# A Design of a Communication Processor for Implementing Local Network in Zonal Architecture

Kwonneung Cho[1], Kwanghyun Go, Seongmo An and Seung Eun Lee[a]
Department of Electrical Engineering, Seoul National University of Science and Technology
E-mail: [1] chokwonneung@seoultech.ac.kr

***Abstract* – As the complexity of in-vehicle network is increased, zonal architecture is proposed for the next generation network architecture. The zonal architecture divides the in-vehicle network into local networks and interconnects the local networks in a central gateway. This paper proposes a local network processor based on local interconnect network (LIN). The processor includes a light-weight core and dedicated LIN controller to achieve design goals such as design area and performance. The processor is fabricated with TSMC 180nm CMOS process and the characteristics are compared to Samsung 28nm process.**

***Keywords*—Zonal architecture, local network, LIN controller, bus functional model**

## I. INTRODUCTION

The trends of in-vehicle network systems are migrating to the zonal architectures to deal with the increasing network traffics [1,2]. A modern automotive includes enormous electrical devices such as multi-modal sensors and digital processors that require high speed and high bandwidth communications [3,4]. In addition, the complex connectivity of electrical devices causes intensive power consumption and communication errors. The zonal architecture divides the region of in-vehicle network into local networks and interconnects each local networks in a central gateway [5]. The distributed networks effectively address the issues by alleviating the network loads and bottlenecks [6].

In the zonal architecture, the central gateway needs to adopt high speed communication interface such as CAN with flexible data rate (CAN-FD) and ethernet in order to minimize the latency between the local networks [7,8]. CAN-FD is widely employed to conventional network domain achieving higher bandwidth than controller area network (CAN) [9]. Ethernet is employed to deal with the large size of data acquired from processing images or sensor data. High-level security and safety features are also critical part of the network systems [10]. To achieve the constraints, hardware encryption engines such as advanced encryption standard (AES) and safety mechanism such as lock step or error correction code (ECC) are applying to the network communication processors [11,12].

The local networks distribute the communication loads by processing the data derived from the local devices. As the adjacent ECUs and sensors are connected in the local networks, not only high-speed interface but also flexible, efficient, and power reasonable interfaces are essential. The in-vehicle network interfaces such as CAN, CAN-FD, FlexRay, single edge nibble transmission (SENT), and local interconnect network (LIN) are employed according to the requirements of connected electrical devices [13,14].

In this paper, a communication processor for local network is proposed. The designed processor includes a Cortex-M0 core and a hardware dedicated LIN controller. The Cortex-M0 is a lightweight core which advantages in design area and power budget but lacks in computing performance. In order to complement the performance of processor, the LIN controller performs communication functions such as decoding LIN frame header, responding messages, error detecting and handling. The Cortex-M0 core configures the LIN controller by accessing registers through advanced high-performance bus (AHB) interface, and the LIN controller offloads the communication loads. The proposed processor was designed with Verilog HDL, implemented with field programmable gate array (FPGA) and TSMC 180nm CMOS process.

The contributions of this paper are as follows. The hardware architecture of processor and LIN controller are proposed. A simulation model and method for verification of the processor are presented, and the simulation scenarios can migrate to FPGA implementation or application specific integrated circuit (ASIC) demonstration. The function of the processor is successfully demonstrated in ASIC level, and the experimental environment is presented. Finally, the design area and power results are compared with TSMC 180nm CMOS process and Samsung 28nm CMOS process.

The remained sections are described as follows. Section 2 explains the background of LIN communication and hardware required parts. Section 3 presents the architectures of the proposed processor and LIN controller. Section 4 describes the simulation model. Section 5 analyzes implementation and experiment results, and Section 6 concludes this paper.

a. Corresponding author; seung.lee@seoultech.ac.kr

Manuscript Received May 25, 2023, Accepted Jul. 29, 2023

## II. BACKGROUND

### A. LIN protocol

LIN communication provides low-speed data rate and employs single line bus interface. Fig. 1 shows the LIN frame format and network architecture. LIN network is implemented with one master node and multiple slave nodes. All the nodes in the LIN network share the bus line and communicate with data rate up to 20kbps. The LIN bus has two physical states called dominant and recessive. The dominant state is logically low and takes priority to the recessive state that is logically high. The LIN frame consists of two-part, header and response. The header includes the break field, break delimiter, sync field, PID field. Only a master node asserts the frame header, therefore LIN communication is scheduled by the master node. The response includes the data field and checksum field. A master node or slave node can release the response part.

The break field informs a start of the frame by remaining 13-bit of dominant state. The sync field sets the data rate of current frame by switching recessive and dominant state at 1-bit intervals. All the nodes have to synchronize the data rate dynamically according to the sync field. The PID field contains 6-bit of frame ID and 2-bit of the parity of the frame ID. The frame ID determines the publisher and subscriber of the data field according to the pre-defined rules. The parity bits indicate a validation of current frame header. When invalid parity bits are received, all the nodes in the LIN network assert error flag. The data field has dynamic data length from 1-byte to 8-byte. The data length is also pre-defined by the frame ID. The checksum field is calculated with the data from PID field to data field. The subscriber calculates the checksum value and compares the value to the received checksum field. When the checksum values are not matched, the subscriber asserts error flag.

### B. Hardware friendly parts

The 13-bit times of dominant state distinguishes the break field from the other fields. The fields from sync field to checksum field comply with a universal asynchronous receiver transmitter (UART) data format. A dominant bit is released at the start of the data and a recessive bit follows the 8-bit data. Therefore, from sync field to checksum field, the dominant state drives only up to 9-bit times. As a result, the 13-bit length of break field is a key point to detect a valid frame header. Detecting a valid break field is advantageous to hardware as measuring exact bit time is required.

The sync field determines the data rate of current frame. The 13-bit length of break field is also measured based on the 1-bit time of sync field. Estimating the data rate in sync field is a time sensitive task that is related to detecting valid frame header and synchronization. Therefore, a dedicated hardware for processing sync field and break field pair is required to detect valid frame header exactly.

LIN communication provides several error conditions. A parity error occurs when the frame ID and parity bits are not matched. A frame error indicates that the received data do not comply the stop bit condition defined in UART data format. A checksum error occurs when the checksum values are not matched. The error conditions are detected by checking the frame data bit sequentially. Therefore, detecting errors is a hardware friendly component. In this work, the designed LIN controller supports detecting valid frame header, checking data integrity, and error detecting.

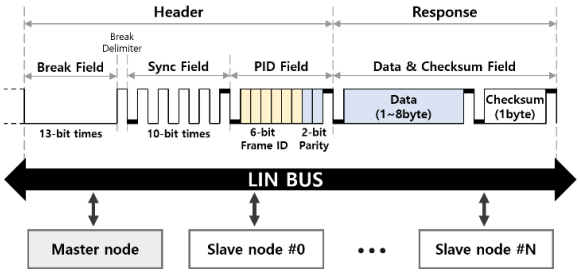


Fig. 1. LIN frame and network architecture.

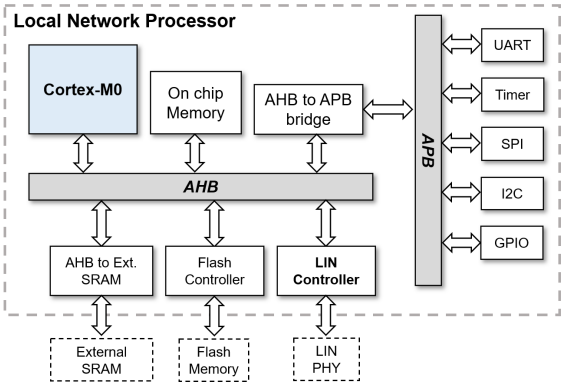


Fig. 2. Architecture of local network processor.

## III. ARCHITECTURE

### A. Local network processor

Fig. 2 shows the entire architecture of the local network processor. The processor employs a Cortex-M0 core and includes memory, bus, external interface, and peripherals. The Cortex-M0 core is a master node of the AHB for read and write operation. The on-chip memory, flash controller, external SRAM interface, and LIN controller are connected to the AHB. Program instructions are stored in the external flash memory and the instruction data are copied to the on-chip memory or external SRAM at reset sequence. The Cortex-M0 core fetches instructions from the internal memory or external SRAM and executes the program.

The LIN controller provides LIN functions such as detecting frame header, responding data, checking data integrity, and error handling. The Cortex-M0 accesses the registers of LIN controller through AHB and configures the LIN controller. A LIN bus is physically implemented through the external LIN PHY.

The advanced peripheral bus (APB) is implemented as a AHB slave node. The AHB to APB bridge converts AHB signals to APB protocol. The UART, timer, serial peripheral interface (SPI), inter-integrated circuit (I2C), and GPIO are connected as APB slave nodes. The UART, SPI, and I2C modules support serial communication interfaces for interconnecting with external devices. The timer and GPIO provides time-related functions and external pin controls

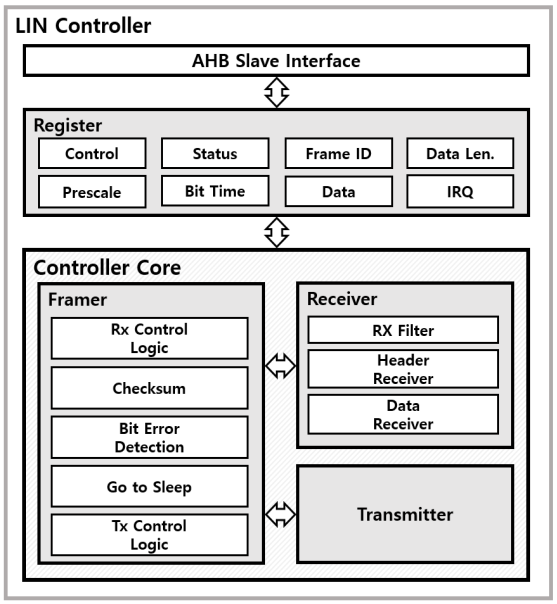


Fig. 3. Architecture of LIN controller.

### *B. LIN controller*

Fig. 3 is a block diagram of the designed LIN controller. The LIN controller is interfaced with AHB, and the processor can access the register by AHB to perform LIN communication. The receiver is a module that receives frames and includes a rx filter, a header receiver, and a data receiver. The rx filter mitigates LIN bus noises for stable receiving operations. The frequency bandwidth of filter is able to be configured according to the bus clock frequency and filter counter register value. The header receiver is designed to detecting valid frame header condition and data rate by dedicated hardware. The header receiver compares dominant bit time periodically to find break, sync field pair.

After receiving a valid frame header and estimating data rate, the data receiver samples data from the PID field to the checksum field. During the reception and transmission, the framer generates frames according to the data structure of LIN protocol. Framer also calculates the checksum and detects error conditions such as frame error, parity error, and checksum error. When an error occurs, the framer informs an interrupt request to the processor. Transmitter is a module that transmits frame responses based on transmission data stored in the register. Transmitter controls transfer sequence dynamically through the commands from register when exceptions occur.

## IV. SIMULATION

### *A. LIN controller simulation*

The designed LIN controller was simulated at block level and top level. A block level simulation is efficient for simulation time since the simulation does not include other design sources. The Fig. 4 shows the block level simulation environment of LIN controller. The LIN controller is a design under test (DUT). In order to modeling LIN bus and communication functions, the bus functional model (BFM) for AHB and LIN bus are implemented. The AHB BFM includes AHB read, write task and interrupt request (IRQ) model.

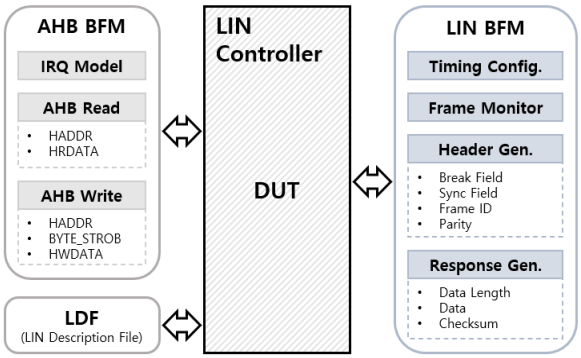


Fig. 4. Simulation environment of LIN controller.

CPU behaviors for control LIN controller were implemented through the AHB BFM. The register of LIN controller was configured with the AHB task according to the simulation scenarios.

The LIN BFM includes timing configuration, frame monitor, header generation, and response generation tasks. The timing configuration task sets data rate of LIN BFM. The header generation task asserts a LIN frame header to the receive pin of LIN controller. The frame ID is an argument of header generation task. The parity values are calculated automatically by the frame ID argument. The bit rate of header is determined through the configuration of timing task. When a frame header is received to the LIN controller, the LIN controller responds or ignores the header according to the register configuration. The response generation task asserts a LIN frame response to the receive pin of the LIN controller. The length and values of the receive data values are arguments of the task. The response generation task only calls after a header generation task. The checksum values are calculated internally from PID field of previous header generation task to the data of current response generation task. Receiving functionality of LIN controller is covered by the header generation task and response generation task. The frame monitor task samples the transmit pin of LIN controller with the data rate of timing configuration task. The frame monitor task detects transfer errors such as frame error and checksum error.

A LIN description file (LDF) is a standard format for LIN network implementation. The LDF includes the information such as communication speed, node names of network, communication schedule table, frame descriptions, and the relations of publisher and subscriber according to the PID. The simulation scenario of block level simulation is synchronized to the LDF by implementing LIN network through the BFMs. The LDF scenario is migrated to top level simulation and ASIC functional demonstration. Therefore, the same scenario is employed from simulation to demonstration. The functional demonstration environment is described in section 5.

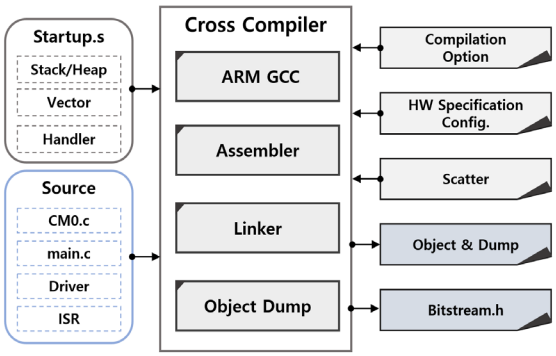


Fig. 5. Compile environment for top simulation.

### *B. Top simulation*

Fig. 5 shows the compile environment for top level simulation. The compiler includes GNU compiler collection (GCC), assembler, linker, object dump, etc. The assembler translates a program code into assembly languages and the GCC compiles the assembly to binary instructions for Cortex-M0. The linker maps the address of the compiled code and outputs the final object file. Object dump disassemble the compiled object files for debugging. In compile process, a hex format text file is also extracted to employ the instructions in top level simulation. The hex data is read in internal memory model when top level simulation starts.

Top simulation verifies the core functionality, bus interconnect, and peripherals. Test codes are written according to the simulation scenarios such as memory access test, flash interface, program download sequence, peripheral function test. In order to simulate the scenarios, simulation models for external SRAM and flash memory are implemented.

Especially the top-level simulation for LIN controller is a key feature. The LIN BFM is migrated to top level simulation for verifying communication functions. The Cortex-M0 is programmed to implement a LIN node described in the LDF and the LIN network is implemented through the LIN BFM. Fig. 6 shows the simulation result for frame header reception and response transmission of the LIN controller. A frame header including the break field, sync field, and PID field is inserted through the LIN receiver pin. When the frame header is received, the register informs current state of LIN communication and samples the receive data. The LIN controller responds to the frame header when the PID field is valid to the register configurations.

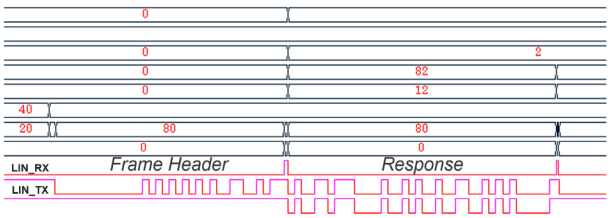


Fig. 6. Simulation result of LIN controller.

A LIN bus is implemented in the simulation environment, therefore the output of LIN transmitter signal is reflected to the LIN receiver signal. The simulation results confirm the integrity of the design.

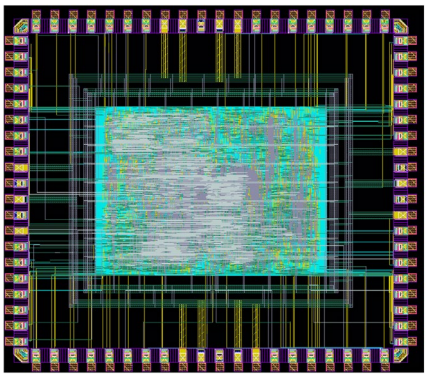

Fig. 7. Chip layout

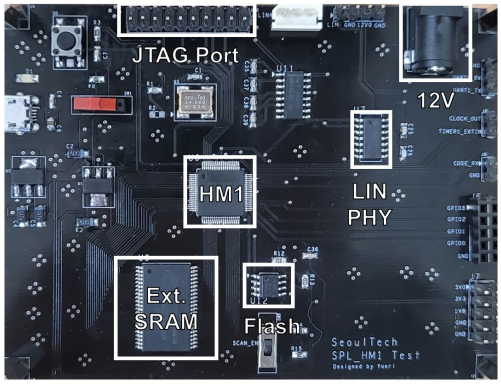


Fig. 8. Prototype PCB.

## V. CHIP IMPLEMENTATION AND RESULTS

### *A. Chip layout*

Fig. 7 shows the chip layout of the processor designed by the TSMC 180nm process. The designed processor includes a Cortex-M0 core, debug modules, peripherals such as UART, SPI, Timer and GPIO, and the LIN controller as a communication controller. Internal SRAM is not included in the design. Instead, an external SRAM is employed for code memory and data memory. The chip size is $3.4mm \times 3mm$ and the core size is $1785um \times 1385um$. The operating frequency of the processor is 24MHz, and each operating voltages of the standard cell and IO cell are 1.8V and 3.3V.

### *B. PCB prototyping*

A prototype PCB including the implemented chip (HM1), external SRAM, Flash memory, LIN PHY, and signal ports is designed as shown in Fig. 8. The flash memory receives instructions which compiled by Cortex-M0 compiler through the JTAG port. Additionally, the JTAG port is employed for debugging of the HM1, reading the internal registers and memory values of the specific addresses. When the PCB board powered-on or reset, the HM1 reads program instructions from flash memory and stores the instructions into the external SRAM. After a booting sequence, the HM1 performs the program sequentially by accessing the external SRAM. The LIN PHY employed for the physical layer of LIN communication, and the 12V connecter provides power to the LIN PHY.

### C. Verification

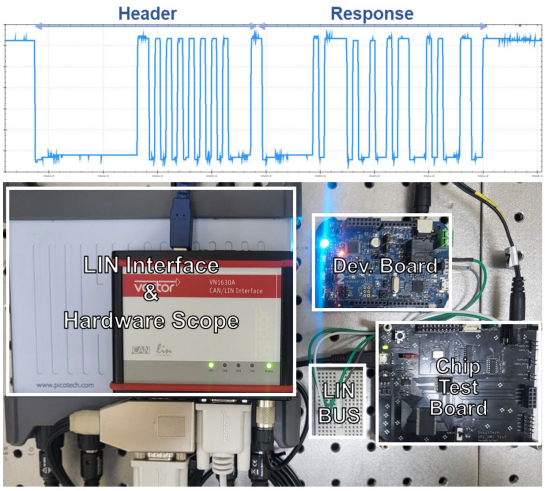


Fig. 9 Experimental environment of LIN communication.

In order to verify the designed processor and LIN controller, the verification environment is implemented as shown in Fig. 9. The prototype PCB, LIN interface, hardware scope, and NXP automotive development board are employed. Vector-VN1630A and PicoScope-5000 are employed as a LIN interface and hardware scope. The hardware scope provides a LIN physical layer and captures a LIN frame sampled through the LIN bus. A LIN network is implemented with the LIN interface, chip test board and NXP development board. The LIN interface links a LDF to Vector-Canoe software and provides virtual LIN nodes described in the LDF. Therefore, the LDF network condition is migrated from simulation to verification. A target LIN node that is specified in the LDF is implemented with the chip test board by programming the designed processor. Another LIN node in the LDF is implemented with the NXP development board. The NXP development board supports LIN communication functions and physical layers. By employing the NXP board as a LIN node, we demonstrated that the designed processor is able to communicate with the commercial boards. Other LIN nodes in the LDF are modeled with virtual LIN node. We verified that the designed processor successfully detects valid LIN headers and responds to the headers exactly. The captured waveform is presented in Fig. 9.

TABLE I. Area and power report.

| Block | Area ($um^2$) | | Power (mW) | |
|---|---|---|---|---|
| | Samsung 28nm | TSMC 180nm | Samsung 28nm | TSMC 180nm |
| Cortex-M0 | 5,478 | 178,812 | 0.203 | 1.450 |
| LIN Ctrl. | 2,102 | 39,140 | 0.0819 | 1.972 |
| UART | 476 | 9,744 | 0.0207 | 0.549 |
| SPI | 253 | 4,623 | 0.0115 | 0.275 |
| Timer | 710 | 7,468 | 0.0286 | 0.377 |
| Total | 9,019 | 239,787 | 0.346 | 4.623 |

### D. Chip specification

Table 1 presents the area and power estimation of the designed processor according to the TSMC 180nm CMOS process and Samsung 28nm CMOS process. Area and power for the detailed modules commonly included in the processors are compared. The design area of the Cortex-M0 core is measured as 5,478$\mu m^2$ for the Samsung 28nm and 178,812$\mu m^2$ for the TSMC 180nm. As the core design in TSMC 180nm includes additional logic such as debug and debugger access port (DAP) modules, the area of Samsung 28nm process takes 3.06% of the design area compared to TSMC 180nm process. The LIN controller, UART, SPI, and Timer modules include the same design in both processes, and Samsung 28nm process accounts for an average design area of 5.81% compared to TSMC 180nm process.

To compare the power consumption of the designed processor, the power is measured using design compiler (DC) based on a saif of the simulation. We employ a vectored power estimation to estimate the power in a power intensive situation. The saif includes the toggle information according to the simulation results. The core consumes 1.45mW in the TSMC 180nm process and 0.203mW in the Samsung 28nm process. The Samsung process power result occupies 14% of power consumption compared to TSMC process. The total power values including core, LIN controller, UART, SPI and timer are estimated 0.346mW, 4.623mW according to the process. As a result, Samsung 28nm process consumes 7.48% of power compared to the TSMC 180nm process in average.

## VI. CONCLUSION

In this paper, a communication processor for local network is proposed. The processor includes a Cortex-M0 core, dedicated LIN controller and peripherals for achieving performance and power budget. To verify the design, block-level simulation and top-level simulation are performed. Simulation models such as bus functional model, external SRAM and flash memory are implemented. The simulation model implements LIN network scenarios by employing a LDF which includes LIN nodes, frame definitions, and network schedules. The LIN network scenarios are migrated to physical demonstration to synchronize results. The proposed processor is successfully demonstrated in ASIC level.

The experimental environment is implemented with a test PCB board, LIN interface, hardware scope, and NXP development board. We verify the functionality of LIN communication by capturing LIN frames through the experimental environment.

The designed processor is fabricated with TSMC 180nm CMOS process. We compared the design area and power with Samsung 28nm process.

## ACKNOWLEDGMENT

The chip fabrication and EDA tool were supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] Z. Threet et al., "Securing Automotive Architectures with Named Data Networking," *2022 IEEE 25th International Conference on Intelligent Transportation Systems*, pp. 2663-2668, 2022.

[2] L. Tippe, L. Pilgrim, J. Fröschl and H. -G. Herzog, "Modular Simulation of Zonal Architectures and Ring Topologies for Automotive Power Nets," *2021 IEEE Vehicle Power and Propulsion Conference*, pp. 1-5, 2021.

[3] K. Cho, J. Kim, D. Choi, Y. Yoon, J. Oh, S. Lee., "An FPGA-Based ECU for Remote Reconfiguration in Automotive Systems," *Micromachines*, vol. 12, no. 11, pp.1309, 2021.

[4] M. L. Han, B. I. Kwak and H. K. Kim, "Event-Triggered Interval-Based Anomaly Detection and Attack Identification Methods for an In-Vehicle Network," *IEEE Transactions on Information Forensics and Security*, Vol. 16, pp. 2941-2956, 2021.

[5] A. Frigerio, B. Vermeulen and K. G. W. Goossens, "Automotive Architecture Topologies: Analysis for Safety-Critical Autonomous Vehicle Applications," *IEEE Access*, Vol. 9, pp. 62837-62846, 2021.

[6] H. Tsuda et al., "Proposal for a Highly Reliable In-Vehicle Optical Network: SiPhON (Si-Photonics-Based In-Vehicle Optical Network)," *2022 27th OptoElectronics and Communications Conference (OECC) and 2022 International Conference on Photonics in Switching and Computing*, pp. 1-3, 2022.

[7] H. M. Kadry, A. Gupta, J. M. Lawlis and M. Volpone, "Electrical Architecture and In-Vehicle Networking: Challenges and Future Trends," *2022 IEEE International Symposium on Circuits and Systems*, pp. 1009-1013, 2022.

[8] O. Alparslan, S. Arakawa and M. Murata, "Next Generation Intra-Vehicle Backbone Network Architectures," *2021 IEEE 22nd International Conference on High Performance Switching and Routing*, pp. 1-7, 2021.

[9] K. Cho, H. W. Oh, J. Kim, Y. W. Jeong and S. E. Lee, "A Local Interconnect Network Controller for Resource-Constrained Automotive Devices," *2022 IEEE International Conference on Consumer Electronics*, pp. 1-3, 2022.

[10] W. Wu, J. Dai, H. Huang, Q. Zhao, G. Zeng and R. Li, "A Digital Watermark Method for In-Vehicle Network Security Enhancement," *IEEE Transactions on Vehicular Technology*, pp. 1-12, 2023.

[11] A. Shreedhar et al., "Low Gate-Count Ultra-Small Area Nano Advanced Encryption Standard (AES) Design," *2019 IEEE International Symposium on Circuits and Systems*, pp.1-5, 2019.

[12] S. -I. Pae et al., "Minimal Aliasing Single-Error-Correction Codes for DRAM Reliability Improvement," *IEEE Access*, vol. 9, pp. 29862-29869, 2021.

[13] S. Jin, J. -G. Chung and Y. Xu, "Signature-Based Intrusion Detection System (IDS) for In-Vehicle CAN Bus Network," *2021 IEEE International Symposium on Circuits and Systems*, pp. 1-5, 2021.

[14] T. Andreica, C. -D. Curiac, C. Jichici and B. Groza, "Android Head Units vs. In-Vehicle ECUs: Performance Assessment for Deploying In-Vehicle Intrusion Detection Systems for the CAN Bus," *IEEE Access*, vol. 10, pp. 95161-95178, 2022

**Kwonneung Cho** received the B.S. degree in the Department of Electronic Engineering at the Seoul National University of Science and Technology, Seoul, Korea, in 2021. He is a M.S. student in the Department of Electronic Engineering at the Seoul National University of Science and Technology, Seoul, Korea. His research interests include computer programming, digital system design and SoC for Embedded processor.

**Kwanghyun Go** received the B.S. degree in the Department of Electronic Engineering at the Seoul National University of Science and Technology, Seoul, Korea in 2022, where he is pursuing the M.S. degree. His current research interests include digital system design, In-Vehicle network, AI accelerator and System-on-Chip.

**Seongmo An** is a undergraduate student in the Department of Electronic Engineering at the Seoul National University of Science and Technology, Seoul, Korea. His current research interests include digital system design, processor for System-on-Chip and AI accelerator.

**Seung Eun Lee** received the Ph.D. degree in electrical and computer engineering from the University of California, Irvine (UC Irvine) in 2008 and the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon in 1998 and 2000, respectively. After graduating, he had been with Intel Labs., Hillsboro, OR, where he worked as Platform Architect. In 2010, he joined the faculty of the Seoul National University of Science and Technology, Seoul. His current research interests include computer architecture, multi-processor system-on-chip, low-power and resilient VLSI, and hardware acceleration for emerging applications.