Investigation and Reduction of Fault Sensitivity in the FlexRay Communication Controller Registers

Yasser Sedaghat and Seyed Ghassem Miremadi

Dependable Systems Laboratory, Sharif University of Technology, Tehran, Iran
y_sedaghat@ce.sharif.edu, miremadi@sharif.edu

Abstract. It is now widely believed that FlexRay communication protocol will become the de-facto standard for distributed safety-critical automotive systems. In this paper, the fault sensitivity of the FlexRay communication controller registers are investigated using transient single bit-flip fault injection. To do this, a FlexRay bus network, composed of four nodes, was modeled. A total of 135,600 transient single bit-flip faults were injected to all 408 accessible single-bit and multiple-bit registers of the communication controller in one node. The results showed that among all 408 accessible registers, 30 registers were immediately affected by the injected faults. The results also showed that 26.2% of injected faults caused at least one error. Based on the fault injection results, the TMR and the Hamming code techniques were applied to the most sensitive parts of the FlexRay protocol. These techniques reduced the fault affection to the registers from 26.2% to 10.3% with only 13% hardware overhead.

Keywords: Safety-critical applications, Distributed embedded systems, FlexRay protocol, Fault injection.

1 Introduction

Today, many safety-critical applications are implemented as distributed embedded systems [13], e.g. X-by-wire applications. These systems are composed of several different types of hardware units (called nodes), e.g., processing units, sensors, and actuators, interconnected by a communication network.

Communication in a distributed architecture can be triggered either dynamically, in response to an event (event-driven), or statically, at predetermined moments in time (time-driven). Examples of event-triggered protocols are Byteflight [1], CAN [2], LonWorks [3], and Profibus [4]. The main drawback of event-triggered protocols is their lack of predictability [5]. Examples of time-triggered protocols are SAFEbus [6], SPIDER [7], and TTP/C [8]. The main drawback of time-triggered protocols is their lack of flexibility [5]. To resolve the drawbacks of both event-triggered and time-triggered protocols, other protocols such as TTCAN [9], FTT-CAN [10], and Flex Ray [11] are introduced that can support both time-triggered and event-triggered transmissions.

Among the latter protocols, the FlexRay protocol is advancing as the predominant protocol and will become the de-facto industry standard for X-by-wire applications [12], [13], [5], [14], [15]; e.g., the next edition of the BMW X5 will use the FlexRay

M.D. Harrison and M.-A. Sujan (Eds.): SAFECOMP 2008, LNCS 5219, pp. 153–166, 2008.

protocol in its electronically controlled dampers [12].The FlexRay protocol was started by an industry consortium with four founding members (BMW, Daimler-Chrysler, Philips, and Freescale) [15]. Three top design objectives were considered in the standardization of the FlexRay protocol: high speed transmission, deterministic communication, and fault-tolerant communication [15].

In safety-critical distributed embedded systems, a fault-tolerant communication between different nodes has a significant impact on the overall system reliability. It has been reported [16], [13] that the overall reliability of a safety-critical distributed embedded system not only depends on the reliability of the nodes, but also on the reliability of the communication network.

This paper investigates the fault sensitivity of all parts of the FlexRay communication controller using fault injection. The most and the least sensitive registers in the FlexRay are characterized. Then, appropriate fault-tolerant techniques are applied to the most sensitive registers, to protect the communication controller against transient faults.

The remainder of the paper is organized as follows: Section 2 introduces the FlexRay protocol briefly. Error models and error handling mechanisms in the FlexRay protocol are presented in Section 3. In Section 4, the experimental environment is presented. Section 5 includes the experimental results and finally, the conclusions are given in Section 6.

2 The FlexRay Protocol

The FlexRay protocol provides key features of synchronization that include scalable data transmission in both synchronous and asynchronous modes. It can support the data rate up to 10Mbit/sec. The protocol itself offers deterministic data transmission, guaranteed message latency and message jitter. The FlexRay supports dual and redundant transmission channels and transmission mechanism is arbitration free. In addition, it has optional support of optical or electrical physical layers. The physical layer will provide support for bus, star, and multiple star topologies [11].

From the dependability point of view, the FlexRay documents [11] specify solely bus guardian mechanism and clock synchronization algorithms. Other features, such as a membership service or mode management facilities, should be implemented in software or hardware layers on top of the FlexRay. This will allow to conceive and to implement exactly the services that are needed with the drawback that correct and efficient implementations might be more difficult to achieve in a layer above the communication controller [16].

One of the main purposes of this paper is to convince developers of the FlexRay communication controller, by the experimental results, how necessary it is to reduce the fault sensitivity of critical registers. This reduction causes to improve the reliability of the FlexRay protocol, noticeably; and it is possible to reduce the need for expensive fault-tolerant techniques, such as bus guardian mechanism or clock synchronization algorithms.

2.1 Protocol Operation

Communications in the FlexRay protocol are based on predetermined interval times which are named communication cycles (bus cycles). These communication cycles are

executed periodically. In this protocol a communication cycle is a 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 size of each communication window is set statically at design time. The time-triggered window uses a Time Division Multiple Access (TDMA) [17] mechanism; a node in FlexRay might possess several slots in the time-triggered window, but the size of all the slots is identical. In the event-triggered part of the communication cycle, the mechanism is Flexible TDMA (FTDMA) [18]: time is divided into so-called minislots, each station possesses a given number of minislots (not necessarily consecutive), 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 [16]. The symbol window is a communication period in which a symbol can be transmitted on the network. The NIT window is a communication-free period that concludes each communication cycle. Fig. 1 shows an example of communication cycle in the FlexRay protocol.

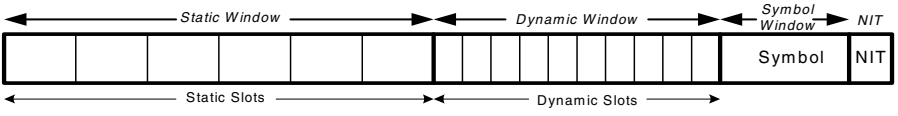


Fig. 1. Communication Cycle in the FlexRay Protocol

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

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.

In the FlexRay protocol, frames are sent in static slots or dynamic slots of each communication cycle. Fig. 2 shows the frame format in the FlexRay protocol.

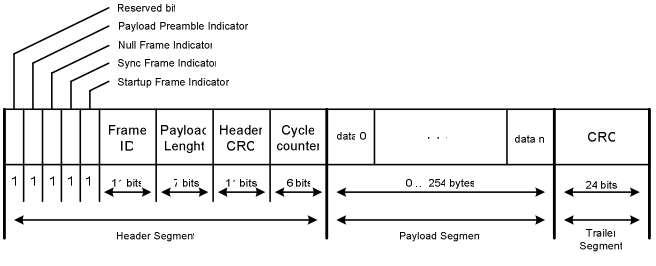


Fig. 2. Frame format in the FlexRay Protocol

2.2 Protocol Structure

The FlexRay communication controller consists of six modules [11]: 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). Fig. 3 shows relation between these modules.

The CHI module, manages data and control flow between the host processor and the FlexRay protocol engine within each node. The CHI module manages all data exchange relevant to the protocol operation and manages all data exchanges relevant to the exchange of messages. Moreover, this module manages protocol configuration data, protocol control data, and protocol status data.

Operational modes of FlexRay modules are adjusted by POC module. Proper protocol behavior can only occur if the mode changes of the core modules 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 modules in a synchronous manner, and to provide the host with the appropriate status regarding these changes.

The CODEC module is responsible for encoding the communication elements into a bit stream and is responsible for receiving communication elements, making bit streams and investigating correctness of bit streams.

The MAC module controls access to the bus. In the FlexRay protocol, media access control is based on a recurring communication cycle. Within one communication cycle, the FlexRay offers the choice of two media access schemes, i.e., TDMA scheme and FTDMA scheme. The communication cycle is the fundamental element of the media access scheme within FlexRay.

The FSP module checks the correct timing of received frames and received symbols with respect to the TDMA scheme, applies further syntactical tests to received frames, and checks the semantic correctness of received frames.

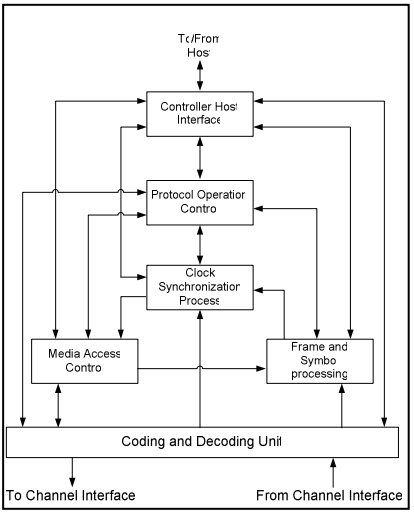


Fig. 3. The FlexRay Structure [11]

Finally, the CSP module is responsible for generation of timing units in the FlexRay communication controller, e.g., communication cycles. Moreover this module uses a distributed clock synchronization mechanism in which each node individually synchronizes itself to its cluster by observing the timing of transmitted sync frames from other nodes.

3 Error Models and Error Handling Mechanisms in the FlexRay Protocol

Safety-critical applications have to function correctly even in presence of faults. Faults can be permanent (e.g., damaged microcontrollers or communication links), transient (e.g., caused by single event upsets or electromagnetic interferences), or

intermittent (appear and disappear repeatedly). The transient faults are the most common, and their number is continuously increasing due to the continuously raising level of integration in semiconductors [19]. These transient single bit-flip errors are more common consequences of transient faults [22].

3.1 Error Models in the FlexRay Protocol

According to the FlexRay protocol, the following three categories of errors are possible [11]:

1) Syntax error

Syntax error denotes the presence of a syntactic error in a time slot, and occurs in following conditions:

- The node starts transmitting while the channel is not in the idle state.
- A decoding error occurs.
- A frame is decoded in the symbol window or in the network idle time.
- A symbol is decoded in the static segment, the dynamic segment, or the network idle time.
- A frame is received within the slot after the reception of a semantically correct frame.
- Two or more symbols are received within the symbol window.

2) Content error

Content error denotes the presence of an error in a received frame, and occurs in following condition:

- In the static segment, the header length the header of the received frame does not match the stored header length in a special register (this register contains globally configured value of the payload length of a static frame).
- In the static segment, the startup frame indicator, contained in the header of the received frame, is set to one while the sync frame indicator is set to zero.
- In the static or in the dynamic segment, the frame ID, contained in the header of the received frame, does not match the current value of the slot counter or the frame ID equals to zero in the dynamic segment.
- In the static or dynamic segment, the cycle count, contained in the header of the received frame, does not match the current value of the cycle counter.
- In the dynamic segment the sync frame indicator, contained in the header of the received frame, is set to one.
- In the dynamic segment the startup frame indicator, contained in the header of the received frame, is set to one.
- In the dynamic segment the null frame indicator, contained in the header of the received frame, is set to zero.

3) Boundary violation error

Boundary violation error denotes whether a boundary violation has occurred at the boundary of the corresponding slot. A boundary violation occurs if the node does not consider the channel to be idle at the boundary of a slot.

3.2 Error Handling Mechanisms in the FlexRay Protocol

In order to respond to errors, two basic mechanisms are provided in the POC module [11]. For significant errors, the POC:halt state is immediately entered. The POC also

contains a three-state degradation model for errors that can be endured for a limited period of time. In this case entry to the POC:halt state is deferred, at least temporarily, to support possible recovery from a potentially transient condition.

Errors causing immediate entry to the POC:halt state
There are three general conditions that trigger entry to the POC:halt state:

- Product-specific error conditions such as BIST errors and sanity checks.
- Error conditions detected by the host that result in a FREEZE command being sent to the POC via the CHI.
- Fatal error conditions detected by the POC or one of the core mechanisms.

Product-specific errors are accommodated by the POC, but not described in FlexRay specification. Similarly, host detected error strategies are supported by the POC's ability to respond to a host FREEZE command, but the host-based mechanisms that trigger the command are beyond the scope of this specification, hence they were not considered in this paper.

Errors handled by the degradation model
Integral to the POC is a three-state error handling mechanism referred to as the degradation model. It is designed to react to certain conditions detected by the clock synchronization mechanism that are indicative of a problem, but that may not require immediate action due to the inherent fault tolerance of the clock synchronization mechanism. This makes it possible to avoid immediate transitions to the POC:halt state while assessing the nature and extent of the errors. The degradation model is embodied in three POC states - POC:normal active, POC:normal passive, and POC:halt.

In the POC:normal active state, the node is assumed to be either error free, or at least within error bounds that allow continued “normal operation”. Specifically, it is assumed that the node remains adequately time synchronized to the cluster to allow continued frame transmission without disrupting the transmissions of other nodes.

In the POC:normal passive state, it is assumed that synchronization with the remainder of the cluster has degraded to the extent that continued frame transmissions cannot be allowed because collisions with transmissions from other nodes are possible. Frame reception continues in the POC:normal passive state in support of host functionality and in an effort to regain sufficient synchronization to allow a transition back to the POC:normal active state.

If errors persist in the POC:normal passive state or if errors are severe enough, the POC can transit to the POC:halt state. In this state it is assumed that recovery back to the POC:normal active state cannot be achieved, so the POC halts the core mechanisms in preparation for reinitializing the node. The conditions for transitioning between the three states comprising the degradation model are configurable. Furthermore, transitions between the states are communicated to the host allowing the host to react appropriately and to possibly take alternative actions using one of the explicit host commands.

3.3 Error Indicator Registers of the FlexRay Communication Controller

In this protocol, there are some registers that are set in the mentioned error conditions. In this paper, these registers are named “error indicator registers”. Table 1 shows these registers and their locations in the FlexRay communication controller.

Activating each of these registers, may result in one or more main error types. Faults, depending on when and where they occur, may change the expected value of some of these registers and cause one or more main error types. In this paper, the type of occurred error is not considered. However, if any of registers in the Table 1, is unexpectedly changed, this change is considered as an error.

Table 1. Error indicator registers (registers of the FlexRay showing the error occurrences) in the FlexRay protocol

Registers	Module	Registers	Modules
decoding_error_on_A	CODEC	vPOC_Freeze	POC
TSS_ok		vPOC_CHIHaltRequest	
TSS_too_long		vPOC_ErrorMode	
FSS_ok		zSyncCalcResult	CSP
payload_ok		Content_error_on_A	FSP
trailer_ok		Fatal_protocol_error	
BSS_ok		T_StatusSlot_ValidFrame	
FES_ok		T_StatusSlot_SyntaxError	
zBssError		T_StatusSlot_ContentError	
Header_Crc_error		T_StatusSlot_TxConflict	
Frame_Crc_error		T_StatusSlot_BViolation	

4 Experimental Environment

The FlexRay communication controller was implemented by hardware description language, Verilog HDL, and specifications of this controller, e.g. timing and configuration, were tested according to the FlexRay protocol conformance test specification [20]. This controller, according to its specifications [11], has six modules 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), and coding and decoding (CODEC). A cluster was formed consisting of 4 nodes with single bus topology (Fig. 4). In this topology, a node is composed of a host and a communication controller. The host typically is a hardware unit that generates data to exchange with other nodes through a communication channel.

In the experiments, instead of a real host, a data generator was implemented to generate static frames with fixed length and dynamic frames with variable length at the start of the communication cycles. In this cluster, any node was allowed to send and receive frames on the communication channel.

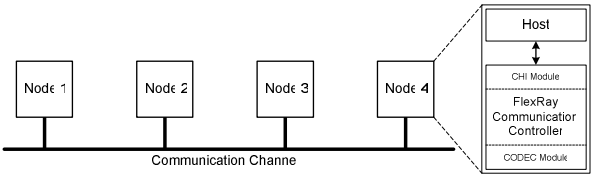


Fig. 4. Experimental setup

To investigate the fault tolerance of the FlexRay communication controller, transient single bit-flip faults were injected in all accessible registers of communication controller modules of the node 2 and their effects on the error indicator registers were observed in node 2 and node 4 (for observing more fault effects); and results were stored.

4.1 Fault Sensitivity Calculation Process

To inject the transient single bit-flip faults at the behavioral level in node 2, the SINJECT fault injection tool [21] was used.

A fault sensitivity calculation process of a bit, by using SINJECT tool, consists of four steps:

1- When the given workload is applied, behaviors of the error indicator registers in a fault-free network are simulated and stored.
2- During the second step, to consider fault effects, the given workload is applied again to the network, a single transient bit-flip fault is injected to a bit of a communication controller register of node 2, at a random time, and the behavior of the error indicator registers of node 2 and node 4 are observed.
3- During the third step of the fault sensitivity calculation process, the faulty network behavior is compared with the behavior of the fault-free network, which is gathered at first step, and if there is a mismatch, this injected fault is considered as an activated fault and otherwise, this injected fault is considered as an overwritten fault.
4- To achieve accurate fault sensitivity of a bit, several faults should be injected to this bit (repeating the first three steps). After injecting enough bit-flip faults and determining the number of activated faults (be Equation 1), the fault sensitivity of this bit is calculated by Equation 2:

$$\#\text{injected faults} = \#\text{activated faults} + \#\text{overwritten faults} \tag{1}$$

$$\text{fault sensitivity of a bit} = \frac{\text{Number of activated faults}}{\text{Number of all injceted faults to that bit}} \times 100\% \tag{2}$$

The process was repeated for all bits in all accessible registers in FlexRay communication controller and the fault sensitivity of these registers was determined.

4.2 Fault Tolerance Improvement Strategies

After determining the fault sensitivity of a register, if its sensitivity was more than an acceptable value, a proper fault-tolerant technique would be used to reduce its vulnerability. The Hamming code technique with single bit correction ability and Triple Modular Redundancy (TMR) technique were used for this purpose.

The Hamming technique was implemented on several sets of vulnerable registers. Those sets were organized such that most related registers were encapsulated in a set; and the size of each set varied between 10 bits and 32 bits. This implementation did not incur any delay or limitation to access to protected registers. After changing value of a protected register in a register set, due to protocol operations, Hamming bits of that register set is calculated while other parts of communication controller were allowed to access to that register set.

TMR or Hamming techniques should be used consciously; for example, if there is a highly fault sensitive register which immediately triggers other parts of communication controller by changing its value, the Hamming code technique should not be used to reduce the sensitivity of this register. The main reason is that if a bit-flip fault occurs in this register, the other parts of communication controller react to that fault immediately and some errors may occur in other parts of communication controller; in such situation, if Hamming technique is used, the occurred fault in the register is detected and corrected while other parts of communication controller react to this changing value again. Consequently, a bit-flip fault causes two incorrect reactions in other communication parts. Also, if Hamming technique is implemented such that the accessibility to that register is not allowed until the Hamming bits of this register are calculated, some delay is inserted into the operation of communication controller and this delay may corrupt the timing of FlexRay protocol operations. In this situation, the TMR technique is the better option, but if the imposed delay due to Hamming technique for this type of register does not damage the FlexRay protocol timings, by checking and testing according to the FlexRay protocol conformance test specification [20], it is beneficial to use Hamming technique instead of TMR technique.

On the other hand, if there is a highly fault sensitive register which does not trigger immediately other parts of communication controller by changing its value, the TMR technique should not be used because of its ultra-high hardware overhead (200%). In this condition, the Hamming technique is the better option.

In this paper, with respect to properties of the FlexRay communication controller registers, the TMR technique and the Hamming technique (without incurring any delay) are suggested to improve fault tolerance of this controller.

5 Experimental Results

In this paper, to assess the fault sensitivity of the FlexRay communication controller, the nodes were connected through a passive bus network. The main reason of selecting bus topology is to prevent some error propagations in star coupler of star topology. This prevention results in hiding the fault sensitivity of some communication controller registers.

To simulate the experiments, the ModelSim 5.5 simulation environment was used. The simulation includes four communication cycles; in the first two cycles, single transient bit-flip fault was injected randomly, then simulation was resumed two cycles to guarantee that the injected fault caused an error or overwritten.

5.1 The FlexRay Communication Controller Modules

To reach an accurate fault sensitivity of each register, 50 transient bit-flip faults were injected to each bit of all accessible FlexRay controller registers (according to the fault sensitivity calculation process) and gathered results were investigated. If there existed a register with more than 20% of fault sensitivity, a proper fault-tolerant technique based on properties of this register were used to reduce its vulnerability.

As discussed in the previous section, the TMR technique was only used for registers which were immediately triggered other communication controller parts with

their changing values. In this controller, all of these kinds of registers were single-bit register; consequently, for each single-bit register, two redundant flip-flops were added to implement the TMR technique. Furthermore, to implement the Hamming technique, all vulnerable registers which were not improved by TMR technique, were grouped in some sets. These register sets were organized as discussed in the previous section.

The results show that the TMR technique masks all injected faults but the Hamming technique is not able to tolerate all injected faults; because it is probable that faulty registers are used immediately before they are corrected. The experiment results are presented in Table 2, whereas the modeled FlexRay communication controller is not still synthesizable, the estimated hardware overhead is based on the number of accessible flip-flops. Table 3 contains hardware overhead of implemented techniques.

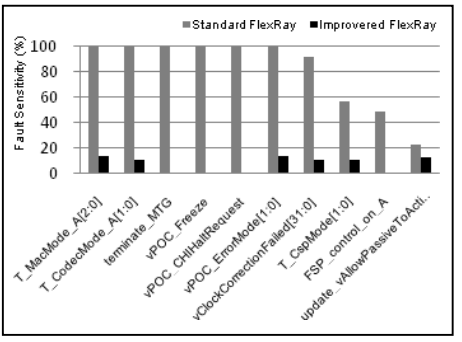
Table 2. Fault injection results

FlexRay Module	# Injected Faults	Standard FlexRay		Improved FlexRay		Improvement (%)
		Activated Faults		Activated Faults		
		#	%	#	%	
POC	5200	2343	45.1	512	9.8	357
CODEC	32300	5396	16.7	3586	11.1	50
MAC	11050	1805	16.3	1181	10.7	52
CSP	47850	16574	34.6	4774	10	246
FSP	6850	1230	18	688	10	80
CHI	32350	8154	25.2	3300	10.2	147
ALL	135600	35502	26.2	14041	10.3	154

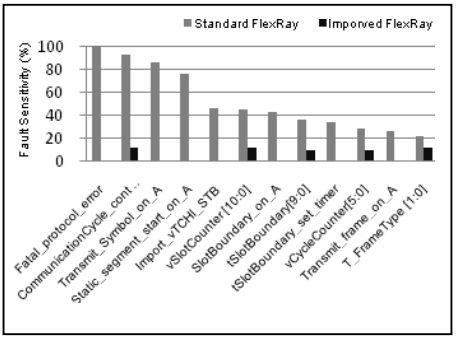
Table 3. Hardware overheads

FlexRay Modules	Standard FlexRay		Improved FlexRay	HW Overhead (%)
	# Registers	# Flip-Flops	# Flip-Flops	
POC	28	104	(104 + 32) = 136	30.8
CODEC	104	646	(646 + 46) = 692	7.1
MAC	64	221	(221 + 28) = 249	12.7
CSP	94	957	(957 + 162) = 1119	16.9`
FSP	41	137	(137 + 20) = 157	14.6
CHI	77	647	(647 + 66) = 713	10.2
ALL	408	2712	(2712 + 354) = 3066	13.0

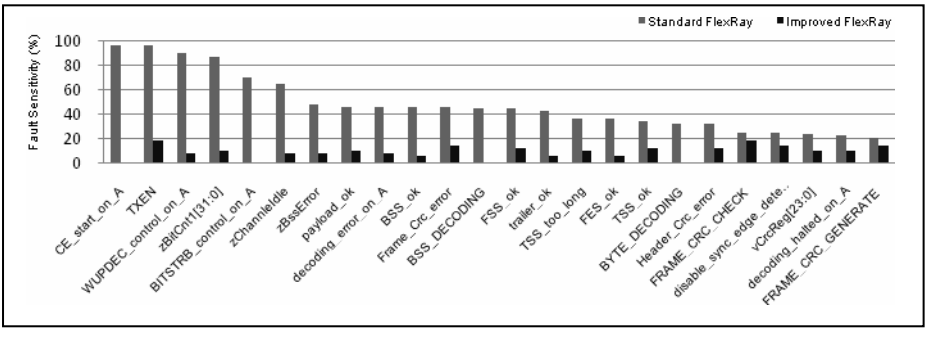
In the modeled FlexRay communication controller, all registers, signals and other components are named based on FlexRay specification document (version 2.1, revision A) [11]; for more details about their responsibilities, readers are referred to [11]. Based on experimental results, Fig. 5 shows the fault sensitivity of all FlexRay communication controller modules in the standard implementation (according to the FlexRay specifications [11]) and in the improved implementation. In this figure, the fault sensitivities of the FlexRay module registers which are more than 20% sensitive to injected faults are presented. For more clarity, fault sensitivities are sorted in a descending order.



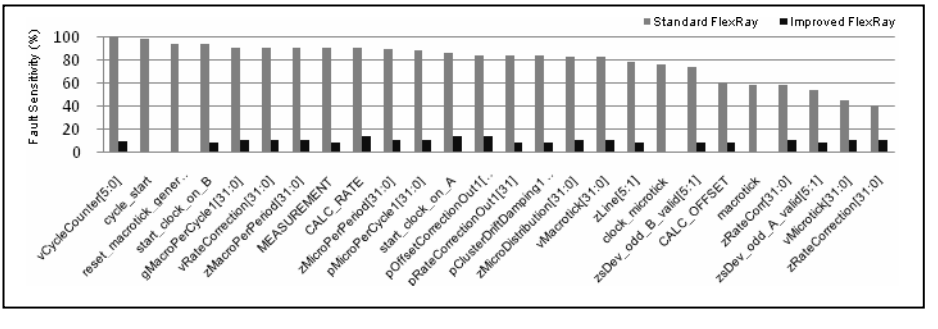
(a) Fault sensitivities of FlexRay Registers (POC Module)



(b) Fault sensitivities of FlexRay Registers (MAC Module)

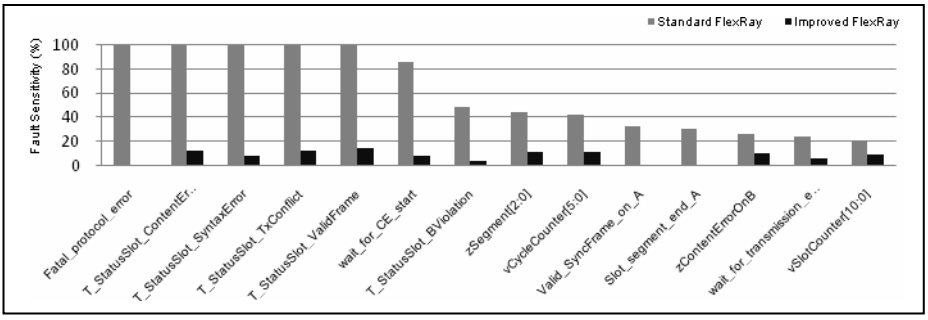


(c) Fault sensitivities of FlexRay Registers (CODEC Module)

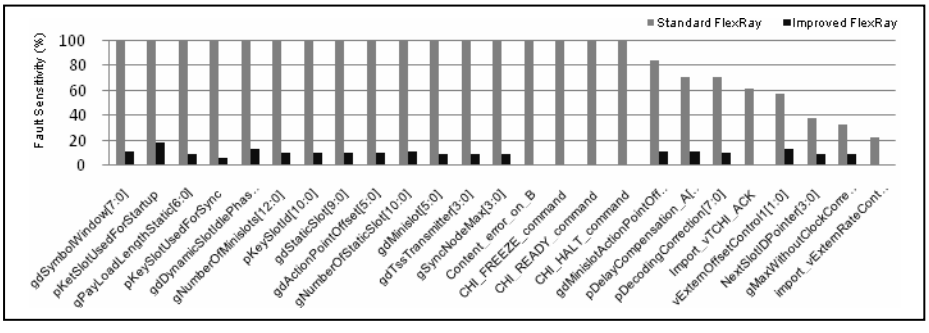


(d) Fault sensitivities of FlexRay Registers (CSP Module)

Fig. 5. Fault sensitivities of FlexRay modules in standard and improved implementations



(e) Fault sensitivities of FlexRay Registers (FSP Module)



(f) Fault sensitivities of FlexRay Registers (CHI Module)

Fig. 5. (*continued*)

5.2 Overall Results

In general, fault sensitivity analysis of the FlexRay modules shows that there were 30 registers with 100% sensitivity. This fact may question use of this protocol for safety-critical applications. In addition, in Fig. 5 there is a severe variance in the fault sensitivities of the FlexRay controller registers. Our improvements make them smooth.

The FlexRay communication controller contains 408 single-bit and multiple-bit registers in total. A number of 135,600 transient single-bit flip faults were injected to them. 35,502 faults caused at least one error; consequently, the fault sensitivity of the whole controller was about 26.2%.

After improving the fault sensitivity of the FlexRay communication controller, its sensitivity was reduced from 26.2% to 10.3% (about 154% improvement), while adding 354 extra flip-flops costs the controller about 13% flip-flop overhead.

Fig. 6 shows the fault sensitivity of each module in the standard implementation and the improved implementation of FlexRay communication controller. This figure also shows that the POC module is the most sensitive part of FlexRay communication controller and CODEC module is the least sensitive part. Furthermore, our results show that we were able to reduce the sensitivity of FlexRay modules to almost equal

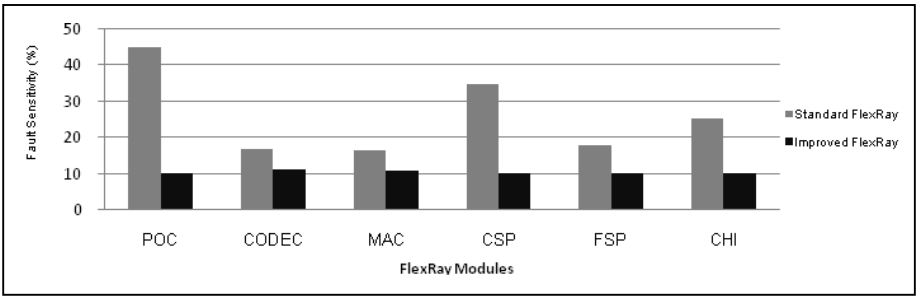


Fig. 6. Fault Sensitivity of FlexRay Communication Controller

values (difference about 1%) as compared with previous values in the standard implementation.

6 Conclusions

Safety-critical automotive control systems are nowadays complex distributed embedded systems and the communication protocol is an essential part of them. The FlexRay communication protocol is now expected to become the future standard for in-vehicle communication.

In this paper, the fault sensitivities and vulnerabilities of FlexRay communication controller registers, based on 135,600 single-bit flip fault injections to all accessible registers, are investigated.

The results show that the fault sensitivities of POC, CODEC, MAC, CSP, FSP, and CHI modules are 45.1%, 16.7%, 16.3%, 34.6%, 18%, and 25.2% respectively. Moreover, according to the fault injection results, among all 408 accessible registers, 30 registers were immediately affected by the injected faults, 84 registers were affected between 20% and 99%, while the remaining (294) registers were affected by less than 20%.

After determining the sensitive registers, proper fault masking and fault-tolerant techniques, based on their properties, are applied to reduce the vulnerability of these registers. This caused, the fault sensitivity of POC, CODEC, MAC, CSP, FSP, and CHI modules to reduce to 9.85%, 11.1%, 10.7%, 10%, 10%, and 10.2% respectively.

In general, the fault sensitivity of FlexRay communication controller was improved more than 2 times and in this improved implementation, none of the registers has more than 20% fault sensitivity.

References

1. Byteflight Specification, `http://www.byteflight.com/`
2. CAN Specification 2.0, `http://www.can-cia.org/`
3. LonWorks networks, `http://www.echelon.com`
4. PROFIBUS DP Specification, `http://www.profibus.com`
5. Pop, T., Pop, P., Eles, P., Peng, Z.: Bus Access Optimization for FlexRay-based Distributed Embedded Systems. In: Design, Automation & Test in Europe Conference & Exhibition 2007 (DATE 2007), pp. 1–6. EDA Consortium, Nice (2007)

6. Hoyme, K., Driscoll, K.: SAFEbus. In: IEEE Aerospace and Electronic Systems Magazine (ISSN 0885-8985), vol. 8(3), pp. 34–39. IEEE Press, Los Alamitos (1992)
7. Miner, P.S., Malekpour, M., Torres-Pomales, W.: Conceptual design of a Reliable Optical BUS (ROBUS). In: 21st AIAA/IEEE Digital Avionics Systems Conference, pp.13D3-1–13D3-11. IEEE Press, Irvine (2002)
8. Kopetz, H., Bauer, G.: The Time-Triggered Architecture. J. IEEE. 91(1), 112–126 (2003)
9. Road Vehicles—Controller Area Network (CAN)—Part 4: Time-Triggered Communication, ISO 11 898-4 (2000)
10. Ferreira, J., Pedreiras, P., Almeida, L., Fonseca, J.A.: The FTT-CAN protocol for flexibility in safety-critical systems. J. IEEE Micro. (Special Issue on Critical Embedded Automotive Networks) 22(4), 46–55 (2002)
11. FlexRay Communications System - Protocol Specification V2.1 Revision A, http://www.flexray.com
12. Sethna, F., Stipidis, E., Ali, F.H.: What Lessons Can Controller Area Networks Learn From FlexRay. In: Vehicle Power and Propulsion Conference (VPPC 2006), pp. 1–4. IEEE Press, Windsor (2006)
13. Pop, T., Pop, P., Eles, P., Peng, Z., Andrei, A.: Timing Analysis of the FlexRay Communication Protocol. In: 18th Euromicro Conference Real-Time Systems (ECRTS 2006), pp. 203–216. Kluwer Academic Publishers, Dresden (2006)
14. Hagiescu, A., Bordoloi, U.D., Chakraborty, S.: Performance Analysis of FlexRay-based ECU Networks. In: 44th ACM/IEEE Design Automation Conference (DAC 2007), pp. 284–289. ACM, San Diego (2007)
15. Makowitz, R., Temple, C.: FlexRay- A Communication Network for Automotive Control Systems. In: IEEE International Workshop on Factory Communication Systems (WFCS 2006), pp. 207–212. IEEE Press, Torino (2006)
16. Navet, N., Song, Y., Simonot-Lion, F., Wilwert, C.: Trends in Automotive Communication Systems. J. IEEE 93(6), 1204–1223 (2005)
17. Tindell, K., Clark, J.: Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. J. Microprocessing & Microprogramming 40, 117–134 (1994)
18. Cena, G., Valenzano, A.: Performance analysis of byteflight networks. In: IEEE Workshop on Factory Communication Systems (WFCS 2004), pp. 157–166. IEEE Press, Vienna (2004)
19. Izosimov, V., Pop, P., Eles, P., Peng, Z.: Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. In: Design, Automation and Test in Europe Conference and Exhibition 2005 (DATE 2005), vol. 2, pp. 864–869. IEEE Computer Society, Munich (2005)
20. FlexRay Communications System - Protocol Conformance Test Specification V2.1, http://www.flexray.com
21. Zarandi, H.R., Miremadi, S.G., Ejlali, A.: Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models. In: 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 485–492. IEEE Press, Boston (2003)
22. Armengaud, E., Rothensteiner, F., Steininger, A., Horauer, M.: A Method for Bit Level Test and Diagnosis of Communication Services. In: IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems 2005 (DDECS 2005), p. 6. IEEE Press, Hungary (2005)