

The SPI Synchronization Method for Data Efficiency and Integrity between MCUs

Jae Kyu Lee*, Jae Woo Ahn*, Jong Oh Kim**, Seong Gon Choi*

*Information & Communication Engineering, ChungBuk National University, Cheongju-si Chungcheongbuk-do, Korea

** Fisys, 117 Dunsan-Daero Seo-Gu Daejeon, Korea

princeljk@cbnu.ac.kr, jwahni@cbnu.ac.kr, jokim@fisys.co.kr, choisg@cbnu.ac.kr

Abstract— In this paper, we propose an additional data transferring SPI synchronization method to increase efficiency when data transferring between MCUs.

In the MCU of the Wi-Fi module, when the size of data to be transmitted or received exceeds a predetermined size, data is transmitted in an inefficient manner. Therefore, if the size of the network increases or the number of devices to communicate increases, the data transmission rate becomes more and more inefficient. Also, since a data loss occurs depending on a situation, a method of preventing data loss is needed.

Therefore, we propose an additional data transferring SPI synchronization method that checks total length of the data to be transmitted, efficiently transmits data according to the data size, and prevents data loss.

The additional data transferring SPI synchronization method transmits data more efficiently than the existing data transmission method, increases the data transmission speed, and prevents data loss to ensure integrity of data.

Keywords— SPI, STM32, CC3220, Additional Data Transfer, Synchronization

I. INTRODUCTION

Advances in Internet of Things(IoT) and sensor network technologies continue to generate large amounts of data, and wireless technologies continue to be researched and developed to meet the needs of various user service quality, such as fast data transfer rates and low latency. The number of wireless devices is growing exponentially, and in many network infrastructures, the importance of wireless infrastructure is higher than wired infrastructure. By 2019, 92% of manufacturers will adopt IoT technology. New architectures are being developed in a variety of areas, including wireless interfaces, smart antennas, wireless sensor networks, wireless low-power automation systems, and cloud computing. In addition, transmission technologies used for wireless communications, such as directional antennas, continue to be researched and developed. Using a wireless infrastructure, while minimizing wired infrastructure, efforts are increasing to build a network that has many advantages[1]-[2].

Microprocessor Unit(MCU) is widely used as part of infrastructure construction equipment for wireless telecommunication infrastructures. MCU is widely used to build telecommunication infrastructures as it has the advantage of controlling input and output of various

peripheral devices in a single chip. Synchronization is essential to communication between multiple MCUs that control various I/O functions. The Serial Peripheral Interface(SPI) is then used for the synchronization process. The SPI enables full-duplex communication and does not require a transmitter, there is no need to use transceiver. It is also suitable for a variety of tasks because the interface is simple and fast using only four pins in an Integrated Circuit(IC) package. Using these various benefits of SPI, it is widely used for communication between MCUs[3]-[4].

In order to develop high-capacity multi-channel data collection modules using MCU chips, many developers also develop around SPI communications links. It is also addressed by SPI-MCU data communication techniques to eliminate bottlenecks in radio sensing systems[5].

Therefore, we believe that efficient data communication between wired equipment MCU and wireless Wi-Fi module MCU used to build IoT and sensor network environments will bring great efficiency to the entire wireless infrastructure. In this paper, we discuss the problems of traditional MCU-to-MCU communication methods and propose a new method using SPI.

We will compare the data communication rates between MCUs using the existing methods with the data communication rates between MCUs using the methods proposed in this paper. Comparing the two measurements will prove that the SPI synchronization method proposed in this paper is a more efficient way than the one used previously.

Chapter 2 describes the SPI synchronization process that operates on an existing MCU. Chapter 3 describes the overall structural description of the experiment and the theory of how it is proposed. Chapter 4 analyzes the experimental environment, methods, and results of experiments. Chapter 5 discusses the analysis of test results and the direction in which they will be studied.

II. RELATED WORKS

There are a number of studies that focus on MCU and SPI communications. To improve communication performance between MCUs, a variety of studies are underway, such as securing large-capacity data storage, fast reading and writing speeds, and efficient data transmission[3]-[4].

However, the SPI communication process used at the API driver level in communication between MCUs performs an inefficient operation. This is not mentioned in technologies that reducing data transfer time and eliminate bottlenecks through SPI process between MCUs[5]. Failure to improve the inefficient way of communicating between MCUs will limit performance improvement and will result in inefficient performance relative to the size of the system.

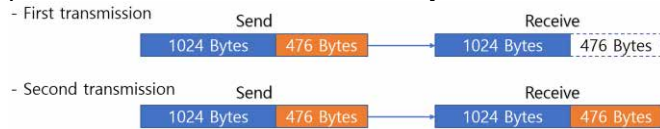


Figure 1. Example of Inefficient Data Transfer Process

The existing inter-MCU SPI communication method carries out an inefficient operational process. MCU of Wi-Fi module sends and receives data up to 1,024 bytes at a time. In other words, the maximum transmission unit(MTU) of MCU is 1,024 bytes. If it is necessary to send and receive 1,500 bytes of data, it should be sent and received twice. This is because it can only send and receive up to 1,024 bytes at a time, so it has to send and receive 1,500 bytes of data until it is stored.

Figure 1 shows the inefficient data transfer process. Assuming that 1,500 bytes must be sent and received, the sender must send 1,500 bytes twice to receive the first 1,500 bytes from the receiving side to store 1,024 bytes and the remaining 476 bytes from the second 1,500 bytes. In other words, 3,000-byte data buffers were used to transmit 1,500 bytes and 1,500-byte meaningless data were created.

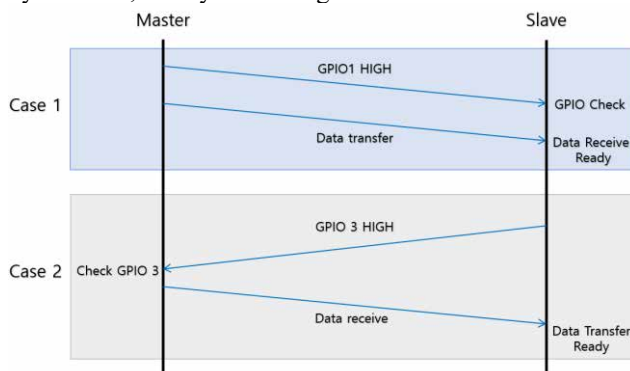


Figure 2. Example of SPI Data Loss

In addition, there are cases where data loss occurs if the existing SPI synchronization process is still used between MCUs. Figure 2 shows when data loss occurs. The SPI process is divided into Master and Slave, which then proceeds with the synchronization process. When sending data to Slave, the Master sends a data transfer signal(GPIO1 HIGH) to Slave and immediately transmits the data. However, if Slave is not ready to receive data, data loss occurs(Case 1 in Figure 2). Another case occurs when transferring data from Slave to the Master. For data transfer, Slave sends a signal to the Master(GPIO3 HIGH) to prepare for data reception. Master checks the signal and immediately proceeds to receive data. However, since Slave is not ready to send data, it can prepare

to send data while the Master receives the data and send Slave only after the Master receives the data(Case 2 in Figure 2).

Traditional methods of communication between MCUs are inefficient for data transfer and can result in data loss. This requires a new approach that is efficient and does not result in data loss.

III. SPI SYNCHRONIZATION

In this clause, we explain the overall structure of the experiment and the SPI synchronization process between MCUs. It then explains the proposed data transferring SPI synchronization method in the paper.

A. Overall Structure

Figure 3 shows the overall structure of the experiment performed in this paper. Reason for communication between Ethernet and Wi-Fi is to establish IoT and sensor network environment. The entire experimental environment is based on TCP/IP network used in IoT and sensor network environments. This is a structure that transfers incoming packets from Ethernet to Wi-Fi side or delivers incoming packets to Ethernet side. STM32 chips were selected as the MCU for forwarding incoming packets to Wi-Fi and CC3220 chips as Wi-Fi module MCU. STM32 is a chip based on a 32-bit ARM processor core and CC3220 is a chip that contains a network processor as part of the MCU platform. In other words, the tested communication between MCUs is the SPI synchronization process between STM32 and CC3220.

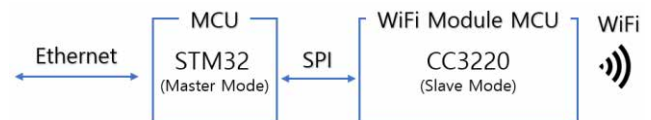


Figure 3. Overall Structure of SPI Synchronization Process

B. Master Mode & Slave Mode for SPI synchronization

Serial Peripheral Interfaced Bus(SPI) is a synchronous serial data connection standard that operates in full-duplex mode. It consists of Master Out Slave In(MOSI) and Master In Slave Out(MISO), Clock(SCK) and SS(Slave Select). MOSI is the transmission of data from Master mode to Slave mode and MISO is the transmission of data from Slave mode to Master mode. Because it operates in both modes, full-duplex communication is possible that can be sent and received at the same time. It is called 3-wire or 4-wire depending on whether using SS. 3-wire mode is a mode that does not use the SS. In this paper, we experimented with 3-wire mode because the Master mode and Slave mode communicate one-to-one[6]-[7].

Because data communication of SPIO is based on full-duplex communication, transmission and reception are simultaneous. It is on the Master side that determines the timing of the data transmission. MCU(STM32) uses Master mode and Wi-Fi module MCU(CC3220) uses Slave mode. In Slave mode, polling mode is always maintained to wait for the data of Master side. When the Master gives the data, the Slave receives the data and uses full-duplex communication for the case that the Slave transfers the data to the Master at the same

time. To implement full-duplex communication method, we used GPIO to handle SPI.

C. Data Loss Preventing SPI Synchronization Process

This section describes the SPI communication process that prevents data loss between the Master(STM32) and the Slave(Wi-Fi module MCU). There are two cases of data loss: The first is when a Slave is not ready for reception and the Master is sending data, the second is when the Slave is not ready for transmission and the Master receives the data. To avoid this, we changed blocking mode to non-blocking mode. The reason is that both Master and Slave must remain in their present condition. For example, in situations where the Master must send data to Slave, the Slave needs to ask the Master to receive the data while it remains ready to receive the data. The maintained state called back to the original state after the data is received.

The following will explain the process of transmitting and receiving between Master and Slave using a method that prevents data loss. Figure 4 shows the SPI synchronization process for transferring data from the Master to the Slave. When Ethernet data is received by the Master, a “GPIO1 HIGH” signal is sent to Slave. “GPIO1 HIGH” means that there is data to be transferred from the Master to the Slave. When the signal is received, Slave prepares to SPI receive. Send “GPIO2 HIGH” to the Master as a signal that SPI ready. Use semaphores to maintain the current state. The Master that checks the value sends the data to Slave. When the data transfer is complete, the GPIO2 changes back to the LOW value and returns to the polling mode. The value of GPIO1 is changed to LOW and means that the Master is not currently transferring data.

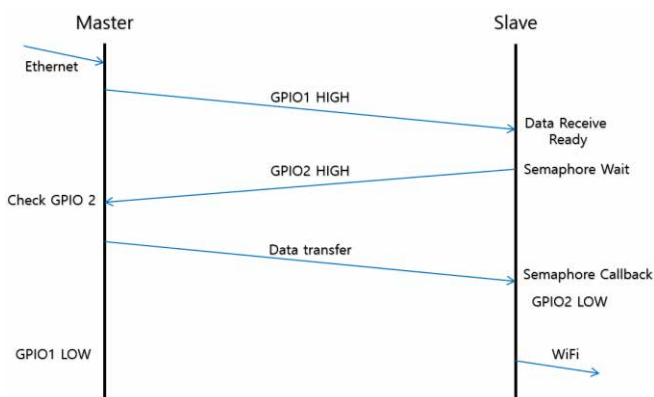


Figure 4. SPI Synchronization Process, Master to Slave

Figure 5 shows the SPI synchronization process for transferring data from Slave to Master. When the data to be sent from Slave to the Master is generated, check the GPIO1 first to determine the status of the Master. If the Value of GPIO1 is LOW, Slave transmits the “GPIO3 HIGH” to the Master, which stands for SPI transmission is ready. GPIO3 generates a rising interrupt on the Master. The Master that receives the “GPIO3 HIGH” checks the interrupt and receives the data. At this time, the data can be transferred immediately

after the interrupt is generated to the “GPIO3 HIGH” value because the GPIO1 was already verified. When the data transfer is complete, GPIO3 is changed to LOW.

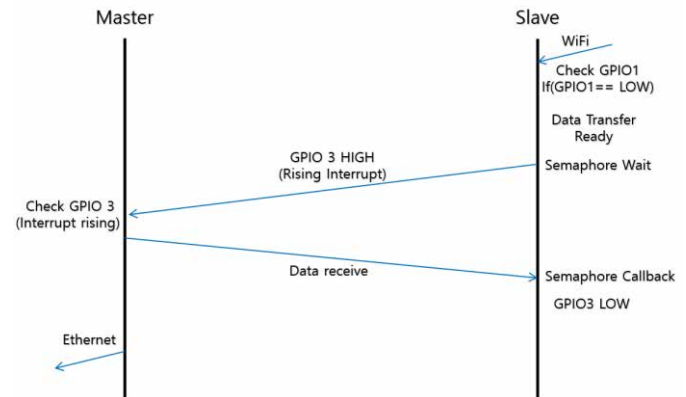


Figure 5. SPI Synchronization Process, Slave to Master

D. Additional Data Transferring SPI Synchronization

The maximum size of data that is sent and received by Wi-Fi module MCU is 1,024 bytes. Transmitting and receiving 1,500 bytes of data in excess of 1,024 bytes works in an inefficient way that requires sending and receiving 1,024 bytes twice. However, additional data transferring SPI synchronization method works differently and do not work in an inefficient way.

Assume transferring 1,500 bytes of data. Additional data transferring method first receives data of the size of the Wi-Fi module MCU transmission data(1,024 bytes, Figure 6). It then looks at the IP header of the packet to determine if it should be sent or received further. It checks the total length of the data by checking the 2-byte “Total Length” field in the IPv4 Header(Figure 7). From the IP header, inspect the “Total Length” field that represents the total length of the data to determine whether additional data should be sent or not. If it does not exceed 1,024 bytes, no additional data transfer will take place, and if it exceeds 1,024 bytes, additional data will be sent to receive the remaining data. Figure 8 illustrates the additional data transfer.

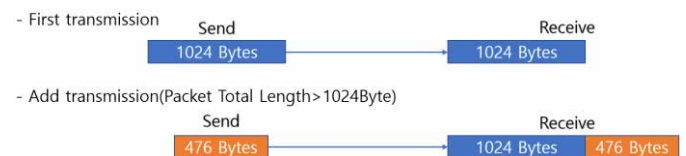


Figure 6. First and Extra Reception at Additional Data Transferring Method



Figure 7. IPv4 Packet Header

The following describes the process of additional data transferring SPI synchronization. Figure 9 illustrates the SPI synchronization process for additional data transfer in the communication between Master and Slave. If the Ethernet data to be sent from the Master to the Slave is generated, the “GPIO1 HIGH” signal is sent to the Slave to prepare for SPI reception. When ready for reception, “GPIO2 HIGH” is sent to the Master, then the Master sends the data. At the end of the data transfer, GPIO2 is changed to LOW, and the packet size is checked. Check the “Total Length” of the IP header to finish the process if it does not exceed 1,024 bytes. If it exceeds 1,024 bytes, Slave prepares to SPI receive by sending “GPIO2 HIGH” signal to the Master. On receipt of the “GPIO2 HIGH” signal, the Master sends the remaining data to Slave. When the data transfer is complete, GPIO2 is changed to LOW.

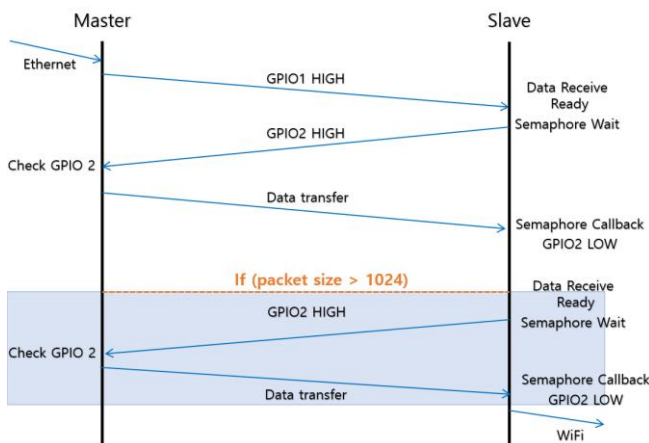


Figure 8. Process of Additional Data Transferring SPI Synchronization

E. Packet Header - Total Length

The additional data transferring SPI synchronization method performs additional data transfer when the MCU exceeds the maximum size of 1,024 bytes. The total length of the packet is determined through the “Total Length” field by checking the header of the packet to send additional data.

Figure 9 shows the pseudocode that obtains “Total Length” from the packet header. When a packet to be sent and received during SPI synchronization process between MCUs is generated, the generated packet is placed in the receiving and sending buffer. We named the buffer “transfer”. `transfer[2]` multiplied by 256 to change to decimal because the “Total Length” value exists in hexadecimal in the third and fourth fields of the packet. This changed decimal value is used as the value of the total length of the packet.

Data Transmission Direction: Master -> Slave

```
packsize is transfer[2]*256
packsize is packsize + transfer[3]
```

Figure 9. Pseudocode to obtain Total Length from the Packet Header

IV. IMPLEMENTATION

Chapter 4 describes the experimental environment and methods for implementation and discusses analysis of the results of the experiment.

1) Experimental Environment

Programming tasks to implement the entire experimental structure used Keil uVision and Code Composer.

We used iperf3 to compare the speed of data transfer between the previously used MCU communication method and the proposed additional data transferring SPI synchronization method.

2) Experimental Method

We will compare our data transfer rates using iperf3 for program codes that have implemented the existing inter-MPU SPI synchronization process and those that have applied our suggested additional data transferring SPI synchronization. SPI synchronization process that was used previously was applied with a method that prevents loss of data. The experimental scenario was the same as the Master transmitting the data to Slave. Explanation of how the existing method and proposed method are implemented is explained using pseudocode.

Data Transmission Direction: Master -> Slave

set GPIO1 to HIGH

while(SPI Receive Ready)

SPI Transfer

set GPIO1 to LOW

Figure 10. The Master Pseudocode for Sending Data to Slave

It is consistent with the process of the Master communicating the data to Slave as described in Chapter 3. Figure 10 shows the Master pseudocode for sending data to Slave, and Figure 11 shows the Slave pseudocode for receiving data from the Master. The pseudocode in Figure 10 and Figure 11 is easy to understand by referring to the graph in Figure 4. When “GPIO HIGH” is set and the Slave is ready to receive data, the data is transferred. “Semaphore_Wait” is activated to maintain the state until it is transmitted, and “Semaphore_Wait” is called back when the transmission is finished. And both GPIO1 and GPIO2 are changed to LOW.

Additional data transferring SPI synchronization method includes determining whether additional data should be sent based on packet size. This is the pseudocode of the added part.

Data Transmission Direction: Slave <- Master

read GPIO1

```
if(GPIO1 is HIGH)
    SPI_data_size is 1024
    SPI Receive Ready
```

```
set GPIO2 to HIGH
Semaphore Wait
set GPIO2 to LOW
```

Figure 11. The Slave Pseudocode for Receiving Data from Master

Data Transmission Direction: Master -> Slave

Figure 10...

```
Check_packet_size is 1500
If(Check_packet_size > 1024){
    while(SPI Receive Ready)
        SPI Transfer
    }
set GPIO1 to LOW
```

Figure 12. The Master Pseudocode for Sending Data to Slave

Data Transmission Direction: Slave <- Master

Figure 11...

```
Check_packet_size is 1500

if(Check_packet_size > 1024){
    SPI_data_size is 476
    SPI Receive Ready

    set GPIO2 HIGH
    Semaphore Wait
    set GPIO2 LOW
```

Figure 13. The Slave Pseudocode for Receiving Data from Master

Figure 12 shows the Master pseudocode for sending data to Slave. Subsequent to Figure 10, check the packet size before “SPI_Transfer” ends and becomes “GPIO1 LOW”. If the packet size exceeds 1,024 bytes, perform one more action in Figure 10 and change GPIO1 to LOW because there is data remaining to be sent to Slave. The reason why “SPI_data_size” is set to 476 in the portion for transmitting additional data is because the packet size is 1,500 bytes. In this case, the reason why the packet size is fixed to 1,500

bytes is because the overall structure of the experiment is the IoT and sensor network structure, and there is no case of exceeding 1,500 bytes.

Figure 13 shows the Slave pseudocode for receiving data from Master. Figure 13 also shows that after the operation of Figure 11 is finished and the packet size is over 1,024 bytes, the operation of Figure 11 is performed once more.

3) Implementation Result

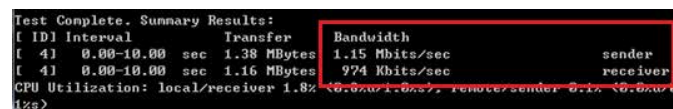


Figure 14. Data Transmission Rate of Existing Method

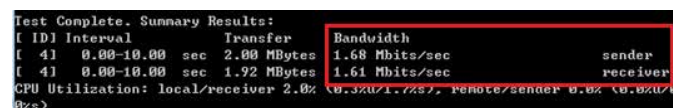


Figure 15. Data Transmission Rate of Additional Data Transferring Method

Figure 14 shows data transmission rate of SPI synchronization using existing method, and Figure 15 shows data transmission rate of SPI synchronization using additional data transferring method. When the proposed method is used, the data transmission rate is about 0.5 Mbytes/sec faster.

V. CONCLUSION

We have found that the data loss and the transmission data operate inefficiently when a certain size is exceeded in the existing SPI synchronization method between MCUs. During the SPI synchronization process, Master sends data before it is ready to receive Slave, or Master performs data reception before it is ready to send Slave, resulting in data loss. To prevent this, the blocking mode was changed to non-blocking mode to be implemented to maintain and operate the current state using semaphore. Data efficiency was enhanced by SPI synchronization method that determines whether additional data should be sent after checking the size of packets in the packet header. When the proposed method is used, data transmission rate is about 0.5 Mbytes/sec faster.

Therefore, we have prevented data loss through additional data transferring SPI synchronization and verified that data transfer rates between MCUs are improved.

Further research shows that better performance can be expected if only the additional data that needs to be sent and received is available instead of the fixed value 476.

ACKNOWLEDGMENT

This work was supported by “Human Resources Program in Energy Technology” of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade, Industry & Energy, Republic of Korea. (No. 20164030201330).

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information

Technology Research Center) support program(2018-0-01396)
supervised by the IITP(Institute for Information &
communications Technology Promotion).

Corresponding Author: S.G. Choi (choisg@cbnu.ac.kr)

REFERENCES

- [1] Mamta Agiwal, Abhishek Roy, Navrati, "SaxenaNext Generation 5G Wireless Networks: A Comprehensive Survey" IEEE Communications Society, 19 February 2016
- [2] M. Kohvakka, M. Hannikainen, T.D. Hamalainen, "Wireless sensor prototype platform," IEEE, 2-6 Nov. 2003
- [3] Zhenghua Xin1, Hongmei Lu1, Liangyi Hu2, Jianxin Li1, "Implementation of SPI and Driver for CC2430 and C8051F120," IEEE, 17 May 2012
- [4] Wei Zhang, "The design of Atmega32 MCU SPI and MAX191 serial A/D communication," IEEE, 23 June 2011
- [5] Seung - mok Yoo Pai H. Chou, "MHP: Master - Handoff Protocol for Fast and Energy - Efficient Data Transfer over SPI in Wireless Sensing Systems, ETRI Journal, 01 August 2012
- [6] Akos Szekacs, Tibor Szakaill and Zoltan Hegyk6zi, "Realising the SPI communication in a multiprocessor system," 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 24-27 May 2015
- [7] Dhananjay V. Gadre, Sarthak Gupta, "Serial Communication: SPI and I2C," Springer New Delhi, 17 October 2017



Jae Kyu Lee received B.S degree in the College of Electrical & Computer Engineering, Chungbuk National University, Korea in 2017. He is currently a M.S. candidate in School of Electrical & Computer Engineering, Chungbuk National University. His research interest is network security



Jae Woo Ahn received B.S. degree in the College of Electrical & Computer Engineering, Chungbuk National University, Korea in 2017. He is currently a M.S. candidate in School of Electrical & Computer Engineering, Chungbuk National University. His research interest is network programming.



Jong Oh Kim received B.S. and M.S. degree in Electronics Engineering from Kyeongbuk National University in 1990 and 1992 respectively. He is currently CEO in Fisys Inc. His research interests is SDN, NFV.



Seong Gon Choi received B.S. degree in Electronics Engineering from Kyeongbuk National University in 1990, and M.S. and Ph.D. degree from KAIST, Korea in 1999 and 2004, respectively. He is currently a professor in the College of Electrical & Computer Engineering, Chungbuk National University. His research interests include smart grid, IoT, mobile communication, high-speed network architecture and protocol.