Preliminary Design on AUTOSAR CP Ethernet Simulation using POSIX-based Socket

Harim Lee, Jeonghun Cho

Kyungpook National Univ, Kyungpook National Univ

hw05165@knu.ac.kr, jcho@knu.ac.kr

POSIX 기반 소켓을 이용한 AUTOSARCP 이더넷 시뮬레이션의 예비설계

이하림, 조정훈

경북대학교, 경북대학교

Abstract

As the complexity of Hardware and Software increases, the need for a virtual co-simulation environment comes out. Virtual co-simulation means that it can be simulated without any other hardware. That is, software-in-the-loop simulate (SILS) is possible. The complexity also induces the need for Ethernet. Because as the number of connections increases, a simple but noise-resistant interface, advantages of CAN communication, increases the number of Buses, and the limit of data rate results in a decrease in communication efficiency. Then Ethernet, having a high data rate and switch mechanism for communication efficiency, is needed for not only external communication but also internal communication. Therefore, settings for SoAd, Tcp/IP are needed to use Ethernet, also an environment is needed to simulate these settings. The SoAd is the first layer for using Ethernet blow PDU Router (PduR), so this paper proposes the SoAdFmi to use an already well-designed Linux socket (POSIX-based) for Ethernet.

Ⅰ. Introduction

As software development accelerates, there has also been a change in embedded system development that hardware based. Some direct the development of the software to meet the specifications of the hardware that already exists, but the others attach or exchange the specifications of the hardware for the desired operation of the software. In the former case, this change will not be significant because it is developed to a fixed hardware specification. However, in the latter case, if the customer's requirements change during the development process, then the hardware needs to change, and then the software development direction will change, or in the worst case, the code developed will have to be discarded and redeveloped. If the software is implemented before the hardware, you must also wait for a prototype or part of the final product to verify its actual behavior, hardware-in-the-loop simulation (HILS) [1]. Even if the SW is satisfied with the hardware configuration, it will be modified during the validation phase for better performance, which will also be an unexpected additional cost to development. This leads to the need for a virtual simulation environment that can minimize the costs (might not be eliminated), not a hardware-based simulation. Vehicles are also unavoidable that require this simulation. As vehicle embedded system developed, a platform called AUTOSAR was created. (The AUTOSAR partnership is an alliance of OEM manufacturers. The partnership established a virtually open industry standard for automatic software architecture. It will serve as a basic infrastructure for

the management of functions within both future applications and standard software modules [2]). Because vehicles are not small, and prototypes for verification are more expensive and consume more time these days. In particular, as the number of built-in ECUs increases, the physical aspects of the integration validation process can be considered, because they might be created by different development environments as a component. Therefore, we need a simulation environment called vECU (virtual ECU) [3], and in this paper, we propose SoAdFmi to validate SoAd (Socket Adaptor) [4] among AUTOSAR CP modules. SoAd is responsible for creating sockets for the desired domain/type/protocol and passing the PDUs from the PDU Router [5-6] in the higher layer to the TcpIp [7], using the SoAdFmi proposed in this paper allows us to simulate the Ethernet below in a Unix-based environment (in this paper Linux). Unix already has an API defined for the BSD socket (Berkeley socket), and we want to use it to define a Functional Mock_up Interface (FMI) that meets the POSIX standard. You might wonder if SoAdFmi can simply replace SoAd. This can be implemented due to the characteristics of AUTOSAR CP and FMI. One of the main concepts of AUTOSAR CP's standardized ECU software architecture is the separation of hardware-independent application software and hardware-oriented basic software (BSW) by means of the software abstraction layer RTE (runtime environment) [8]. Therefore, if compatibility is met, the independence will make it possible to simulate using SoAdFmi. The characteristics of FMI will be explained in the next section. In the next section (sec2.), we briefly introduce FMI to be used for SoAdFmi that we proposed. In section 3, (a) we will introduce what SoAd is in AUTOSAR CP, (b) how it looks when SoAdFmi is applied, (c) and how to implement SoAdFmi. And the last section, we conclude with the final goal of FMI for implementing communication between vECU.

Ⅱ. Functional Mock-up Interface

If the software is implemented before the hardware, you must also wait for a prototype or part of the final product to verify its actual behavior, hardware-in-the-loop simulation (HILS) [1]. Even if the SW is satisfied with the hardware configuration, it will be modified during the validation phase for better performance, which will also be an unexpected additional cost to development. This leads to the need for a virtual simulation environment that can minimize the costs (might not be eliminated), not a hardware-based simulation. Vehicles are also unavoidable that require this simulation. As vehicle embedded system developed, a platform called AUTOSAR was created. (The AUTOSAR partnership is an alliance of OEM manufacturers. The partnership established a virtually open industry standard for automatic software architecture. It will serve as a basic infrastructure for the management of functions within both future applications and standard software modules [2]). Because vehicles are not small, and prototypes for verification are more expensive and consume more time these days. In particular, as the number of built-in ECUs increases, the physical aspects of the integration validation process can be considered, because they might be created by different development environments as a component. Therefore, we need a simulation environment called vECU (virtual ECU) [3], and in this paper, we propose SoAdFmi to validate SoAd (Socket Adaptor) [4] among AUTOSAR CP modules. SoAd is responsible for creating sockets for the desired domain/type/protocol and passing the PDUs from the PDU Router [5-6] in the higher layer to the TcpIp [7], using the SoAdFmi proposed in this paper allows us to simulate the Ethernet below in a Unix-based environment (in this paper Linux). Unix already has an API defined for the BSD socket (Berkeley socket), and we want to use it to define a Functional Mock_up Interface (FMI) that meets the POSIX standard. You might wonder if SoAdFmi can

simply replace SoAd. This can be implemented due to the characteristics of AUTOSAR CP and FMI. One of the main concepts of AUTOSAR CP's standardized ECU software architecture is the separation of hardware-independent application software and hardware-oriented basic software (BSW) by means of the software abstraction layer RTE (runtime environment)[8]. Therefore, if compatibility is met, the independence will make it possible to simulate using SoAdFmi. The characteristics of FMI will be explained in the next section. In the next section (sec2.), we briefly introduce FMI to be used for SoAdFmi that we proposed. In section 3, (a) we will introduce what SoAd is in AUTOSAR CP, (b) how it looks when SoAdFmi is applied, (c) and how to implement SoAdFmi. And the last section, we conclude with the final goal of FMI for implementing communication between vECU.

Ⅲ. Overview of SoAdFmi

A. SoAd

In AUTOSAR CP, SoAd module aims at bridging the gap between these two concepts, including TcpIp, which uses the dynamic configuration and routing to communicate over Ethernet, and AUTOSAR CP, which follows the concept of a predetermined and static communication relationship at compile time. For this purpose, the functions that call SoAd at the upper layer of the SoAd (PduR) open the connection of a specific (static) socket and pass the PDU to that socket. In addition, the callback functions that SoAd passes to the upper module update the connection status of the current socket or the message's Tx/Rx status. On the other hand, some functions allow SoAd to assign sockets to the lower layer TcpIp. In TCP, the client requests to connect to the server (if using 'ANY' could be 'dynamic'), and the server checks the connection request of the other client and updates the connection status. There are also UDP that carries data without connections between these specific sockets

B. Preview of including the SoAdFmi

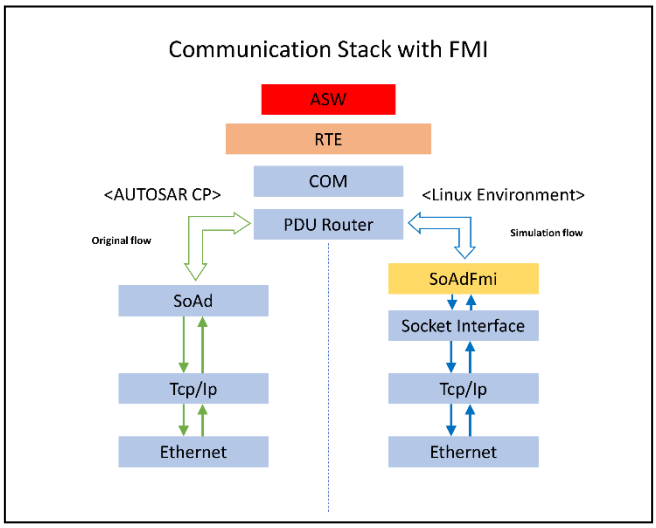

Figure 1. Preview of Communication Stack[11]

If FMI is introduced, the above flow will be possible. On the left, it will send and receive data through the lowest Ethernet driver along the original flow, as shown in the Extended AUTOSAR Communication Stack [4]. and on the right is the Linux environment's stack. If the simulation environment is used, the API between the existing PduR and SoAd will have to work the same between SoAdFmi. This is because it has to show the same behavior when removing the simulation environment and replacing it with the same hardware. In addition, because it uses Linux (UNIX-based) APIs, applications created in Linux (Win or UNIX) environments will need to work with FMI. Only then will it be compatible with other FMUs, furthermore, a vECU environment can be created to simulate a completely assembled ECU.

C. To implement SoAdFmi

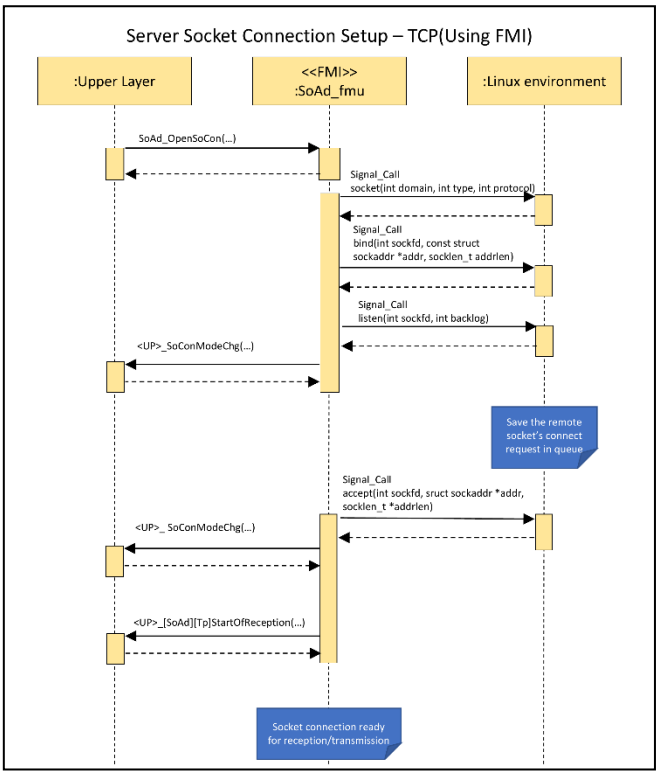

Figure 2. Sequence diagram with FMU

In order to implement SoAdFmi, all actions within AUTOSAR CP API related SoAd [12] will have to be done first. So when we organize the API, there are not only Types used in SoAd, but also functions that call from the top and to the bottom, and call back functions that to the top and from the bottom. We need to connect SoAd/TcpIp API and Linux socket API[10] in relation to the sequence diagram (Fig-2). Starting with the most basic TCP socket connection setup, SoAd_OpenSoCon() opens the socket connection specified by SoConId, and TcpIp_SoAdGetSocket() receives which to use IpV4 or IpV6 through DomainType, and whether to use the corresponding TCP Connect via ProtocolType. This process corresponds to socket() on Linux, and the different thing with SoAd's is that Linux manages everything as a file, so SoConId corresponds to file description (blow fd). In addition, domain, the input parameter of socket(), has more types than Ipv4/Ipv6, and the type corresponding to protocolType is replaced by SOCK_DGRAM for UDP and SOCK_STREAM for TCP. These represent the types of data that each carries, Datagram and Stream. (In addition, there are SOCK_RAW, SOCK_SEQPACKET, but it is not a protocol used by AUTOSAR CP, so this paper will not deal with them.) TcpIp_Bind() connects the SoConId of the opened socket directly with the LocalAddrId (IpAddress) and Port Figure 1. Preview of Communication Stack[11] Figure 2. Sequence diagram with FMU corresponding to the local resource in the input argument (assigned via Pointer in TcpIp_SoAdGetSocket). At this time, if the wildcard 'ANY' is included in the LocalAddrId, it is automatically determined in the case of the client. And in the case of the server, data is received regardless of the IP address. This corresponds to the bind() function in Linux, where SoConId is also fd in Linux, and the previous return value of socket() is used. The 'addr' corresponding to localAddrId gives the storage location of the socket to be managed as a file as an input. And in the next case of the sequence, through TcpIp_Listen(), the socket corresponding to SoConId can be changed to a state where the client can request a connection, 'Listen state', and to a MaxChannel with a parameter that can receive a maximum connection request. When TcpIp accepts this request and returns E_OK(Std_returnType), SoAd sends a Notification to the upper layer that the socket connection status has changed. This process is replaced by the listen() function in Linux and replaces the socket corresponding to fd with Listen state, but the one corresponding to MaxChannel is set to 'int backlog'. The ECU corresponding to the client then passes through TcpIp_TcpConnect() from the socket corresponding to the SoConId of the client to the local address of the server socket, RemoteAddrPtr. The client and server then exchange data with the TCP flag to establish a Transport Layer Security (TLS) connection through the Tcp 3-way handshake. The server then listens for these client connection requests and sends SoAd_TcpAccepted(), Notification, at the end of the TLS connection process. Once again,

SoAd receives these Notifications and sends _SoConModeChg and _StartOfReception to the upper layer in order. This process is replaced by accept() in Linux, but the difference from AUTOSAR CP is that if the client's connection request is detected, the connection status is changed by Notification, while if accept() is not called even if the connection request is received, it is blocked until there is request. And if there is a request and accept() is called and successful, the fd of the new socket is returned, which is a file that has the address information of the remote socket (in this case client's) and uses this address to send and receive data with the connected remote socket. (There are other sequences of UDP connections and Tx/Rx processes, but all of them can not be covered in this paper.) And the FMI Implement needs to bus (terminal, in FMI) [13] because FMI has principles. One of these is 'Simulator independence'. It is possible to compile, link, and distribute an FMU without knowing the environment in which the FMU will be loaded. So if using other simulation tools, UNIX-based, they may not be able to call the functions within the FMUs. To call functions with terminals, it have to use predefined signals, like meaning that 'want to call socket() function'. Additional using terminal, it can import terminal using for PDU, frame bus.

Ⅳ. Conclusion

The need for vECU in the development of vehicle embedded systems also increases the need for FMI to support it. Therefore, we first want to define FMI for the communication stack for communication between multiple vECU. We proposed SoAdFmi for the use of Ethernet, which is a part of vECU, so we wrote a preview for the implementation as above. If implemented, communication between different FMUs will be possible, as well as simulation close to the actual ECU.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 1711160343, Development of virtual ECU-based vehicle-level integrated simulation technology for vehicle ECU application software development and verification automation)

REFERENCES

[1] M. Zlatkovic, P. T. Martin and I. Tasic, "Implementation of transit signal priority and predictive priority strategies in ASC/3 software-inthe-loop simulation," 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), 2011, pp. 2130-2135, doi: 10.1109/ITSC.2011.6082847.

[2] A. U. T. O. S. A. R. development cooperation, AUTOSAR, 15-Nov2022. (https://www.autosar.org/)

[3] Prostep IVIP association, WhitePaper V-ECU, 04-May-2020. (https://www.prostep.org/fileadmin/downloads/WhitePaper_VECU_2020_05_04-EN.pdf)

[4] AUTOSAR, "Specification of Socket Adaptor", 25-Nov-2021. (https://www.autosar.org/fileadmin/user_upload/standards/classic/21- 11/AUTOSAR_SWS_SocketAdaptor.pdf)

[5] AUTOSAR, "Specification of PDU Router", 25-Nov-2021. (https://www.autosar.org/fileadmin/user_upload/standards/classic/21- 11/AUTOSAR_SWS_PDURouter.pdf)

[6] AUTOSAR, "Glossary", 25-Nov-2021. (https://www.autosar.org/fileadmin/user_upload/standards/foundation/ 21-11/AUTOSAR_TR_Glossary.pdf)

[7] AUTOSAR, " Specification of TCP/IP Stack", 25-Nov-2021. (https://www.autosar.org/fileadmin/user_upload/standards/classic/21- 11/AUTOSAR_SWS_TcpIp.pdf)

[8] A. U. T. O. S. A. R. development cooperation, "Classic platform," AUTOSAR, 06-Dec-2021. (https://www.autosar.org/standards/classic-platform/)

[9] "Functional mock-up Interface," Functional Mock-up Interface. (https://fmi-standard.org/)

[10] Functional mock-up interface specification, 08-Nov-2022. (https://fmi-standard.org/docs/3.0/)

[11] G. A. Fink, P. Muessig, and C. North, "Visual correlation of host processes and network traffic," IEEE Workshop on Visualization for Computer Security, 2005. (VizSEC 05)., Nov. 2005.

[12] M. Kerrisk, "Ch.56-61, SOCKETS," in The linux programming interface a linux und UNIX system programming handbook, San Francisco, CA: No Starch Press, 2010. (https://man7.org/tlpi/index.html)

[13] "FMI Layered Standard Network Communication," Network communication and FMI 3.0, v1.0. (https://modelica.github.io/fmi-ls-bus/main)