Switched FlexRay: Increasing the Effective Bandwidth and Safety of FlexRay Networks

Paul Milbredt
AUDI AG
I/EE-81
Email: paul.milbredt@audi.de

Bart Vermeulen
NXP Semiconductors
Central R&D / AAL / DiSA
Email: bart.vermeulen@nxp.com

Gökhan Tabanoglu
Volkswagen AG
EEFA/4
Email: goekhan.tabanoglu@volkswagen.de

Martin Lukasiewycz
Institute for Real-Time Computer Systems,
TU Munich, Germany
Email: martin.lukasiewycz@rcs.ei.tum.de

Abstract

With the continued demand for more and innovative functions in series automobiles, significantly higher datarates and more reliable communication are necessary than traditional automotive bus systems, e.g., the controller area network (CAN), provide. In this paper, we present a novel hardware device for FlexRay networks, which splits the bus into separate branches and operates as a selective central switch.

The proposed device is capable of increasing both the available bandwidth and safety of existing FlexRay networks. The bandwidth is increased by enabling the scheduling of multiple messages in parallel on disjoint sets of branches. The device can furthermore serve as a central bus guardian (CBG), protecting the majority of the participants on the network from so-called babbling idiots and short circuits, by isolating a faulty component on its respective branch.

The proposed device is fully compatible with the existing FlexRay specifications and transparently extends a system, requiring no modifications to the existing FlexRay bus participants. The architectural challenges of a FlexRay switch/CBG, its subsequent proof-of-concept implementation on an FPGA-platform, and our experimental results are described in detail.

1 Introduction

Automotive electronic systems have become the key driver of innovation in series automobiles. Numerous *electronic control units* (ECUs) enable a boost in functionality with the goal to increase the safety, comfort, reliability, and environmental efficiency of modern series automobiles. In particular, this technological innovation comprises (1) an increase in the accuracy of existing functions, (2) the replacement of mechanical and/or hydraulic control systems with electronic control systems, and (3) the introduction of new functionality, such as drive- or brake-by-wire, adaptive cruise control, fuel and traction control systems, intelligent parking assist, and electronic stability control.

The *Controller Area Network* (CAN)[5] bus is the predominant communication system in automotive networks. It uses an event-triggered architecture and can reach bandwidths up to 500 Kbps. To increase the effective bandwidth of CAN bus systems, a common procedure is to use gateways to assemble multi-cluster systems from several, separate CAN buses. In these multi-cluster systems, the ECUs communicate with each other via multiple CAN buses, and one or more gateways. In the case of Audi[8], the topology is always based on a single, central gateway[6]. This central gateway interconnects all buses present in the car, and as such, can route data from and to all buses, making it the preferred location to implement those functions that are purely software-based and require inputs that can be broadcast. With the continued increase in bandwidth demands for new functions, this procedure quickly becomes impractical. This has already resulted in real-world networks with many different CAN buses[6] that are very difficult to maintain.

The *FlexRay* communication system[3] has been developed by an industrial consortium as a solution to this problem. FlexRay offers a time-triggered architecture, a bandwidth up to 10 Mbps, and support for different topologies (linear bus, star and hybrid topologies). Due to its time-triggered nature, FlexRay will become the defacto bus standard for *X-by-wire* systems that require a high level of safety.

The three innovations described above however require significantly higher effective bandwidth and more reliable communication than these traditional automotive bus systems can provide. In this paper, we therefore introduce the concept of a *switched* FlexRay bus and a new device, the FlexRay *switch*. This device increases the effective bandwidth, available on a network, by operating as a central switch, synchronizing to the FlexRay cluster clock, and physically splits the bus into several branches. It thereby enables the scheduling and transmission of multiple messages in parallel on disjoint sets of branches

This device furthermore increases safety by behaving as a *Central Bus Guardian* (CBG), and protecting the data communicated on the network against nodes that develop a run-time error. So-called babbling idiots and short circuits are isolated to a single, specific branch.

The switch device is fully compatible with the existing FlexRay specifications, and transparently extends a system, requiring no modifications to the other FlexRay bus participants.

The remainder of this paper is organized as follows. Section 2 provides a short introduction to FlexRay, and identifies the potential benefits of using a switch device in a FlexRay topology, comparing the development of FlexRay networks with the development of Ethernet. In Section 3, the operating principle of the switched FlexRay bus is explained in more detail and the high-level, functional requirements for a central switch device are described. Our proof-of-concept implementation and use of the switch for a real-life automotive application are described in Section 4, while experimental results are presented in Section 5. This paper concludes with Section 6.

2 FlexRay Basics and a Comparison with Ethernet

In the 1980s and the early 1990s, 10BASE2[4] was the dominant variant of Ethernet. It used a simple bus topology, where all nodes were connected with short stubs in line by T-connectors. At the end of the bus, a resistor of 50 Ω had to be used to terminate the bus for avoiding reflections. This topology is identical to the one used in CAN networks, and is widely used within the automotive industry. Minor differences exists, for instance in cable types (twisted-pair instead of coaxial cables), cable lengths, and the termination resistor's value.

2.1 FlexRay Topologies

The FlexRay specification offers numerous different topologies. The simplest (and cheapest) is the single bus topology, which is equal to the CAN topology, but has higher restrictions regarding the length of the bus and the stubs.

FlexRay further adds support for a passive star topology, as a special case of a single bus topology. In a passive star topology, the stubs are directly connected to each other and the length of the bus line is zero. In an active star topology, every node is directly connected to a central device, i.e., the *active star*, via a branch. This analog device refreshes the incoming electrical signal and broadcasts it to all other branches. The active star in a FlexRay network resembles a *hub* in the Ethernet domain. Active stars and Ethernet hubs may be cascaded, i.e., one may connect an active star directly to another. In the automotive domain, a single line may connect an active star at the front of the car with an active star at the rear. Each active star distributes the signal locally. The FlexRay *Electrical Physical Layer* (EPL) Specification [1] allows a maximum of two active stars between any pair of nodes. This is also the most robust topology against *electromagnetic interference* (EMI), due to the fact that no additional nodes can influence the physical layer of any existing branch.

At the ECU containing the active star, many branches have to be connected. On the one hand, a central device with many branches is more expensive than a single transceiver. On the other hand, a simple bus is cheaper, but not as robust against EMI. Therefore the best compromise is typically a hybrid topology. In the case of Audi, currently one active star and a maximum of three nodes per branch are allowed [8].

2.2 Bus access scheme

FlexRay uses a *time-division multiple access* (TDMA) approach. Every node has one or more unique time intervals, called *slots*, in which it is allowed to send messages. These slots are assigned at automobile design time in a schedule. The layout of such a schedule is shown in Figure 1.

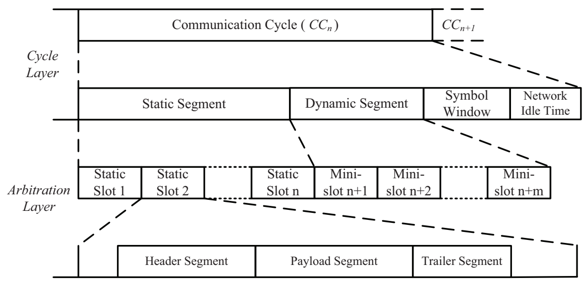

Figure 1. FlexRay timing hierarchy

For the *static segment*, all slots are of equal size. Each static slot may only be assigned to a single node. One node may however use more than one slot. A FlexRay frame is sent in a slot after the *action point offset*, which is a small amount of time to tolerate some jitter on the local clock and the propagation delay of each node.

Flexible TDMA (FTDMA) is used during the *dynamic segment*. This segment is divided in so-called *minislots*, which occupy only a small amount of time. Only if a node actually wants to transmit data it starts sending data in a minislot. This minislot is then extended to a full slot, the duration of which is dependent on the amount of data that has to be sent.

FlexRay offers a maximum of 64 communication cycles. During the dynamic segment the minislots may be multiplexed over time, meaning that the same slot may be assigned to a different node in each communication cycle.

To avoid conflicts, the schedule has to be designed with care.

The *symbol window* may optionally be used to transmit a media test symbol to verify that a bus guardian blocks communication. The *network idle time* (NIT) is used to calculate and apply clock correction.

2.3 Clock synchronization

FlexRay was designed as a fault-tolerant protocol, i.e., the global timebase is reconstructed locally in each node instead of using a global “timing master”. FlexRay therefore defines a subset of nodes as *sync nodes*. These nodes are responsible for sending one of their static frames with the sync bit set in the header.

Every node in the cluster builds up a table, which contains the difference between the expected arrival time and the actual arrival time of each of the sync frames in the local time base. The *fault-tolerant midpoint* (FTM) algorithm is used every odd cycle to calculate a rate and an offset correction value based on this table. The offset correction value is then applied during the NIT. It shifts the local start of the next communication cycle. The rate correction value shortens or lengthens the duration of some macroticks during the next two communication cycles.

Before applying the corrections an ECU can check the calculated correction values against pre-configured limits. If the values are inside the limits the ECU assumes it is in synchronous mode and the correction is applied. If one of the values is outside of the limits, the ECU assumes it is in asynchronous mode and subsequently enters a mode where no data is sent and will probably be forced to reintegrate by the application.

2.4 Wakeup and startup

When a FlexRay cluster is in a low-power (sleep) mode and communication is required, first the synchronous mode has to be established in a fault-tolerant way. The first step towards communication is the wakeup phase. A node, which is woken up by any external event, must wakeup the rest of the FlexRay cluster. It distributes a simple wakeup pattern on the network, after ensuring that no ongoing communication would be disturbed.

After this wakeup phase, the startup phase establishes the common, synchronized view of the global time. A *coldstart* node starts transmitting a *collision avoidance symbol* (CAS) to inform all nodes that it will start transmitting. Afterwards, it transmits sync frames on the network. These frames have the startup bit set in their header. All other nodes synchronize their local clocks to the coldstart node. As soon as more sync nodes start transmitting, all nodes adjust their clocks depending on the available sync frames.

2.5 Switching in Ethernet

Ethernet is usually used as an event-triggered system, therefore messages are not known a-priori and switching has to be done based on the frame content. Switches are the de-facto standard of current Ethernet based system. If an Ethernet switch receives a message it forwards it according to the unique MAC addresses, which are part of the header of the frame. It stores the MAC address of the sender and the incoming port in a table. If it does not know the port of the receiver yet, it forwards the frame to all branches. If an entry with the receivers MAC address exists, it forwards the frame only to the corresponding port.

In principle, two operation modes exist: (1) *store-and-forward*, and (2) *cut-through*. In store-and-forward mode, the complete frame is received (and stored) by the switch, then analyzed and forwarded only, if the checksum is correct. In cut-through mode, the frame is forwarded immediately after the MAC addresses were received. This mode saves memory space in the switch and offers lower latency. However, also erroneous frames may be forwarded and occupy bandwidth.

Both modes add a delay of many bits to the frame, which is not acceptable in a time-triggered system, such as FlexRay, because the frame would violate the boundaries of its timeslot. In FlexRay, switching will have to depend on the specific slot and cycle to minimize the delay to the 150 ns allowed by the specification[1]. At 10 MBps, this delay corresponds to only 1.5 bits.

2.6 Central bus guardian architectures

TTP/C[7] offers special star coupler devices called *central bus guardians*, which add fault-containment to each branch. A preliminary, draft specification of a FlexRay Central Bus Guardian also exists[2]. These architectures protect faultless branches from messages sent by faulty branches. In a central switch device, which already requires schedule knowledge as described above, these concepts can also be supported.

3 FlexRay Switch Concept

In this section, we present a proposal on how to adopt the concept of a switch in a FlexRay network. Since we already have another central device (the active star), the main goal is to replace this active star by a switch.

3.1 Bus Access Scheme

The most obvious difference with Ethernet is the use of an TDMA (and FTDMA during the dynamic segment) approach in FlexRay in contrast to the use of the *Carrier Sense Multiple Access / Collision Detection* (CSMA/CD) approach in Ethernet. In order to guarantee correct communication, FlexRay frames sent by a node have to arrive at the receivers within a specific amount of time. The FlexRay EPL specification[1] allows a maximum delay of 250 ns for active star devices, which corresponds to 2.5 bits at a speed of 10 Mbps. This constraint prevents content-based switching for FlexRay, in contrast to packet based switches in other networks. Instead the activation of the actual switching paths has to be done in each slot, according to the switch communication schedule, and therefore has to be *time-triggered*. This implies that the switch itself also has to perform clock synchronization to get a notion of the global time of the FlexRay system it is a part of.

3.1.1 Switching during the Static Segment

The switching schedule consists of the timing information, i.e., the slot and cycle counter, and the switching paths. Each path is a set of *one* source branch and one or more destination branches. A branch can be disconnected during some slots by not having a switching entry during these slots (branch local communication). As an example, a switching entry for one specific slot may be represented by Matrix M in Equation 1 (using 4 branches in total).

$$M = \begin{pmatrix} 0 & s_{2,1} & s_{3,1} & s_{4,1} \\ s_{1,2} & 0 & s_{3,2} & s_{4,2} \\ s_{1,3} & s_{2,3} & 0 & s_{4,3} \\ s_{1,4} & s_{2,4} & s_{3,4} & 0 \end{pmatrix} \tag{1}$$

The n-th row is the destination branch n. Each destination branch may have zero source branches (branch local communication) or one source branch (switched communication). The source branch i is marked by a "1" in the i-th column. Of course, no more than one source branch may exist during one slot, hence the constraint in Equation 2.

$$\forall x \in n : \sum_{i=1}^{n} s_{i,x} = 1 \vee 0 \tag{2}$$

Whenever a branch has another branch as its switched source, it can not be the source of another branch, which leads to the constraint in Equation 3

$$\forall x \in n : \sum_{i=1}^{n} s_{x,i} > 0 \Rightarrow \sum_{j=1}^{n} s_{j,x} = 0 \tag{3}$$

One complete switching entry e_i for the static segment consists of the slot ID S_i, the base cycle C_i, the cycle repetition r_i with $r_i \in \{1, 2, 4, 8, 16, 32, 64\}$, and the switch matrix M_i, and can be represented by the 4-tuple e_i, given in Equation 4.

$$e_i = \{S_i, C_i, r_i, M_i\} \tag{4}$$

When N represent the total number of switching entries, the complete schedule can be represented as the set E of all switching entries in Equation 5.

$$E = \bigcup_{i=1}^{N} e_i \tag{5}$$

3.1.2 Switching during the Dynamic Segment

During the dynamic segment, switching becomes more complicated due to the FTDMA approach. The whole segment is divided into equally-sized minislots. If a node has no data to transmit, the bus remains in the idle state and with the next minislot also the next (logical) slot starts. As nodes may send frames with an arbitrary payload length (up to 254 bytes), the next (logical) slot starts after the end of the frame (including the idle times to compensate clock drifts) *and* together with the next minislot. This is done by all senders and receivers, so that all nodes share the same view of the slot counter.

Method 1: No switching A trivial method to treat the dynamic segment is to not switch in it. In contrast to the static segment, the switch would have to behave like an active star. In contrast to the static segment, one branch can be an input *and* an output in different minislots.

Method 2: Broadcast groups Instead of broadcasting all branches, also groups of branches can be built that share all communication. In addition, this can be different for each communication cycle. The configuration can be similar to that of a static slot. The only difference is that during the dynamic segment a branch can be both an input and an output. In other words, the switch would support half-duplex, undirected switching. This grouping ensures, that within each group and each cycle, the slot counter still matches, but it does not need to match for the entire cluster, as the groups are seperated (see Figure 2).

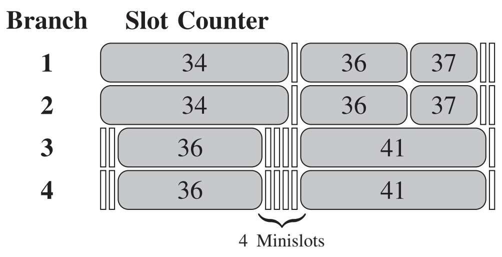

Figure 2. Switched Dynamic Segment Example (Method 2), branches 1 and 2 form one group, branches 3 and 4 another

Method 3: Building and refining broadcast groups It is possible, but impossible to foresee, that the slot counters of different groups may match up again at a later point in time. Therefore, the next possible refinement strategy is to reduce the set of branches (without adding other branches) which belong to a group over time within the dynamic segment of one cycle. For configuration purpose, there should exist an entry for every refinement point in time (slot), which must contain a strict subset of branches. It should be up to the system designer to verify correctness of the entries. This requires a lot more configuration memory, but can theoretically help to gain more bandwidth.

We propose to use the second method. It does not add too much complexity in comparison to Method 1 and requires far less configuration memory than Method 3, while still offering possibilities to increase the effective bandwidth. Overall, the amount of achievable bandwidth still depends to a large extend on the message schedule.

3.2 Safety Features

The switch concept also enables safety features by only activating communication paths for those slots that are assigned to a node on a branch. It thereby automatically blocks a node from sending across the switch in slots that are not assigned to its branch. This behavior is basically the same as specified for a central bus guardian in its synchronous state[2]. As the switch decodes frames to perform clock synchronization, three additional features can be implemented during its synchronous operation: (1) abort frames with incorrect header and/or frame CRCs (e.g., by just stopping transmission), (2) do not forward frames that start too late within a slot when it is sure, that it will violate the slot boundaries, and (3) do not forward sync frames to a branch, which has produced numerous errors within a certain time frame.

This last feature is comparable to the punishment feature of the central bus guardian, and will lead to an asynchronous faulty branch or node. The application of the node will have to handle this error, and maybe, reset the node before reintegration. If the fault is still present, the node may be completely disconnected by our switch.

Changing the standard FlexRay broadcasting to uni- or multicasting, one can achieve security and privacy of the data sent in those slots. A switched message will not be received by nodes that are not connected to the branches involved.

3.3 Enhanced Wakeup

In a FlexRay system with wakeup capabilities, at least one node has to be woken up by an external source. Then this node boots, configures its FlexRay controller, and triggers it to enter its wakeup state. The controller checks the bus whether the bus is idle for at least two cycles. When this is the case, it sends a wakeup pattern. If there are active stars in the system, the first symbols of the pattern wake up the (first) active star, and the remaining symbols are forwarded by this active star. The more active stars are cascaded in a system, the more wakeup symbols per pattern have to be configured.

If a (faulty) node falls asleep during normal communication, there is currently no possibility to wake this node up again. There is a chance that a node will wake up by normal communication frames. This is however not guaranteed to work. A wakeup pattern may be emulated within the payload of a frame as a possible solution to wake this node up again. However this emulation can never be perfect, as a frame requires both a header and byte start sequences that have fixed bit patterns.

Assume now a topology where our switch is the (only) central component. When the switch detects a silent branch (e.g., no receptions from this branch and only transmissions to this branch for a duration of two communication cycles), the switch can stop forwarding any frames to this branch, generate a number of wakeup patterns exclusively on this branch, and finally transmit all frames to this branch again for startup.

Of course, the switch still cannot detect silent nodes but it can detect silent branches. If only one node is connected to a branch, then this is equivalent. Usually however, this is only the case in a safety-critical environment. One can add the information, which node sends which frame, but this will increase the amount of memory needed in the switch (see Section 3.1). In highly safety-critical systems, where a pure star topology is used, the information of branches and nodes is equal, hence, no additional configuration is needed.

3.4 Enhanced Startup

The FlexRay startup has a few known issues, which may prevent the system from transitioning to the normal active state. Next to the classical babbling idiot failure, which prevents coldstart nodes from even starting transmission of startup frames, there exist several possible scenarios of cyclically resetting nodes, which begin startup, then reset and startup again. Other nodes start their integration, but before it is successful, the starting node resets. They then start the integration phase again and before they might start the transmission of startup frames, the original node starts again.

Another possible impact is the occurrence of cliques, which are groups of nodes with a slightly different notion of the global time. Being synchronized within the clique enables a node to communicate with nodes within the same clique, but not with nodes in other cliques, as the different notion of global time causes frames to be semantically incorrect.

Our switch may protect the startup phase like a central bus guardian would[2]. Alternatively the switch could start the system up itself, but this would require the switch to become an active component in the system, as opposed to a passive observer and forwarder.

4 Proof-of-concept Implementation

A prototype switch was programmed in VHDL at the registered-transfer level (RTL), and implemented on a FPGA, which is part of an automotive platform containing eight physical FlexRay transceivers. The behavior of the switch is primarily defined by the configured schedule, which controls routing during synchronous operation. In the asynchronous operation mode, e.g., startup, the switch behaves as a Central Bus Guardian device, protecting the cluster from faulty nodes.

The switch is divided into several levels of hierarchy (see Figure 3). Large parts of the design are generic, making it possible for the user to enable or disable certain functions prior to synthesis to save hardware resources. The cluster, node and bus guardian parameters, as well as the switching schedule can be configured during run time by the host via the host interface. The Host Data Translator (HDT) transforms incoming commands and data from the platform-specific host interface to the internally used format. The Digital Clock Management (DCM) generates the required 80 MHz clock for eight times oversampling at 10 Mbps and bit strobing. Since the rest of the design is independent of the external interfaces, it is possible to port the switch to other platforms without making changes to the main configuration and schedule execution modules.

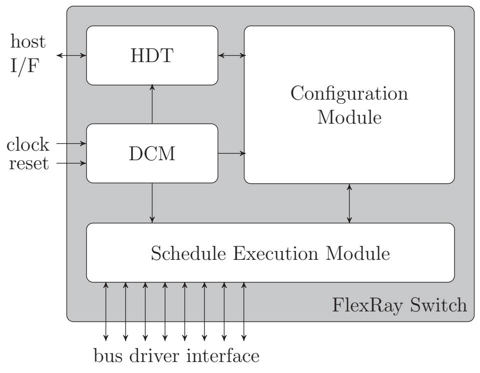

Figure 3. Schematic of the switch top-level.

4.1 Configuration module

The switch configuration takes place in the configuration module, which executes host commands and contains an internal memory to store the FlexRay parameters and the switching schedule. The schedule memory is configured to store two separate schedules so that on-the-fly rescheduling without resynchronization is possible.

The switch schedule memory is organized as follows: The set E, given by Equation 5, is sorted by the slot ID S_i. In a worst case, for each of the 64 possible cycles, a different switch matrix M_i may be necessary. Hence, we can have 64 entries with $r_i = 1$ and the corresponding C_i are $0, 1 \ldots 63$. Remembering that the switch acts time-triggered, so it will have to find the right entry in E before the slot starts. As it is a sorted list, the switch can move to the first entry during the current slot and search the next entry within the maximum of 64 entries during this slot. Likewise, the last slot of the previous cycle is used to find the entry for slot 1.

Changing the cluster, node or bus guardian parameters requires a resynchronization of the switch to the cluster. The switching entry information for the current static slot are passed on to the schedule execution module.

4.2 Schedule execution module

The Schedule Execution (SE) module implements the clock synchronization algorithm, coding and decoding of communication elements, and controls the interconnection of the branches. The data path of a FlexRay frame can be seen in Figure 4.

The Coding And Decoding (CODEC) module extracts synchronization information from the incoming bitstreams and passes it on to the ClockSync module. This

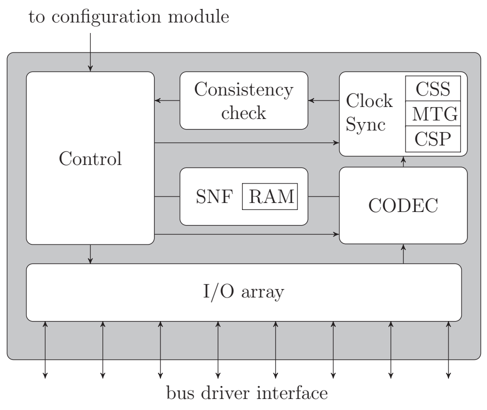

Figure 4. Schematic of the schedule execution module.

module controls the time base of the switch by implementing the FlexRay clock synchronization algorithm. It uses offset and rate correction compliant to the FlexRay specification to adapt its local view of the time to the global cluster time. Responsible for the startup of the clock is the clocksync startup submodule (CSS), which integrates upon a single coldstarter and passes the initial values for rate correction, slot- and cyclecounter to the macrotick generation process (MTG).

The MTG stores the current micro- and macrotick and updates the timebase accordingly. It informs the rest of the design about the start of communication cycle and segments, thus performing the media access control functions and eliminating the need for a separate module. Measurement of the deviation values and calculation of the correction values is done by the clock synchronization process (CSP). Beginning with the conclusion of the static segment, the correction values are determined using the fault-tolerant midpoint (FTM) algorithm.

The data path through the I/O array of the FlexRay frames is implemented using combinatorial logic only, reducing the amount of signal propagation delay introduced by the FPGA.

Table 1 shows the resource usage of the switch components after synthesis and mapping on a Xilinx Spartan 3E-1200. The implementation of the complex clocksync algorithm consumes the largest part of the allocated resources.

5 Experimental Results

For the first prototyping board we chose our existing FPGA board, developed by the University of Applied Science Technikum Vienna [10]. It offers a central FPGA (Xilinx Spartan 3-1500), to which 8 single FlexRay transceivers are directly connected. All VHDL modules have been tested at RTL by simulation. We have written testbenches for all modules, especially the clock synchronization process and the central bus guardian features. At

Table 1. FPGA resource usage

Module	Slices	Slice FF	LUTs	Area
Glitch Filter	18	15	37	0%
Consist. Chk	24	7	43	0%
SNF	75	82	145	0%
MDHI	86	61	162	0%
I/O array	149	102	279	1%
SE Control	546	288	998	6%
CODEC	602	429	1160	6%
Configuration	737	1022	1331	8%
ClockSync	1273	1230	2284	14%

the system level, we switched a couple of slots and verified correct switching behavior at the nodes, whether they received correct data or not.

5.1 Experiment 1: Switching Delay

As the FlexRay specification allows a maximum delay of 150 ns for the active star and we want to keep compatibility, the switch has to stay within this bound. Our first experiment is the delay from the input of one transceiver of the branch of the sending node to the output pin of one transceiver at one destination branch. The results are given in Table 2. The delay verification passed successfully, even despite the fact, that two single transceivers and an FPGA are not the optimal (one single chip) solution.

Table 2. Delay measurements

Delay	Value [ns]
Average	130.65
Minimum	122.0
Maximum	139.5
Std. Dev.	3.6816
No. of Measurements	100,000

5.2 Experiment 2: System stability

To test the influences on the system itself, another experiment was chosen. We built a system, consisting of eight FlexRay nodes connected in a pure active star topology. Four of the eight nodes were sync nodes, like it is in the Audi A8 [8] for example.

During the runtime of the system, we measured the delay of the sync frames of the four sync nodes from the cycle start during 45 hours. This delay is a good indication for the accuracy of the clock correction calculation.

After exchanging the active star with our switch prototype, the same measurement was done (see Table 3). During the whole measurement, the network stayed synchronous and four monitoring tools could not detect any FlexRay protocol violations (boundary violation etc.).

Obviously, our switch has an influence on the system, but this is just due to the fact, that it is a prototype system in contrast to the COTS active star device and therefore has longer delays between the branches. But it still operates within the FlexRay specification and the additional delay is acceptable.

Table 3. Measurement of the sync frame deviations

Sync Node	σ of Sync Frame Delay [ns]	
	active star	switch
Node 1	23.54	31.12
Node 2	23.71	32.00
Node 3	23.65	31.45
Node 4	23.64	32.55

5.3 Potential of a switched schedule

After validating the correct switching behavior as described we designed two FlexRay schedules to demonstrate the potential of the switched communication. For this purpose two FlexRay clusters were designed as a star topology with four branches. In the first one a standard active star was used to couple the branches whereas in the second one a FlexRay Switch was used. Both schedules, the switched one and the standard one, were designed using the same example where 8 ECUs are sending an overall number of approximately 2700 signals grouped into 220 PDUs (Protocol Data Units). Since the same example was also used in [9] to show the applicability of two novel methods for standard FlexRay schedules, we adopted the results for the purpose of comparison.

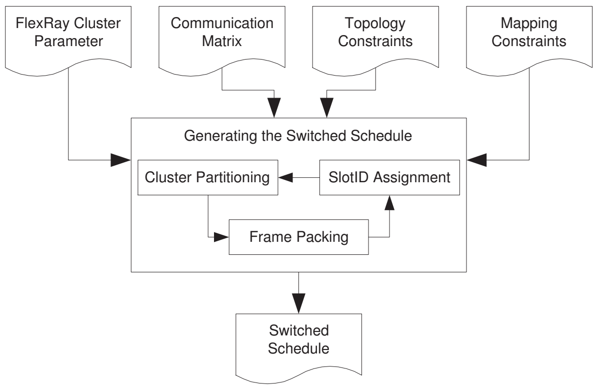

Figure 5. Enhanced scheduling process

To generate the switched schedule, the standard schedule generation process had to be extended (see Figure 5). In addition to the so-called frame packing step a partitioning step has to be performed where ECUs are mapped to branches. Since the communication in a switched FlexRay cluster is not based upon a broadcast scheme anymore, the partitioning step has an crucial influence to the efficient slot utilization in a switched schedule. However, when partitioning the ECUs to branches we aim to maximize the number of not-allocated slots on each branch as well as to minimize the number of used branches, where the first objective takes priority over the second.

The slot utilization on each branch for both schedules is depicted in Table 4. Since the communication scheme in a standard schedule is based on a broadcast scheme the number of allocated slots on each branch is identical. For this example, we used a total number of 27 slots, which had to be allocated on each branch. In the switched schedule, the number of allocated slots differ from branch to branch since the switching scheme enables the parallelization of independent communication links. In consideration of the objectives only a total number of 17 slots on branch 1 and 16 slots on branch 3, respectively, must be allocated for the switched schedule. Since no ECU is mapped to branch 2 and 4 no slot needs to be allocated on these branches. The results show that by using a switched cluster instead of a standard cluster a significant number of slots can be saved which in turn improves the extensibility of a FlexRay cluster.

Table 4. Slot utilization in a standard and a switched schedule

cluster	**# of allocated slots per branch**			
type	branch 1	br. 2	br. 3	br. 4
standard	27	27	27	27
switched	17	0	16	0

6 Conclusion

We showed that a time-triggered switch can be successfully applied in a FlexRay based system. The constraints given in the FlexRay specifications can be met, even with a first prototype built using an FPGA and COTS transceivers.

The most bandwidth gains are possible when the communication elements only need to be unicast, or multicast to a limited number of receivers. In a system where (nearly) all communication must be received by all nodes, the switch can of course not gain any bandwidth at all, However even then, it still adds fault containment and safety to the system. In practice, it is a mixture of both and a switch can gain a lot bandwidth, if the physical architecture is designed appropriate. This means that the system design becomes more complex since there is a direct relationship between the schedule and the physical network architecture.

To get ready for series production into cars, an application specific integrated circuit (ASIC) implementation becomes necessary. Current active stars do already have a configuration interface (usually Serial Peripheral Interface (SPI)), and would only require a small amount of digital logic and a clock interface.

The scheduling of the system is even today a hard-to-solve problem [9] and it would get even more complex. This is part of our ongoing research and we will publish results when they become available.

Acknowledgement

This document is based on the INDEXYS project in the framework of the ARTEMIS program. The work has been funded in part by the ARTEMIS Joint Undertaking and National Funding Agencies of Austria, Germany, Hungary and the Netherlands under the funding ID ARTEMIS-2008-1-100021. The responsibility for the content rests with the authors.

References

[1] FlexRay Consortium. *FlexRay Communications System, Electrical Physical Layer Specification, Version 2.1*, May 2005.

[2] FlexRay Consortium. *FlexRay Communications System, Preliminary Central Bus Guardian Specification, Version 2.0.9*, December 2005.

[3] FlexRay Consortium. *FlexRay Communications System, Protocol Specification Version 2.1 Revision A*, Dec. 2005.

[4] IEEE. IEEE Standards for Local Area Networks: Supplements to Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *ANSI/IEEE Std 802.3a,b,c, and e-1988*, 1987.

[5] ISO. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling. 11898-1:2003.

[6] J. Koetz. Elektronik Architekturen über Baureihen - Chancen und Risiken. In *12. Internationaler Kongress Elektronik im Kraftfahrzeug*, Baden-Baden, 6.-7.10.2005. VDI. In German.

[7] H. Kopetz. A Comparison of TTP/C and FlexRay. Technical report, Technische Universität Wien, Austria, 2001.

[8] G. Linn, W. Sichert, P. Milbredt, and G. Kistler. Serieneinführung eines weckfähigen FlexRay-Bussystems. *Elektronik Automotive*, Sonerausgabe Audi A8:102–104, Februar 2010. In German.

[9] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. FlexRay Schedule Optimization of the Static Segment. In *Proceedings of the 7th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 363–372, 2009.

[10] P. Milbredt, A. Steininger, and M. Horauer. Automated Testing of FlexRay Clusters for System Inconsistencies in Automotive Networks. In *4th IEEE International Symposium on Electronic Design, Test & Applications*, Hong Kong, China, January 2008.