

Permanent faults on LIN networks: On-line test generation

*Original*

Permanent faults on LIN networks: On-line test generation / Vaskova, A.; Portela Garcia, M.; Garcia Valderas, M.; Lopez Ongil, C.; SONZA REORDA, Matteo. - STAMPA. - (2014), pp. 176-181. ( 2014 IEEE 20th International On-Line Testing Symposium (IOLTS)) [10.1109/IOLTS.2014.6873665].

*Availability:*

This version is available at: 11583/2559937 since:

*Publisher:*

IEEE - INST ELECTRICAL ELECTRONICS ENGINEERS INC

*Published*

DOI:10.1109/IOLTS.2014.6873665

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

(Article begins on next page)

# Permanent Faults on LIN Networks: On-line Test Generation

A. Vaskova, M. Portela-García, M. García-  
Valderas, C. López-Ongil  
Electronic Technology Department  
Carlos III University of Madrid  
Leganés, Spain  
avaskova@ing.uc3m.es

M. Sonza Reorda  
Dipartimento di Automatica e Informatica  
Politecnico di Torino  
Torino, Italy  
matteo.sonzareorda@polito.it

**Abstract.**— Permanent faults (e.g., due to electronic components aging) represent a real problem in nowadays digital systems working in automotive vehicles. Mandatory tests should be done at the vehicle key-on in order to detect damaged elements. Generation and validation of these tests can be improved in a great manner considering the characteristics of the target distributed subsystems. In this work, an approach for in-field detection of permanent faults in a LIN network is proposed.

**Keywords-** Permanent Faults, LIN Bus, On-line Testing

## I. INTRODUCTION<sup>1</sup>

Due to the increasing usage of electronic systems in many different application domains, there is a growing attention towards the effects of faults affecting them. Industries ask for effective techniques able to detect possible faults not only at the end of the manufacturing process, but even in-field, so that a faulty system can be early labeled as such, and the negative consequences of the faults can be limited.

In several domains, standards and regulations explicitly mandate for fault coverage figures to be achieved by in-field test. For example, the ISO 2626 standard for automotive applications [1] requires that 90% of static permanent faults are detected even when the Automotive Safety Integrity Level (ASIL) is B, denoting a relatively low level of required safety. Obviously, higher figures are required when the ASIL is C or D.

As a consequence, companies are aiming at devising effective solutions to perform in-field test, able to match the different requirements coming from the product specification, e.g., in terms of cost, design effort, available time, power. Some of these solutions are based on Design for Testability (DfT), which typically allows the designer to achieve good defect coverage with very limited time requirements. On the other side, the adoption of DfT solutions requires introducing some changes into the hardware, which is not always feasible in practice, e.g., because the adopted devices are provided by third parties, or because the related cost is not acceptable.

In other cases, a functional approach is preferred, in which the system is properly stimulated through its functional inputs, and its behavior observed, thus detecting possible failures. This approach does not require any change into the hardware, but

the defect coverage it can achieve strongly depends on how good the stimuli are, and on the constraints coming from the environment the system is embedded in. Moreover, the cost for developing the stimuli may be relevant, especially because we still lack suitable EDA tools to automate this phase.

In this paper we focus on the LIN network, which is commonly used in the automotive domain to connect low speed sensors and actuators, and explore the possibility offered by a functional test. A small but representative LIN network is considered, and the percentage of permanent stuck-at faults that can be detected by such a test is evaluated. Moreover, an analysis of the faults that cannot be detected in this way is performed, leading to the identification of “on-line functionally untestable faults” [1] that cannot affect the behavior of the network in the considered configuration. Finally, some guidelines to improve the test stimuli in order to increase the fault coverage are provided.

The paper is organized as follows. Section II summarizes previous work in the in-field detection of permanent faults for distributed networks, in the automotive sector. Section III details the system under test chosen (LIN network) and the fault injection campaign performed. Section IV describes a methodology applied for improving functional workloads for in-field aging detection. Finally, section V states the conclusions of the presented work.

## II. PREVIOUS WORK

Robustness of communication protocols used in automotive applications is an interesting issue for scientific community and industry [3]. In particular, there are several researching works that are aimed at improving in-field fault detection and diagnosis in automotive networks with CAN, FlexRay or LIN protocols.

In [4], the authors propose reusing the manufacturing test infrastructures available in integrated circuits to perform test in-field of automotive systems. The approach consists in adding a programmable component in the Electronic Control Unit (ECU) to test that manages the test infrastructures. Experiments with permanent faults require a test time in the order of minutes.

In [5] and [6] presented approaches are focused on CAN protocol. In order to diagnose faults in CAN nodes, in [5], a dedicated hardware is added to monitor transmit and receive error counters. Thus, the approach is based on the fault

---

<sup>1</sup> This work was supported in part by the Spanish Ministry of Science and Technology; code TEC2010-22095-C03-03. RENASER+ project.

detection mechanisms of the CAN protocol. In [6], the authors present a method to check the startup process in a vehicle: a start authorization supervisory control algorithm. This method is based on exchanging messages among the CAN network nodes, checking if the responses are positive in a proper time range. It is not aimed at detecting permanent or transient fault in the electronic modules but at detecting possible system errors like the start button has not been pressed enough time.

FlexRay has been also widely studied since it is assumed like the main communication protocol for future automotive systems. In [7], the behavior of a FlexRay implementation under transient faults is analyzed and, starting from the observed fault effects (freeze errors), a low cost on-line mechanism is proposed in order to detect those kinds of errors (diagnosis) by including an error monitor. In [8], an approach to test FlexRay networks is proposed. It consists in connecting a tester to the bus in order to observe the traffic and inject faults to study their effects. In this case, the main objective is test the module instead of perform fault detection, but it can be applied on-line.

LIN bus imposes softer dependability requirements than CAN or FlexRay and there are less researching works with it. In [9] and [10], methods applicable at design stage are proposed, but there are no works with LIN networks about in-field fault detection. Authors of [9] present a LIN tester applicable during design stage, and in [10], a sensitivity analysis against transient faults of a given LIN controller is presented. However, the increasing number of ECUs in automotive applications involves more complex networks where LIN bus is used in sub-networks instead of CAN and in-field test is becoming a concern.

Summarizing, most of the existing works related to in-field test over automotive communication protocols are based on using additional hardware modules or the available test infrastructures. Besides, LIN protocol has not been studied in this sense to the best of our knowledge. In this work, we propose an in-field test method based on functional issues for LIN networks considering permanent faults.

### III. PERMANENT FAULT INJECTION CAMPAIGN

Fault injection through hardware emulation of circuit under test, on reconfigurable devices (FPGAs) has become a common solution for early sensitivity assessment during first stages of design cycle. There are many tools performing transient fault injection campaigns on circuits descriptions or soft cores through emulation on FPGAs [11][12][13]. These campaigns are intended to test the effect of ionizing radiation and to check the on-line testing capabilities of the circuit. Early identification of weak elements in the circuit, as well as hardening techniques insertion and evaluation are the main advantages provided by these tools.

In-field permanent faults detection has been adopted recently as an efficient technique against devices aging. Specific workload execution at the startup of electronic systems and subsystems in many applications (automotive, industrial, medical equipment, etc.) detects permanent faults caused by device aging.

Efficient test workloads are difficult to generate and to apply. Manufacturing test is developed considering every circuit output is continuously observable. In-field tests imply many restrictions in terms of controlling and observing circuit inputs and outputs. Two requirements are imposed in many applications. First, workload should be the only tool to detect permanent faults (no internal test structures can be used). Secondly, although solutions are proposed for microprocessor-based systems, few approaches are focused on protocols in distributed networks. Therefore, algorithms and tools for helping in the generation of these specific workloads are required by industry and researched by academia.

In this sense, permanent fault injection through hardware emulation, for assessing test capabilities is currently a reality in many industrial applications. In this work, authors have adopted the Autonomous Emulation [11] as the fault injection tool for proposing and proving a methodology for in-field permanent faults detection in distributed networks of automotive applications.

#### A. Permanent Fault Injection Tool

Autonomous Emulation is a tool for sensitivity assessment of digital circuits through transient fault injection on memory elements. It provides very good fault injection rates ( $10^6$  faults/s) and enables complete fault list injection (in every memory element in every workload clock cycle), giving not only representative cross section results, but also locating the weakest areas in the circuit to be hardened. Authors have enhanced this tool, including the permanent fault injection capability on every flip-flop and starting the fault effect from any clock cycle of the workload.

In this tool, two copies of the circuit memory elements under test are prototyped on an FPGA, the golden and the faulty. Parallel execution of both versions of the circuit allows cycle-to-cycle comparison at the circuit outputs and in all the internal memory elements. The injected faults are classified by the produced effect on the system behavior according to the following categories:

1. The effect of the fault is propagated to the outputs
  - a. A fault is classified as *Detected* when some of the error detection mechanisms implemented in the LIN module is able to detect the fault (e.g., the interruption flag becomes active when not expected or vice versa).
  - b. A fault is classified as *Failure* when a faulty transmission is performed but the LIN core does not detect any misbehavior (i.e., Data are corrupted but Parity and Checksum are correct).
2. The effect of the fault is not propagated to the outputs:
  - a. A fault is classified as *Silent* if its effect has completely disappeared from the circuit
  - b. A fault is classified as *Latent* if its effects are still remaining in some memory elements of the circuit.

#### B. System under test

The target distributed system chosen to evaluate the methodology is a LIN network. The LIN protocol is an open serial communication protocol for distributed electronic devices in vehicles. The low cost of implementation and the

easy scalability of the system make it suitable for no safety-critical car distributed systems, where low bit-rates are required.

A small but representative LIN network is considered, taken from an existing application note by Xilinx™ [14], composed of two LIN nodes (a master and a slave). The design is provided in VHDL, and prototyping is proposed for a cheap and low power consumption device (CPLD CoolRunner-II [15]), although any technology can be selected. Xilinx provides also a functional workload which applies main commands of the LIN protocol, as well as checking on error detection nodes capability (faults are injected in the LIN bus).

Every LIN node is divided into eight blocks, as shown in Figure 1. The *Transmitter* and *Receiver* blocks are in charge of serialize/de-serialize data, while *Parity\_Gen* and *Checksum\_Gen* are in charge of detecting errors in the received data. Transmitter and receiver are implemented with two shift registers, while received data are preprocessed in a specific block (*majority sampler*) which detects bits according to the standard time windows. The control part of the circuit is composed of some *Configuration\_Registers* and a *Control block*, which is composed of a set of finite state machines. The frame reception and composition is controlled by two (slave) or three (master) finite state machines, located in this *Control Block*. These finite state machines are also in charge of error detection and reporting, as well as header decoding.

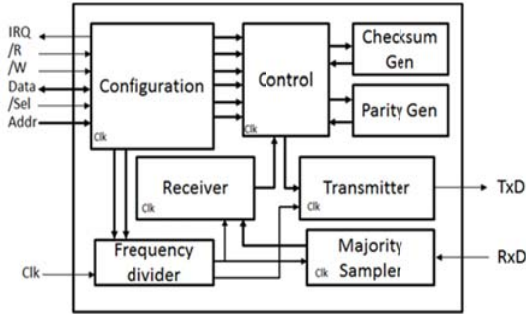


Figure 1. Architecture of a LIN bus controller

The functional workload provided includes eleven different tests. First, there are four basic transmission tests from master to slave. The first three tests send 2, 4 and 8 bytes, respectively; while the fourth test sends an ID data and the slave must respond. Basic tests verify that the two nodes communicate successfully. Other tests cases, like parity error, checksum error detection and framing error required to force the bus directly. The typical test scheme involves writing to a configuration register, waiting for an interruption, and then acting on that interruption, reading a received byte or checking the various status registers.

In a LIN network, LIN slaves are hardly observable from the point of view of upper levels in the whole vehicle system. In slave elements the LIN node interface (Figure 2) is connected to payloads except to Rx/Tx LIN bus signals. Generally, only master node in a LIN network is easily readable and writable from upper levels (Figure 3). Therefore, controllability and observability of internal elements in the slave nodes of the network is restricted to the LIN bus.

### C. Fault Injection Campaign

The fault injection campaign has been performed over the two LIN cores in the network (a master and a slave). A single permanent fault has been injected in every flip-flop (out of the 410 FFs) and at the beginning of the workload (37,381 clock cycles). Only sequential elements in the circuit have been considered in this study. Authors plan to extend the analysis to combinational elements. Observation points are the Data Bus and the Interruption Flag outputs from the master node (see Figure 2 and Figure 3).

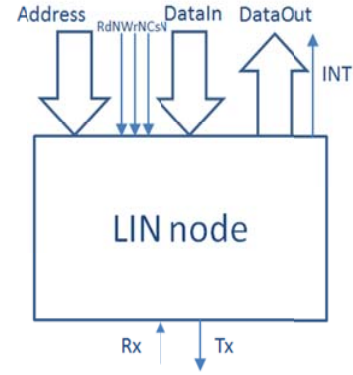


Figure 2. Interface of a LIN node

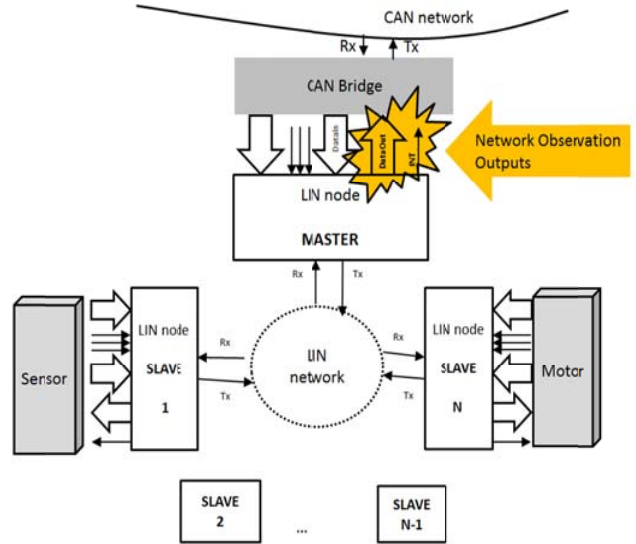


Figure 3. Observable outputs in a typical LIN network

Table 1 and Table 2 show the results (for master node) of the Fault Injection campaign for *Stuck-at-0* and *Stuck-at-1*, respectively. The tables report the fault classification per circuit blocks. The considered blocks (see Figure 1) are the configuration registers, the control block, the frequency divider, the reception blocks (receiver and majority sampler), the transmitter and the checksum generator.

As a global result, with this functional workload around 93% of stuck-at-1 faults are detected in the memory elements of the LIN master node, and around 68% of stuck-at-0 faults are detected. This difference in results is mainly due to the

initialization value of most of flip-flops ('0') which helps in the detection of *stuck-at-1* faults. With respect to the other fault classifications, around 30% of the *stuck-at-0* faults is not provoking different state values (silent), while around 2% of them remains latent in the circuit, but no propagation is activated to the circuit outputs. This result differs with respect to *stuck-at-1* faults, where controllability is better (only 4% of silent faults) but observability is slightly worse (3%).

TABLE 1. BLOCK GROUPED FAULT CLASSIFICATION FOR MASTER NODE, STUCK-AT-0

	#FF	Failure	Latent	Silent
CONFREG	56	27	0	29
CRL_CHECKSUM	2	0	0	2
CRL_COUNTER	22	19	0	3
CRL_ERROR	9	3	0	6
CRL_FRAME	4	4	0	0
CRL_MASTER	9	9	0	0
CRL_SLAVE	23	21	2	0
CRL_RECEIVE	9	7	1	1
CRL_SER_IN_LAST	1	0	1	0
CRL_STATUS	3	3	0	0
DIVIDER	17	6	1	10
RECEIVER	23	23	0	0
SAMPLER	4	4	0	0
TRANSMITTER	14	13	0	1
CHECKSUM	9	0	0	9
<b>Total</b>	<b>205</b>	<b>139</b> 67,80%	<b>5</b> 2,44%	<b>61</b> 29,76%

TABLE 2. BLOCK GROUPED FAULT CLASSIFICATION FOR MASTER NODE, STUCK-AT-1

	#FF	Failure	Latent	Silent
CONFREG	56	52	0	4
CRL_CHECKSUM	2	0	2	0
CRL_COUNTER	22	18	3	1
CRL_ERROR	9	9	0	0
CRL_FRAME	4	4	0	0
CRL_MASTER	9	9	0	0
CRL_SLAVE	23	21	0	2
CRL_RECEIVE	9	7	1	1
CRL_SER_IN_LAST	1	1	0	0
CRL_STATUS	3	3	0	0
DIVIDER	17	17	0	0
RECEIVER	23	23	0	0
SAMPLER	4	4	0	0
TRANSMITTER	14	14	0	0
CHECKSUM	9	9	0	0
<b>Total</b>	<b>205</b>	<b>191</b> 93,17%	<b>6</b> 2,93%	<b>8</b> 3,90%

Analyzing in detail the fault classification on every block in the LIN master node, there are three blocks where no stuck-at-0 fault is detected at the node outputs; all of them are related with the Checksum calculation and checking. Even, in one of them (Control part related to Checksum) no stuck-at-1 fault is detected. But the blocks with worse results are Configuration Registers (only a 48% of permanent stuck-at-0 faults are detected) and the Frequency Divider (only a 35% of permanent stuck-at-0 faults are detected) where even no control action is taken. With respect to stuck-at-1 better detection capability is observed with this workload, although four flip-flops in Configuration Registers block remain untested.

As mentioned before, the ISO 2626 standard for automotive

applications requires that 90% of static permanent faults are detected even when the Automotive Safety Integrity Level (ASIL) is B, denoting a low level of required safety. Therefore, improving this functional workload is mandatory, especially for *stuck-at-0* faults.

On the other hand, if we analyze the fault classification for the Slave nodes, worse results are obtained. Configuration registers are not directly accessible through LIN network bus. Therefore the effects of faults injected in these elements are only observable through malfunctioning in commands processing (frame decoding in Control Block). Table 3 and Table 4 detail results for Slave node. Only around the 16% of injected *stuck-at-0* faults are detected and around the 22% of injected *stuck-at-1* faults. Internal blocks with lower fault coverages are similar to master node; the only difference is *Transmitter* block which cannot be observed anyway in the slave nodes, from the master side.

TABLE 3. BLOCK GROUPED FAULT CLASSIFICATION FOR SLAVE NODE, STUCK-AT-0

	#FF	Failure	Latent	Silent
CONFREG	56	8	9	39
CRL_CHECKSUM	2	1	0	1
CRL_COUNTER	22	5	1	16
CRL_ERROR	9	1	0	8
CRL_FRAME	4	1	1	2
CRL_MASTER	9	2	0	7
CRL_SLAVE	23	3	1	19
CRL_RECEIVE	9	0	2	7
CRL_SER_IN_LAST	1	0	0	1
CRL_STATUS	3	0	1	2
DIVIDER	17	1	3	13
RECEIVER	23	3	0	20
SAMPLER	4	1	0	3
TRANSMITTER	14	4	0	10
CHECKSUM	9	2	0	7
<b>Total</b>	<b>205</b>	<b>32</b> 15,61%	<b>18</b> 8,78%	<b>155</b> 75,61%

TABLE 4. BLOCK GROUPED FAULT CLASSIFICATION FOR SLAVE NODE, STUCK-AT-1

	#FF	Failure	Latent	Silent
CONFREG	56	13	0	43
CRL_CHECKSUM	2	1	0	1
CRL_COUNTER	22	4	0	18
CRL_ERROR	9	4	0	5
CRL_FRAME	4	0	0	4
CRL_MASTER	9	3	0	6
CRL_SLAVE	23	4	0	19
CRL_RECEIVE	9	2	0	7
CRL_SER_IN_LAST	1	1	0	0
CRL_STATUS	3	0	0	3
DIVIDER	17	3	0	14
RECEIVER	23	5	0	18
SAMPLER	4	1	0	3
TRANSMITTER	14	2	0	12
CHECKSUM	9	2	0	7
<b>Total</b>	<b>205</b>	<b>45</b> 21,95%	<b>0</b> 0,00%	<b>160</b> 78,05%

#### IV. EFFICIENT SELF-TEST ON STARTING THE SYSTEM

In-field detection of permanent faults in distributed systems for automotive applications requires an efficient workload,

developed in the early stages of design cycle, which takes into account the controllability and observability of every node in the network.

#### A. Efficient Workload generation

Main elements required for this workload generation are:

- *Circuit description*, ready to be simulated or emulated.
- *Fault injection tool*, capable of injecting permanent faults in the internal elements of the network nodes.
- *Functional workload* to start with. This workload should check main functionality of the circuit/network considering operation modes, error modes, configuration and commands/data interchange.

The methodology followed for achieving high fault coverage is composed of four steps:

1. **Permanent fault injection campaign** on the circuit descriptions with the functional workload provided by manufacturer or developed by test engineer, attending circuit specifications.
2. **Fault classification** for *stuck-at-0* and *stuck-at-1* faults. Failure classification computation and untested faults identification.
3. **Location of uncovered elements** with the workload applied. Identification of Latent and Silent faults.
  - a. **Latent faults** imply the fault is injected but there is no propagation of its effect to the observable outputs. Observability should be assured.
  - b. **Silent faults** imply the fault is not injected correctly. A lack of controllability is identified.
4. **Re-generation of new workload** which assures the injection of all possible faults and the observation of their effects where failure classification is done.
5. **Repeat steps again from 1 to 4** till fault coverage reaches 90% for *stuck-at-0* and *stuck-at-1*.

The enhancing of workload to assure fault injection in the internal elements that produce *Silent faults*, suppose the identification and activation of writing procedures for these memory elements, according to Protocol specification.

On the other hand, observability improving for those elements that produce *Latent faults*, imply the generation of reading procedures for internal registers, but also checking frequency dividers, finite state machines and error detection mechanisms.

After some iteration, test engineers will discover there are some faults that are untestable for the systems under analysis. These faults arise two options. First, affected elements are not used during a typical workload, so permanent faults have no

importance. Secondly, affected elements are not visible within the system under analysis (distributed network where only some nodes are visible like LIN network). In this latter case, the correct behavior of the system is also assured, from the point of view of network operation, but some functions will be degraded (such as sensor data processing, control of actuators, etc.), Figure 3.

Automatic tool for permanent fault injection on a circuit description is required to apply this methodology for efficient *test* workloads generation. This tool should classify the effects of injected faults in three categories at least: *Failure*, *Silent* and *Latent*.

#### B. Efficient workload for LIN network

Experimental results presented in previous sections, on a LIN network, show the fault coverage for *stuck-at-0* faults with a functional workload is lower than the standard requirements (ISO-26262, ASIL level B, 90%), while the fault coverage for *stuck-at-1* faults is higher than 90%. Those results presented in section III are referred to faults injected in the master LIN node. When considering faults in the slave nodes, fault coverage is much less.

Authors have applied the methodology detail in subsection A, identifying Latent and Silent faults and including writing and reading commands in the original workload, as well as more configuration modes for frequency dividers. Only one iteration step has been performed. From the original workload (37,381 clock cycles) 6,776 extra test vectors have been added. The new classification results are shown in Table 5 and Table 6 for master node. In this module, most of *stuck-at-1* faults are detected (96%), while there are still elements in the Configuration Registers which are not controlled with this workload for *stuck-at-0*. This implies fault coverage for *stuck-at-0* faults is 89%. Further iterations can be done in order to improve the fault coverage, especially from the point of view of the slave nodes, whose results have not been improved at all.

TABLE 5. BLOCK GROUPED FAULT CLASSIFICATION FOR MASTER NODE, STUCK-AT-0 WITH IMPROVED WORKLOAD

	#FF	Failure	Latent	Silent
CONFREG	56	43	1	12
CRL_CHECKSUM	2	2	0	0
CRL_COUNTER	22	20	0	2
CRL_ERROR	9	8	0	1
CRL_FRAME	4	3	1	0
CRL_MASTER	9	9	0	0
CRL_SLAVE	23	23	0	0
CRL_RECEIVE	9	7	0	2
CRL_SER_IN_LAST	1	1	0	0
CRL_STATUS	3	3	0	0
DIVIDER	17	17	0	0
RECEIVER	23	21	0	2
SAMPLER	4	4	0	0
TRANSMITTER	14	14	0	0
CHECKSUM	9	8	0	1
<b>Total</b>	<b>205</b>	<b>183</b> 89,27%	<b>2</b> 0,98%	<b>20</b> 9,76%

The main conclusion of these results is that, with a functional workload it is possible to detect permanent faults in master nodes of a LIN network (e.g., due to device aging); while slave nodes require further efforts. In general, due to initialization scheme, *stuck-at-0* faults are harder to be detected than *stuck-at-1*. The use of a fault injection tool in the early stages of design cycle will help in the generation of an efficient workload to detect aging effects in the final system. This is especially relevant if distributed system can be emulated/simulated with the observation points selected by the test engineer.

TABLE 6. BLOCK GROUPED FAULT CLASSIFICATION FOR MASTER NODE, STUCK-AT-1 WITH IMPROVED WORKLOAD

	#FF	Failure	Latent	Silent
CONFREG	56	54	2	0
CRL_CHECKSUM	2	2	0	0
CRL_COUNTER	22	22	0	0
CRL_ERROR	9	9	0	0
CRL_FRAME	4	2	1	1
CRL_MASTER	9	8	1	0
CRL_SLAVE	23	23	0	0
CRL_RECEIVE	9	9	0	0
CRL_SER_IN_LAST	1	1	0	0
CRL_STATUS	3	3	0	0
DIVIDER	17	17	0	0
RECEIVER	23	22	1	0
SAMPLER	4	4	0	0
TRANSMITTER	14	13	0	1
CHECKSUM	9	9	0	0
<i>Total</i>	205	198	5	2
		96,59%	2,44%	0,98%

For those untestable faults, not accessible due to the network architecture, additional solutions should be proposed. OEM companies can hardly accept any hardware modification in the LIN nodes, as time and cost of the final product would raise significantly. In this sense, most critical slave LIN nodes could be externally tested with some signature monitoring.

## V. CONCLUSIONS

In this work, we analyzed the presence of permanent faults on a LIN network and their detection through functional workloads. The paper proposes a methodology for improving these functional workloads, thanks to the localization of not detected faults with the application of a permanent fault injection campaign, as well as the categorization of the effects of these undetected faults. Specific characteristics of LIN networks are considered with respect to fault injection and effects observation, but the methodology detailed is applicable to any network of distributed nodes.

## VI. REFERENCES

- [1] ISO 26262-1:2011 "Road vehicles-Functional Safety". (www.iso.org)
- [2] P. Bernardi, M. Bonazza, E. Sanchez, M. Sonza Reorda, O. Ballan, "On-line functionally untestable fault identification in embedded processor cores", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013, pp. 1462 - 1467
- [3] Book Chapter: J. Suwatthikul, "Fault detection and diagnosis for in-vehicle networks". Book: "Fault Detection", Wei Zhang (Ed.), ISBN:

- 978-953-307-037-7, InTech, Available from: <http://www.intechopen.com/books/faultdetection/fault-detection-and-diagnosis-for-in-vehicle-networks>
- [4] A. Cook, et al. "Reuse of Structural Volume Test Methods for In-System Testing of Automotive ASICs", 21<sup>st</sup> Asian Test Symposium, pp. 214-219, 2012.
- [5] H. Huangshui, Q. Guihe, "Online Fault Diagnosis for Controller Area Networks", International Conference on Intelligent Computation Technology and Automation, pp. 452-455, 2011.
- [6] S. Thanagasundram et al. "Failure Management for Reliable Automotive Start Up Process", IEEE International Conference on Computer Science and Automation Engineering, pp. 215-219, 2011.
- [7] Y. Sedaghat, S. G. Miremadi, "A Low-Cost On-Line Monitoring Mechanism for the FlexRay Communication Protocol", 4<sup>th</sup> Latin-American Symposium on Dependable Computing, pp. 111-118, 2009.
- [8] E. Armengaud et al. "Towards a Systematic Test for Embedded Automotive Communication Systems", IEEE Transactions on Industrial Informatics, Vol. 4, No. 3, August 2008.
- [9] M. Popa, V. Graz, A. Botas, "Lin Bus Testing Software", Canadian Conference on Electrical and Computer Engineering, pp. 1287-1290, 2006.
- [10] A. Vaskova et al. "Hardening of serial communication protocols for potentially critical systems in automotive applications: LIN bus", 19<sup>th</sup> IEEE International On-Line Testing Symposium, pp. 13-18, 2013.
- [11] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena, "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation," IEEE Trans. Nucl. Sci., vol. 54, no. 1, pp. 252-261, Feb. 2007.
- [12] J.M. Mogollón et al. "FTUnshades2: A Novel Platform for Early Evaluation of Robustness Against SEEs" RADECS 2011. Seville, Spain.
- [13] M. Alderighi et al. "Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform" 22<sup>nd</sup> IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems. 2007. pp 105-113
- [14] XAPP432 (v1.1) Xilinx Application Note "Implementing a LIN Controller on a CoolRunner-II CPLD", April 3, 2007
- [15] "CoolRunner-II CPLD Family, v3.1, DS090, Product Specification", Xilinx Inc. Sept. 2008 (www.xilinx.com)