Startup error detection and containment to improve the robustness of hybrid FlexRay networks

Alexander Kordes
Faculty IV / ETI / OSDS
University of Siegen
Siegen, Germany

Bart Vermeulen
Central R&D
NXP Semiconductors
The Netherlands

Abhijit Deb
Central R&D
NXP Semiconductors
The Netherlands

Michael G. Wahl
Faculty IV / ETI / MSE
University of Siegen
Siegen, Germany

***Abstract*—The research and development on in-vehicle networks (IVNs) is driven by two main requirements: bandwidth and robustness. In this paper we address the robustness requirement. We focus on FlexRay IVNs that are used for safety-critical applications. We analyze and discuss faults that may affect the startup and operation of a FlexRay network. These failures may not only occur during the startup phase of the vehicle, but they may also happen due to a bus problem that requires the bus to be reinitialized during normal operation. Here any startup failure leads to a critical situation like a brake system failure.**

The fault scenarios we discuss in this paper are the resetting leading coldstart node (RLCN), the deaf coldstart node (DCN), and the babbling idiot (BI). These faults are described in literature, but neither the precise behavior of all involved nodes, nor a clear solution is provided to contain their impact. The idea of a bus guardian (BG) is given in a draft specification of the FlexRay consortium, but no details are given. In this paper, we extend on these ideas by investigating and implementing a detailed (BG) concept, based on our fault analysis. We subsequently evaluate the successful containment of the three fault types in simulation. We also quantify the chip area cost of our solution.

I. Introduction

Robust communication is a stringent requirement for safety-critical applications on in-vehicle networks (IVNs), e.g., when they are used for x-by-wire applications such as brake-by-wire [1]. Unfortunately, the perfect fail-safe network has not yet been defined. The FlexRay network [2], [3] is one of the IVNs that is used to connect electronic controller nodes. Conceptually, each node contains an electronic control unit (ECU), sensors, actuators, and a bus driver (BD) to transmit and receive data. Typically, the ECU implements the communication controller (CC), which realizes the FlexRay protocol. FlexRay has been designed with safety-critical applications in mind. It is one of the most fail-safe network protocols known [1]. Still, under very special circumstances a failure may occur. One critical phase is the network startup process. A failure in this phase can cause the network to not start at all. If a transient error occurs during driving, it may be necessary to reinitialize the IVN communication. An error occurring at this point in time can be critical. This is why we focus in this paper on the possible problems in the startup phase of the FlexRay IVN.

In literature, as discussed in more detail in Sections III and IV, three different failure scenarios are described, which may prevent a FlexRay IVN from starting up. These are the resetting leading coldstart node (RLCN), the deaf coldstart node (DCN), and the babbling idiot (BI). They are based on mathematical models and do not reflect the precise timing of all involved nodes during the presence of a fault. The FlexRay consortium published a draft specification, which outlines a central bus guardian (CBG). Its tasks are to observe the traffic on the FlexRay IVN, identify potential faults, and prevent a failure of the IVN through the containment of the faulty node. This draft specification however does not provide enough guidelines to realize this bus guardian (BG) functionality. In this paper, we analyze the three types of faults, and detail how a central and a local FlexRay BG can be implemented to recognize them and take appropriate action to prevent these faults from affecting the startup of a FlexRay IVN.

This paper is organized as follows: we describe the relevant parts of the FlexRay startup process in Section II. In Section III we state the problem consisting of these three known failure scenarios. The related work is discussed in Section IV. In Section V, we present the proposed detectors and containment methods based on our failure analysis. We also discuss their hardware implementation and integration in an existing FlexRay switch design. The effectiveness of these BGs and their implementation costs are reviewed in Section VI. We conclude this paper in Section VII.

II. FlexRay Startup process

The FlexRay protocol [2] partitions time into consecutive communication cycles, where each cycle consists of a static segment, a dynamic segment, a symbol window, and a network idle time (NIT) interval. The static segment is further divided into constant length time division multiple access (TDMA) slots, called static slots. In each static slot, one node can send a frame. These frames consist of a header, a payload, and a trailer. The assignment of frames to slots is specified at design time in a communication schedule. Typically, the ECU of each FlexRay node contains a CC, which realizes the FlexRay protocol. The nodes may be connected on a single bus; one node on each branch of a central star component; or using a hybrid topology that contains one or multiple nodes on each branch of a star component. The process of initiating a network startup is called a coldstart. The nodes that are configured to initiate a startup are called the coldstart nodes

978-3-9815370-2-4/DATE14/©2014 EDAA

(CNs). All CNs first listen to the IVN for two communication cycles. If no other node transmits a valid frame header or a collision avoidance symbol (CAS) in this period, one or more CNs transmit a CAS to initiate startup. In the special case in which more than one CN transmit a CAS at the same time, the nodes involved cannot decode the CAS of the other node(s). This is because the FlexRay protocol specification specifies that nodes cannot listen to the IVN while they transmit. The designer of the FlexRay IVN assigns each CN a unique so-called key slot, in which this node transmits its startup frame. Hence, the node with the lowest configured key slot transmits its frame first.

The FlexRay protocol furthermore specifies that all nodes select the node that transmits its frame in the earliest TDMA slot to be the leading coldstart node (LCN). The other nodes decode the frame header from the LCN, and abort their own startup attempt to join the on-going startup process. These CNs become known as following CNs. The FlexRay protocol specification defines the signals, frames, and the timing that are necessary to establish the time base for communication.

The LCN transmits startup frames in five consecutive communication cycles to synchronize the following CNs. In the absence of errors, each synchronized following CN answers in the fifth communication cycle with a synchronization frame, after it has synchronized. The nodes therefore count correctly-decoded frames starting from an even cycle. The frame header contains the cycle number for this purpose. If no other CN is answering, the LCN stops transmitting for one communication cycle and aborts its coldstart attempt. It subsequently performs another coldstart attempt, consisting of another set of five startup frames in consecutive cycles. Every CN is initially configured with a number of possible coldstart attempts. This number is decremented on each attempt. To startup a FlexRay IVN, a CN needs to have at least two coldstart attempts left. The first coldstart attempt is used to transmit the CAS. Other nodes join the communication when they decode two startup frames per cycle in two consecutive cycles. Afterwards, the FlexRay IVN is in normal operation mode and the data communication starts.

III. Problem statement

A FlexRay IVN can be prevented from starting up or from communicating after startup by several, specific failure scenarios. Earlier studies mention the following three scenarios: RLCN [4]–[6], DCN [6], [7] and BI [8], [9]. In each of these scenarios, the failure prevents one or more FlexRay nodes from behaving fault-free. These failures need to be analyzed, detected and handled.

A. Resetting leading coldstart node

If the LCN is periodically reset due to a failure, then it can be prevented from transmitting messages in four consecutive communication cycles. The reset of the LCN causes the following CNs to abort the startup phase [5]. While the CNs reinitialize the startup process, the reset node takes the lead again and so forth. The network experiences a startup failure [4]. We call this node an RLCN. In the worst-case, this periodic reset occurs indefinitely [4]. In the case of a FlexRay IVN, the four consecutive frames in the first four communication cycles of a startup attempt are transmitted in the *coldstart collision resolution* state of a node.

B. Deaf coldstart node

The DCN failure, also called an incoming link failure, prevents the affected node from listening to the IVN [1], [2], [6], [7]. This causes a startup problem when the node is one of three CNs and overwrites parts of the header of the LCN with its CAS and subsequently every trailer of the third CN, which becomes the next LCN. This causes noise on the IVN instead. Hence, startup frames from the LCN cannot be decoded as valid. All CNs lose their startup attempts and stop transmitting, causing a startup failure.

C. Babbling idiot

The BI failure scenario is defined as a node transmitting outside of its specified time interval [8]. By transmitting noise or frames at unspecified time intervals in a FlexRay IVN, it is possible that the BI interrupts the startup process as well as any ongoing communication after startup. A FlexRay IVN with a BI can therefore lose synchronization, if the BI sends data or when other nodes can no longer transmit valid data.

IV. Related work

For the detection and possible containment of failures on an IVN, the authors of [10] describe a so-called BG solution. This is a device, which observes the ongoing communication of a transmitting device in a network. When the observed device generates a failure, the BG disconnects the device from the IVN to contain its failure. The first application of this concept to TDMA-based communication, without the highly-redundant aspect, is described in [11]. The BG can disable the communication when it detects faulty behavior from the observed node. Therefore, the BG has to know the TDMA schedule of the observed node. When the node violates the timing requirements, it is silenced by deactivating its BD. This approach is called fail-silence.

For FlexRay, a local bus guardian (LBG) concept is described in a preliminary specification [12]. This specification only provides the BG concept, but does not provide any implementation details. It uses most of the functionality of a FlexRay CC. To separate the BG from the local CC, it is necessary for the BG to have its own clock synchronization process (CSP). Therefore, an ECU contains two CCs, one for the node and one for the LBG. This separation makes the BG independent from possible failures in the observed CC and provides it with its own local view of the global time. The CC of the BG is not used to transmit any frames.

A BI failure scenario can be partly avoided in a time-triggered communication system with the concept presented in [11]. The authors describe that the node is set in a fail-silent mode, when the BG detects communication from the observed node at incorrect points in time. This concept is

extended in [12] to additionally detect content errors of the frame header. The authors of [11] however indicate that their approach can only prevent a special form of this failure, called the babbling CAS idiot that periodically transmits a CAS. The endless startup scenario describes a CN, which periodically initiates the coldstart sequence. This can refer to the RLCN scenario, but it is not discussed in more detail. The authors of [12] describe that the LBG could detect and contain such a failure scenario.

The FlexRay consortium also published the concept of a CBG [5]. This CBG is also a device with a functional subset of a FlexRay CC. Nodes are connected directly or via a bus on several branches to a CBG. In general, it follows the concept of the LBG. It cannot transmit frames itself. The CBG integrates and observes the FlexRay communication and uses a CC to determine the communication cycle number, segment, and slot numbers. The schedule of critical nodes, such as the coldstart and synchronization nodes, is stored in a CBG to protect the critical functions [13]. The CBG limits the bandwidth of the connected nodes or branches when failures are detected.

This conceptual CBG can also be used to contain an RLCN in a FlexRay IVN [5]. The CBG punishes the branch after a configurable number of faults, whenever an RLCN error is detected. An active branch is detected by a race-arbitration state. The first transmitting node is allowed to broadcast to all other nodes. The other branches are forced to listen until the end of the transmission or an error is detected. This prevents a BI from sending at incorrect points in time. If an error is detected and assigned to a branch, the CBG punishes the branch for the configurable period of time.

The related work described above focuses on an LBG or a CBG as an approach to increase the robustness of an IVN. The BG specifications are however preliminary and do not focus on specific error detection or the implementation of a BG for a specific IVN. In this paper, we present details of our implementation of a CBG and an LBG. These implementations and their evaluation are part of a larger project. Some early analysis results on a CBG have previously been published in [14]. These results however needed to be updated and extended as a result of a more extensive evaluation, which resulting in new insights and subsequent architectural changes to the design of the CBG. In this process, we also evaluated the consequences for the implementation of a LBG. This has, among others, led to significantly different values for the silicon area cost of our detectors. The effects of all these changes are described in more detail in Section VI. For completeness and to be able to properly position this new work, we first summarize our analysis results below and extend them with details on our CBG and LBG implementations.

V. Bus Guardian Analysis and Implementation

We describe in this section our analysis and implementation of both a FlexRay CBG and a FlexRay LBG. Both BGs have to detect and handle the three startup failures described previously. We first summarize the options to detect and contain these failures. For this, we use the models of the RLCN, DCN, and BI failure types available in literature. We subsequently provide implementation details on how these detectors can be integrated in a FlexRay IVN.

A. RLCN detector

The analysis of the RLCN has been done by scrutinizing the first 11 communication cycles of the startup process. This analysis led to a succession of 19 identified phases $P1$ to $P19$, as shown in Fig. 1. A phase is a time interval in which a reset of the RLCN has the same effect. The process starts with the CAS symbol, followed by specific frames required for the initialization of the IVN. Critical phases are e.g. $P2$ and $P10$. In $P3$ and $P7$, all CNs lose one coldstart attempt per communication cycle. Non-critical phases usually cause another CN than the RLCN to take the lead and to successfully start the network. In the worst case ($P13$), the completion of the startup process is delayed with the duration of all coldstart attempts added to the time the RLCN takes to calculate the result of its synchronization. This phase is outside of the *coldstart collision resolution* state [5]. Our RLCN detector monitors the IVN and tracks the progress through these phases. It stops monitoring as soon as a coldstart attempt is successful and transitions back to the start state to wait for a CAS indicating a new coldstart attempt.

Our central detector stores the branch information of the active branch when a communication element (CE) start is decoded. This is the case for the first frame. When all CNs start at the same time, the node with the first assigned key slot takes the lead (refer to Section II). Therefore the first critical phase starts after decoding the header of the frame sent by the RLCN. Our detector implementation extracts the frame identifier (id) of the RLCN, thereby identifying the node in the IVN. It signals all detectable failure cases. The decoded frame id is used to check if an RLCN is trying to startup the network for a second time. Another indicator for an RLCN is when a frame is expected and not received after the time specified in the specification ("tSecondFrame"). All other identified reset scenarios cause corrupted frames, because the RLCN resets while it transmits a frame. The bus is set to idle when no node is transmitting anymore. Therefore, the detector checks the header and trailer for a decoded channel idle recognition point (CHIRP). When the detector decodes it, it will signal an RLCN failure.

To contain the failure, the RLCN is identified by the arguments of the RLCN error signal, the frame id, and the branch information. The LBG sets the misbehaving node to fail-silent, whereas the CBG sets the identified branch to fail-silent. The other CNs are no longer affected by the RLCN,

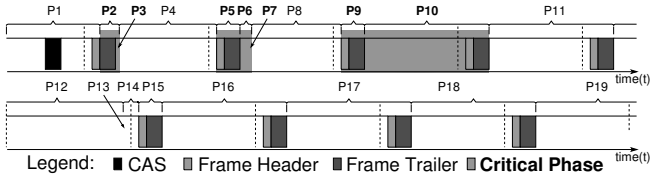

Fig. 1. RLCN phases

and one of them can subsequently take the lead and start up the network. Hence, it is necessary to configure the network in such a way that two CNs are not affected by the containment. The containment of the RLCN is released after a normal startup. The RLCN subsequently behaves like a following CN and does not impact the communication any more.

B. DCN detector

We analyzed the four known failure modes of a deaf node that affect the startup process. First, a DCN is not listening to the IVN. It tries to start the network up by sending out a CAS and transmitting startup frames. Because it cannot decode valid frames, it waits one cycle after each coldstart attempt until it runs out of coldstart attempts and subsequently remains quiet. We analyzed the second and third DCN failure mode of [1] and [6] and concluded that these modes do not follow the FlexRay protocol specification. We therefore do not consider these two failures modes for the implementation of our DCN detector. A further study, however, led us to another DCN failure mode [14]. This failure consists of three CNs with the minimal configured startup attempts of two. This mode was confirmed later on in [7]. We decided to consider this failure mode for our DCN detector, even though it can be circumvented during the design of the FlexRay IVN, by configuring the CNs with a number of startup attempts that is larger than two.

Our DCN detector observes the startup of the IVN from the first CAS until the network is started up. A started network consists at least of three startup frames decoded in the correct schedule of a communication cycle. If the CE is the start of a frame, the detector decodes it and checks for decoding errors. A decoding error during a frame is signaled when one of the sequences Frame Start Sequence, Byte Start Sequence, Frame End Sequence, or when one of the cyclic redundancy check (CRC) checks is corrupted. In contrast to a FlexRay node, our detector does not stop decoding the frame after a decoding error is detected, but it continues to check the next sequence. If the check returns another decoding error, the frame was overwritten by another node and the detector flags this as a DCN error.

The detector knows the following sequence for the second check, because it gets the specific location of the first decoding error as an argument of the decoding error signal. Therefore, we extended the existing decoding error signal of the decoding process with a more specific error type. This narrows the decoding error of the first sequence down to the sequence type, the location in the frame and the first or second bit of the sequence, if it is a two bit sequence.

Our detector also reacts when a second CAS is transmitted during a time window "tAllowCAS." This indicates that the transmitting node has not reacted to the first CAS by aborting the coldstart attempt. Hence our detector signals a DCN error.

If the detector is used in a central device, it receives the active branch for each CE start and decoding error. This allows our detector to compare the active branches from a CE start with the ones from a decoding error. If the branches differ, then two nodes are communicating at the same time, and a DCN error is flagged. The information on the active branches is used by the containment method. When a branch with the DCN is identified, this containment method sets it to fail-silent. After the time the DCN needs to transmit all coldstart attempts the fail-silence is released. The DCN cannot affect the communication anymore and the network successfully starts up. This requires that the CNs are placed on different branches in a star or hybrid network.

C. BI detector

We restricted our analysis of the BI scenario to the static segment, in which the nodes communicate their synchronization information. We assume the network has successfully started up and all nodes are already synchronized. A BI can transmit data at any point in the schedule. If the idle phase between the frames is corrupted, other nodes try to decode a frame. A starting frame has not been decoded when no *CHIRP on A* event is decoded at the end of the idle phase. Decoding errors occur when the faulty node transmits data that collides with a frame. This causes receiving nodes to stop decoding the frame and wait for the next generated CE start event. A BI can also transmit correct frames in wrong slots. These frames are decoded by the other nodes and cause a content error, because the decoded frame id or cycle counter are incorrect. These frames are subsequently discarded. The other nodes may however lose synchronization when this process causes synchronization frames to be discarded. They restart and try to synchronize again.

The BI detector monitors the IVN for the correct schedule and possible collisions. It is started when the BG becomes synchronized. The idle phases on slot boundaries are defined by the FlexRay protocol parameters. The detector observes the IVN during these phases and flags an BI error if it decodes a CE. This is done in the static segment and the NIT. Frames are transmitted in a time window where a CE is allowed. The detector only reacts on content and decoding errors in those time windows. A content error indicates a wrong schedule and a decoding error indicates a collision. The detector is designed to increase an error counter if such an event is generated. The BI detector signals an BI error when the counter exceeds a configurable parameter.

To contain a BI failure, it is necessary for our detector to identify the misbehaving node and exclude it by itself from the bus communication. It is also possible to set a whole branch fail-silent. Therefore, the central device stores the active branches when it receives a CE start or an decoding error and transmits it as an argument of the BI error signal. In both cases the network has to be configured in a way to provide enough startup and synchronization nodes that are not affected by the containment, to possibly startup and synchronize the network after the impact of the BI. The contained node or branch has to be checked after some time. If the failure is not present anymore, the containment can be released.

D. BG implementations

In general, BG functionality can be placed locally, near the CC and bus driver of an ECU (LBG), or in a central device (CBG). Our detector modules are suited for both. The basis for our CBG and LBG implementations is an existing VHDL RTL design of a FleyRay Switch [15] with six branches. We made four key changes with respect to the existing FlexRay switch implementation to be able to add our BG functionality to it. First, we added branch identification logic for the case in which the BG functionality is used in a central device. This logic helps identify the branch with a faulty node and is based on the FlexRay CODEC process [2]. By leaving the unused CODEC output signals unconnected, the synthesis tool automatically removes any unused CODEC logic. Second, we added extra registers to be able to control and query our detectors from the outside of the BGs via a serial peripheral interface (SPI). Third, we made changes in the underlying router module that handles the communication between the six branches. With the updated router module, we can support race arbitration and bit reshaping [5], and can more easily constrain the timing during synthesis and future layout steps. Fourth, we have implemented the branch containment functionality, by adding extra masking logic in the signal router module of the switch, to contain a branch with a faulty node.

We derived the implementation of the LBG from our CBG implementation through a number of deconfiguration steps. First, a LBG does not require six branch inputs and outputs. Instead, we only support a single, local branch in our LBG implementation. We also removed the branch identification logic and parametrized the instantiation of our detectors to the local use case. This last step among others removes the logic in the detectors that is used to store the identity of the branch that initiated certain CEs. Furthermore, we removed all switching functionality, which includes the slot table addressing logic.

VI. Experimental results

In this section, we first evaluate the effectiveness of our BG implementation. We do this using the CBG implementation, as the CBG implementation contains a superset of the functionality of our LBG implementation. We subsequently evaluate the silicon area cost associated with the BG functionality, both for the CBG as well as for the LBG.

A. Failure Detection

The FlexRay network architecture used for our simulation is shown in Fig. 2. To verify our CBG functionality, we connected a configurable number of FlexRay nodes to its branches. These FlexRay nodes are configurable in their behavior and are implemented in SystemC [16]. We subsequently performed mixed-level simulations to verify the functionality of our FlexRay CBG for each failure type using four consecutive steps. First, we simulated the implementation without the presence of a failure and confirmed the correct behavior of the IVN. In our second step, one node was configured with one of the failure types. In the subsequent simulation, we confirmed that the associated detector successfully detects the failure

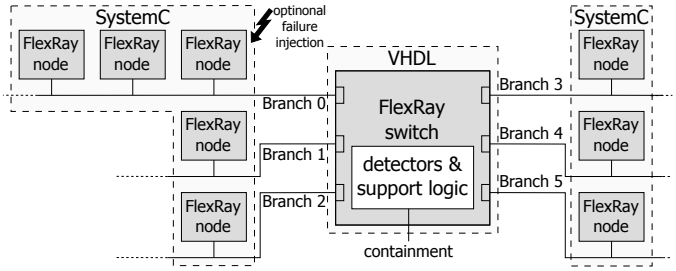

Fig. 2. Implementation of BG and SystemC nodes

and that the IVN is indeed affected with a startup failure. The containment of the branch with the failing node after this detection was validated in simulation as part of our third step. We successfully validated that in all cases, the IVN starts up after this containment. In the fourth and last step, we cancel the containment after the network has started up successfully without the failing node. We discovered that, in case of the RLCN and the DCN, the IVN is no longer affected by the failing node. However, the containment of a node affected by the BI failure cannot be released because the IVN cannot properly function with the BI present.

Example behavior of the implemented solution for an RLCN is depicted in Fig. 3. The CNs on the three branches of the IVN each try to become the leading CN by issuing a CAS. The RLCN takes the lead by transmitting the frame header. After the transmission of the frame trailer it resets (refer to $P2$ of Fig. 1). The RLCN node tries to take the lead for the second time, which is detected by our CBG as an RLCN error after the time "tAllowCAS." The containment functionality in this CBG blocks the branch with the RLCN, causing the failing node to become silent and allowing another node, CN 2, to take the lead and initialize the FlexRay IVN. Afterwards, CN 1 joins in in the second TDMA cycle, because both nodes started with a CAS. The network is started up later and the containment of the branch with the RLCN can be released. An example of a DCN scenario is depicted in Fig. 4. The DCN overwrites the frame from the CN 3 with a CAS. The collision between

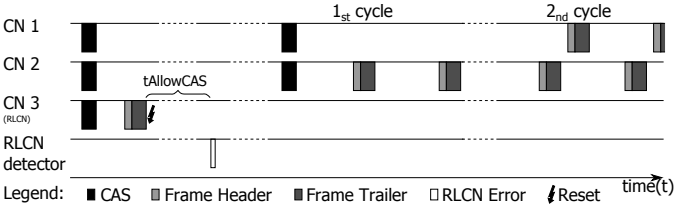

Fig. 3. Simulation of an RLCN failure scenario with containment

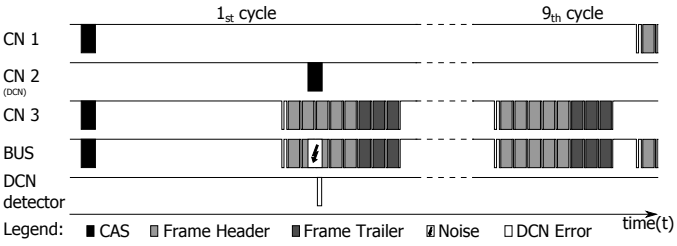

Fig. 4. Simulation of a DCN failure scenario with containment

the CAS and the frame header of CN 3 generates noise on the IVN, which is detected by our CBG. The containment functionality in our CBG sets the DCN to fail-silent and the network subsequently synchronizes with no problems in the ninth cycle. An example of a BI scenario is shown in Fig. 5. The startup process has been finished and CN 1 and CN 2 are sending frames as expected. The BI then sends noise in the slot boundary between the valid frames from CN 1 and CN 2. This is detected immediately by our CBG, causing the branch with the BI to be contained.

B. Silicon area cost

The silicon area costs for our CBG and LBG implementations are shown in Table I. These results have been obtained by synthesizing the VHDL RTL design files of our complete CBG and LBG, and comparing it with the original FlexRay switch. We used a commercially-available synthesis tool and an automotive-qualified cell library. All numbers have been normalized using a total CBG area set to 10,000 units for confidentiality reasons. The other logic entry in Table I refers to the baseline logic of the switch [15]. The difference between the area cost of our LBG with respect to our CBG is caused by the differences detailed in Section V-D. The area increase for the DCN detector in an LBG is caused by its ability to provide more error information in that case. Overall, we observe that the area cost of our CBG is approx. 32 % larger than our FlexRay switch and the area of our LBG is approx. 75 % of our switch.

VII. Conclusion

There are a few, rare but erroneous, situations in which a failure can prevent the FlexRay IVN from starting up. Our failure analysis showed that the startup process is not completely fault-tolerant. We reviewed and analyzed the RLCN, DCN and BI fault scenarios described in literature. The RLCN, the DCN, and the BI scenarios can prevent a FlexRay network from starting up under very specific conditions. We developed both central and local bus guardian implementations that generate fault information based on the observed behavior of all nodes of the IVN. These BGs use three different detectors to observe the FlexRay IVN and to identify the occurrence of these three failures. We have described the integration of these detectors into an existing FlexRay switch design. Our implementations extend this design to a CBG and an LBG. We subsequently verified their functionality through mixed-level simulations.

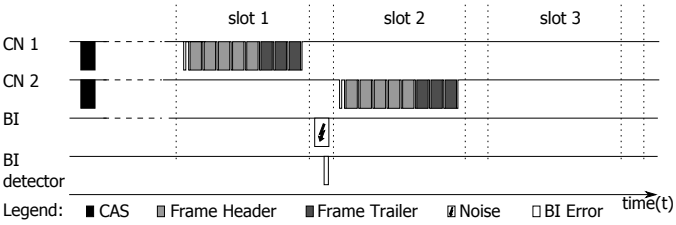

Fig. 5. Simulation of a BI failure scenario

TABLE I
Relative area cost of the detectors and their support logic

Module	CBG	LBG
RLCN detector	233	233
DCN detector	187	196
BI detector	43	43
Branch identification logic	1,974	0
Additional SPI registers	2	2
Other logic	7,560	7,104
Total	10,000	7,579

Our evaluation shows that our BGs are able to detect these three failure scenarios. We validated that the network does start up normally, when the branch with the identified faulty node is contained using our BG functionality. Synthesis results for our BGs show the required amount of silicon area for a FlexRay CBG and a FlexRay LBG. The area cost of a CBG was shown to be smaller than the area cost of two LBGs. This difference may translate into a different component cost, which the designer of a FlexRay IVN can take into account, when trading off component cost against fault protection.

Our next step is to validate our detectors in a real FlexRay IVN using an FPGA implementation of our BGs and to add run-time fault protection functionality.

References

[1] M. Rausch, *FlexRay: Grundlagen, Funktionsweise, Anwendung*. Hanser Fachbuchverlag, 2007.

[2] FlexRay Consortium, *FlexRay Protocol Specification, Version 3.0.1*, Oct. 2010. [Online]. Available: www.flexray.com

[3] ——, *Electrical Physical Layer Specification, Version 3.0.1*, Oct. 2010. [Online]. Available: www.flexray.com

[4] W. Steiner, "Model-Checking Studies of the FlexRay Startup Algorithm," TU Wien, Institut für Technische Informatik, Research Report, 2005.

[5] FlexRay Consortium, *Preliminary Central Bus Guardian Specification, Version 2.0.9*, Dec. 2005. [Online]. Available: www.flexray.com

[6] S. Cranen, "Model checking the FlexRay startup phase," TU Eindhoven, 2012, Computer Science Report 12-01, Jan. 2012.

[7] ——, "Model checking the FlexRay startup phase," in *17th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, ser. Lecture notes in computer science (LNCS), M. Stoelinga and R. Pinger, Eds., vol. 7437. Springer, August 2012, pp. 131–145.

[8] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*, ser. Real-time systems. New York: Springer, 2011.

[9] G. Buja, A. Zuccollo, and J. Pimentel, "Overcoming babbling-idiot failures in the FlexCAN architecture: a simple bus-guardian," in *10th IEEE conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 2, sept. 2005.

[10] A. Hopkins, T. Smith, and J. Lala, "FTMP 8212;A highly reliable fault-tolerant multiprocess for aircraft," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1221 – 1239, oct. 1978.

[11] C. Temple, "Avoiding the babbling-idiot failure in a time-triggered communication system," in *28th Annual Int. Symposium on Fault-Tolerant Computing, 1998. Digest of Papers*, june 1998.

[12] *Preliminary Node-Local Guardian Specification, Version 2.0.9*, FlexRay Consortium, Dec. 2005. [Online]. Available: www.flexray.com

[13] M. Dehbashi, V. Lari, S. G. Miremadi, and M. Shokrollah-Shirazi, "Fault Effects in FlexRay-Based Networks with Hybrid Topology," in *Proc. International Conference on Availability, Reliability and Security*, 2008.

[14] A. Kordes, B. Vermeulen, A. Deb, and M. Wahl, "Erhöhung der Robustheit von hybriden FlexRay-Netzwerken durch Erkennung und Eingrenzung von Laufzeitfehlern," in *4th VDE GMM-Fachtagung AmE "Automotive meets Electronics", Dortmund, Germany*, Feb. 2013.

[15] B. Vermeulen, J. Staschulat, M. Struck, and S. Lorenz, "Flexray Switch: More Bandwidth and Better Robustness in Flexray Networks," *ATZelektronik worldwide Edition*, vol. 5, 2010.

[16] M. Baumeister, P. Fuhrmann, and F. Armbruster, *Taking concept models from standardization to silicon*, Hanser Automotive Electronics + Systems, 2005.