International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 6 (2017) pp. 921-927

A Design of Fail-safe Gateway-embedded System for In-vehicle Networks

Sukhyun Seo, Junsu Kim, *Su Min Kim

Department of Electronics Engineering, Korea Polytechnic University, 15073 Siheung, Republic of Korea.
**Corresponding author*

Abstract
This paper presents a reliable gateway-embedded system for in-vehicle networks based on LIN, CAN, and FlexRay. We propose fault-tolerant hardware architecture and software diagnostics to improve the reliability of the gateway system. The proposed system is based on the extended asymmetric controller scheme and implemented by using two micro-controllers and networks. One microcontroller performs the main function of achieving a gateway, often called the main microcontroller. The second microcontroller checks and monitors the health of the main microcontroller. Several experiments are employed to evaluate the performance of the proposed system.

Keywords: fail-safe system, hard real-time system, gateway, in-vehicle network, fault-tolerance, reliability

INTRODUCTION

Current vehicles include several different networks using a local interconnect network (LIN), controller area network (CAN), and FlexRay protocols [1]. Gateway systems are becoming more important and more integrating with advanced features, because they enable seamless communication between heterogeneous networks and combine with similar control units, as depicted in Figure 1. In some cases, the gateway plays a role of a domain control unit (DCU) that manages all nodes within a cluster, the role of a gateway is not complex, but the gateway as domain controller is a potential supervisor to manage all of the operation of other connected ECUs and networks [2, 3]. For the reason, the gateway can be regarded as a hard real-time system. Reliability and fault-tolerance are the most important requirements of real-time systems to guarantee fail-safe automotive control systems [4].

Therefore, we propose a fail-safe gateway-embedded system that can maintain reliable in-vehicle networks (IVNs) consisting of LIN, CAN, and FlexRay sub-networks. The proposed system provides fail-safe operation using the proposed fault-tolerant hardware architecture, real-time watchdog, self-diagnosis, and hand-over mechanism.

Starting from this perspective the structure of this article is as follows. Section 2 introduces fault-tolerant controller strategies in the automotive control system. Section 3 proposes an extended asymmetric architecture based on asymmetric and distributed approaches. Section 4 defines faults to be detected and proposes our fail-safe gateway-embedded system based on the proposed extended asymmetric architecture. Section 5 describes the experimental evaluation and results. Finally, we summarize and conclude this paper in Section 6.

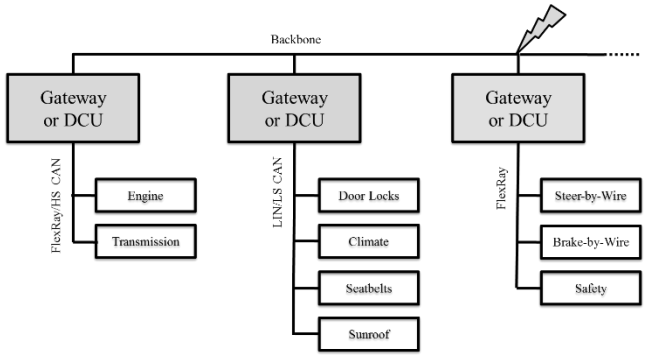
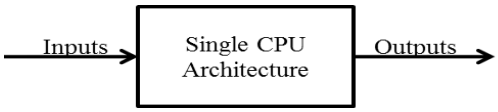


Figure 1. General network configuration of in-vehicle networks for future vehicles.

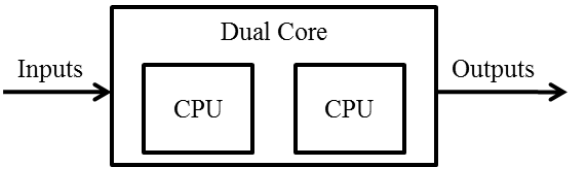
OVERVIEW OF FAULT-TOLERANT SYSTEMS

Fault-tolerance is a crucial issue in the design of hard real-time systems. Comprehensive surveys on fault-tolerant real-time systems can be found in [5]-[8]. In this section, we review the basic concept of the various fault-tolerant architectures used in the automotive control systems. Figure 2 shows the five types of fail-safe controller architecture: (a) single controller, (b) multiple-core controller, (c) symmetric controller, (d) distributed controller, and (e) asymmetric controller. A number of redundant or backup devices are required to achieve fault-tolerant systems. However, these systems consider the negative influences implied with fault-tolerant architecture. In this paper, we mainly cover an asymmetric-controller and distributed controller approach. Other approaches are introduced briefly with the view to develop a gateway-embedded system for IVNs.

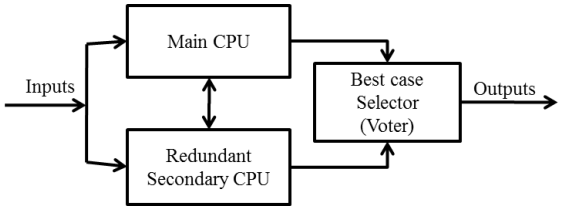
A single-controller failsafe strategy consists of a single controller executing the target application software and self-checking diagnostics, along with a simple watchdog circuit. Although a single controller fail-safe strategy spends significant overhead and resources on self-checking diagnostics, there is no solution for hardware failure.



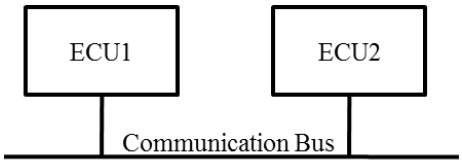
(a) Single controller strategy



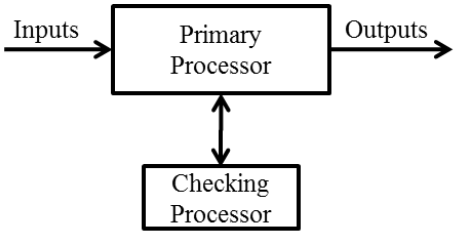
(b) Multi-core controller strategy



(c) Symmetric controller strategy



(d) Distributed controller strategy



(e) Asymmetric controller strategy

Figure 2. Approaches to a fail-safe controller researched over the years consider cost, performance, and level of technology.

A multi-core controller strategy has a duplicate CPU. The multiple CPUs receive the same data and control inputs, and the outputs of the CPUs are compared to detect discrepancies. A multiple-core controller strategy has the problem of ensuring the reliability of a highly integrated circuit. In particular, internal hardware failures are difficult and complex to handle in software, and software architecture design cannot be flexible due to its strong dependency on hardware architecture.

The symmetric controller approach uses two identical controllers to cross-check controller operation, but there is a cost issue. Synchronization between the controllers is another difficult issue. Triple modular redundant (TMR) architecture can be included in this strategy. Three identical controllers, in the TMR configuration, execute the same code and a majority vote of the outputs masks any possible single CPU fault. Of course, this approach has the same crucial issues to overcome, as in the symmetric controller approach.

The distributed controller strategy is based on a networked control scheme. Although this strategy can use both independent redundant algorithm execution and independent primary processor state-of-health checks, this strategy can be non-deterministic due to a number of communication dependent factors, including the availability of network bandwidth, fault response times, available controller throughput, synchronization issues, and controller sourcing decisions.

The asymmetric controller approach uses an additional simple and low-cost secondary checking processor. This strategy provides a high level of detection and correction at low cost compared to other approaches (except the single controller approach). The secondary processor periodically requests diagnostic checks by the primary processor to help detect hardware/software failures. The performance of this strategy depends on the throughput of the checking processor, and on the communication media between the primary and a checking processor. However, the checking processor can miss errors (i.e. transient faults) that occur transiently due to the periodic request. That is, the period needs to be optimized by simulation or experiment. In addition, the performance of the primary processor can decrease slightly, since the primary processor responds periodically to the checking processor's requests. However, the design of a fault-tolerant system can be flexible, because this approach can be adjusted depending on the level of system safety requirements.

In this paper, an extended asymmetric architecture combined with a distributed controller and asymmetric strategy is proposed in order to implement a fail-safe gateway embedded system. We utilize the asymmetric strategy based hardware architecture and complement the drawbacks of the strategy using the distributed controller strategy. The additional improvement of the architecture is achieved by fail-safe mechanism using alternative processes, shared resources, and simple software diagnostics.

EXTENDED ASYMMETRIC ARCHITECTURE

As described in the previous section, state-of-the-art architecture for fail-safe systems considers the relative merits that imply unresolved issues according to cost, performance, and implementation. In this paper, we propose an extended asymmetric architecture that combines the two strategies between asymmetric and distributed controllers. In this section, especially, we explain the extended asymmetric approach that may be applied to the gateway within heterogeneous networks.

The extended asymmetric architecture, like the symmetric architecture, consists of two microcontrollers as depicted in Figure 3. The main microcontroller performs primary functions to control the networks, and the checker is an additional simple and low-cost secondary checking processor, such as a watchdog.

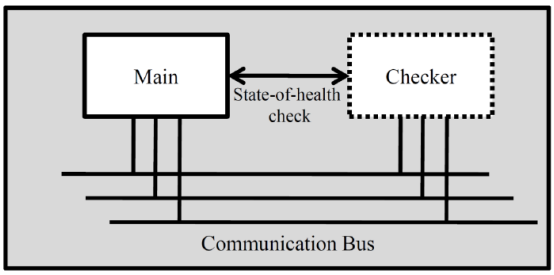


Figure 3. The concept of extended asymmetric architecture.

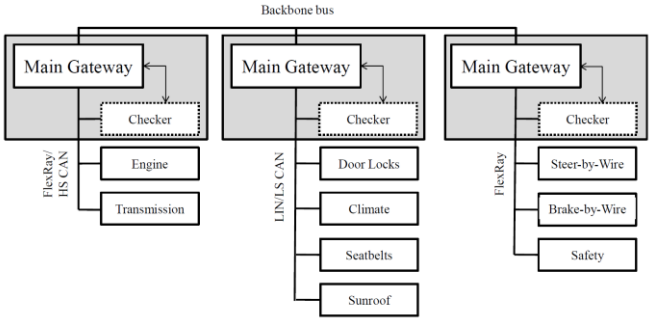


Figure 4. Extended asymmetric architecture based gateway within IVN.

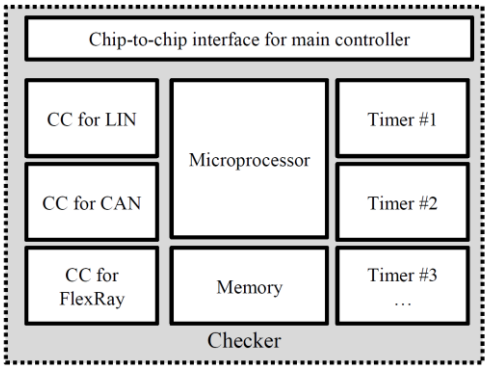


Figure 5. The composition of checker controller.

The checker monitors the state-of-health of the main controller via a chip-to-chip serial interface (e.g. serial peripheral interface) periodically. The required checker specification is determined by the level of safety requirements.

The checker is required to act as watchdog and to have more sophisticated backup functions in the gateway design. Therefore, the asymmetric approach is limited to the gateway design. As explained in the previous section, the weaknesses of the asymmetric architecture are the probability of missing transient faults and the burden of overhead for diagnostic response.

Thus, the checker is shared with communication buses and listens to all messages via the buses. The main controller and checker are regarded as independent nodes within the same cluster in the distributed control system. Figure 4 shows the position of the main controller and checker in the network. The gateway consists of a main controller and checker. Although the main controller and checker is one system that performs information exchange with different networks, two microcontrollers are distinct in the network. As the checker listens to all of the messages on the bus, the main controller overhead required for diagnostics and message monitoring decreases. That is, the incorrect operation of the main controller can be monitored by listening to the outgoing data from the main controller without additional diagnostic requests. In some cases, the checker may operate independently of the main gateway, when the main controller is out of control. The checker coverage replaced by the main controller depends on the hardware capability. For example, if the checker is identical to the main controller, the checker is fully replaced by the main controller. Otherwise, if the checker operates with a lower performance than the main controller does, the checker may optionally process only highest priority tasks.

The checker uses all of information received from the main controller for transient faults. For example, if the diagnostics information received from the main controller via the chip-to-chip interface is mismatched to outgoing data to the bus logically, the checker can temporarily transmit the corrected information to the bus. The checker may consist of microprocessor, multiple watchdog timers, and communication controllers, as depicted in Figure 5, to provide these operations. Multiple communication controllers (CC) are required to connect to the network bus; multiple timers are used to monitor the main controller periodically and check the messages periodicity. In some cases, a complex programmable logic device (CPLD) can be added to the system to interface shared or redundant peripheral devices (resources) with the checker, instead of the main controller.

FAIL-SAFE SYSTEM

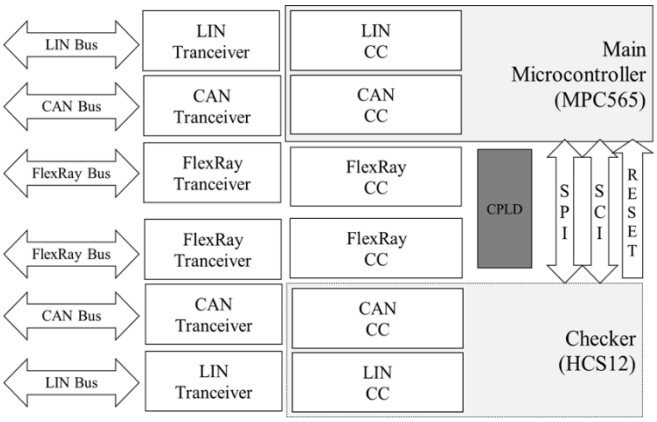
A. Fault Definition and Coverage

In this paper, we say a system is fail-safe, if the system guarantees fail-safe execution at the hardware fault and software error issued. In addition, it is a fault-tolerant system if the system has specified operations against faults. A fault is the origin of failure. The faults occur in hardware modules or as software bugs. The hardware modules are system physical components, such as microcontroller, memory, communication controller, transceiver IC, and power management IC. In this paper, we regard hardware faults as power loss and abnormal operation of system hardware components. In addition, software bugs are illogical operations, such as infinite loop, deadlock, and inconsistent task sequence. As the proposed gateway-embedded system is based on the extended asymmetric architecture, fail-safe operation is supported against these faults. The following

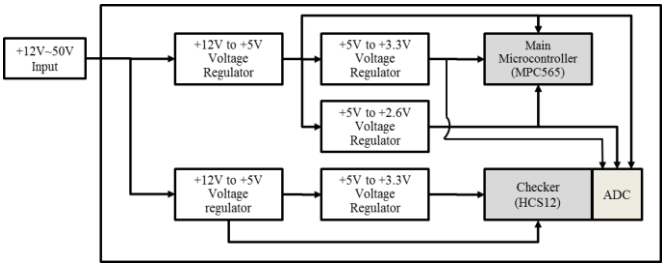
section describes further specified fault-tolerant operations.

B. Fault-tolerant Hardware architecture

The proposed fault-tolerant gateway-embedded system consists of a main microcontroller, a checker, and a CPLD, as depicted in Figure 6.



(a) Block diagram of communication module in the gateway-embedded system.

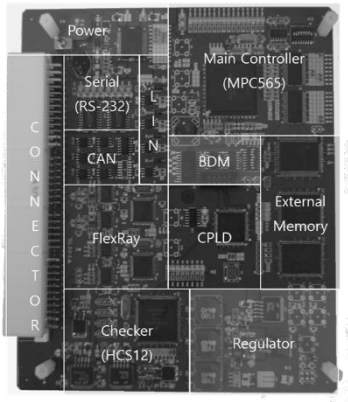


(b) Block diagram of power supply and monitor module in the gateway-embedded system.

Figure 6. Main microcontroller is monitored by the checker.

Figure 6-(a) shows a block diagram of the fault-tolerant communication module for the gateway-embedded system. The main microcontroller was designed using an MPC565 microcontroller, one of the MPC5xx families (Freescale Semiconductor, Inc.). The MPC565 includes serial peripheral interface (SPI), serial communication interface (SCI), and CAN modules. The LIN protocol was implemented using SCI interface modules. FlexRay was externally implemented using the MFR4310 Freescale FlexRay CC. The checker used a HCS12 (16-bit based microcontroller), a microcontroller from Freescale Semiconductor, Inc. The HCS12 includes SPI, SCI, and CAN communication modules. The LIN protocol was implemented using SCI modules. The LIN, CAN and FlexRay transceivers were placed externally as a physical layer. The three transceivers were connected to each physical network bus. CPLD is used to enable/disable CCs and transceivers for the main microcontroller and checker. For example, when the checker receives control directly from the main microcontroller, the CPLD disables CCs and transceivers for the main microcontroller and enables CCs and the transceiver for the checker.

In addition, the supplied voltages to the main microcontroller were monitored via the analog to digital converter (ADC) included in the checker, as depicted in Figure 6-(b). If the voltages depart from the desired range, the checker recognizes the failure of the main microcontroller. Figure 7 shows the developed gateway-embedded system and its specification.



(a) Developed fault-tolerant gateway hardware

MCU	Main micro-controller : MPC565 Checker : HCS12	**LIN Bus Clock**	~20 Kbit/s (1 master, 1 slave)
Operating Clock	MPC565 : 56 MHz HCS12 : 16 MHz	**CAN Bus Clock**	~1 Mbit/s (3 channels)
OS	OSEK OS	**FlexRay Bus Clock**	~10 Mbit/s (2 channels)
Middleware	OSEK COM, OSEK NM	**SCI/SPI Clock**	~4 Mbit/s
Internal Flash EEPROM	MPC565 : 1 Mbyte HCS12 : 256 Kbyte	**External RAM**	512 Kbyte
CLD	XCR5128XL-VQ100	**Power Input Range**	12 ~ 50V

(b) Specification of gateway embedded system.

Figure 7. Developed gateway-embedded system

C. Fail-safe operation for gateway-embedded system

In this study, we consider three types of state: normal state, transient fault state, and permanent fault state, as depicted in Figure 8. Normal state infers that the main microcontroller operates correctly. Fault state infers that the main microcontroller has a fault and executes the process incorrectly. In the transient fault state, the main microcontroller can be recovered quickly and the fault ignored, because the fault is temporary. The states are determined by the checker using diagnostics information and listening to the messages.

In the permanent fault state, the main microcontroller cannot be recovered within the predefined time, because the fault exists for

Table 1. Fault detection

State	Condition
Transient Fault	- Mismatch or inconsistency detection between the information received from main controller via SPI and messages on the bus. - single task deadline violation - single message deadline violation
Permanent Fault	- Power loss detection of the main controller - No outgoing message from main controller within the predefined time - bus passive or bus off detection - dead-lock or infinite loop detection - multiple task deadline violation - multiple message deadline violation

a long time. In this study, we do not fully consider the methods of classifying and detecting transient and permanent faults. The developed gateway-embedded system is handled by the physical health of the main microcontroller, or the execution sequence of the target-dependent application program. Thus, the concepts of fault and failure vary, depending on the target application requirements. Table 1 describes conditions to detect the classified faults under different conditions.

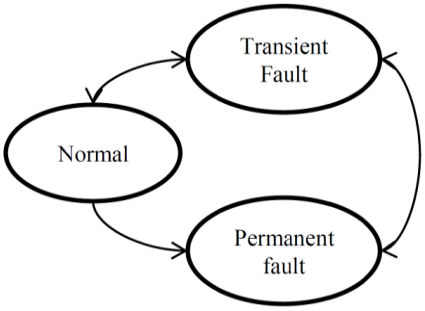


Figure 8. State diagram of the main controller.

Figure 9. Example diagnostic information format.

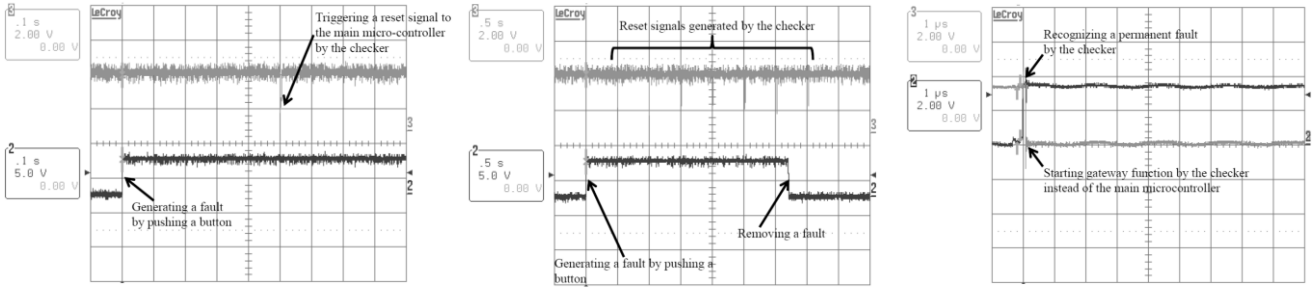
In the normal state, the checker receives diagnostic information from the main controller and only listens to all the messages on the buses, because the main controller operates correctly. The checker does not access the connected buses. In general, embedded software consists of sequential procedures that are divided into several modules. The execution sequences of each module inform the checker to detect software failures; it then monitors if the program sequence is correct. If the sequence is incorrect, the checker recognizes the state of the main microcontroller as a failure. Thus, diagnostic information includes message type, operation mode, the identification of activated tasks, and timestamp. Figure 9 shows an example of the diagnostic information format. The message type classifies diagnostic information. For example, type #1 represents LIN-to-CAN gateway process or type #2 denotes the FlexRay-to-LIN gateway process. The mode represents the operational mode of the main microcontroller, such as initialization, configuration, normal operation, and emergency mode.

If the main microcontroller operates in the transient fault state during a predefined time, the checker triggers a reset signal to the main microcontroller. If the main microcontroller returns to the normal state before the predefined time, the checker ignores the fault and monitors the health of the main microcontroller continuously.

If the main microcontroller fails permanently, CPLD disables the peripheral devices connected to the main controller, and then enables the devices connected to the checker. The checker can access the external resources, such as transceivers and CCs, and temporarily performs the operation of the main microcontroller. The following functions are performed by the checker.

- **F1**. Monitoring the supplying voltages to the main microcontroller.
- **F2**. Monitoring the sequence of processing tasks of the main microcontroller.
- **F3**. Monitoring the states of the communication controllers
- **F4**. Performing the operation of the main microcontroller temporarily when the main microcontroller has failed.
- **F5**. Transmitting warning messages to the connected nodes when the main microcontroller has failed.
- **F6**. Helping the main microcontroller recover its normal operation.

Function 1 is achieved by the ADC included in the checker, such as depicted in Figure 6-(b). The ADC module samples the supply power voltage level every predefined period (e.g. 1 millisecond). The sampling rate is related to the worst-case detection time. When power loss is detected by the checker, the checker recognizes the permanent fault state of the main controller and performs the gateway function temporarily in the emergency mode. The checker can know the power loss within the sampling rate period. Functions 2 and 3 are performed using the diagnostic information that includes processing task sequence and the state-of-health of the main controller. Diagnostic information is requested periodically by the checker. The period is related critically to the fault-tolerance capability. As the period becomes shorter, the fault-tolerance can be



(a) (b) (c)

Figure 10. Experimental result: (a) fault detection (b) recovery issued by a transient fault (c) Checker hand-over issued by a permanent fault

increased. However, the overhead for processing diagnostics increases. Functions 4 and 5 are realized by the switching mechanism of the CPLD. The checker controls the connected buses in the emergency mode, because the CPLD disables or enables the external devices depending on the state of the main controller. In the normal state, the checker cannot transmit any messages on the bus, but the checker is able to send information on the bus in the emergency mode after the CPLD enables the external devices connected to the checker. In function 6, the checker helps the main controller recover its normal state by stimulating the main controller, such as triggering a reset signal.

The monitoring functions are performed via SPI and SCI modules. The main microcontroller transmits the state of each module to the checker periodically via SPI. The checker can also request the state of the main microcontroller. SCI is used when the main microcontroller needs to write and store important data to the flash memory or random access memory (RAM) within the checker. In the proposed architecture, a parallel interface is avoided due to electronic-magnetic interference that can occur in a vehicle's harsh environment.

If the main microcontroller enters the failure state category, the checker performs the following processes.

Step 1. Recognizes the failure of the main microcontroller.
Step 2. Transmits a warning message to the connected nodes.
Step 3. Triggers a reset signal to the main microcontroller.
Step 4. Temporarily performs the operations of the main micro controller based on the priority of the messages.
Step 5. If the main microcontroller is recovered, the checker completes the processing tasks and then stops performing the operations of the main microcontroller.

As the checker performs the alternative operation, instead of the main microcontroller, the checker performs the constrained main operation selectively, based on the priority of the received messages, because the checker's performance is not identical to that of the main controller and cannot handle all the operations normally performed by the main microcontroller.

D. Functional Mechanism for Gateway

In this paper, we cover the gateway function simply. We previously published the work in [9]. The published gateway routing functions are used as well in the study. The gateway performance of the developed gateway embedded system does not evaluate in this paper again.

The gateway-embedded system function primarily provides the translation process between LIN, CAN, and FlexRay. Translation is done via message reformatting; i.e., the received message is converted from the source protocol format to the destination protocol.

The gateway-embedded system includes message receiving (transmitting) interrupt service routines (ISRs) for each protocol and message-processing tasks for each message. The gateway sets the message-processing tasks to a full-preempted task with priority based on the real-time operating system. The proposed gateway-embedded system activates the processing tasks when it receives/transmits messages from/to other nodes. Therefore, the proposed gateway-embedded system handles received messages based on priority. The message-processing tasks should know the destination protocol and node ID. We use routing look-up table mechanisms in our current version to determine the destination protocol and node ID.

EXPERIMENTAL RESULT

Latencies were measured when a fault occurred. The latencies are as follows: A length of time from the instant of recognizing the fault to the instant of responding with a signal to the main microcontroller, or to the instant of receiving a control directly from the main microcontroller, as shown in Figure 10. The latency is related to the response time against a fault. The response times against to the faults are measured by a digital oscilloscope A fault is triggered by pushing a button. When a fault is inserted into the main microcontroller, the main microcontroller becomes out-of-control in the software. After the checker recognizes the fault, the checker tries to help the main microcontroller recover. Figure 10-(a) shows the instant of fault generation, and the instant of the checker's respond. The checker generates a blue line. The checker tries to make the main microcontroller recovery periodically. The fault is recognized

and handled by the checker within 0.5 s (this is predefined criteria value). When the fault is removed, the checker stops the recovery operation, as depicted Figure 10-(b). The checker does not try to recover main microcontroller anymore after a fault is removed. Finally, if the checker recognizes the permanent fault from the main microcontroller, the checker immediately gets a control from the main microcontroller within 0.1μs. Figure 10-(c) represents the time elapsed until receiving a control from the main microcontroller. This elapsed time is not included in the time required to boot and initialize the checker. The measured time is the response time from the instant of recognizing the permanent fault by the checker to the instant of starting the gateway function by the checker alternatively. In Figure 10-(c), the red line remains at a high level when a permanent fault is recognized by the checker. The blue line remains at a low level when the gateway function is started by the checker.

CONCLUSION

Automotive distributed control systems are used in a wide range of control units. Several communication protocols are used to implement IVNs using LIN, CAN, and FlexRay. Gateway systems are needed to provide seamless communication between different networks. The gateway system requires high reliability and fault-tolerance as hard real-time systems. In this paper, we reviewed several fault-tolerant approaches and proposed and implemented a fail-safe gateway-embedded system using the extended asymmetric fault-tolerant architecture and fail-safe operation mechanism. The developed system is targeted at a gateway interfacing with LIN, CAN and FlexRay, but the concept of the proposed architecture can be applied to many applications. We evaluated the developed system and presented the results. We will establish the developed architecture to highly reliable automotive control units such as domain control units in near future.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No.2017R1C1B1004322)

REFERENCES

[1] W. Zeng, M. A. S. Khalid and S. Chowdhury, "In-Vehicle Networks Outlook: Achievements and Challenges," in IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 1552-1571, third-quarter 2016.

[2] W. Haas, P. Langjahr, "Cross-domain vehicle control units in modern E/E architectures", Internationales Stuttgarter Symposium: Automobil- und Motorentechnik, 2016.

[3] J. He, J. Li, L. Zhang, "A Distribution-Based Model for Electric/Electronic Architectures of Automotive," Lecture Notes in Electrical Engineering, vol. 364, pp. 587-598, 2016

[4] S. Philippi, "Analysis of fault tolerance and reliability in distributed real-time system architectures," Reliability Engineering & System Safety, Volume 82, Issue 2, pp. 195-206, November 2003.

[5] M. Ghadhab, M. Kuntz, D. Kuvaiskii, C. Fetzer, "A Controller Safety Concept Based on Software-Implemented Fault Tolerance for Fail-Operational Automotive Applications," International Workshop on Formal Techniques for Safety-Critical Systems, pp. 189-205, 2015.

[6] G. Weiß, P. Schleiß, C. Drabek, "Fail-operational E/E Architecture for Highly-automated Driving Functions," ATZelektronik worldwide, vol. 11, Issue 3, pp. 16-21, June 2016.

[7] R. Mariani, P. Fuhrmann, B. Vittorelli, "Cost-effective Approach to Error Detection for an Embedded Automotive Platform," SAE 2006 World Congress & Exhibition, Detroit, April 2006.

[8] Padma Sundaram and Joseph G. D'Ambrosio, "Controller Integrity in Auomotive Failsafe System Architectures," SAE 2006 World Congress & Exhibition, Detroit, April 2006.

[9] J. H. Kim, S. Seo, N. Hai, B. M. Cheon, Y. S. Lee, J. W. Jeon, "Gateway Framework for In-Vehicle Networks Based on CAN, FlexRay, and Ethernet," IEEE Transactions on Vehicular Technology, vol. 64, Issue 10, pp. 4472-4486, 2015.