# Autonomous Parameter and Schedule Configuration for TDMA-based Communication Protocols such as FlexRay

Patrick Heinrich, Dirk Eilers, Rudi Knorr
Fraunhofer Institute for Communication Systems ESK
Munich, Germany
{forename.surname}@esk.fraunhofer.de

Markus Königer[1], Bernd Niehoff[2]
University of Applied Sciences Kempten,
Kempten (Allgäu), Germany
[1]markus.koeniger@gmx.de,
[2]niehoff@fh-kempten.de

***Abstract*— One goal of research activities is finding ways to manage the growing complexity of embedded systems using self-configuration methods. While autonomous configuration could potentially be used in safety-critical and real-time systems, the basic requirements are not yet in place. This paper will outline a concept for the real autonomous configuration of TDMA-based communication processes, which currently does not exist.**

**The paper initially addresses the TDMA-specific framework conditions and a potential solution. The issue of the mandatory a-priori known schedule is resolved using a generic schedule, because a simple method based on "free-slot-reserved-for-further-nodes" is not feasible. The most difficult part – the startup – was implemented through the generic schedule and an ID-based collision resolution process. To demonstrate the viability of the concept, the configuration method was implemented using a FlexRay communication system. This also satisfied the goal of eliminating the need for additional hardware and preserving the fault tolerant multimaster structure of the FlexRay system. The functionality of the concept was validated under different scenarios. The configuration times were analyzed, the results of which are also detailed here.**

## I. INTRODUCTION

Software has played an increasingly important role in automotive systems over the past two decades. With thousands of equipment combinations and variations, today's vehicles have become extremely complex. Electric mobility has also followed this trend because many of the previously mechanical components are now controlled via electronics and software. The additional safety-critical functions demanded by E-mobility are also being implemented via software, placing even more urgency on the development of methods to manage these increasingly complex applications.

The first step in reducing this complexity involves eliminating the need to manually configure each system during the design phase. This challenging process is error-prone and time consuming, even for experts. On the one hand it limits the variability of the system. On the other, manual configuration is an impractical approach given the enormous number of system variations possible in current and future automotive systems. Autonomous configuration offers a viable way to reduce this complexity. Research has previously been carried out on this and related topics [1] [2] [3] [4].

Apart from reducing complexity, autonomous configuration has the advantage of increasing component reusability and driving down development costs. Enhanced features such as self-healing are also possible. However, the pros and cons of concepts such as those outlined in [5] should be evaluated.

Self-configuration offers similar benefits in the area of safety-critical applications, although other framework conditions and more stringent requirements apply. Ensuring a correct configuration is absolutely necessary within this area. The problem is that this requires extensive effort. One way to resolve this issue is to configure the system at assembly time, thus allowing for detection of incorrect configurations without impacting the user. This would also permit the use of external devices such as a computer. The downside is that self-healing could not be utilized during runtime. The objective here is to demonstrate the feasibility of self-configuration within time division multiple access (TDMA) communication.

The first step in creating autonomous configuration is enabling the self-configuration of TDMA-based communication protocols, which have specific issues that must be resolved. These issues are outlined in section II. A concept for solving them are discussed in section III. The configuration method itself is explained in detail in section III-B. The performance of the configuration concept was analyzed using a FlexRay cluster. The paper then concludes with a discussion of the results and future outlook.

## II. TDMA-BASED COMMUNICATION

Most of the TDMA-based communication protocols were designed to enable hard real-time communication, such as for safety-critical applications like brake- and steer-by-wire. Communication is based on time slots and organized in periodically-reoccurring cycles. The cycle is occasionally divided into different areas, such as a static and a dynamic segment in the FlexRay protocol [6]. However, all TDMA-based protocols require a common schedule that is known to all nodes. This schedule must be a-priori knowledge to all nodes, otherwise message transmission cannot be carried out. Even listening to existing communication is not supported by standard hardware. The dynamic segment of FlexRay is unusable in this scenario as well because flexible TDMA requires an a-priori known schedule anyway.

Schedules are currently created at development time and are designed to meet the communication requirements of

all components, even if they are not included in all vehicle variants. Some concepts attempt to reduce those limitations by switching between different predefined schedules [7] [8]. This merely reduces the limitations and does not eliminate them.

Another typical feature of TDMA-based communication is a specialized communication controller (CC), which manages the correct startup and communication process according to the specifications. This is required because of the need for highly accurate timing. This places a very heavy load on the processor, an undesirable scenario for the host controller. For this reason it is not possible to enable the self-configuration properties by changing the protocol implementation. A configuration process must utilize the CC ”as is” and without the cost of additional hardware.

These specialized communication controllers also have several protocol-specific properties, which must be taken into account during implementation of the self-configuration method. One of the most common properties involves restarting the communication process following a schedule change. In summary, any configuration concept must be able to handle the following characteristics of TDMA-based communication:

- TDMA-based communication functions only when all controllers operate with the same global schedule.
- Listening to existing communication is not possible without knowing the schedule.
- The communication process must end before the changing the schedule
- No additional hardware should be required.

FlexRay exhibits these features, in addition to others such as:

- The requirement for a minimum of two nodes with the same schedule and cold start capability.

FlexRay was selected as the protocol for the self-configuration mechanism because it is one of the most important TDMA-based protocols in the automotive industry. The following sections contain a description of the configuration mode and an evaluation of the FlexRay implementation.

## III. SELF-CONFIGURATION MECHANISM

### A. Principles of the Self-Configuration Mechanism

The goal of the self-configuration mechanism is to reach a state where a valid, application-specific schedule is active. To achieve this, the current application-specific communication must stop and enter configuration mode. If there is no valid schedule, the controller must enter configuration mode directly. A new valid schedule is created within that mode, which includes all nodes. An application-specific schedule is available if the self-configuration process is error-free. Communication then starts up after the controller leaves self-configuration mode.

The self-configuration mechanism should not require additional hardware in order to function. Hence the described mechanism has to work with standard FlexRay components, which is the reason why FPGA-based solutions as described at [9] and [10] are not feasible here.

Figure 1 shows the different states of the self-configuration mechanism, which are described in more detail in the following sections.

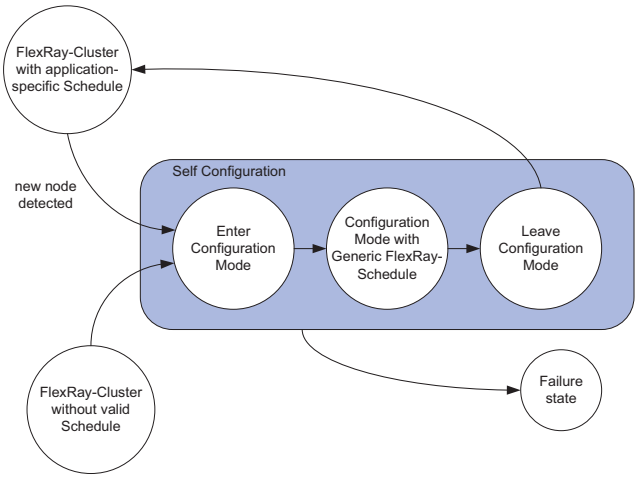


Fig. 1. Overview of the self-configuration mechanism

*1) Scenarios for Entering Configuration Mode:* The self-configuration mechanism can be initiated under two different scenarios.

In the first scenario, a new node must be added to a FlexRay cluster that is already running a communications process based on a common schedule. This new node is unaware of the common schedule used by the other nodes. In addition, the cluster has no reserved slots for the new node. The self-configuration mechanism is triggered after the new node attaches to the cluster.

In the second scenario, several nodes are connected, but they don’t have a valid schedule with which to run the communication process. The self-configuration mechanism must calculate a valid schedule for all nodes, which will enable node-wide communication once the new schedule is in place.

*2) Entering Configuration Mode:* As outlined in the previous section, there are two scenarios that activate configuration mode.

In the first example, a new node is connected to the cluster when the cluster already has a schedule for its application-specific operation. In the second case, the cluster is powered up and there is no schedule stored in the system for normal operation. This leads to a cluster startup in the configuration mode. The self-configuration mechanism relies on a master-slave principle at the application level. The master initiates the switch from normal operation to configuration mode. Both scenarios are described in more detail in section III-B.

The master requires input in order to know that configuration mode must be activated. The input can originate from a user or a node. A completely different approach, which is presented in [9] and [10], determines the FlexRay parameters from the current communication process. A fast signal processing unit is required to evaluate the FlexRay signals. The authors of both documents use FPGAs for this purpose. Once the parameters and schedule have been determined, communication with the master can be established as long as a free slot is available for the new node.

In some cases, activation of the configuration node must be avoided during normal operation of a FlexRay cluster. This includes instances of high processor loads within one or more nodes or a safety-critical state of operation. To resolve this issue, a process was introduced that requires every registered node to provide an acknowledgement before the schedule is switched to configuration mode.

*3) Configuration Mode:* As previously indicated, configuration mode is completely controlled by one master node. Each FlexRay node can theoretically act as the master. The remaining slave nodes respond in accordance with the requirements of the master. This master node has no special function in normal operation mode apart from triggering configuration mode. This means the fault tolerant multimaster structure of FlexRay is not disabled by the self-configuration concept.

As mentioned in section II, a commonly known schedule is required to facilitate FlexRay-based communication. A generic schedule was developed with this in mind. It is stored in every node and enables the negotiation of a new schedule for normal, application-specific operation. When this generic schedule is used for communication, the system is in configuration mode. Configuration mode is used to autonomously configure the cluster or add new nodes to an existing communication process. The first step is to register the nodes as slaves in the master node, which is required because the system does not know which nodes are connected to the cluster. A registered slave obtains a transmit slot, which it uses to communicate its bandwidth requirements. The corresponding schedule is calculated with aid of the previously received information.

*4) Leaving Configuration Mode:* The configuration process is completed by transmitting the calculated schedule from the master to all other nodes. The schedule is verified by the slaves and communication is then restarted with the new schedule.

### B. The Configuration Mode in Detail

This section describes the generic schedule required for the initial communication between the FlexRay nodes, as well the individual self-configuration steps. These are depicted in Figure 2.

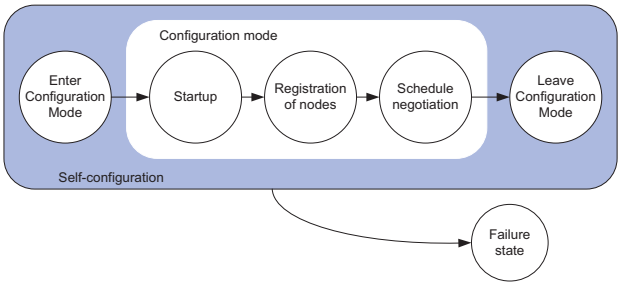


Fig. 2. Processed steps in the configuration mode

*1) Generic Schedule:* Communication in configuration mode is completely controlled by one master node. This affects the layout of the generic configuration mode schedule. The schedule is depicted in Figure 3 and described below.

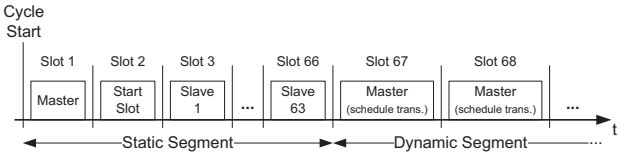


Fig. 3. Design of the generic schedule for the configuration mode

There is one transmit slot for every slave node in the static segment. The FlexRay protocol specification [6] does not limit the number of nodes in one cluster. However, the FlexRay requirements specification [11] calls for support of at least 64 nodes in a cluster. For this reason the maximum number of slave nodes is currently set to 63. Including the master node, the generic schedule currently supports 64 nodes, but this number can be easily expanded. If less than 64 nodes are connected to the cluster, some slave slots remain empty. A slave node can use its slot to communicate its bandwidth requirements or respond to requests from the master, e.g. acknowledge messages.

One slot in the static segment is reserved for starting slaves. This slot is called the ”start slot.” Each connected slave starts its transmission in this slot, since the nodes do not know which slot to use for communicating its bandwidth requirements. Once registration is complete, the master shifts the node to one of the reserved slots. The start slot then becomes idle again and is used by the next starting slave. Although this design requires $n$ cycles for integrating $n$ new nodes, the advantage is that it simplifies startup since fewer collisions occur.

The master reserves one transmit slot in the static segment for transferring various control information, e.g. requests to the slaves or the management of the start slot. The master also uses the dynamic segment, which allows flexible data transmission. That makes it suitable for transferring the calculated schedule to the slaves. If a TDMA does not support a dynamic segment, the transfer can be carried out within the static segment.

The system is also unaware of the cluster bus topology up front. It may even be dynamic, since nodes can be

added or removed during operation. The application-specific schedules can easily deal with this situation because the FlexRay schedule parameters adapt to the current topology. The generic configuration schedule does not change. For this reason, the FlexRay parameters related to the bus topology must be selected accordingly. This means the schedule must tolerate the maximum line length and the maximum number of stars that are specified.

*2) Startup:* The cluster is started in configuration mode only when there is no schedule stored in the nodes (except the generic schedule for the configuration mode). If a schedule is available, the cluster will start with the rules for that particular schedule. Figure 4 shows the startup of a slave. This section describes the startup when no schedule is stored.

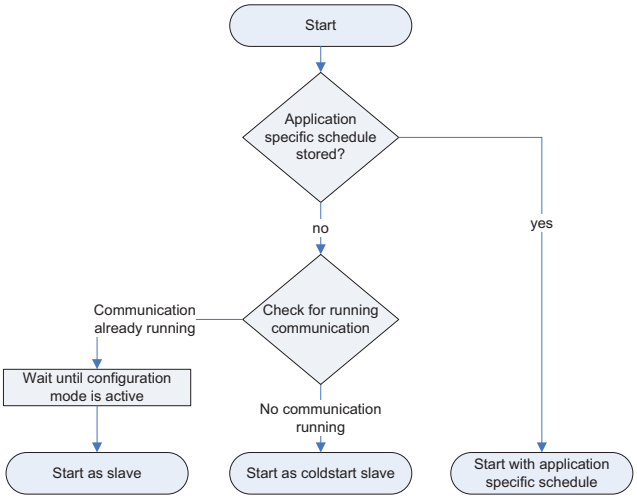


Fig. 4. Startup of slaves

At least two cold start nodes are required to establish a FlexRay communication process. Cold start nodes can establish FlexRay communication, but those nodes without cold start functionality can only join an existing communication process. The first cold start node is the master. Given that the self-configuration algorithm is controlled by the master, there is no reason to establish an initial communication without it. The second node is called the cold start slave. Any slave may become a cold start slave as long as no communication process is running. This is true for communication within the configuration mode or any other communication process. Moreover, there can be no other cold start slaves attempting to activate the cluster. Once this has been verified, the slave takes over the cold start function and initiates the communication process in conjunction with the master.

*3) Node Registration:* Node registration is carried out in configuration mode. As pointed out earlier, starting slaves begin their transmission at the start slot.

The master controls access to the start slot. It sends ”start slot free” or ”start slot occupied” messages for this purpose. This is done to prevent the starting slaves from transmitting frames in the start slot if the frames of another node are detected.

A starting slave sends a frame with a unique identifier in the start slot. This identifier must be unique within every node in the cluster, because if two slaves simultaneously transmit, their frames will overlap in the start slot. If the clocks are properly synchronized and the spatial distance is small, this could occur without errors, but would lead to problems with the registration process. Overlapping frames with different payloads will cause errors. The unique identifiers are transmitted in the frames. This guarantees a unique payload in every node. The master node monitors the start slot and if an error is detected, it tells the start nodes to delay transmission and retry.

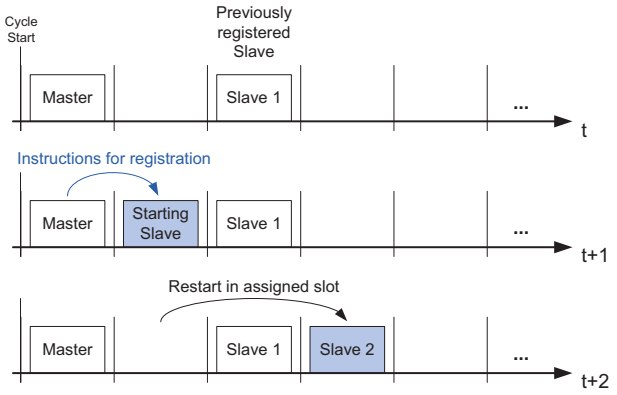


Fig. 5. Registration of slaves in configuration mode

The registration can continue as long as no errors have been detected. The starting slave is told which slot of the generic schedule to use. It then restarts within the assigned slot. The slave is now registered with the cluster. This procedure is repeated until all starting slaves are registered. Figure 5 shows the registration of a second slave node after one slave has already registered. The current starting slave receives its instructions and then restarts as *Slave 2* within its assigned slot.

*4) Schedule Negotiation:* FlexRay communication controllers need three essential pieces of information: the global FlexRay parameters, the local FlexRay parameters and the distribution of the TDMA slots to specific nodes. With this information, a valid schedule for an application-specific operation can be calculated. This paper does not focus on the development of sophisticated or optimized methods for calculating schedules. As a result, the calculation method was kept simple.

The global FlexRay parameters are calculated by the master. The general properties and bandwidth requirements of the nodes, as well as some basic settings, are required as input. The parameters received from the nodes include (1) priority of the node, (2) bandwidth requirements for the static segment, (3) bandwidth requirements for the dynamic segment, (4) maximum allowed cycle time (5) startup and clock synchronization behavior. The basic settings needed

to calculate the schedule include the maximum line length, the maximum number of active stars used in the cluster and startup or wakeup parameters. To determine the schedule, the duration of a FlexRay cycle is set to the shortest allowed cycle time based on the parameters received from the node. The duration of the static segment with corresponding length and quantity of the static slots is then calculated using the node parameters. The remainder of the cycle is identified as the dynamic segment (with respect to the network idle time, which is required for clock synchronization). A check is performed to ensure that the dynamic segment is long enough to meet the bandwidth requirements for the dynamic segment of the nodes. The global FlexRay parameters are then transmitted to all slaves after the calculation has been performed.

The master distributes the slots using a simple rule. High-priority nodes receive the low-number slots. This applies to the static and dynamic segments. The number of slots assigned to a node is extracted from the required bandwidth for both segments. Each node needs to know its own slot numbers and those of the partner nodes. The master broadcasts the slot distribution information to all nodes. A unique identifier is assigned to each node slot. The node compares the unique identifier, which it receives with a slot number, to its own unique identifier. If there is a match, the respective slot is assigned for transmission to this node. If not, the node determines if the slot can be used for receive mode. The unique identifiers for all partner nodes are stored in a list. They must be application-specific and unique. If the unique identifier does not match, it is compared with the list of communication partners. If a match occurs, the slot is allocated for receiving. Additionally, every slave must be able to receive messages from the master slot. Otherwise the master cannot order a switch to configuration mode. Likewise, the master must be able to receive messages in order for the slaves to acknowledge the switch to configuration mode. In the current implementation, the key slots are always used to exchange these messages, but this is not mandatory.

The local FlexRay parameters are calculated individually in every node. Global FlexRay parameters, node-specific settings and the distribution of the slots serve as input. Examples of node-specific parameters are the key slot ID, the used channels and the clock correction parameters. The sequence of the schedule negotiation is illustrated in Figure 6.

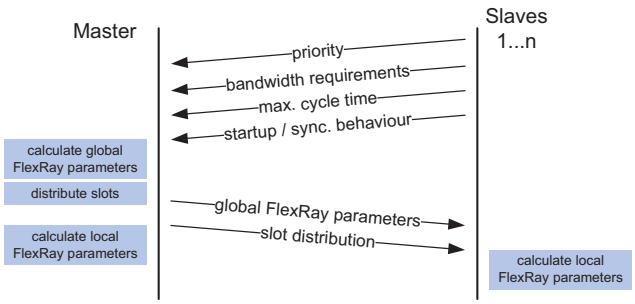


Fig. 6. Sequence of the schedule negotiation

### C. Performance

A configuration mechanism is normally analyzed using the required configuration time, but as mentioned in section I, ensuring a correct result within safety-critical systems is more important.

To analyze the performance, a FlexRay cluster of seven Freescale MPC5554 Controllers with an external FlexRay CC (Freescale MFR4300) and transceivers (NXP TJA1080) were connected using a bus topology. The Controller runs on the embedded configurable operating system (eCos). FlexRay communication is analyzed using the IXXAT FlexRay CCM, which permits the analysis to be carried out without knowing the current schedule. The FlexRay cluster is configured using channel A at 10 Mbps.

Because TDMA is being used, the configuration time consists of components that are near-constant and linear to the number of connected nodes. The number of nodes is needed only for the negotiation during the node registration process and of course for the schedule calculation. The configuration times nevertheless vary according to the different start-up times of the nodes. The indicated times are based on a single self-configuration process with seven nodes.

The components of the configuration time include:

- $t_{start}$ = time required to start configuration mode. The FlexRay cc takes roughly 80 ms to switch from an existing communication. The time to switch from cold start to configuration mode is also fixed. This takes longer however, because the system has to arbitrate the selection of the cold start slave. The time to directly startup the FlexRay cluster is about 500 ms.
- $t_{col}$ = collision resolution time. This is required for coordinating the starting nodes. It depends on the number of connected nodes and is necessary because of the number of unknown nodes. Collision resolution takes about 20 ms using seven nodes.
- $t_{reg}$ = the time to register a node at the master multiplied by the number of unregistered nodes $n$. The registration time per node is about 90 ms.
- $t_{calc}$ = the time to calculate the new schedule, which is about 20 ms using the simple algorithm presented here.
- $t_{tr}$ = the time needed to transmit the calculated schedule to the slaves. This is based on the bandwidth requirements of the nodes, not the number of nodes. A schedule with seven nodes and low bandwidth requirements requires roughly 20 ms.
- $t_{end}$ = the time needed to end configuration mode and switch to the new schedule. This time is provided by the FlexRay CC and is about 28 ms.

That results in a total time for configuration $t_{conf}$, represented by equation 1.

$$t_{conf}(n) = t_{start} + n \cdot (t_{col}(n) + t_{reg}) + t_{calc} + t_{tr} + t_{end} \quad (1)$$

The functionality was tested under various scenarios. The single configuration times were measured using the

IXXAT FlexRay analyzer hardware. While different types of collision resolution or optimization of the transmissions between the master and the slaves could potentially reduce the times even further, this research focused on the proof of concept.

## IV. CONCLUSION AND FUTURE WORK

The aim of this research was to create a self-configuration method for TDMA-based communication protocols with its specific framework conditions. The concept was implemented for a FlexRay communication system using standard components. With this system, the parameters and bandwidth requirements in FlexRay communication processes can be adapted. The research illustrates the capability to integrate new nodes into an existing communication process and to configure a cluster without an application-specific schedule.

The research demonstrated how and when the system enters and leaves configuration mode. The project also provided a detailed description of the generic schedule and the processes within configuration mode, which include (1) startup, if no specific application schedule is available (2) registration of new nodes (3) schedule negotiation. The concept was then tested under different scenarios, followed by an explanation of the configuration times.

The results were sufficient for demonstrating proof of concept within a laboratory environment. Practical implementation would involve situations such as asynchronous starts and dropped or corrupted messages. These issues will be addressed in future research activities. There is also the potential to optimize (reduce) the configuration time. A more complex algorithm to satisfy the demands of online schedule calculation is also a subject for future work.

Some FlexRay clusters use star topologies with active stars. This would require evaluating the impact of active stars on this concept due to the collision isolation mechanisms provided by such topologies. Another possible improvement is replacement of the current master-slave principle. This could be achieved with a moderate amount of effort by dynamically determining the master and slave functionality. This means one of the nodes declares itself as master, provided no other node has assumed this role. Expanding the current implementation to a fully distributed autonomous configuration solution, without master and slaves but equal nodes, would be a challenge.

An important point for future work is the addition of an application-specific configuration. This involves identifying and broadcasting the functionality of new nodes to the other nodes in the cluster. The current configuration mechanism enables communication between nodes. In real applications however, the system must be enhanced with information regarding the content of the data.

## REFERENCES


[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] K. Geihs, "Selbst-adaptive software," *Informatik-Spektrum*, vol. 31, no. 2, pp. 133–145, 2008.

[3] R. P. Würtz, *Organic computing*. Berlin: Springer, 2008.

[4] G. Weiss, M. Zeller, and D. Eilers, "Towards automotive embedded systems with self-x properties," in *New Trends and Developments in Automotive System Engineering*, pp. 411–432, 2011.

[5] K. Klobedanz, G. B. Defo, W. Mueller, and T. Kerstan, "Distributed coordination of task migration for fault-tolerant FlexRay networks," in *International Symposium on Industrial Embedded Systems (SIES)*, pp. 79 – 87, IEEE, 2010.

[6] FlexRay Consortium, "Flexray communications system protocol specification version 2.1 revision a," 2005.

[7] M. Mitzlaff, R. Kapitza, and W. Schroeder-Preikschat, "Enabling mode changes in a distributed automotive system," in *European Dependable Computing Conference*, pp. 75 – 78, ACM, 2010.

[8] R. Brendle, T. Streichert, D. Koch, C. Haubelt, and J. Teich, "Dynamic reconfiguration of FlexRay schedules for response time reduction in asynchronous fault-tolerant networks," in *Proceedings of the 21st international conference on Architecture of computing systems (ARCS)*, pp. 117 – 129, 2008.

[9] M. Heinz, V. Hoess, and K. D. Mueller-Glaser, "Physical layer extraction of FlexRay configuration parameters," in *International Symposium on Rapid System Prototyping*, pp. 173 – 180, IEEE/IFIP, 2009.

[10] E. Armengaud, A. Steininger, and M. Horauer, "Automatic parameter identification in FlexRay based automotive communication networks," in *IEEE Conference on Emerging Technologies and Factory Automation ETFA*, pp. 897 – 904, IEEE, 2006.

[11] FlexRay Consortium, "Flexray requirements specification v2.1," 2005.