Journal of KIIT. Vol. 16, No. 2, pp. 67-73, Feb. 28, 2018. pISSN 1598-8619, eISSN 2093-7571 67
http://dx.doi.org/10.14801/jkiit.2018.16.2.67

다중 제어 장치 연결을 위한 Modbus 통신 클래스 설계

김동환*[1], 김보헌**, 송정호*[2], 김황래***

A Design of Modbus Communication Class for Multiple SCU Connections

Dong-Hwan Kim*[1], Bo-Heon Kim**, Jeong-Ho Song*[2], and Hwang-Rae Kim***

요 약

모드버스(Modbus) 프로토콜에서는 SCU의 상태 값을 읽는 서로 다른 서비스 요청 메시지에 대해 값만을 응답하기 때문에 어떤 전송 메시지에 대한 응답인지 식별할 수 없다. 따라서 하나의 서비스 요청 메시지를 전송하고 이를 처리한 후에 다음 메시지를 전송할 수 있다. 본 논문에서는 통신 클래스 내부에 전송 메시지의 전송 순서를 관리하는 인덱스 코드 정의 표(ICDT)를 작성하여 모드버스 통신을 사용하는 환경에서 서로 다른 SCU에 다중의 동시적인 메시지를 전송할 수 있는 통신 클래스를 제안하였다. 실험 결과, MCU에서 전송한 서비스 요청 메시지에 대한 처리가 완료되지 않은 상태에서 다음 서비스 요청 메시지의 전송이 가능하였으며, ICDT의 인덱스 코드를 이용하여 SCU에서 응답한 메시지의 서비스 유형을 식별 및 통신 에러 발생 시 에러가 발생한 SCU 위치를 식별할 수 있었다. 또한, MCU 통신 프로그램에서 동시적인 다중의 서비스 요청이 발생한 경우에 전송 메시지들의 처리시간이 약 56% 정도의 성능 향상을 보였다.

Abstract

Modbus protocol does not identify which the SCU response message is corresponding to the service request message because the SCU response messages include only the response value. Also the Modbus protocol can transmit a service request message after transmitting the previous service request message. In this paper, we propose a Modbus application protocol communication class with index code definition table to solve the above problems. As a experimental result, the proposed protocol shows that service request messages can be transmitted without processing the previous request messages, and, it is possible to identify which type of error occurred by using the index code definition table when a communication error occurred. And when simultaneous multiple service requests are generated, the proposed protocol shows that the processing time can be improved about 56% compared to the Modbus protocol.

Keywords

master-slave communication, modbus, communication protocol, communication class, remote control

* 공주대학교 컴퓨터공학과
- ORCID[1]: https://orcid.org/0000-0002-6697-7565
- ORCID[2]: https://orcid.org/0000-0001-8447-1747
** ㈜이지에스 기술이사
- ORCID: https://orcid.org/0000-0001-5875-8090
*** 공주대학교 컴퓨터공학부 교수(교신저자)
- ORCID: https://orcid.org/0000-0001-9378-4139

· Received: Jan. 26, 2018, Revised: Feb. 11, 2018, Accepted: Feb. 14, 2018
· Corresponding Author: Hwang-Rae Kim
Dept. of Computer Engineering, Kongju National University
Tel.: +82-41-521-9227, Email: plusone@kongju.ac.kr

Ⅰ. 서 론

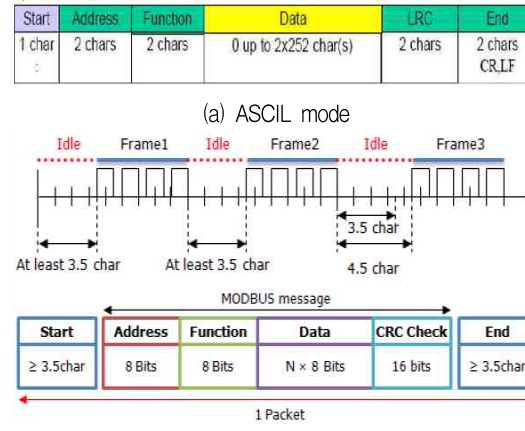
무선 네트워크, 센서 기술의 발달 및 스마트폰과 같은 지능화된 단말기의 발전으로 모든 사물이 인터넷과 연결되는 IoT(Internet of Things) 시대가 열리고 있으며, IoT 기술의 발전은 사회 전반적으로 응용되고 있다[1]-[3]. 이런 IoT 기기 및 산업 장비에서 사용되는 각종 제어 장치들은 일반적으로 서비스를 처리하고 그 결과를 제공하는 슬레이브 제어 유닛(SCU, Slave Control Unit)으로 사용되고 있다. 이런 SCU들은 PC, PLC 또는 임베디드 제어 시스템 등과 같은 마스터 제어 유닛(MCU, Master Control Unit)이 요청하는 온도, 속도 및 압력 제어 등 다양한 서비스를 처리하기 위해 통신 프로토콜을 이용하여 정보 취득 및 제어를 수행한다[4].

모드버스(Modbus) 프로토콜은 다양한 종류의 SCU를 연결하기 위해 일반적으로 사용되는 프로토콜로써, 메시지 포맷의 주소와 함수 코드 및 데이터 주소를 이용하여 하나의 통신 회선을 통해 여러 SCU들에 대한 서비스를 처리할 수 있다. 또한, SCU에 전송된 메시지에 대한 응답 메시지가 항상 존재하기 때문에 응답 메시지를 통해 서비스 요청 메시지가 정상적으로 전달되었는지를 확인할 수 있다. 그러나 MCU의 읽기 요청에 대한 SCU의 응답 메시지는 처리 결과로 값만을 반환하기 때문에 어떤 종류의 서비스 요청에 대한 응답인지 식별할 수 없다. 따라서 MCU 통신 프로그램 작성 시 하나의 서비스 요청 메시지를 전송하고 해당 메시지 처리가 완료된 후에 다음 메시지를 전송하므로 처리 효율에 한계가 있다.

본 논문에서는 이런 문제점을 개선하기 위해 통신 클래스 내부에 전송 메시지들을 관리하는 별도의 ICDT(Index Code Definition Table)을 작성하여 어떤 서비스에 대한 응답 메시지인지 식별이 가능하고 서비스 처리 속도를 향상시킬 수 있는 모드버스 통신 클래스를 제안하였다.

Ⅱ. 관련 연구

다양한 SCU 장치들을 연결할 수 있도록 표준화된 모드버스 프로토콜은 전 세계적으로 널리 보급되어 사용되고 있는 자동화 프로토콜의 하나로서, RS-232/422/485 및 이더넷 디바이스를 지원한다. 또한 PC, PLC, DCS, HMI, 계측기 등의 많은 공업 기기 및 국가 주요 핵심 기반 시설을 관리하는 SCADA 시스템에 사용되고 있으며[5]-[7], 전송 방식으로는 ASCII 모드와 RUT 모드가 있다. 그림 1은 모드버스 프로토콜의 메시지 형식을 나타낸다[8].



(a) ASCIL mode

(b) RTU mode

그림 1. Modbus 프로토콜의 메시지 형식
Fig. 1. Message format of Modbus protocol

모드버스 프로토콜은 SCU 제조업체에서 제공하는 메모리 맵의 레지스터 값을 쓰거나 읽음으로써, SCU의 정보를 취득하거나 제어를 수행할 수 있기 때문에, 숫자 코드만을 이용하는 모드버스 프로토콜의 단점을 보완하기 위해 일부 SCU 제조업체에서는 별도로 작성한 모드버스 응용 프로토콜을 추가로 제공하는 경우도 있다.

표 1과 2는 SCU 제조업체에서 제공하는 모드버스 응용 프로토콜의 형식과 명령어 예를 나타내는 것으로[9][10], 대부분의 모드버스 응용 프로토콜은 시작 문자와 끝 문자를 재 정의하고 함수 코드를 대신하여 문자열로 인식할 수 있는 명령 코드를 별도로 제공하고 있다.

표 1. 모드버스 응용 프로토콜 형식
Table 1. Modbus application protocol format

Start Char	Address	Command	Data	Check Sum	End Char
STX 0x02	1~99	Reference of Command		Sum Addr to Data	CRLF 0x0D 0x0A

표 2. 모드버스 응용 명령어 표
Table 2. Modbus application command list

Command	Description
RSD	D-Rregister Continuous Read
RRD	D-Register Random Read
WSD	D-Rregister Continuous Write
WRD	D-Register Random Write
…	…
AMI	Information of Controller

그러나 여전히 SCU의 응답 메시지만으로는 어떤 서비스 요청에 대한 응답인지 식별할 수 없고 연속적인 서비스 요청에 대한 처리가 어렵다.

III. 통신 클래스 설계

3.1 모드버스 프로토콜 문제점 분석

그림 2는 모드버스 프로토콜을 이용한 통신 예를 나타내는 것으로, 01번 주소를 가지는 SCU에게 현재 온도 설정 값(SV: 0x0039)과 현재 온도 값(PV: 0x03e8)을 읽기 요청(Function Code: 04)하였다. 그러나 SCU에서 응답한 메시지에서는 읽기 요청한 결과 값(0x00fa, 0x00f5)만을 반환하기 때문에 응답 메시지로는 어떤 서비스 요청에 대한 응답인지 식별할 수 없다.

3.2 통신 에러 검색 알고리즘 설계

표 3은 MCU의 서비스 요청 메시지에 대한 에러 검색 및 응답 메시지 식별을 위해 인덱스 코드와 메시지 정보 및 전송 시간 등을 정의한 인덱스 코드 정의 표이다. 표의 해당 항목들은 통신 클래스 내부에서 송·수신 메시지 처리 시 값을 설정한다.

그림 3은 표 3의 인덱스 코드 정의 표를 이용한 통신 에러 검색 알고리즘에 대한 함수 처리 과정을 나타내는 의사코드이다. 정상은 -1, SCU의 미 응답 에러는 0~255까지의 인덱스 코드 값, SCU에서 에러를 응답한 경우 해당 인덱스 코드에 256을 더한 256부터 511까지의 값을 반환한다.

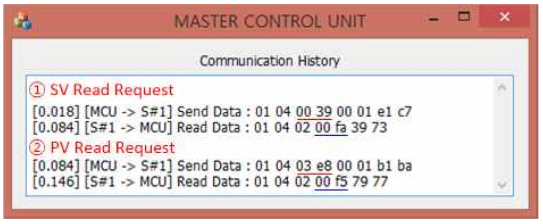


그림 2. 모드버스 통신 예
Fig. 2. Modbus communication example

표 3. 인덱스 코드 정의 표
Table 3. Index code definition table (ICDT)

	Index Code	Send Message	Kind of Receive	Send Time (tick count)
first	0	Message #1	1	1000
	1	Message #2	1	1200
	2	Message #3	1	1500
	…			
	255	Message #256	1	4700
	0	Message #257	0	5000
	…			
current	n	Message #idx	1	6100

```
* Define List
  - CUR_RECEIVE_STEP : Last Receive Index (Initialize value = 0)
  - _ERR_TIME_OUT_ : Send Message Timeout define (default: 1000msec)
  - GET_Kind_of_Receive() : Kind of Receive Value (in Table)
  - GET_TimeElapsed() : Measurement of elapsed time (unit: msec)
* Return Value : Error is greater than or equal to 0, No error is -1

Sub Get_ErrorSearch() as INTEGER
 INTEGER iIndexCode = 0
 FOR i = (CUR_RECEIVE_STEP + 256) TO (CUR_RECEIVE_STEP + 1) STEP -1
  iIndexCode = i MOD 256
  IF GET_Kind_of_Receive(iIndexCode) = 0 THEN
    IF GET_TimeElapsed(iIndexCode) > _ERR_TIME_OUT_ THEN
      RETURN iIndexCode
    END IF
  ELSE IF  GET_Kind_of_Receive(iIndexCode) = 2 THEN
    RETURN iIndexCode + 256
  END IF
 NEXT i
 RETURN -1
End Sub
```

그림 3. 에러 검색 알고리즘
Fig. 3. Error search algorithm

3.3 통신 클래스 설계

그림 4는 본 논문에서 설계한 통신 클래스의 구성을 나타내는 것으로, MCU 응용 프로그램 영역, 각 SCU 제조업체에서 제공하는 서비스를 처리하기 위한 SCU별 처리 클래스 영역 및 SCU와 통신을 수행하는 통신 클래스 영역으로 구성하였다.

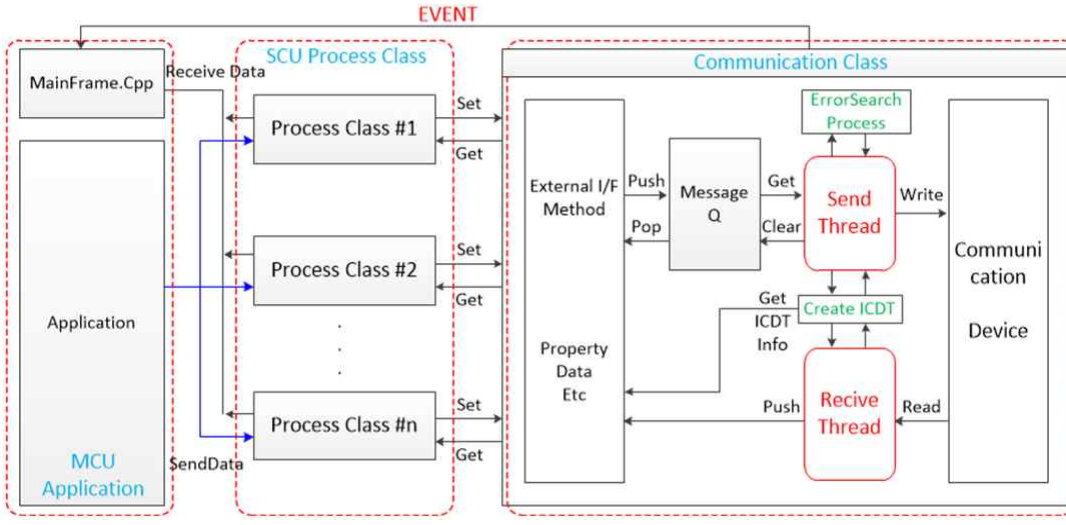


그림 3. 에러 검색 알고리즘
Fig. 3. Error search algorithm

동작 과정으로는 MCU 응용 프로그램에서 SCU 처리 클래스를 이용하여 SCU의 정보를 취득하거나 제어를 수행하는 함수를 호출하면 SCU 처리 클래스에서는 프로토콜 포맷에 맞춰 변환한 메시지를 통신 클래스로 전달한다. 통신 클래스에서는 해당 메시지를 큐에 등록하고, 이후 큐에 등록된 메시지는 통신 클래스 내부의 송신 스레드(Send Thread)에 의해 SCU로 메시지를 전송하게 된다.

또한, 송신 스레드에서는 전송 시점에 인덱스 코드 정의 표(ICDT)의 수신 쪽(Kind of Receive)값을 0으로 설정하고, 일정 주기 간격으로 ICDT의 정보를 이용하여 통신 에러를 검색한다. 통신 에러 발생 시 ICDT의 인덱스 코드와 함께 에러 정보를 이벤트를 통해 MCU 응용 프로그램 내의 MainFrame.cpp에 전달한다.

MainFrame.cpp 내의 이벤트 처리 함수 내에서 전달 받은 메시지는 각 SCU 처리 클래스에 전달하며, SCU 처리 클래스에서는 ICDT의 인덱스 코드를 이용해 서비스 유형을 식별하고 에러 형태에 따라 메시지 박스 등을 통해 알람을 발생하거나 메시지를 재전송할 수 있다.

SCU에서 응답한 메시지는 수신 스레드(Receive Thread)에서 메시지 구문 분석을 통해 얻어진 수신된 메시지의 SCU 주소를 이용해 ICDT에서 해당 SCU에 최근 전송한 인덱스 코드를 취득한 후, 정상 메시지인 경우 ICDT의 수신 쪽 값을 1로 설정하고, 에러 응답 메시지인 경우 2로 설정한다. 이후 해당 인덱스 코드와 함께 수신된 메시지 정보를 이벤트를 통해 MCU 응용 프로그램 내의 MainFrame.cpp에 전달한다.

MainFrame.cpp 내의 이벤트 처리 함수에서는 수신된 메시지를 각 SCU 처리 클래스에 전달한다. SCU 처리 클래스에서는 수신된 메시지 및 인덱스 코드를 이용하여 서비스 유형을 판단 한 후, 해당 서비스 유형에 따라 수신된 값을 설정하거나 정보로 이용할 수 있다.

Ⅳ. 실험 및 성능 분석

4.1 실험 환경

본 논문에서 실험은 설계한 통신 클래스의 처리 성능을 비교·분석하기 위해 혈액 검사 시 사용되는 진공 채혈관 조립 설비의 운영 환경을 토대로 제안한 통신 클래스의 적용 여부에 따른 비교 실험을 실시하였으며, ① SCU 장치에서 에러 발생 시 비교 실험, ② MCU의 연속적인 서비스 요청 시 전송 메시지 처리시간에 관한 비교 실험을 수행하였다.

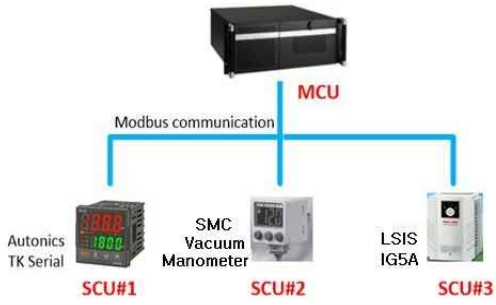


그림 5. 실험 환경
Fig. 5. Experimental environment & configuration

표 4. 진공 채혈관 장비의 SCU 정보
Table 4. SCU information in vacuum blood equipment

SCU Name	SCU Address	Usage
Thermo Controller	01	Thermo Control and Monitoring
Pressure Controller	02	Control Vacuum Pressure
Inverter	03	Control Air Speed

그림 5는 실험에서 사용된 진공 채혈관 조립 설비의 시스템 구성으로, SCU 장치는 약품 주입 공정에서 온도를 제어하기 위해 사용하는 온도 제어기, 드라이어(Dryer) 공정에서 사용되는 인버터, 일정량의 혈액을 주입하기 위해 사용되는 진공압력 제어기 등으로 구성된다. 각 SCU 주소는 표 4와 같이 정의하여 실험을 진행하였다.

4.2 SCU 에러 발생 시 비교 실험

본 실험은 특정 SCU에서 에러가 발생할 경우 처리 과정에 관한 실험으로, 표 5와 같은 순서 및 조건으로 실험을 진행하였으며, 그림 6은 SCU에서 에러가 발생한 경우의 MCU 화면이다.

그림 6에서 ① 통신 클래스를 적용하지 않은 상태에서는 베큠(Vacuum) 읽기 요청 후 미 응답 에러가 약 1초 후에 발생하고, 이후 다음 메시지가 전송되는 것을 확인할 수 있다. 그러나 ② 제안한 통신 클래스를 적용하면, 베큠 읽기 요청에 대한 응답을 기다리지 않고 다음 서비스인 PV값 읽기 요청 메시지가 전송되고, 이후 약 1초 후에 베큠 읽기 요청에 대한 미 응답 에러가 발생하는 것을 확인할 수 있다.

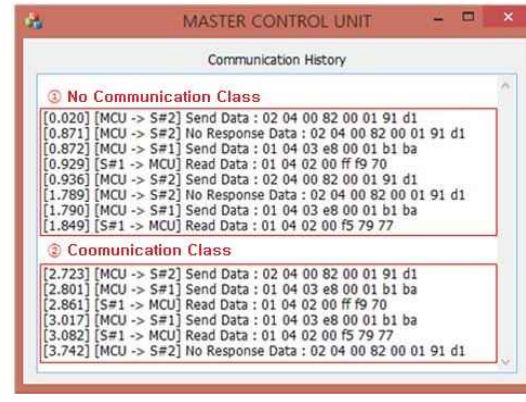


그림 6. SCU 에러 시 MCU 화면
Fig. 6. MCU screen in case of SCU error

표 5. 실험 순서와 조건
Table 5. Experiment sequence and conditions

Content	Explanation
Experiment Sequence	1) Read current Vacuum value 2) Read current Thermo value 3) Read current Vacuum value 4) Read current Thermo value
Experiment Conditions	1) Send time interval : 100 ms 2) No Response time interval : 1sec 3) Error SCU address: #2

이는 제안한 통신 클래스를 적용하면, 전송 메시지를 관리하는 ICDT을 이용하면 이미 전송된 메시지에 대해 에러 검색 및 SCU에서 응답한 메시지의 식별이 가능하므로 먼저 전송한 메시지의 응답이 없는 상태에서도 다음 서비스 요청 메시지를 전송할 수 있기 때문이다.

4.3 전송 메시지 처리시간 비교 실험

본 실험은 통신 처리시간에 관한 실험으로 표 6과 같이 현재 온도 값 읽기는 500msec 간격, 진공값 읽기는 50msec 간격, 인버터의 공기 속도(Air speed) 읽기는 100msec 간격으로 약 5초 동안 160개의 서비스 요청 메시지를 SCU에게 전송하도록 프로그램을 작성하여 실험하였다.

그림 7은 동시적인 다중의 서비스 요청 메시지 전송 시 처리 시간에 대한 결과를 나타내는 것으로, 실험은 총 30회에 걸쳐 반복 실험을 하여 평균 처리 시간을 구했다.

표 6. 처리시간 실험 조건
Table 6. Processing time experiment conditions

Service Message	Send Time Interval
Read current PV(Thermo) value	500 msec
Read current Vacuum value	50 msec
Read current Air speed value	100 msec
SCU Response time	about 50 msec

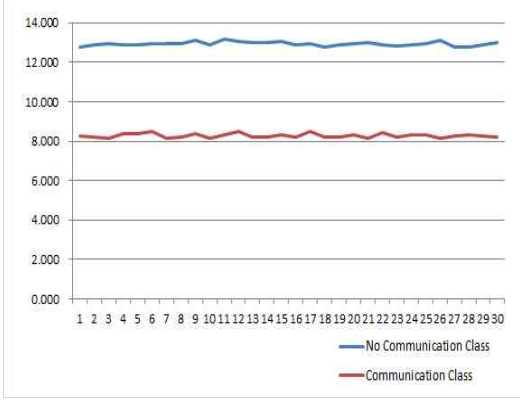


그림 7. 처리 시간 결과
Fig. 7. Processing time result

통신 클래스를 적용하지 않은 경우 평균 처리 시간은 12.913sec가 소요되었으며, 통신 클래스 적용한 경우 평균 처리 시간은 8.279sec가 소요되어 제안한 통신 클래스를 적용할 경우 약 56% 정도의 처리속도가 향상되었음을 확인하였다. 이는 제안한 통신 클래스 내부에서 전송 메시지를 관리하는 ICDT를 이용하면 응답한 SCU의 주소 및 서비스 유형을 식별할 수 있기 때문에 이전에 전송한 메시지에 대한 응답이 없는 경우에도 다음 메시지를 전송할 수 있기 때문이다.

V. 결 론

기존 모드버스 프로토콜은 응답한 메시지가 어떤 서비스 요청에 대한 응답인지를 식별할 수 없어 하나의 서비스 요청이 처리된 후 다음 서비스 요청 메시지를 전송할 수 있다.

본 논문에서는 이런 문제를 해결하기 위해 통신 클래스 내부에 인덱스 코드 정의 표를 작성하여 전송 메시지를 관리할 수 있는 모드버스 통신 클래스를 제안하였다. 통신 클래스 내부에 정의된 인덱스 코드 정의 표의 인덱스 코드는 메시지의 전송 순서를 파악하기 위한 코드로 이를 이용하여 이전 전송한 서비스 요청 메시지가 처리되지 않은 상태에서도 또 다른 서비스 요청 메시지를 전송할 수 있도록 하였다. 또한, SCU에서 응답한 메시지가 어떤 서비스에 대한 응답 메시지인지 식별할 수 있도록 하였다.

실험 결과, 제안한 통신 클래스를 적용하면 하나의 서비스 요청 메시지 전송 후 응답 메시지 처리가 완료되지 않은 상태에서 다음 서비스 요청 메시지의 전송이 가능하였으며, 에러 발생 시 인덱스 코드 정의 표를 이용하여 에러가 발생한 SCU 위치를 확인할 수 있었다. 또한, 동시적인 다중의 서비스 요청 메시지 전송 처리시간에 관한 실험에서는 제안한 통신 클래스를 적용하면 약 56% 정도의 처리시간이 향상된 것을 확인하였다.

향후 연구에서는 산업용 장비의 스마트 제어를 위한 다중 IoT 장치 연결 시 취약한 보안 문제를 해결하기 위한 경량화 인증 방식 및 메시지 암호화 방법에 관한 연구를 수행하고자 한다.

References

[1] Seung-Hyeok Shin, "Study On Lightweight IoT Sensor Gateway Using Open Source Hardware", Journal of the KIIT, Vol. 13, No. 10, pp. 85-90, Oct. 2016.

[2] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite, "The Future of Industrial Communi cation: Automation Networks in the Era of the Internet of Things and Industry 4.0", IEEE Industrial Electronics Magazine Vol. 11, No. 1, pp. 17-27, Mar. 2017.

[3] DaeHeon Park, "A Study on Integrated Operation Systems of Plant Factory based on Internet of Things", Ph.D Thesis, SunChon University, 2015.

[4] Bo-Hun Kim, "A Design and Implementation of PC-based Control System for Efficiency

Improvement of Industrial Equipments", Ph.D Thesis, KongJu University, 2017.

[5] Jeyasingam Nivethan and Mauricio Papa, "A SCADA Intrusion Detection Framework that Incorporates Process Semantics", ACM CISRC '16 Proceedings of the 11th Annual Cyber and Information Security Research Conference, Article No. 6, DOI:10.1145/2897795.2897814, Apr. 2016.

[6] Saptarshi Naskar, Krishnendu Basuli, and Samar Sen Sarma, "Serial port data communication using Modbus protocol", Journal of ACM Ubiquity, Vol. 2008, No. 1, p. 1, Jan. 2008.

[7] Jae-gu Song, Sungmo Jung, Seoksoo Kim, Taihoon Kim, Dong-Ju Kang, and SeokJu Kim, "Design of Hacking Test System for Modbus based SCADA", Journal of the KIIT, Vol. 7, No. 5, pp. 183-190, Oct. 2009.

[8] Bo-Heon Kim, Jeong-Ho Song, and Hwang-Rae Kim, "A Study on Enhancement of the MOD-BUS RTU Protocol for Multi-Device Connection", Proceedings of the KAIS Fall Conference, pp. 452-453, Dec. 2016.

[9] SamWon Tech Nova Communication Manual: http://www.samwontech.com/bbs/bbs/board.php?bo_table=data_04&wr_id=10 [Accessed: Jul. 21, 2017]

[10] Kisan System KM60xx Modbus Manual: http://www.kisansystem.kr/datasheet/KM60xx_Modbus_Modbus_Manual.pdf [Accessed: May. 10, 2017]

저자소개

김 동 환 (Dong-Hwan Kim)

2017년 2월 : 공주대학교 컴퓨터공학과(공학사)
2018년 2월 : 공주대학교 컴퓨터공학과(석사과정)
관심분야 : 정보 보호, 컴퓨터 네트워크, 네트워크 보안

김 보 헌 (Bo-Heon Kim)

2013년 8월 : 공주대학교 IT공학과(공학석사)
2017년 8월 : 공주대학교 컴퓨터공학과(공학박사)
2015년 3월 ~ 현재 : ㈜이지에스 기술이사
관심분야 : 스마트 장비제어S/W, 컴퓨터네트워크 및 보안

송 정 호 (Jeong-Ho Song)

2001년 8월 : 아주대학교 대학원 컴퓨터공학과 공학석사
2016년 2월 : 공주대학교 대학원 컴퓨터공학과 박사수료
1999년 3월 ~ 현재 : 동일공업고등학교 컴퓨터미디어보안과 교사
관심분야 : NCS, 정보 보호, 컴퓨터 네트워크, 네트워크 보안, 공업 교육

김 황 래 (Hwang-Rae Kim)

1991년 2월 : 중앙대학교 컴퓨터공학과(공학석사)
1994년 2월 : 중앙대학교 컴퓨터공학과(공학박사)
1983년 3월 ~ 1994년 2월 : 한국전자통신연구원 책임연구원
1994년 3월 ~ 현재 : 공주대학교 컴퓨터공학부 교수
관심분야 : 정보보안, 컴퓨터네트워크, 네트워크 생존성 관리, NCS