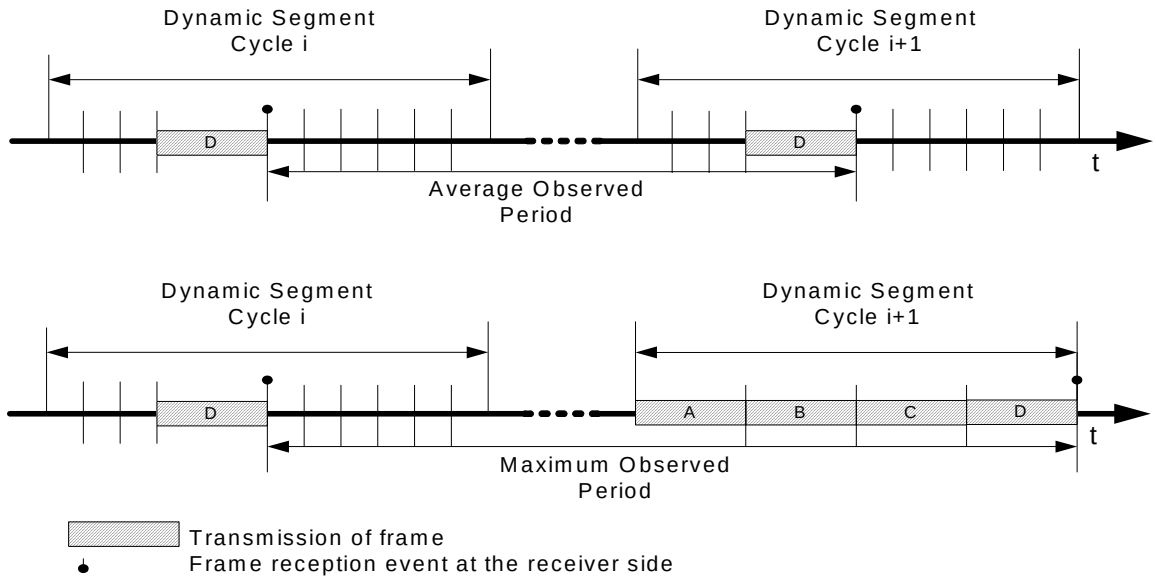

Figure 5: Observed Period in Guaranteed Part of Dynamic Segment

The upper part of Figure 5 illustrates the case, where no other frames are sent in the minislots prior to frame D (for which the period is observed) resulting in a period of a whole FlexRay communication cycle. In the lower part of Figure 5 a scenario is depicted where the minislots preceding the minislot for the transmission of frame D are occupied (and thus stretched) causing the transmission of frame D to be shifted to the end of the communication cycle, resulting in an observed period of one FlexRay communication cycle plus (in a worst case scenario with long frames and short minislots) almost the length of the dynamic segment.

For the best effort part of the dynamic segment, no such upper bound on the possible deviation can be given since indefinite postponement of the transmission of frames scheduled in the best effort part of the dynamic segment might take place. For data entities different from frames (which require the involvement of higher software layers) like PDUs, messages or signals, the period is governed by the temporal schedule of the FlexRay Interface's communication operations as well. Figure 6 illustrates this impact of the temporal schedule of the FlexRay Interface's communication jobs on the actual latency between the send request

(issued by the layer on-top the FlexRay Interface) and the actual send event on the communication media (and vice versa on the recipient side).

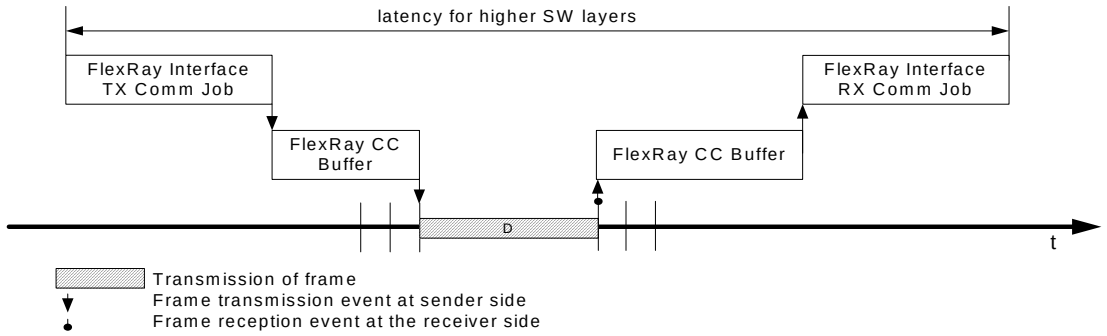

Figure 6: Impact of FlexRay Interface Communication Operations

For the mean latency requirement the values of the latencies observed by the receiver have to be measured and the mean value has to be computed. Again for frames transmitted in the static segment of FlexRay, the FlexRay protocol itself ensures a constant latency of one TDMA slot due to the static schedule. In the guaranteed part of the static segment, the mean latency will be in the granularity of the length of the dynamic segment (Figure 7), whereas in the best effort part of the dynamic segment, the mean latency can even be unbounded in case of high network load.

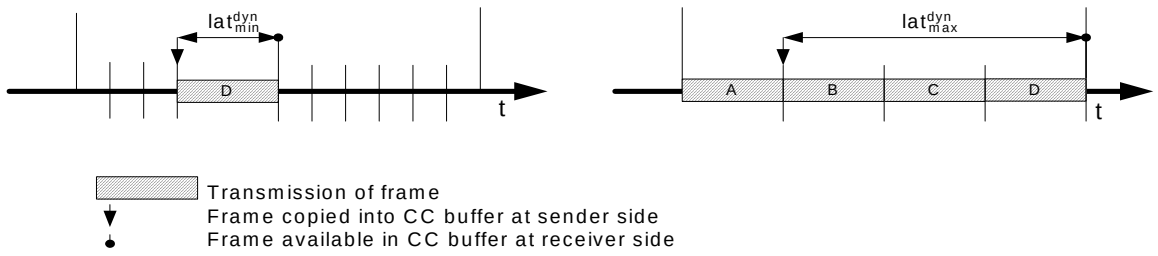

Figure 7: Latency and Latency Jitter in Dynamic Segment

For data entities different from frames (which require the involvement of higher software layers) like PDUs, messages or signals, similar to the observed period the mean latency is governed by the temporal schedule of the FlexRay Interface's communication jobs as well.

As far as the maximum latency jitter requirement is concerned, the FlexRay protocol causes frames scheduled in the static part to show a maximum jitter in the granularity of a single macrotick. Frames scheduled in the guaranteed part of the dynamic segment might exhibit a latency jitter of up to the length of the dynamic segment (i.e., the difference between $lat_{\max}^{dyn}$ and $lat_{\min}^{dyn}$ in Figure 7), whereas frames scheduled in the best effort part of the dynamic segment may be postponed indefinitely causing an infinite jitter.

Again for data entities different from frames (which require the involvement of higher software layers) like PDUs, messages or signals, the maximum latency jitter is governed by the temporal schedule of the FlexRay Interface's communication operations as well.

For testing the temporal redundancy degree requirement, the number of observed temporal replicas of a information exchange has to be counted and compared against the required temporal redundancy degree. Since neither the FlexRay protocol nor the AUTOSAR basic software does provide any inherent support for temporal redundancy, the proper handling of temporal redundancy is a matter of the application software.

Meeting the minimum temporal distance requirement between two consecutive replicas of the information exchange is partly supported by the FlexRay protocol. By scheduling the replicas in proper TDMA slots (with a sufficient temporal distance between the slots), the temporal distance requirement is enforced by the FlexRay protocol when using the static segment. For the dynamic segment a worst case calculation can be made (assuming that all minislots between the minislots of the consecutive replicas are unoccupied) in order to have the FlexRay protocol ensure this requirement. Since this approach however is based on a rather pessimistic assumption, the observed temporal distance will mostly be way larger than the minimum temporal distance.

2.1.2 Deviations in the Value Domain

Two main kinds of deviations in the value domain can be observed. First, the information content is invalid, since a protecting checksum (in case of FlexRay frames for example a frame CRC) indicates that the information has been mutilated.

Secondly, an information content that differs from a known content leads to the conclusion that there is a deviation in the value domain. Nevertheless, in order to come to this conclusion, knowledge about the correct information content is required. For information entities of limited range (e.g., enumeration values), exact knowledge about the correct information is often available (e.g., because the tester knows the exact position of ignition key). For information entities of rather large range (e.g., for signal values of 32 bits), however, in most cases only a validity interval is available. In that case, the observed information content can only be validated against this validity interval.

2.1.3 Deviations in the Code Domain

When looking at deviations in the code domain, the following deviations have to be considered.

The bit encoding on the physical layer differs from the specification. This deviation are mostly caused by faulty transceivers and/or encoding units in the FlexRay controller.

The observed frame format on the data link layer differs from the frame format defined in the FlexRay specification. Such a deviation can be caused by a faulty transmitter unit of the FlexRay communication controller or a faulty star coupler.

A last deviation is observed when the observed signal packing (i.e., the ways signals are packed into frames) differs from the specified signal layout in the frame. Such deviations are most probably caused by incorrect configurations of the AUTOSAR COM layer.

2.1.4 Other Deviations

For testing the spatial redundancy degree requirement, the number of observed spatial replicas of an information exchange have to be counted an compared against the required spatial redundancy degree. To achieve this, the available channels of the communication system have to be monitored for occurrences of replicas of the information exchange.

2.2 Operational Scenarios for Testing and Monitoring

As far as operational scenarios are concerned FlexRay-based systems (like any other systems) have to be tested in the fault-free case to ensure the system's proper operation when executed in the absence of faults.

As already mentioned in Section 1.1, however, FlexRay has been developed for the deployment in safety-related application areas. Since systems intended for safety-related purposes need to remain functional even in the presence of faults[1], the system inherently has to be able to tolerate these faults. Therefore testing of safety-related systems has to take place under faulty conditions as well, since the previously addressed fault tolerance requirements on the system mandate that faulty conditions are part of the system's "normal" operational scenario.

In order to be able to test the fault tolerance properties of the system, the faulty conditions have to be induced intentionally as part of the respective test case. The induction of these faulty conditions is termed fault injection. – Depending on whether the faults are injected by means of hardware (e.g., by electromagnetic interference bursts) or by software (e.g., by intentionally flipping single bits in memory) we speak of hardware- or software-implemented fault injection (see Section 3.1.2) respectively.

[1] as long as the number, the frequency of occurrence, the duration, and the nature of these faults is covered by the system's fault hypothesis

3 Monitoring and Testing Approaches

In order to achieve the test objectives introduced in the previous section, different monitoring and test approaches are possible. In the following, a basic distinction between software-based and hardware-based approaches is made. Software-based validation uses a simulation model of the system to analyze the behavior of the application. Hardware-based validation includes a hardware setup of the system for the investigation. Both approaches can be used either for the analysis of the total system or for the analysis of parts of the system, i.e. one or a number of ECUs. In this chapter the focus is always laid on the effects of the software running on the MCU, and therefore on the application layer and the basic software layers.

3.1 Software Based Validation

Software-based validation provides powerful means to analyze the application behavior at an early stage of the development.

3.1.1 FlexRay Abstraction Levels

Computation effort is a big issue for each simulation. Modeling a complete FlexRay network might become quite complex: distributed ECUs each including FlexRay controllers and MCUs with basic software layers and the application layers. Furthermore, these networks can be connected with others via gateways (see Figure 1).

To minimize the complexity of such simulations, we propose to model the application only in full detail, whereas the model of the FlexRay controller and network is significantly simplified utilizing the deterministic timing behavior of the FlexRay protocol. Therefore, we introduce so-called abstraction levels on the architecture and timing level.

3.1.1.1 Architecture Level

As described in Section 1.1.1, the architecture of a FlexRay network consists of ECUs interconnected by a shared communication media. Each ECU includes one or more FlexRay controllers and microcontroller units (MCUs). On the MCU, the application layer and the basic software layers are running. The application layer contains a number of software components which implement the actual application functionality (e.g. ABS calculation routines). The basic software layers provide means and services to transmit and receive data via the FlexRay controllers. The application software components use the services of the basic software layers to communicate with software components running on other ECUs.

Figure 8 illustrates this ECU architecture whereas two interfaces are introduced. First, there is a so-called application interface between the application layer and the basic software layers, and second, there is a controller-host interface between the basic software layers on the MCU and the FlexRay Communication Controller. These interfaces can be used to reduce the simulation effort.

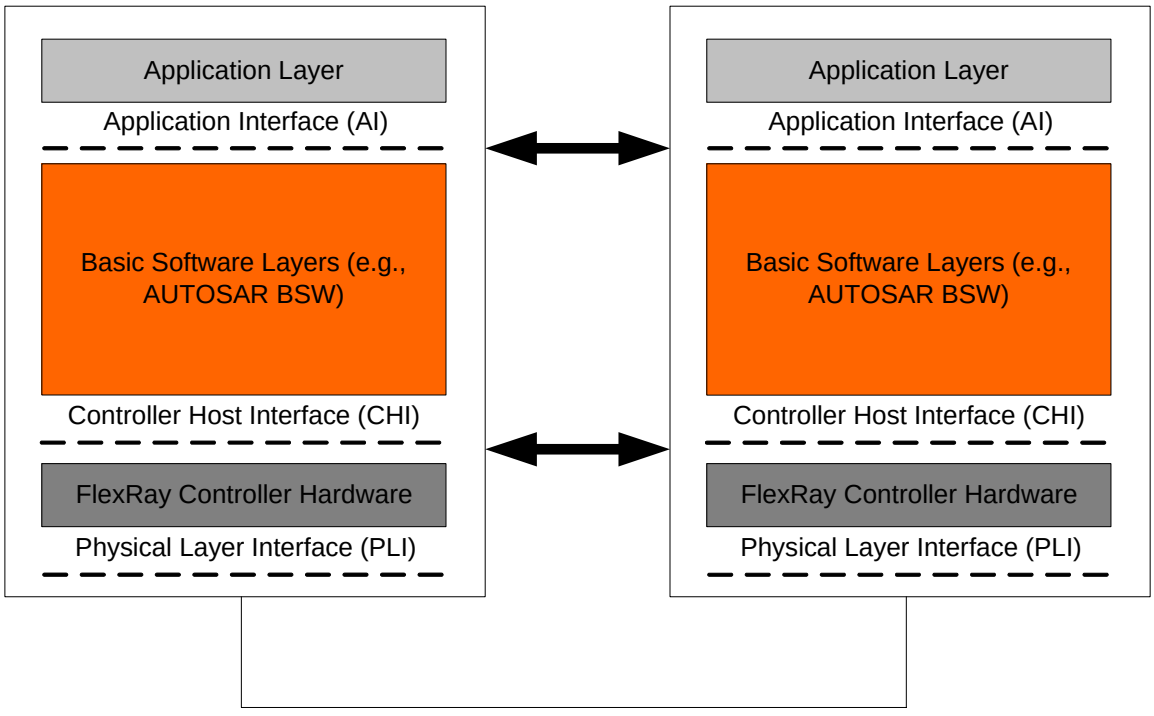

Figure 8: Abstraction Level on Architecture

Application Interface (AI)

At this level, the simulation model contains the full functionality of the software components of the application. The functionality of the basic software layers and the FlexRay Controller Hardware is emulated providing an application interface.

The application interface is a signal-based interface that delivers updates of the signals according to the timing of the basic software layers and FlexRay controllers. Emulating this interface in simulation allows considerable simplifications of the simulation model. Only the timing aspects of the signal updates have to be considered. Issues like signal-to-frame packing and the details of the FlexRay timing do not need to be modeled.

Controller Host Interface (CHI)

The full functionality of the software components of the application and the basic software layers is simulated at this level while the functionality of the FlexRay Communication Controller is emulated providing, therefore, a controller host interface abstraction.

The controller host interface can be modeled as a buffer-based interface that delivers updates of the FlexRay frames according to the FlexRay timing. Emulating this interface in simulation also allows considerable simplifications of the simulation model. Only the timing aspects of the frame updates have to be considered. The FlexRay controller functionality like clock synchronization and startup etc. do not need to be modeled in detail.

Physical Layer Interface (PLI)

A last interface can be exhibited: the Physical Layer interface between the functionality of FlexRay Communication Controllers and the network physical layer. Simulation at this level basically does not make much sense, since the computational effort required to simulate the correct bit timing (which is required at this level of abstraction) is rather huge. Note that, for hardware-based validation however (see Section 3.2) the physical layer interface is an

important interface which is subject to faults and thus has to be considered in the validation process.

3.1.1.2 Timing Level

Simulation effort can also be significantly reduced by choosing an adequate time resolution. Due to the time-driven nature of the FlexRay protocol, the transmission of data is triggered by pre-defined points in time. This timing hierarchy (Section 1.2) can be used to introduce abstraction levels on the timing level.

Timing Level 1: Communication Cycle

On this timing level, the updates of frames and the contained signals are performed in the granularity of communication cycles. As shown in Figure 9, at the beginning and end of each communication cycle the contents of all signals and frames are updated. The data to be transmitted is written at the beginning of each communication cycle, the data to be received is read at the end of each communication cycle.

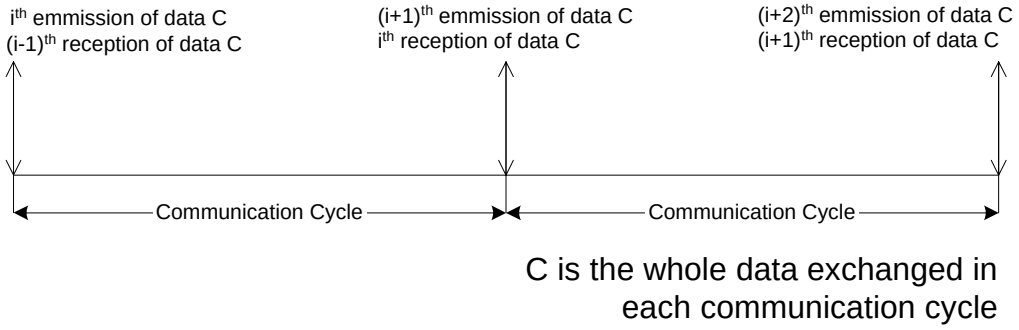

Figure 9: Timing Level 1 – Communication Cycle

This timing level can be optimally combined with the application interface level to provide a fast and quite abstract view of the communication timing of signals. It is assumed that transmission latencies introduced by the basic software layers is small enough so that an update of the signal is available at the end of each communication cycle. In combination with the controller-host interface, however, this timing level does not provide the necessary accuracy

to analyze the detailed effects of the transmission latency introduced by the basic software layers.

Timing Level 2: Static Slots and Simple Dynamic Segment Arbitration

This is a more accurate level where the updates of frames and the contained signals are performed in a more detailed manner. In the static segment, the timing of the static slots is emulated. The data to be transmitted is read at the beginning of each slot, the data received within the slot is provided at the end of the static slot. Figure 10 shows the update times for slot 1 of the static segment. The other slots of the static segment are updated in the same way.

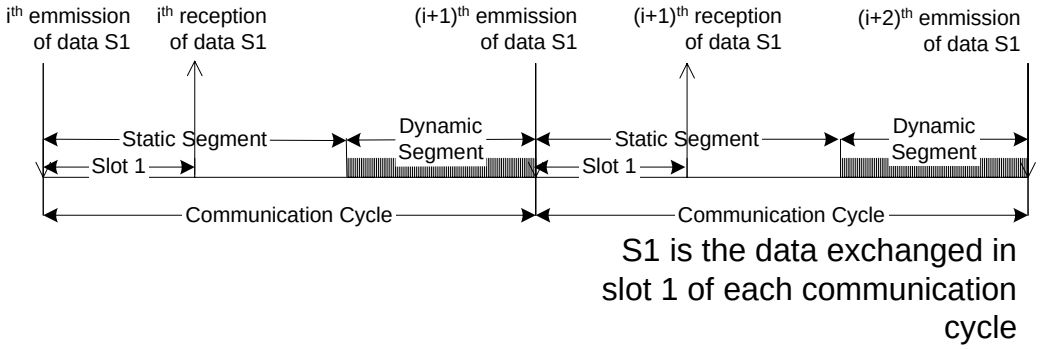

Figure 10 Timing Level 2 –Static Slot Arbitration

For the dynamic segment, only the beginning and the end of the segment are emulated. Data to be transmitted is read at the beginning, data that has been received is provided at the end of the dynamic segment.

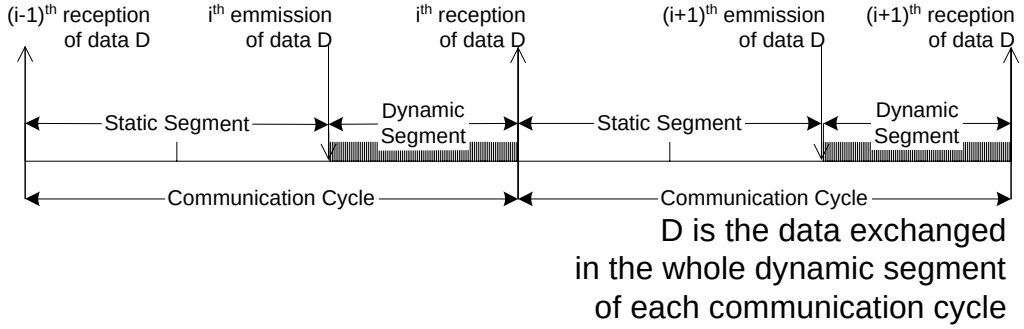

Figure 11 Timing Level 2 - Dynamic Segment Arbitration

This timing level can be optimally combined with the controller-host interface level to analyze the effects of the transmission latency introduced by the basic software layers. For example, the AUTOSAR FlexRay Interface layer contains communication operations that are scheduled synchronously to the FlexRay communication to read and write to the FlexRay buffers. The configuration of the timing of these communication operations determines the earliest and latest point in time an update of the transmitted data is available. These effects can be analyzed in combination with the software components of the application layer.

In combination with the application interface level, this timing level provides a very accurate update of the frames without any considerations of the latencies introduced by the basic software layers. This may lead to erroneous assumptions about the earliest and latest point in time a transmitted data is available.

Timing Level 3: Static Slots and Advanced Dynamic Segment Arbitration

This timing level is a refinement of the timing level 2. The sampling points are chosen in the same way as in level 2, however, the behavior of the mini-slotting media access scheme is emulated in more detail. This priority-based algorithm postpones the transmission of frames scheduled in the best-effort part of the dynamic segment in case of peak-load (Section 1.2).

More Detailed Approaches

Further simulation approaches include the emulation of the timing of the dynamic segment in a more detailed way.

3.1.2 Fault injection

Fault injection in the simulation model enables the designer to analyze the dependability characteristics of the system at an early stage of development. The main objectives are fault removal – testing the correct behavior of fault tolerance mechanisms – and fault forecasting – investigating the robustness of the application in fault scenarios.

A simulation model provides good means to fully control the location and point in time a fault shall be injected. In our simplified model of the FlexRay controller and FlexRay network we focus on the effects of failures that can be seen on the previously described architectural and timing abstraction levels.

On the architectural level, in the simulation model, the application interface as well as the controller host interface, are emulated as shared memories. So, in accordance to the chosen timing abstraction level the contents of the shared memories can be accessed for fault injection.

3.1.2.1 Application Interface

Fault injection on the application interface supports directly influencing and investigating the behavior of the application model. The application interface provides access to the services of the basic software layers. In the simulation model, these services are modeled in a very simplified manner providing signal updates and status information of services such as network management.

Signal updates can be easily influenced in the time and value domain. The time domain depends on the chosen timing abstraction level that corresponds to the update periods of the FlexRay protocol. In contrast to CAN, where the periods of low priority data may significantly jitter even in the fault-free case, FlexRay provides deterministic transmission periods for the static segment and the guaranteed part of the dynamic segment as specified during design time. Even in case of a failure of the application (assuming the basic software layers and the FlexRay controller are working correctly), the update periods of the signals stay constant, however, the signal values may be incorrect or obsolete. This protocol property significantly eases the simulation of application faults in the time domain: either a correct signal is available or is not available at the application interface within the specified transmission period. Within the best-effort part of the dynamic segment, signal updates may

also be lost or postponed in case of peak-load scenarios, i.e., without any specific failures. This behavior can also be easily emulated by mutilating the value of the signal at the application interface.

Deviations in the value domain can also be achieved by mutilating the contents of the correct signals provided by the application interface. This kind of fault simulation allows analyzing how the application model reacts in case of wrong input data. Focusing on communication network failures, this method enables investigating the robustness of the total system in case of inconsistent data transmissions. Due to failures, it might occur that only a subset of ECUs receives correctly signals, whereas others receive no signal updates.

Other services of the basic software layer such as network management can also be influenced by fault injection. The status of the service can be changed at one or a subset of ECUs to achieve erroneous states and analyze how the application model reacts in those cases.

Timing level 1 is the adequate granularity for all this kind of fault simulation scenarios at the application interface. Investigations at finer granularity level (i.e., using a higher timing level) provide only realistic data, when the latency introduced by the basic software layer is also considered. Therefore, the below described controller-host interface level is necessary.

3.1.2.2 Controller Host Interface

Fault injection on the controller host interface is aimed at analyzing the effects of the basic software layer and the application model. The controller host interface provides access to the controller status and buffer contents of the FlexRay communication controller. In the simulation model, the data transmission is emulated by updating the controller host interfaces of all controllers within the specified timing abstraction level. The basic software layers and the application model make uses of these values after the update.

At this interface, all failures that might occur in the FlexRay controller or on the communication network can be simulated. These failures include invalid frame receptions, CRC errors, incoming or outgoing link failures, communication channel failures and more.

The FlexRay specification does not specify the register and buffer layout of the controller host interface of a FlexRay controller. Rather, the services and the contents of the information provided are described. The most important services for the fault simulation are:

Clock Synchronization

The FlexRay protocol requires that all controllers share a common time-base with each other in order to execute receive and transmit operations in a time-driven manner. To establish this common time-base, a distributed fault tolerant clock synchronization algorithm [11] is used. Upon power-up, each controller performs a specific startup procedure to establish the common time-base. During operation, each controller performs a synchronization algorithm to maintain the synchronization with the other controllers. If a controller is not synchronized, it may not fully participate in the data transmission.

Reasons why a controller is not synchronized can be manifold. Examples include that no other controllers have powered-up, the FlexRay communication channels are broken, the frequency of the FlexRay controller clock is out of specification, etc.

To analyze the behavior of a FlexRay-based application, we propose to model only a very abstract view of the clock synchronization algorithm. This abstract view includes two modes: a synchronized and a not synchronized mode of the FlexRay controller. In the synchronized mode, the controller may correctly receive and transmit frames, in the not synchronized mode, the controller does not receive correct frames. Furthermore, it is assumed the controller does not transmit any frames when residing in the not synchronized mode.

Frame Reception

For each frame that is received, a "receive" status is provided in the controller host interface. This "receive" status indicates whether the frame has been received correctly or whether any problems have occurred during reception. In our simplified model of the FlexRay controller the status of the frame reception is mutilated on the controller-host interface so that various failure cases can be emulated. These cases include invalid cyclic redundancy check (CRC) or check sums, coding violations, clock synchronization problems etc. These cases can affect one or a certain subset of frames, for example:

- All frames received from a specific node,
- All frames received on a specific channel,
- All frames received within a specific time period,
- And other subsets of frames.

This method of selecting certain subsets of affected frames enables to inject efficiently different kinds of faults. In the simulation model the behavior of the basic software layers and the application can then be analyzed, e.g. whether the failure of one node or communication channel has been correctly detected.

Frame Transmission

The success or the failure of a frame transmission results in the value of the "receive" status in the controller-host interface of all receiving controllers. Thus, mutilating the controller host interface of all receiving controllers enables to emulate various frame transmission problems in the simulation model. Furthermore, the controller host interface of the transmitting controller must be mutilating indicating a failed transmit confirmation for the sender.

Choice of Timing Abstraction Levels

Fault injection at the controller host interface enables to analyze the behavior of the basic software layers and the application model. Whereas the reaction of the application model can be investigated at the application interface, the behavior of the basic software layer is of specific interest for this type of fault injection. Timing level 1 is not sufficient for these investigations since it does not provide the earliest point in time frames are available on the controller host interface. Timing levels 2 and 3 and more detailed approaches are adequate for these fault simulations at the controller host interface.

3.2 Hardware-Based Validation

In hardware-based validation the device under test is not simulated in software. In this case, the whole ECU (or even a combination of multiple ECUs) and thus a compound of hardware and its embedded software is exercised in the course of the validation. In order to be able to conduct such a hardware-based validation, the test-bed itself must be hardware-based and the interface between the test-bed and the device under test is a hardware interface. In networked systems this interface is, on the one hand, the communication media and on the other hand, the input/output interface to the environment (i.e., sensors and actuators).

Based on whether the response of the device under test is or is not fed back into the test vector generation, we distinguish between *open loop* and *closed loop* approaches.

3.2.1 Open Loop Approach

In open loop approaches the responses of the device under test to test stimuli are *not* fed back into the test stimuli generation in order to produce new test stimuli. Therefore the test bed itself is divided into three main parts, namely a test stimuli generation part, a monitoring part, and a controlling part. While the former is responsible for providing test stimuli to the device under test, the responsibility of the latter lies in the monitoring of the responses to these test

stimuli. For the coordination of the provision of test stimuli and the recording of test responses, the test-bed comprises a controlling part as well which contains a database for retrieving test stimuli and storing test responses for later evaluation.

Hereby the provided test input as well as the recorded output is generated/obtained at the network interface (network stimulus provider and network monitoring device) and the environmental interface (environmental stimulus provider and environmental monitoring device) of the device under test (DUT). Figure 12 illustrates this test setup.

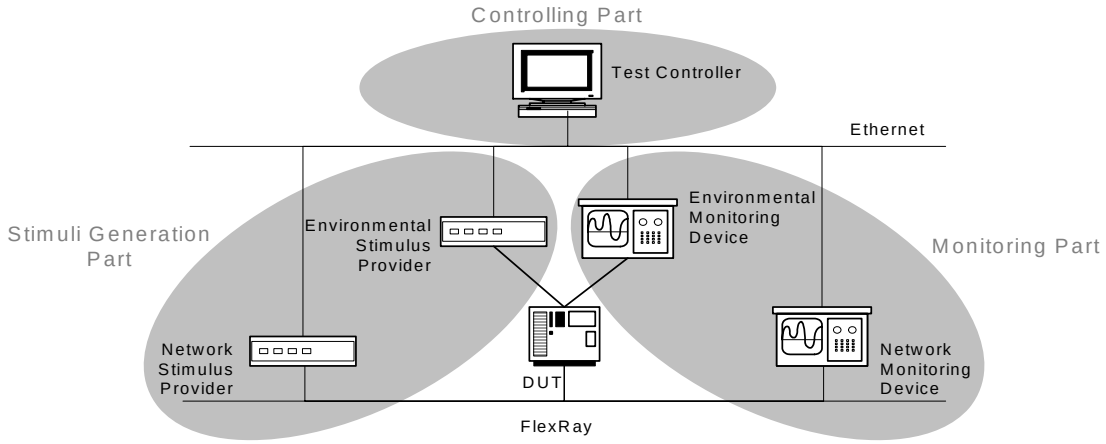

Figure 12: Test Setup for Open Loop Approach

As far as the environmental interface is concerned, FlexRay-based system are not different from non FlexRay-based systems. Thus we do not further address this interface in the following sections.

3.2.1.1 Test and Monitoring Levels

Testing and monitoring in FlexRay-based systems takes place at the interfaces introduced in Section 3.1.1.1

Physical Layer Interface

Testing in fault free scenarios at the physical layer interface is basically testing the correct operation of the FlexRay communication controller and the proper setup of the

communication media (e.g., proper termination in case of a bus topology). Given a correctly operating FlexRay communication controller (including line driver) properly setup communication media and the absence of faults on the communication media (e.g., no EMI burst, etc.), no other causes for faults are possible.

With these units of failure in mind the following faults have to be considered during testing (from the monitoring device's perspective):

Faults in the value domain: At the physical layer interface, faults in the value domain mean a deviation of the observed voltage level from the level specified in the FlexRay physical layer specification. This deviation can be caused by improper termination of the communication media, by short circuiting one or both lines of a FlexRay channel to ground or to supply voltage, or by faults in the star coupler device in case of star topologies.

Faults in the time domain: Faults in the time domain can be a deviation of the observed bit timing from the specified bit timing, meaning that, for example, the duration of a bit cell is too short or too long. They can also come from a deviation of the bit timing in the first derivative; in this case, it means that the edge steepness is either to high or too low. Possible causes for these kinds of faults are a faulty oscillator in the FlexRay communication controller, a faulty encoder unit or an improper configuration of the FlexRay communication controller.

Faults in the code domain: Deviations from the specified coding are termed coding failures. On the bit level FlexRay uses not return to zero (NRZ) coding. In order to facilitate bit level synchronization between transmitter with the receiver(s), each byte is additionally framed with a dedicated start and stop bit (each exhibiting a different logical level), thus enforcing at least one edge per byte. Causes for coding faults are

faulty transceivers and/or faulty encoder units in the FlexRay communication controller.

Controller Host Interface

Faults in the value domain: Faults in the value domain at the controller host interface mean a deviation of frame content from the specified frame content. Hereby a distinction has to be established between a frame with incorrect CRC, on the one hand, and a frame with correct CRC and invalid payload, on the other hand. The former case can be caused by a faulty transmitter or a faulty CRC unit of the sending FlexRay communication controller, or by a faulty MCU, a defect in the application program, or by an improper configuration of the FlexRay communication controller. In addition to the previous causes, the latter case might be induced by a faulty controller host interface. Both cases can be generated by a faulty communication media as well.

Faults in the time domain: At the controller host interface the following faults in the time domain are possible: early frames (i.e., frames are transmitted prior to the specified point in time) and late frames (i.e., frames are transmitted after the specified point in time). A special case of late frames are the cases where frames are not transmitted at all (omission failure). Possible causes for timing failures are a faulty oscillator in the FlexRay communication controller, an improper configuration of the FlexRay communication controller, a faulty star coupler, a faulty communication media, or, a faulty MCU or application program.

Faults in the code domain: As far as coding at the controller host interface is concerned, FlexRay uses a defined frame format consisting of a frame start sequence, a frame header, the frame payload and a frame trailer containing the frame's CRC. Each of these frame parts has a defined length. Any deviation from this frame format

is considered as a coding fault at the controller host interface. The cause for such a fault is a faulty transmitter unit of the sending FlexRay communication controller or a faulty star coupler (inducing an oversized truncation of the frame start sequence).

Application Interface

Faults in the value domain: Faults in the value domain at the application interface mean a deviation of signal content from the specified signal content. Hereby a distinction between a signal tagged as invalid and a signal that is tagged as valid but exhibits an incorrect value can be made. Both cases might be caused by a value domain fault on the data link layer, a faulty MCU or application program, or a faulty configuration of the AUTOSAR COM layer.

Faults in the time domain: At the application interface, faults in the time domain are either caused by faults in the time domain at the controller host interface or by late or early activation of communication tasks which are responsible for packing signal to be transmitted into the respective FlexRay frames. The effects of these faults are a deviation for the signal's temporal requirements (see Section 1.1.2.1).

Faults in the code domain: If we consider coding at the application interface, multiple signals are packed into a single frame according to a specified signal layout for each frame. Any deviation from this specified signal layout is interpreted as a coding fault at the application interface. Causes for such a coding fault are mostly improper configuration of the AUTOSAR COM layer or faults of the MCU.

3.2.1.2 Testing under Faulty Conditions

As stated before in order to perform test under faulty conditions, fault injection is usually required. In open loop hardware test setups, the device under test in general cannot be modified. Thus the DUT must be considered as a black box leaving only the network

interface and environmental interface as targets for fault injection. In the following we focus on the network interface.

With proper hardware modifications, the network stimulus provider (under control of the test controller) can inject all faults described in the previous section at the physical line interface, at the controller host interface and on the application interface as well.

Coding faults at the physical line interface, for example, can be injected by implementing a special encoding unit that provides controlled means to intentionally violate the bit encoding scheme defined in the FlexRay specification. Faults in the value domain at the controller host interface, for example, can be injected by deliberately producing an incorrect CRC at the network stimulus provider. Value faults at the application interface, for example, can be injected by intentionally sending incorrect signal values or by tagging signals intentionally as invalid while leaving the frame's CRC intact.

3.2.2 Closed loop approach

In contrast to open loop approaches, in closed loop ones, the responses of the device under test are fed back into the stimuli generation in order to produce new test stimuli. Thus, the environmental stimuli provider and environmental monitoring device as well as the network stimuli provider and network monitoring device of Figure 12 are tightly connected with each other or implemented as one component (see Figure 13).

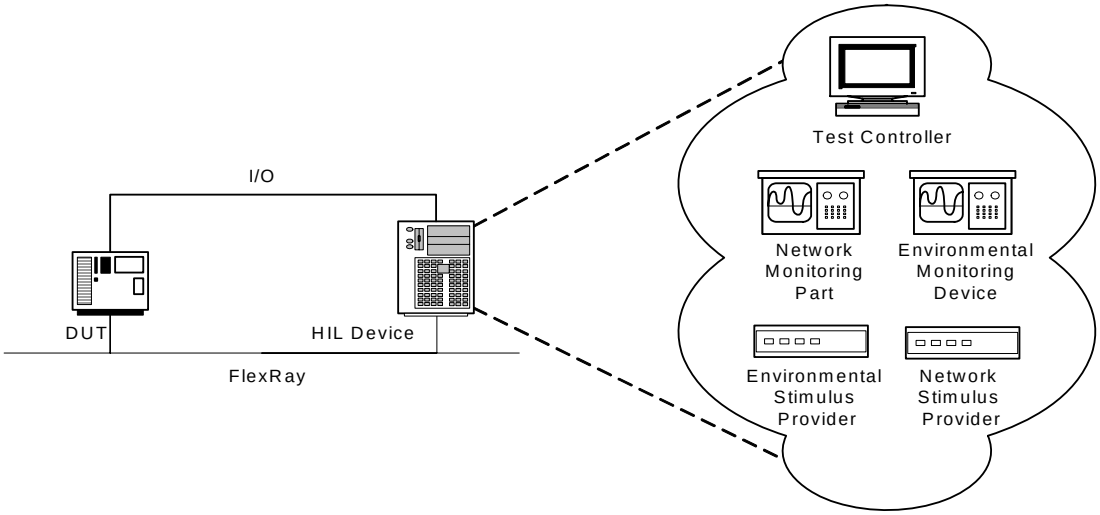

Figure 13: Test Setup for Closed Loop Approach

This method is often also called residual bus simulation (in German language "Restbussimulation") or hardware in the loop (HIL) system. HIL systems include a complete model of the other ECUs and a detailed model of the controlled environment, e.g., a braking system including the behavior of brakes, the car and the road. Residual simulation usually includes a simplified model of the other ECUs and no or only a basic model of the controlled environment.

The test and monitoring levels for FlexRay based applications are the same as introduced for open loop approaches.

4 Discussion of Approaches

Having identified several possible testing approaches in the previous section, the question remains which of these techniques is the one best suited for the development of FlexRay-based applications and systems. In this section we compare the presented approaches by evaluating them in terms of cost, accuracy, and easiness of the testing process.

4.1 Software-Based Approaches

Purely software-based approaches, where no target hardware is involved, are flexible and adaptive to the test purpose, and therefore it becomes easy to control the test execution, the monitoring of the test responses, and the injection of faults. This results in better reproducibility of tests and fault injections.

The possibility to conduct test before the target hardware has been built is a further benefit of this approach. It makes tests possible very early in the development life cycle and supports the detection of development faults in the first step of the development cycle. This is a strong factor for dramatically reducing the cost induced by these faults [12].

On the other hand, the major drawback of the approach is the accuracy of the software model and, in particular, the compliance between this model and the actual hardware platform in terms of timing behavior. From the functional point of view, the purely software-based approaches can perfectly mimic the behavior achieved in the actual system. As far as the temporal aspects (e.g., the latencies, the achieved accuracy, a.s.o.) are concerned however, the purely software based approaches reach their limits as soon as the desired accuracy of the tests and of the corresponding monitoring reaches the lower (finer grained) timing levels (see Section 3.1.1.2). Therefore, another important aspect is the cost of software-based testing approaches in terms of complexity of the model and computational resources for its analysis. In any case the accuracy of the software model has to be compliant with the granularity of the timing requirements of the application software otherwise the results obtained by the

performed test might not provide a sufficiently strong guarantee. While testing and monitoring on large scale timing levels (e.g., using a granularity of a single communication cycle) the computational resources required for the simulation are rather moderate, when moving to lower (finer grained) timing levels however, the computational resources may increase dramatically in terms of memory and CPU usage. Therefore when applying purely software-based approaches a trade-off has to be made between duration and the accuracy of a test run.

Hardware-Based Approaches

In hardware based approaches, the cost factor related to the accuracy of a model and therefore of its ability to fit to a given temporal granularity is no more a problem. In fact, there are no additional computational costs for a fine grained level of timing requirement against a coarse grained one.

However hardware-based approaches exhibit the disadvantages that they are less flexible and, in particular, the means of control over the test execution, monitoring and fault injection is rather limited or comes at great costs (e.g., expensive special purpose devices for reproducible deterministic fault injection [13]).

A further drawback of hardware-based approaches is the fact that usually the availability of the final target hardware is rather late in the development life cycle; this either requires test to be conducted on early prototype samples (which exhibit flaws of their own) or to postpone the test till the final hardware is available. Both choices cause increased cost, either due to testing on immature hardware or due to late detection of development errors and thus increased cost for correcting and fixing these errors).

As mentioned formerly, two methods can be used for hardware based tests: open-loop tests and closed-loop tests. An example of functionality where a closed-loop based test is needed are complex communication services like transport protocols, network management, and

diagnostic communication management. Each of these services implements a more or less complex communication protocol, requiring state machines at sender and receiver, where the state transitions are triggered by the messages received and/or transmitted. Without the possibility to react to the test responses of the DUT, the implementation of this kind of complex communication protocols in the test bed is not possible. Another example is the testing of distributed control loops where the DUT is one of the ECUs participating in the control loop. Similar to the complex protocols, distribute control loops necessitate that the test bed is capable of responding to the test responses of the DUT, which renders an open-loop approach inadequate in these situations.

In general closed-loop approaches are more expensive (due to need for more computational resources and the fact that the tester itself has to be carefully designed and developed) than open-loop ones. Therefore in practice a combination of both setups is used. Testing with closed-loop testing setups only takes place when the previous tests conducted in cheaper (i.e., open-loop) setups have successfully been completed.

5 Conclusion

Testing FlexRay-based systems has to take place at the interfaces of the different levels of abstraction of the hard- and software architecture, namely at the physical layer interface, at the controller-host interface, and at the application interface. Due to the nature of the FlexRay communication protocol, the responses of the DUT have to be monitored and examined not only for deviations in the value domain, but also doe deviations in the time and the code domain.

These tests have to be conducted in fault-free scenarios as well as under faulty conditions, which must be cause by means of fault injection. The application of both pure software-based tests and hardware-based tests in open- and closed loop test scenarios makes perfect sense, since each of the approaches has its own merits and drawbacks. Each one has its own niche, where it provides the most benefits.

We therefore recommend using a combination of all these approaches with respect to the different development stages of the whole system. We propose starting with pure software-based approaches as long as no hardware is available. In the pure software-based approach general functional tests can be conducted at a rather large scale timing levels. Once these tests have been completed successfully, additional tests at a smaller timing scale can be conducted.

Once hardware is available open-loop tests can be conducted to verify the correct timing and functionality of the simple application parts of the DUT. Once these tests have been successfully completed as well, closed-loop testing approaches can be used to verify the complex protocols and the distributed control loop functionality.

References

[1] T. Scharnhost, H. Heinecke, K.-P. Schnelle, H Fennel, J. Bortolazzi, L. Lundh, P. Heitkämper, J. Leflour, J.-L. Maté, and K. Nishikawa, *AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures*, Convergence 2004, International Congress on Transportation Electronics, Detroit, Michigan, USA, 2004

[2] Pop, T., Pop, P., Eles, P., Peng, Z., and Andrei, A., *Timing Analysis of the FlexRay Communication Protocol*. In Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS); pp. 203-216, IEEE Computer Society, Washington, DC, July 5-7, 2006, DOI= http://dx.doi.org/10.1109/ECRTS.2006.31

[3] Gaujal, B. and Navet, N., *Maximizing the Robustness of TDMA Networks with Applications to TTP/C*. Real-Time Systems, vol. 31, issue 1-3, pp 5-31, December 2005. DOI= http://dx.doi.org/10.1007/s11241-005-2743-4

[4] ISO (International Organization for Standardization), *Road vehicles – Unified Diagnostic Services (UDS) – Part1: Specification and Requirements*, ISO/DIS 14229-1, 1, rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland, 2004

[5] ISO (International Organization for Standardization), *Road Vehicles – Diagnostic Systems – Keyword Protocol 2000 – Part 3: Application layer*, ISO/DIS 14230-3, 1, rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland, 1999

[6] ISO (International Organization for Standardization), *Road Vehicles – Diagnostics on Controller Area Networks (CAN) – Part 2: Network Layer Services*, ISO/DIS 15765-2.2, 1, rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland, June 2003

[7] J.-C. Laprie, B. Randell, A. Avizienis, and C. Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, 1(1), pages 11-33, Los Alamitos, CA, USA, 2004

[8] R. Mores, G. Hay, R. Belschner, J. Berwanger , S. Fluhrer, E. Fuchs, B. Hedenitz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, M. Sprachmann, *FlexRay - The Communication System for Advanced Automotive Control Systems*, Proc. SAE, Paper 2001-01-0676

[9] FlexRay Consortium Web Page, http://www.flexray.com

[10] J. Berwanger, A. Schedl, BMW Group, Ch. Temple, Freescale Semiconductor, *FlexRay Hits the Road*, Automotive DesignLine, November 15, 2006, Available at http://www.automotivedesignline.com

[11] J. L. Welch and N. A. Lynch, *A New Fault-Tolerant Algorithm for Clock Synchronization*, Information and Computation, vol. 77, no. 1, pp. 1-36, April 1988.

[12] G. Tassey, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, NIST Report 02-3; National Institute of Standards and Technology, Acquisition and Assistance Division, Building 101, Room A1000, Gaithersburg, MD 20899-0001, USA; May 2002; Available at http://www.nist.gov/director/prog-ofc/report02-3.pdf

[13] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, G. H. Leber, *Comparison of Physical and Software-Implemented Fault Injection Techniques*; IEEE Transactions on Computers, Volume 52 , Issue 9, pp. 1115-1133, September 2003