# An New Methodology on Fault Clearance Technique for Intra-Vehicular Networks

R.Latha

[1] Department of Electrical & Electronics Engineering, KCG CT, Chennai, India.

**Abstract**—The increase in electronic components and sensors of contemporary automobiles raises the complexity of network design. The development of automotive electronics reinforces the significance of an optimal and fault tolerant hybrid network via different communication protocols. CAN (Controller Area Network) is globally intended for vehicle segments to communicate with electronic units like Engine Management System, Stability Control Units, Braking Systems, and Door functions. The CAN and LIN (Local Interconnect Network) are taken into consideration to enable the implementation of hierarchical vehicle network gateway for quality fortification and cost dwindling of vehicles. The standardization will diminish the assorted on hand low-end multiplex solutions among development cost, production rate, service fee, and logistics charges of vehicle electronics. The proposed hybrid architecture leads to the gateway implementation in the electronic units. It encompasses the capability to share the data between various networking protocols with optimum utilization of the available control information. This system uses two separate gateways for CAN and LIN which efficiently differentiates the high-speed and low-speed applications pertinent to critical ECUs in the network.

***Keywords*—Controller Area Network, Local Interconnect Network, Intra-Vehicular Communication, Vehicular Faults.**

## I. Introduction

As the number of electronic control units (ECUs) within the vehicle increases, it significantly raises the electrical complexity on board. i.e., sensors integrated upon ECUs is getting upsurge twice of that for every 10 years. Due to the evolution of automotive electronics, the number of ECUs in a vehicle has surpassed over hundred [1]. Consequently, there is a complex electronics system existing within the vehicle. Hence the system requires an efficient gateway to interact between several ECUs to enable safety, fuel efficient, expediency and infotainment. Fig. 1 signifies to realize a number of functions that have been incorporated within single and multiple ECUs from the 1970s to present.

In consonance with the user expectation vehicle demand to guarantee informations such as traffic management, easy maintainance, and effective infotainment system. All these are possible with safer roads, driver comfort, passenger safety and data exchange between nearby vehicles.

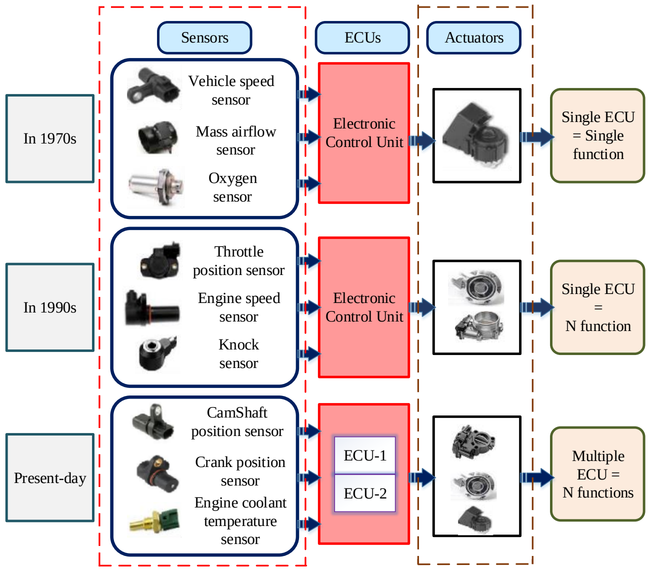


Fig. 1. Progression of ECUs

Vehicular communication can be categorized as Inter-vehicular communication which is managed by wireless networks and organized by vehicular ad-hoc networks (VANETS) pertaining to enable communication with neighboring vehicles [2]. By enabling the incessant exchange of periodic and event-triggered information, intelligent vehicles can enhance road safety and provide support for console applications. Nizar Alsharif *et.al* has proposed connectivity aware routing protocol in VANETS to increase the routing performance with the aid of selecting routing paths in a dynamic manner, data off-loading, Internet-based services and less delivery delay. Examples are vehicle to vehicle V2V communication, Traffic management, and multimedia transmissions. In view of Intra-vehicular communication which is managed by wired interfaces (LIN, CAN, and FlexRay) for barter of data between several ECUs within the vehicle. Examples are safety and navigation, chassis, and distributed control system based applications, human machine interface (HMI), global positioning system (GPS), communication functions like radio, antenna etc. A massive amount of sensors as well as processors are used in several parts of the vehicle, in turn, to handle time critical functions like airbags, emergency call, anti-lock brakes, electronic stability control. While camera plays an important role in resolving greater challenges like environmental sensing and vehicle to vehicle V2V communication. The serial network protocols like CAN, LIN, FlexRay and multimedia-oriented system transport (MOST) are proposed for In-Vehicle Networks (IVNs) [3].

In the vehicular electronic systems, information from the ECU is directed towards the respective field buses. The information trail with different automotive devices, field buses connected upon, conversion of dissimilar entities, and also handles the different bus speed via integrating the hybrid gateway. The gateway can recognize an abstraction of various physical layers and different protocols. Automotive buses (CAN, MOST, LIN, and FlexRay) can be added easily owing to the flexible software design. An application bus will allow third party application to transmit and receive information on the buses by means of hybrid gateway [4]. Ethernet controller is used for high bandwidth applications like flashing and diagnostics to have the access with the hybrid gateway and linked automotive buses as shown in Fig. 2.

The gateways framework provides cutting-edge utilities that comprise of fault management, calibration of vehicles and software reprogramming [5]. Based on the standardized interfaces across manufacturers Automotive Open System Architecture AUTOSAR R4.0 complies with functional safety standard ISO26262 [6]. In current software development environment, software reusability is not possible due to transparency with OEM's was denial. Hence AUTOSAR provides a common software infrastructure dedicated to automotive applications that impact on achieving the goals alike to reduced development time and costs, reusability of software increases to strengthen the quality and efficiency [7].

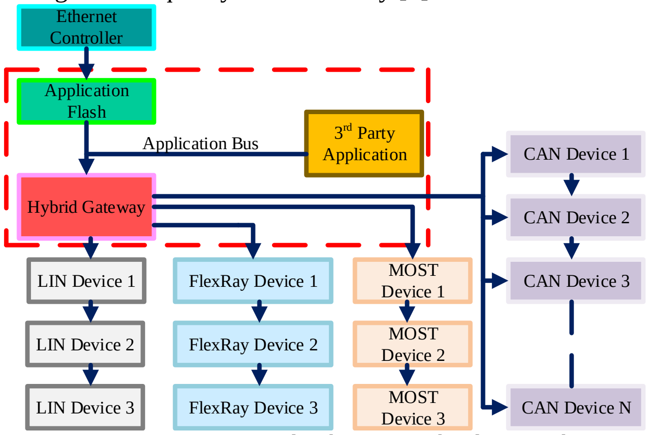


Fig. 2. Gateway associated with existing vehicular networks

The main challenges that are faced in Vehicular Adhoc Networks are scalability, bandwidth limitation, privacy, and safety. As the intricacy in the ECU rises the network should be adaptable and the information must effectively distribute through the network in dense situations. Therefore relevance based techniques will provide a wide-ranging concepts that functions on complicated wired networks and control the delivery of information where it is necessarily required. Since the data sensed by the vehicle is communicated to the user which may contain any private information. Thus to authenticate in-vehicle report pseudonym based approach can be used for avoiding anonymous communications. Whereas there will be a significant trouble in managing the pseudonym. The wired counterparts with in-vehicle scenario are much distributed upon vehicular networks. Consequently it adds more responsibilities among the nodes for successful data delivery along with effective bandwidth utilization among the available nodes. The understanding of global positioning system and vehicular wired communications combined with intra-vehicle computation and information sensing capabilities will provide outstanding development in safety issues[22]. From the user perspective constantly the vehicle was expected to deliver the safety information through signaling and voice based communication thereby to evade the catastrophic failure. In this work the CAN and LIN protocol are taken into account since it diminishes wiring and distributed control which enhances the system performance. Both protocols will offer the ability to operate in various electrical environments and guarantee noise free transmission. It present fault free broadcast as every node can check for errors during the transmission of information and send the error frame.

The remainder of this paper provides the information on the deployment of central network gateway based on CAN and LIN to achieve fast and effective communication between control modules which could minimize the complexity as well. Section II discusses the overview of CAN and LIN interfaces related to arbitration and error management principles. Section III states the hybrid network gateway operating mechanisms and illustrated the design flow of CAN and LIN networks using the algorithms. Experimental results are presented in section IV, and section V concludes this article.

## II. CAN AND LIN OVERVIEW

### A. *Controller Area Network Overview*

CAN network is widely used to transfer the greater part of in-vehicle communication signals. It has features like multi-master, safety, arbitration, speed, and distance. IVNs often use the CAN for node connected sensor based applications, which offers a data rate up to 1MBPS [8]. Although several networks have been proposed, none of them meet the requirements that CAN accomplish. The foremost reason for the continuous reach of CAN protocol is its relatively low cost. As quite large energy efficient and sophisticated CAN transceivers are available in the market, the hardware cost of the CAN network has an immense downhill. The CAN protocol employs carrier sense multiple access with a collision avoidance (CSMA/CA) mechanism to arbitrate access to the bus (CIA, 2007). It uses a priority mechanism by means of numerical identifiers to overcome collision when two or more nodes wish to transmit concurrently. It is illustrated from Fig. 3, on the CAN bus a dominant bit 'zero' is used to update a recessive 'one' bit. Such that, when there are two nodes Node A and Node B, the first node (Node A) transmitting a one while another node (Node B) transmitting a zero then the bus results in a zero level. When more than one node wishes to transmit, it observes the entire bus to ensure if there is any bus activity taking place. If there is no activity on the bus, then nodes start to transmit their message identifiers (MSB first), prior to checking the bus levels. If one node broadcast a recessive bit over the bus and another transmits a dominant bit, the bus outcome will be a dominant level. Consequently, the node transmitting a recessive bit will notice a dominant bit on the bus (state where B loses) and halts the transition of any

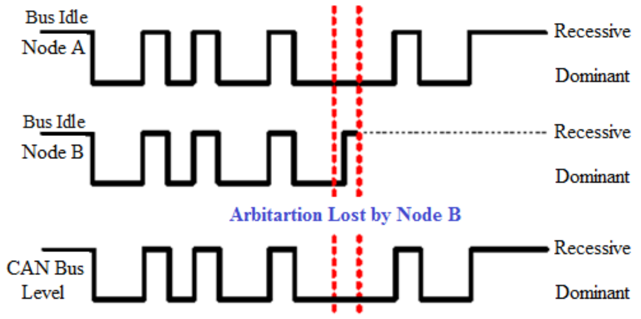


Fig. 3. Controller Area Network (CAN) Bus Arbitration

additional information in sequence. In this state, the node with the lowest message identifier number will gain access to the bus and transmit its message. The node which has lost through the arbitration process then wait until the bus opens before re-transmitting the message. The CAN protocol makes use of the bus arbitration method to guarantee that the node among the highest priority (lowest value in the identifier field) will persist to transmit without having to back off the bus. It affirms that CAN have an expected behavior and is proficient in its use of the bus bandwidth.

## *B. CAN Error (Fault) Handling & Confinement*

The flexibility of systems can be identified by remotely analyzing how the system reacts and processes when an error arises. In conventional error detection method, the data delivery is ensured when receiver handover an acknowledgment (which is commonly the received station address) to the transmitting station. In the CAN perception, identifier 'labeling' message is communicated and received by all members through the network, which causes mandatory execution of the task for error check in every local station extant in the network. To attain this idea, the CAN protocol uses a permutation of positive ($P_{ACK}$) and negative ($N_{ACK}$) acknowledgments [9] and [10]. Based on working out, receiver sends the acknowledgement to the master node. A receiver may send either positive or negative acknowledgement. A dominant bit in the acknowledgement slot represents a positive ACK, whereas a recessive level in the slot represents a negative ACK. The ACK delimiter will be transmitted always most importantly for the purpose of error tracing. As the sender transmits both ACK delimiter and ACK slot in accordance with its characteristics. To ensure the exactitude of the message broadcast to the sender one positive acknowledgement is acceptable. If there is not even single positive ACK and the recessive ACK slot is not updated by any receiver, the processing message transmission will be terminated by sending an error flag while sender detects an error ACK. This error is either caused due to the sender or when there are no receivers on the bus. The $P_{ACK}$ is defined by the expression:

$$P_{ACK} = ACK + \text{(i) for any value of (i)} \quad (1)$$

Where, $P_{ACK}$ is sent from all the accessible stations (i) which received the correct message during a distinct time interval (ACK time slot). Thus, $P_{ACK}$ gives a sign of at least one successful message transfer. The $N_{ACK}$ evinces that there is a minimum one error exist in the whole system.

In the beginning, each station should be having two separate errors offset (counters), one counter will keep an eye on circumstances while the message is being transferred and other will execute a related task during the reception. In accordance with the error type reported and the station working conditions the offsets are incremented to several weightings and definite conditions are decremented accordingly. The purpose of these offsets is to trace the initiating information from all additional stations that are sent directly or indirectly. When too many errors are acquired in a particular station, then station state may switches from 'error active' to 'error passive' state. In which the particular station cannot communicate, whereas active forever in the supervision of errors which may happen on the network. If there are too many transmission errors emerging in the network, then the network could be blocked, making all transfer unfeasible. When error gets detected, an active error flag is sent using bus link by the error-active network node. This is default node-state during reset. An error-passive network node has already accrued with relatively high transmit or receive error count, hence this node monitors a significantly higher error rate over a longer period of time. A node in the bus-off state is restricted to have any influence on the bus. Fig. 4 illustrates the error state diagram of a CAN node. After a reset, a node is in the error-active state. If any of the two error count crosses the value 127, then the monitor will demand the MAC sub-layer and enters into the error-passive state. Although node becomes error active again during both receive and transmit error count drops below the value 128. A node is disconnected from the bus when the transmit error count surpasses 255 and this results in the bus-off state. From the bus-off state, a node can re-enter into error-active state after perceiving 128 sequences of eleven consecutive recessive bits. Immediately the error count will be reconfigured to 0 upon reset. This measure ensures that a possibly erroneous reset node cannot disturb conveyance again right away after reset. Proportionately up to 128 further frames can be transmitted without interrupt even at a very high busload [11].

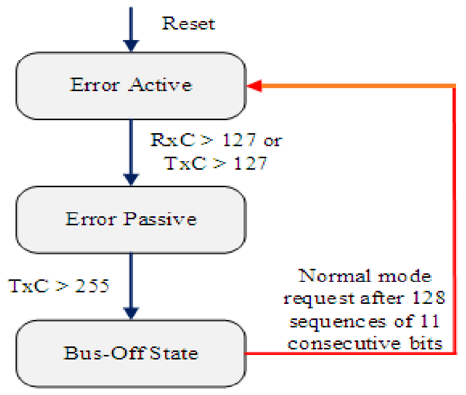


Fig. 4. Controller Area Network (CAN) Bus – Error State Diagram

### C. Local Interconnect Network (LIN) Overview

The LIN protocol is proposed for strong support in controlling mechanical-electronic components existent in distributed systems of automotive applications. LIN exploits the concept of master-slave architecture, where the network contains single master and fixed number of slave nodes.

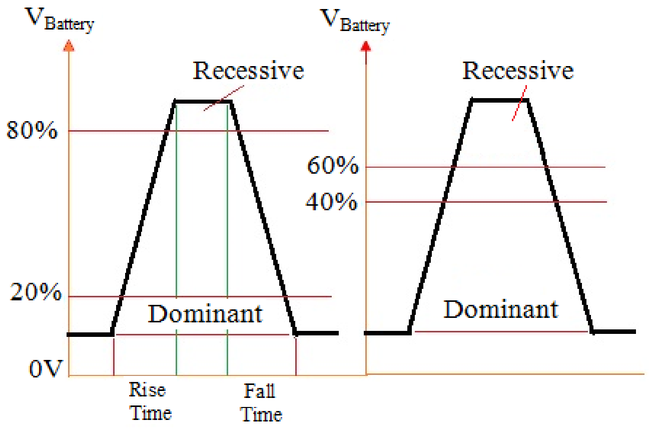


Fig. 5. Local Interconnect Network (LIN) Bus Arbitration

The LIN frame comprises a frame header and response fields. The header contains protected response identifier [12]. Once the LIN master sends frame header message, the LIN slave respond to it by sending a frame. Data is transferred successively as 8 data bits with 1 start and 1 stop bit without parity, thus 10 bits are transmitted per byte. As shown in Fig. 5 data on the bus is separated into recessive (logic high) and dominant (logic low). At the transmitter the least voltage level should be fewer than 20% of the battery voltage ($V_{Battery}$) or approximately 1Volt, henceforth resulting in logic 0. In contrast, the maximal level voltage, which represents logic 1, should be greater than 80% of the battery voltage. At the receiver logic, 0 will be lesser than 60% of the battery voltage, whereas logic 1 will be higher than 60% of the battery voltage. Specific slave node reacts to the identifier and sends the frame response, which holds data and checksum fields. It acts as a sub bus for CAN with limited utility, lower cost, bit rate and reduced bandwidth in the network. The real-time implementation of LIN connection is based on ‘single’ wire, which reduces the cost of cabling and connectors than in CAN [13]. LIN provides guaranteed transmitted signal latency, daisy chain configuration, and ensures the safety of transmitted data by adopting CRC and error detection. Thus LIN can find the faulty nodes in the network besides with reduced complexity compared to conventional UART or SCI based systems.

### D. LIN Error Signaling & Confinement

Due to peerless master architecture, LIN is implausible to accommodate an error signaling tool. Though the errors are identified nearby these error data can be provided on request in the type of diagnostic communication messages. The LIN nodes are capable of discriminating malfunctions from short term to eternal ones and accomplish their own confined diagnoses and get counteractive action.

In this work, proof of concept is carried out using Freescale microcontroller which is predominantly used in automotive applications. The entire test bench consists of single master node and multiple slave nodes. The device results were achieved for a frequency of 25MHz, while the execution speed may be doubled when over clocking of 50MHz. The efficiency of the system completely depends upon the computation speed, memory and low-speed CAN transceiver. To overcome the delay occurred in this device, a Tricore controllers can be deployed without utilizing the controller bandwidth or resources. Through introducing event-triggered messages, average network efficiency can be achieved to overcome the performance issues in LIN protocol. Whereas the major limitation is that, it impairs the worst-case performance and set hurdles network diagnosis as well.

## III. Embedded Vehicular Network Gateway

Newfangled vehicles are controlled electronically via processors in addition to mechanical components. In the vehicle, an ECU will be controlling the devices or subsystems. In general, the processor in an ECU takes the input value from the available sensor then processes the data and distributes the same data within the ECU or another ECU in few clock cycles to furnish efficient performance. The role of ECU contributes in all aspects right from simpler tasks such as wiper movement or brake light control to time critical functions like airbag control and adaptive cruise control. Moreover, the various subsystems in the vehicle encompass in operating the task of processors in order to control actuators and processing sensor values. Precisely in order to fulfill the whole requirements, onboard CAN and LIN networks have been designed. By the moment the information from several ECUs are transferred through the field buses to the hybrid gateway, there is a chance for the occurrence of a fault in one particular CAN node. On that instance, other nodes will be competent in communicating to the gateway irrespective of the particular node. To overcome this issue a switch has been employed with the gateway towards field bus switching to conquer well-organized data delivery with proper diagnostic interface [14].

The role of the gateway is essential to establish a communication between the existing field buses. In future Ethernet will play a dominant role on Intra Vehicular Networks (IVN) [15], due to low cost, higher bandwidth and it can support for vehicle diagnosis and infotainment systems. Similarly, the hybrid gateway architecture has been replaced by backbone based architecture in which every sub-network will be communicating with its own domain control unit (DCU). For example, the CAN and MOST have got separate DCUs. Therefore, the each DCU will act as a dedicated gateway for available sub-networks. At this juncture, the major constraints are developing a hardware-software platform and it should be easy to setup and verify. There should be a different gateway implemented for different vehicle models and/or options should be provided to choose the sub-network and its appropriate gateway for specific vehicle model.

A reliable gateway will be able to solve the message conversion issues over the existing field buses and will focus on implementing efficient gateway mechanism using OSEK/VDX environment [16]. Predominantly the hybrid gateway cannot oversee the functionalities like dynamic routing, parallel reprogramming, security and GUI based software configuration and verification in order to reuse the same software for different vehicle models. Fig. 6 depict the architecture of CAN and LIN protocol working together as a hybrid structure to accrue the efficiency of the entire automobile system. The time critical functions (EMS, ABS, SR) are solely handled by the CAN node and non-critical functions (Temperature Control, Wiper, Cluster, and Infotainment) are governed by the LIN node [17]. The CAN and LIN bus are restrained by a single gateway in which CAN node acts a  master that is connected to many LIN slaves.

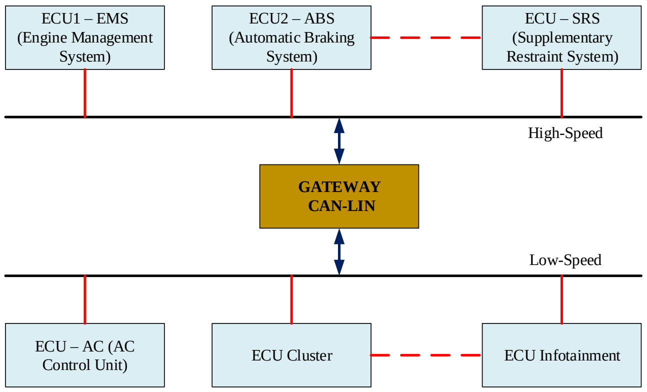

ECU1 – EMS (Engine Management System)
ECU2 – ABS (Automatic Braking System)
ECU – SRS (Supplementary Restraint System)
High-Speed
GATEWAY CAN-LIN
Low-Speed
ECU – AC (AC Control Unit)
ECU Cluster
ECU Infotainment

Fig. 6. CAN/LIN architecture with single gateway approach

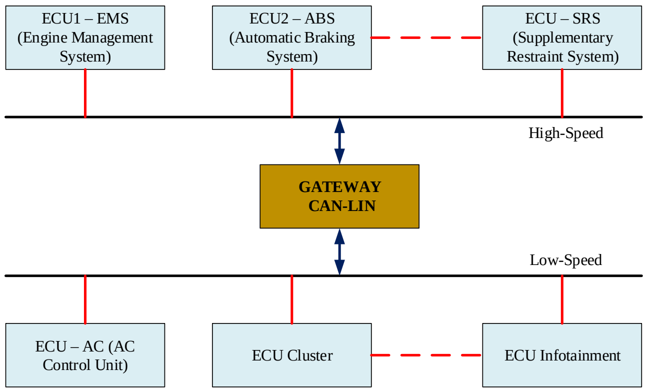

ECU1 – EMS (Engine Management System)
ECU2 – ABS (Automatic Braking System)
ECU – SRS (Supplementary Restraint System)
High-Speed
GATEWAY CAN-LIN
Low-Speed
ECU – AC (AC Control Unit)
ECU Cluster
ECU Infotainment

Fig. 7. Hybrid configuration handling High/Low-speed Applications

In accordance with single gateway approach, the critical and non-critical applications are presided over by a single gateway. It represents the hybrid structure, where the high/low-speed applications are differentiated. As shown in Fig. 7 green lines indicates that, if the CAN node turn out to be faulty, the LIN node starts operating and control the functions handled by the CAN momentarily until the CAN node is fixed. On the other hand, red lines point out that, if LIN node fails, the CAN node will perform the utility handled by the LIN temporarily until the CAN node gets resolved as point out by the red lines.

In the proposed architecture CAN node is implemented using Freescale MC9S12XDP512 16-bit microcontroller in code warrior IDE. It uses MC33388 high-speed CAN transceiver and furthermore uses MC33661 low-speed LIN transceiver. The CAN module is constructed in order to simulate and guarantee the data delivery. If any error arises at CAN node, the LIN onset functioning momentarily till CAN node is resuming back. This architecture has another additional feature i.e., LIN node is designed to monitor CAN in such a way that it checks nodes availability by assigning designated pins which are programmed in the microcontroller to check its status while gaining control of CANs functionality. The faults in CAN gateway can arise due to software gateway failure besides protocol error of CAN like bit errors, stuff error, acknowledge error, and CRC error [18]. Any of these errors will be identified by the CAN gateway all the way through its CAN driver kernel coding. Additionally kernel will identify the Transmission and Reception errors of CAN which will be monitored by the software as well.

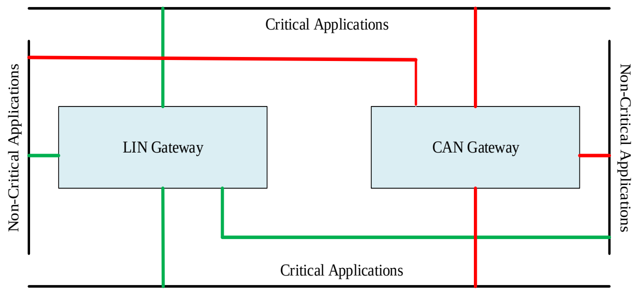


Fig. 8. Proposed Hybrid Architecture Model

In Fig. 8, the red line indicates the connectivity intended for the LIN gateway when the CAN get malfunction due to gateway failures or CAN protocol errors. Whereas the green lines indicate the connectivity for CAN gateway as soon as LIN goes wrong. Due to ease of software design, the CAN and LIN gateway together receives the critical and non-critical signals from the ECUs to facilitate their own functionality unless the fault is identified in either gateway. The internal local buffers deliberated in the software of CAN and LIN gateway will have the data of both critical and non-critical applications. During failure scenario, these data buffers will be updated on each occurrence of input from CAN or LIN nodes. Henceforth the data mismatch will not ensue as both the gateways have dedicated local buffers for both applications.

The performance of the CAN bus during the occurrence of wire fault can be examined as follows. Whenever an ECU transmits information on the bus, a bit error will be detected by the transceiver and goes to the dominant state. Consequently there is a voltage drop take place in resistance of the wire. In order to detect the fault in wire the resistance or voltage drop need to be examined at regular intervals. Therefore resistance of the wire is frequently measured through four wire kelvin resistance method. This method will reduce 20% of the measurement error in contrast with two wire resistance measurement method

[19]. To find out resistance of the wire, initially the voltage and current should be measured. Voltage can be determined by observing with different positions. Whereas, a low cost shunt resistor is exploited to measure the current on the CAN Bus. The CAN cable fault is located using Time Domain Reflectometry (TDR) method. Based on the reflections in the CAN cable (twisted pair), if the cable has normal impedance then there will not be any reflections in the cable and the same transmitted signal is absorbed at the other end. While there is any impedance change in the cable then the transmitted signal will be reflected back to the source. If the received signal has step increase in impedance, the same signal will be reflected back. Whereas, the received signal has decrease impedance the reflected signal might have decrease impedance [20]. The amplitude of the reflection may not only depend on the resistance change but also due to cable loss. The change in amplitude is considered as fault intensity and the difference in reflected signal time is intended as the length of the cable. The TDR approach is implemented by means of T connector, where one end of the connector will have known signal and opposite end will have the cable to be tested. The upper end of the connector must be associated to the device wherein magnitude and time of the signal is calculated [21].

To visualize the result, the CAN protocol error is created using the CAN stress hardware from Vector. The error frames generated by CAN and TX/RX error count in the software will get incremented. Once there is an increment in error count, the LIN gateway software will recognize that there is some failure in CAN. Gateway of CAN immediately checks its internal buffers to ensure which data was transmitted during the time of fault occurrence. Subsequently, LIN gateway will start transmitting the CAN failed message as well as also its own service messages in the network. The protocol conversion from CAN to LIN is written in software. When the CAN node resumes its functionality, the LIN node will send the frame that was transmitting at the moment and inform the CAN gateway to execute its utility.

The Fig. 9 shows the flow diagram of gateway module is to check for any protocol errors, then consequently to carry out the CAN to LIN message conversion for the fault tolerant gateway system. The CAN to LIN translation algorithm is presented in Algorithm 1.

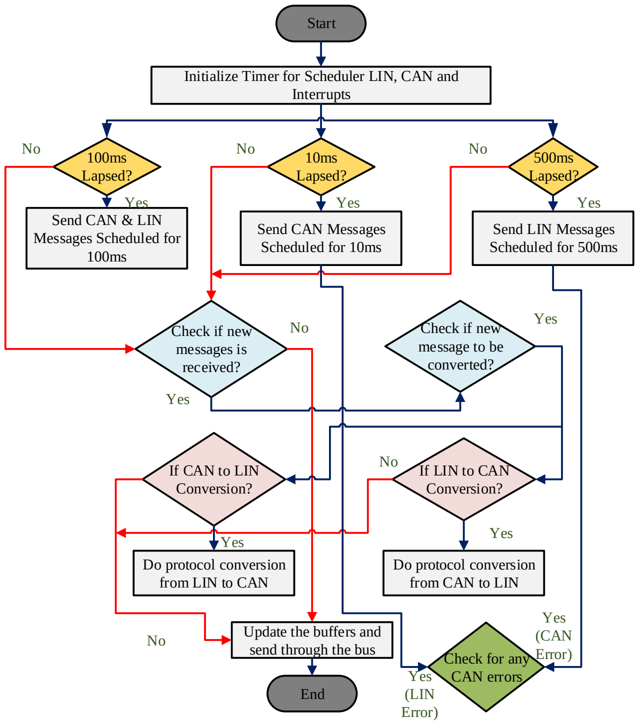

Start
Initialize Timer for Scheduler LIN, CAN and Interrupts
100ms Lapsed?
10ms Lapsed?
500ms Lapsed?
No
Yes
Send CAN & LIN Messages Scheduled for 100ms
Send CAN Messages Scheduled for 10ms
Send LIN Messages Scheduled for 500ms
Check if new messages is received?
Check if new message to be converted?
If CAN to LIN Conversion?
If LIN to CAN Conversion?
Do protocol conversion from LIN to CAN
Do protocol conversion from CAN to LIN
Update the buffers and send through the bus
Check for any CAN errors
Yes (CAN Error)
Yes (LIN Error)
End

Fig. 9. Flow diagram of gateway module

*A. Algorithm 1. CAN to LIN conversion*

1. CAN and LIN device driver software files will be called under the main program
2. Gateway function is implemented in the scheduler part (Scheduler is designed as a time slice in the software which will run in 10ms, 200ms, 500ms, 1000ms according to the clock frequency of the microprocessor. In the proposed gateway design the oscillator clock frequency is 25MHZ)
3. Initialize the CAN and LIN timers for 10ms, 100ms, 500ms in scheduler and also the interrupts for CAN and receive buffers to send and receive the message frames respectively
4. If the timers are getting elapsed as triggered in step 3, send the message frames that are required to be sent to the CAN buffers which will consign the message frames along with the data in the data bus of CAN. The periodicity of the message frames of CAN is done by Scheduler. The same procedure is followed for the LIN
5. Check the bus periodically for the messages received either on CAN or LIN as well
6. If the new message is available to move the new message that is updated in CAN and LIN buffers into the local software buffers
7. Check whether the CAN to LIN conversion has to be done. This decision is made based on the requirements whether any new message is received in CAN node. If yes, CAN bus will sent the data to the node connected to LIN network
8. If CAN to LIN conversion is required, do it in software (Mostly the data received that is moved to the local buffers are converted to the LIN data format and sent to the LIN network).

9. Update the buffers and send on the required network
10. Check for any protocol errors of CAN. This error checking is carried out by means of device driver software
11. If there are any errors, increment the error detection counter, error message is delivered by LIN or CAN either has error and send on bus
12. Step 11 is repeated until the CAN/LIN errors are amended.

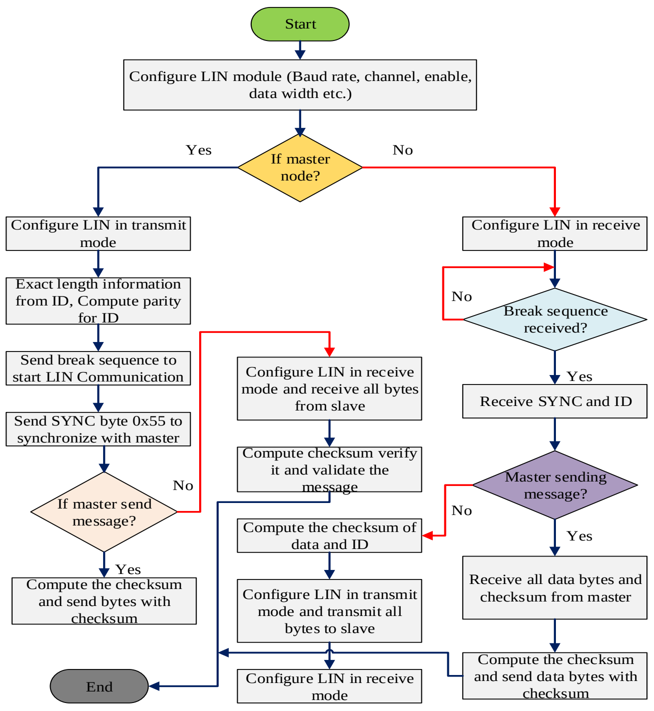

Start
Configure LIN module (Baud rate, channel, enable, data width etc.)
If master node?
Yes
No
Configure LIN in transmit mode
Exact length information from ID, Compute parity for ID
Send break sequence to start LIN Communication
Send SYNC byte 0x55 to synchronize with master
If master send message?
No
Yes
Compute the checksum and send bytes with checksum
End
Configure LIN in receive mode and receive all bytes from slave
Compute checksum verify it and validate the message
Compute the checksum of data and ID
Configure LIN in transmit mode and transmit all bytes to slave
Configure LIN in receive mode
Configure LIN in receive mode
No
Break sequence received?
Yes
Receive SYNC and ID
Master sending message?
No
Yes
Receive all data bytes and checksum from master
Compute the checksum and send data bytes with checksum

Fig. 10. Flow diagram of LIN module

Fig. 10 illustrates the flow to send and receive the LIN message in the LIN network. The data transfer operation of LIN is presented in Algorithm 2.

## *B. Algorithm 2. LIN Data Transfer Operation*

1. Configure the LIN module by giving the required values to the data registers of the microcontroller device. The values chosen are baud rate, channel selection, data size
2. LIN employs Master and slave architecture. Thus master node and slave nodes are to be selected
3. Check whether the node is master or not
4. If node is master node, then configure the LIN in transmit mode and extract the information required for length, Identifier and check the parity for the ID
6. Start the LIN communication by synchronizing with the Master and Check whether slave or the master needs to send the message, If master needs to send the message, compute the checksum as per the message protocol format and send the data bytes with checksum on the LIN network
8. If master is not sending the message, then configure the LIN in receive mode and obtain all data bytes from the slave nodes
9. After updating the received message in the local LIN software buffers, check for the checksum in the received message to verify it
10. If the node is not a master (check for Step 3), configure the LIN in slave mode, Make the synchronization and ID definition, and receive the message and put into the data buffers
11. Compute the checksum to verify whether the message received is correct or not.

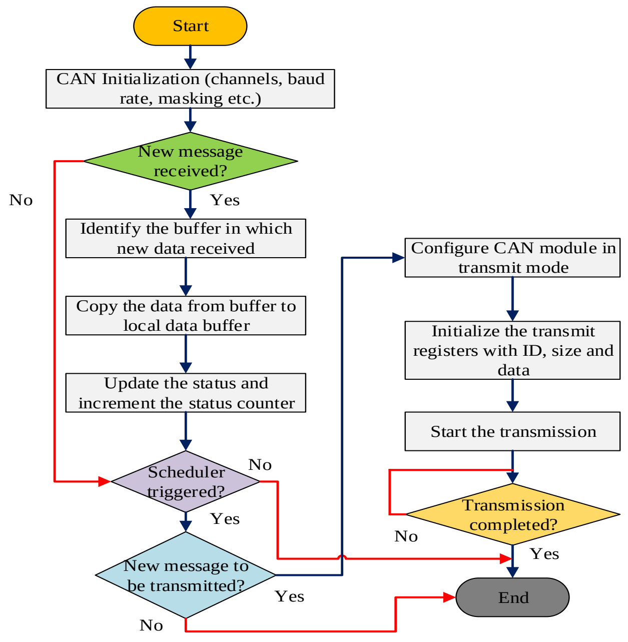

Start
CAN Initialization (channels, baud rate, masking etc.)
New message received?
No
Yes
Identify the buffer in which new data received
Copy the data from buffer to local data buffer
Update the status and increment the status counter
Scheduler triggered?
No
Yes
New message to be transmitted?
Yes
No
Configure CAN module in transmit mode
Initialize the transmit registers with ID, size and data
Start the transmission
Transmission completed?
No
Yes
End

Fig. 11. Flow diagram of CAN module

Fig. 11 shows the CAN module flow diagram to send and receive the data frames over the CAN network. The data exchange function of CAN is presented in Algorithm 3.

## *C. Algorithm 3. CAN Data Transfer Operation*

1. Initialize the CAN buffers for setting the baud rate, masking, and filtering of CAN ID registers
2. Check whether new CAN message has been received or not and identify in which CAN buffer in which the new message is received
4. When the buffer is updated with the message forward the message to the local CAN data buffer
5. Check whether the CAN message need to be transmitted or not, If any message needs to be transmitted, update the transmit buffer and send the data on CAN bus.
6. Check whether the scheduler is triggered. If scheduler has triggered and more message is to be transmitted then configure the CAN module in transmit mode.
7. Initialize the transmit registers with ID, size and data.
8. Start the transmission and check whether the transmission is carried out successfully.
10. Repeat the step 9 until the transmission completes.

## IV. EXPERIMENTATION

The time taken for transmission of messages from master to slave and vice versa is examined by linking CAN buses with Vector CANoe and effectively tested the system etiquette. The Vector CANoe monitors the bus for peak load, error frames, etc. The CANcaseXL hardware is used to link the CAN bus with Vector CANoe and the CAN interface is connected to the Freescale development board by tapping the CAN High and CAN low pins from the channel 2 of the board as shown in Fig. 12.

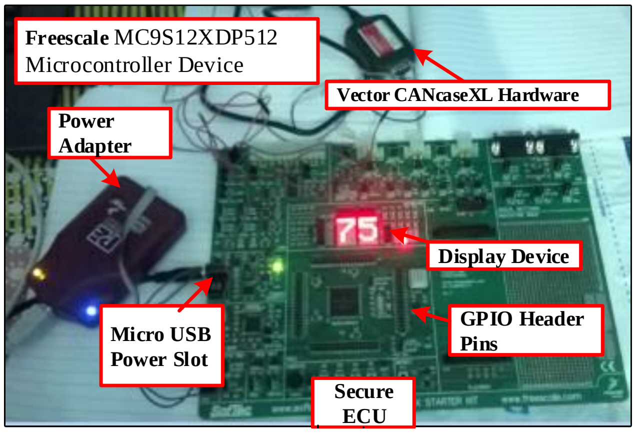
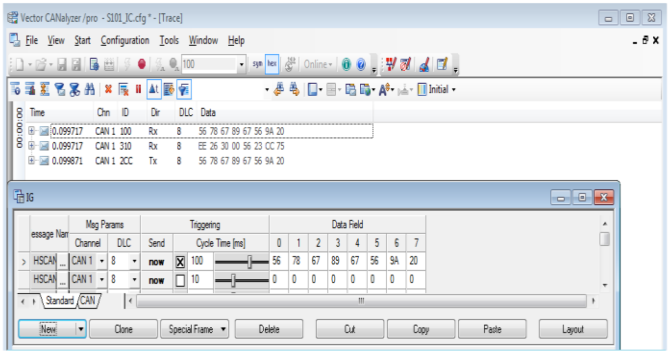


Fig. 12. Board connected with Vector CancaseXL for Bus monitoring

The fault tolerant and intelligent gateway is examined using MC9S12XDP512 Freescale device and outcome is accomplished. CANoe development environment is exploited for experimentation of automotive serial interfaces. Fig. 13 illustrates the CAN-CAN (C2C) gateway i.e. The CAN ID 2CC is simulated from IG block of Vector CANalyzer and sends to CAN channel 1. The same message is received by another channel and transmitted back in the CAN ID 100, which can be discerned in the trace window of Vector CANalyzer. The Fig. 14 depicts the decoding of the CAN message IDs 0x2CC which is shown in LECROY wave surfer. The waveform captured ensures the decoded CAN frame by means of start of frame, data bytes, CRC in line for the end of the frame with bit stuffing. Fig. 15 illustrates the LIN transmission with message ID 0x01. The decoding of LIN frame demonstrates the resynchronization, ID number, and bytes of data. Fig. 16 represent that the CAN gateway failure achieved by fusing both the CAN lines (CAN High and CAN Low) thus error frames can be generated over the bus. Moreover, protocol failure of C2C gateway failure arises as there is zilch message acquired in CAN bus during broadcasting. Fig. 17 portrays the error frame on CAN bus. Fig. 18 shows the simulation of the true time simulator (debugger) of the Freescale code warrior IDE. Notice the data window the transmit error count will get incremented when the CAN lines are fused (as outlined in the device driver software of CAN). At this instance, LIN gateway will be accomplished to retrieve and transmit the CAN messages which stand in error state. The gateway of CAN bus in error state and LIN bus transmitting the CAN data frames (which will be decoded and framed as per CAN data in form of LIN message) has experimented on LIN bus in the network as shown in Fig. 19. As a result, momentary failure of gateway and deterrence of bus is confined. Thus the proposed gateway will be constructive in terms of hybrid architecture where the CAN and LIN networks have well-built communication towards each other with reduced cost and high performance.

## V. CONCLUSION

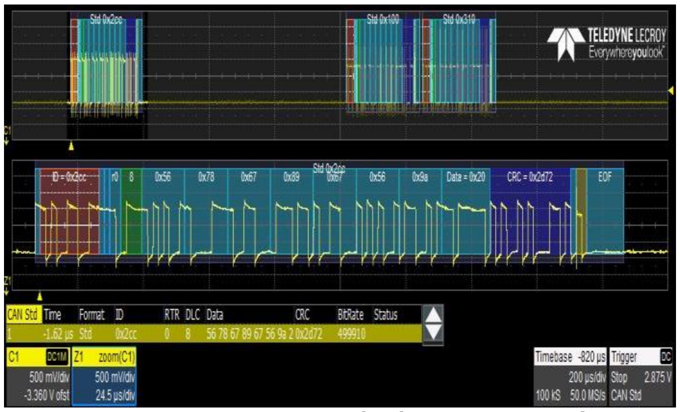
This paper produces the overview of CAN and LIN protocol concepts which are demonstrated with the help of Vector CANoe tool and Freescale hardware. During the early stage of the development process, the impact of this gateway based approach helps in diagnosing the intra-vehicle networks thereby the significant amount of design time has been saved for real-time applications. The ECU modules are positioned according to the vehicle gateway standards beneficial to avoid data loss due to increased wire length and complexity. The gateway brought into this article will assist CAN and LIN networks. For the experimental analysis as illustrated in previous sections rigid effort was effectuated to improve the gateway performance and resolve the faults that crop up while sharing the data among the protocol. As a part of future work the approaches outlined in this article can be extended for other intra-vehicular networks like FlexRay and MOST. In addition to this, the CAN FD protocol can be utilized for higher bandwidth applications with payload of up to 64 byte, which is more suitable for complex sensor systems and bid like benefits.


Vector CANalyzer /pro - S101_IC.cfg * - [Trace]
File View Start Configuration Tools Window Help
Online
Initial
IG
Standard
CAN
New
Clone
Special Frame
Delete
Cut
Copy
Paste
Layout

| Time | Chn | ID | Dir | DLC | Data |
|---|---|---|---|---|---|
| 0.099717 | CAN 1 | 100 | Rx | 8 | 56 78 67 89 67 56 9A 20 |
| 0.099717 | CAN 1 | 310 | Rx | 8 | EE 26 30 00 56 23 CC 75 |
| 0.099871 | CAN 1 | 2CC | Tx | 8 | 56 78 67 89 67 56 9A 20 |

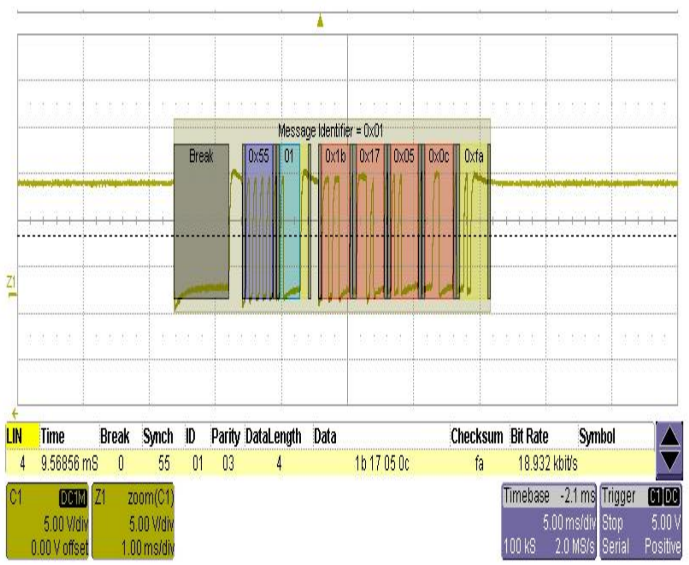
| essage Nam | Msg Params | | Triggering | | | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Channel | DLC | Send | | Cycle Time [ms] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| HSCAN | CAN 1 | 8 | now | ☒ | 100 | 56 | 78 | 67 | 89 | 67 | 56 | 9A | 20 |
| HSCAN | CAN 1 | 8 | now | ☐ | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig. 13. CAN messages Transmission and Reception (Refer to CAN ID)



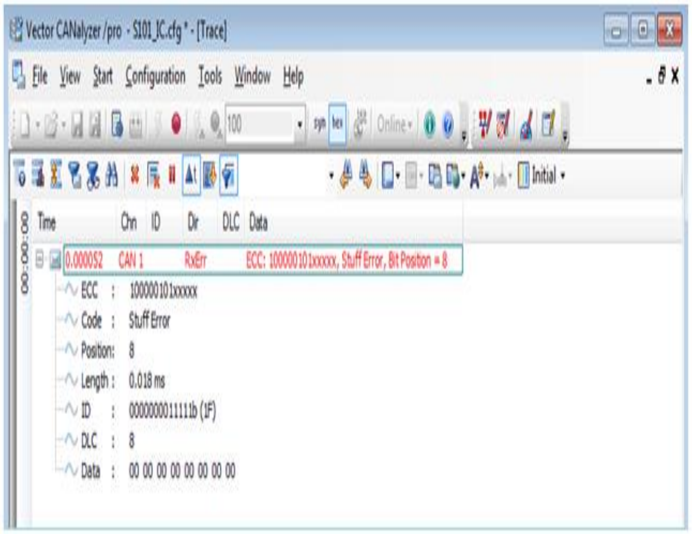
| CAN Std | Time | Format | ID | RTR | DLC | Data | CRC | BitRate | Status |
|---|---|---|---|---|---|---|---|---|---|
| 1 | -1.62 µs | Std | 0x2cc | 0 | 8 | 56 78 67 89 67 56 9a 2 | 0x2d72 | 499910 | |

Fig. 14. CAN Message Frame Captured with LECROY Bus Analyzer



| LIN | Time | Break | Synch | ID | Parity | DataLength | Data | Checksum | Bit Rate | Symbol |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 9.56856 mS | 0 | 55 | 01 | 03 | 4 | 1b 17 05 0c | fa | 18.932 kbit/s | |

Fig. 15. LIN Frame Captured using LECROY Scope


Vector CANalyzer /pro - S101_IC.cfg * - [Trace]

File View Start Configuration Tools Window Help

| Time | Chn | ID | Dir | DLC | Data |
|---|---|---|---|---|---|
| 0.000052 | CAN 1 | | RxErr | | ECC: 100000101xxxxx, Stuff Error, Bit Position = 8 |

```
ECC      :  100000101xxxxx
Code     :  Stuff Error
Position:   8
Length  :   0.018 ms
ID       :  000000001111b (1F)
DLC      :  8
Data     :  00 00 00 00 00 00 00 00
```

Fig. 16. CAN Error Frames Captured using Vector CANalyzer

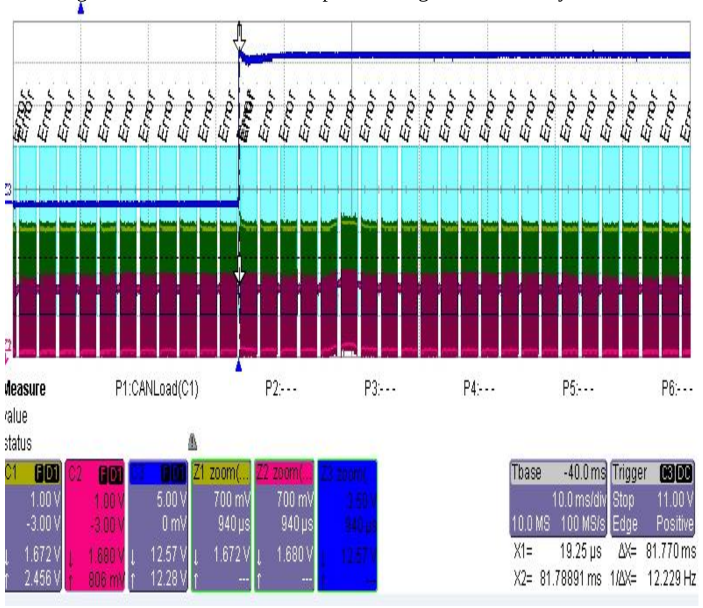
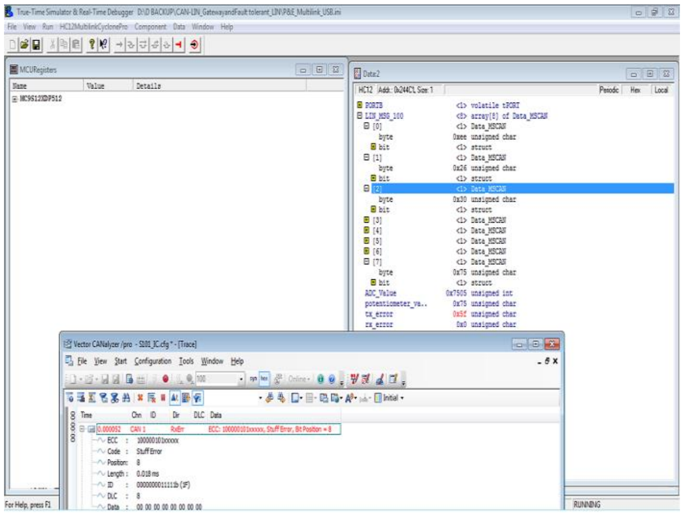


Fig. 17. CAN Bus Error Frames Captured using LECROY Scope


True-Time Simulator & Real-Time Debugger D:\D BACKUP\CAN-LIN_GatewayandFault tolerant_LIN\P&E_Multilink_USB.ini

File View Run HC12MultilinkCyclonePro Component Data Window Help

MCURegisters

| Name | Value | Details |
|---|---|---|
| MC9S12XDP512 | | |

Data2 — HC12 Addr: 0x244C1, Size: 1 — Periodic Hex Local

```
PORTB                  <1> volatile tPORT
LIN_MSG_100            <8> array[8] of Data_MSCAN
  [0]                  <1> Data_MSCAN
    byte              0xee unsigned char
    bit                <1> struct
  [1]                  <1> Data_MSCAN
    byte              0x26 unsigned char
    bit                <1> struct
  [2]                  <1> Data_MSCAN
    byte              0x30 unsigned char
    bit                <1> struct
  [3]                  <1> Data_MSCAN
  [4]                  <1> Data_MSCAN
  [5]                  <1> Data_MSCAN
  [6]                  <1> Data_MSCAN
  [7]                  <1> Data_MSCAN
    byte              0x75 unsigned char
    bit                <1> struct
ADC_Value           0x7505 unsigned int
potentiometer_va..    0x75 unsigned char
tx_error              0x5f unsigned char
rx_error               0x0 unsigned char
```

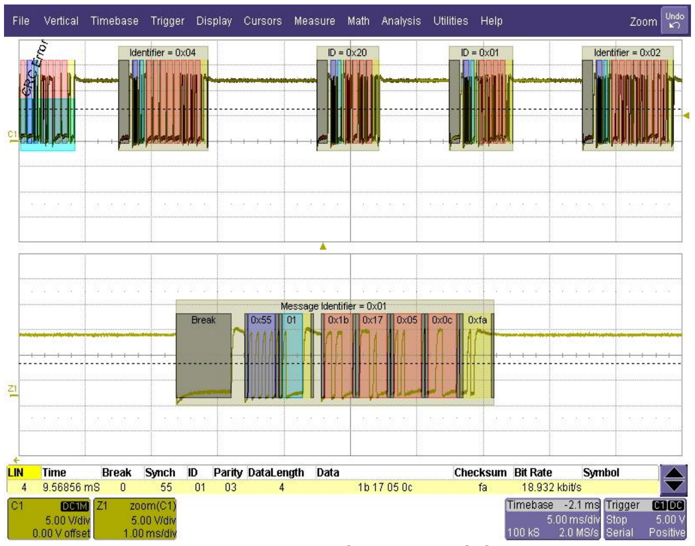
Vector CANalyzer /pro - S101_IC.cfg * - [Trace]

| Time | Chn | ID | Dir | DLC | Data |
|---|---|---|---|---|---|
| 0.000052 | CAN 1 | | RxErr | | ECC: 100000101xxxxx, Stuff Error, Bit Position = 8 |

```
ECC      :  100000101xxxxx
Code     :  Stuff Error
Position:   8
Length  :   0.018 ms
ID       :  000000001111b (1F)
DLC      :  8
Data     :  00 00 00 00 00 00 00 00
```

For Help, press F1 — RUNNING

Fig. 18. LIN Message Transmission when CAN Node has Error Frames using Real-Time Debugger



| LIN | Time | Break | Synch | ID | Parity | DataLength | Data | Checksum | Bit Rate | Symbol |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 9.56856 mS | 0 | 55 | 01 | 03 | 4 | 1b 17 05 0c | fa | 18.932 kbit/s | |

Fig. 19. LIN Message Transmission when CAN Node has Error Frames using LECROY Scope

## REFERENCES

[1] Jihas Khan, "Design of a High-Efficiency In-Vehicle Network with a Single ECU for a Network (SEN)," *SAE Technical Paper*, 2014-01-0246, May 2014.

[2] Azam Ramazani, Tahereh Mohammadi, Wathiq Mansoor Hamed Vahdat-Nejad, "A survey on context-aware vehicular network applications," *Vehicular Communications, Elsevier Inc.*, vol. 3, pp. 43-57, January 2016.

[3] Carlos T. Calafatey, Juan-Carlos Canoy, Nasreddine Lagraa, Pietro Manzoni Chaker Abdelaziz Kerrache, "Trust management for Vehicular Networks: An Adversary-Oriented Overview," *IEEE Access*, no. 99, 2016.

[4] Nizar Alsharif, "iCAR-II: Infrastructure-based Connectivity Aware Routing in Vehicular Networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 4231-4244, May 2017.

[5] Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi, Liam Kilmartin Shane Tuohy, "Intra-Vehicle Networks: A Review," IEEE Transactions On Intelligent Transportation Systems, vol. 16, no. 2, pp.534-545, April 2015.

[6] J.H. Kim, T.Y. Moon, S.H. Hwang, K.H. Kwon, J.W. Jeon S.H. Seo, "Gateway Framework for In-Vehicle Networks," Science Direct, IFAC Proceedings, vol. 41, no. 2, pp. 12081-12086, 2008.

[7] Jin Ho Kim et al., "Gateway Framework for In-Vehicle Networks Based on CAN, FlexRay, and Ethernet," IEEE Transactions on Vehicular Technology, vol. 64, no. 10, pp. 4472 - 4486, 2015.

[8] Heiko Doerr and Ingo Stuermer, "Managing an ISO 26262 Safety Case: A Software System Perspective," SAE - Technical Paper 2016-01-0137, June 2016.

[9] Abdolreza Fallahi - Navistar Inc., David Zhang - Navistar Inc., Kumar Kuppam - Navistar Inc. Saleh Mirheidari - Navistar Inc., "AUTOSAR Model-Based Software Component Integration of Supplier Software," SAE Int. J. Commer. Veh. vol.8, no.2 , pp. 544 - 548, September 2015.

[10] M. Farsi, K. Ratcliff, and M. Barbosa, "An Overview of Controller Area Network," IET Journals & Magazines, vol. 10, no. 3, pp. 113-120, June 1999.

[11] Dominique Paret, Multiplexed Networks for Embedded Systems CAN, LIN, Flexray, Safe-by-Wire. Chichester, England: John Wiley & Sons Ltd, 2007.

[12] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee, "A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN," IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 2, pp. 993-1006, April 2015.

[13] Yong Lei, Haibo Xie, Yong Yuan, and Qing Chang, "Fault Location for the Intermittent Connection Problems on CAN Networks," IEEE Transactions on Industrial Electronics, vol. 62, no. 11, pp. 7203-7213, November 2015.

[14] M. Ruff, "Evolution of local interconnect network (LIN) solutions,", vol. 5, pp. 3382-3389, 2003

[15] Seung-Han Kim et al., "A gateway system for an automotive system: LIN, CAN, and FlexRay," in 6th IEEE International Conference on Industrial Informatics, Daejeon, Korea, pp. 967-972, 2008.

[16] Tae-Yoon Moon, Suk-Hyun Seo, Jin-Ho Kim, Sung-Ho Hwang, and Jae Wook Jeon, "Gateway system with diagnostic function for LIN, CAN and FlexRay," in Control, Automation, and Systems, 2007.ICCAS '07. International Conference on, Seoul, Korea, pp. 2844 – 2849, 2007.

[17] Weiying Zeng, Mohammed A. S. Khalid, and Sazzadur Chowdhury, "In-Vehicle Networks Outlook: Achievements and Challenges," IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 1552-1571, Third Quarter 2016.

[18] Jin-Ho Kim, Sung-Ho Hwang, Key Ho Kwon, and Jae Wook Jeon Suk-Hyun Seo, "A reliable gateway for in-vehicle networks based on LIN, CAN, and FlexRay," ACM Transactions on Embedded Computing Systems (TECS), vol. 11, no. 1, March 2012.

[19] Chung Y.C, Lo C., Pendalaya P, and Furse C, "A critical comparison of reflectometry methods for location of wiring faults," *Smart Structures and Systems*, vol. 2, no. 1, pp. 25-46, January 2006.

[20] Furse C. and Gunther J Smith P, "Analysis of spread spectrum time domain reflectometry for wire fault location," *IEEE Sensors*, vol. 5, no. 6, pp. 1469-1478, 2005.

[21] Auzanneau F, Peres F. and Tchangani A Hassen B, "Diagnosis sensor fusion for wire fault location in CAN bus systems," in *SENSORS, 2013 IEEE*, Baltimore, MD, USA, 2013.

[22] A.Bindu, M.Carolin Mabel, and C.Bharatiraja, "A Real-Time Energy Management Approach And Its Power Converter For PV Powered Plug-In Electric Vehicle", in ***Journal of Electrical Engineering***. vol. 17, no.2, 2017, pp.241-247, 2017