EVALUATION OF BABBLING IDIOT FAILURES IN FLEXRAY-BASED NETWORKES*

Vahid Lari, Mehdi Dehbashi, Seyed Ghassem Miremadi, Mojtaba Amiri

Sharif University of Technology, Tehran, Iran

Abstract: This paper evaluates the error propagation and its effects in babbling idiot failure in a FlexRay-based network. The evaluation is based on about 35680 bit-flip fault injections inside different parts of the FlexRay communication controller. To do this, a FlexRay communication controller is modeled by Verilog HDL at the behavioral level. Then, this controller is exploited to setup a FlexRay-based network composed of four nodes. Nodes in this experiment are considered in two forms: 1) node without bus guardian, 2) node with bus guardian. The results of fault injection show that in first form about 4.57% of faults lead to the babbling idiot failures. Also in second form about 0.75% faults lead to babbling idiot failures. After this experiment and evaluation of the results, one improvement in the bus guardian is done. With this improvement, the results show that babbling idiot failures are eliminated completely.

Keywords: FlexRay protocol, Babbling idiot failure, Fault injection, Error propagation

1. INTRODUCTION

Safety in distributed systems such as automotive systems and avionics is of decisive importance due to system failures which may threat human life. In a distributed system, each node consists of three parts (Kopetz, 1998): 1) I/O part, 2) host part, and 3) communication controller. Among these three parts, the communication controller has a key role in the system operation.

In general, communication activities can be triggered either dynamically, in response to an event (event-triggered), or statically, at predetermined moments in time (time-triggered). Examples of time-triggered protocols are the SAFEbus (Hoyme and Driscoll, 1992), SPIDER (Miner, 2000), and Time-Triggered Protocol (TTP) (Kopetz and Bauer, 2003). The main drawback of the time-triggered protocols is their lack of flexibility (Pop, et al., 2006). Examples of event-triggered protocols are the Byteflight (Berwanger, et al., 2000) introduced by BMW Company for automotive applications, CAN (Bosch, 1991), LonWorks (Echelon and LonWorks, 2005) and Profibus (Profibus, 2005). The main drawback of the event-triggered protocols is their lack of predictability. A large consortium of automotive manufacturers and suppliers has proposed a hybrid type of protocol, namely, the FlexRay communication protocol (FlexRay, 2005a). FlexRay allows the sharing of the bus among event-triggered and time-triggered messages, thus offering the advantages of both protocols. It is reported that FlexRay will very likely become the de-facto standard for in-vehicle communications (Pop, et al., 2006; Navet, et al., 2005). FlexRay defines a communication cycle (bus cycle) as the combination of a time-triggered (or static) window, an event-triggered (or dynamic) window, a symbol window and a network idle time (NIT) window. The time-triggered window is similar to TTP, and employs a time-division multiple-access (TDMA) mechanism. The event-triggered window of the FlexRay protocol is similar to Byteflight protocol and uses a flexible TDMA (FTDMA) bus access method. The symbol window is a communication period in which a symbol can be transmitted on the network. The NIT

* This work was partially supported by a grant from Iran Telecommunication Research Center (ITRC).

window is a communication-free period that specifies the end of each communication cycle.

The importance of safety in critical distributed applications signals to pay specific attention to the reliability of communication protocols. One way to evaluate the reliability of communication protocols is by fault injection. In (Salmani and Miremadi, 2005a), simulation-based fault injection has been used for the assessment of message missing in the CAN protocol. Effects of masquerade failures have been investigated using simulation-based fault injection in the CAN protocol (Salmani and Miremadi, 2005b). Evaluation of TTP/C communication controller by heavy-ion fault injection (hardware-based fault injection) has been performed in (Sivencrona, et al., 2003a). The purpose of the experiments in that paper was to validate the fail silence property of the TTP/C by injecting faults in a single node. The relationship between cluster size and occurred slightly-off-specification (SOS) failures has been assessed using heavy-ion fault injection (Sivencrona, et al., 2003b). In (Ademaj, et al., 2003), the TTP/C protocol with bus and star topology has been investigated using SWIFI fault injection. Here, the effects of SOS failures with respect to the start of frame transmission have been studied. In (Pallierer, et al., 2005; Armengaud, et al., 2005a; Armengaud, et al., 2005b), a generic tool was developed for monitoring and diagnosis of a FlexRay-based system as well as for a CAN-based system. This tool has been used by the FlexRay consortium to perform extended fault injection for evaluating of the FlexRay communication protocol. One important limitation of this tool is that faults can not be injected inside different parts of the FlexRay protocol.

Although researchers have performed different evaluations for protocols, none of them have performed babbling idiot failure evaluation in the FlexRay protocol. This paper evaluates conditions in which faults in the FlexRay protocol cause the babbling idiot failure. In the babbling idiot failure, a node sends messages without obeying the bus access rules imposed by the bus access methodology, thus corrupting the messages transmitted by the non-faulty nodes (Temple, 1998). This evaluation is done by 35680 bit-flip fault injection inside different parts of the FlexRay protocol. To do this, a FlexRay communication controller was modeled by Verilog HDL at the behavioral level. A FlexRay-based network composed of four nodes was established using this controller. Faults were injected only in one of the nodes. The evaluations are done in three phases: in the first phase the error propagation in the FlexRay network and its relation to the babbling idiot failure is analyzed. The dependency of the babbling idiot failure to fault injection locations (FlexRay protocol parts) is investigated. The effects of bus guardian in decreasing the babbling idiot failure are assessed in the second phase. Also in this phase, one weakness of the bus guardian for controlling message transmissions in dynamic window is identified. Finally, in third phase, a method for improving bus guardian is presented and the effects of this method on the babbling idiot failure are evaluated.

This paper is organized in six sections. Section 2, introduces FlexRay protocol, and section 3, presents the failure and error models found in this protocol. The experimental organization is given in section 4, and the results are presented in section 5. The last section concludes the work.

2. FLEXRAY PROTOCOL

A consortium of major automotive companies which includes BMW, Bosch, DaimlerChrysler, General Motors, Motorola, Philips, and Volkswagen, is currently developing the FlexRay protocol. The FlexRay network is very flexible with regard to topology and transmission support redundancy. It can be configured as a bus, a star or a multistar. It is not mandatory that each station possess neither replicated channels nor a bus guardian, even though this should be the case for critical functions such as steer-by-wire. In this section, protocol operation and protocol structure have been explained briefly.

2.1 Protocol Operation

At the media access control (MAC) level, FlexRay defines a communication cycle as the concatenation of a time-triggered (or static) window, an event triggered (or dynamic) window, a symbol window and a network idle time (NIT) window. The communication cycles are executed periodically. The time-triggered window uses a TDMA MAC mechanism; a station in FlexRay might possess several slots in the time-triggered window, but the size of all the slots is identical (Figure 1). In the event-triggered part of the communication cycle, the mechanism is Flexible TDMA (FTDMA): the time is divided into so-called minislots, each station possesses a given number of minislots, and it can start the transmission of a frame inside each of its own minislots. A minislot remains idle, if the station has nothing to transmit which actually induces a loss of bandwidth (Cena and Valenzano, 2004). The symbol window is used to transmit specific symbols on the network. The NIT window is a communication-free period that concludes each communication cycle.

The FlexRay frame consists of three parts: the header segment, the payload segment and trailer segment. The FlexRay header segment consists of 5 bytes. These bytes contain a reserved bit, payload preamble indicator, null frame indicator, sync frame indicator, startup frame indicator, frame ID, payload length, header CRC and cycle count.

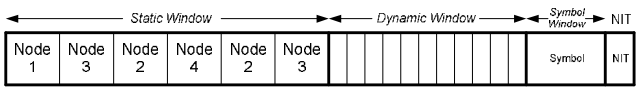

Fig. 1. Communication cycle in FlexRay protocol

The payload segment contains 0 to 254 bytes (0 to 127 two-byte words) of data. Because the payload length contains the number of two-byte words, the payload segment contains an even number of bytes. The FlexRay trailer segment contains a single field, a 24-bit CRC for the frame. The Frame CRC field contains a cyclic redundancy check code (CRC) computed over the header segment and the payload segment of the frame. The computation includes all fields in these segments.

2.2 Protocol Structure

The FlexRay protocol controller consists of six parts: controller host interface (CHI), protocol operation control (POC), coding and decoding (CODEC), media access control (MAC), frame and symbol processing (FSP) and clock synchronization process (CSP).

The CHI manages data and control flow between the host processor and the FlexRay protocol engine within each node. The CHI contains two major interface blocks: the protocol data interface and the message data interface. The protocol data interface manages all data exchange relevant to the protocol operation and the message data interface manages all data exchange relevant to the exchanges of messages. The protocol data interface manages the protocol configuration data, the protocol control data, and the protocol status data. The message data interface manages the message buffers, the message buffer configuration data, the message buffer control data, and the message buffer status data. In addition, the CHI provides a set of services that define self-contained functionality that is transparent to the operation of the protocol (FlexRay, 2005a).

The core parts of the protocol are moded by POC. Proper protocol behavior can only occur if the mode changes of the core parts are properly coordinated and synchronized. The purpose of the POC is to react to host commands and protocol conditions by triggering coherent changes to core parts in a synchronous manner, and to provide the host with the appropriate status regarding these changes (FlexRay, 2005a).

The CODEC contains two sections: coding section and decoding section. Coding section is responsible for encoding the communication elements into a bit stream and how the transmitting node represents this bit stream to the bus driver for communication onto the physical media. Decoding section is responsible for receiving communication elements, make bit streams and investigate correctness of bit streams.

The MAC controls access to the bus. In the FlexRay protocol, media access control is based on a recurring communication cycle. Within one communication cycle, FlexRay offers the choice of two media access schemes. These are a TDMA scheme and a FTDMA scheme. The communication cycle is the fundamental element of the media access scheme within FlexRay. It contains the static segment, the dynamic segment, the symbol window and the NIT (FlexRay, 2005a).

The FSP is the main processing layer between CODEC and CHI. This part checks the correct timing of received frames and symbols with respect to the TDMA scheme, applies further syntactical tests to received frames, and checks the semantic correctness of received frames (FlexRay, 2005a).

Finally, the CSP uses a distributed clock synchronization mechanism in which each node individually synchronizes itself to the cluster by observing the timing of transmitted sync frames from other nodes. A fault-tolerant algorithm is used for synchronizing the clock. So, CSP is responsible for generating Microticks, Macroticks and Cycles. Figure 2 shows relation between these parts.

3. FAILURE AND ERROR MODELS

Error models

The FlexRay protocol has different mechanisms for detecting errors in the controller. At the end of each time slot, FSP part checks the presence of any error in that slot and informs the host about it. This protocol defines three main errors that can occur in each slot: syntax error, content error and boundary violation error. The syntax error denotes the presence of a syntactic error in a time slot, the content error denotes the presence of an error in content of a received frame and boundary violation error denotes whether a boundary violation occurred at boundary of the corresponding slot.

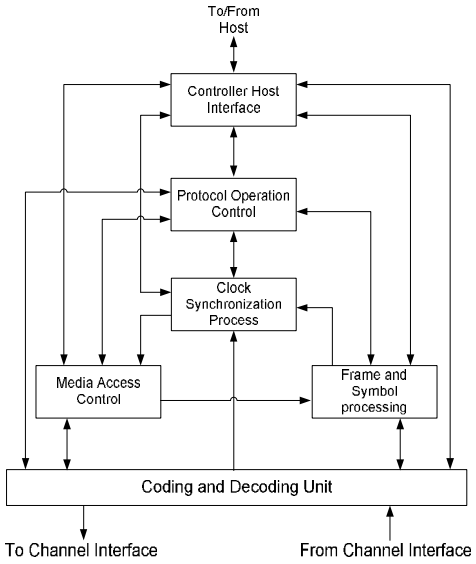

Fig. 2. FlexRay structure (FlexRay, 2005a)

Babbling idiot failures

A node that sends messages at arbitrary points in time, colliding with frames sent from other nodes (transmission conflict), exhibits the most serious failure in a distributed system based on a broadcast bus. Nodes that are affected by this kind of failure mode are called babbling idiots (Temple, 1998; Ademaj, et al., 2003). Babbling idiots send messages without obeying the bus access rules imposed by the bus access methodology thus corrupting the messages being transmitted by the non-faulty nodes (Temple, 1998). Such a failure makes impossible the communication of non-faulty nodes within the cluster.

4. EXPERIMENTAL ORGANIZATION

This section discusses the basic characteristics of the experiment.

4.1 Experimental setup

In order to perform an experiment on the FlexRay controller a network consisting of nodes that have this controller should be set up. So, a model of FlexRay controller has been implemented at behavioral level according to the FlexRay protocol specification (FlexRay, 2005a). This controller has been implemented by hardware description language, Verilog, and Modelsim 6.1 simulator.

The implemented controller has usual capabilities of FlexRay protocol such as sending and receiving the static and dynamic frames and symbols. This controller according to the specifications in (FlexRay, 2005a) has six parts to perform its functions: controller host interface (CHI), protocol operation control (POC), clock synchronization process (CSP), frame and symbol process (FSP), media access control (MAC), coding and decoding (CODEC). In addition, instead of a real application, a data generator is implemented to generate static frames with fixed length and dynamic frames with variable length at the start of the communication cycles.

Then, a cluster is formed consisting of four nodes with single bus topology. Any node is allowed to send and receive frames on communication channel. As depicted in figure 3, faults are injected in node 2 and their error propagation effects are observed in node 4. After each fault injection, the results in node 4 will be saved. As discussed each node on this network consists of three main parts: Host that

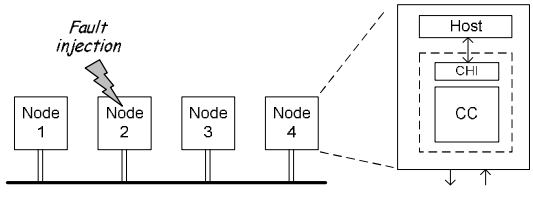

Fig. 3. Experimental setup

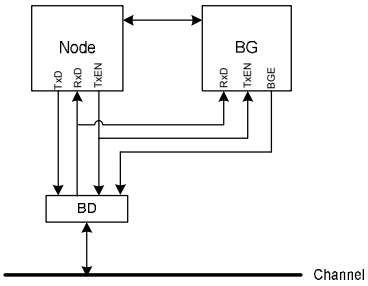

Fig. 4. Block diagram of a node with BG

generates the frames, an interface between host and controller, namely, the controller host interface (CHI) and at the lowest part there is the communication controller (CC). In this experiment, faults are injected in five parts of the communication controller of the node 2, including CHI, POC, CSP, MAC and CODEC. Because of the FSP part doesn't have any effect in error propagation; there is no fault injection in the FSP part. The effects of fault injection are observed in communication controller of the node 4 by FSP part.

In this paper, for experimental setup, nodes are configured in two forms: 1) node without bus guardian, 2) node with bus guardian. In the latter form, each node includes a bus guardian (BG) that controls transmission behaviour of CC. The functionality of the BG is composed of a subset of the functionality of a CC with an additional process which supervises the local CC (FlexRay, 2005b). The BG operates with an independent clock synchronization process and supervises the CC. Also, BG and CC have separate clock oscillators. The BG is able to receive frames on channel, but it does not transmit frames on channel. Figure 4 shows the block diagram of a node with BG that composed of a node, a BG and a bus driver (BD).

The BG generates the BGE signal, which enables the output (transmission) of the BD. Additionally the BG supervises the TxEN signal from CC. In case of a misaligned slot scheduling of the CC the BG generates an error interrupt to the host (FlexRay, 2005b).

Figure 5 shows the bus guardian schedule for one communication cycle. The BG enables transmission to the communication medium for all configured slots according to bus guardian configuration parameters (set of numbers of active static slots).The BG enables transmission for the entire duration of the dynamic segment. The BG disables transmission for the entire duration of the symbol window and the NIT (FlexRay, 2005b).

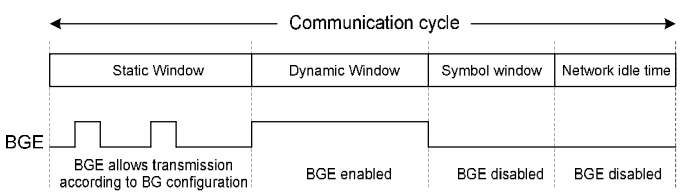

Fig. 5. Bus guardian schedule (FlexRay, 2005b)

4.2 Fault injection tool

The SINJECT fault injection tool (Zarandi, et al., 2003) is used to inject fault, at behavioural level, in nodes, collecting the results, and analyzing them.
A fault injection process usually consists of three steps:
1- When the given workload is applied, the behaviour of a fault-free network is computed and stored.
2- During the second step, to consider faults effects, the given workload are applied again to the network, the fault is injected, and the behaviour of the network is observed.
3- During the third step of the fault injection process, the faulty network behaviour is compared with the behaviour of the fault-free network, which is gathered at first step, and therefore the fault effects are specified and saved.

5. EXPERIMENTAL RESULTS

As discussed, for doing this experiment a network consisting of four nodes was set upped. Afterwards, totally 35680 bit-flip faults were injected in five different parts of communication controller of node 2. These five parts included: CHI, CSP, MAC, POC, and CODEC. Each experiment last for 3 communication cycles, in cycle 1 the faults were injected and the effects of them observed in cycle 1 through 3. In each communication cycle 6 slot IDs in static window and 6 slot IDs in dynamic window were allocated to different nodes.

In this section the results of these experiments are evaluated. The evaluations are done in three phases. In the first phase the error propagation in a FlexRay-based network and its relation to the babbling idiot failure is evaluated without any bus guardian. Then, in second phase, the babbling idiot failure is assessed in presence of a bus guardian. Finally, by analyzing the results of second phase, an improvement is added to bus guardian for reducing the babbling idiot failure in FlexRay-based networks. This improvement and its results are discussed in third phase.

5.1 Error propagation and babbling idiot failure evaluation without presence of bus guardian

The FlexRay protocol defines three main error models: content error, syntax error and boundary violation error. In this phase after injecting the faults inside the communication controller of node 2, the errors that occur in node 4 are observed. Table 1 shows the results of this experiment. Figure 6 shows this information by bar diagram.

The importance of investigating the error propagation in this experiment is that the babbling idiot failure can be assessed by error propagation results. One of the babbling idiot failure factors is that a node occupies the communication channel more than its time quota. In the FlexRay protocol, the boundary violation error occurs when a node occupies the bus more than a slot length. If this error continues until conflicts with other nodes transmission, babbling idiot failure occurs. So, boundary violation errors may increase the babbling idiot failures. As illustrated in figure 6, fault injection inside the CSP part cause most boundary violation errors. This part synchronizes the local clock of the node with other nodes in the cluster.

The occurrence of babbling idiot failure not only defects the operation of fault injected node (destructs the scheduling of communication cycle and corrupt the sent message), it also influences the operation of other nodes. Especially it may corrupt the messages that are sent by other nodes. One of the most important conditions that have to be kept in communication protocols is that nodes do not access the bus simultaneously. Otherwise, a transmission conflict occurs. In the FlexRay protocol the transmission conflict occurs when a node wants to send a message while it receives another one from other nodes. The main babbling idiot failure effect is transmission conflict. The transmission conflict occurrence rate is different in different nodes. Usually this error more often occurs in a node that should send message after faulty node message sending. In this experiment, the babbling idiot failures are detected by observing the transmission conflicts. Table 2 shows the babbling idiot failures.

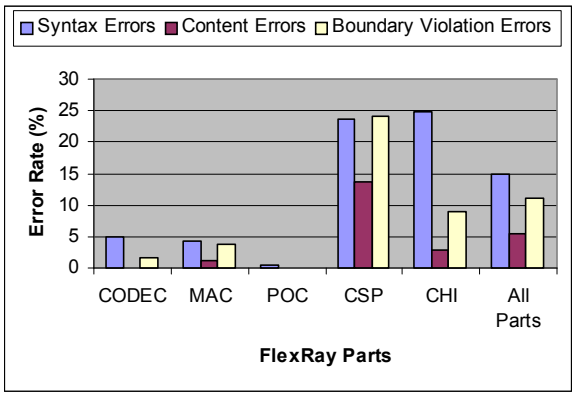

Fig. 6. Effect of fault injection in FlexRay parts without presence of BG

Table 2 Babbling idiot failures without presence of BG

FlexRay Parts	No. of Faults	Babbling idiot failures	
		#	%
CODEC	9300	157	1.69
MAC	4100	113	2.76
POC	2800	0	0.00
CSP	12480	1007	8.07
CHI	7000	355	5.07
All Parts	**35680**	**1632**	4.57

Table 1 Effect of fault injection in FlexRay parts without presence of BG

FlexRay Parts	No. of Faults	Syntax Errors		Content Errors		Boundary Violation Errors	
		#	%	#	%	#	%
CODEC	9300	457	4.91	2	0.02	164	1.76
MAC	4100	175	4.26	53	1.29	159	3.87
POC	2800	13	0.46	0	0	0	0
CSP	12480	2939	23.54	1724	13.81	2994	23.99
CHI	7000	1745	24.92	204	2.91	635	9.07
All Parts	**35680**	**5329**	**14.93**	**1983**	**5.55**	**3952**	**11.07**

5.2 Error propagation and babbling idiot failure evaluation in presence of bus guardian

In this phase a bus guardian (BG) is used beside each node. This device prevents the babbling idiots in faulty node. It has a BGE output that is enabled in slot IDs that allocated to the node in static window, exactly equal to one static slot length. Also this output is enabled during dynamic window. The node can send a message when the bus guardian output is enabled. If the bus guardian disables its output, the node will be disconnected from the bus. Thus, the bus guardian does not let the node to occupy the bus more than its quota. Figure 7 shows error propagation results after the use of bus guardian. As illustrated the boundary violation errors rate decreases noticeably.

According to bus guardian specification (FlexRay, 2005b), this device prevents illegal message transmissions in static window but in dynamic window it just controls the dynamic window length and has no controls on message transmissions in this window. Thus, the transmission conflicts may still occur in dynamic window yet. Table 3 shows the babbling idiot failures after use of the bus guardian. From this table it can be seen that these failures are decreased greatly but they are not eliminated completely. These failures occur in dynamic window because of weakness of the bus guardian for controlling message transmission in dynamic window. In next phase a mechanism is introduced to eliminate this problem.

5.3 Improvement of bus guardian

The results of section 5.2 showed that there were still some babbling idiot failures that had not eliminated by bus guardian. This was because of the bus guardian weakness in controlling the message transmissions in dynamic window. In this phase a method for improving bus guardian will be presented.

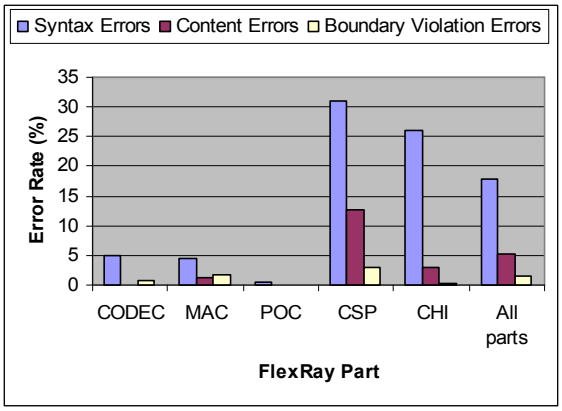

Fig. 7. Effect of fault injection in FlexRay parts in presence of BG

Table 3 Babbling idiot failures in presence of BG

FlexRay Parts	No. of Faults	Babbling idiot failures	
		#	%
CODEC	9300	75	0.81
MAC	4100	51	1.24
POC	2800	0	0.00
CSP	12480	125	1.00
CHI	7000	17	0.24
All parts	**35680**	268	0.75

In dynamic window the messages are sent eventually and a node may send a message in its IDs if it has a message for sending, otherwise it does not send a message. Furthermore the length of message is unfixed in dynamic window. So, it is difficult to predict the behaviour of the node by bus guardian and do some controls on it. By using minisloting mechanism features in the FlexRay protocol, an approach can be applied. In this approach the bus guardian know the slot IDs that the node is permitted to transmit a dynamic message. At the start of a minislot that its number is equal to the one of the node's slot IDs number, bus guardian enables BGE

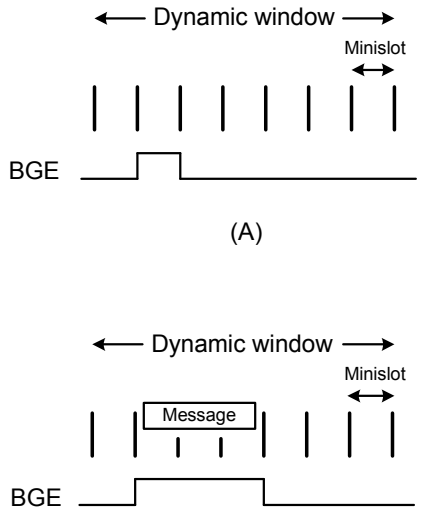

Fig. 8. Improvement of BG in dynamic window: (A) Node has no message for transmitting; (B) Node has a message for transmitting

and observes the bus. If the node sends a message, bus guardian waits until the end of the message. At the end of the message it disables the BGE on slot boundary (the slot boundary is calculated similar to MAC part of FlexRay protocol). Otherwise, if the node does not have any message for transmitting, at the end of that minislot bus guardian disables the BGE. Figure 8 shows the operation of bus guardian in these two situations.

Figure 9 shows the results of error propagation after improving the bus guardian. As this table shows, the boundary violation errors are completely eliminated. It means that the node does not occupy the bus more than its time quota in each time slot. Table 4 shows the babbling idiot failures that occur after applying this method to bus guardian. In this table, the improved bus guardian eliminates the babbling idiot failures completely.

6. CONCLUSIONS

This paper evaluated the error propagation and its effects in babbling idiot failure in a FlexRay-based network. The evaluation was based on about 35680 bit-flip fault injections inside different parts of the FlexRay communication controller. To do this, a FlexRay communication controller was modelled by Verilog HDL at the behavioural level. Then, this controller was exploited to setup a FlexRay-based network composed of four nodes. The evaluations were done in three phases: in the first phase the error propagation in the FlexRay network and its relation to the babbling idiot failure without presence of bus guardian was evaluated. The effects of bus guardian in decreasing the babbling idiot failures were assessed in the second phase. Also in this phase, one weakness of the bus guardian for controlling message transmissions in dynamic window was identified. Finally, in third phase, a method for improving bus guardian was presented and the

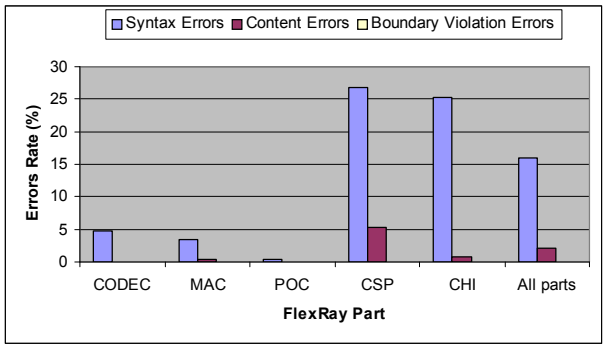

Fig. 9. Effect of fault injection in FlexRay parts after improvement in BG

Table 4 Babbling idiot failures after improvement in BG

FlexRay Parts	No. of Faults	Babbling idiot failures	
		#	%
CODEC	9300	0	0.00
MAC	4100	0	0.00
POC	2800	0	0.00
CSP	12480	0	0.00
CHI	7000	0	0.00
All parts	**35680**	0	0.00

effects of this method on the babbling idiot failure were evaluated. Results showed that in first, second and third phase about 4.57%, 0.75% and 0.00% of faults led to babbling idiot failures, respectively.

REFERENCES

Ademaj, A., H. Sivencrona, G. Bauer, and J. Torin (2003). Evaluation of Fault Handling of the Time-Triggered Architecture with Bus and Star Topology. *Proc. of the International Conference on Dependable Systems and Networks*, pp. 123-133.

Armengaud, E., F. Rothensteiner, A. Steininger, and M. Horauer (2005a). A Method for Bit Level Test and Diagnosis of Communication Services. *Proc. of the IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems.*

Armengaud, E., A. Steininger, and M. Horauer (2005b). An Efficient Test and Diagnosis Environment for Communication Controllers. *Proc. of the Austrochip Conference.*

Berwanger, J., M. Peller, and R. Griessbach (2000). Byteflight-A New High Performance Data Bus System for Safety-Related Applications. BMW 2000, available in http://www.byteflight.de.

Bosch GmbH, R. (1991). CAN Specification. v2.0.

Cena, G. and A. Valenzano (2004). Performance Analysis of Byteflight Networks. *Proc. of the IEEE Workshop Factory Communication Systems*, pp. 157-166.

Echelon, and LonWorks (2005). The LonTalk Protocol Specification. available in http://www.echelon.com.

FlexRay Consortium (2005a). FlexRay Communications System - Protocol Specification. v2.1 Revision A.

FlexRay Consortium (2005b). FlexRay Communications System - Preliminary Node-Local Bus Guardian Specification. v2.0.9.

Hoyme, K. and K. Driscoll (1992). SAFEbus. The *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, no. 3, pp. 34-39.

Kopetz, H. (1998). A Comparison of CAN and TTP. Vienna University of Technology, Real-Time System Group, Research Report 23.

Kopetz, H. and G. Bauer (2003). The Time-Triggered Architecture. *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112-126.

Miner, P.S. (2000). Analysis of the SPIDER Fault-Tolerance Protocols. *Proc. of the 5th NASA Langley Formal Methods Workshop.*

Navet, N., Y. Song, F. Simonot-Lion, and C. Wilwert (2005). Trends in Automotive Communication Systems. *Proceedings of the IEEE*, vol. 93, no. 6.

Pallierer, R., M.Horauer, M. Zauner, A. Steininger, E. Armengaud, and F. Rothensteiner (2005). A Generic Tool for Systematic Tests in Embedded Automotive Communication Systems. *Proc. of the Embedded World Conference.*

Pop, T., P. Pop, P. Eles, Z. Peng, and A. Andrei (2006). Timing Analysis of the FlexRay Communication Protocol. *Proc. of the 18th Euromicro Conference on Real-Time System*, pp. 203-216.

Profibus International (2005). PROFIBUS DP Specification. available in http://www.profibus.com.

Salmani, H. and S. G.Miremadi (2005a). Assessment of Message Missing Failures in CAN-based Systems. *Proc. of the Parallel and Distributed Computing and Networks,* pp. 387-392.

Salmani, H. and S. G. Miremadi (2005b). Contribution of Controller Area Networks Controllers to Masquerade Failures. *Proc. of the 11th Pacific Rim International Symposium on Dependable Computing***,** pp. 310- 316.

Sivencrona, H., P. Johannessen, M. Persson, and J. Torin (2003a). Heavy-ion Fault Injections in the Time-triggered Communication Protocol. *Proc. of the Latin American Symposium on Dependable Computing*, pp. 69-80.

Sivencrona, H., M. Persson, and J. Torin (2003b). Using Heavy-Ion Fault Injection to Evaluate Fault Tolerance with Respect to Cluster Size in a Time-Triggered Communication Systems. *Proc. of the IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS-06)*, pp. 171-176.

Temple, C. (1998). Avoiding the Babbling-Idiot Failure in a Time-Triggered Communication System. In Processing of the 28th Annual International Symposium on fault-Tolerant Computing.

Zarandi, H. R., S. G. Miremadi, and A. Ejlali (2003). Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models. *Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 485-492.