SMIC

Embedded System Development

기업체 교육 안내

교육과정	일자	교육인원	방문교육
1. Embedded –C Programming	2일	20명까지	
2. Device Communication	2일	20명까지	실습장비제공
3. Network Programming (TCP/IP Socket)	2일	20명까지	

교육 견적 담당자: 김한규

전화번호: 010-4254-7104

SM Information and Communication

Homepage : http://www.smic21.com
Support : +82-2-6292-2100 khg@smic.co.kr, smic21@smic21.com
Address : 서울시 금천구 가산동 SK트윈테크 타워A-401
우편번호 153-023

교육과정: Embedded-C Programming

교육과정개요:

Embedded C Programming개발 필요한 기본지식과 C-Code Programming 기법 습득하기 위한 과정. Embedded System의 기본개념과 개발과정, 교차개발환경, 프로세스, 컴파일러, 커널에 대한 이해와 일반 C-programming 과 Embedded 개발시 C-Programming의 차이점 및 C-Programming기법, 최적화 기법 등에 대하여 이해하는 강의와 실습과정입니다.(이론 40%, 실습:60%)
실습환경: VMware, Fedora12, 실습코드제공

교육내용:

- Embedded System 기본개념
- Embedded System 용어 및 개발과정.
- ARM Processor와 Embedded System
- Compile과정의 이해 (Flash, RAM)
- Embedded Linux 개발환경 구축 및 실습
- C-Language기본
- 변수 속성
- Bit Manipulation
- Endian , Pointer와 Array
- Memory Allocation(malloc)
- Floating Point 구조 이해
- 최적화 기법
- Linux Embedded Target 구축 및 Firmware Download 실습

기대효과:

- Embedded System 기본개념과 개발 환경에 대한 이해
- Embedded Program 개발과 일반 PC Program과의 차이점 이해
- Embedded System구조와 기본 용어 개발 과정에 대한 기대
- C-언어 기본과 H/W bit 제어, Pointer, Floating Point등 구현기법습득

교육과정: Device Communication 과정

교육과정개요:

일반적으로 많이 사용되는 Device 들 간의 통신을 가능하게 하는 프로그램개발능력을 확보하는 과정. RS232, RS422, RS485, Ethernet, Digital Input, Digital M Output, 등 디바이스 통신 종류 및 odbus 프로토콜 에 대한 이해 socket, serial 통신을 통한 P Modbus 프로토콜 C-Language Based rogramming 실습

교육내용:

- Data Communication Concepts
- 현장의 Device Communication 종류
- RS232, RS485, RS422 Ethernet, WiFi
- Digital Input/Digital Output
- Modbus Protocol Concept
- Modbus Map Concept
- Modbus Protocol Programming
- Modbus Protocol를 통한 DI/DO/PI 제어

기대효과:

- Data Communication Concept
- Device Communication Concept.
- Modbus Protocol Concept
- Modbus Map Concept
- Modbus Protocol Programming 기법

교육과정: C언어기반- Network Programming과정

교육과정개요:

TCP/IP concept, Protocol Architecture (TCP/UDP/ICMP), Address Schematics, Setup and verification, Routing, Service등 TCP/IP관련 개념과 Socket Programming 개발 방법 sample제공 실습

교육내용:

- **TCP/IP Concept**
- **TCP/UDP/ICMP 등 Protocol 해석**
- **Routing Concept**
- **TCP/IP Administration**
- **TCP/IP Service (NFS, FTP)**
- **Socket Programming Feature**
- **Server/Client/Broadcast/FD_SET/select Message spilt 처리**
- **IPC Concept – Message IPC, Shared Memory, Semaphore**
- **IPC Programming with Socket Programming**

기대효과:

- **TCP/IP Concept 확보**
- **TCP/IP 구조에 대한 이해 확보**
- **Administration and Service 기능에 대한 이해 확보**
- **Socket Program Application 구현 기능 확보**
- **Network & Communication Programming 이해**

Embedded System Development

Device Communication 과정

SM Information and Communication

Homepage : http://www.smic21.com
Support Address : +82-2-6292-2100 khg@smic.co.kr, smic21@smic21.com
: 서울시 금천구 가산동 SK트윈테크 타워 A-401
우편번호 153-023

권리사항

문의처

㈜신명정보통신 솔루션사업부

주 소 : 서울시 금천구 가산동 345-9 SK트윈테크타워 A-401

대표번호 : (02)6292-2100

담당자전화: 김한규 010-4254-7104

팩스번호 : (02)6292-2108

문의메일 : khg@smic21.com

Contents

Contents

Contents

Contents

Contents

Contents

1. Data Communication

1.1. Data Communication

- 정의 : Sender와 Receiver 인 2개의 Agent가 전송매체를 통하여 Digital Data를 Message라는 Frame을 만들어 교환(Exchange)하는 것.

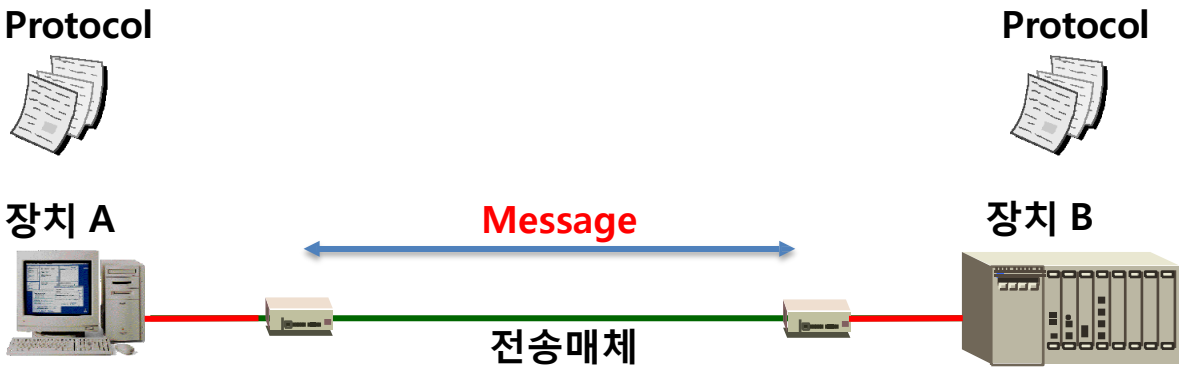
- Data and Information
  - Data : Data Communication을 위하여 일정한 형태를 갖춘 Bits들의 Stream (Raw Data).
  - Message: Data로 이루어진 Frame (Format을 갖고 있음).
  - Information: Raw Data를 가공한(인간에게 의미가 있는) 실용성 있는 Data의 모음.
- Data Communication Type
  - Local : 환경이 같은 Local 영역 안에서 일어나는 Data Communication (동일한 구조).
  - Remote : 모뎀, 혹은 외부 망을 통하여 원격지 사이에서 일어나는 Data Communication.

1. Data Communication

2. Data Communication 구성요소

- **Message** : 전송되어지는 Information의 **Data Frame**.
- **Sender** : 송신장치(**Device**)
- **Receiver** : 수신장치(**Device**)
- **Medium** : 전송매체 (**Transmission Medium**) Ethernet(UTP/STP), 전용선, Fiber 무선전파…
- **Protocol** : 송신장치와 수신장치 사이에 데이터를 교환하기 위한 **통신절차 (규약 : Rule)**.

  Frame으로 구현



2. Analog Data and Digital Data

1. Analog Data and Digital Data

- **Analog Data :**
  - 연속적인 신호로서 전압, 전류, 주파수 (온도, 압력, 습도, 진동, 산도)등.
  - 전송매체를 통하여 복수전송이 가능하다.
  - 신호감쇄현상이 덜하다.
  - 전자파를 발생시킨다.
  - Modem, 전용선 등.
- **Digital Data :**
  - 이산적인 신호 1 혹은 0의 디지털 신호.
  - 전송매체 한 신호 선을 통하여 한번에 한 신호만 전송 할 수 있다.
  - 장비의 구조가 간단하고 그에 따라 비용이 덜 든다.
  - 거리에 따라 신호감쇄현상이 발생한다.

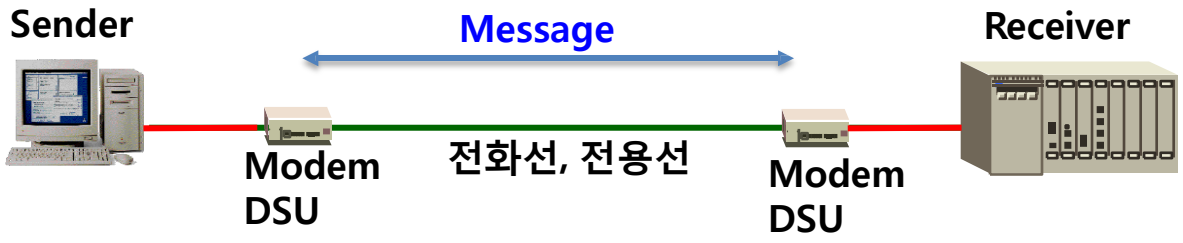
2. Analog Data and Digital Data

2. 변조와 복조 (Modulation and Demodulation)

- **변조(Modulation) :**
  - Digital 신호를 Analog신호로 바꾸어 주는 것.
- **복조(Demodulation):**
  - Analog 신호를 Digital 신호로 바꾸어 주는 것.

2.3. 변복조 장치

- Modem (Modulator Demodulator) Analog 신호의 변복조 장치 (기존 음성 전화선이용)
- DSU(Digital Service Unit) Digital 신호의 변복조 장치 (디지털 신호를 증폭하는 방식)



3. Transmission Media

3.1. 전송매체

- Twist Pair (UTP/STP/FTP)
- 미국 EIA/TIA568 UTP 규격
- UTP(Unshielded Twisted Pair) - 무 차폐 이중 와선(꼬인선)으로 된 케이블 종류
- STP (Shield Twisted Pair) – 피복 안쪽과 내선 차폐가 된 Twisted Cable, 고가격
- FTP (Foil Screened Twisted Pair) – 피복 안쪽에만 은박으로 차폐를 함.
- Coaxial Cable - 동축케이블
- Fiber Optic Cable - 광케이블

UTP케이블

UTP케이블용 플러그,

FTP케이블

FTP케이블용 플러그

3. Transmission Media

3.2. UTP Cable Category

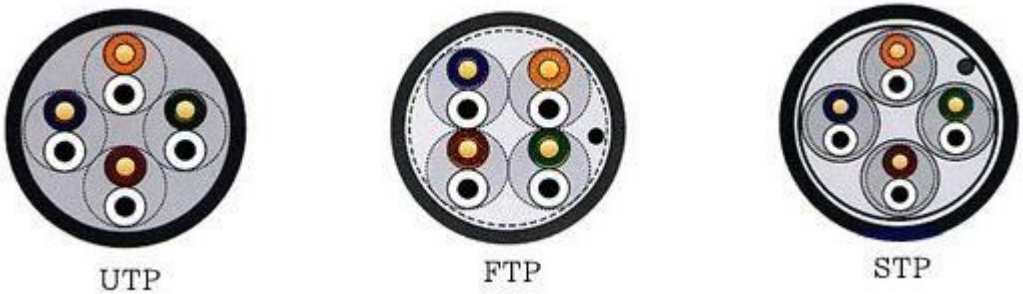
등급	전송대역	사용네트워크	용도
Cat-1/2	1~4MHz	-	음성전화망
Cat-3	16MHz	10Base-T, 100Base-T	전화망/Data방
Cat-4	20MHz	Token Ring	현재는 사용 안됨
Cat-5/5e	100MHz	100Base-T 155Mbps 1000Base-T	고속전화/Data망
Cat-6	250MHz	1000Base-T, 10GBase-T	초고속전화/Data망
Cat-7	600MHz	1000Base-T, 10GBase-T	초고속전화/Data망

- Category 가 높을 수록 더 높은 주파수에서도 적은 유전체 손실, 더 좋은 절연 더 많은 꼬임을 갖게 됨.
- Category 3, 5, 6이 많이 쓰임.

3. Transmission Media

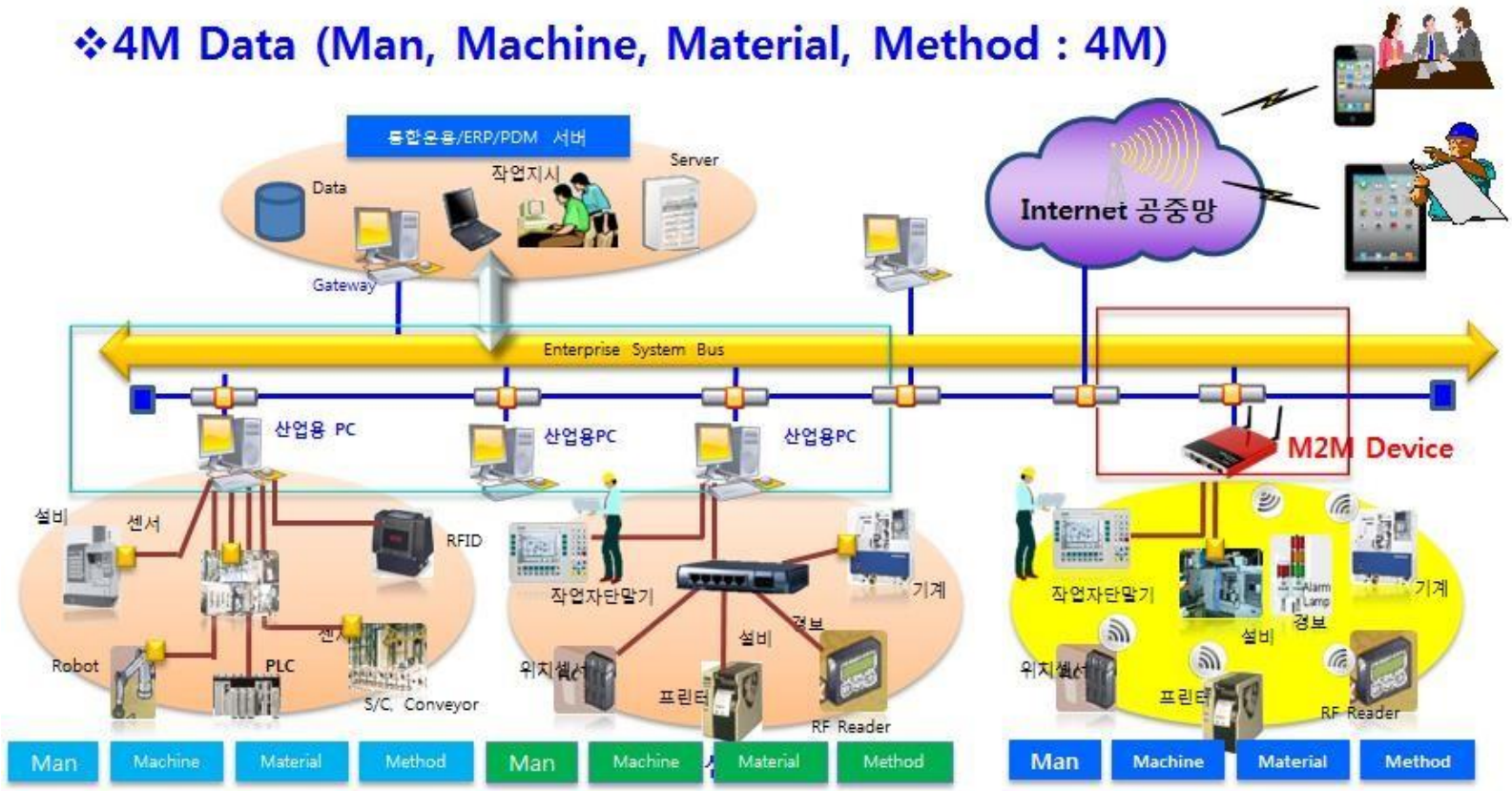
3.3. UTP/STP/FTP Cable

Cable	전송대역
UTP	절연된 구리 선이 전자기 유도를 줄이기 위하여 꼬여져 있는 케이블 사무실 배선용
FTP	알루미늄 은박이 피복 안쪽에 내선을 한꺼번에 감싸고 있는 케이블 절연기능이 좋아 공장 배선용으로 사용
STP	외부피복 아래 접지 역할을 하는 차폐제가 있고 내선의 모든 선들도 각각 차폐시킨 케이블 외부 Noise 차단 , 전기적 신호의 간섭에 탁월한 성능



4. Device Communication

4.1. Raw Data Source (Sensor Node, M2M Device, Middleware)



4. Device Communication

4.2. 현장의 데이터 분류 4M Data(부록참조)

Machine	Material	Man	Method
PLC Relay, 스위치 카운터,위치 성형기 공작기계 센서,설비	자재 가공품 반제품 불량품	작업자 단말 불량내용보고 고장내용정보 작업지시정보	작업방식 표준작업 시험방법 검사방법

Data, DI/DO/AI/AO, Counter, PLC, 설비Controller와의 통신

Shop Floor


Serial


Sensor


PLC


설비


로봇


PLC

4. Device Communication

4.3. Device Communication의 종류

- Ethernet (IEEE 802.3)을 가진 장비
- RS232/422/485 Serial 통신 장비
- Field Bus 통신(DCS, 공작기계 등 설비산업 분야)
- Wireless Zigbee (IEEE802.15.4)
- Wireless WiFi (IEEE802.11b/g/n :150Mbps~600Mbps)
- RF-ID 통신
- Can 통신 (자동차 분야)
- MT connect (공작기계 분야)
- SECS 반도체 분야
- Digital Input, Digital Output, Pulse Input, Analog Input, Analog Output.
- SPI, I2C – On board Chip 과 Chip 사이의 통신

4. Device Communication

4.4. Device Communication 구성요건

- 통신을 하려는 2개의 Device가 존재함.
- 전송매체
- Device에 맞는 Communication Protocol
- Standard Data Representation (Data 형태 5,6,7,8 bit)
- Serial or Parallel 등 통신 방법.
- Bit synchronization using Start/stop bits in case of Asynchronous Transmission
- In Synchronous Transmission the agreed pattern of Flag
- Signal encoding rules
- Other Higher Protocol

4. Device Communication

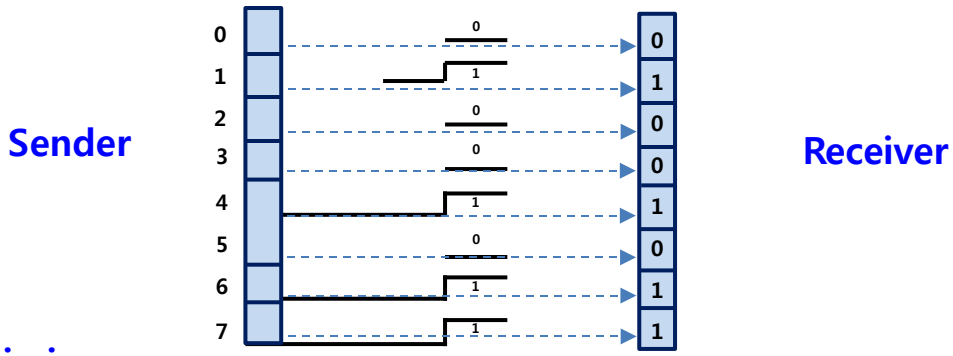
4.5. Data

- Data Representation – bit 의 크기가 지원하는 character, number, control character를 나타냄.
  - 5 bit code -32 symbols
  - 7 bit code -128 symbols
  - 8 bit code – 256 symbols
- ASCII Code Set – ANSI 7-bit American Standard Code
  - 8th bit – Parity bit for Error detecting (MSB : Most Significant bit)
  - 94 Codes – 화면에 표시 가능. Space 와 DEL character는 화면에 표시되지 않음.
  - 32 Codes – Control Character, STX, ETX, EOT, CR, LF … 화면에 표시되지 않음.
- EBCDIC Code Set – IBM의 8bit Extended Binary Coded Decimal Interchange code
  - 256개의 symbol을 나타냄, Parity 없음.
  - 화면에 보여지는 character는 ASCII와 같음.
  - Control Character ASCII와 다름.

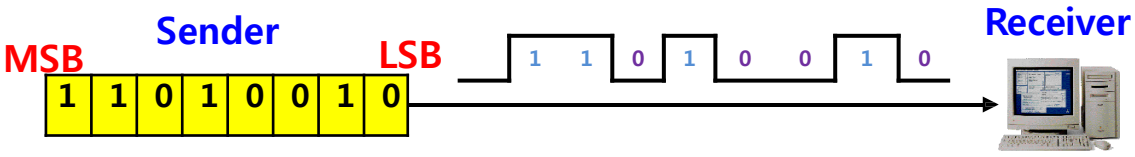
4. Device Communication

4.6. Data Transmission(Parallel/Serial)

- **Parallel Transmission**
  - 데이터 bit 들이 모두 서로 다른 라인을 통하여 동시에 전송됨.
  - 전송거리가 길지 않은 컴퓨터의 프린터 및 GPIB(계측기), 고속의 외장기기 등 주변기기에 사용됨.

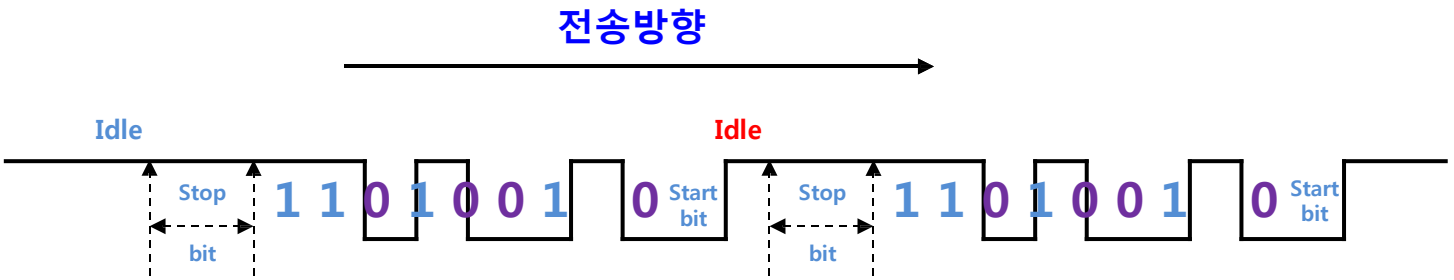


- **Serial Transmission**
  - 데이터 bit 들이 한 신호 라인을 통하여 순차적으로 전송됨.
  - LSB(Least Significant Bit)가 먼저 전송됨.
  - 거리가 먼 경우의 데이터 전송에 적합함.



4.7. Data Transmission(Async/Sync) 1

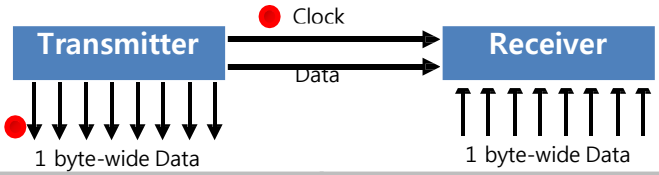
- **Asynchronous Transmission**
  - Start-Stop Synchronization
  - 임의시간에 한꺼번에 Data가 전송된다.



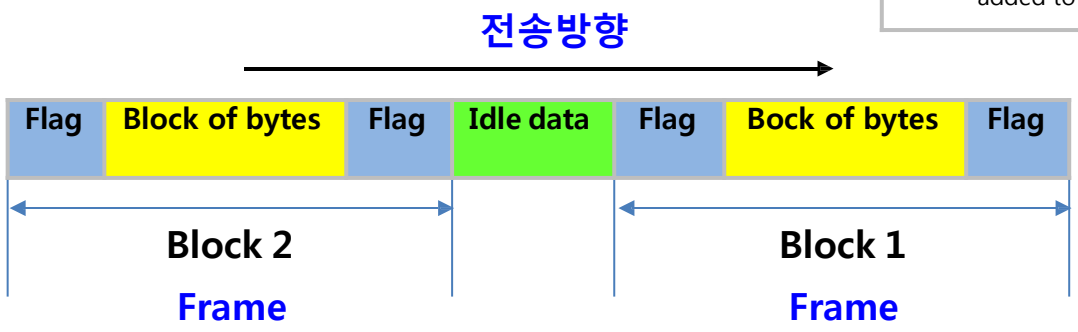
  - Start bit is always 1 bit duration
  - Start bit is always equal to '0' Low신호
  - Stop bit may be 1 or 1.5 or 2 bits duration
  - Stop bit is always equal to '1' High신호
  - Idle period time is arbitrary (variable) '1' 이 보내진다.
  - RS232, 422, 485 동일

4.7. Data Transmission(Async/Sync) 2

- **Synchronous Transmission**
  - Flag가 전송 Block의 Start 와 End 를 식별함.(Preamble)
  - 수신 측은 항상 제일 먼저 Flag부터 식별함.
  - Flag는 고정된 Data Pattern임, Header and Tailor.
  - Frame은 Flag를 포함한 Data Block임.
  - 별도의 Clock 신호 선이 있음
  - 시간에 동기 되어 수행됨.
  - X.25, HDLC.



Transmitter	Receiver
클락에 동기되어 데이터가 전송된다.	Extracts the data using the **clock provided by the transmitter**
the clock 신호를 위한 별도의 **signal line이 있다**	시리얼데이터를 패러렐 데이터로 변환한다.
No start, stop, or parity bits added to data	



4.8. Field Bus

- **Field Bus**

▪PLC, DCS, 공작기계 또는 제어용 PC·HMI(Human Machine Interface) 상호의 정보교환에 이용 되는 네트워크(Real Time 지원)

▪필드버스(Field bus)의 시작은 1980년대 초에 GM(General Motors) 사가 주도한 MAP (Manufacturing Automation Protocol)

- 시스템이 복잡 고액이 되기 때문에 상업적으로는 성공하고 있지 않음.
- 국제 규격화의 움직임은 1985경부터 시작됨
- 2000년에 이르러 간신히 국제 규격 IEC61158, IEC62026가 제정되었음.
- IEC 규격에서는 3층의 네트워크 모델에 대해 정의하고 있습니다

4. Device Communication

4.9. Field Bus 종류

- **Factory Automation**
  - Device Net — Allen Bradley
  - PROFIBUS-DP — 고속 대용량
  - CC-Link — PLC, 미쓰비시 전제품의 지원 통신 사양
  - Interbus — 독일 phoenix Contact
  - Ethernet/IP — 산업용 Ethernet (실시간 지원)
  - MTconnector — 공작기계 전용 Protocol 지원 하드웨어 소프트웨어
- **Process Automation**
  - PROFIBUS-PA — 2 wires, 방폭
  - Foundation Fieldbus
- **Control Bus**
  - Control Net — PLC, DCS, HMI 간 communication
  - CANopen — 구미(USA)
  - FL-Net — 일본 전기공업회
  - Modbus RTU/TCP — 미국 RS485
  - Modbus Plus — 1Mbps 광섬유
- **Sensor Bus**
  - CAN — GM 자동차 생산 현장
  - EC-Net

4. Device Communication

4.9. Field Bus 종류

●EtherCAT What

is EtherCAT

EtherCAT is **a real-time Ethernet Master/Slave network** developed by Beckhoff. Today, it is an open standard, managed by the EtherCAT technology group. EtherCAT sets new limits for real-time perfor mance. Regarding topology, EtherCAT supports a simple low cost line structure, a tree structure, dais y chaining or drop lines - no expensive infrastructure components are required.

●CANOpen What

is CANopen?

CANopen is a very popular industrial communication network originally designed for **motion-oriented machine control networks**, such as handling systems. It is used in many industries, such as **medical equipment, off-road vehicles, maritime electronics, public transportation and build ing automation**. CANopen is maintained by the CIA (CAN-in-Automation) international user and manufacturers group and is standardized in the European standard EN 50325-4.

4.9. Field Bus 종류

●Modbus TCP, Modbus RTU

What is Modbus TCP?

Modbus is an open **Master/Slave application protocol** that can be used on several different physical layers. Modbus-TCP means that the Modbus protocol is used on top of Ethernet-TCP/IP. It is an open Industrial Ethernet network which has been specified by the Modbus-IDA User Organization in co-operation with the Internet Engineering Task Force (IETF) as an RFC Internet standard.

What is Modbus RTU?

Modbus-RTU (Remote Terminal Unit) means that the Modbus protocol is used on top of a **serial line with an RS-232, RS-485 or similar physical interface**. Numerous automation systems have Modbus-RTU interfaces for communication. Modbus devices are certified by the Modbus User Organization for interoperability and conformance to the Modbus specification

4.9. Field Bus 종류

- **DeviceNtet**

What is DeviceNet?

DeviceNet is a popular network for time-critical applications based on **Rockwell Automation technology.** DeviceNet offers robust, efficient data handling since it is based on a **Produce/Consume model**. It is standardized in the international standard series IEC 61158 and maintained by ODVA (Open DeviceNet Vendors Association).

- **CC-Link**

What is CC-Link,

CC-Link is a Fieldbus network that processes both **cyclic I/O data and acyclic parameter data at high speed**. CC-Link was developed by **Mitsubishi** and today, it is managed by the CC-Link Partner Association (CLPA). CC-Link is a very popular network in Asia.

What is CC-Link IE Field?

CC-Link IE Field network is the **first gigabit industrial Ethernet network extended down to the field device level.** It combines the best of existing technologies and applies them in a highly reliable architecture that provides exceptional data bandwidth and transaction rates.

4.9. Field Bus 종류

- **PROFINET**

What is PROFINET?

PROFINET is an open standard for Industrial Ethernet. The network was developed by **Siemens and the Profibus User Organization (PI).** PROFINET is used for factory and process automation, for **safety applications**, and for the entire range of drive technology right up to **clock-synchronized motion control**. PROFINET is used worldwide but is especially strong in **central Europe**.

- **PROFIBUS**

What is PROFIBUS

PROFIBUS is the world's most widely used fieldbus. This industrial network originates **from Siemens** and is today managed by the organization PROFIBUS and PROFINET International (PI). The network is especially strong in central Europe and have **approximately 40 million nodes installed around the world**

4.9. Field Bus 종류

●**BACnet What**

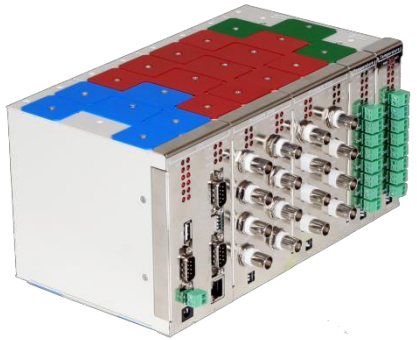
is BACnet?

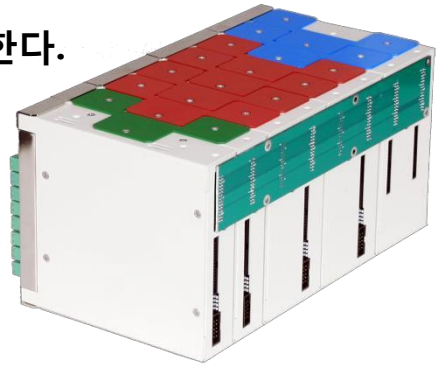
BACnet is a data communication protocol mainly used in **the building automation and HVAC industry (Heating Ventilation and Air-Conditioning).**
There are a couple of different **serial and Ethernet based-versions of BACnet**. The most common serial version is called **BACnet MS/TP** while the dominant Ethernet version is **BACnet/IP.**

10. SPI

- SPI (Serial Peripheral Interface) 개요

  - 모터로라가 처음 고안한 하드웨어 보드 안에서 MCU와 주변장치간에 Chip과 Chip 사이의 통신 방법. 단순하고 범용성이 높다.
  - 기본적으로 Serial 통신이다.
  - Full-duplex, 전송속도 : 20Mbps 이상 80Mbps
  - 용도: ADC, DAC, 통신chip과의 통신에 쓰임.
  - 외부 주변장치와 Clock을 통하여 bit 단위로 동기화 시키는 동기화 통신 방법.
  - 하나의 Master와 하나 또는 여러 개의 Slave Device간 통신을 한다.
  - GPIO와 Device Driver를 통하여도 통신 할 수 있다.
  - Embedded MCU는 SPI 기능을 내부에 갖고 있다. (ARM, Atmega)





10. SPI

- **SPI (Serial Peripheral Interface) 4개의 Signal Pin으로 통신한다.**

MISO (Master Input Slave Out)

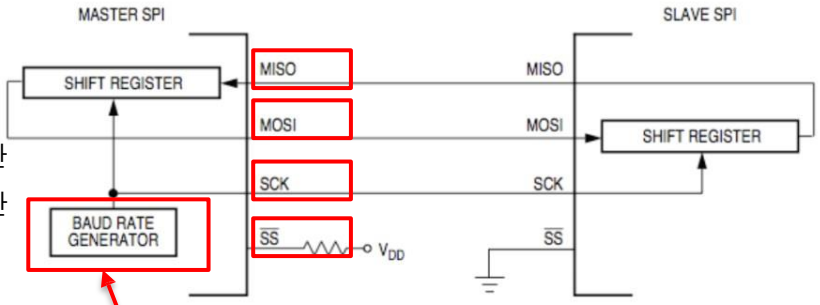
: 마찬가지로 Device 가 Master 로 설정되어 있는 경우 수신을 위한 Input Pin 으로 사용되며 , Slave 일 경우 출력을 휘한 Output Pin 으로 사용된다.

MOSI (Master Out Slave Input)

: Device 가 Master 로 설정되어 있는 경우 이 Pin 은 출력을 위한 Output pin으로 사용되며 Slave 로 설정되어 있다면 수신을 위한 Input Pin으로 사용된다.

SS (Slave Select)

: SPI 는 여러 Device 간 통신이 가능하다. 이때, 현재 통신을 하고 있는 Device 를 구분하기 위한 Pin 이다. SPI 는 동기화 방식이다. 외부 Device 간의 **Serial Data 의 동기화를 위한 Clock Pin** 이다.

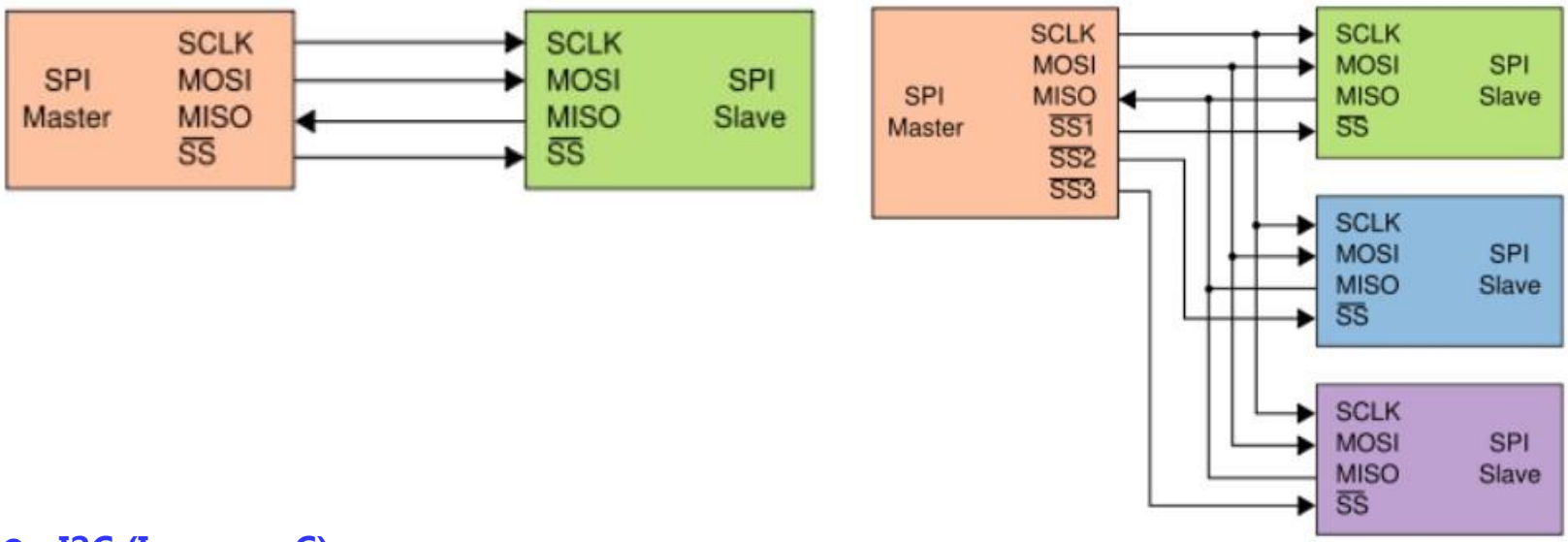


Master Baud Rate generator

SCK (Serial Clock)

: **Master가 제어권을 갖는다.**

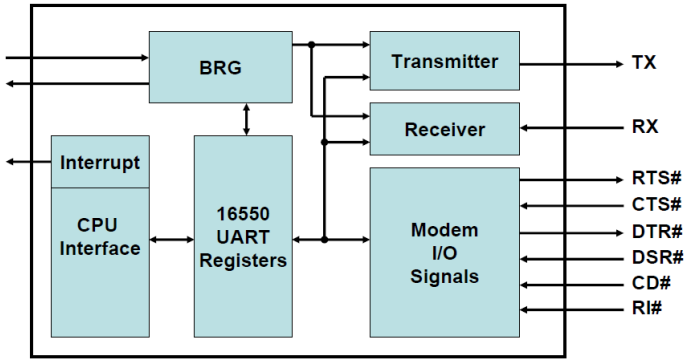
4.10. SPI



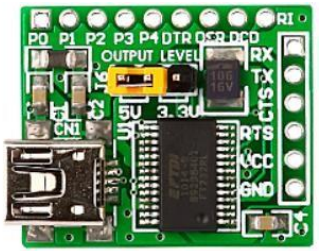
- I2C (I-square-C)
  - TV, VCR, Audio 장비 통신 을 위하여 필립스가 제안한 통신 방법
  - 2가닥 신호선(Clock and Data), 양방향 Serial 버스 규격
  - Half-duplex 3.4Mbps
  - EEPROM통신

4.11. UART

- Universal Asynchronous Receiver and Transmitter
  - Traditional Definition:
    Converts parallel (8-bit) data to serial data and vice versa
    - 범용 비 동기 송수신 장치
    - TTL level의 시리얼 통신
    - 칩과 칩 사이 통신
    - 또는 RS-232, 422, 485, USB 등으로 변환하여 다른 장비와 통신에 가장 많이 사용
    - Internal / External UART



16550 UART

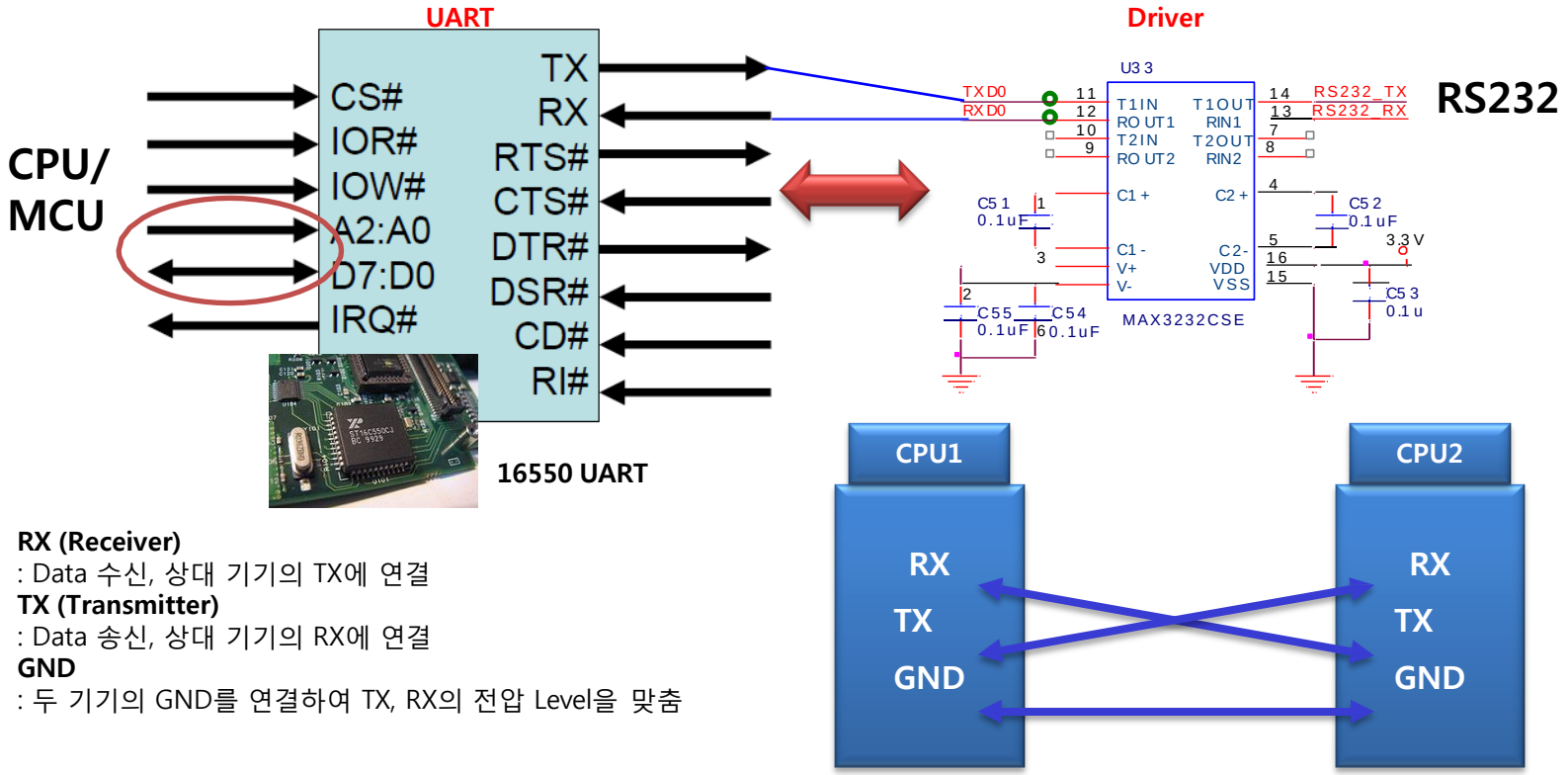


USB UART

4. Device Communication

4.11. UART

- **Universal Asynchronous Receiver and Transmitter**



RX (Receiver)
: Data 수신, 상대 기기의 TX에 연결
TX (Transmitter)
: Data 송신, 상대 기기의 RX에 연결
GND
: 두 기기의 GND를 연결하여 TX, RX의 전압 Level을 맞춤

The **MAX232** is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.

4.11. UART

- **Frame Formats**
  - **한번의 Data 송신 또는 수신 시 1 bit의 시작 bit로 시작하여 1~2 bit의 Stop bit로 끝**

1 bit Start bit
: Data 전송의 시작을 알리는 시작 bit
대기상태 high에서 low로 falling 되는 신호(값 0)

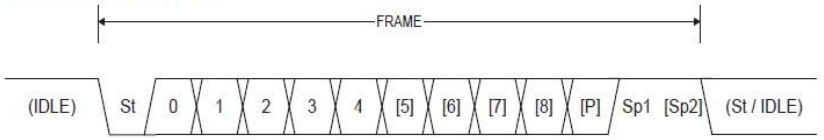
5~9 Data bit
: UART 통신은 데이터 비트의 크기가 다양
보통은 8bit 사용

0~1 Parity bit
: 패리티 검사를 하여 데이터가 잘 전달 되었는지 확인
홀수 또는 짝수 패리티가 있으며 각 비트의 합에 1 또는 0을 더해 계산

1~2 Stop bit
: Data 전송의 끝을 알리는 정지 bit
데이터 전송이 끝나면 마지막 high신호로 전송을 끝내고 대기상태 high를 유지 (값 1)

Baud Rate
: 2400 ~ 921600 bps로 통신속도의 범위가 넓다
주로 9600 ~ 115200 bps를 사용

Figure 82. Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

5. ISO OSI(Open System Interconnection) Reference Model

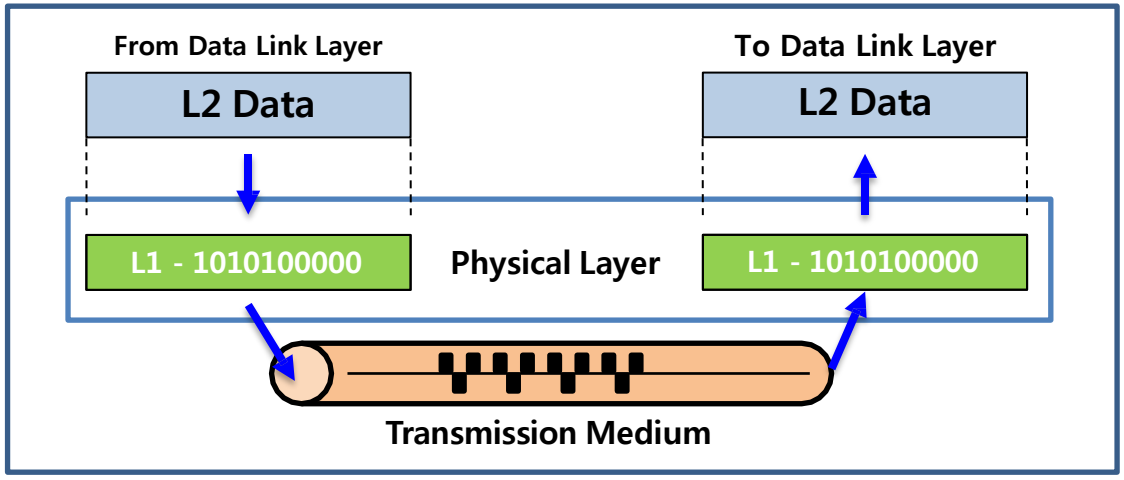
5.1. OSI Reference Model

- 국제표준화 기구 ISO(International Standards Organization) 가 개방형 통신을 할 수 있도록 만든 통신 모델
- OSI(Open System Interconnection) Reference Model
- 7 Layers
- Hierarchical Communication (계층구조)
- 각 Layer는 아래쪽 Layer를 Call 한다.
- 인접한 Layer들은 Software Interface로 Message를 교환한다.
- 각 Layer들은 고유의 Header를 갖는다.

5. ISO OSI(Open System Interconnection) Reference Model

5.2. Physical Layer 1

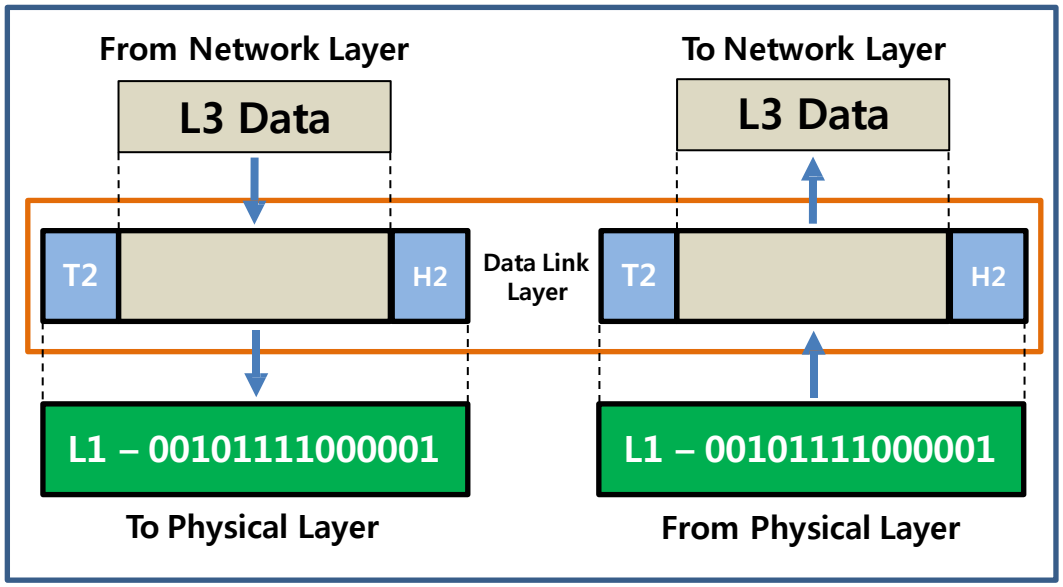
- 비트들이 보내질 수 있도록 물리적 링크를 설정, 유지 그리고 단절하기 위한 전기적 신호로 통신을 하는 전송매체에 관한 규정 (RS232C, X.21, UTP Cable, Optical, Coaxial, RF)
- Data Rate, Transmission Rate, Bit Duration, Clock Synchronization 정의
- Point to point, Multi-Drop, Mesh, Star, Ring
- Simplex, Half-duplex, Full duplex, 전압의 Level, Timing.
- 0과 1이 전기적 신호(High, Low)에 대하여 어떻게 Mapping 되는가.



5. ISO OSI(Open System Interconnection) Reference Model

5.3. Data Link Layer 2

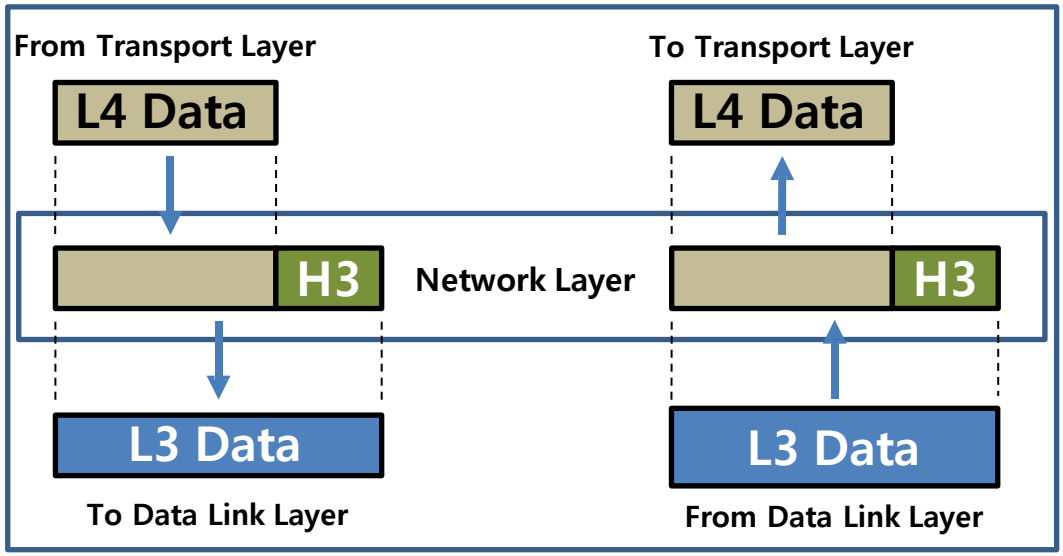
- Data Frame의 시작과 끝을 구분, 전송에러 검출, 전송Data의 겹침, 혼돈 방지.
- Link by Link Flow Control, CSMA/CD: Carrier Sense Multiple Access / Collision Detect
- Bit Stream을 Data Frame으로 만들거나 분해하고 Error Detect 및 재전송기능을 제공.
- Physical Addressing (ex: Mac Address)



5. ISO OSI(Open System Interconnection) Reference Model

5.4. Network Layer 3

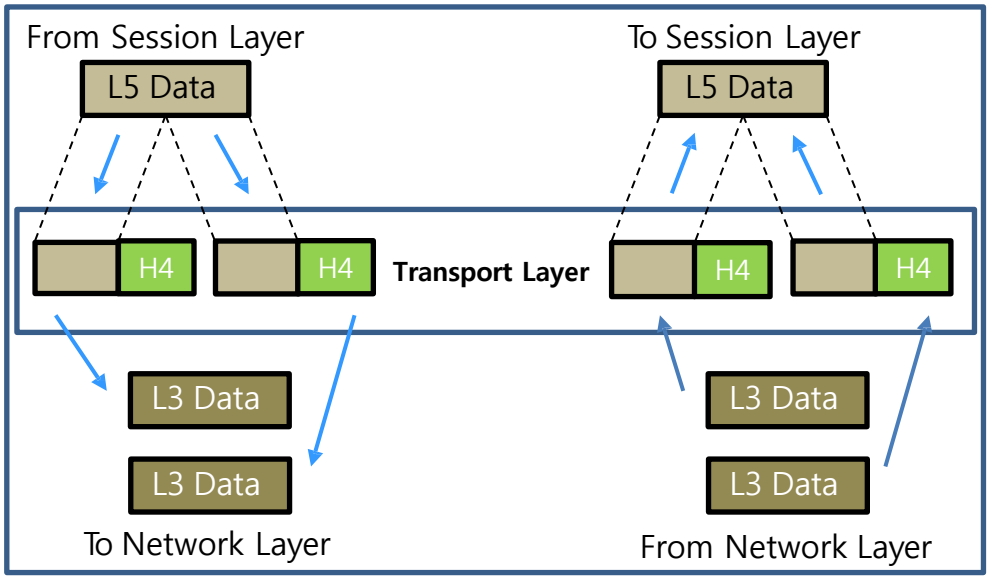
- 75년 이후에 규정된 이용자 기계와 Network들 사이의 Interface에 관계된 규정.
- Source Node 와 Destination Node 사이의 Data Packet 전송에 대한 규정 Lo
- gical Address (ex: IP Address) 관련된 규정
- Data Packet의 전송경로 (Routing 방법)에 대한 규정
- Header를 붙여 Data Frame의 형태로 Layer 2로 보낸다.



5. ISO OSI(Open System Interconnection) Reference Model

5.5. Transport Layer 4

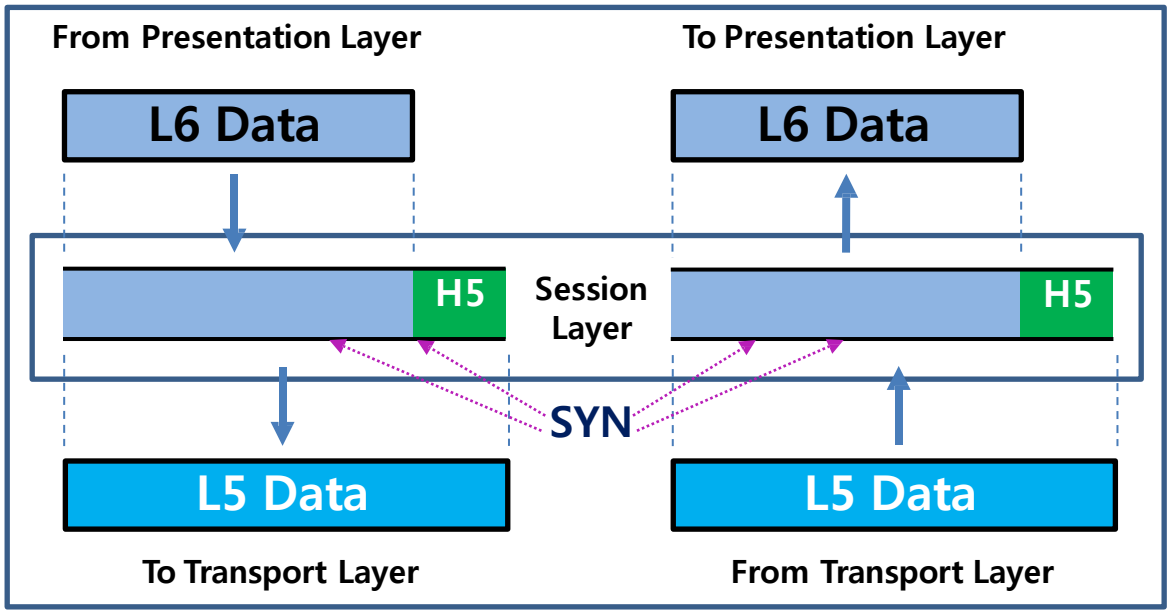
- End-to-End 상호작용, Data Packet Transaction의 분실이나 이중처리 방지, 이용자기계의 프로세스 주소선정.
- Data Packet으로 이루어진 Entire Message의 전달에 대한 규정.
- Virtual Circuit, Process 주소선정에 대한 규정 (ex: Port No)
- Error시 Packet의 재전송요구, Roll back, Message Reassembling 에 대한 규정.



5. ISO OSI(Open System Interconnection) Reference Model

5.6. Session Layer 5

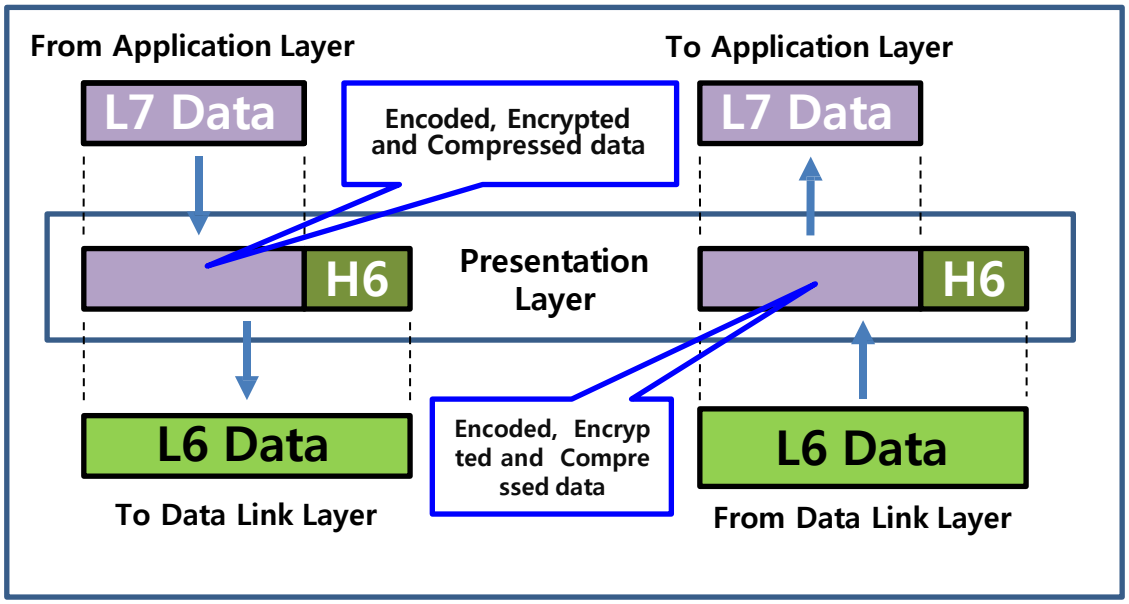
- 한 쌍의 프로세스들 사이에서 세션이라는 연결을 확립하고 유지 하는 것에 대한 규정.
- Process Blocking Non-Blocking.
- Data flow, Transport layer 회복.
- MSN의 Messenger의 출발지와 목적지간에 Session을 관리



5. ISO OSI(Open System Interconnection) Reference Model

5.7. Presentation Layer 6

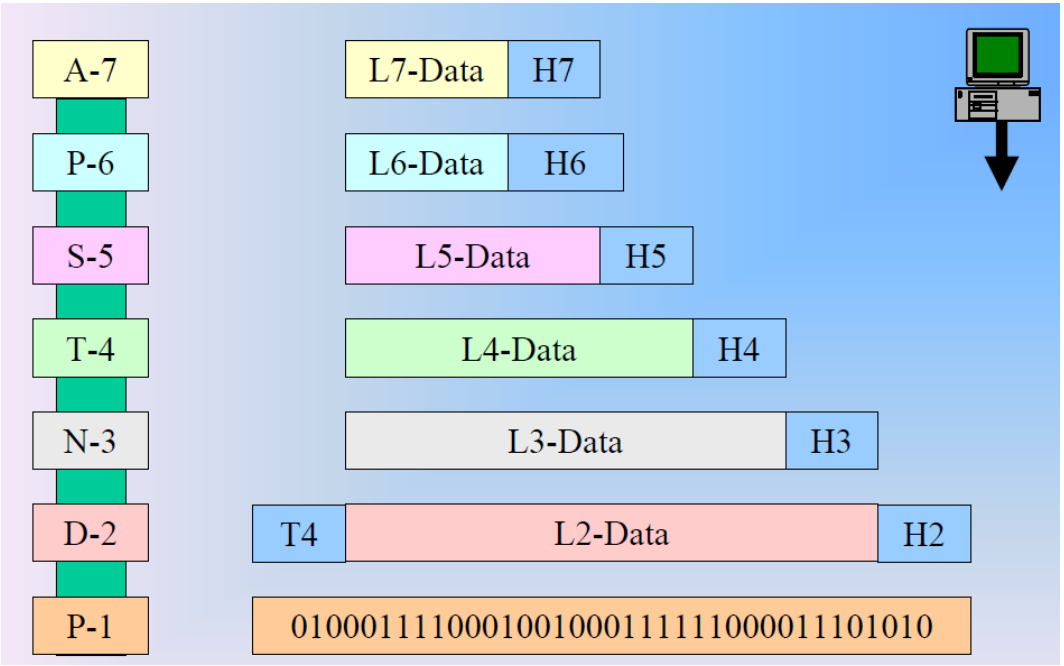
- Data가 세션계층에 보내지기 전에 일반적으로 유용한 형태로 변화 시키는 일을 수행 한다. (Sender와 Receiver사이에 Information의 Common Format으로 변경하는 규정.)
- Syntax and Semantics(구문과 의미)에 관한 규정
- Character Encoding, HTML, Encryption, Decryption, Compression



5. ISO OSI(Open System Interconnection) Reference Model

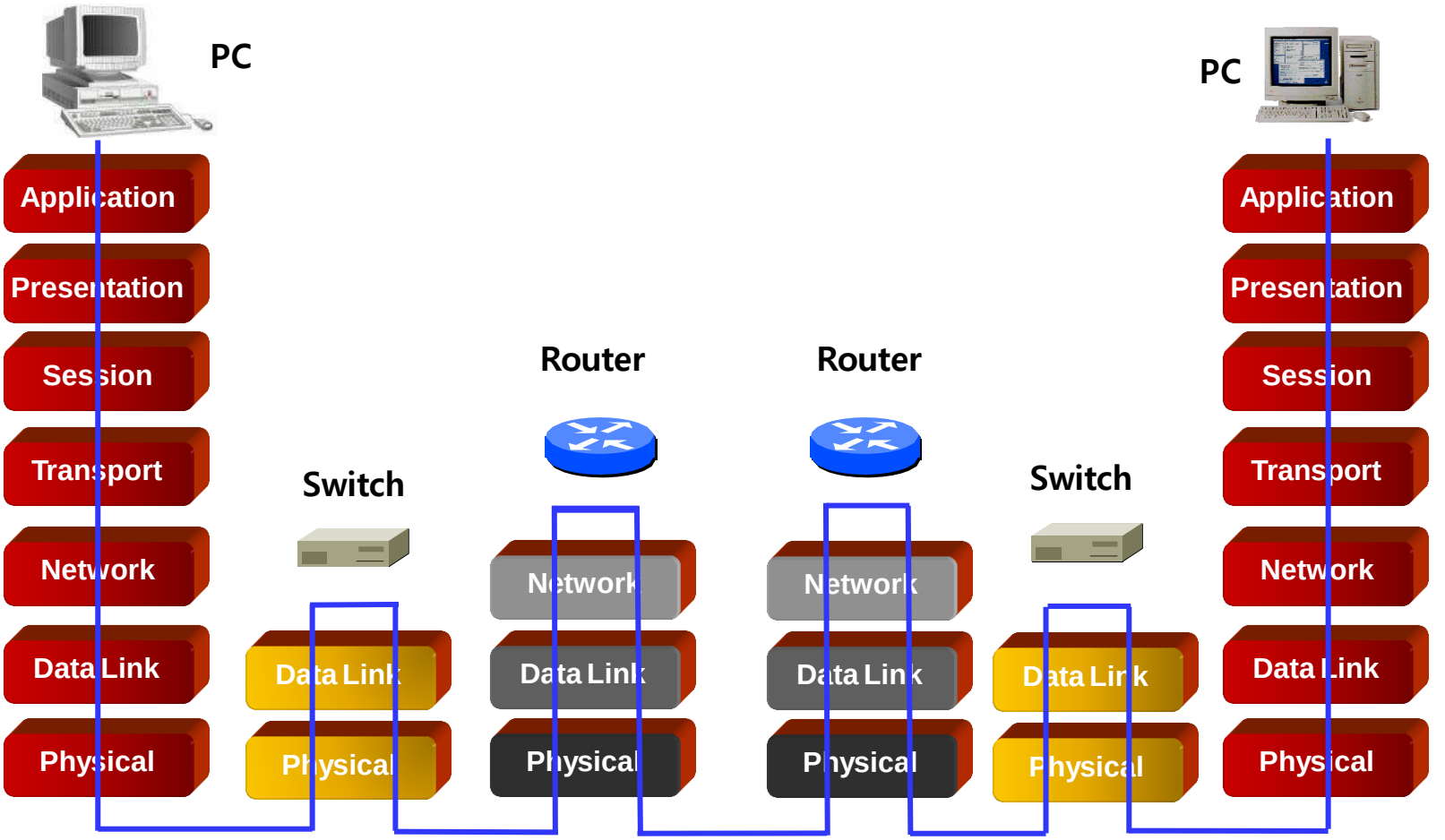
5.8. Application Layer 7

- Network User & Network Designer 사이의 경계로서 표준 없음.
- User Interface, Service를 제공하여 User나 Software가 Network에 접근하도록 함.
- FTP, Telnet, Mail service, 응용프로그램



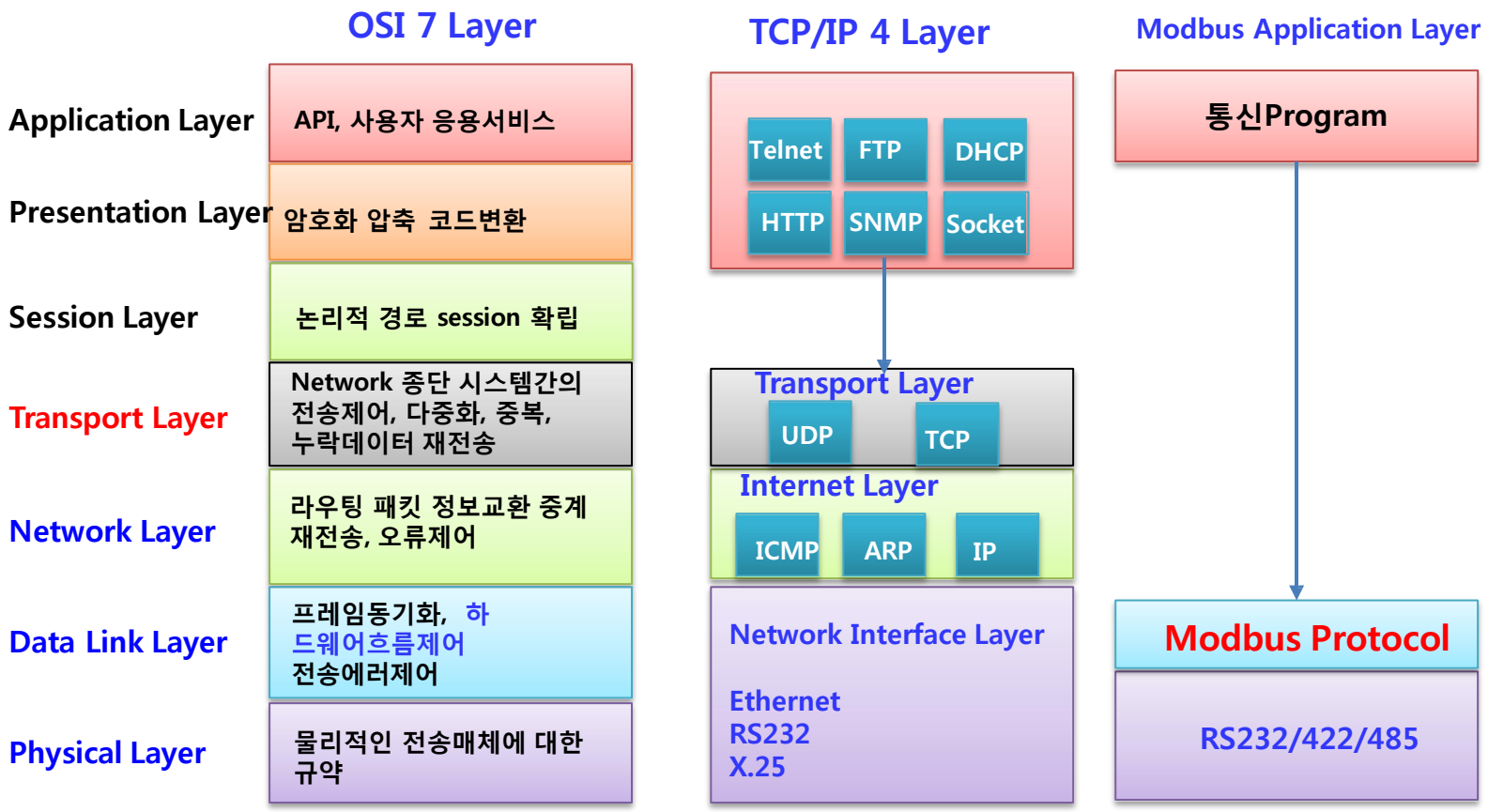
5. ISO OSI(Open System Interconnection) Reference Model

5.9. OSI 7 Layer Communication Stack



5. ISO OSI(Open System Interconnection) Reference Model

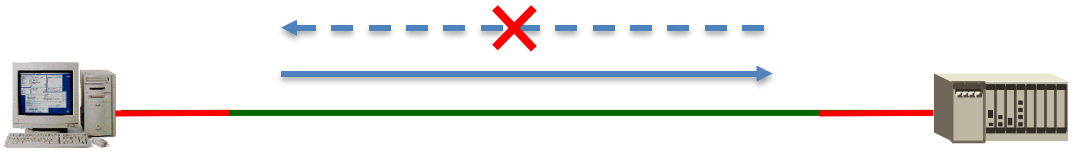
5.10. Communication Layer 비교



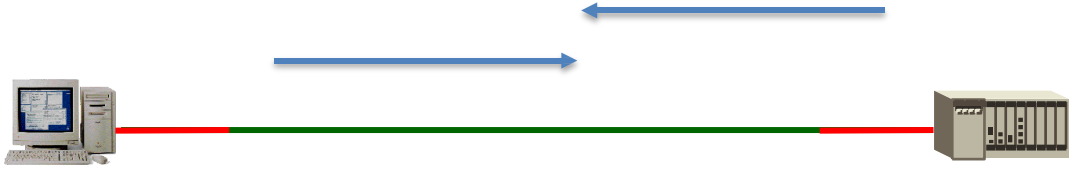
6. Serial Communication

6.1. Serial Communication Mode

- **Simplex :** 단 방향 통신



- **Half duplex :** 상호 단 방향 통신 : 송신과 수신을 교대로 하는 방식임. (Transmission Delay)

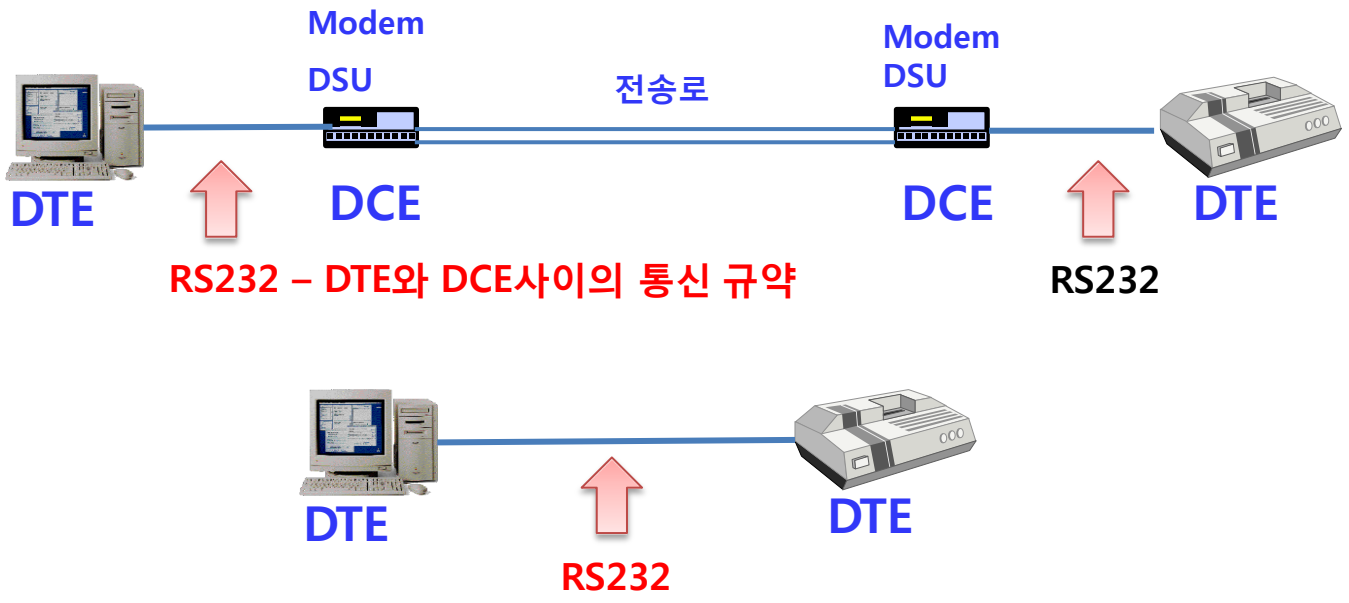


- **Full duplex :** 양방향 통신: 송신과 수신을 동시에 하는 방식임.

6. Serial Communication

6.2. DTE and DCE

- DTE – Data Termination Equipment (종단장치: PC 등 데이터사용장치)
- DCE- Data Circuit-Termination Equipment (모뎀 등 전송변복조장치)
- Modem – Analog 전송로의 변복조기
- DSU-Digital 전송로의 Digital Service Unit



6.3. Balanced and Unbalanced 1

- **Unbalanced : 비 평형 회로 (RS232)**

▪An unbalanced signal is represented by a single signal wire where a voltage level on that one wire is used to transmit/receive binary 1 and 0

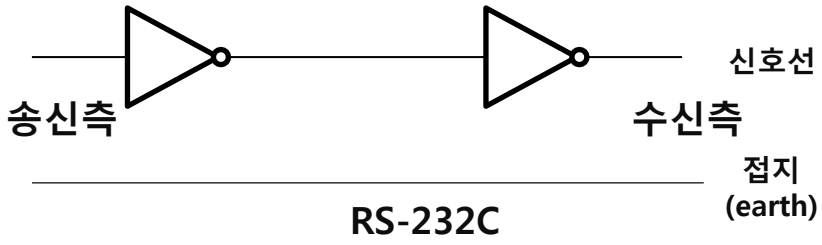
▪ **1 과 0 의 이진수를 송수신하기 위하여 하나의 신호 선을 사용하며 Ground를 기준으로 전압의 Level로서 1 과 0을 식별한다.**

송신 측

- 최대 출력 전압(무 부하) ± 25 V
- 최소 출력 전압(부하) ± 5 ~ ± 15V

수신 측

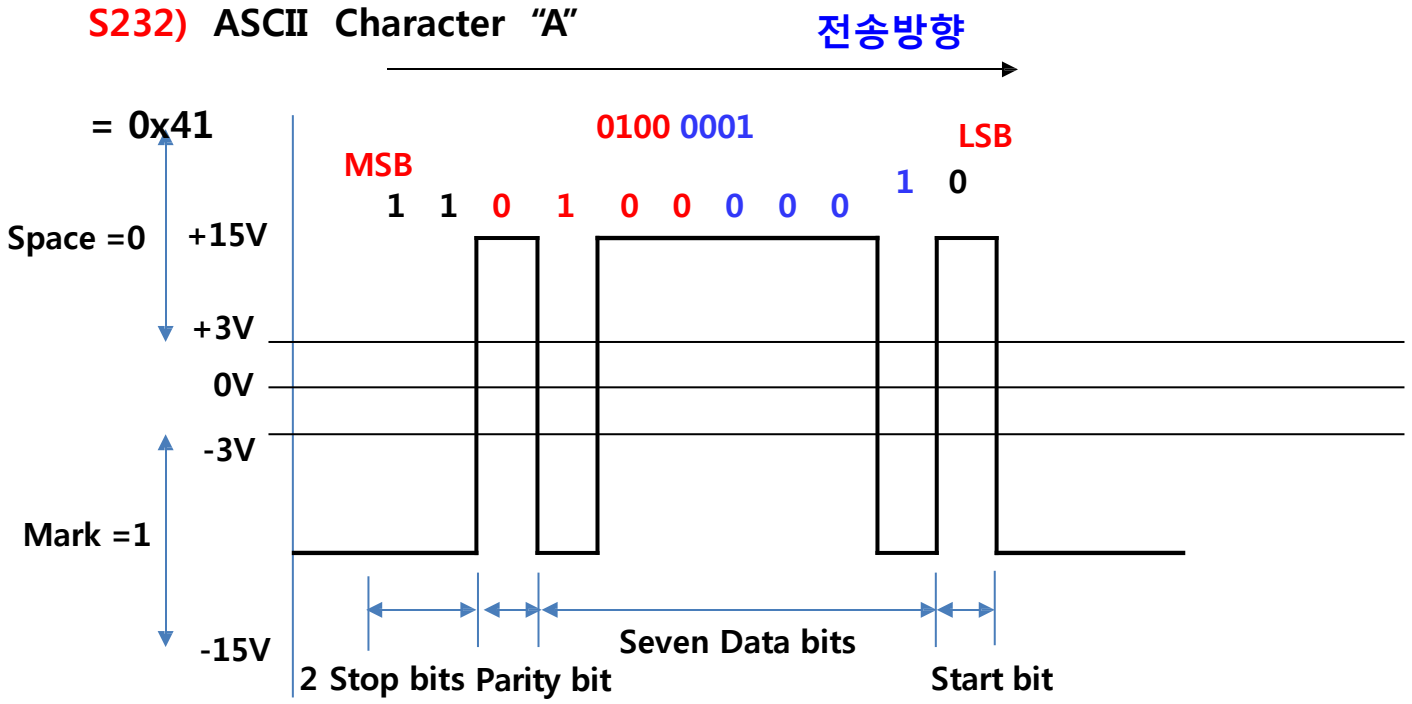
- 입력 저항 3 ~ 7 KW
- 출력 Threshold ± 3V
- 입력 전압 ± 25V(절대값)



6.3. Balanced and Unbalanced 2

- Unbalanced : 비 평형 회로 (R

S232) ASCII Character "A"

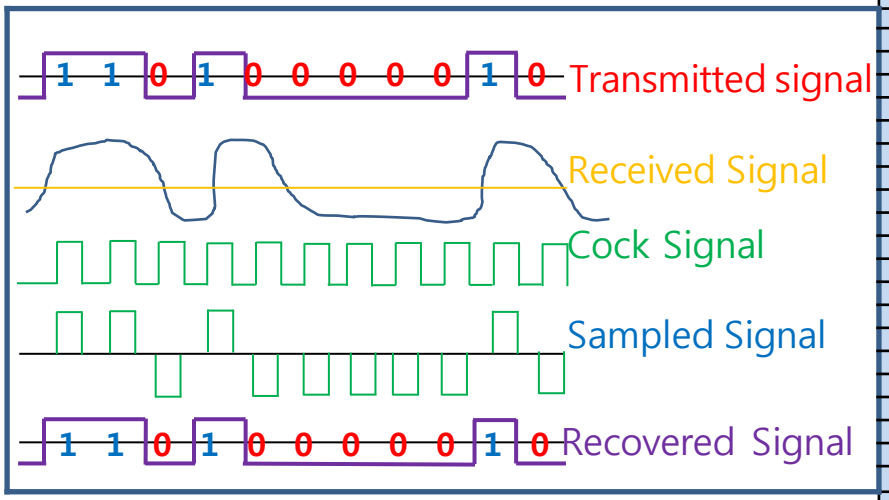


6. Serial Communication

6.3. Balanced and Unbalanced 3

- Unbalanced : 비 평형 회로 (RS232)

- Receiving Data bits



- ASCII Code 표 1930년대 개발

DEC	HEX	OCT	Char	DEC	HEX	OCT	Char	DEC	HEX	OCT	Char
0	00	000	Ctrl-@ NUL	43	2B	053	+	86	56	126	V
1	01	001	Ctrl-A SOH	44	2C	054	'	87	57	127	W
2	02	002	Ctrl-B STX	45	2D	055	-	88	58	130	X
3	03	003	Ctrl-C ETX	46	2E	056	.	89	59	131	Y
4	04	004	Ctrl-D EOT	47	2F	057	/	90	5A	132	Z
5	05	005	Ctrl-E ENQ	48	30	060	0	91	5B	133	[
6	06	006	Ctrl-F ACK	49	31	061	1	92	5C	134	₩
7	07	007	Ctrl-G BEL	50	32	062	2	93	5D	135	]
8	08	010	Ctrl-H BS	51	33	063	3	94	5E	136	^
9	09	011	Ctrl-I HT	52	34	064	4	95	5F	137	_
10	0A	012	Ctrl-J LF	53	35	065	5	96	60	140	`
11	0B	013	Ctrl-K VT	54	36	066	6	97	61	141	a
12	0C	014	Ctrl-L FF	55	37	067	7	98	62	142	b
13	0D	015	Ctrl-M CR	56	38	070	8	99	63	143	c
14	0E	016	Ctrl-N SO	57	39	071	9	100	64	144	d
15	0F	017	Ctrl-O SI	58	3A	072	:	101	65	145	e
16	10	020	Ctrl-P DLE	59	3B	073	;	102	66	146	f
17	11	021	Ctrl-Q DC1	60	3C	074	<	103	67	147	g
18	12	022	Ctrl-R DC2	61	3D	075	=	104	68	150	h
19	13	023	Ctrl-S DC3	62	3E	076	>	105	69	151	i
20	14	024	Ctrl-T DC4	63	3F	077	?	106	6A	152	j
21	15	025	Ctrl-U NAK	64	40	080	@	107	6B	153	k
22	16	026	Ctrl-V SYN	65	41	081	A	108	6C	154	l
23	17	027	Ctrl-W ETB	66	42	082	B	109	6D	155	m
24	18	030	Ctrl-X CAN	67	43	083	C	110	6E	156	n
25	19	031	Ctrl-Y EM	68	44	084	D	111	6F	157	o
26	1A	032	Ctrl-Z SUB	69	45	085	E	112	70	160	p
27	1B	033	Ctrl-[ESC	70	46	086	F	113	71	161	q
28	1C	034	Ctrl-₩ FS	71	47	087	G	114	72	162	r
29	1D	035	Ctrl-] GS	72	48	090	H	115	73	163	s
30	1E	036	Ctrl-^ RS	73	49	091	I	116	74	164	t
31	1F	037	Ctrl_US	74	4A	092	J	117	75	165	u
32	20	040	SPACE	75	4B	093	K	118	76	166	v
33	21	041	!	76	4C	094	L	119	77	167	w
34	22	042	*	77	4D	095	M	120	78	170	x
35	23	043	#	78	4E	096	N	121	79	171	y
36	24	044	$	79	4F	097	O	122	7A	172	z
37	25	045	%	80	50	100	P	123	7B	173	{
38	26	046	&	81	51	101	Q	124	7C	174	|
39	27	047	'	82	52	102	R	125	7D	175	}
40	28	050	{	83	53	103	S	126	7E	176	~

6.3. Balanced and Unbalanced 4

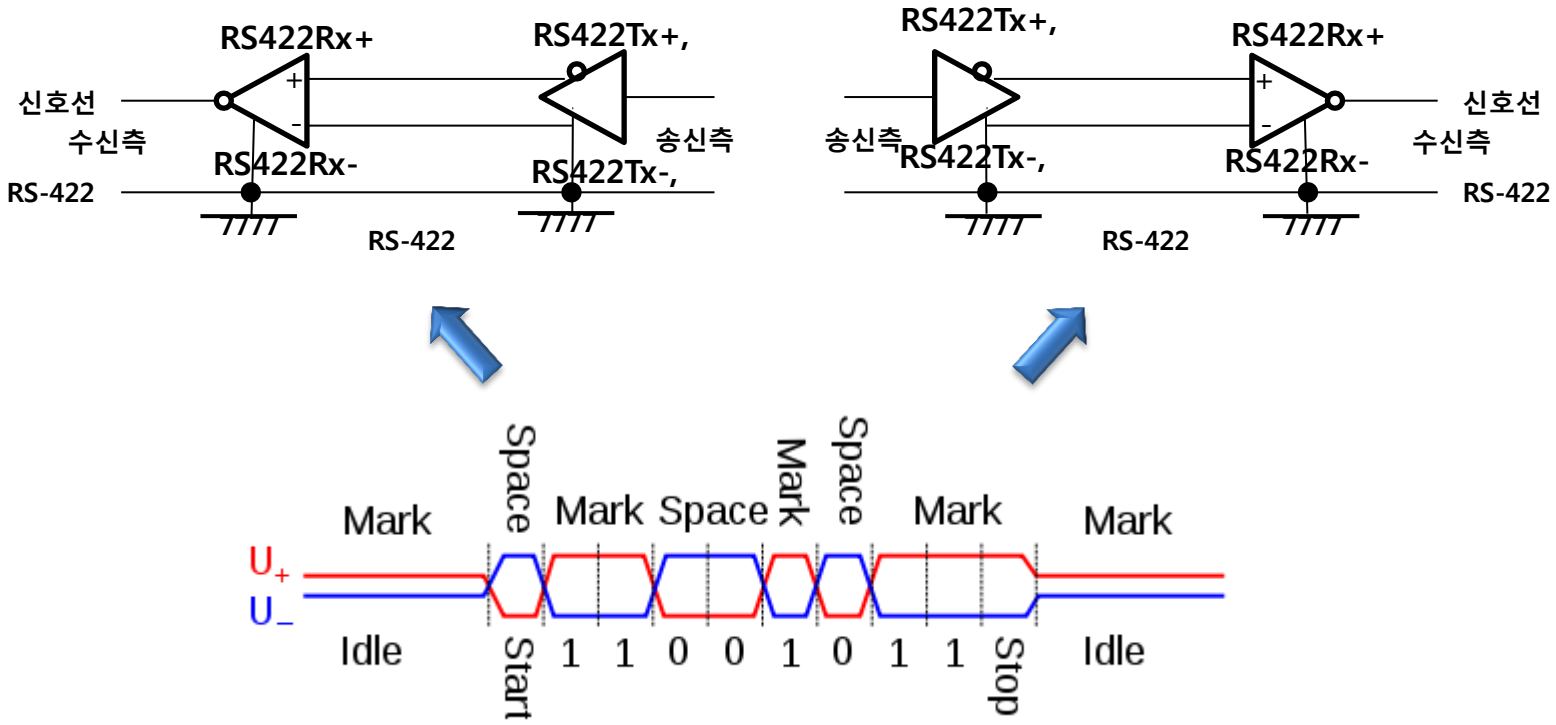
- Balanced : 평형 회로 (RS422)
  - Noise에 대한 성능보완을 위하여 데이터 비트를 보낼 때 2개의 신호선의 선간 전압차이를 이용함, Full Duplex로서 4개의 신호 선을 사용함.
  - Noise가 있어도 전압 차는 나타남.
  - a balanced signal is represented by a pair of wires where a voltage difference is used to transmit/receive binary information:

RS422Tx+, RS422Rx+	-7~12VDC
RS422Tx-, RS422Rx-	-7~12VDC

Mark = 1	RS422Tx+, RS422Rx+ 의 전압 (+1VDC) < (RS422Tx-, RS422Rx- 전압 (+4VDC)
Space = 0	RS422Tx+, RS422Rx+ 의 전압 (+1VDC) > (RS422Tx-, RS422Rx- 전압 (+4VDC)

6.3. Balanced and Unbalanced 5

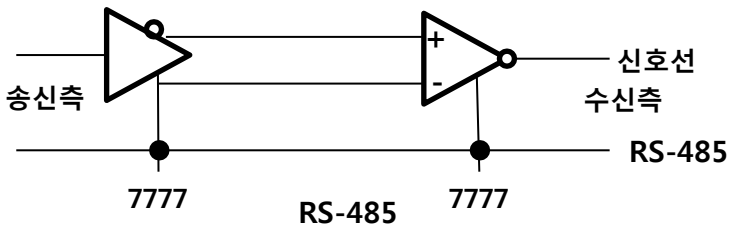
- Balanced : 평형 회로 (RS422)
- Point to Point 와 Multi-drop 모두 가능

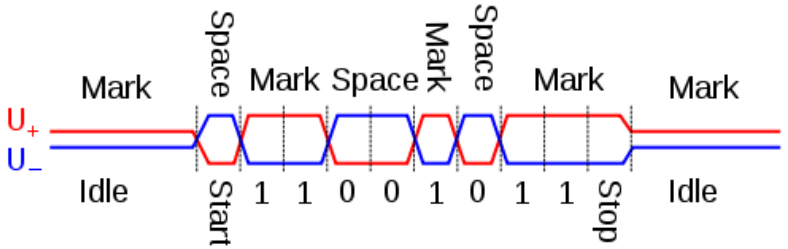


6.3. Balanced and Unbalanced 6

- **Balanced : 평형 회로 (RS485)**
  - **Noise에 대한 성능보완을 위하여 데이터 비트를 보내기 위하여 2개의 신호선의 선간 전압차이를 이용함. Half Duplex로 2개의 신호선 사용**
  - a balanced signal is represented by a pair of wires where a voltage difference is used to transmit/receive binary information:

RS485+	-7~12VDC	Mark = 1	RS485+ (+1VDC) < RS485- (+4VDC)
RS485-	-7~12VDC	Space = 0	RS485+ (+4VDC) > RS485- (+1VDC)





6.4. RS232

- RS232C – Recommended Standard
  232의 약어 C 는 표준 Version을 나타냄

  - 공식적인 명칭은 EIA-232-C로서 CCITT의 권고에 따라 미국의 EIA(미국 전자 공업회)가 정한 규격임.
  - 데이터를 직렬 전송 방식으로 전송할 때 통신회선에서 사용하는 전기적인 신호의 특성과 연결장치의 형상 등 물리적인 규격을 정하고 있음.
    (전압의 Level, Impedance ,데이터의 제어와 타이밍)
  - C version – 케이블 길이 15m제한, 최대 데이터 전송률 20Kbps.
  - D version – 케이블길이 제한 대신 최대 수용 용량을 명시함.
  - 초기 25pin, 후에 9pin

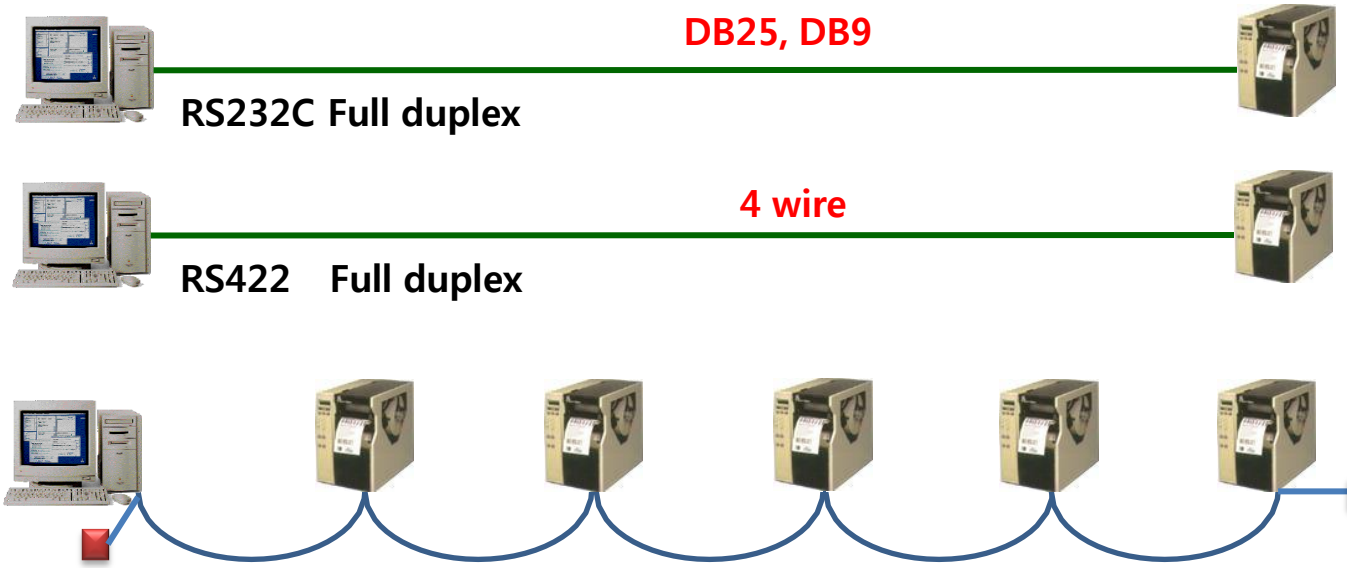
6.5. RS232/485/422 사양비교

- RS232C, RS22, RS485 비교

구분	RS232C	RS422	RS485
송, 수신기	1:1	1:1 (1:N)	1:N
통신방식	Point to Point	Point to Point (Multi-drop)	Multi-drop
기본거리	30~60m (<)	1200m>	1200m>
Volt for Ground	12v to Ground	Tx+, Rx+ Tx-, Rx- 전압차	485+, 485- 전압차
Wire 수	9pins, 25pins	4 wires	2 wires
duplex	Full duplex	Full duplex	Half duplex
속도	저속	저속	저속

6. Serial Communication

6.6. RS232/485/422 Configuration 비교

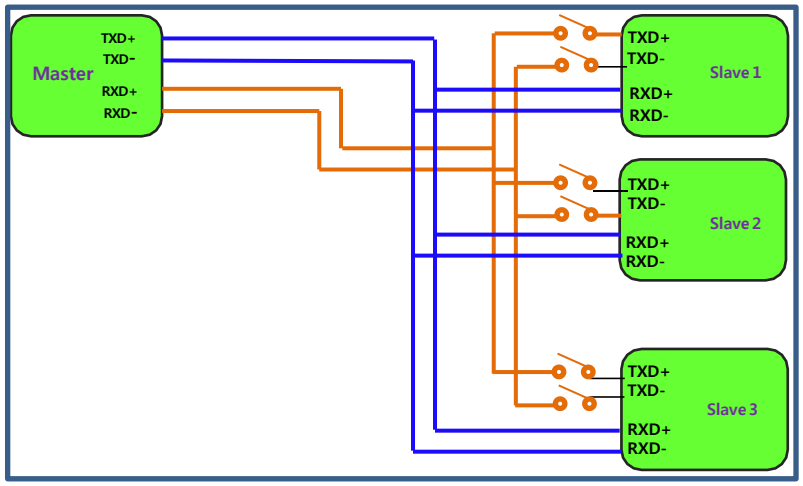


- RS485 Multi-drop Half duplex : 2 wire
- RS422 Multi-drop Full duplex : 4 wire

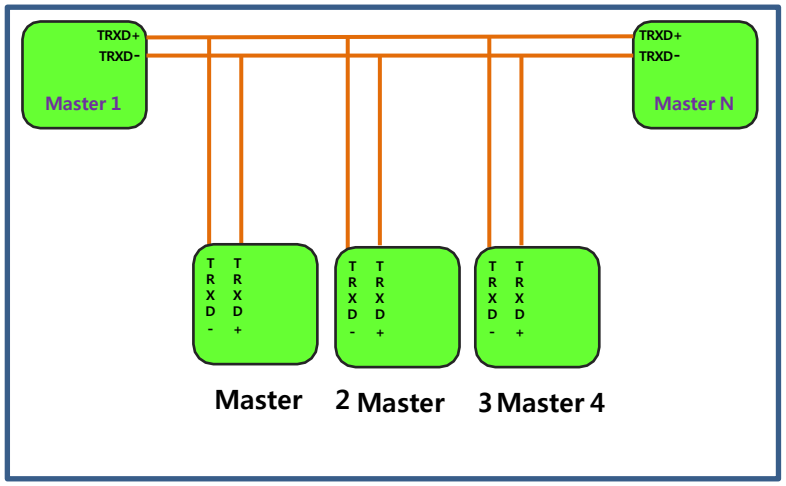
- 선로 도통 시험시 단선을 식별한다.
- 평상시 약한 전류가 흘러 수신 부에서 단선유무를 확인함.
- 저항 없이 연결 시 장치가 가동되는 것으로 판단함 (switch 기능).
- 반사형상에 의한 Noise 방지.

6.6. RS232/485/422 Configuration 비교

RS422 Multi Drop



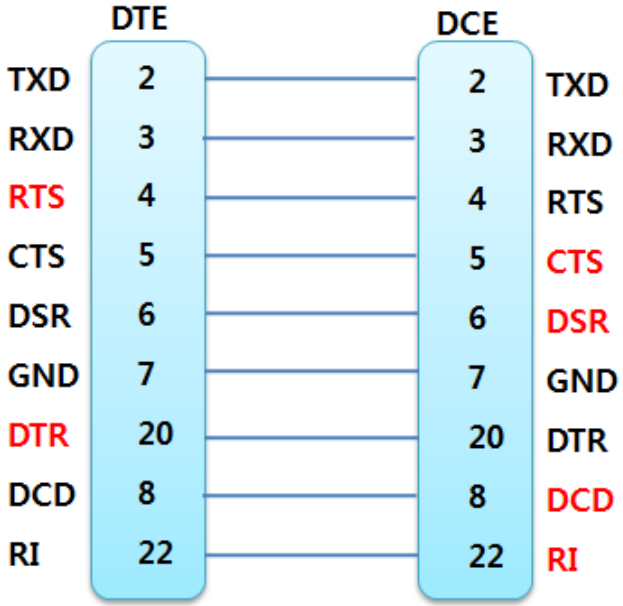
RS485 Multi Drop



6. Serial Communication

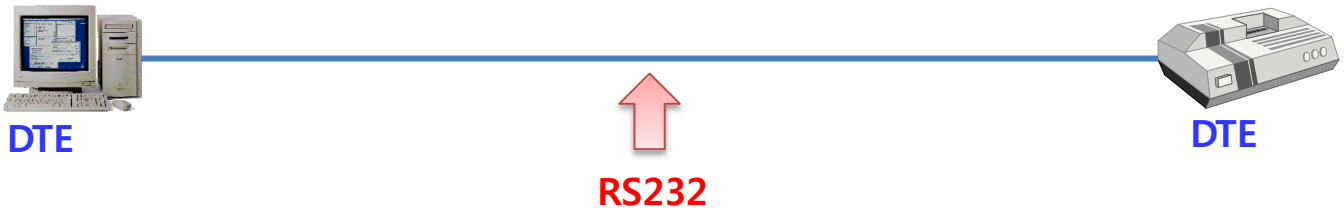
6.7. RS232 Handshaking

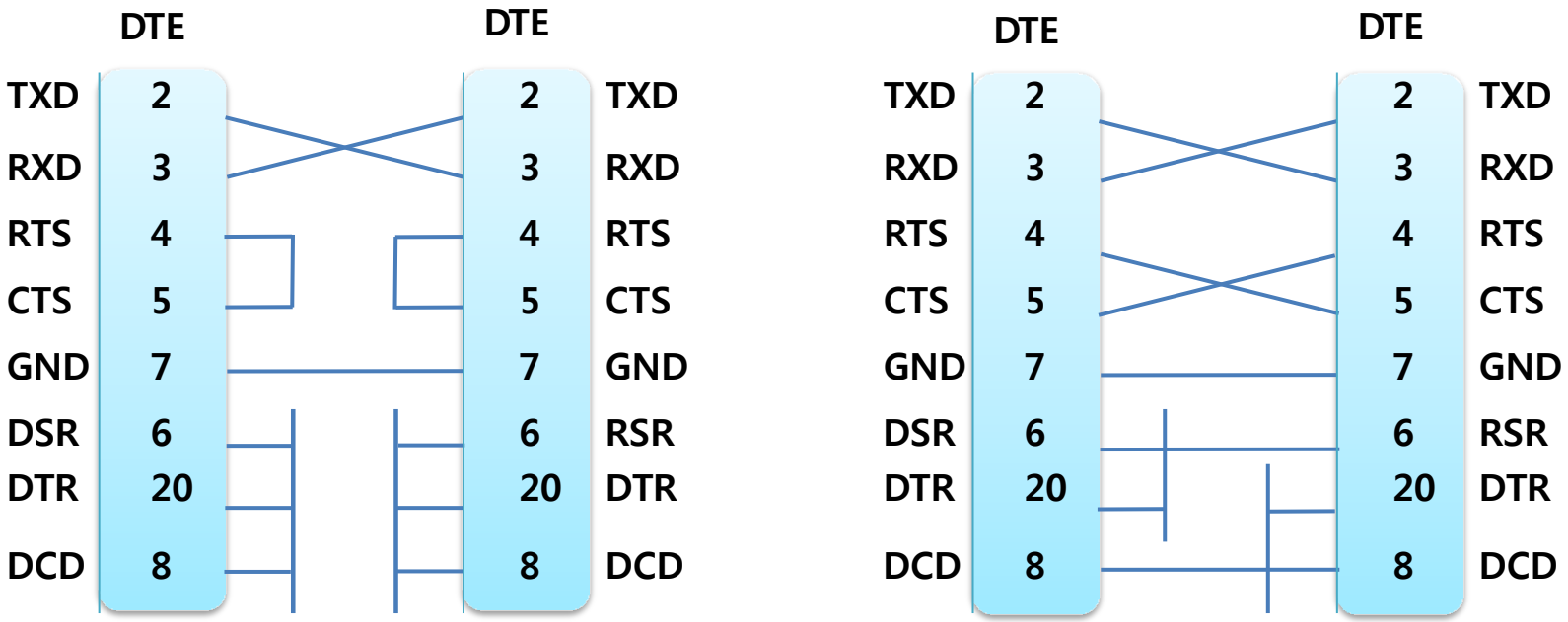
RS232 – DTE와 DCE사이의 통신 규약



Signal	Name	Description
TXD	Transmit Data	Data 송신 신호선
RXD	Receive Data	Data 수신 신호선
RTS	Ready to Send	DTE가 Data를 보낼 준비가 되었음을 DCE에게 알리는 신호선
CTS	Clear to Send	DCE가 Data를 받을 준비가 되었음을 DTE 에게 알리는 신호선
DSR	Data Set Ready	모뎀이 전원 On 후 자신의 상태가 송수신DTE에게 가능함을 알리는 신호선
GND	Signal Ground	Signal Ground
DTR	Data Terminal Ready	DTE에 전원이 인가되고 통신포트를 초기화 시킨 후 DCE에게 보내는 신호선
DCD	Data Carrier Detect	모뎀이 상대편 모뎀의 Carrier를 Detect 했음을 DTE 에게 알리는 신호선
RI	Ring Indicator	상대편 모뎀의 전화벨신호를 감지 하였을 때 DTE에게 알리는 신호선 (컴퓨터의 응답 프로그램을 호출)

6.8. RS232 Handshaking Without Modem

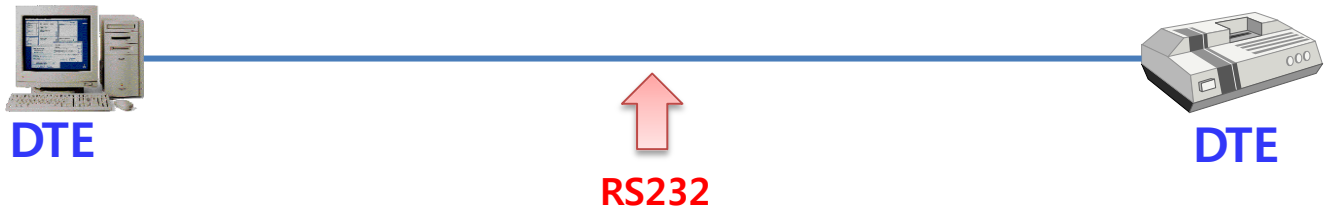


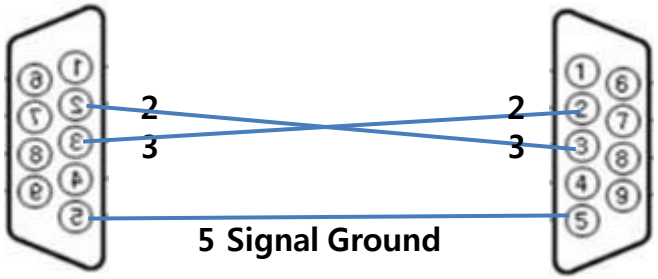


6.9. RS232 Parameters

Parameters	Value
Baud Rate	1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
Data Bits	5,6,7,8
Parity Check	NONE, odd, even, Mark, Space (Character에 한 bit (0 혹은 1)를 추가하여 항상 1의 개수가 even, 이나 odd가 되게 함. Mark : 항상 '1' 이 되게 함. Space : 항상 '0' 이 되게 함.
Stop bits	1, , 1.5, 2 한 Character 보내고 쉬는 시간
Force Packet Transmit time	0~65536 mls
Force Packet Transmit Length	0~65536 bytes
Delimiter	Hex Value
Flow Control	None, Xon/Xoff, CTS/RTS

6.10. Null Modem



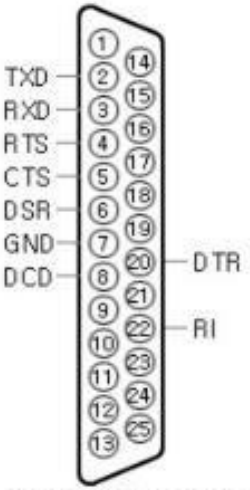


DB 9 Null Model

6. Serial Communication

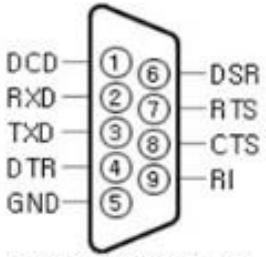
6.11. Connector Pin Specification

25pins



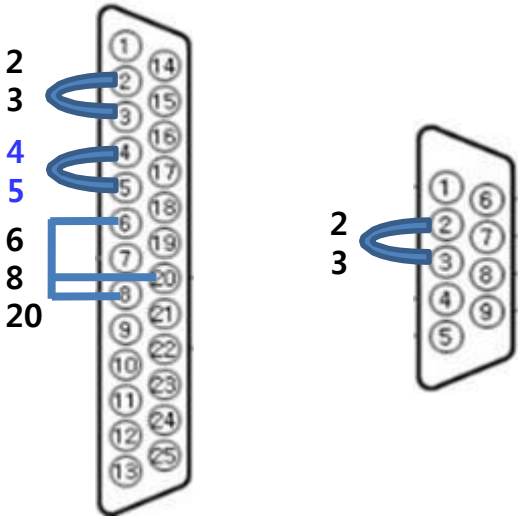
DB-25 RS232 코넥터

9pins



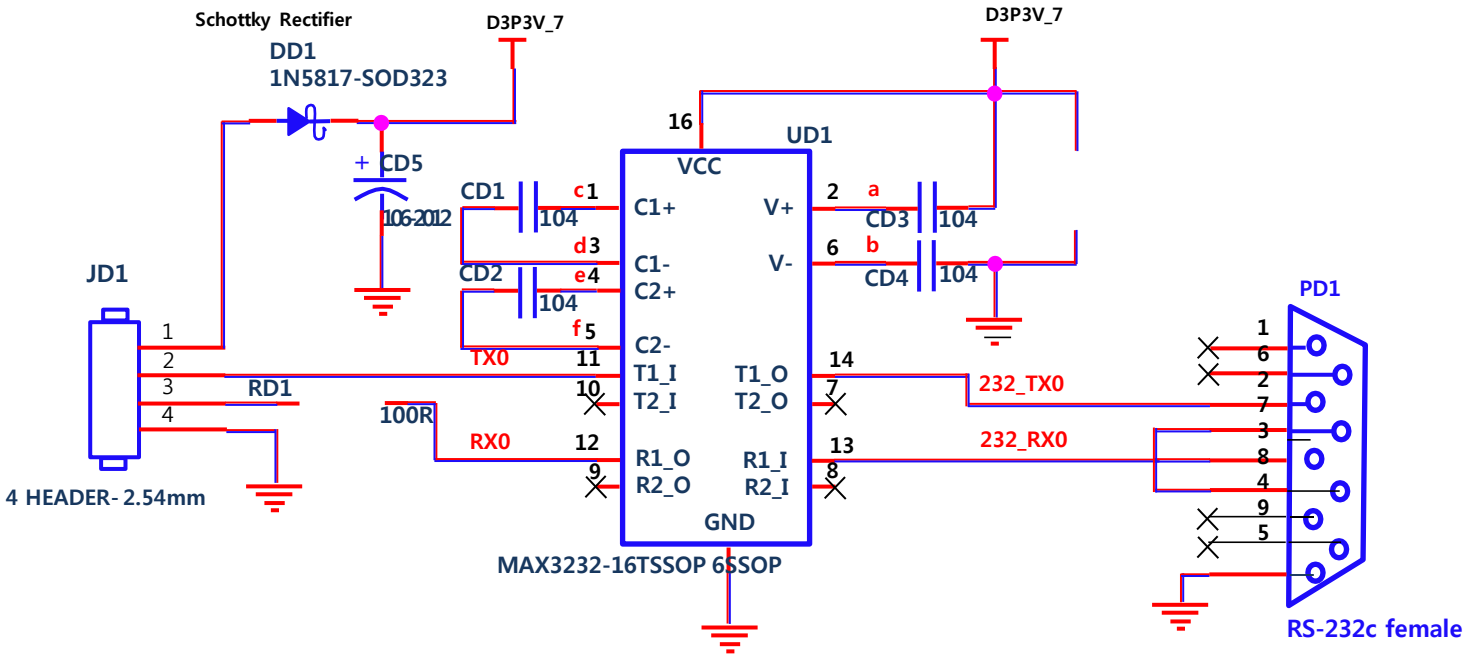
DB-9 RS232 코넥터

Loop Back Connector



6. Serial Communication

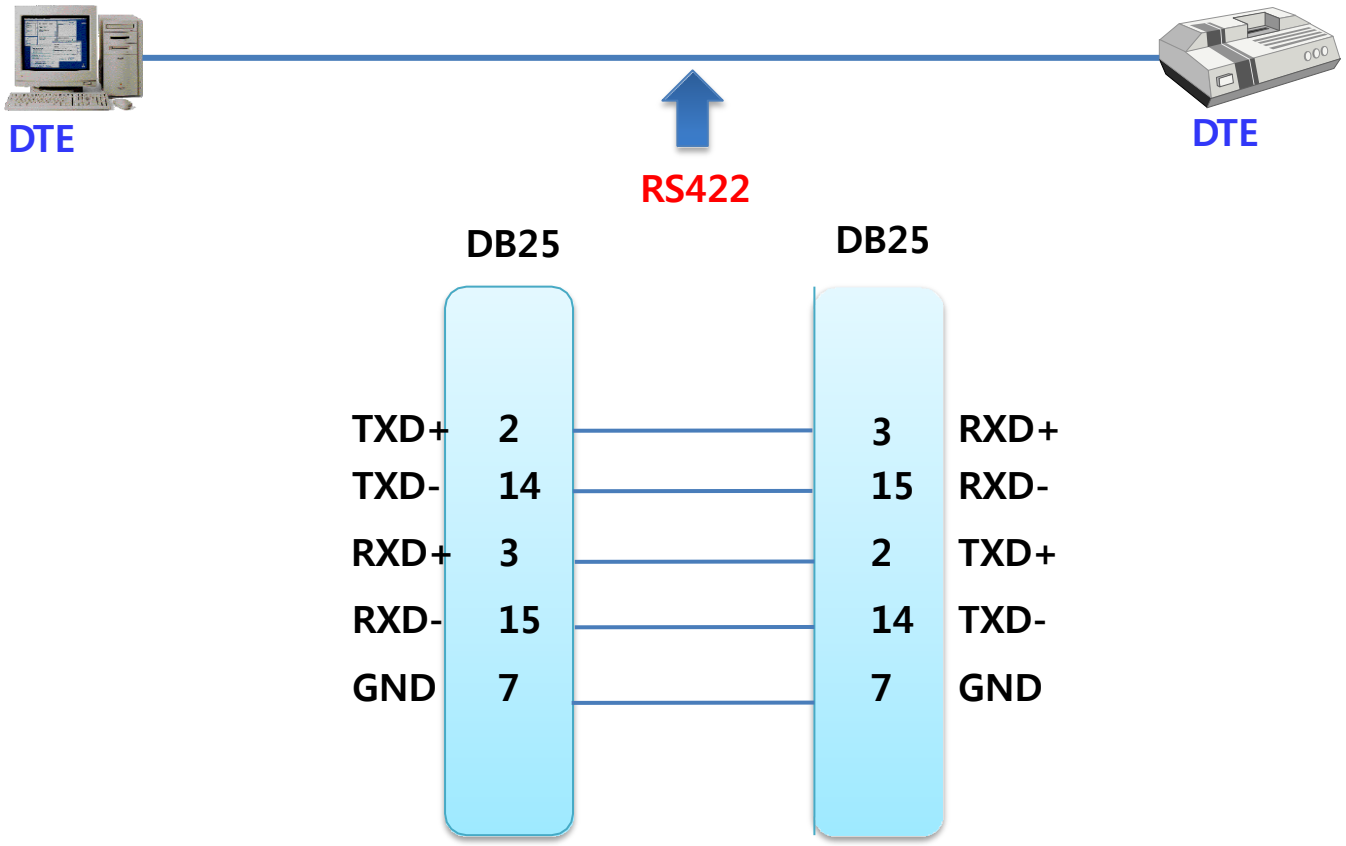
6.12. RS232 Driver Chipset



- **Drive Chip 을 거친 후 전압 Level :** -24V~+24V
- 실제로는: -5V ~ -15V, +5V ~ +15V
- Mark State -5 ~ -15V (Logic Level 1)
- Space State +5 ~ +15V (Logic Level 0)

6. Serial Communication

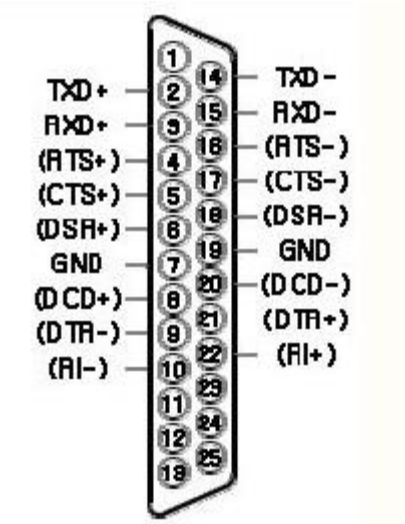
6.13. RS422 -4 wire



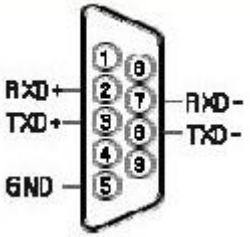
6. Serial Communication

6.14. RS422 Connector Pin Specification

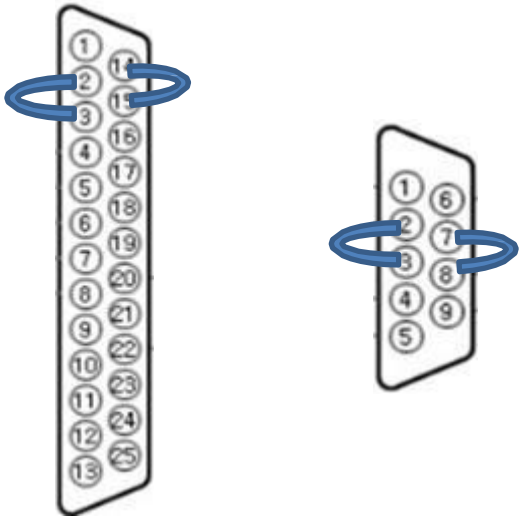
25pins



9pins

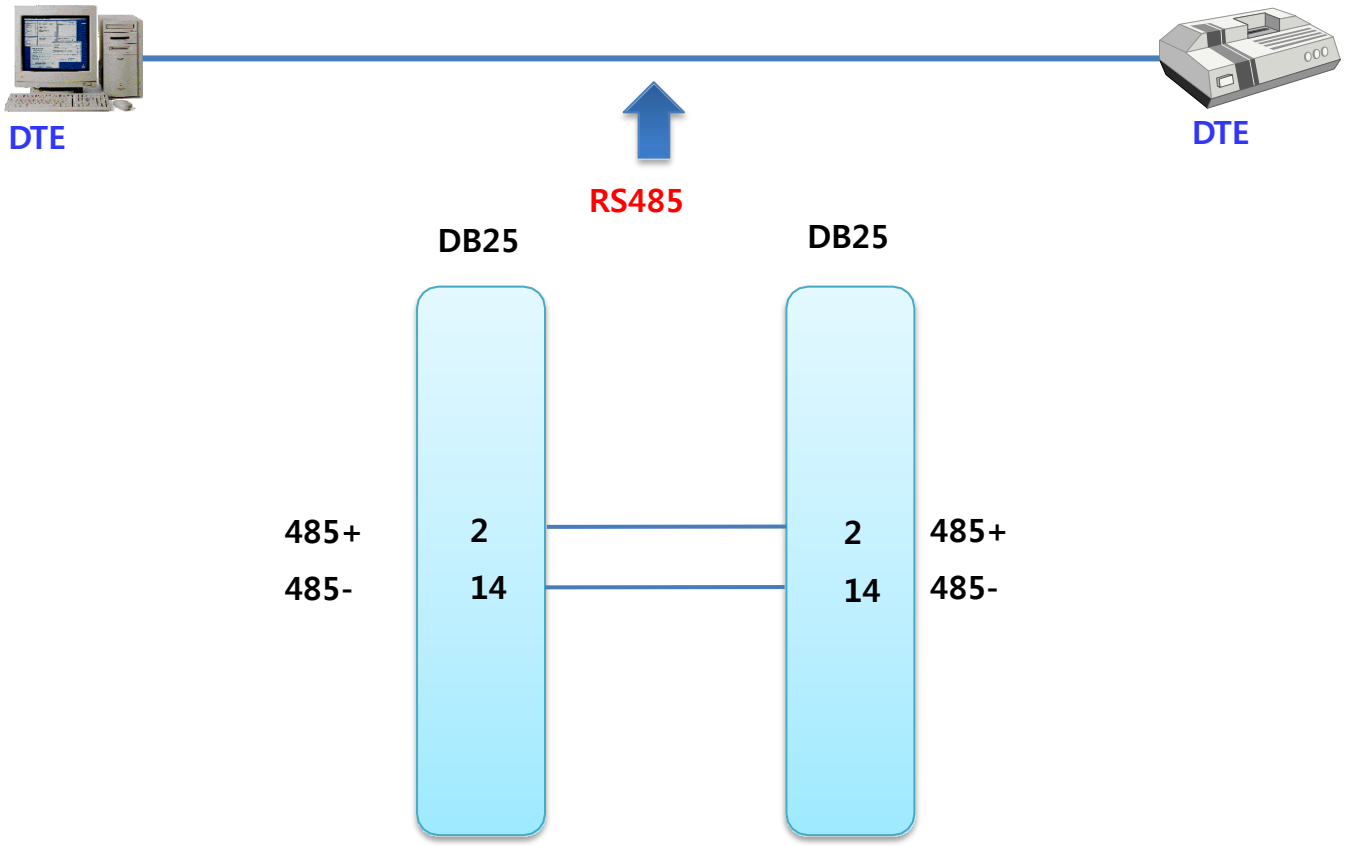


Loop Back Connector



6. Serial Communication

6.15. RS485 – 2 wire

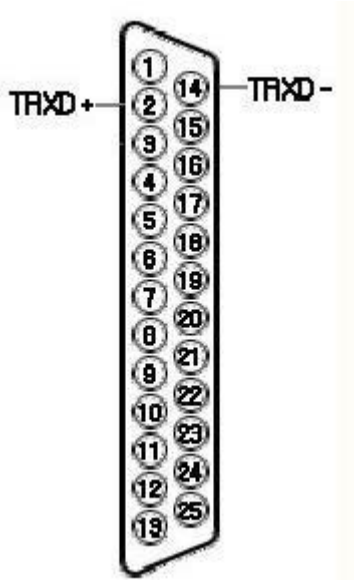


6.16. RS485 Parameters

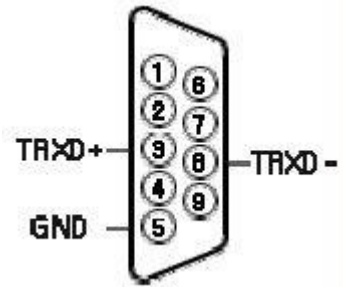
Parameters	Value
Baud Rate	1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
Data Bits	5,6,7,8
Parity Check	NONE, odd, even, Mark, Space (Character에 한 bit (0 혹은 1)를 추가하여 항상 1의 개수가 even, 이나 odd가 되게 함. Mark : 항상 '1' 이 되게 함. Space : 항상 '0' 이 되게 함.
Stop bits	1, 2
Force Packet Transmit time	0~65536 mls
Force Packet Transmit Length	0~65536 bytes
Delimiter	Hex Value
RS485 Transmission Delay time	0~65536 us

6.17. RS485 Connector Pin Specification

25pins



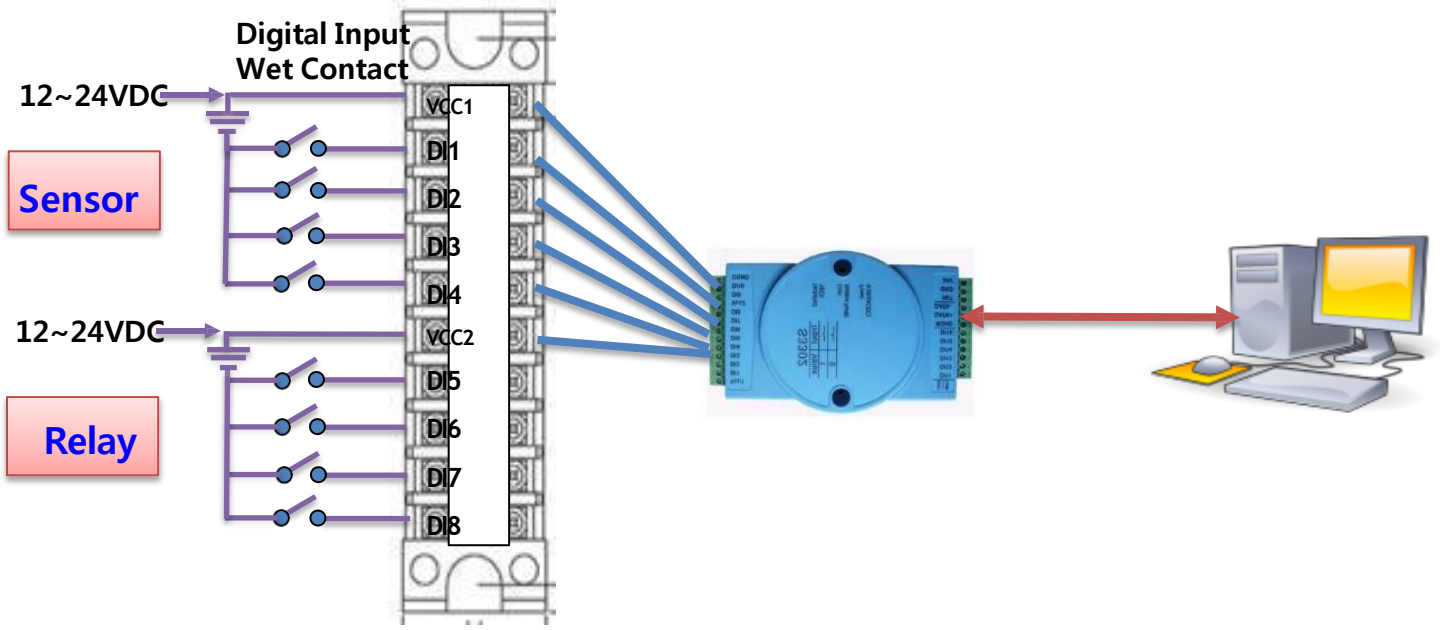
9pins



7. DI/DO/AI/AO

7.1. Digital Input 1

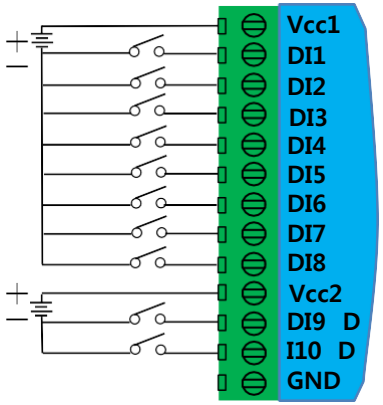
- 접점의 On/Off 상태를 감시하고 디지털 부호 '1'과 '0'로 변환하여 Computer Program에 전달한다.
- 접점은 On/Off를 상태를 발생시키는 Switch, 근접 Sensor, Relay등에 연결된다.



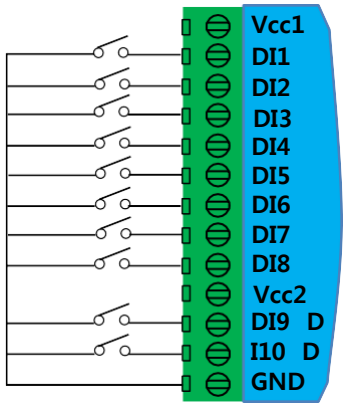
7.1. Digital Input 2

- Digital Input 의 3가지 종류
- Wet Contact Input
- Dry Contact Input
- Open Collector Input

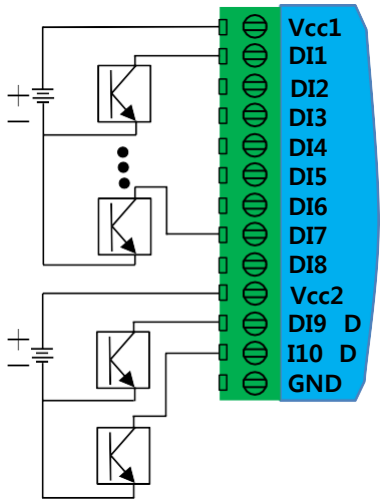
Wet Contact Input



Dry Contact Input



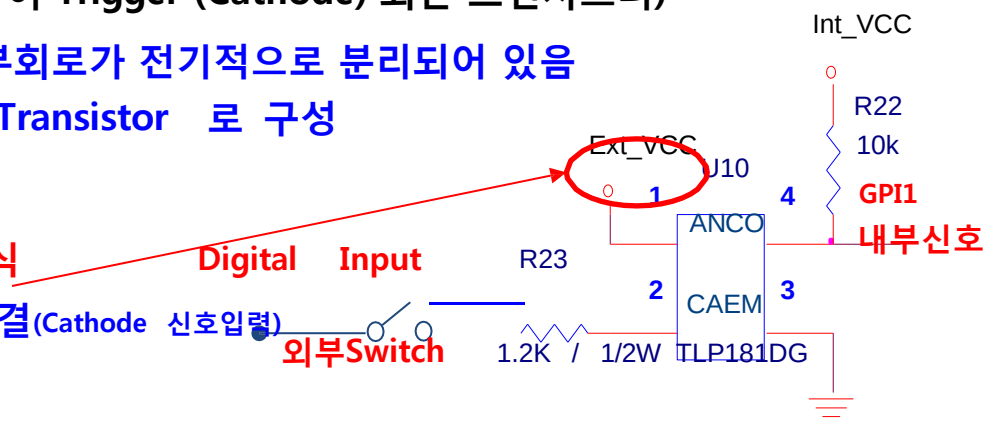
Open Collector Input



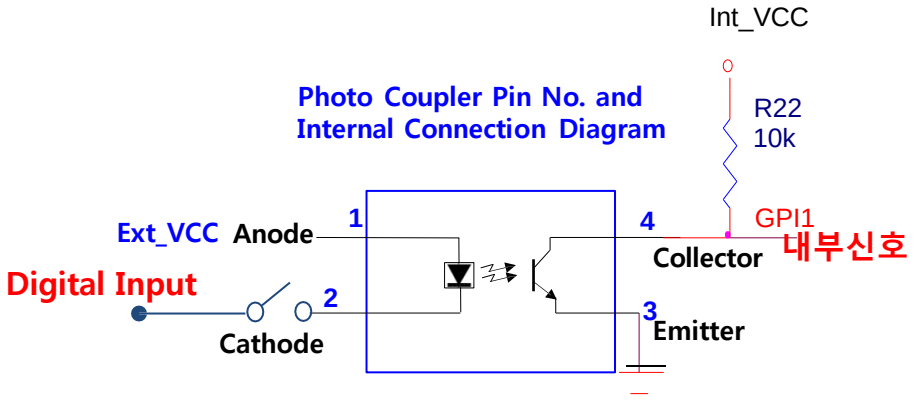
7. DI/DO/AI/AO

1. Digital Input 3

- **Wet Contact :**
  - **Photo Coupler (빛에 의하여 Trigger (Cathode) 되는 트랜지스터)**
    - 외부 Input 회로와 내부회로가 전기적으로 분리되어 있음
    - Input Led 와 Output Transistor 로 구성
  - **Input(외부신호)**
    - Common Anode방식
    - Ground에 Switch 연결(Cathode 신호입력)
  - **Output (Internal Signal)**
    - NPN형 Transistor
    - LED 신호가 Base로 입력



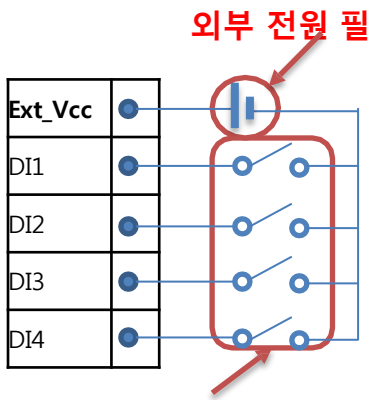
Digital Input	LED	TR	Output
Switch Open	Off	Off	High(Int_Vcc)
Switch Short	On	On	Low(Int_GND)



7. DI/DO/AI/AO

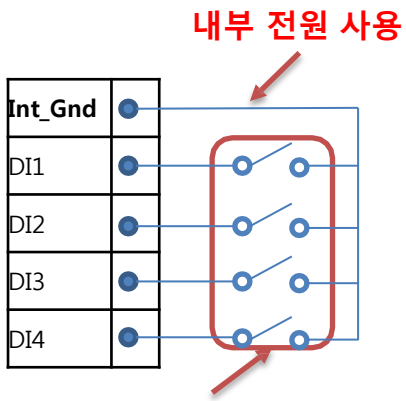
1. Digital Input 4

- **Application:**



릴레이 또는 스위치

Wet Contact

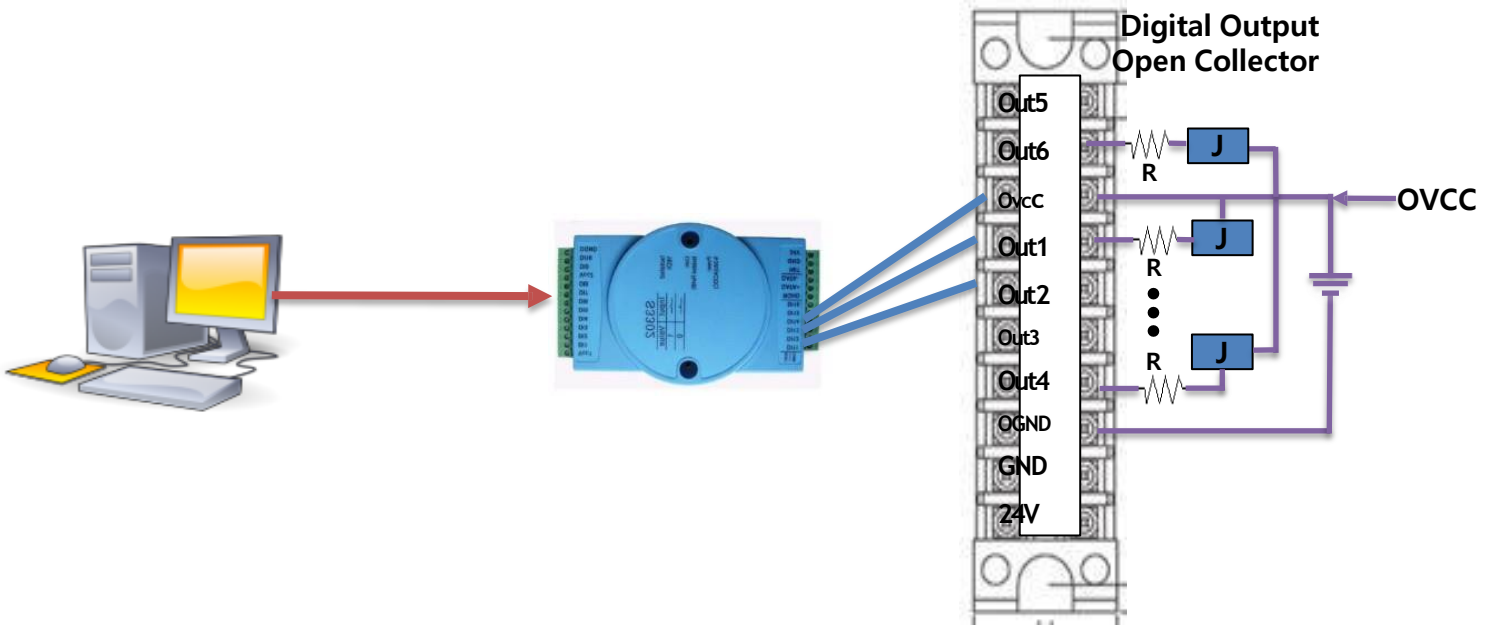


릴레이 또는 스위치

Dry Contact

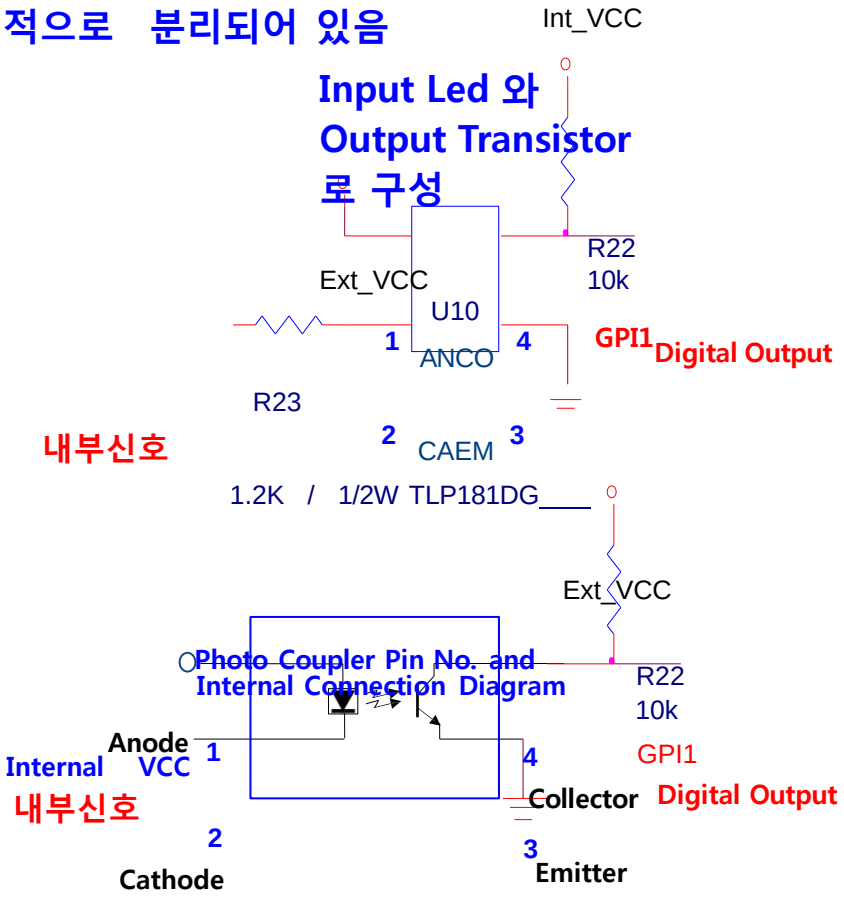
7.2. Digital Output 1

- 외부의 접점을 전압의 Level을 통하여 On/Off 상태로 만들어 줌.
- 접점은 Relay, 경광등, 설비의 가동 스위치 등에 연결되어 기계설비를 제어함.



2. Digital Output 2

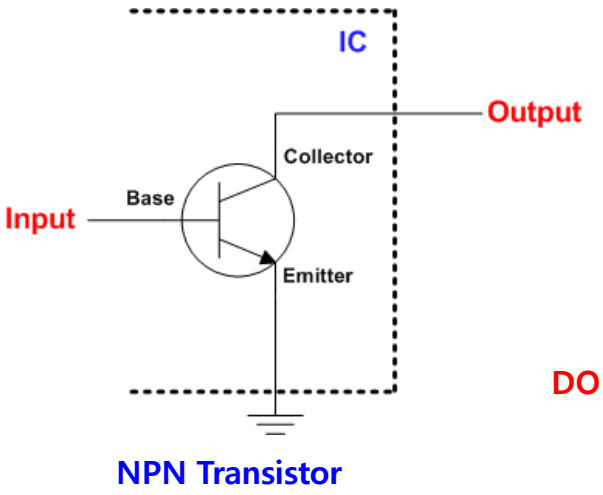
- **Open Collector:**
  - **Photo Coupler (빛에 의하여 Trigger (Cathode) 되는 트랜지스터)**
    - 외부 Input 회로와 내부회로가 전기적으로 분리되어 있음
    -
  - **Input(외부신호)**
    - **Common Anode방식**
    - **Cathode에 신호입력**
  - **Output (Internal Signal)**
    - **NPN형 Transistor**
    - **LED 신호가 Base로 입력**



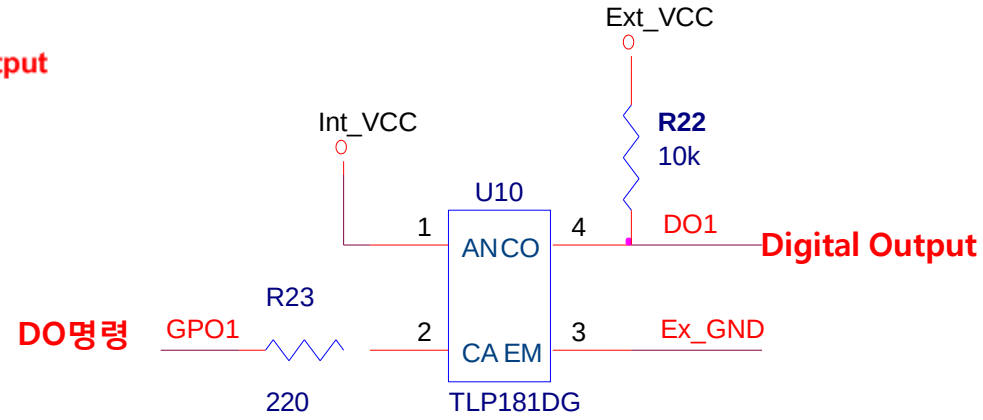
Digital Output 명령 내부신호	LED	TR	Digital Output
ON ('1')	Off	Off	High(Ext_Vcc)
OFF('0')	On	On	Low(Ext_GND)

2. Digital Output 3

- **Open Collector:**
  - Input이 Low면 Transistor는 ON 이 되어 Output은 GND와 연결, Output은 Low 가 출력
  - Input이 High면 Transistor는 OFF가 되어 Output은 Hi-Impedance
  - R22의 Pull Up 저항에 의해 Hi-Impedance 가 아닌 high 상태



NPN Transistor



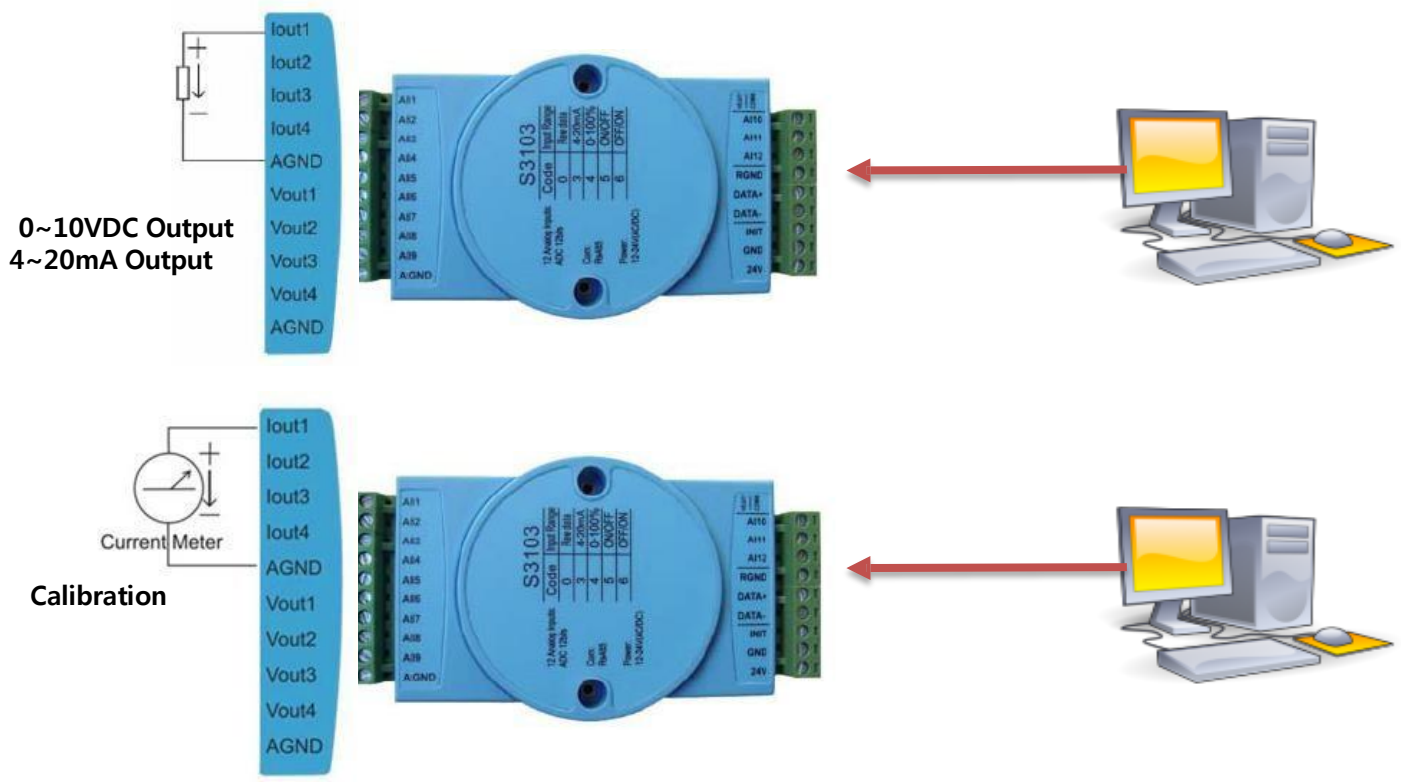
7. DI/DO/AI/AO

7.3. Analog Input

- 전압, 전류, 전하 등 다양한 Analog값을 감지하여 Amplifier, AD converter를 통하여 Digital Value로 전환하여 컴퓨터에 전달함.
- Analog값은 Digital Value로 전환 될 때 정해진 범위 안에서 변환됨.(0~10VDC, 4~20mA등)
- Analog값 (0~10VDC)의 경우 Digital Value (0~4095) 사이의 값으로 변환된다면 이것은 0~10VDC를 4095 level로 표현한다는 의미.
- Resolution(1개 데이터의 정밀도)과 Sampling(시간당 데이터 수집 량)
- Calibration

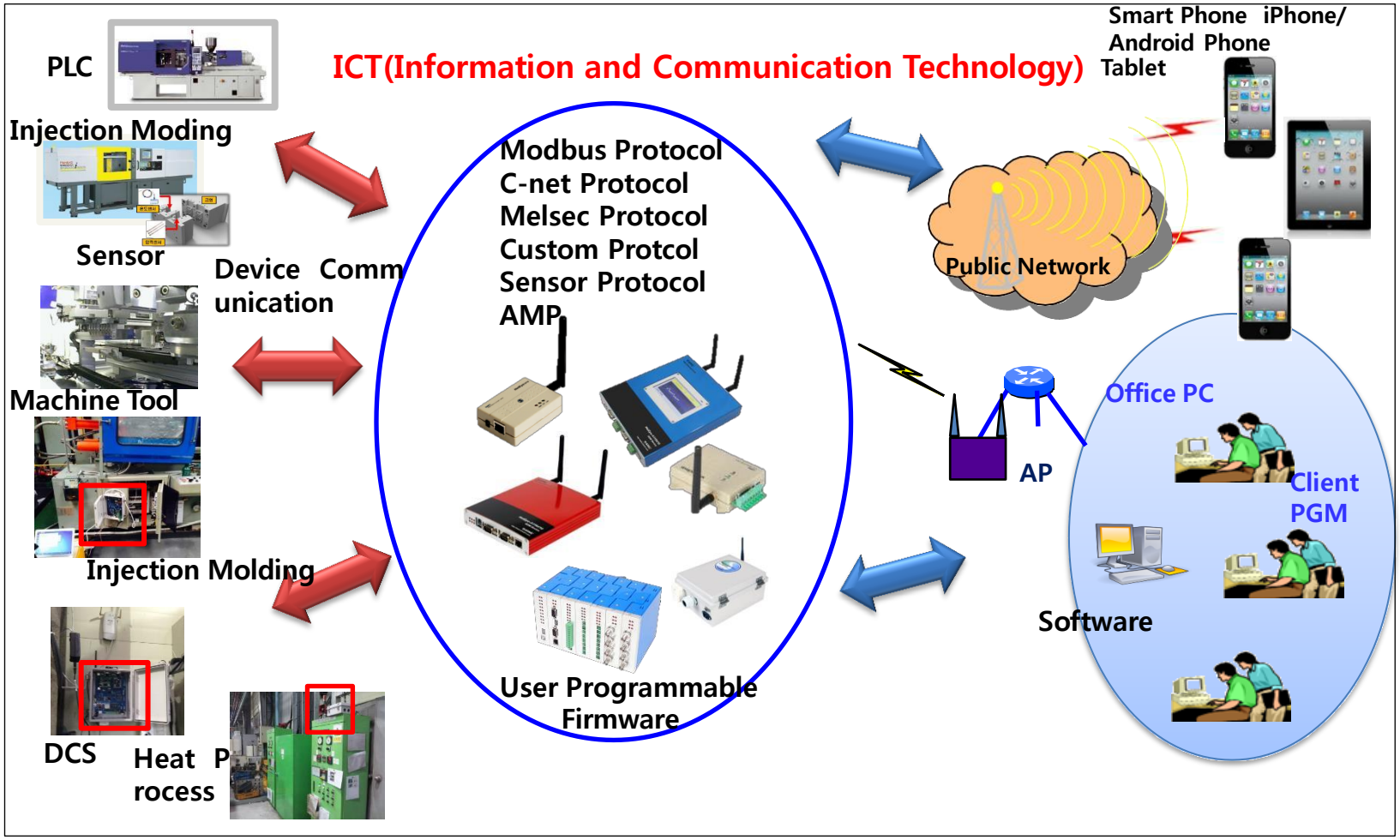
7.4. Analog Output

- 전압, 전류, 전하 등 다양한 Analog값을 출력시킴.



7. DI/AI/AO

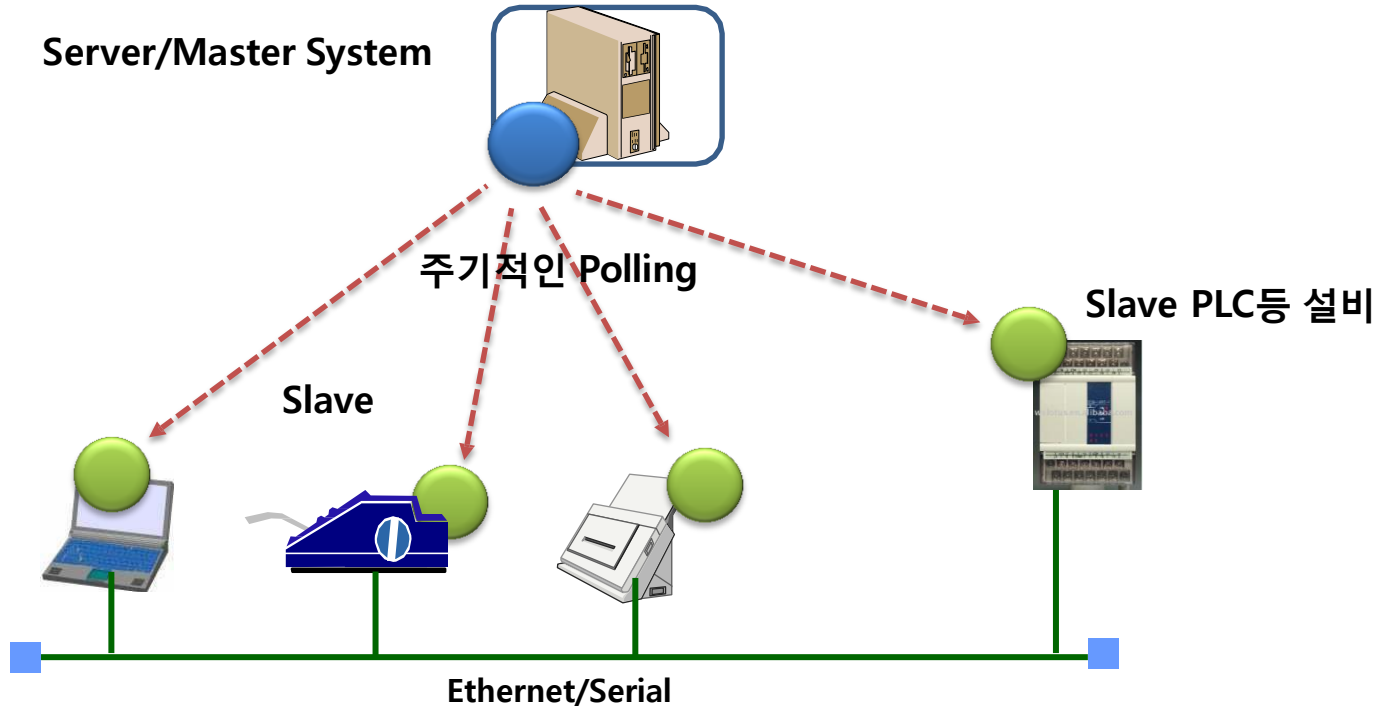
7.5 현장 설비의 정보화



8. Communication Programming Mode

8.1. Polling Mode

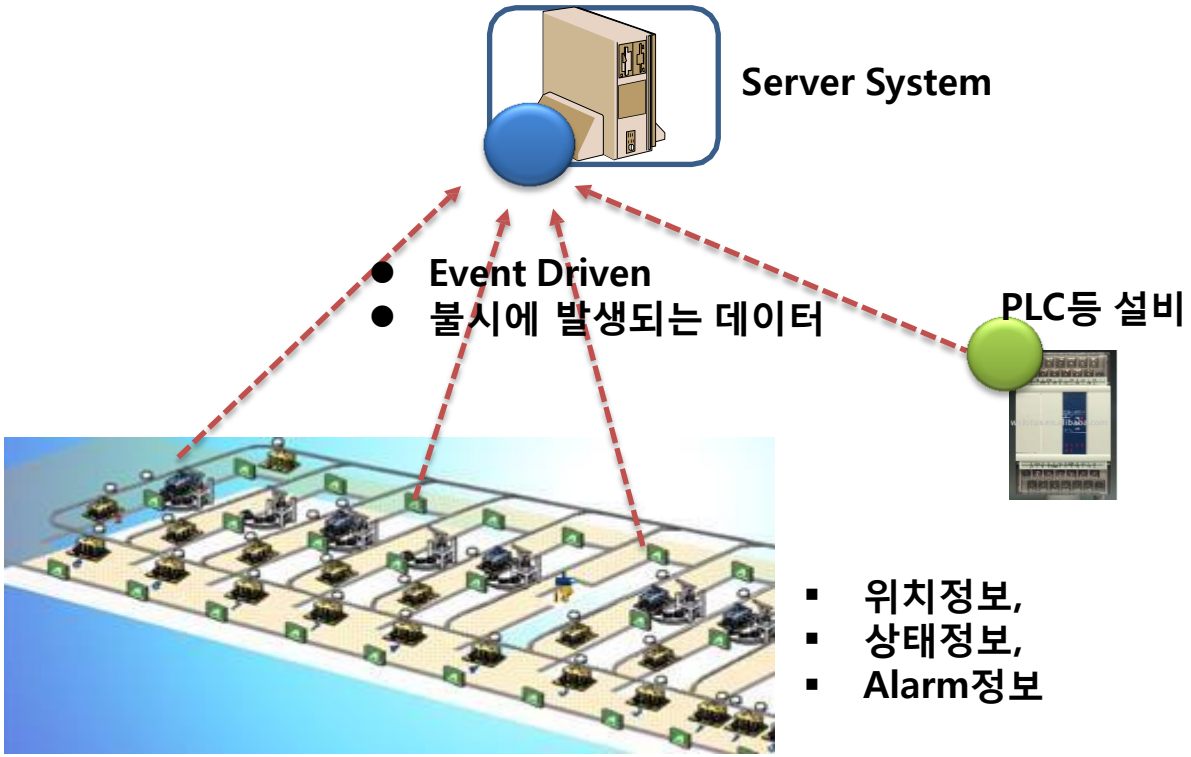
- Computer(Master)가 제어권을 가지고 상대방에게 Request를 함.
- Mater는 주기적인 Polling으로 상대방의 상태정보를 수집함.
- Slave는 Mater의 Request가 있어야만 응답만 함.



8. Communication Programming Mode

8.2. Event Driven Mode

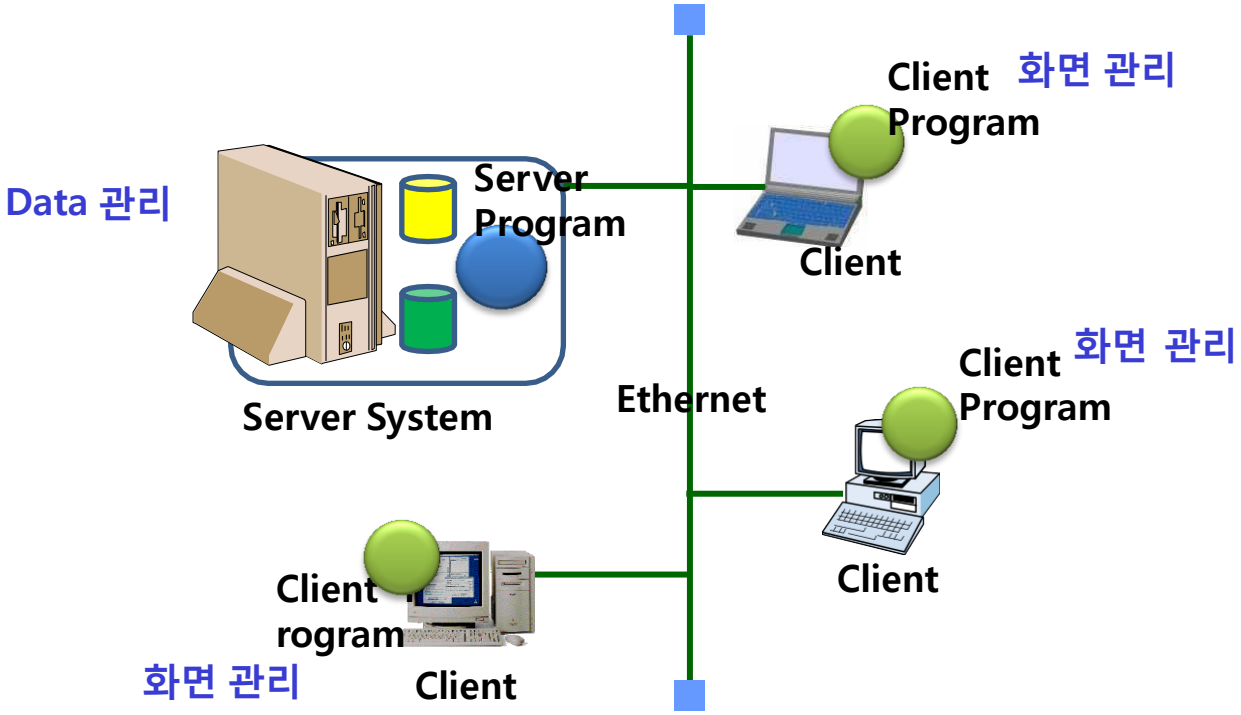
- Server와 Client가 동등한 제어권을 가지고 서로에게 Request를 함.
- Server는 항상 수신 할 준비를 하고 있어야 함.
- 임의의 시간에 예고 없이 데이터가 발생됨.



8. Communication Programming Mode

8.3. Client/Server

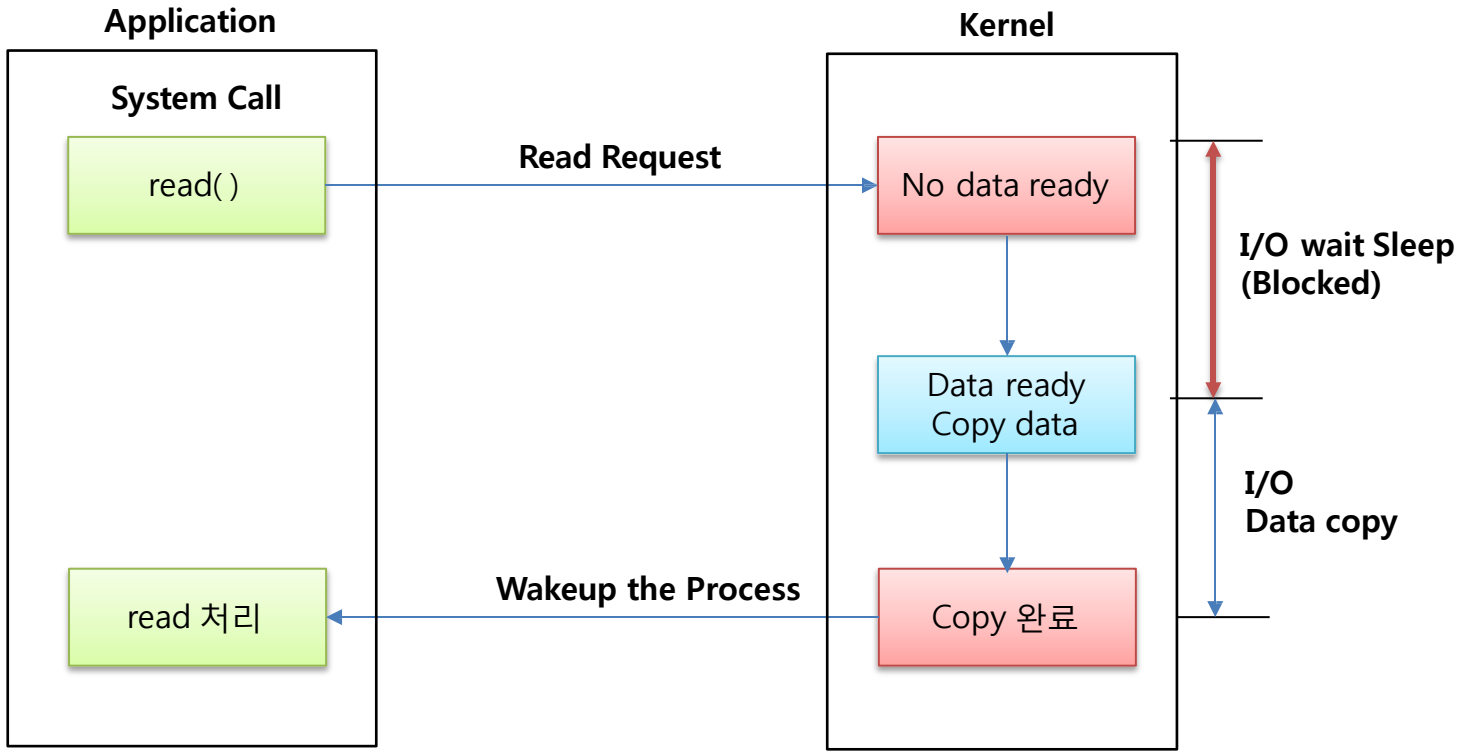
- 일반 Computer System의 전형적인 구조를 따름.



8. Communication Programming Mode

8.4. Blocking and None-Blocking 1

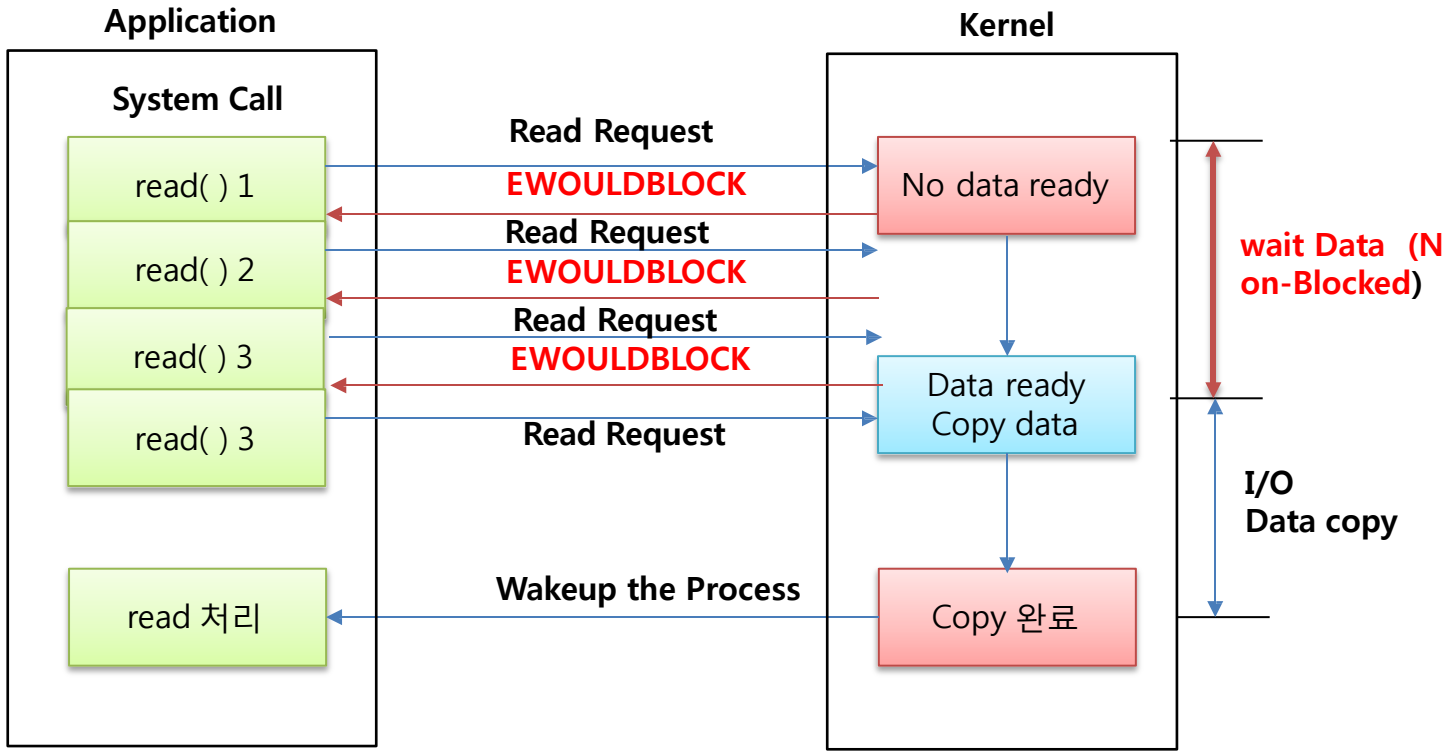
- **Blocking Mode :통신프로그램에서 A Program이 B Program으로 부터 Data가 수신될 때까지 기다리며 Blocked되어 있는 Mode (I/O Waiting Sleep 상태)**



8. Communication Programming Mode

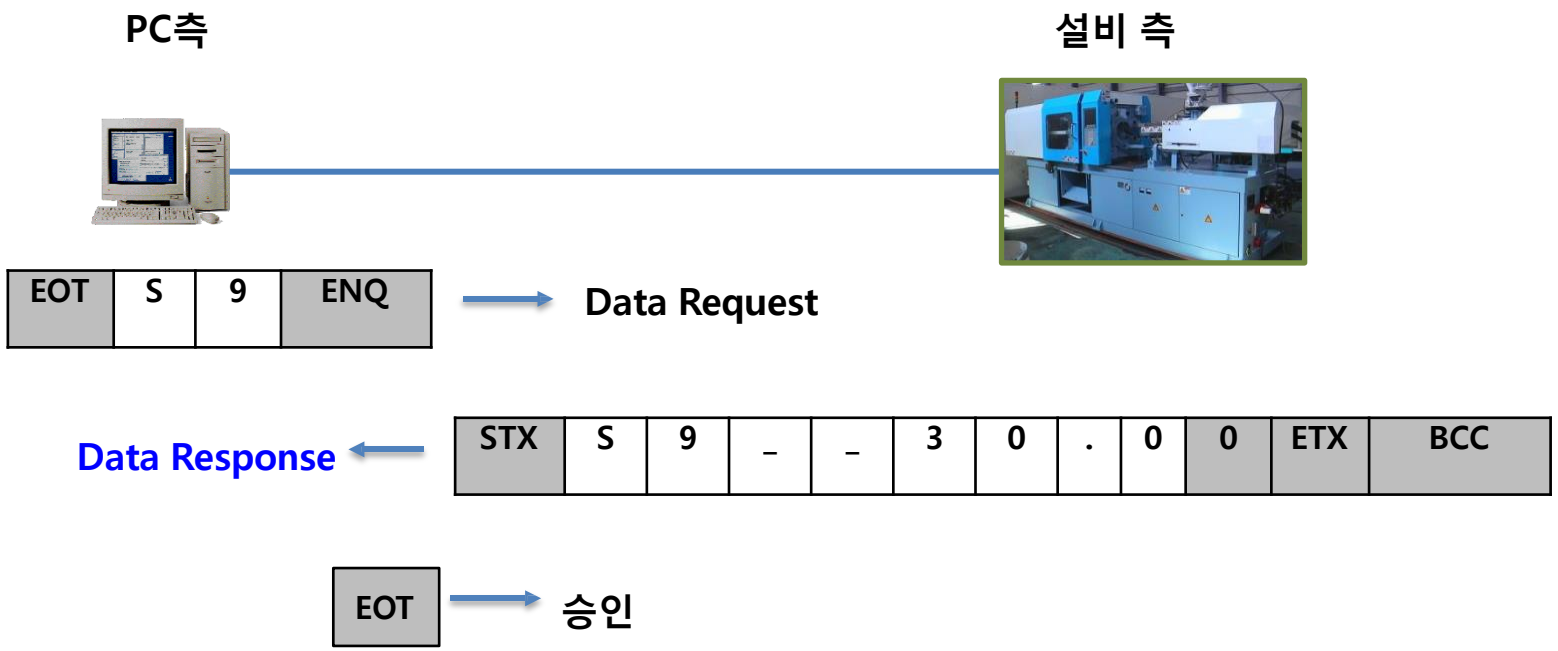
8.4. Blocking and None-Blocking 2

- **Non-Blocking Mode :통신프로그램에서 A Program이 B Program으로 부터 Data가 수신될 시간에 Data가 없으면 현재의 상태 code를 Return하고 다음 라인을 수행함.**



8.5. ASCII Protocol Communication

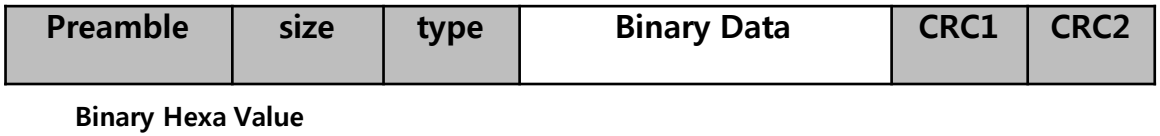
- ASCII Protocol Communication



- Data 의 전송량이 많아짐.
- Character 단위로 Operation함
- Synchronization 을 위하여 Control Character를 사용함.

8.6. Binary Protocol Communication

- **Binary Protocol Communication**



- Data 의 전송량이 작아짐.
- Bit 단위로 Operation함.
- Synchronization Preamble을 사용함

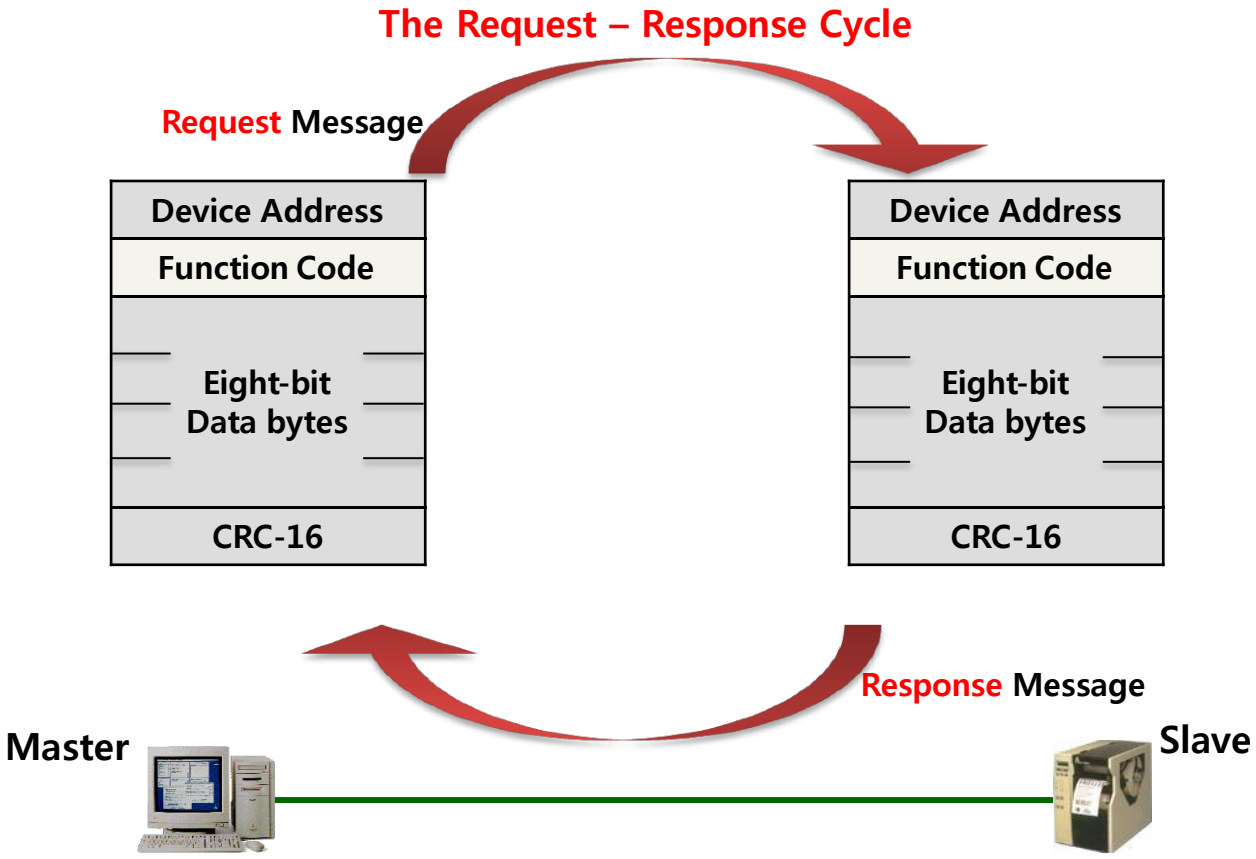
9. Modbus Protocol

9.1. Modbus Protocol Introduction

- Modbus - 1979년 미국의 Modicon 사에서 개발된 모디콘사의 전용프로토콜.
- 오래된 프로토콜로서 기본구조에서 몇 차례 버전이 향상되어 있음(Version이 여러 가지).
- 사용권에 제약 없어, 생산 현장에서 가장 많이 사용되는 통신 방식.
- 사용자 User Group이 방대하여 컨트롤러 제조업체에서 제2의 전용프로토콜로 채용
- PLC, 생산설비, 데이터수집장치 등에 적용됨.
- 안정적인 통신 기능 및 강력한 응용 성 제공.

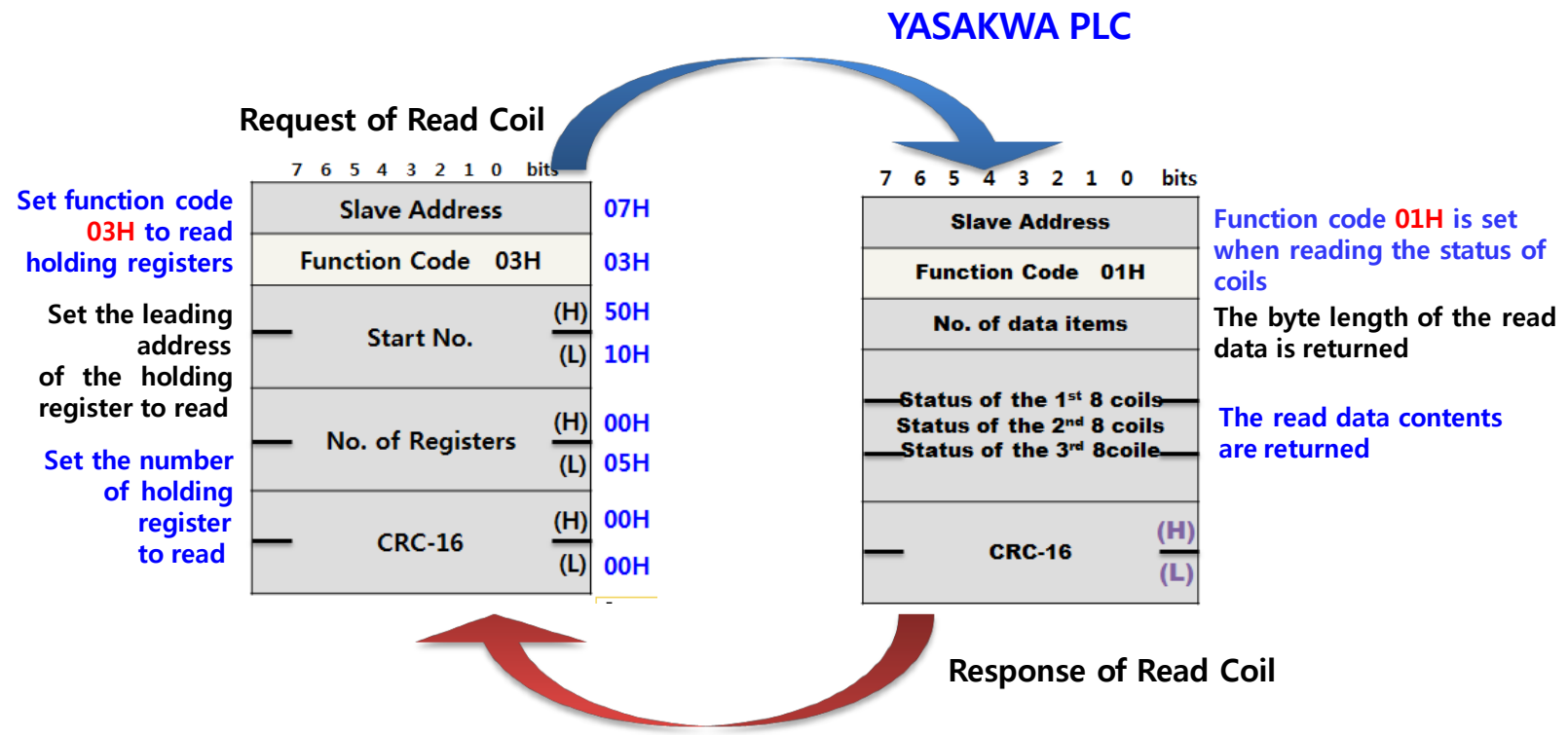
9. Modbus Protocol

9.1. Modbus Protocol Introduction 1



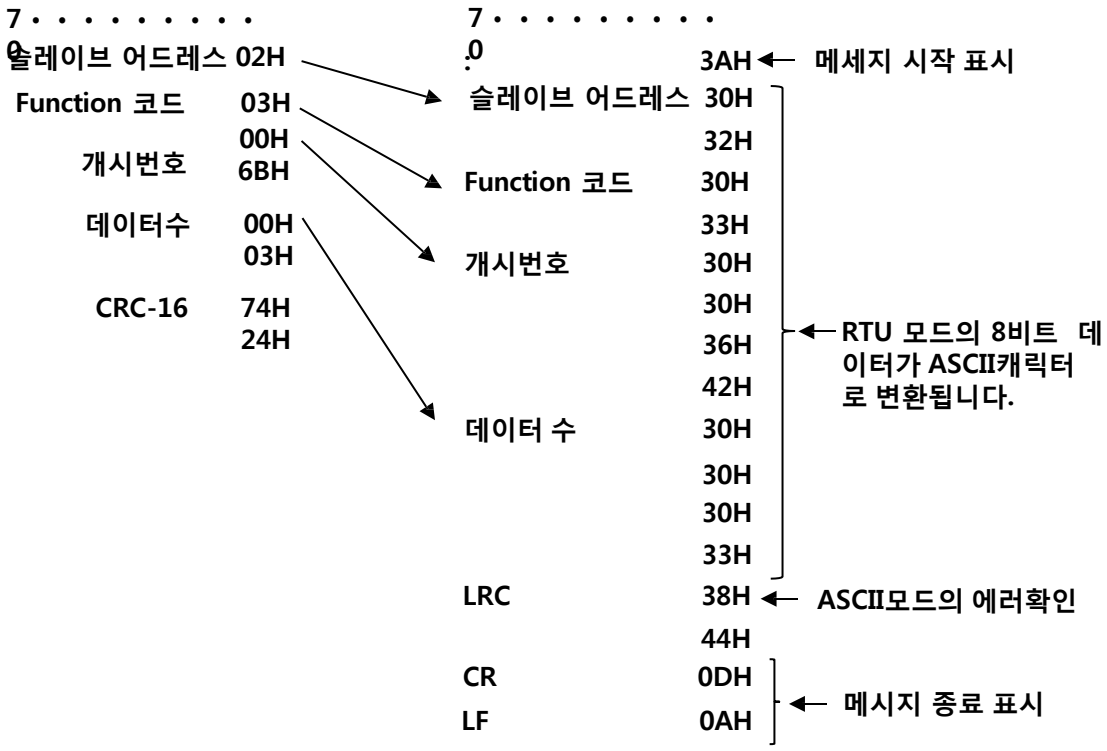
9.2. Modbus Base Protocol 종류 1

- MEMOBUS RTU Mode (CRC16 is not checked)



9.2. Modbus Base Protocol 종류 2

- MEMOBUS ASCII Mode



9. Modbus Protocol

2. Modbus Base Protocol 종류 3

- **MELSEC Protocol - 미쓰비시 PLC Protocol**

Function코드	MELSEC ACPU 공통 코맨드	기능	오므론 점수
01H/02H	WR	비트디바이스를16점단위로읽는다	32워드(512점)
03H/04H		워드디바이스를1점단위로읽는다.	64점
0FH	WW	비트단위를16점단위로쓴다.	10워드(160점)
10H		워드디바이스를1점단위로쓴다.	65점
08H	TT	루프백 테스터	254캐릭터

디바이스	ACPU 공통 코맨드 디바이스 범위	10/16진	MEMOBUS 코맨드	개시번호	레지스터번호
X	X0000~X07FF	16진	02H : 입력릴레이	0~2047	MB000000~MB00127F
Y	Y0000~Y07FF	16진	01H/0FH : 코일	0~2047	MB000000~MB00127F
M	M0000~M2047	10진	01H/0FH : 코일	2048~4095	MB001280~MB00255F
M	X9000~X9255	10진	01H/0FH : 코일	4096~4351	MB002560~MB00271F
B	B0000~B03FF	16진	01H/0FH : 코일	4352~5375	MB002720~MB00335F
F	F0000~F0255	10진	01H/0FH : 코일	5376~5631	MB003360~MB00351F
TS	TS000~TS255	10진	02H : 입력릴레이	2048~2303	MB001280~MB00143F
TC	TC000~TC255	10진	02H : 입력릴레이	2304~2559	MB001440~MB00159F
CS	CS000~CS255	10진	02H : 입력릴레이	2560~2815	MB001600~MB00175F
CC	CC000~CC255	10진	02H : 입력릴레이	2816~3071	MB001760~MB00191F

MELSEC 비트 디바이스

9. Modbus Protocol

2. Modbus Base Protocol 종류 4

- **MELSEC Protocol - 미쓰비시 PLC Protocol**

디바이스	ACPU 공통 코맨드 디바이스 범위	10/16진	MEMOBUS 코맨드	개시번호	레지스터번호
TN	TN000 ~ TN255	10진	04H : 입력 레지스터	0 ~ 255	**MW00000 ~ MW00255**
CN	CN000 ~ CN255	10진	04H : 입력 레지스터	256 ~ 511	**MW00256 ~ MW00511**
D	D0000 ~ D1023	10진	03H/10H : 유지 레지스터	0 ~ 1023	**MW00000 ~ MW01023**
D(특수)	D9000~D9255	10진	03H/10H : 유지 레지스터	1024 ~ 1279	**MW01024 ~ MW01279**
W	W0000~W03ff	16진	03H/10H : 유지 레지스터	1280 ~ 2815	**MW01280 ~ MW02303**
R	R0000~R8191	10진	03H/10H : 유지 레지스터	2816 ~ 3071	**MW02304 ~ MW10495**

MELSEC 워드 디바이스

9. Modbus Protocol

2. Modbus Base Protocol 종류 4

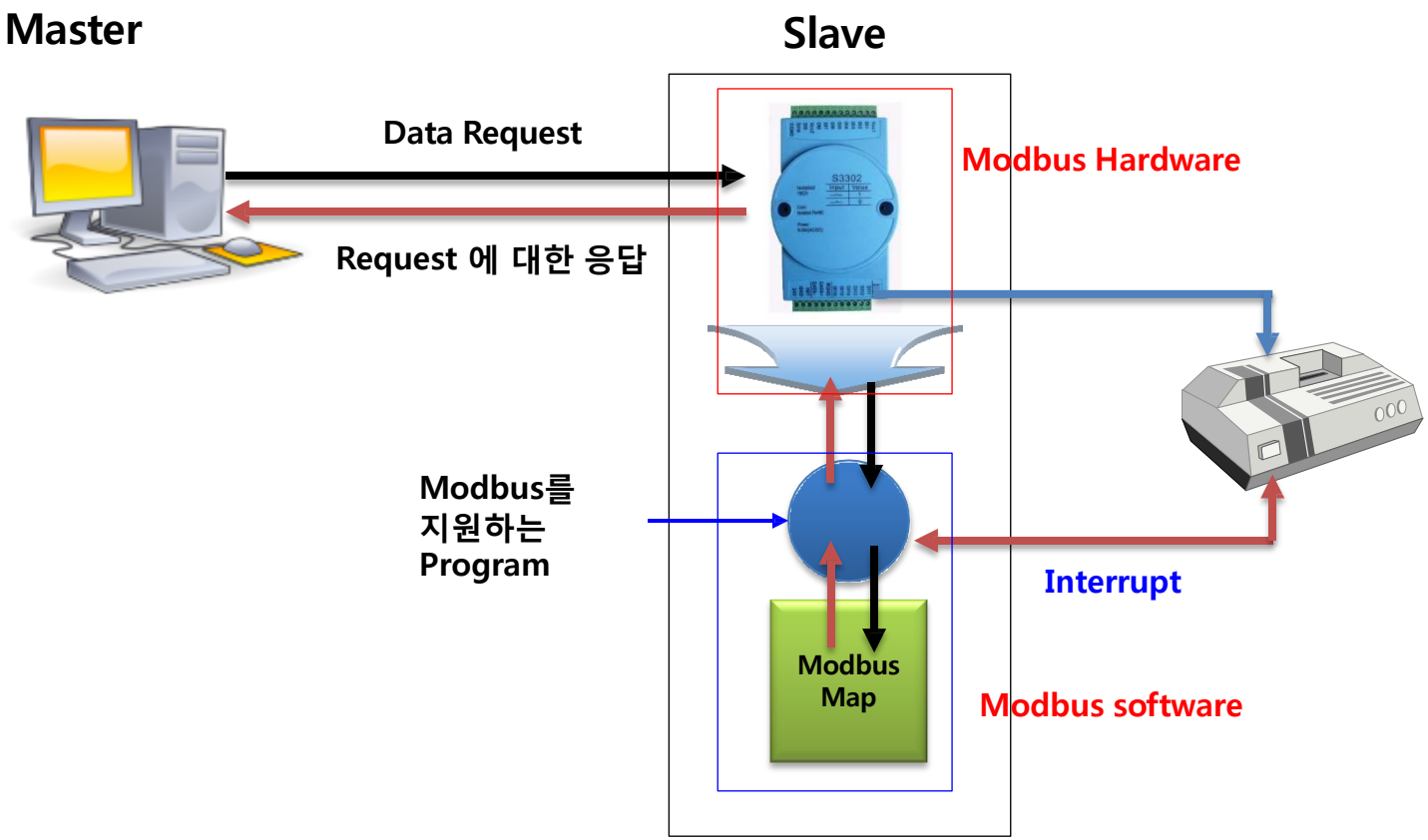
- **OMRON Protocol - OMRON PLC Protocol**

OMRON 프로토콜 사양			MEMOBUS 프로토콜 사양	
OMRON 헤더 코드	내용	점수	1회의 지령으로 엑세스 가능한 점수	Function 코드
RR	입출력 릴레이/내부 보조 릴레이/특수 보조 릴레이 읽기	256워드	125워드	01H
RD	데이터 메모리 에어리어 읽기	2000워드	125워드	03H
WR	입출력 릴레이/내부보조 릴레이/특수 보조 릴레이 쓰기	252워드	50워드	0FH
WD	데이터 메모리 에어리스 쓰기	2000워드	100워드	10H
TS	테스트	-	-	08H

OMRON 프로토콜 사양			MEMOBUS 프로토콜 사양		
명칭	채널 번호	릴레이 번호	Function 코드	개시번호	레지스터 번호
입출력 릴레이	000 ~ 039	00000 ~ 03915	01H/0FH	0 ~ 639	MB000000~ MB00039F
내부보조 릴레이	040 ~ 246	04000 ~ 24615	01H/0FH	640 ~ 3951	000400 ~ MB00246F
특수보조 릴레이	247 ~ 255	24700 ~ 25507	01H/0FH	3952 ~ 4088	MB002470 ~ MB002557
데이터 메모리	0000 ~ 9999	DM0000 ~ 9999	03H/10H	0000 ~ 9999	MW00000 ~ MW09999

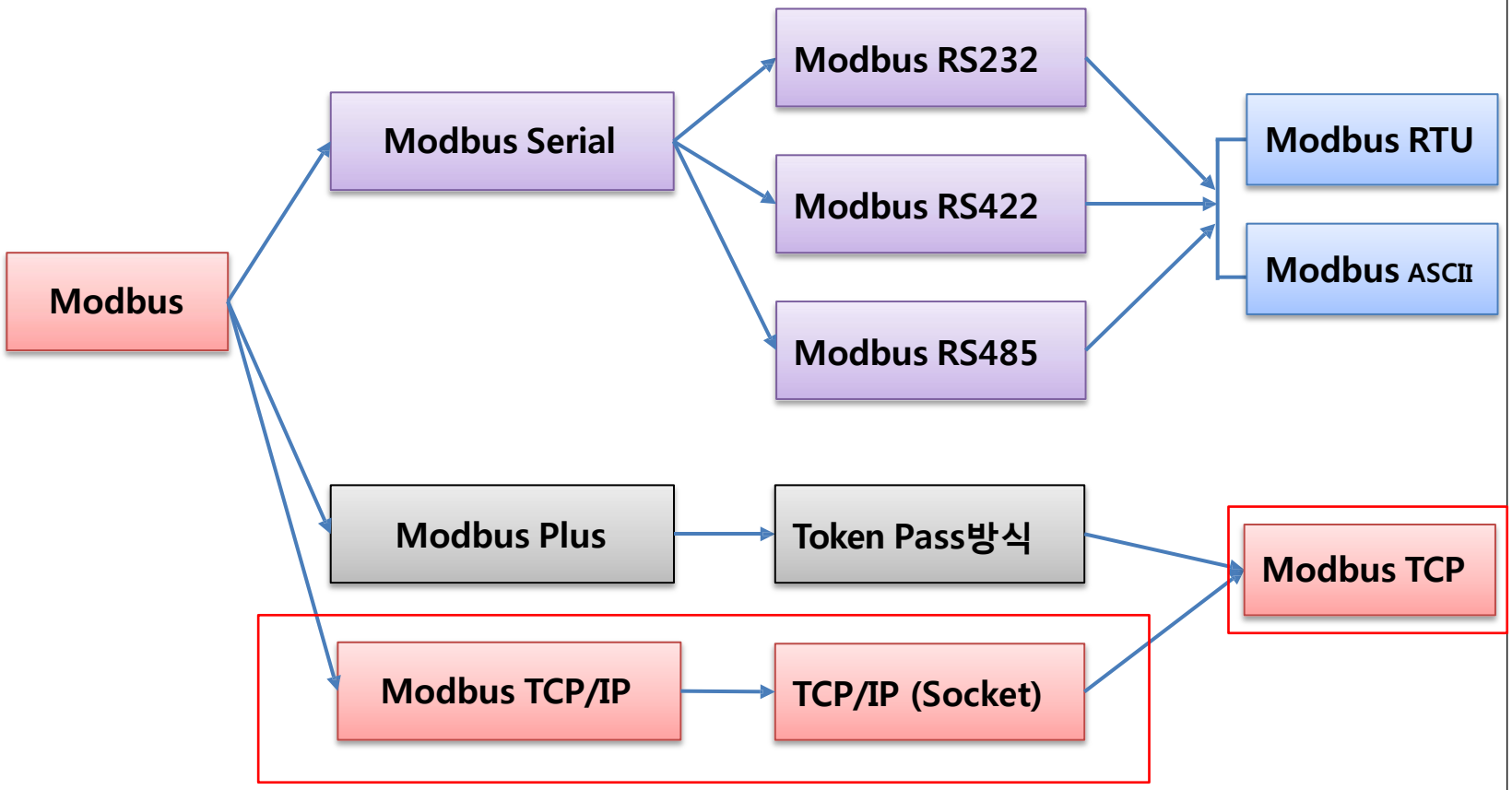
9. Modbus Protocol

9.3. Modbus Device Configuration



9. Modbus Protocol

9.4. Modbus Protocol종류

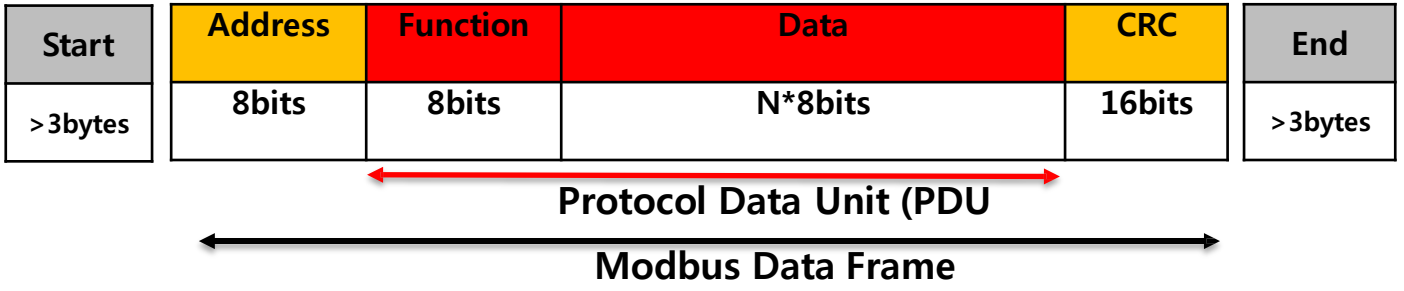


9. Modbus Protocol

9.5. Modbus RTU and ASCII 1

- Serial 통신을 이용하는 Modbus Protocol
- Modbus RTU : Data를 Binary, bit (Hexa)를 사용함.
- Modbus ASCII : Data를 ASCII code를 사용함.

Modbus RTU



Modbus ASCII

Start	Address	Function	Data	LRC	End
1char	2char	2char	0~512chars	2chars	2chars
					CR LF

9. Modbus Protocol

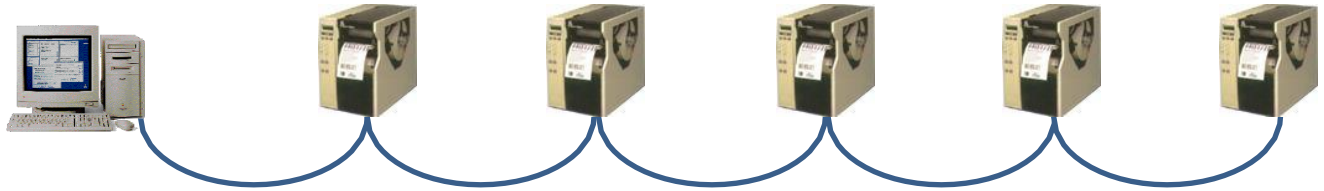
9.5. Modbus RTU and ASCII 2

QUERY			
Field Name	**Example (Hex)**	**ASCII Characters**	**RTU 8-Bit Field**
Header		: (colon)	None
Slave Address	06	0 6	0000 0110
Function	03	0 3	0000 0011
Starting Address Hi	00	0 0	0000 0000
Starting Address Lo	6B	6 B	0110 1011
No. of Registers Hi	00	0 0	0000 0000
No. of Registers Lo	03	0 3	0000 0011
Error Check		LRC(2 chars.)	CRC (16 bits)
Trailer		CR LF	None
Total Bytes :		17	8

RESPONSE			
Field Name	**Example (Hex)**	**ASCII Characters**	**RTU 8-Bit Field**
Header		: (colon)	None
Slave Address	06	0 6	0000 0110
Function	03	0 3	0000 0011
Byte Count	06	0 6	0000 0110
Data Hi	02	0 2	0000 0010
Data Lo	2B	2 B	0010 1011
Data Hi	00	0 0	0000 0000
Data Lo	00	0 0	0000 0000
Data Hi	00	0 0	0000 0000
Data Lo	63	6 3	0110 0011
Error Check		LRC(2 chars.)	CRC(16bits)
Trailer		CR LF	None
Total Bytes :		23	11

9. Modbus Protocol

9.6. Modbus Device Feature



- Master-Slave Protocol
  - Master: 한 개
  - Slave : 최대 254개 연결가능
- Master Request
  - Master: Request(Query)
  - Slave : Reply(Response)
  - Address: 1 ~ 254
- 통신속도: 1200, 2400, 4800, 9600, 19200bps, 56Kbps, 115.2kbps
- 최대 통신 가능거리: 1200m
- Termination : 150 Ohms / 0.5W

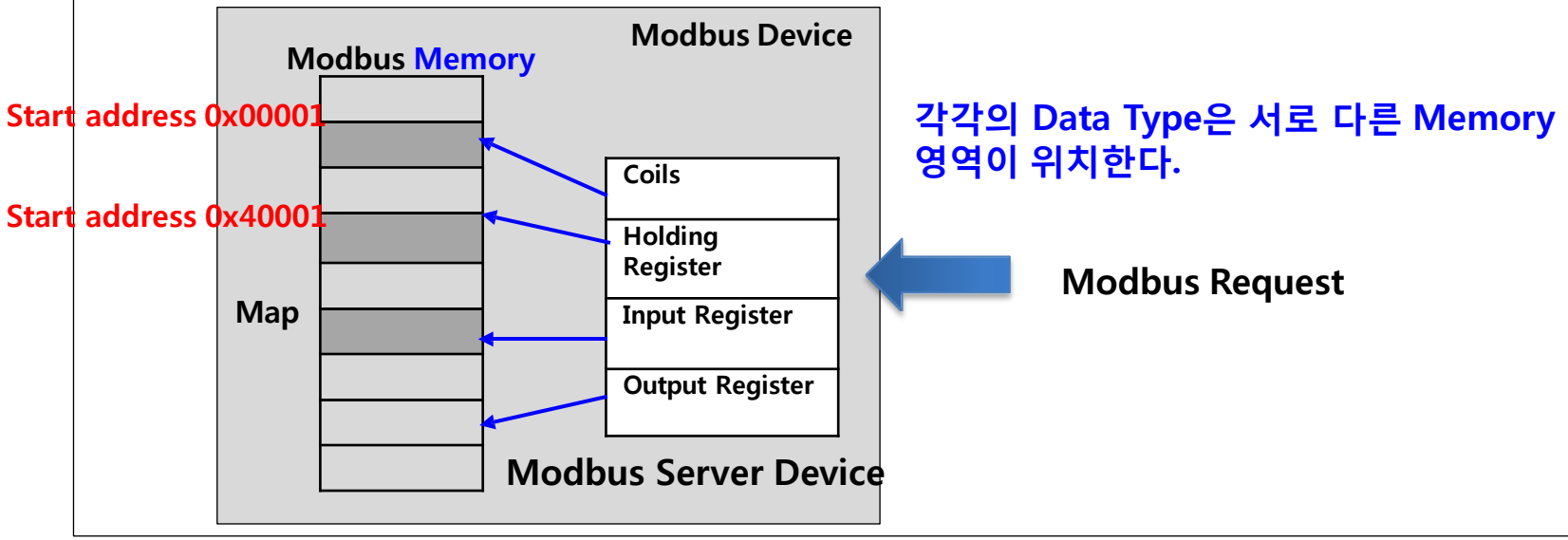
9. Modbus Protocol

9.7. Modbus Protocol Data Format

Bytes	Field Name	Description	Example Value
Bytes1	Address	RS485의 Device ID (Decimal ID 18)	0x12
Bytes2	Function Code	Read Holding Register	0x03
Bytes3	Data	읽거나 쓰게 될 (Register) Memory Map의 시작 주소의 High byes	0x00
Bytes4		시작 주소의 Low byte (Decimal 100번지)	0x60
Bytes5		읽어올 데이터 수의 High byte	0x00
Bytes6		읽어올 데이터 수의 Low byte (3개 Register를 읽겠다.	0x03
Bytes7	CRC1	CRC high bytes	0x46
Bytes8	CRC2	CRC low bytes	0xB7

9.8. Modbus Protocol Data Type and Memory

Name	Object Type	Function
Discrete Input	Single bit	Read-Only
Coils	Single bit	Read-Write
Input Registers	16 bit word	Read-Only
Holding Registers	16 bit word	Read-Write



각각의 Data Type은 서로 다른 Memory 영역이 위치한다.

9. Modbus Protocol

9.9. Modbus Function Code 1

Data Type	Function Code	비고
Read Discrete Inputs	02	Bit access
Read Coils Write Single Coil	01 05	
Write Multiple Coils	15	16bit Access
Read Input Register	04	
Read Holding Registers	03	
Write Single Register Write Multiple Register	06 16	
Read/Write Multiple Registers	23	
Read Exception Status	07	Diagnosis (Bit)

- http://www.modbus.org 참조
- 실습장비는 0x03 과 0x06 만 사용한다.

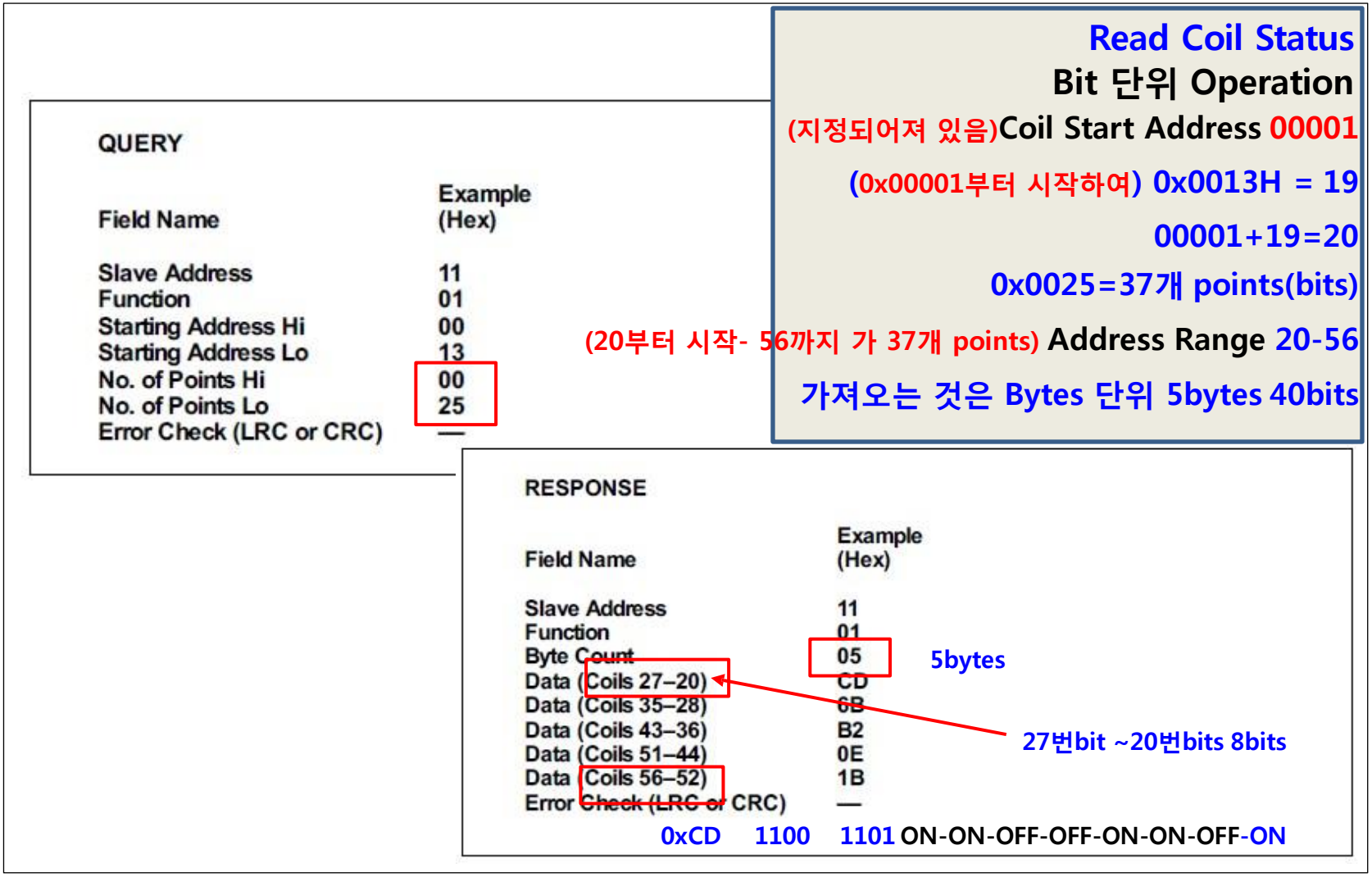
9.9. Modbus Function Code 2

Modicon Modbus Protocol Reference Guide

PI-MBUS-300 Rev. J June 1996

Code	Name	Code	Name
01	Read Coil Status	22	Mask Write 4X Register
02	Read Input Status	23	Read/Write 4X Registers
03	Read Holding Registers	24	Read FIFO Queue
04	Read Input Registers		
05	Force Single Coil		
06	Preset Single Register		
07	Read Exception Status		
08	Diagnostics (see Chapter 3		
09	Program 484		
10	Poll 484		
11	Fetch Comm. Event Ctr.		
12	Fetch Comm. Event Log		
13	Program Controller		
14	Poll Controller		
15	Force Multiple Coils		
16	Preset Multiple Registers		
17	Report Slave ID		
18	Program 884/M84		
19	Reset Comm. Link		
20	Read General Reference		
21	Write General Reference		

9. Modbus Protocol



9. Modbus Protocol

9.9. Modbus Function Code 4

QUERY

Field Name	Example (Hex)
Slave Address	11
Function	03
Starting Address Hi	00
Starting Address Lo	6B
No. of Points Hi	00
No. of Points Lo	03
Error Check (LRC or CRC)	—

RESPONSE

Field Name	Example (Hex)
Slave Address	11
Function	03
Byte Count	06
Data Hi (Register 40108)	02
Data Lo (Register 40108)	2B
Data Hi (Register 40109)	00
Data Lo (Register 40109)	00
Data Hi (Register 40110)	00
Data Lo (Register 40110)	64
Error Check (LRC or CRC)	—

6 bytes

Holding Register
Word 단위 Operation

(지정되어 있음.)Holding Register Start Address 40001

0x006B = 107

40001+107=40108

Address Range 40108-40110

3개 register (points) 6 bytes를 가져온다.

9.10. Modbus Exception Code 1

- **Function Code 07**

Controller Model	Coil	Assignment
M84, 184/384, 585, 984	1 – 8	User Defined
484	257	Battery Status
	258 – 264	User defined
884	761	Battery Status
	762	Memory Protect Status
	763	RIO Health Status
	764 – 768	User defined

9.10. Modbus Exception Code 2

- **Function Code 07**

QUERY	
Field Name	Example (Hex)
Slave Address	11
Function	07
Error Check (LRC or CRC)	—

RESPONSE	
Field Name	Example (Hex)
Slave Address	11
Function	07
Coil Data	6D
Error Check (LRC or CRC)	—

0x6D 0110 1101 OFF-ON-ON-OFF-ON-ON-OFF-ON

264, 263, 262, 261, 260, 259, 258, 257

Controller 484의 coil 264 ~ 257 (8bit) 이 상태임.

257이 ON 이므로 배터리 상태가 OK 를 나타냄.

9.11. Error Detect 1

- **Parity Bit**

  None: None Parity bit.

  Odd: Character에 Parity bit (0 혹은 1)를 추가하여 항상 1의 개수가 odd가 되게 함.

  Even: Character에 Parity bit (0 혹은 1)를 추가하여 항상 1의 개수가 even이 되게 함.

  Mark: Parity bit가 항상 1 임.(지금은 사용되지 않음)

  Space: Parity bit가 항상 0 임.(지금은 사용되지 않음)

  장점: 간단한 Error Detect 방법

  단점: 한꺼번에 짝수개의 bits 가 Error가 발생하면 Error를 검출하지 못함.

9.11. Error Detect 2

- Check Sum

송신측: 보내려는 데이터의 SUM 에 2의 보수를 취하여 얻은 Chen Sum Code를 함께 보냄.

수신측: 받은 데이터와 Check Sum Code를 모두 더하여 '0' 이 아니면 Error임.

송신측 : 0x05, 0x6F, 0x35, 0x81

```
   0x0000 0101 (0x05)
   0x0110 1111 (0x6F)
+  0x0111 0100
   0x0011 0101 (0x35)
+  0x1010 1001
   0x1000 0001 (0x81)
+  0x0010 1010 (0x2A)
   0x1101 0101  (1의 보수)
+             1  (2의 보수)
   0x1101 0110  Check Sum Code (0x2A + Check Sum=0)
```

수신측 : 0x05, 0x6F, 0x35, 0x81

```
   0x0000 0101 (0x05)
   0x0110 1111 (0x6F)
+  0x0111 0100
   0x0011 0101 (0x35)
+  0x1010 1001
   0x1000 0001 (0x81)
+  0x0010 1010 (0x2A)
   0x1101 0110  Check Sum Code
+0x0000  0000(0x00)  No Error
```

9.11. Error Detect 3

- Check Sum 문제점 : 2bit가 Error 일 때 검출하지 못함.

수신측 : 0x05+1, 0x6F, 0x35-1, 0x81

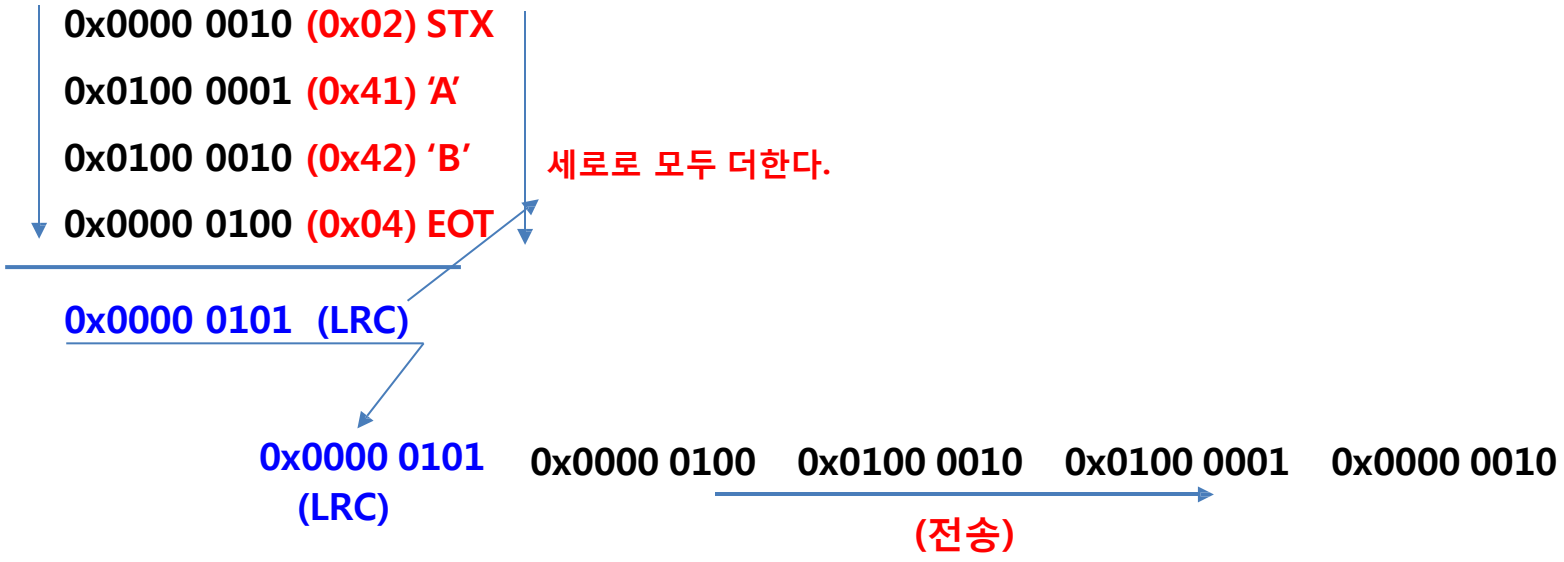
```
    0x0000 0110 (0x06)
    0x0110 1111 (0x6F)
+   0x0111 0101
    0x0011 0100 (0x34)
+   0x1010 1001
    0x1000 0001 (0x81)
+   0x0010 1010 (0x2A)
    0x1101 0110  Check Sum Code
+ 0x0000  0000(0x00)  No Error
```

9.11. Error Detect 4

- LRC(Longitudinal Redundancy Check: 세로중복검사)

LRC(Longitudinal Redundancy Check) Error Check 방법은 송신 측 LRC와 수신 측 LRC를 서로 비교하여 같으면 Error가 없는 것으로 판단하고 다르면 Error가 존재하는 것으로 판단하여 Error를 검출하는 Error Check 방법.

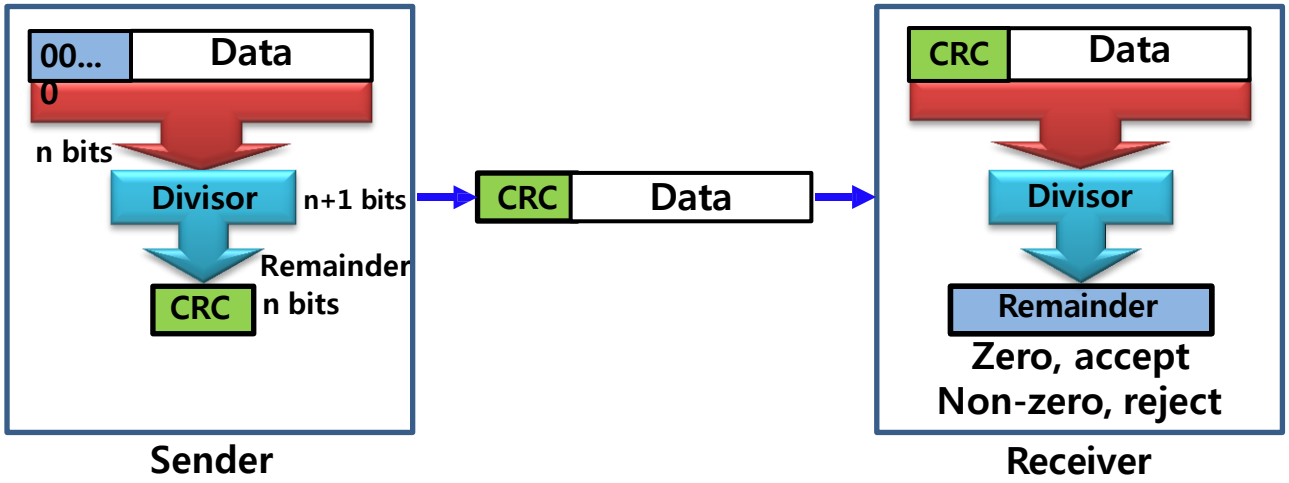
Modbus ASCII 에서 사용하는 Error Detect방법



9.11. Error Detect 5

- **CRC (Cyclic Redundancy Check)**

  CRC(Cyclic Redundancy Check)Error Check방법은 K비트의 메시지에 N비트의 F
  CS(Frame Check Sequence)를 더하여 미리 정한 숫자로 나누어 나머지가"0"이면
  Error가 없는 것으로 판단하고 나머지가 존재하면 Error가 있는 것으로 판단하는 E
  rror Check 방법.



9.11. Error Detect 6

- CRC (Cyclic Redundancy Check) Polynomial

CRC-12

$$x^{12}+x^{11}+x^{3}+x+1$$

CRC-16

$$x^{16}+x^{15}+x^{2}+1$$

CRC-ITU

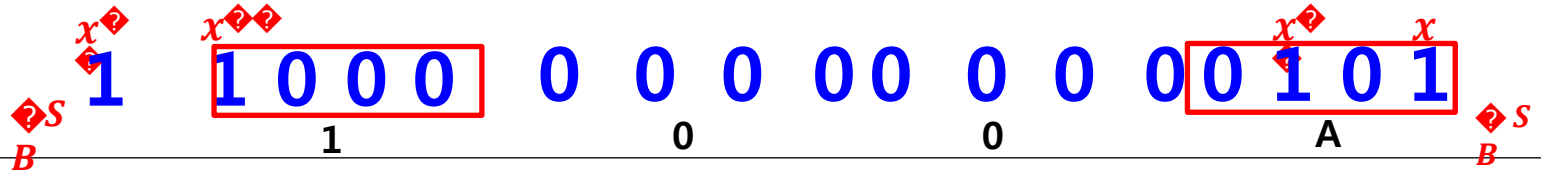
$$x^{16}+x^{12}+x^{5}+1$$

CRC-32

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$$

Name	Polynomial	Application
CRC-8	$x^{8}+x^{2}+x^{1}+1$	ATM Header
CRC-10	$x^{10}+x^{9}+x^{5}+\ldots\ldots 1$	ATM기계
CRC-16	$x^{16}+x^{15}+x^{2}+1$	Modbus Polynomial
CRC-32	$x^{32}+x^{26}+x^{23}+\ldots\ldots 1$	LANs

0xA001



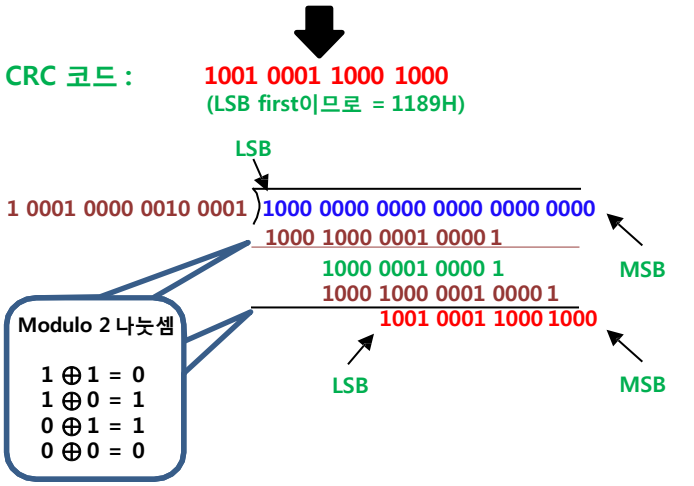
9.11. Error Detect 6

- **CRC (Cyclic Redundancy Check)**

생성 다항식에는 $x^{16}+x^{12}+x^5+1$을 사용한다.
이 연산 결과에서 얻어지는 나머지를 CRC 코드라 부른다.

LSB first로 01H를 송/수신 할 경우의 CRC 연산

생성 다항식 : 1 0001 0000 0010 0001($X^{16}+X^{12}+X^5+1$)



[주] CRC 연산 회로를 사용하여 MSB first시의
CRC 코드를 생성할 경우에는 통신 데이터와의 LSB측과 MSB 축을
바꿔서 CRC 연산을 실행한다

9. Modbus Protocol

9.12. Modbus Map

Modbus Register list : Note : *means default value

Address	Bytes	Value range		Description	Property
		Min	Max		
0 – 3	4	1	4294967295	Serial number, unique for each product	R
4 – 5	2	100	65535	Firmware version number	R
6	1	1	254	Device address	R/W
7	2	3301	3301	Product model	R
8	1	1	255	Hardware version	R
9	2	12	1152	Baudrate setting Value – Baudrate 12 – 1200 24 – 2400 48 – 4800 96 – 9600 192* – 19200 384 – 38400 576 – 57600 1152 – 115200 For example : write to register 9 to set the baudrate 9600.	R/W
10 – 99				Gain range setting	R
100	1	0	255	Status For digital input channel 1 through 8, 0=contact active, 1=contact inactive. Bit0 correspond to channel1, bit1 correspond to channel 2 and so on.	R
101	1	0	255	Open-collector output, 0=active, 1=inactive. Bit0 correspond to output1, bit1 correspond to channel 2 etc	R/w

9. Modbus Protocol

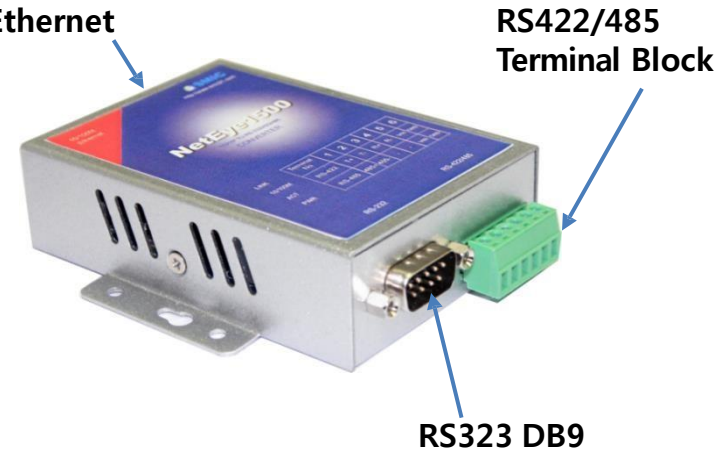
9.12. Modbus Map - continue

101	1	0	255	Open-collector output, 0=active, 1=inactive. Bit0 correspond to output 1, bit1 correspond to channel 2 etc	R/W
102	2	0	65535	High word for counter input 1	R/W
103	2	0	65535	Low word for counter input 1, value of counter = (102)*65536 + (103)	R/W
104	2	0	65535	High word for counter input 2	R/W
105	2	0	65535	Low word for counter input 2, value of counter = (104)*65536 + (105)	R/W
106	2	0	65535	High word for counter input 3	R/W
107	2	0	65535	Low word for counter input 3, value of counter = (106)*65536 + (107)	R/W
108	2	0	65535	High word for counter input 4	R/W
109	2	0	65535	Low word for counter input 4, value of counter = (108)*65536 + (109)	R/W
110	2	0	65535	High word for counter input 5	R/W
111	2	0	65535	Low word for counter input 5, value of counter = (110)*65536 + (111)	R/W
112	2	0	65535	High word for counter input 6	R/W
113	2	0	65535	Low word for counter input 6, value of counter = (112)*65536 + (113)	R/W
114	2	0	65535	High word for counter input 7	R/W
115	2	0	65535	Low word for counter input 7, value of counter = (114)*65536 + (115)	R/W
116	2	0	65535	High word for counter input 8	R/W
117	2	0	65535	Low word for counter input 8, value of counter = (116)*65536 + (117)	R/W
118	1	1	100	Respond delay for serial communication, and default is 10ms	R/W
119	1	1	255	Filter time for counter input, the units is 10us andthe default is 200us	R/W
121	1	0	1	Input status selection. 0=ON/OFF, 1=OFF/ON, default is ON/OFF	R/W

10. Modbus Protocol 실습장비 구성

10.1. Modbus 실습장비 구성 내용.

- NetEye1500 (Ethernet to Serial Converter)
  - IEEE802.3 LAN 10/100M
  - RS232, RS422, RS485 지원



- S3301 (DI/DO/Counter 데이터 수집장치)
  - 8 Digital Input
  - 6 Digital Output
  - 8 Counter
  - Modbus Map 탑재
  - RS485 Interface

10. Modbus Protocol 실습장비 구성

10.2. Modbus 실습장비 결선도.

- Modbus가 탑재된 S3301
- Digital Input : 접점상태가 연결되었는지 개방되어 있는지 확인.
- Digital Output : Open Collector Output으로 Lamp(혹은 Relay)를 켜거나 끄는 제어를 한다.

Modbus DIDO Device　　LAN to Serial Converter

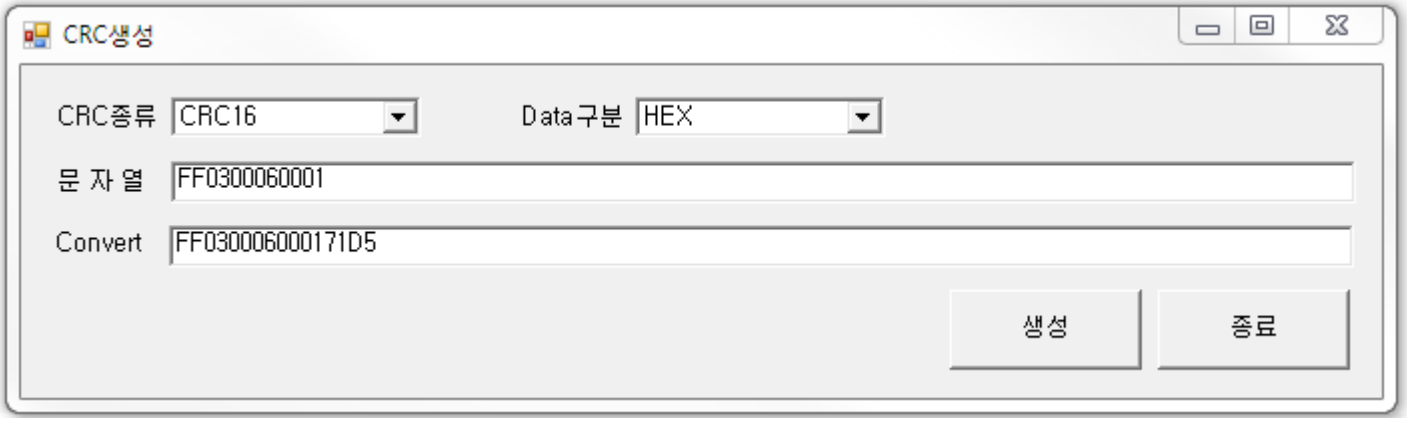
입력 및 출력 simulator　　RS485

10. Modbus Protocol 실습장비 구성

10.3. Modbus 실습장비 Software 1

- Program Coding전에 Modbus Protocol을 사용하는 장비의 작동상황을 확인한다.
- CRCCreate Program : 16bit CRC을 생성하여주는 Software.
- NetEye Simulator Ver2.53 : Modbus RTU 를 위한 Binary(Hexa) Modbus Protocol Data Unit를 전송 하고 그 응답을 수신하는 TCP/IP Socket Simulator Software.

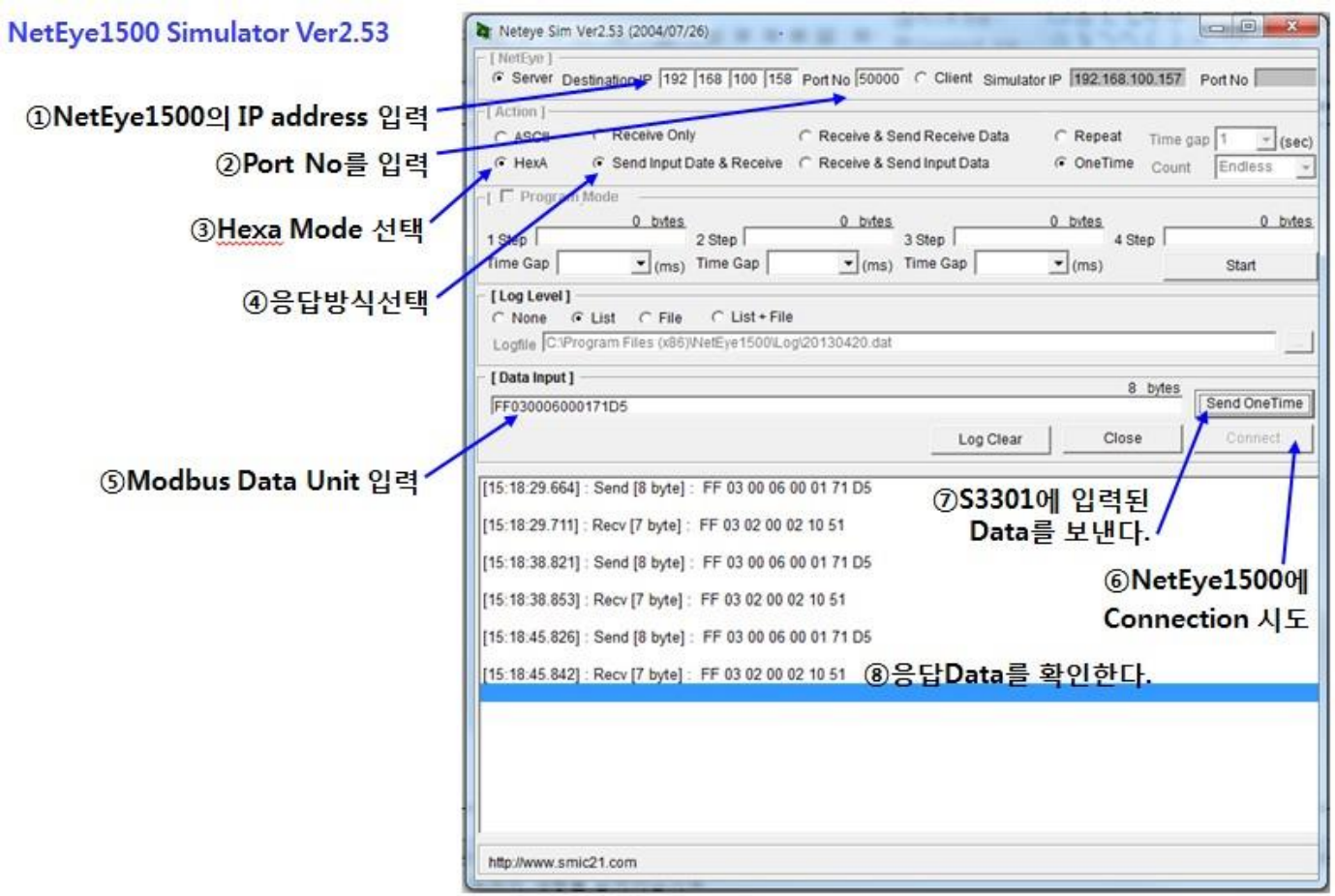
CRCCreate Program



- 사용방법 – 문자열 Field에 해당 Hexa Modbus Protocol Data unit을 넣고 생성 버튼을 누르면 CRC CheckSum 2bytes가 Convert Field에 생성된다.

10. Modbus Protocol 실습장비 구성

10.3. Modbus 실습장비 Software 2



11. Modbus Protocol 실습 (Software)

11.1. Endian의 이해 1

- 서로 다른 디바이스 들이 통신을 하고자 할 때 데이터에 대한 모드전환이 필요하다.
  데이터의 저장은 바이트 단위로 이루어진다.
- **Big Endian** – RISC 기반과 모토롤라 계열의 프로세서가 사용하는 방법 높은 자리 수 를 먼저 저장한다.
- **Little Endian** – Intel 열의 프로세서에서 사용하는 방법 낮은 자리 수 부터 저장

일반 PC
Intel 계열 CPU
Little Endian

PLC 등 설비
MPC 계열 CPU
Big Endian

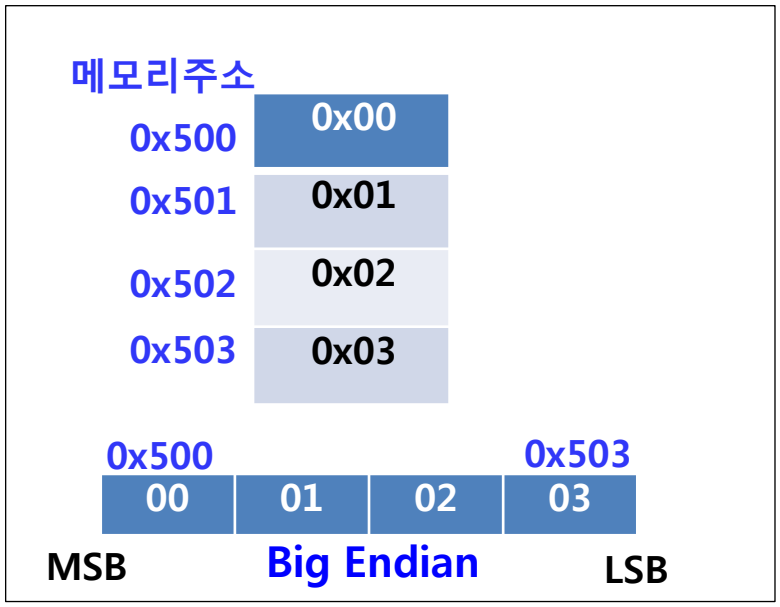
Digital Input /Output
Big Endian

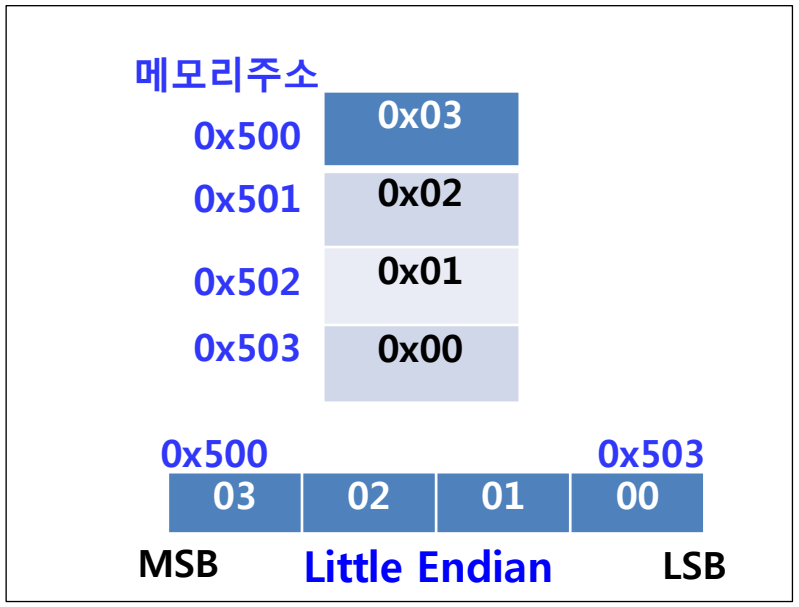
11. Modbus Protocol 실습 (Software)

1. Endian의 이해 2

- MSB – Most Significant Bit
- LSB – Least Significant Bit

32bits 정수 값 0x00010203를 저장해보자



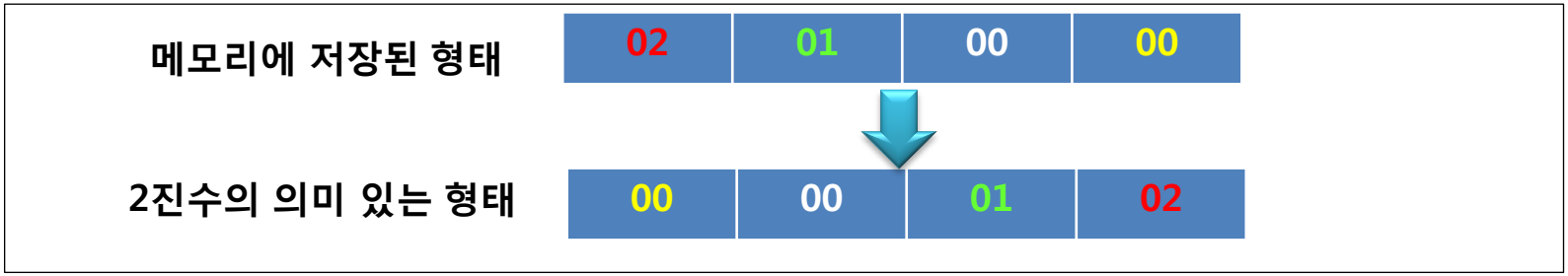


11. Modbus Protocol 실습 (Software)

11.1. Endian의 이해 3

- 인텔 계열인 일반적인 PC에서 Little Endian 데이터가 메모리에 저장 되는 방식을 이해 한다.

float f; 와 integer i; - 각 각 4bytes 들이 메모리에 저장되는 내용을 본다.



float f=1.000000; (&f) 00 0080 3F -> 3F 80 00 00

0011 1111 1000 0000 0000 0000 0000 0000

int i=258; (&i) 02 01 00 00 -> 00 00 01 02

0000 0000 0000 0000 0000 0001 0000 0010 = 258

256 2

11. Modbus Protocol 실습 (Software)

11.2. Modbus Device ID 읽기.

ADDR	내용	Value(HEX)	비고
0	고정	ff	
1	Read/Write	03	Read Mode
2	ADDR-H	00	S3301 ID 가 들어 있는 Address Register 6번지
3	ADDR-L	06	
4	Data Count -H	00	Reading Register의 개수 1개의 Register- 1 word (2bytes)
5	Data Count -L	01	
6	CRC-1	71	계산식
7	CRC-2	D5	계산식

Device ID 확인 방법 :
PC -> S3301 Request

ADDR	내용	Value(HEX)	비고
0	고정	ff	
1	Read/Write	03	Read Mode 대한 응답
2	Count	02	Data Count 2 bytes
3	고정	00	
4	Device Address	초기치 : FE(254)	S3301의 ADDR (Default 0xFE)
5	CRC-1	10	계산식
6	CRC-2	10	계산식

Device ID 확인 방법 :
PC <- S3301 응답

11. Modbus Protocol 실습 (Software)

11.3. Modbus Device ID 변경.

ADDR	내용	Value(HEX)	비고
0	ID ADDR	FE	현재 S3301의 ID
1	Write	06	Write Mode
2	ADDR-H	00	NE3301의 Register ADDR
3	ADDR-L	06	
4	변경할 ADDR-H	00	
5	변경할 ADDR-L	02	NES3301의 변경할 ID (01~FE)
6	CRC-1	BC	계산식
7	CRC-2	04	계산식

Device ID 변경 방법 :
PC -> S3301 변경요청

ADDR	내용	Value(HEX)	비고
0	변경 전 ID	FE	
1	Write	06	Write Mode 대한 응답
2	ADDR-H	00	Data Count (ADDR9~10)
3	ADDR-L	06	
4	변경된 ADDR-H	00	S3301의 ADDR
	변경된 ADDR-L	02	
5	CRC-1	FC	계산식
6	CRC-2	05	계산식

Device ID 확인 방법 :
PC <- S3301 응답

● 6번지 ID 다시 읽어보기

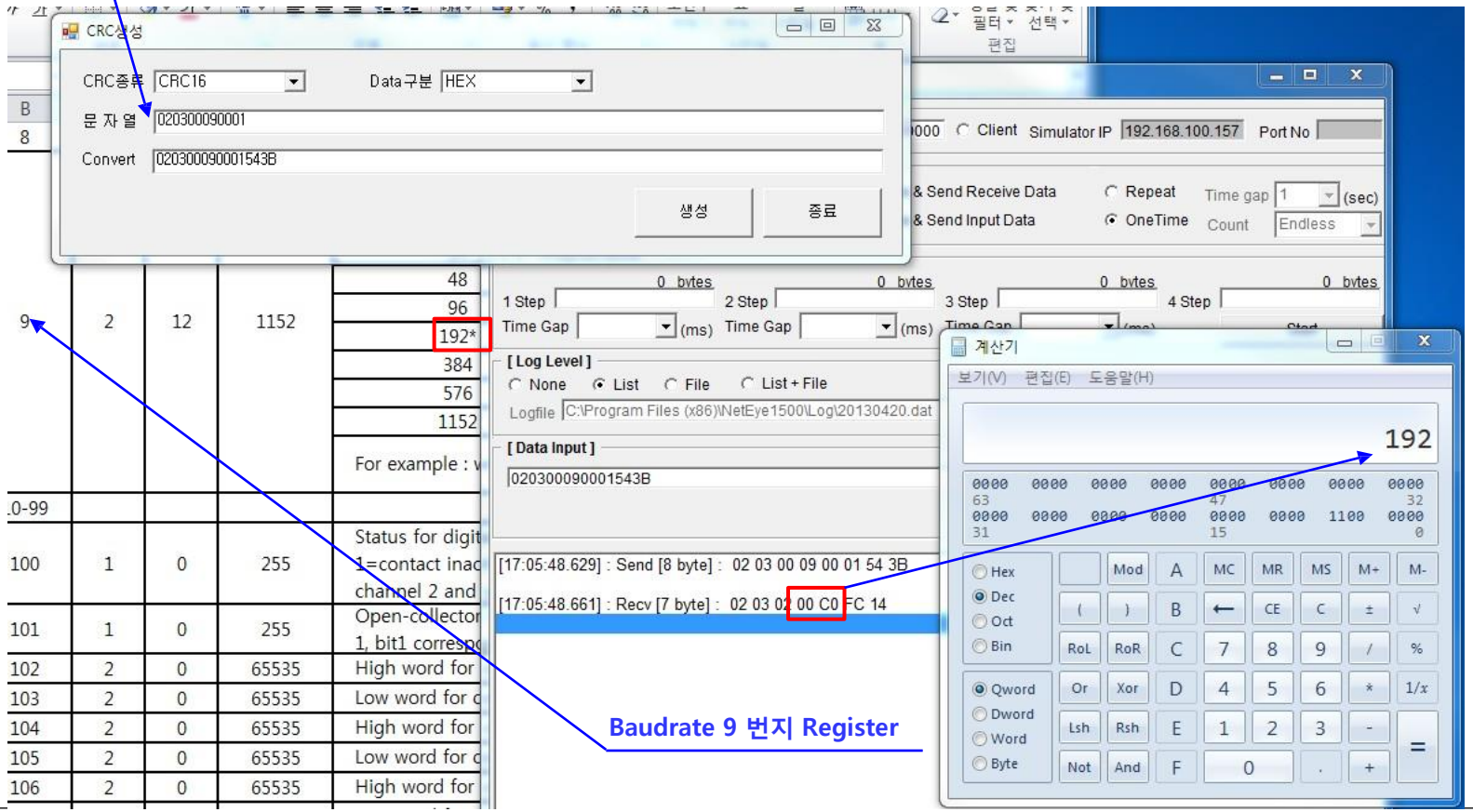
Send [8 byte] : FF 03 00 06 00 01 71 D5

Recv [7 byte] : FF 03 02 00 02 10 51

11. Modbus Protocol 실습 (Software)

11.4. Modbus Baudrate 읽기 1

02 03 0009 0001 54 3B



9	2	12	1152	48 96 192* 384 576 1152 For example :
0-99				
100	1	0	255	Status for digit 1=contact inac channel 2 and
101	1	0	255	Open-collector 1, bit1 corresp
102	2	0	65535	High word for
103	2	0	65535	Low word for c
104	2	0	65535	High word for
105	2	0	65535	Low word for c
106	2	0	65535	High word for

11.5. Modbus Product Model/Serial Number 읽기

- **Product Model 읽어보기**

02 03 0007 0001

Send 02 03 00 07 00 01 35 F8

Recv 02 03 02 C E5 38 CF

0C E5 = 3301

- **Serial Number 읽어보기**

Send 02 03 00 00 00 02 C4 38

Recv 02 03 04 00 01 00 D7 D8 AD

= 65751

11. Modbus Protocol 실습 (Software)

11.6. Modbus DI 8 channel 읽기

- **Digital Input 100번지(0x64) 1개 register 읽어보기**

02 03 0064 0001

[13:51:41.822] : Send [8 byte] : 02 03 00 64 00 01 C5 E6

[13:51:41.844] : Recv [7 byte] : 02 03 02 00 00 FC 44

00 00 = 0000 0000 : 8개의 channel 이 모두 0 (open되어져 있다)

- **Digital Input 1번 ch short시킨 후 상태 읽어보기**

02 03 0064 0001

[13:54:35.405] : Send [8 byte] : 02 03 00 64 00 01 C5 E6

[13:54:35.428] : Recv [7 byte] : 02 03 02 00 01 3D 84

00 01 = 0000 0001 : 1개의 channel Short

11.7. Modbus D0 6 channel 읽기

- **Digital Output 101번지(0x65) 1개 register 읽어보기**

02 03 0065 0001

[14:03:17.574] : Send [8 byte] : 02 03 00 65 00 01 94 26

[14:03:17.605] : Recv [7 byte] : 02 03 02 00 FF BC 04

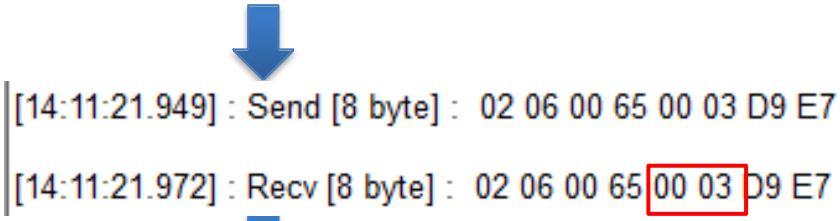
00 FF = 1111 1111: 6개의 channel 이 모두 1 Set 되어져 있다.(상위 2 bit는 쓰지 않음)

11. Modbus Protocol 실습 (Software)

11.8. Modbus D0 6 channel 쓰기

- **Digital Output 101번지(0x65) 에서 하위 2 Channel을 1로 Set하기**

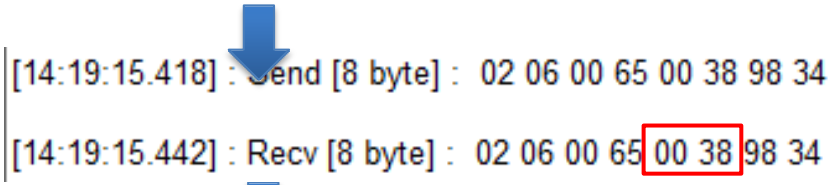
02 06 0065 0003



00 03 = 0000 0011 : 6개의 channel 중 하위 2개 Channel이 1로 Set 되었음.

- **Digital Output 상위 3 Channel을 1로 Set하고 하위 3 Channel은 0 으로 Set 하기**

02 03 0065 0038



00 38 = 0011 1000 : 상위 3 channel 1 로 Set됨.

11. Modbus Protocol 실습 (Software)

11.9. Modbus Counter 읽기

- **Counter 102번지(0x66) 에서 2 개 Register (2 words : 4 bytes 읽기) : integer 4bytes**

02 03 0066 0002

[14:28:19.110] : Send [8 byte] : 02 03 00 66 00 02 24 27

[14:28:19.136] : Recv [9 byte] : 02 03 04 00 00 00 04 C8 F0

00 00 00 04 = Integer 4 bytes에 Counter값은 4 임.

- **Counter값을 2 회 올린 후 그 값을 읽어 본다. (Button 2 회 누름)**

02 03 0066 0002

[14:28:30.271] : Send [8 byte] : 02 03 00 66 00 02 24 27

[14:28:30.303] : Recv [9 byte] : 02 03 04 00 00 00 06 49 31

00 00 00 06 = Integer 4 bytes에 Counter값은 6 으로 2 증가되었음.

11. Modbus Protocol 실습 (Software)

11.10. Modbus Counter Clear

- Counter 102번지(0x66) 에서 103, 104, 105 번지 4 bytes에 모두 0 을 쓴다.

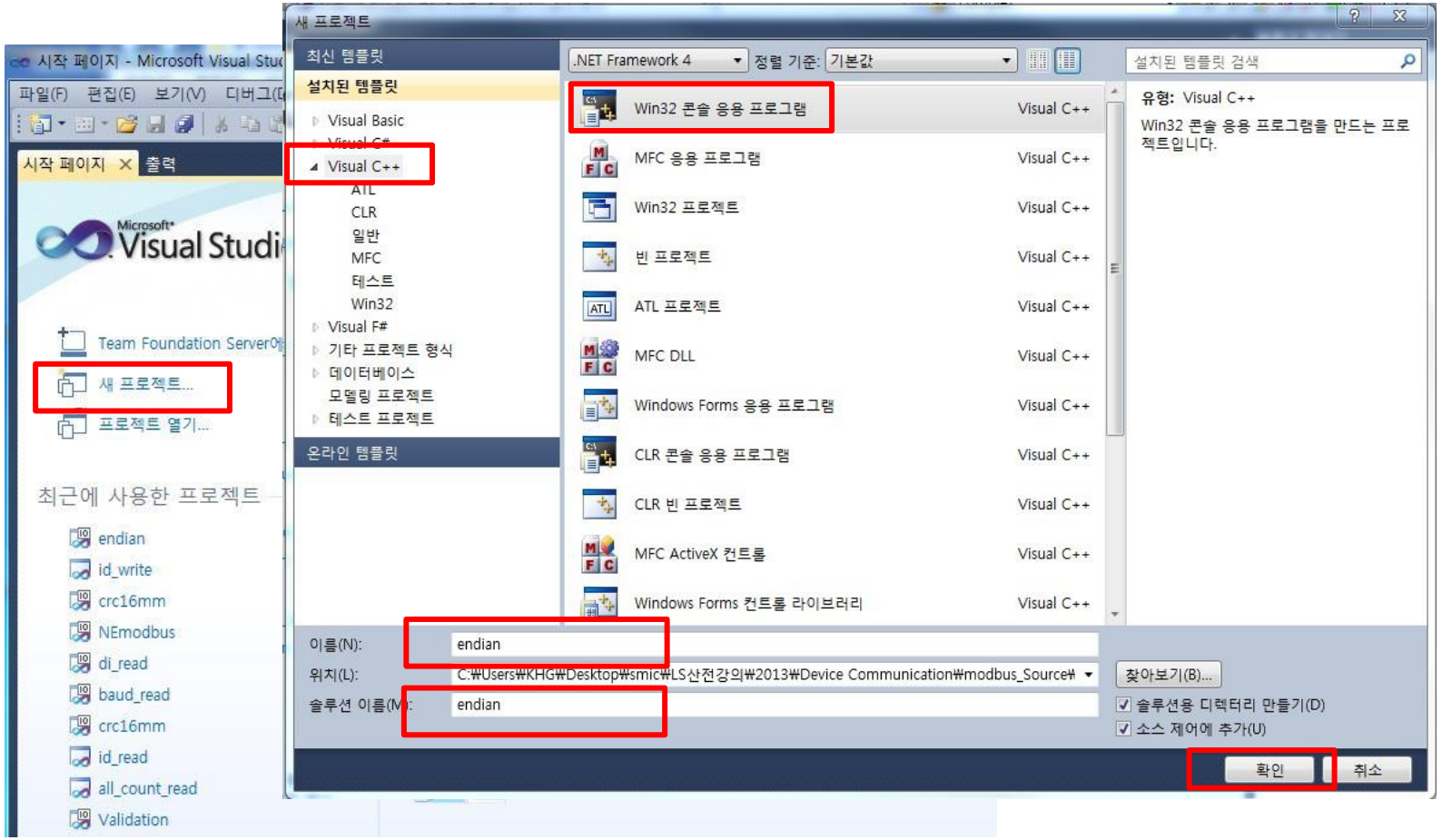
```
[14:43:08.120] : Send [8 byte] :  02 06 00 66 00 00 69 E6
[14:43:08.143] : Recv [8 byte] :  02 06 00 66 00 00 69 E6
[14:43:30.256] : Send [8 byte] :  02 06 00 67 00 00 38 26
[14:43:30.278] : Recv [8 byte] :  02 06 00 67 00 00 38 26
[14:43:51.256] : Send [8 byte] :  02 06 00 68 00 00 08 25
[14:43:51.279] : Recv [8 byte] :  02 06 00 68 00 00 08 25
[14:44:22.752] : Send [8 byte] :  02 06 00 69 00 00 59 E5
[14:44:22.774] : Recv [8 byte] :  02 06 00 69 00 00 59 E5
```

- Counter 102번지(0x66) 에서 2 register를 다시 읽어보면 0으로 Clear되어 있음.

```
[14:44:42.207] : Send [8 byte] :  02 03 00 66 00 02 24 27
[14:44:42.232] : Recv [9 byte] :  02 03 04 00 00 00 00 C9 33
```

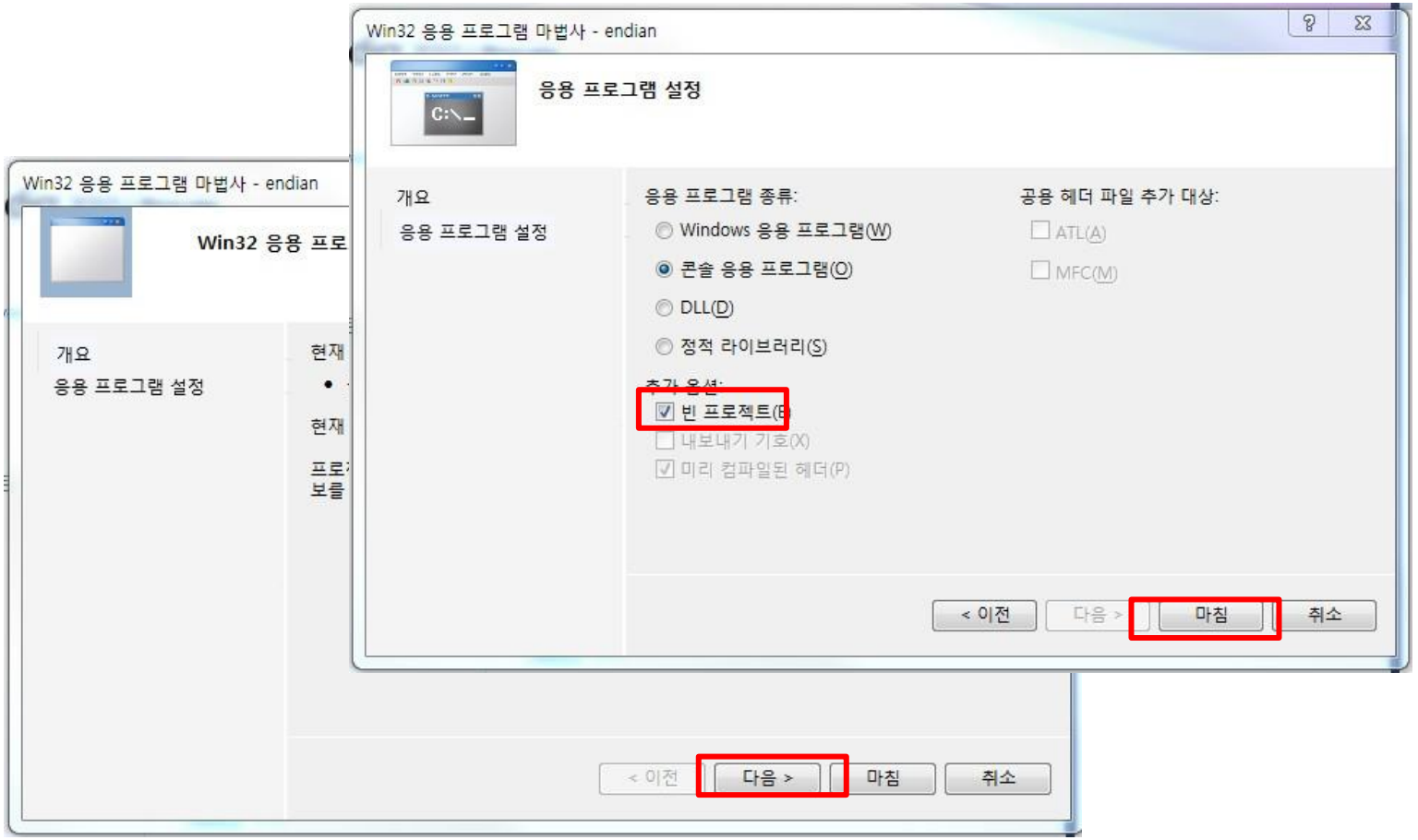
12. Visual Studio 개발환경

12.1. 새 프로젝트 만들기 1



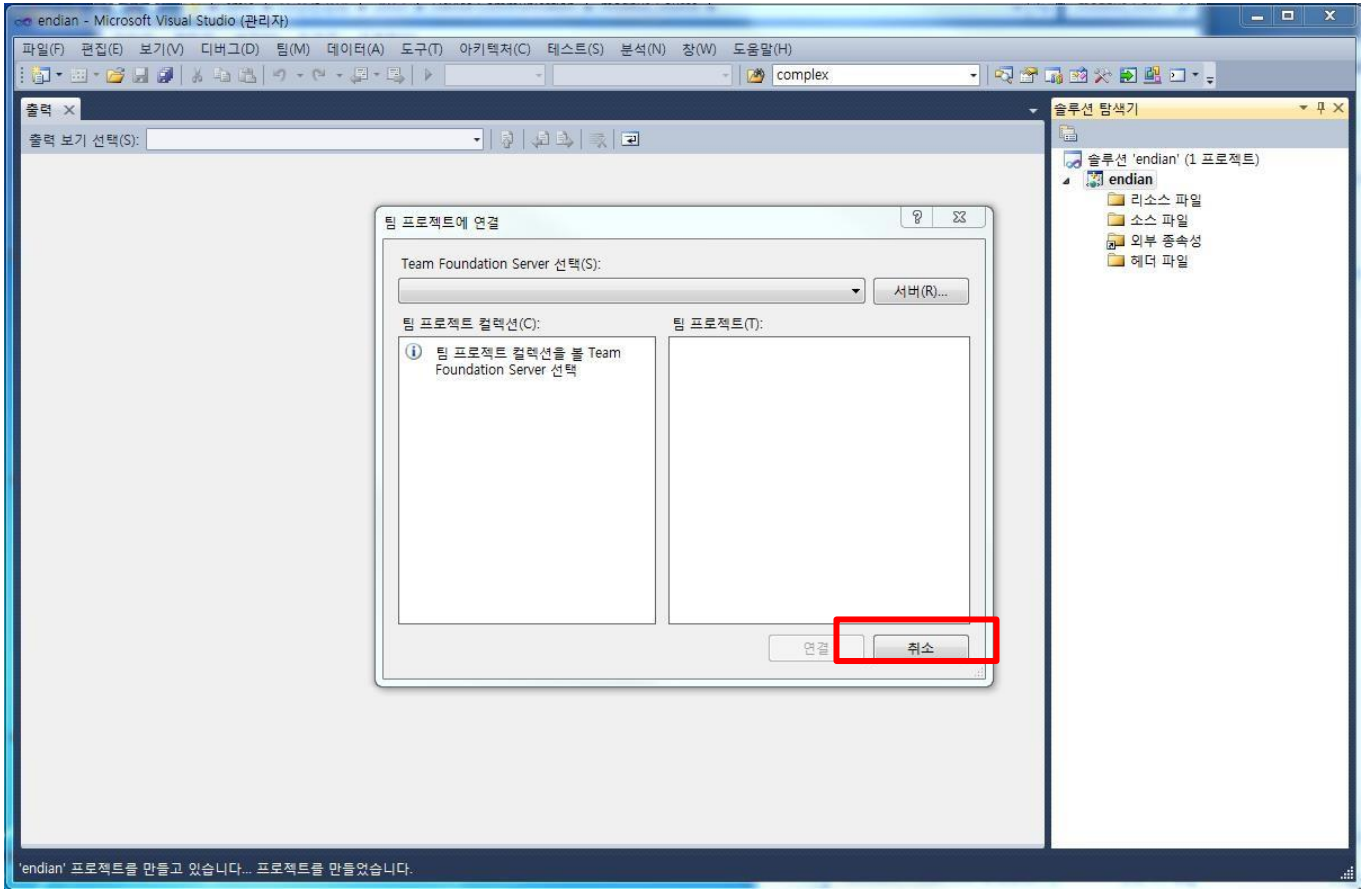
12. Visual Studio 개발환경

12.1. 새 프로젝트 만들기 2



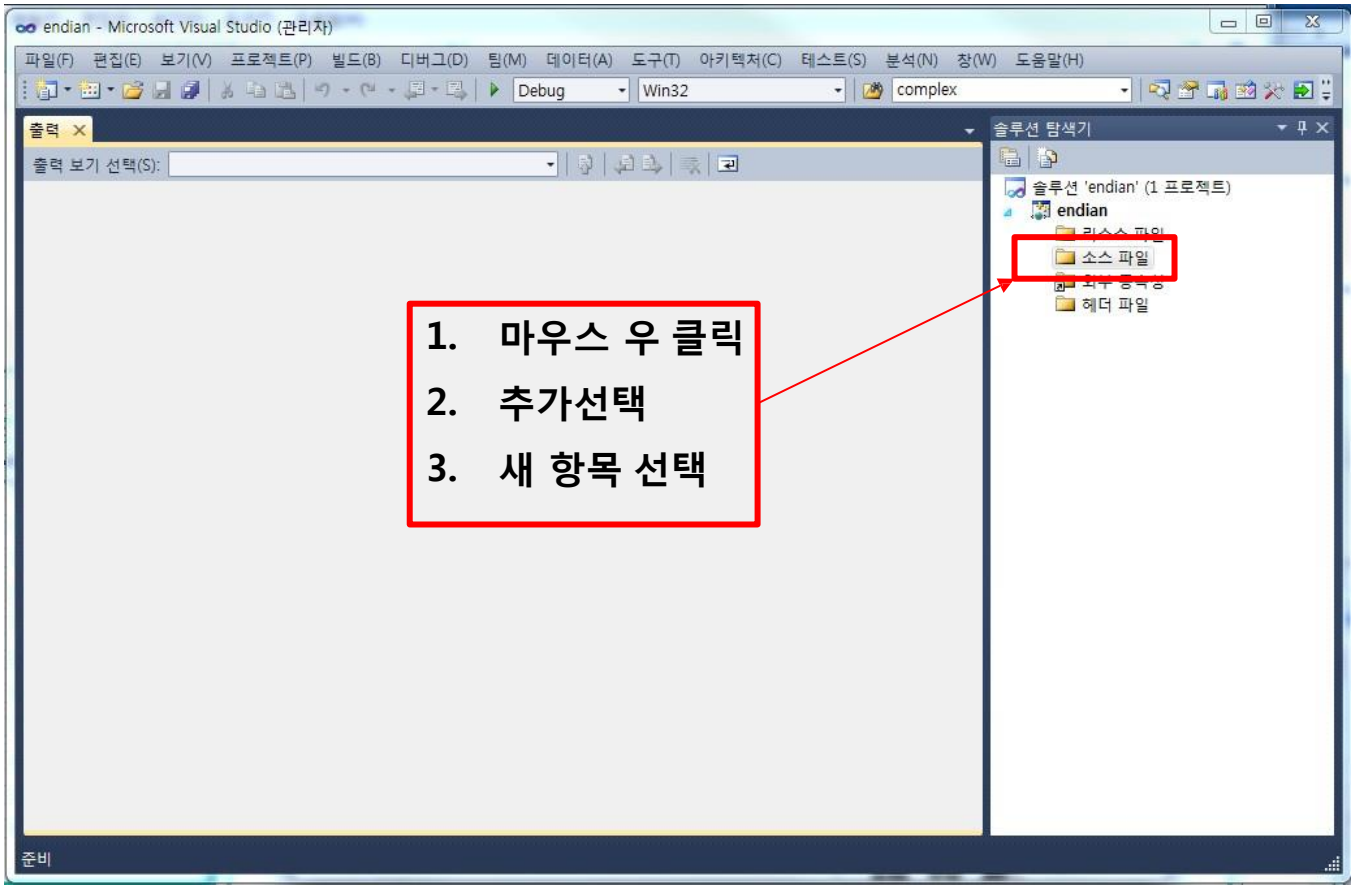
12. Visual Studio 개발환경

12.1. 새 프로젝트 만들기 3



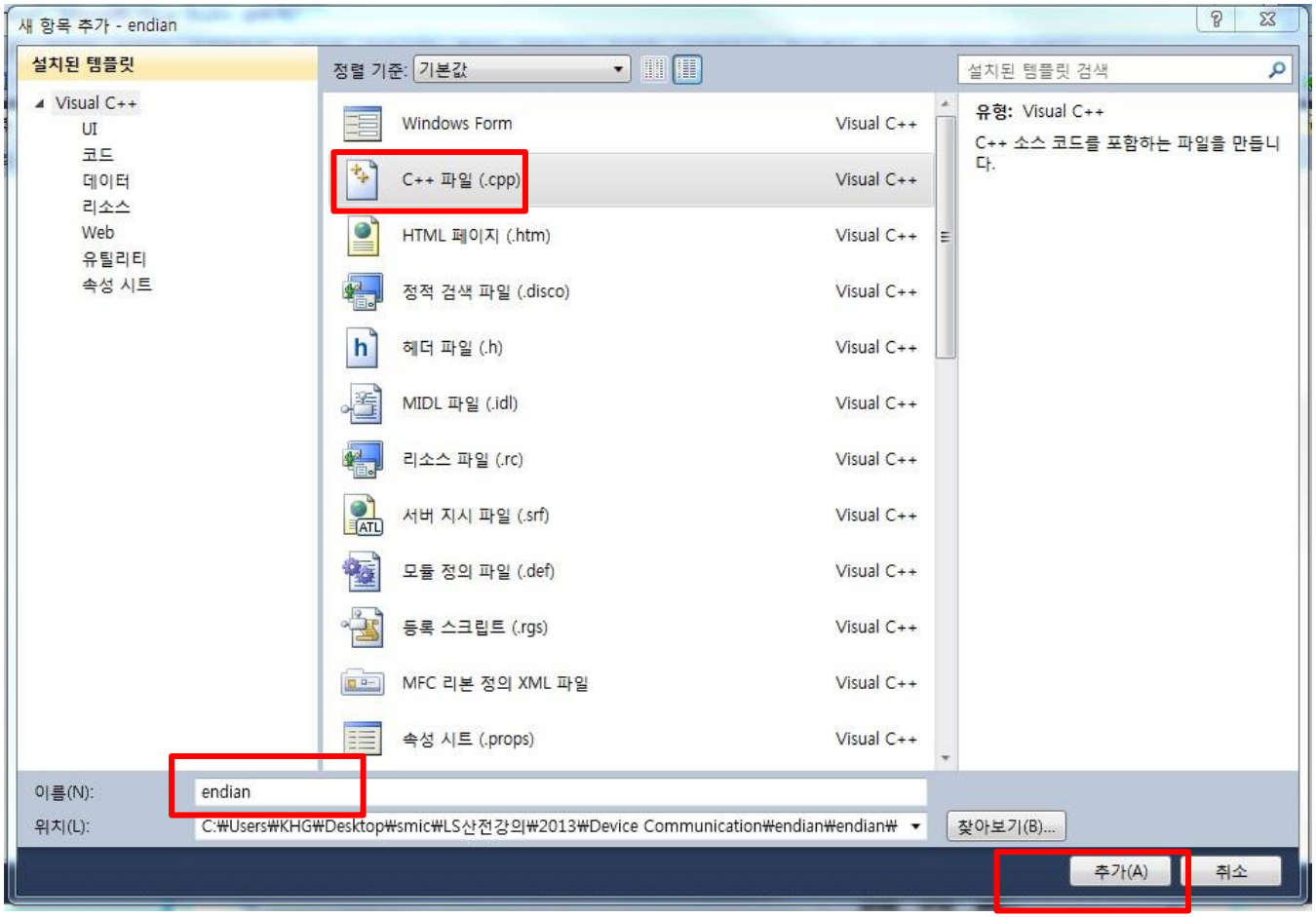
12. Visual Studio 개발환경

12.1. 새 프로젝트 만들기 4



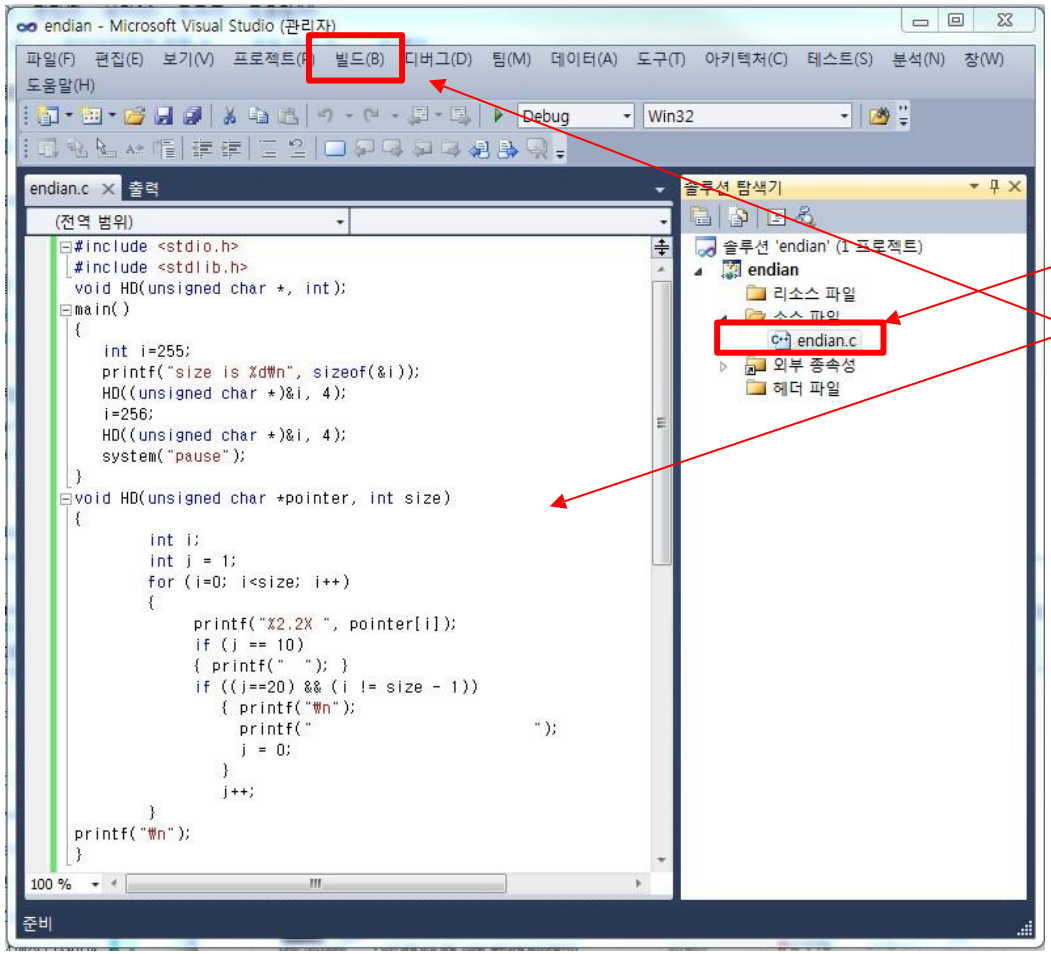
12. Visual Studio 개발환경

12.1. 새 프로젝트 만들기 5



12. Visual Studio 개발환경

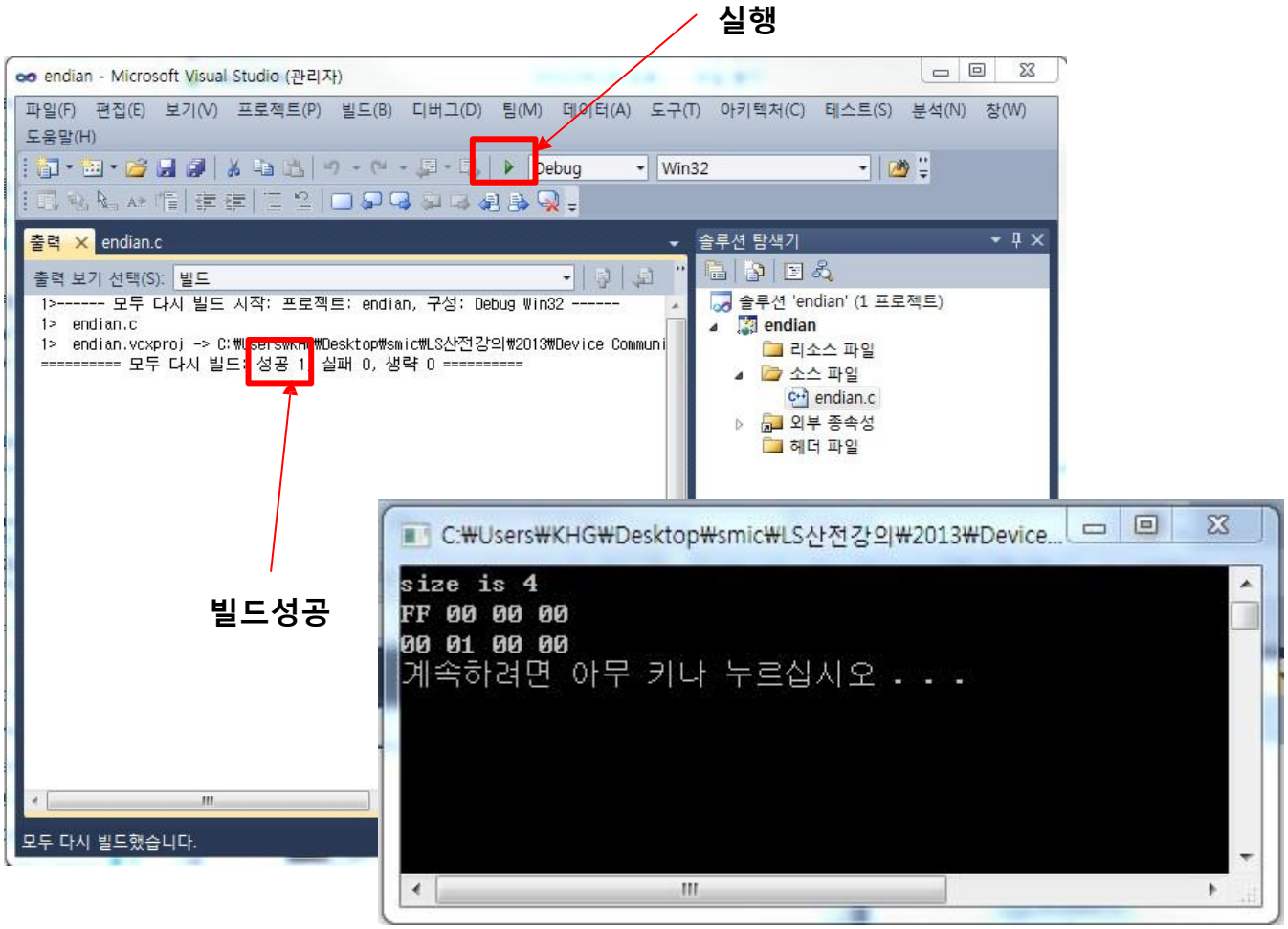
12.2. C-언어 코딩 및 Compile, 실행 1



1. **endian.cpp ->endian.c**
2. **Coding 한다.**
3. 빌드(Compile)한다.

12. Visual Studio 개발환경

12.2. C-언어 코딩 및 Compile, 실행 2

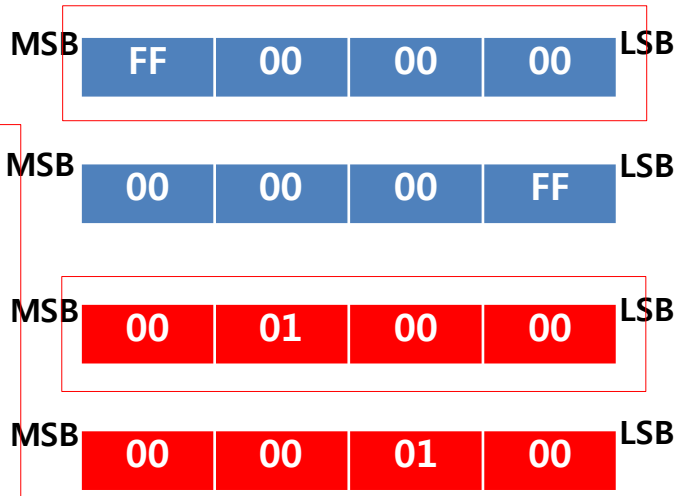


12.3. Endian Coding 실습

```
#include <stdio.h>
#include <stdlib.h>
void HD(unsigned char *, int);
main()
{
    int i=255;
    printf("size is %d₩n", sizeof(&i));
    printf("%d₩n", i);  HD((unsi
    gned char *)&i, 4);  printf("
    %x₩n", i);
    i=256;
    printf("%d₩n", i);
    HD((unsigned char *)&i, 4);
    printf("%d₩n", i);
    printf("%x₩n", i);
    system("pause");
}
```

```
void HD(unsigned char *pointer, int size)
{
        int i;
        int j = 1;
        for (i=0; i<size; i++)
        {
            printf("%2.2X ", pointer[i]);
            if (j == 10)
            { printf("   "); }
            if ((j==20) && (i != size - 1))
              { printf("₩n");
                printf("                    ");
                j = 0;
              } j
              ++;
        }
printf("₩n");
}
```

```
size is 4
255
FF 00 00 00
ff
256
00 01 00 00
256
100
계속하려면 아무 키나 누르십시오 . . .
```



4. CRC Coding 실습 1

❖ **Check field calculation**

The check field allows the receiver to check the validity of the message. The check field value is the Cyclical Redundancy Check (CRC) based **on the polynomial x16+x15+x2+1. CRC** is counted from all message bytes preceding the check field. The algorith m of CRC calculation is introduced below on an example of a **C language function**.

```
unsigned short CreateCRC16(unsigned char *buff, size_t  len)
{
   unsigned short crc16 = 0xFFFF;
   int i,j;
   for(i = 0; i < len; i++)
   {
                    crc16 = crc16 ^ buff[i]; // 모든 비트 반전 : Xor
                    for(j = 0; j < 8; j++)
                    {
                                    if(crc16&1)       //LSB가 1 이면
                                      {
                                        crc16 >>= 1;
                                        crc16 = crc16 ^ 0xA001;  //MSB 에서 1010 000 000  0001.로 Xor  Modbus Polynomial
                                      }
                                    else
                                      {
                                        crc16  >>=  1; //LSB 0 이면
                                      }
                    }
   }
   return crc16;
}
```

12.4. CRC Coding 실습 2

```
71 D5
CRC16 is d571
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <WinSock2.h>

void HD(unsigned char *, int);
unsigned short CreateCRC16(char *buff, size_t  len);
void main()
{
unsigned char SendData[10], *pMsg;
unsigned char Len;
unsigned short crc16=0xffff;
pMsg=&SendData[0];
Len=0x06;

memset(&SendData[0], 0xFF, 1);
memset(&SendData[1], 0x03, 1);
memset(&SendData[2], 0x00, 1);
memset(&SendData[3], 0x06, 1);
memset(&SendData[4], 0x00, 1);
memset(&SendData[5], 0x01, 1);
crc16=CreateCRC16((char *)pMsg, Len);
HD((unsigned char *)&crc16, 2);
printf("CRC16 is %x₩n", crc16);
system("pause");
}
```

```
unsigned short CreateCRC16(unsigned char *buff, size_t  len)
{
   unsigned short crc16 = 0xFFFF;
   int i,j;
   for(i = 0; i < len; i++) {
   crc16 = crc16 ^ buff[i];


     for(j = 0; j < 8; j++) {
     if(crc16&1) {
       crc16 >>= 1;
       crc16 = crc16 ^ 0xA001;
      }
      else {
        crc16 >>= 1;
      }
    }
   }
   return crc16;

}
```

```
void HD(unsigned char *pointer, int size)
{
        int i;
        int j = 1;
        for (i=0; i<size; i++)
        {
             printf("%2.2X ", pointer[i]);
             if (j == 10)
             { printf("  "); }
             if ((j==20) && (i != size - 1))
                { printf("₩n");
                  printf("                    ");
                  j = 0;
                }
                j++;
        }
printf("₩n");
}
```

12.4. CRC Coding 실습 3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

static unsigned char auchCRCHi[ ] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
};
```

12.4. CRC Coding 실습 4

```
/* Table of CRC values for low?order byte */
static unsigned char auchCRCLo[ ] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
};

unsigned short CRC16(unsigned char *, unsigned char);
void HD(unsigned char *, int);
```

12. Visual Studio 개발환경

12.4. CRC Coding 실습 5

```
D5 71
CRC16 is 71d5
계속하려면 아무 키나 누르십시오 . . .
```

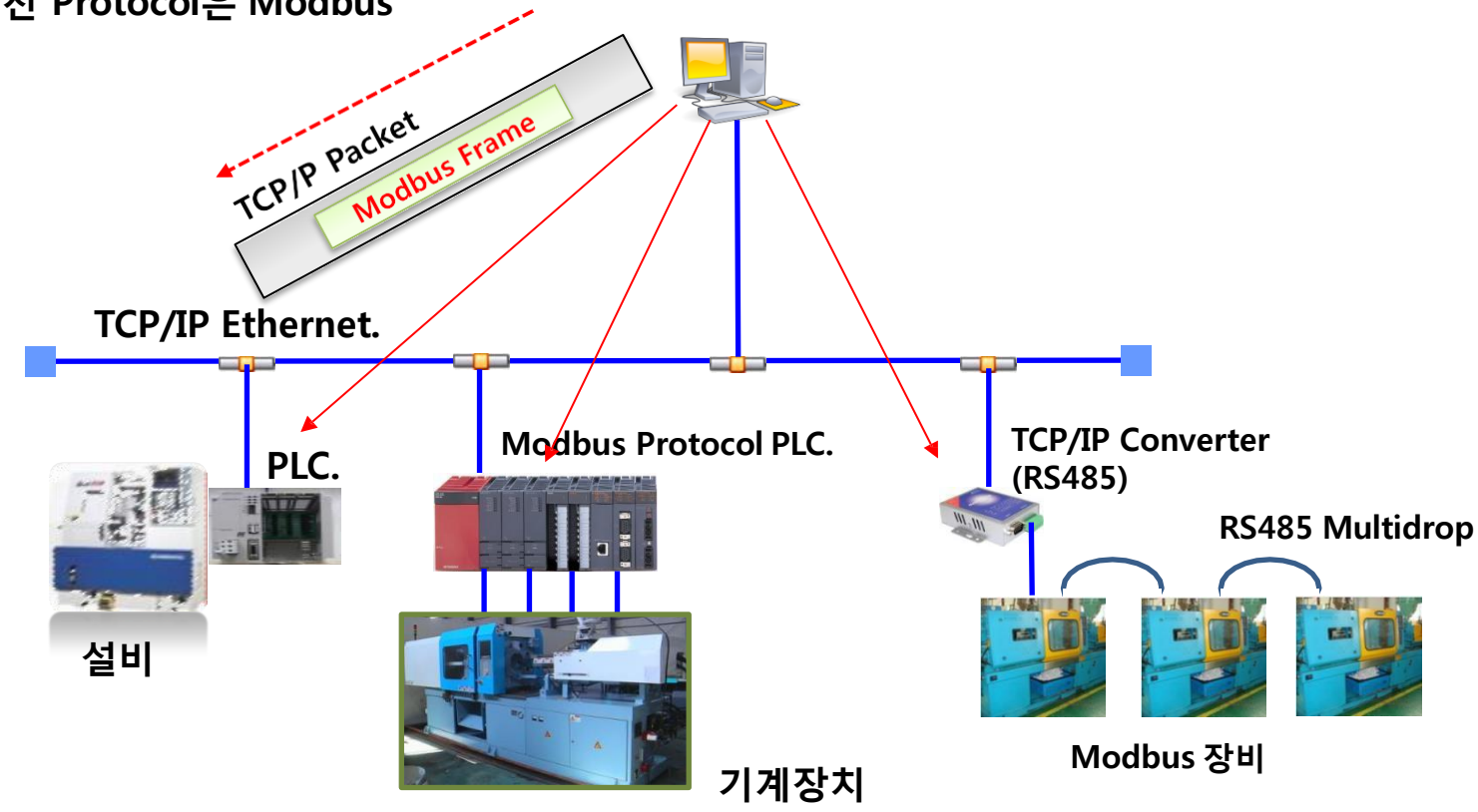
```
void main()
{
unsigned char SendData[10], *pMsg;
unsigned char Len;
unsigned short crc16=0xffff;  p
Msg=&SendData[0];  Len=0x06;

memset(&SendData[0], 0xFF, 1);
memset(&SendData[1], 0x03, 1);
memset(&SendData[2], 0x00, 1);
memset(&SendData[3], 0x06, 1);
memset(&SendData[4], 0x00, 1);
memset(&SendData[5], 0x01, 1);


//memset(&SendData[6], 0xC5, 1);
//memset(&SendData[7], 0xd5, 1);

crc16= CRC16(pMsg, Len);  HD(
(unsigned char *)&crc16, 2);  pri
ntf("CRC16 is %x₩n", crc16);  sy
stem("pause");

}
```

```
unsigned short CRC16(unsigned char *puchMsg, unsigned char  usDataLen)
{
unsigned char uchCRCHi = 0xFF ; /* high byte of CRC initialized */
unsigned char uchCRCLo = 0xFF ; /* low byte of CRC initialized */
unsigned uIndex ; /* will index into CRC lookup table  */
while (usDataLen--) /* pass through message buffer  */
{
uIndex = uchCRCHi ^ *puchMsg++ ; /* calculate the CRC */
uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
uchCRCLo = auchCRCLo[uIndex] ;
}
return (uchCRCHi << 8 | uchCRCLo) ;
}
void HD(unsigned char *pointer, int size)
{       int i;
        int j = 1;
        for (i=0; i<size; i++)
        {
             printf("%2.2X ", pointer[i]);
             if (j == 10)
             { printf("   "); }
             if ((j==20) && (i != size - 1))
                { printf("₩n");
                  printf("                          ");
                  j = 0;
                }
                j++;
        }
printf("₩n");
}
```

13. Modbus TCP Coding 실습

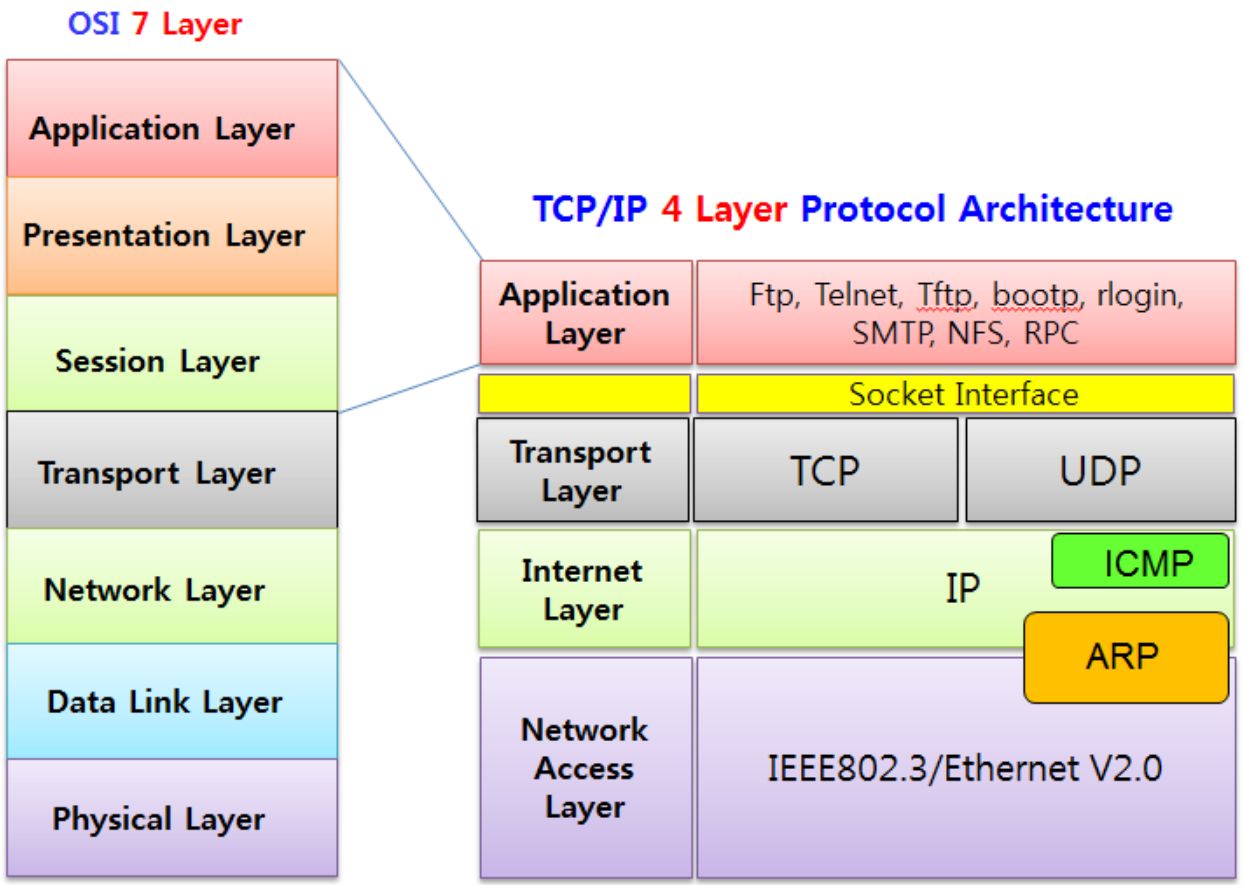
13.1. Modbus TCP Concept

- TCP/IP Socket API로 구현한 Modbus Protocol.
- Physical Layer는 Ethernet (Socket API)
- 통신 Protocol은 Modbus



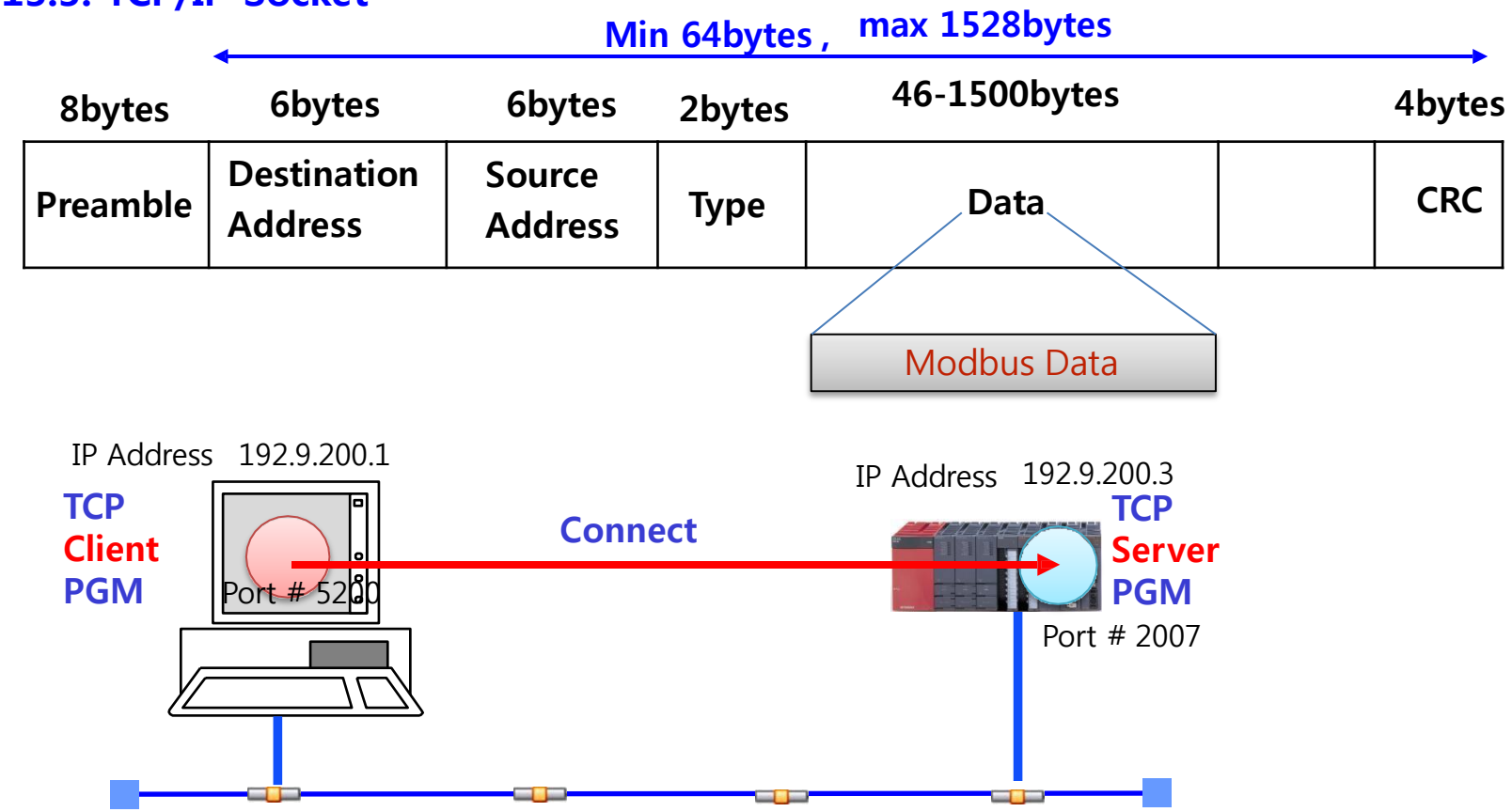
13. Modbus TCP Coding 실습

13.2. TCP/IP Concept



13. Modbus TCP Coding 실습

13.3. TCP/IP Socket

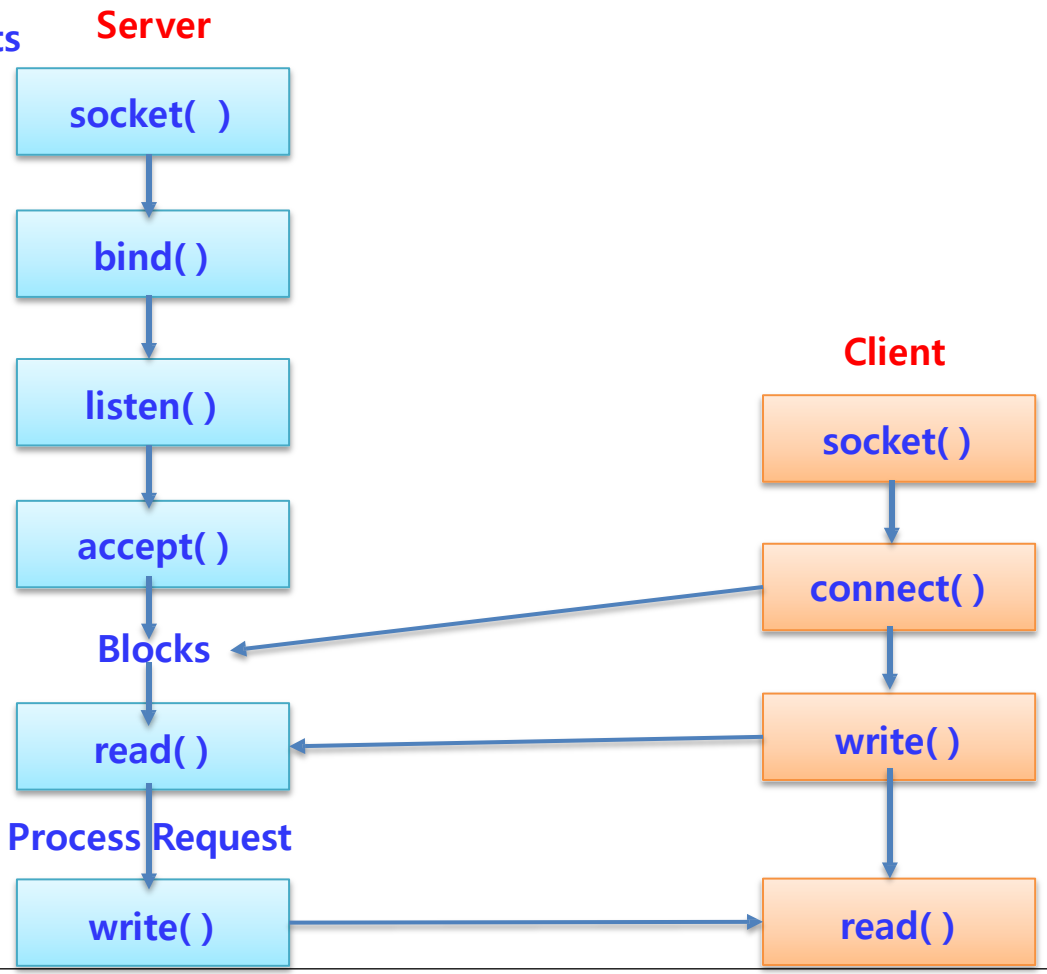


- Server :Connect를 Accept 하는 Program
- Client :Connect를 시도하는 Program

13. Modbus TCP Coding 실습

13.4. TCP/IP Stream Socket Programing 구조

- Stream Sockets



13. Modbus TCP Coding 실습

13.5. Socket Programming Server Coding 1

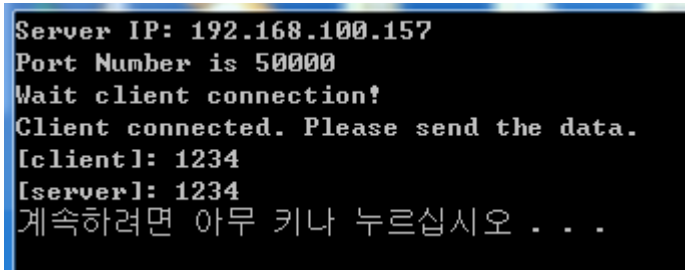
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <WinSock2.h>
```

● Socket Library를 자동으로 설정함.

```
#pragma comment(lib, "WS2_32.LIB")

#define SERVER_PORT 50000

void main()
{
WSADATA wsa;
SOCKET sock, client_sock;
struct sockaddr_in sock_addr, client_addr;
int client_addr_size;

WSAStartup(MAKEWORD(2,2), &wsa);

sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;  sock_addr.sin_ad
dr.s_addr = htonl(INADDR_ANY);
sock_addr.sin_port = htons( SERVER_PORT );

if(bind(sock, (struct sockaddr*)&sock_addr,  sizeof(sock_addr))
== SOCKET_ERROR) {
closesocket(sock);
return;
}
{
   char local_name[1024];
   struct hostent *host_ptr = NULL;
   memset(local_name, 0, 1024);
   gethostname(local_name, 1024);
   host_ptr = gethostbyname(local_name);
   printf("Server IP: %u.%u.%u.%u₩n",
   (unsigned   char)host_ptr->h_addr_list[0][0],
   (unsigned   char)host_ptr->h_addr_list[0][1],
   (unsigned   char)host_ptr->h_addr_list[0][2],
   (unsigned char)host_ptr->h_addr_list[0][3]);
   printf("Port Number is %d₩n", (int)SERVER_PORT);
   printf("Wait client  connection!₩n");
}
if(listen(sock, 1) == SOCKET_ERROR) {
   closesocket(sock);
   return;
}
   client_addr_size = sizeof(client_addr);  me
   mset(&client_addr, 0, client_addr_size);
   client_sock = accept(sock, (struct sockaddr*)&client_addr,  &client_addr_size);
{
   char recv_buff[1024];
   char send_buff[1024];
   printf("Client connected. Please send the data.₩n");
   memset(recv_buff, 0, 1024);
   Recv(client_sock, recv_buff, 1024, 0);
   printf("%s₩n", recv_buff);
```

13.5. Socket Programming Server Coding 2

```
memset(send_buff, 0 , 1024);
sprintf(send_buff, "[server]: %s", &recv_buff[10]);
printf("%s₩n", send_buff);

send(client_sock, send_buff, strlen(send_buff), 0);
}


closesocket(client_sock);
closesocket(sock);

WSACleanup();
system("pause");
}
```



13. Modbus TCP Coding 실습

13.6. Socket Programming Client Coding

```
#include    <stdlib.h>
#include <stdio.h>
#include    <WinSock2.h>
#pragma   comment(lib,   "WS2_32.LIB")
#define   SERVER_IP   "127.0.0.1"
#define   SERVER_PORT   50000
void main()
{
WSADATA    wsa;
SOCKET    sock;
struct   sockaddr_in   sock_addr;

WSAStartup(MAKEWORD(2,2),    &wsa);

sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;
sock_addr.sin_addr.s_addr  = inet_addr( SERVER_IP );
sock_addr.sin_port= htons( SERVER_PORT );
{
char    local_name[1024];
struct hostent *host_ptr = NULL;
memset(local_name, 0, 1024);
gethostname(local_name,    1024);
host_ptr = gethostbyname(local_name);
printf("Client IP: %u.%u.%u.%u₩n",
(unsigned char)host_ptr->h_addr_list[0][0],
(unsigned char)host_ptr->h_addr_list[0][1],
(unsigned char)host_ptr->h_addr_list[0][2],
(unsigned char)host_ptr->h_addr_list[0][3]);
}
connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr));

{
char io_buff[1024];
char comm_buff[1024];
memset(io_buff, 0, 1024);

scanf("%s", io_buff);
memset(comm_buff, 0, 1024);
sprintf(comm_buff, "[client]: %s", io_buff);
printf("%s₩n", comm_buff);
send(sock, comm_buff, strlen(comm_buff), 0);
memset(comm_buff, 0, 1024);

recv(sock, comm_buff, 1024, 0);
printf("%s₩n", comm_buff);

}

closesocket(sock);
WSACleanup();
system("pause");

}
```

```
Client IP: 192.168.100.157
1234
[client]: 1234
[server]: 1234
계속하려면 아무 키나 누르십시오 . . .
```

13. Modbus TCP Coding 실습

13.7. Socket Programming Coding – Modbus Simulator 1

```
#include    <stdlib.h>
#include    <stdio.h>
#include    <string.h>
#include    <WinSock2.h>

#pragma comment(lib, "WS2_32.LIB")

#define   SERVER_PORT   50000
#define   RW_READ   0x03
#define   RW_WRITE   0x06

unsigned  short  ReverseByteOrder(unsigned  short  value);
unsigned  short  CreateCRC16(unsigned  char *buff, size_t  len);

void    main()
{
WSADATA    wsa;
SOCKET  sock,  client_sock;
struct  sockaddr_in  sock_addr,  client_addr;

int client_addr_size;

unsigned  short  reg_map[150];

memset(reg_map,  0x00,  sizeof(short)*150);
reg_map[6]  =  0xFE;

WSAStartup(MAKEWORD(2,2),    &wsa);
```

```
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;
sock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
sock_addr.sin_port = htons( SERVER_PORT );

if(bind(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr))  ==
SOCKET_ERROR) {
closesocket(sock);
return;
}
if(listen(sock, 1) == SOCKET_ERROR) {
closesocket(sock);
return;
}
client_addr_size = sizeof(client_addr);
memset(&client_addr, 0, client_addr_size);
printf("Modbus TCP Simulator for Server is working!₩n");
    printf("Waiting Client Connection!₩n");

client_sock = accept(sock, (struct sockaddr*)&client_addr,  &client_addr_size);
{
unsigned char recv_buff[8];
unsigned char *send_buff;
unsigned short address;
unsigned short value;
unsigned short len;
unsigned short i;
unsigned short crc;
```

13. Modbus TCP Coding 실습

13.7. Socket Programming Coding – Modbus Simulator 2

```
recv(client_sock, (char*)recv_buff, 8, 0);

if( (recv_buff[0] == reg_map[6]) || (recv_buff[0] == 0xFF))
{
if(recv_buff[1] == RW_READ) // 0x03 read function code
{
 memcpy(&address, &recv_buff[2], 2);
 address = ReverseByteOrder(address);
 memcpy(&len, &recv_buff[4], 2);// length
 len = ReverseByteOrder(len);

 send_buff = (unsigned char*)malloc( 3 + len * 2 + 2);
// ID r/w function code, length, register no to read, CRC2bytes
 send_buff[0] = recv_buff[0];
 send_buff[1] = RW_READ;
 send_buff[2] = (unsigned char)len*2;
 for(i = 0; i < len; i++)
  {
   value = reg_map[address + i]; //start address ~ to length
   value = ReverseByteOrder(value);
   memcpy( &send_buff[3 + i*2], &value, 2);// copy to send buffer
  }
crc = CreateCRC16(send_buff, 3+len*2);
memcpy( &send_buff[3+len*2], &crc, 2);
send(client_sock, (char*)send_buff, 3+len*2 + 2, 0); //response sending

free(send_buff);

send_buff = NULL;
}
}
}
```

```
closesocket(client_sock);

closesocket(sock);
WSACleanup();
}
unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;
((char*)&ret)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}
unsigned short CreateCRC16(unsigned char *buff, size_t len)
{
   unsigned short crc16 = 0xFFFF;
   int i = 0;
   while(len--) {
      crc16 ^= *buff++;
      for(i=0; i<8; i++) {
         if(crc16&1) {
            crc16 >>= 1;
            crc16 ^= 0xA001;
         }
         else {
            crc16 >>= 1;
         }
      }
   }
   return crc16;
}
```

```
Modbus TCP Simulator for Server is working!
Waiting Client Connection!
```

13. Modbus TCP Coding 실습

13.8. Modbus DeviceID 읽기 Coding 실습 1

```
#include    <stdlib.h>
#include    <stdio.h>
#include    <WinSock2.h>
#pragma  comment(lib,  "WS2_32.LIB")
#define  NE_IP  "192.168.100.158"
#define  NE_PORT    50000
unsigned  short  CreateCRC16(char  *buff, size_t  len);
unsigned  short  ReverseByteOrder(unsigned  short  value);
void  PrintHexa(char*  buff,  size_t  len);
void   main()
{
SOCKET   sock;
WSADATA   wsa;
char   s_buff[8];
char   r_buff[7];
struct  sockaddr_in  sock_addr;
int   ret;
WSAStartup(MAKEWORD(2,2),   &wsa);
sock  =  socket(PF_INET, SOCK_STREAM,  IPPROTO_TCP);
memset(&sock_addr,  0,  sizeof(sock_addr));
sock_addr.sin_family  =  AF_INET;
sock_addr.sin_addr.s_addr  =  inet_addr( NE_IP );

sock_addr.sin_port= htons( NE_PORT );
if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr))  ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();
return;
}
memset(s_buff, 0, 8);
memset(r_buff, 0, 7);
{
  unsigned char d_id = 0xFF;
  unsigned char d_rw = 0x03;
  unsigned short d_addr = 6;
  unsigned short d_len = 1;
  unsigned short crc16;
  s_buff[0] = d_id;
  s_buff[1] = d_rw;
  d_addr = ReverseByteOrder( d_addr );
  memcpy(&s_buff[2], &d_addr, 2);
  d_len = ReverseByteOrder( d_len );
  memcpy(&s_buff[4], &d_len, 2);
  crc16 = CreateCRC16(s_buff, 6);
  PrintHexa(&crc16, 2);
  memcpy(&s_buff[6], &crc16, 2);
}
 PrintHexa(s_buff, 8);
 if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
 perror("send()");
 closesocket(sock);

 WSACleanup();
 return;
}
```

13. Modbus TCP Coding 실습

13.8. Modbus Device ID 읽기 Coding 실습 2

```
ret = recv(sock, r_buff, 7, 0);
if(ret == SOCKET_ERROR) {
perror("recv()");  c
losesocket(sock);
WSACleanup();  r
eturn;
}
PrintHexa(r_buff, ret);
printf("ID : %02X₩n", (unsigned char)r_buff[4]);
closesocket(sock);
WSACleanup();
system("pause");
}

unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;
((char*)&ret)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}

unsigned short CreateCRC16(char *buff, size_t len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned char*)buff;
while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}

void PrintHexa(char* buff, size_t len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned char)buff[i]);
}
printf("₩n");
}
```

```
71 D5
FF 03 00 06 00 01 71 D5
FF 03 02 00 02 10 51
ID : 02
계속하려면 아무 키나 누르십시오 . . .
```

13. Modbus TCP Coding 실습

13.9. Modbus Device ID 변경(쓰기) Coding 실습 1

```
// ID 변경
#include <stdlib.h>
#include <stdio.h>
#include <WinSock2.h>
#pragma comment(lib, "WS2_32.LIB")
//IP
#define NE_IP "192.168.100.158"
//Port
#define NE_PORT 50000

unsigned short CreateCRC16(char *buff, size_t len); unsi
gned short ReverseByteOrder(unsigned short value); voi
d PrintHexa(char* buff, size_t len);

void main()
{
SOCKET sock;
WSADATA wsa;
char s_buff[8];
char r_buff[8];
struct sockaddr_in sock_addr;

WSAStartup(MAKEWORD(2,2), &wsa);
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr)); soc
k_addr.sin_family = AF_INET; sock_addr.sin_ad
dr.s_addr = inet_addr( NE_IP ); sock_addr.sin_p
ort= htons( NE_PORT );
if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr)) ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup(); ret
urn;
}

memset(s_buff, 0, 8);
memset(r_buff, 0, 8);

{
unsigned char d_id = 0x02; unsig
ned char d_rw = 0x06; unsigned
short d_addr = 6; unsigned short
d_value = 0x0001; unsigned shor
t crc16;

s_buff[0] = d_id;
s_buff[1] = d_rw;
d_addr = ReverseByteOrder( d_addr );
memcpy(&s_buff[2], &d_addr, 2); d_val
ue = ReverseByteOrder( d_value ); me
mcpy(&s_buff[4], &d_value, 2); crc16 =
CreateCRC16(s_buff, 6); memcpy(&s_b
uff[6], &crc16, 2);
}
```

13. Modbus TCP Coding 실습

13.9. Modbus DeviceID 변경(쓰기) Coding 실습 2

```
PrintHexa(s_buff, 8);
if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
perror("send()");
closesocket(sock);
WSACleanup();  r
eturn;
}

if(recv(sock, r_buff, 8, 0) == SOCKET_ERROR) {
perror("recv()");
closesocket(sock);
WSACleanup();  r
eturn;
}

PrintHexa(r_buff, 8);
if(memcmp(s_buff, r_buff, 8) == 0)
printf("done₩n");
else  printf("fai
l₩n");

closesocket(sock);
WSACleanup();

system("pause");
}
```

```
unsigned short ReverseByteOrder(unsigned short  value)
{
unsigned short ret = 0;  ((char*)&re
t)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}

unsigned short CreateCRC16(char *buff, size_t  len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned  char*)buff;

while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}
```

13.9. Modbus DeviceID 변경(쓰기) Coding 실습 3

```
void PrintHexa(char* buff, size_t len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned char)buff[i]);
}
printf("₩n");
}
```

13. Modbus TCP Coding 실습

13.10. Modbus Baudrates 읽기 Coding 실습 1

```
// Baud rate 읽기

#include <stdlib.h>
#include <stdio.h>
#include <WinSock2.h>

#pragma comment(lib, "WS2_32.LIB")

//IP
#define NE_IP"192.168.100.158"
//Port
#define NE_PORT   50000


unsigned short CreateCRC16(char *buff, size_t len);
unsigned short ReverseByteOrder(unsigned short value);
void  PrintHexa(char*  buff, size_t  len);

void main()
{
SOCKET sock;
WSADATA wsa;
char s_buff[8];
char r_buff[7];
struct sockaddr_in sock_addr;
int ret;
WSAStartup(MAKEWORD(2,2), &wsa);
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;  sock_addr.sin_
addr.s_addr = inet_addr( NE_IP );  sock_addr.sin
_port= htons( NE_PORT );
if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr))  ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();  ret
urn;
}
memset(s_buff, 0, 8);
memset(r_buff, 0, 7);{
unsigned char d_id = 0xFF; //
unsigned char d_rw = 0x03;
unsigned short d_addr = 9;
unsigned short d_len = 1;
unsigned short crc16;
s_buff[0] = d_id;
s_buff[1] = d_rw;
d_addr = ReverseByteOrder( d_addr );
memcpy(&s_buff[2], &d_addr, 2);  d_le
n = ReverseByteOrder( d_len );  memc
py(&s_buff[4], &d_len, 2);
crc16 = CreateCRC16(s_buff, 6);
memcpy(&s_buff[6], &crc16, 2);
}
```

13. Modbus TCP Coding 실습

13.10. Modbus Baudrates 읽기 Coding 실습 2

```
PrintHexa(s_buff, 8);
if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
perror("send()");
closesocket(sock);
WSACleanup();  r
eturn;
}

ret = recv(sock, r_buff, 7, 0);
if(ret == SOCKET_ERROR) {
perror("recv()");  c
losesocket(sock);
WSACleanup();  r
eturn;
}

PrintHexa(r_buff, ret);

{
unsigned short baud_rate;  memcpy(&bau
d_rate, &r_buff[3], 2);  baud_rate = Revers
eByteOrder(baud_rate);  printf("baud rate:
%d₩n", baud_rate*100);
}
closesocket(sock);
WSACleanup();

system("pause");
}
```

```
unsigned short ReverseByteOrder(unsigned short  value)
{
unsigned short ret = 0;  ((char*)&re
t)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}

unsigned short CreateCRC16(char *buff, size_t  len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned  char*)buff;

while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}
```

13. Modbus TCP Coding 실습

13.10. Modbus Baudrates 읽기 Coding 실습 3

```
void PrintHexa(char* buff, size_t  len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned  char)buff[i]);
}
printf("₩n");
}
```

```
FF 03 00 09 00 01 41 D6
FF 03 02 00 C0 91 C0
baud rate: 19200
계속하려면 아무 키나 누르십시오 . . .
```

13. Modbus TCP Coding 실습

13.11. Modbus Digital Input 8 channel 읽기 Coding 실습 1

```
// 접점 상태 읽기

#include <stdlib.h>

#include     <stdio.h>
#include     <WinSock2.h>

#pragma   comment(lib,   "WS2_32.LIB")

//IP
#define NE_IP"192.168.100.158"
//Port
#define NE_PORT   50000

unsigned  short  CreateCRC16(char  *buff, size_t  len);
unsigned  short  ReverseByteOrder(unsigned  short  value);
void  PrintHexa(char*  buff,  size_t  len);

void     main()
{
SOCKET     sock;
WSADATA     wsa;
char     s_buff[8];
char     r_buff[7];
struct   sockaddr_in   sock_addr;
int     ret;

WSAStartup(MAKEWORD(2,2), &wsa);

sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;
sock_addr.sin_addr.s_addr = inet_addr( NE_IP );
sock_addr.sin_port= htons( NE_PORT );

if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr)) ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();
return;
}

memset(s_buff, 0, 8);
memset(r_buff, 0, 7);

{
unsigned char d_id = 0xFF; //
unsigned char d_rw = 0x03;
unsigned short d_addr = 100;
unsigned short d_len = 1;
unsigned short crc16;
```

13. Modbus TCP Coding 실습

13.11. Modbus Digital Input 8 channel 읽기 Coding 실습 2

```
s_buff[0] = d_id;
s_buff[1] = d_rw;
d_addr = ReverseByteOrder( d_addr );
memcpy(&s_buff[2], &d_addr, 2);
d_len = ReverseByteOrder( d_len );
memcpy(&s_buff[4], &d_len, 2);
crc16 = CreateCRC16(s_buff, 6);
memcpy(&s_buff[6], &crc16, 2);
}

PrintHexa(s_buff, 8);

if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
perror("send()");
closesocket(sock);
WSACleanup();
return;
}

ret = recv(sock, r_buff, 7, 0);
if(ret == SOCKET_ERROR) {
perror("recv()");
closesocket(sock);
WSACleanup();
return;
}

PrintHexa(r_buff, ret);
{
  unsigned char di;
  int i;
  di = r_buff[4];
  for(i = 0; i < 8; i++)
  {
    printf("CH%d: ", i+1);
    if(di & 0x01)
    printf("OFF ");
    else

    printf("ON ");
    di>>=1;
  }
printf("₩n");
}
closesocket(sock);
WSACleanup();

system("pause");
}

unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;
((char*)&ret)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}
```

13. Modbus TCP Coding 실습

13.11. Modbus Digital Input 8 channel 읽기 Coding 실습 3

```
unsigned short CreateCRC16(char *buff, size_t len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned char*)buff;

while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}

void PrintHexa(char* buff, size_t len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned char)buff[i]);
}
printf("₩n");
}
```

```
FF 03 00 64 00 01 D0 0B
FF 03 02 00 FF D1 D0
CH1: OFF CH2: OFF CH3: OFF CH4: OFF CH5: OFF CH6: OFF CH7: OFF CH8: OFF
계속하려면 아무 키나 누르십시오 . . .
```

13. Modbus TCP Coding 실습

13.12. Modbus Digital Output 6 channel 상태읽기 Coding 실습 1

```
// DO 상태 읽기

#include <stdlib.h>
#include <stdio.h>
#include <WinSock2.h>

#pragma comment(lib, "WS2_32.LIB")

//IP
#define NE_I P"192.168.100.158"
//Port
#define NE_PORT  50000


unsigned short CreateCRC16(char *buff, size_t len);  unsi
gned short ReverseByteOrder(unsigned short value);  voi
d PrintHexa(char* buff, size_t  len);

void main()
{
SOCKET sock;
WSADATA wsa;
char s_buff[8];
char r_buff[7];
struct sockaddr_in sock_addr;
int ret;
WSAStartup(MAKEWORD(2,2), &wsa);
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));  soc
k_addr.sin_family = AF_INET;  sock_addr.sin_ad
dr.s_addr = inet_addr( NE_IP );  sock_addr.sin_p
ort= htons( NE_PORT );
if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr))  ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();  ret
urn;
}
memset(s_buff, 0, 8);
memset(r_buff, 0, 7);
{
unsigned char d_id = 0xFF; //
unsigned char d_rw = 0x03;
unsigned short d_addr = 101;
unsigned short d_len = 1;  un
signed short crc16;  s_buff[0]
= d_id;
s_buff[1] = d_rw;
d_addr = ReverseByteOrder( d_addr );
memcpy(&s_buff[2], &d_addr, 2);  d_le
n = ReverseByteOrder( d_len );  memc
py(&s_buff[4], &d_len, 2);
crc16 = CreateCRC16(s_buff, 6);
memcpy(&s_buff[6], &crc16, 2);
}
```

13. Modbus TCP Coding 실습

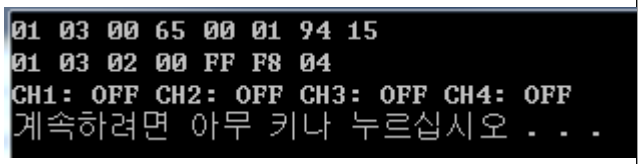
13.12. Modbus Digital Output 6 channel 상태읽기 Coding 실습 2

```
PrintHexa(s_buff, 8);
if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
perror("send()");
closesocket(sock);
WSACleanup();  r
eturn;
}

ret = recv(sock, r_buff, 7, 0);
if(ret == SOCKET_ERROR) {
perror("recv()");  c
losesocket(sock);
WSACleanup();  r
eturn;
}

PrintHexa(r_buff, ret);

{
unsigned char output;
int i;
output = r_buff[4];  f
or(i = 0; i < 4; i++) {
printf("CH%d: ", i+1);
if(output & 0x01)  pri
ntf("OFF ");
else  printf("
ON ");  outp
ut>>=1;
}
printf("₩n");
}
closesocket(sock);
WSACleanup();

system("pause");
}

unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;  ((char*)&re
t)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}

unsigned short CreateCRC16(char *buff, size_t  len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned char*)buff;

while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
```

13. Modbus TCP Coding 실습

13.12. Modbus Digital Output 6 channel 상태읽기 Coding 실습 3

```
else {
crc16 >>= 1;
}
}
}
return crc16;
}

void PrintHexa(char* buff, size_t  len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned char)buff[i]);
}
printf("₩n");
}
```



13. Modbus TCP Coding 실습

13.13. Modbus Digital Output 6 channel 쓰기 Coding 실습 1

```
// DO 변경

#include <stdlib.h>
#include <stdio.h>
#include <WinSock2.h>
#pragma comment(lib, "WS2_32.LIB")
//IP
#define NE_IP"192.168.100.158"
//Port
#define NE_PORT   50000
unsigned short CreateCRC16(char *buff, size_t len);  unsi
gned short ReverseByteOrder(unsigned short value);  voi
d PrintHexa(char* buff, size_t  len);

void main()
{
SOCKET sock;
WSADATA wsa;
char s_buff[8];
char r_buff[8];
struct sockaddr_in sock_addr;

WSAStartup(MAKEWORD(2,2), &wsa);
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));  soc
k_addr.sin_family = AF_INET;  sock_addr.sin_ad
dr.s_addr = inet_addr( NE_IP );  sock_addr.sin_p
ort= htons( NE_PORT );

if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr)) ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();  ret
urn;
}
memset(s_buff, 0, 8);
memset(r_buff, 0, 8);
{
unsigned char d_id = 0x01; //  un
signed char d_rw = 0x06;  unsign
ed short d_addr = 101;  unsigned
short d_value = 0x0000;  unsigne
d short crc16;
/*
값: X X X X 8 4 2 1
채널:       4 3 2 1
*/
// 2번 3번 채널에 출력 ON한다.
//d_value &= (~6);
s_buff[0] = d_id;  s
_buff[1] = d_rw;
d_addr = ReverseByteOrder( d_addr );
memcpy(&s_buff[2], &d_addr, 2);  d_val
ue = ReverseByteOrder( d_value );  me
mcpy(&s_buff[4], &d_value, 2);  crc16 =
CreateCRC16(s_buff, 6);  memcpy(&s_b
uff[6], &crc16, 2);
}
```

13. Modbus TCP Coding 실습

13.13. Modbus Digital Output 6 channel 쓰기 Coding 실습 2

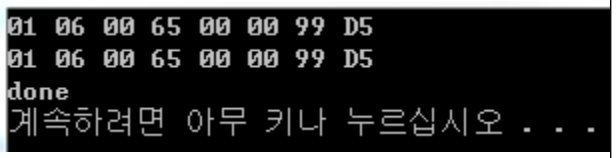
```
PrintHexa(s_buff, 8);
if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
perror("send()");
closesocket(sock);
WSACleanup();  r
eturn;
}

if(recv(sock, r_buff, 8, 0) == SOCKET_ERROR) {
perror("recv()");
closesocket(sock);
WSACleanup();  r
eturn;
}

PrintHexa(r_buff, 8);
if(memcmp(s_buff, r_buff, 8) == 0)
printf("done₩n");
else  printf("fai
l₩n");

closesocket(sock);
WSACleanup();

system("pause");
}
```

```
unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;  ((char*)&re
t)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}

unsigned short CreateCRC16(char *buff, size_t  len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned char*)buff;

while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}
```

13. Modbus TCP Coding 실습

13.13. Modbus Digital Output 6 channel 쓰기 Coding 실습 3

```
void PrintHexa(char* buff, size_t  len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned  char)buff[i]);
}
printf("₩n");
}
```



13. Modbus TCP Coding 실습

13.14. Modbus Counter 읽기 1

```
// 카운터 읽기
#include <stdlib.h>
#include <stdio.h>
#include <WinSock2.h>

#pragma comment(lib, "WS2_32.LIB")

//IP
#define NE_IP"192.168.100.158"
//Port
#define NE_PORT   50000


unsigned short CreateCRC16(char *buff, size_t len);  unsi
gned short ReverseByteOrder(unsigned short value);  un
signed int ReverseByteOrderInt(unsigned int value);  voi
d PrintHexa(char* buff, size_t  len);

void main()
{
SOCKET sock;
WSADATA wsa;
char s_buff[8];
char r_buff[9];
struct sockaddr_in sock_addr;
int ret;
WSAStartup(MAKEWORD(2,2), &wsa);
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));  soc
k_addr.sin_family = AF_INET;  sock_addr.sin_ad
dr.s_addr = inet_addr( NE_IP );  sock_addr.sin_p
ort= htons( NE_PORT );
if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr)) ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();  ret
urn;
}
memset(s_buff, 0, 8);
memset(r_buff, 0, 9);
{
unsigned char d_id = 0x01; //
unsigned char d_rw = 0x03;
unsigned short d_addr = 102; // 1번 채널 102, 2번 채널 104, 106 ... 116
unsigned short d_len = 2;
unsigned short crc16;  s_b
uff[0] = d_id;
s_buff[1] = d_rw;
d_addr = ReverseByteOrder( d_addr );
memcpy(&s_buff[2], &d_addr, 2);  d_le
n = ReverseByteOrder( d_len );  memc
py(&s_buff[4], &d_len, 2);
crc16 = CreateCRC16(s_buff, 6);
memcpy(&s_buff[6], &crc16, 2);
}
```

13. Modbus TCP Coding 실습

13.14. Modbus Counter 읽기 2

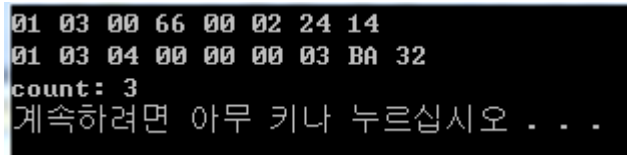
```
PrintHexa(s_buff, 8);
if(send(sock, s_buff, 8, 0) == SOCKET_ERROR) {
perror("send()");
closesocket(sock);
WSACleanup();  r
eturn;
}

ret = recv(sock, r_buff, 9, 0);
if(ret == SOCKET_ERROR) {
perror("recv()");  c
losesocket(sock);
WSACleanup();  r
eturn;
}

PrintHexa(r_buff, ret);

{
unsigned int count;  memcpy(&coun
t, &r_buff[3], 4);  count = ReverseByt
eOrderInt(count);  printf("count: %d
₩n", count);
}
closesocket(sock);
WSACleanup();

system("pause");
}
```

```
unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;  ((char*)&re
t)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}

unsigned int ReverseByteOrderInt(unsigned int value)
{
unsigned int ret = 0;  ((char*)&ret)[
0] = ((char*)&value)[3];
((char*)&ret)[1] = ((char*)&value)[2];
((char*)&ret)[2] = ((char*)&value)[1];
((char*)&ret)[3] = ((char*)&value)[0];
return ret;
}

unsigned short CreateCRC16(char *buff, size_t  len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned char*)buff;

while(len--) {
crc16 ^= *c_buff++;
```

13. Modbus TCP Coding 실습

13.14. Modbus Counter 읽기 3

```
for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;  crc
16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}

void PrintHexa(char* buff, size_t  len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned  char)buff[i]);
}
printf("₩n");
}
```



14. Modbus TCP Application Coding 실습

Digital Input 1,2,3,4,5,6 ch의 접점상태에 따라 Digital Output 1,2,3, 4,5,6 ch의 LED가 켜지고 꺼지는 Application Program을 작성함.

❖ Modbus Protocol Program 실습

● Application : Answer 1

```
#include <stdlib.h>
#include <stdio.h>
#include <WinSock2.h>

#pragma comment(lib, "WS2_32.LIB")

//IP
#define NE_IP "192.168.100.158"
//Port
#define NE_PORT  50000

unsigned short CreateCRC16(char *buff, size_t len);
unsigned short ReverseByteOrder(unsigned short value);
void PrintHexa(char* buff, size_t  len);

void main()
{
SOCKET sock;
WSADATA wsa;
char s_buff[8];
char r_buff[8];
struct sockaddr_in sock_addr;

WSAStartup(MAKEWORD(2,2), &wsa);
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
memset(&sock_addr, 0, sizeof(sock_addr));
sock_addr.sin_family = AF_INET;
sock_addr.sin_addr.s_addr = inet_addr( NE_IP );
sock_addr.sin_port= htons( NE_PORT );
if(connect(sock, (struct sockaddr*)&sock_addr, sizeof(sock_addr)) ==
SOCKET_ERROR) {
perror("connect()");
closesocket(sock);
WSACleanup();  ret
urn;
}
{
unsigned char d_id = 0x01; //
unsigned short di_addr = 100;
unsigned short do_addr = 101;
unsigned short d_len = 1;  unsi
gned short crc16;  unsigned sh
ort d_value;
di_addr = ReverseByteOrder( di_addr );
do_addr = ReverseByteOrder( do_addr );
d_len = ReverseByteOrder( d_len  );

while(1) {  s_buff
[0] = d_id;  s_bu
ff[1] = 0x03;
memcpy(&s_buff[2], &di_addr, 2);
memcpy(&s_buff[4],  &d_len,  2);
crc16  =  CreateCRC16(s_buff,  6);
memcpy(&s_buff[6], &crc16, 2);
PrintHexa(s_buff, 8);
send(sock, s_buff, 8, 0);
recv(sock, r_buff, 7, 0);
PrintHexa(r_buff, 7);
d_value  =  0xFFFF;
d_value &= (unsigned short)r_buff[4];
d_value  =  ReverseByteOrder(d_value);
```

❖ Modbus Protocol Program 실습

● Application : Answer 2

```
s_buff[1] = 0x06;
memcpy(&s_buff[2], &do_addr, 2);
memcpy(&s_buff[4], &d_value, 2);
crc16 = CreateCRC16(s_buff, 6);
memcpy(&s_buff[6], &crc16, 2);
PrintHexa(s_buff, 8);
send(sock, s_buff, 8, 0);
recv(sock, r_buff, 8, 0);
PrintHexa(r_buff, 8);

Sleep(100);
}
}
closesocket(sock);
WSACleanup();
}

unsigned short ReverseByteOrder(unsigned short value)
{
unsigned short ret = 0;
((char*)&ret)[0] = ((char*)&value)[1];
((char*)&ret)[1] = ((char*)&value)[0];
return ret;
}
```

```
unsigned short CreateCRC16(char *buff, size_t  len)
{
unsigned short crc16 = 0xFFFF;
int i = 0;
unsigned char* c_buff = (unsigned char*)buff;

while(len--) {
crc16 ^= *c_buff++;

for(i=0; i<8; i++) {
if(crc16&1) {
crc16 >>= 1;
crc16 ^= 0xA001;
}
else {
crc16 >>= 1;
}
}
}
return crc16;
}

void PrintHexa(char* buff, size_t  len)
{
size_t i;
for(i = 0; i < len; i++) {
printf("%02X ", (unsigned char)buff[i]);
}
printf("₩n");
}
```

❖ 약어

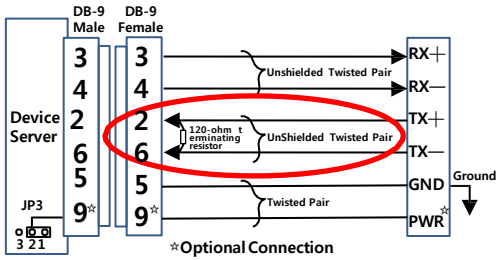
▪ Modem	(Modulator Demodulator)	2. Analog Data and Digital Data
▪ DSU	(Digital Service Unit)	2. Analog Data and Digital Data
▪ UTP	(Unshielded Twist Cable)	3. Transmission Media
▪ STP	(Shield Twist Cable)	3. Transmission Media
▪ Cat	(Category)	3. Transmission Media
▪ UTP	(Unshielded Twisted Pair)	3. Transmission Media
▪ STP	(Shield Twisted Pair)	3. Transmission Media
▪ FTP	(Foil Screened Twisted Pair)	3. Transmission Media
▪ GPIB	General Purpose Interface Board	4. Device Communication
▪ SECS	SEMI Equipment Communication Standard	4. Device Communication
▪ DCS	Distributed Control System	4. Device Communication
▪ MAP	Manufacturing Automation Protocol	4. Device Communication
▪ HMI	Human Man Interface	4. Device Communication
▪ DSU	Digital Service Unit	6. Serial Communication
▪ EIA	Electronic Industries Alliance	6. Serial Communication

주식회사) 신명정보통신
김 한 규

❖ 부록

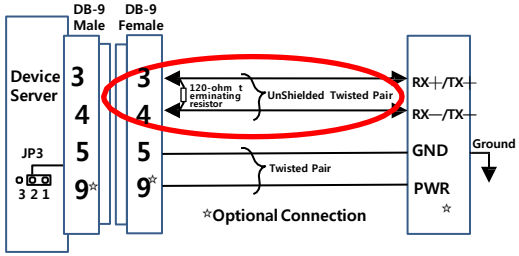
● 종단저항

Pin	설명
3	TX+ (Transmit) Output
4	TX- (Transmit) Output
2	RX+(Receive) Input
6	RX- (Receive) Input
5	Ground
9	+5 VDC Power Input (optional)

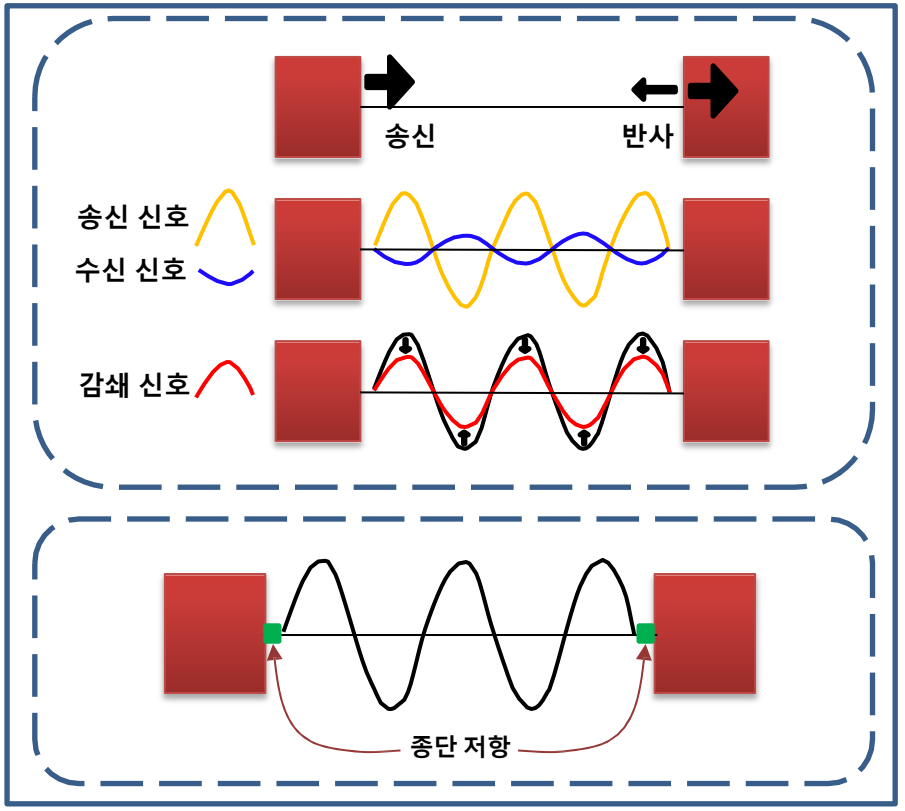


RS-422 방식 종단처리 저항

Pin	설명
3	TX+/RX+ (Bi-Directional) Output
4	TX-/RX- (Bi-Directional) Output
5	Ground
9	+5 VDC Power Input (optional)

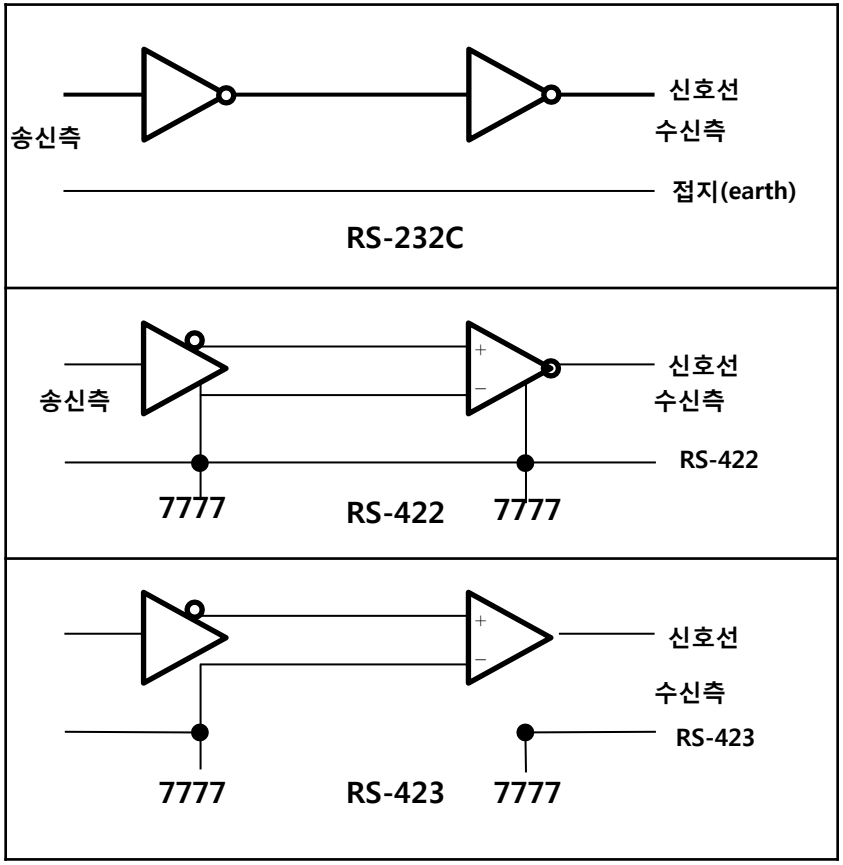


RS-485 방식 종단처리 저항

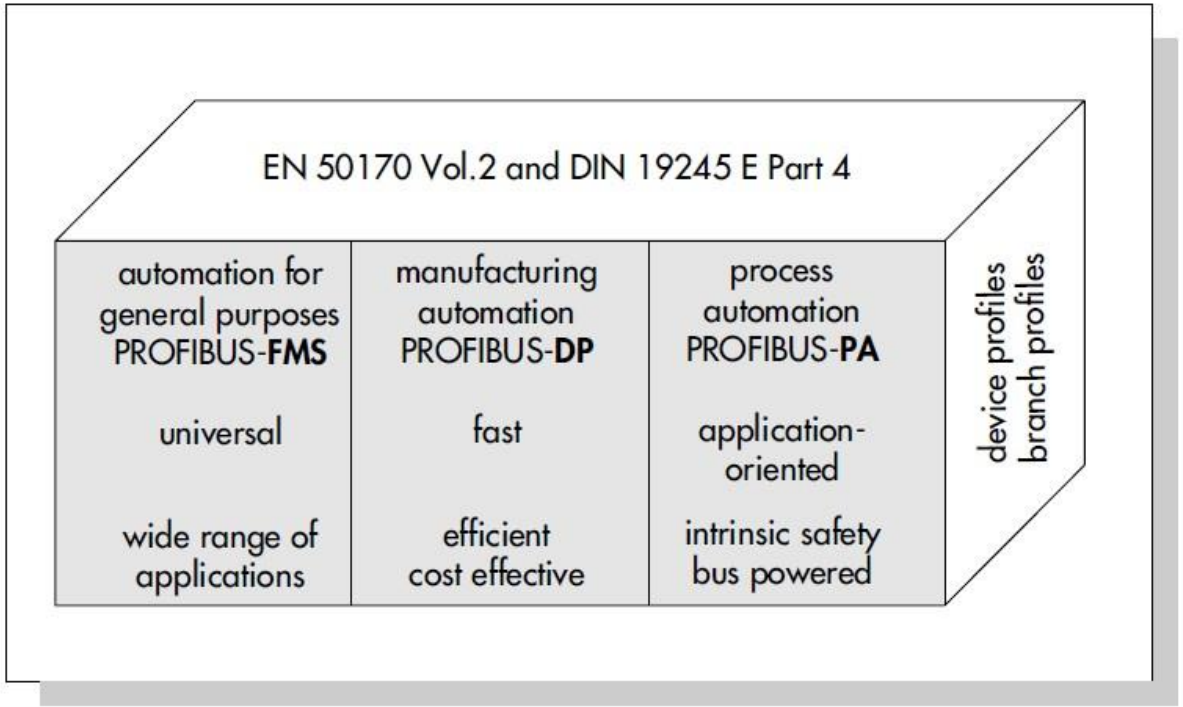


❖ 부록

- **RS423**


송신측
신호선
수신측
접지(earth)
RS-232C
송신측
신호선
수신측
RS-422
7777
RS-422
7777
신호선
수신측
RS-423
7777
RS-423
7777

- **PROFIBUS**



PROFIBUS-FMS (Fieldbus Message Specification) General Solution PROFIBUS-DP (Decentralized Periphery) High Speed Solution (1Kbyte 2mls) PROFIBUS-PA (Process Automation) Explosion Hazardous Area Application

- **PROFIBUS**

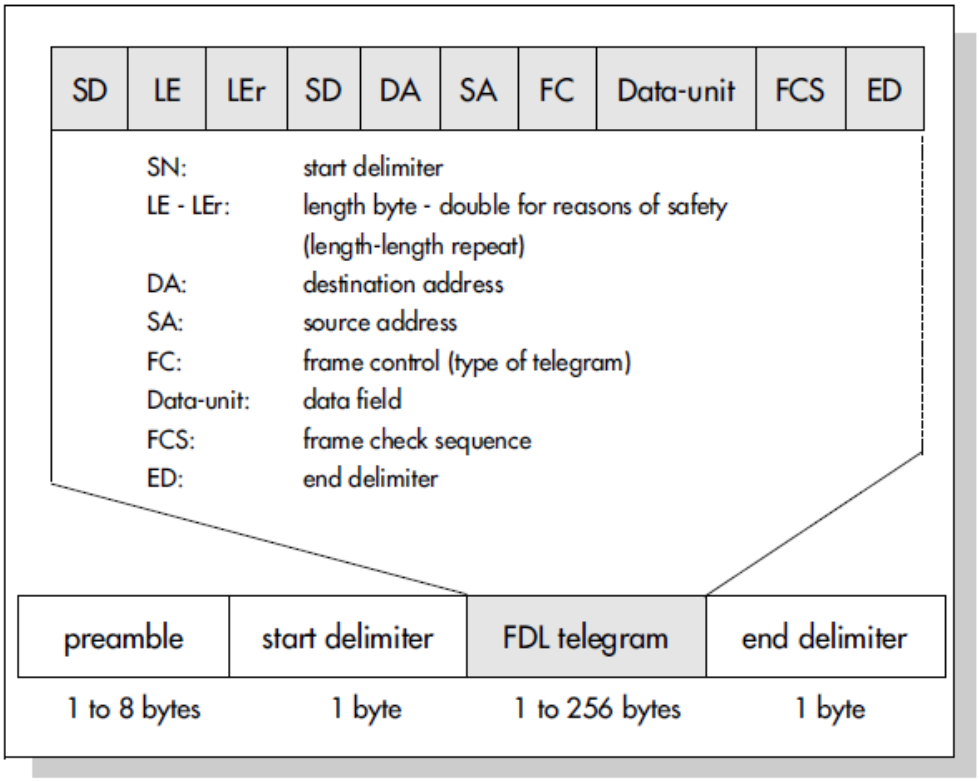
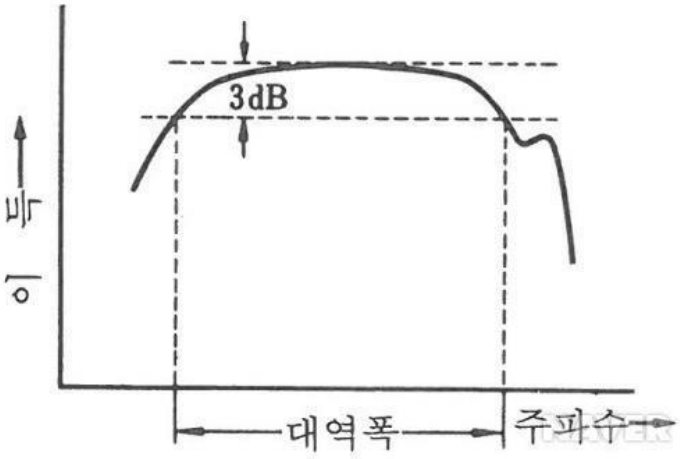


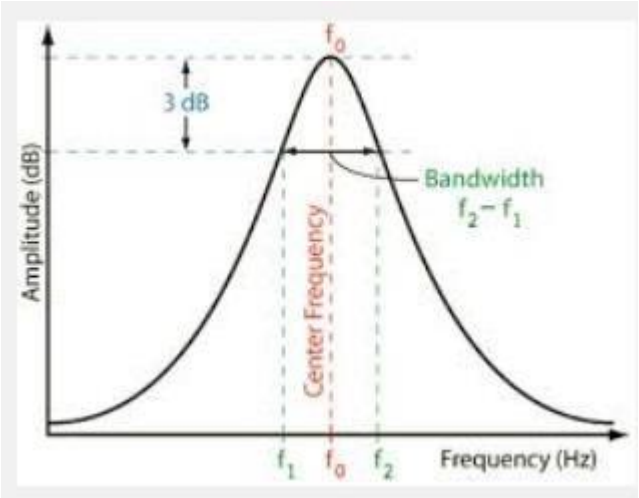
Fig. 17: Bitsynchronous transmission of IEC telegram (bottom) and structure of the embedded FDL telegram

❖ 부록

- **Bandwith(대역폭)**

  데이터통신에서는 어떤 특성의 주파수에 대한 응답에서 특정한 범위로 감소하기까지의 주파수 범위를 말한다. 대역폭은 일반적으로 응답이 기준값에 대하여 3dB 감소하는 점에서 정의된다.

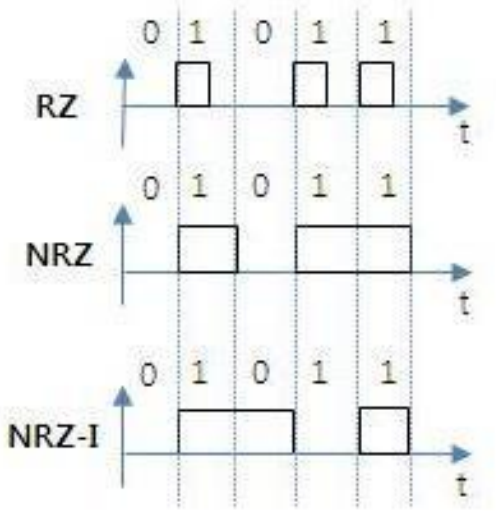




bandwidth ; 대역폭 네트웍에서 이용할 수 있는 신호의 최고 주파수와 최저 주파수의 차이를 말한다. 일반적으로는 통신에서 이용 가능한 최대 전송속도, 즉 정보를 전송할 수 있는 능력을 뜻하며, 그 기본 단위로는 bps를 사용한다. 모뎀에서 전송속도가 28.8 Kbps라는 것은 초당 28,800 비트를 전송할 수 있다는 것을 의미한다. 보통 14.4 ~ 28.8 Kbps 정도는 문자열을 보내고 받기에 적당하고, 음악이나 동영상 같은 멀티미디어 자료를 전송 받으려면 ISDN과 같은 고속 회선을 사용하는 것이 좋다. 전화선을 통한 정보 전송은 이론적으로 수십 Mbps까지 가능하지만, 전화국의 교환기 등에서 대역폭을 64 Kbps로 제한하고 있다.

- **RZ/NRZ/NRZ-I**

  선로 부호화 (encoding) 방식



- **MODBUS Data model**

MODBUS bases its data model on a series of tables that have distinguishing characteristics.

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

The distinctions between inputs and outputs, and between bit-addressable and wordaddressable data items, do not imply any application behavior. It is perfectly acceptable, and very common, to regard all four tables as overlaying one another, if this is the most natural interpretation on the target machine in question.

For each of the primary tables, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span multiple consecutive data items up to a data size limit which is dependent on the transaction function code.

It's obvious that all the data handled via MODBUS (bits, registers) must be located in device application memory. But physical address in memory should not be confused with datareference. The only requirement is to link data reference with physical address. MODBUS logical reference numbers, which are used in MODBUS functions, are unsigned integer indices starting at zero.

- **MODBUS Data model**

The pre-mapping between the MODBUS data model and the device application is totally vendor device specific.

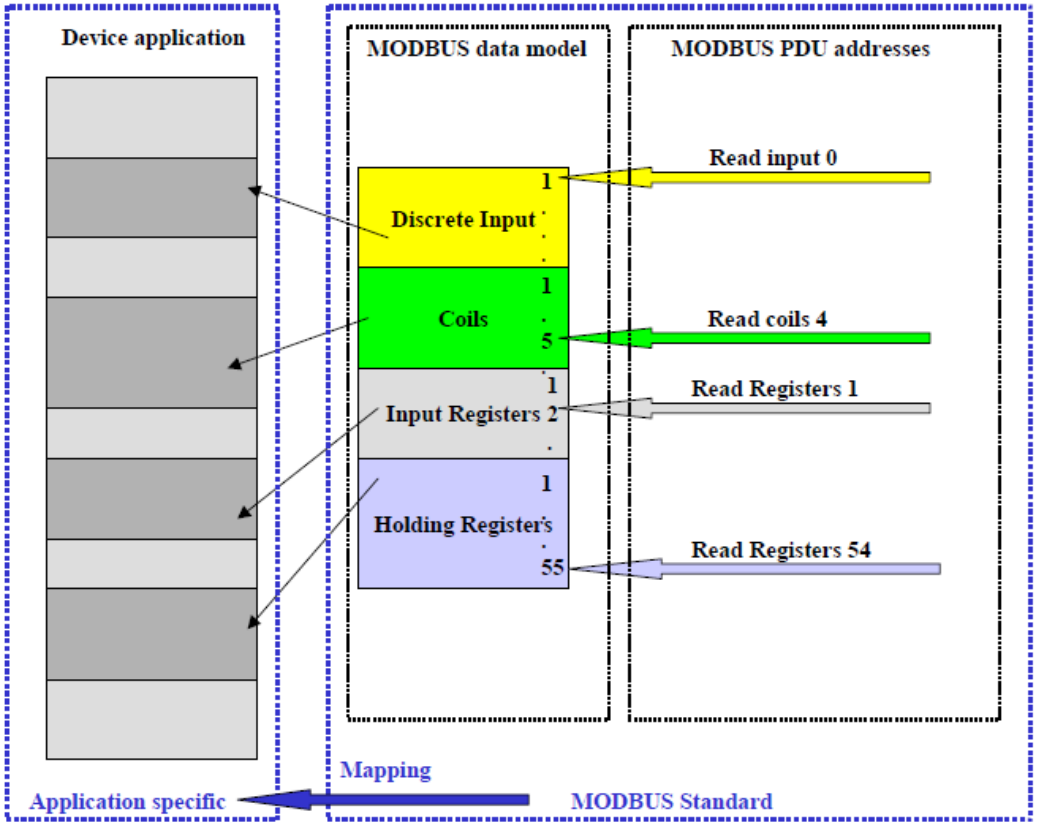
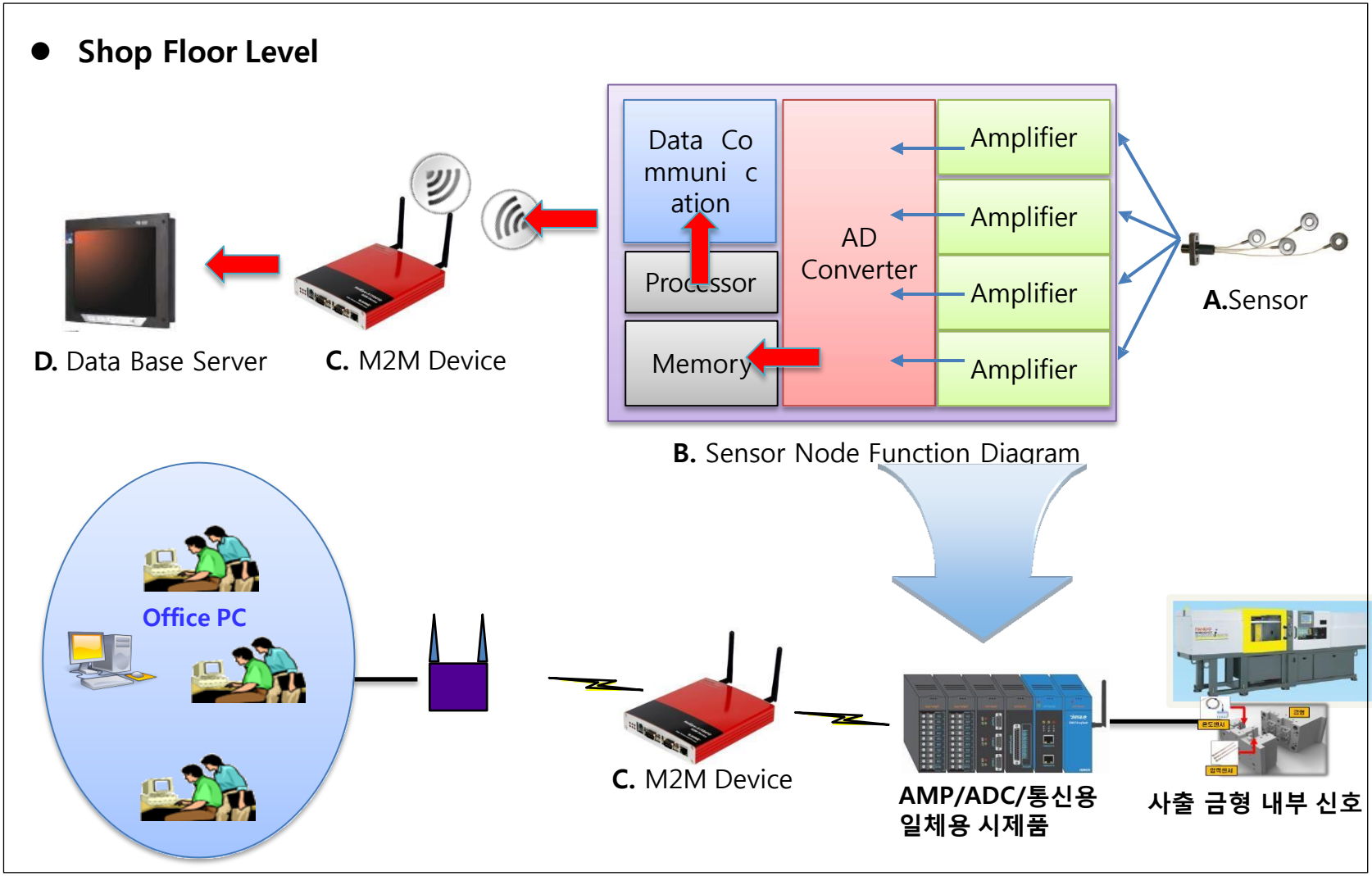


Figure 8 MODBUS Addressing model

❖ 부록


Shop Floor Level
Data Co mmuni c ation
Processor
Memory
AD Converter
Amplifier
Amplifier
Amplifier
Amplifier
A.Sensor
D. Data Base Server
C. M2M Device
B. Sensor Node Function Diagram
Office PC
C. M2M Device
AMP/ADC/통신용 일체용 시제품
사출 금형 내부 신호

❖ 부록

- **Spec**

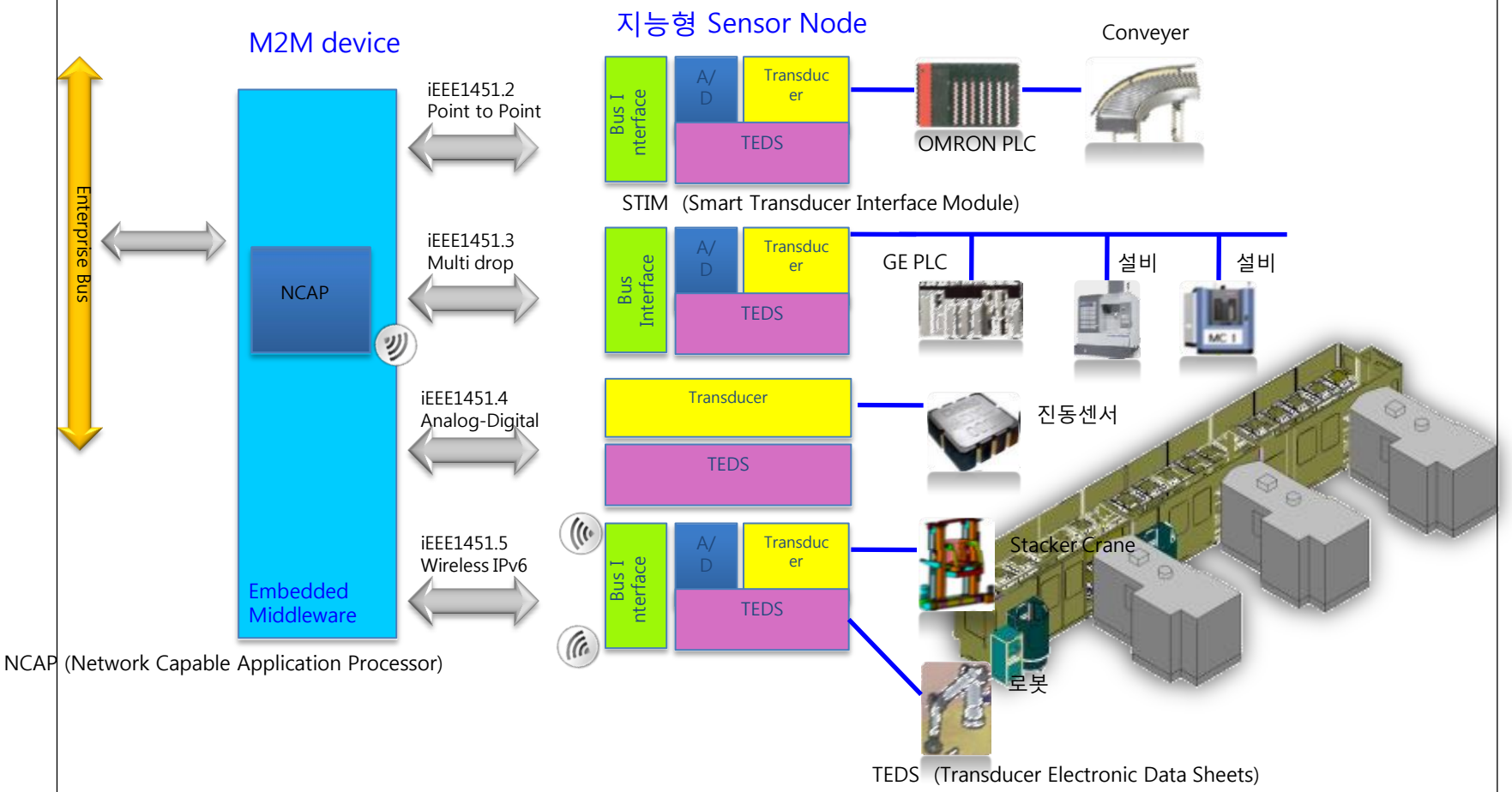
Table I. Output Voltage vs. Thermocouple Temperature (Ambient +25°C, V_S = –5 V, +15 V)

Thermocouple Temperature °C	Type J Voltage mV	AD594 Output mV	Type K Voltage mV	AD595 Output mV
–200	–7.890	–1523	–5.891	–1454
–180	–7.402	–1428	–5.550	–1370
–160	–6.821	–1316	–5.141	–1269
–140	–6.159	–1188	–4.669	–1152
–120	–5.426	–1046	–4.138	–1021
–100	–4.632	–893	–3.553	–876
–80	–3.785	–729	–2.920	–719
–60	–2.892	–556	–2.243	–552
–40	–1.960	–376	–1.527	–375
–20	–.995	–189	–.777	–189
–10	–.501	–94	–.392	–94
0	0	3.1	0	2.7
10	.507	101	.397	101
20	1.019	200	.798	200
25	1.277	250	1.000	250
30	1.536	300	1.203	300
40	2.058	401	1.611	401
50	2.585	503	2.022	503
60	3.115	606	2.436	605
80	4.186	813	3.266	810
100	5.268	1022	4.095	1015
120	6.359	1233	4.919	1219
140	7.457	1445	5.733	1420
160	8.560	1659	6.539	1620
180	9.667	1873	7.338	1817
200	10.777	2087	8.137	2015
220	11.887	2302	8.938	2213
240	12.998	2517	9.745	2413
260	14.108	2732	10.560	2614
280	15.217	2946	11.381	2817
300	16.325	3160	12.207	3022
320	17.432	3374	13.039	3227
340	18.537	3588	13.874	3434
360	19.640	3801	14.712	3641
380	20.743	4015	15.552	3849
400	21.846	4228	16.395	4057
420	22.949	4441	17.241	4266
440	24.054	4655	18.088	4476
460	25.161	4869	18.938	4686
480	26.272	5084	19.788	4896

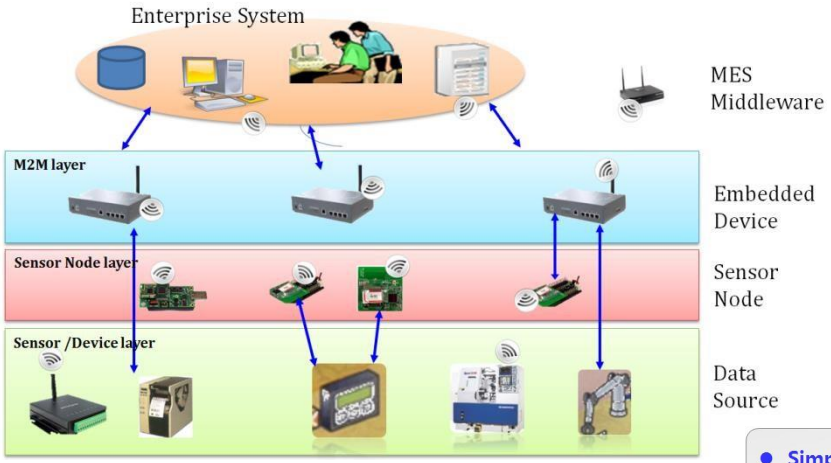
Thermocouple Temperature °C	Type J Voltage mV	AD594 Output mV	Type K Voltage mV	AD595 Output mV
500	27.388	5300	20.640	5107
520	28.511	5517	21.492	5318
540	29.642	5736	22.346	5529
560	30.782	5956	23.198	5740
580	31.933	6179	24.050	5950
600	33.096	6404	24.902	6161
620	34.273	6632	25.751	6371
640	35.464	6862	26.599	6581
660	36.671	7095	27.445	6790
680	37.893	7332	28.288	6998
700	39.130	7571	29.128	7206
720	40.382	7813	29.965	7413
740	41.647	8058	30.799	7619
750	42.283	8181	31.214	7722
760	–	–	31.629	7825
780	–	–	32.455	8029
800	–	–	33.277	8232
820	–	–	34.095	8434
840	–	–	34.909	8636
860	–	–	35.718	8836
880	–	–	36.524	9035
900	–	–	37.325	9233
920	–	–	38.122	9430
940	–	–	38.915	9626
960	–	–	39.703	9821
980	–	–	40.488	10015
1000	–	–	41.269	10209
1020	–	–	42.045	10400
1040	–	–	42.817	10591
1060	–	–	43.585	10781
1080	–	–	44.439	10970
1100	–	–	45.108	11158
1120	–	–	45.863	11345
1140	–	–	46.612	11530
1160	–	–	47.356	11714
1180	–	–	48.095	11897
1200	–	–	48.828	12078
1220	–	–	49.555	12258
1240	–	–	50.276	12436
1250	–	–	50.633	12524

❖ 부록

- **Sensor Node**



● 지능형 Sensor Node



- Simple Device -→ Smart Device
- 자기인식 기능
- 무선 데이터 수집
- Multi Hop 기능

- ●MSP430, CC2420 설계기술
- ●Interrupt 처리 기술
- ●Multi-hop Protocol
- ●TinyOS 구조 이해 및 구현 기술
- ●nesC 프로그래밍 기술
- ●ADC Device Driver 개발 기술
- ●UART Component 사용 기술
- ●손쉬운 fusing과 개발환경
- ●Open Source
- ●AMP 기술