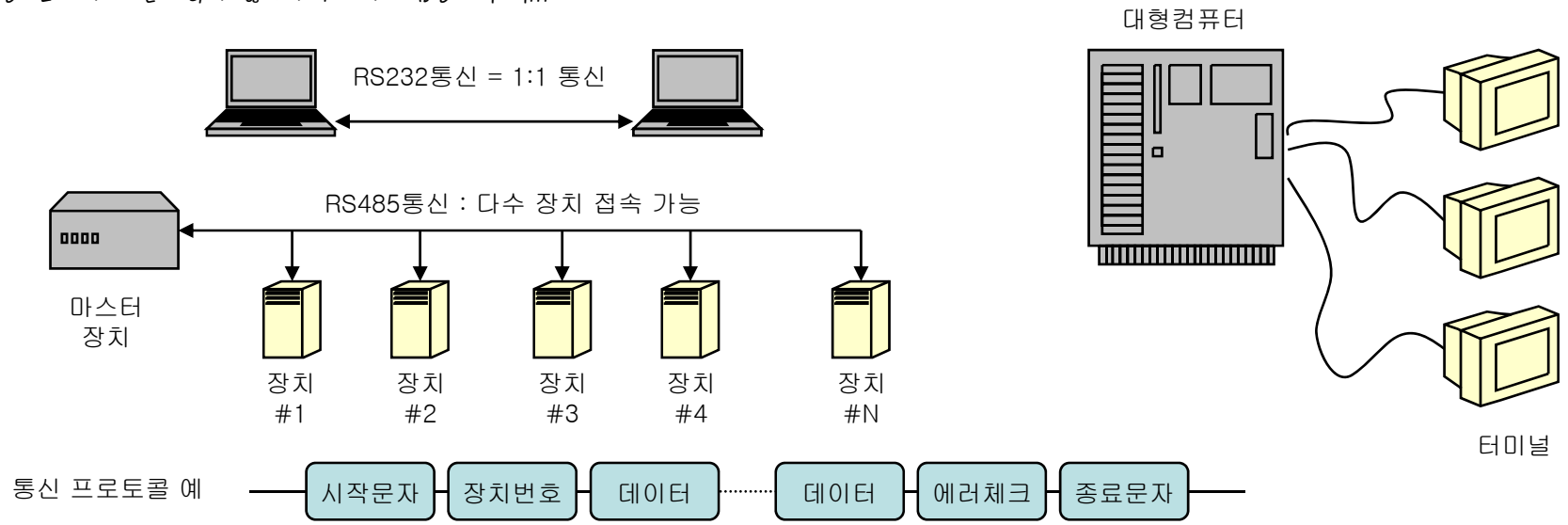
통신 프로토콜

쉽게 이해하는 통신 이야기

2009.5.21 KCO

RealSYS

컴퓨터 COM 포트에서 사용하는 비 동기 통신(UART)은 1문자 단위로 데이터가 전송됩니다.
처음에 대형 컴퓨터와 터미널 장치를 사용하는 경우에는 1문자 단위로도(패리티 체크 사용) 아주 잘 사용하였지만 지금처럼 개인용 컴퓨터가 널리 보급되고, 각종 소형 마이크로 컨트롤러가 많이 사용하는 경우 1문자 단위의 통신보다는 일련의 프로토콜을 형성하여 데이터를 주고 받아야 합니다. 통신 동작 중에 노이즈로 인한 데이터 손상이 있을 수 있으므로 에러 체크 기능도 있어야겠지요.
RS232C 통신처럼 1:1로 연결하는 경우 여러 장치를 함께 연결하는 경우에 많은 통신포트(멀티 포트 장치 같은)를 필요로 하고 비용이 증가하므로 하나로 통신 선로에 여러 대 접속이 가능한 RS485/422 같은 통신을 많이 사용합니다.
이 경우에 통신 프로토콜 상에 각 장치를 구분할 수 있는 장치 번호 개념이 필요합니다.
통신 프로토콜은 업체마다 개인마다 각각 다르게 만들어서 사용이 가능한데, 프로토콜이 다른 경우에는 함께 접속하여 사용하지 못하므로 일종의 표준 프로토콜이 필요하게 됩니다.
표준 규격 협회 등에서 제정하여 발표하는 경우도 있고, 특정 업체에서 널리 보급시키는 경우도 있습니다.
IEEE 등에서 각종 스펙이 나오기도 하고, 모드버스처럼 모디콘 사에서 널리 보급시킨 경우도 있습니다.
표준 프로토콜은 여러 가지 상황을 고려하여 제작되므로 좀 복잡하게 되겠지요?. 제품들을 함께 접속하기 위해 표준 프로토콜을 반드시 따라서 제작하는 경우도 있고, 함께 링크하지 않고 자체적으로 간단하게 프로토콜을 제작하여 사용하는 경우도 있겠지요. 표준을 따르지 않는다 하여 꼭 나쁘다 볼 수 만은 없고, 필요에 따라 간단하게 제작 사용하는 것이 경제적으로 유리할 수도 있습니다. 하지만 가급적 표준을 따르면 더욱 파급효과가 있습니다.
글로만 쓰니까 별로 읽기 싫어지죠? 쓰기도 힘들고 헥 헥!!!



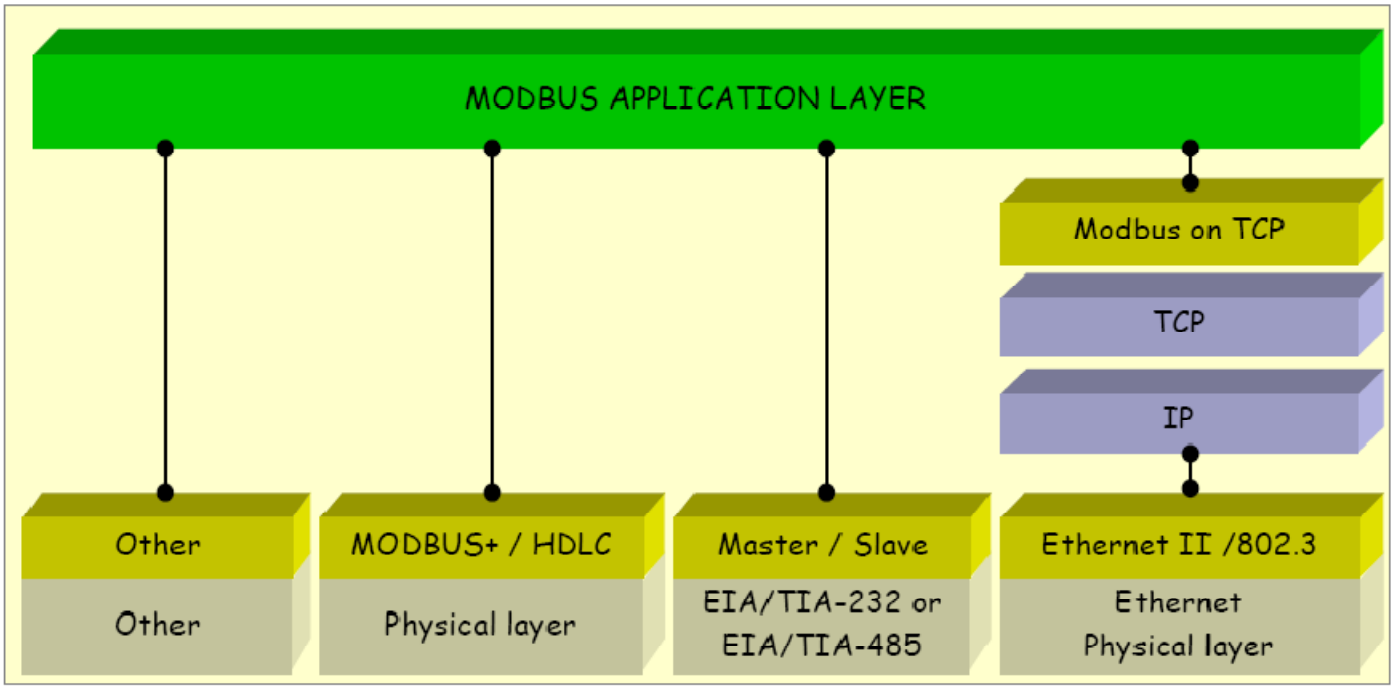
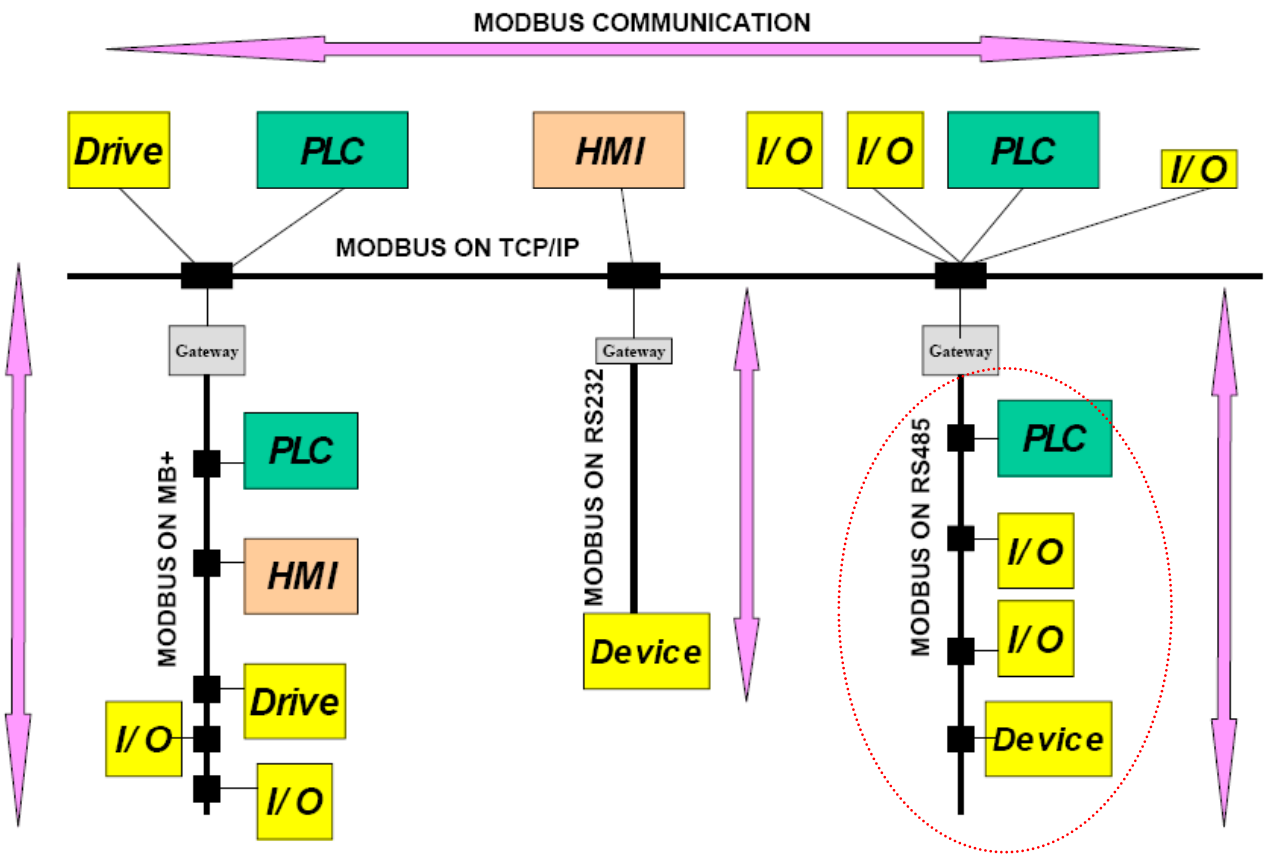


Figure 1: MODBUS communication stack

MODBUS is an application layer messaging protocol for client/server communication between devices connected on different types of buses or networks.

* TCP/IP over Ethernet. See MODBUS Messaging Implementation Guide V1.0a.
•Asynchronous serial transmission over a variety of media
(wire : EIA/TIA-232-E, EIA-422, EIA/TIA-485-A; fiber, radio, etc.)
* MODBUS PLUS, a high speed token passing network.

참조: MODBUS Application Protocol Specification V1.1b 스펙



본 강좌에서는 RS485 통신 부분에 대해서 주로 살펴봅니다.

약어 살펴보기:

HMI Human Machine Interface
I/O Input/Output
TCP Transport Control Protocol
IP Internet Protocol
PLC Programmable Logic Controller
MAC Medium Access Control
MB MODBUS Protocol
MBAP MODBUS Application Protocol
PDU Protocol Data Unit
ADU Application Data Unit
IETF Internet Engineering Task Force
HDLC High level Data Link Control

모드버스는 RTU 및 ASCII 프로토콜이 있습니다. RTU는 공백으로 프레임을 구분하며, 0x00 ~ 0xFF 문자를 사용하므로 통신 시간이 단축됩니다.
ASCII 모드는 특수 시작문자(콜론 ':') 와 끝 문자(CR, LF) 로 프레임을 구분하며 RTU 모드에 비해 통신 시간이 더 걸립니다.
전송 효율 면에서 RTU가 유리하므로 일반적으로 RTU 모드를 많이 사용합니다.

MODBUS ASCII 통신 모드

구분	시작	국번	기능코드	데이터	LRC	끝
값	':'	XX	XX	X...X	HI LO	CR LF
바이트	1	2	2	N	2	2

LRC 계산 범위

- ASCII 데이터를 사용하여 통신
- 시작 문자: 콜론(':', 0x3A)
- 끝 문자: CR(0x0D) LF(0x0A)

LRC를 사용하여 에러 체크
LRC 생성 방법
<1> 시작 문자(':')와 끝 문자(CR LF)를 제외한 모든 문자를 더하여 하위 8비트 데이터를 취함
<2> 2의 보수(반전 후 1 더하기)로 만듦
<3> ASCHEX 형태로 High Low 순서로 전송

MODBUS RTU 통신 모드

구분	국번	기능코드	데이터	CRC
값	XX	XX	X...X	HI LO
바이트	1	1	N	2

CRC 계산 범위

- 16진수 데이터를 사용하여 통신
- 시작 및 끝 문자는 별도로 없으며 국번으로 시작하고 CRC로 끝남
- 프레임의 구분은 공백 시간(3.5 문자 시간)을 사용
- CRC를 사용하여 에러 체크

1문자의 구성

PARITY CHECK의 경우

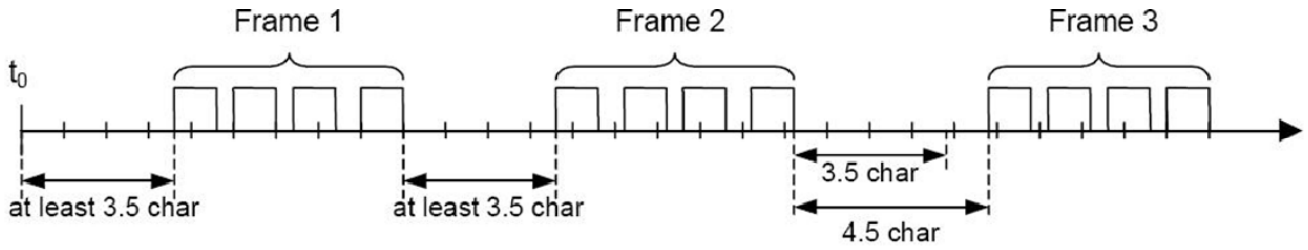
START	LSB(0)	1	2	3	4	5	6	MSB(7)	PARITY	STOP

NO PARITY의 경우

START	LSB(0)	1	2	3	4	5	6	MSB(7)	STOP	STOP

국번: 00 ~ 0xFF 까지 대응, 0번은 Broadcast 용도로 사용,
0번은 모든 슬레이브가 인식은 하지만 응답은 하지 않음.
따라서 국번은 0x01 ~ 0xFF 까지 설정 가능.

RTU(Remote Terminal Unit) Mode



MODBUS message

Start	Address	Function	Data	CRC Check	End
≥ 3.5 char	8 bits	8 bits	N x 8 bits	16 bits	≥ 3.5 char

ASCII Mode

Start	Address	Function	Data	LRC	End
1 char :	2 chars	2 chars	0 up to 2x252 char(s)	2 chars	2 chars CR,LF

FRAME ERROR CHECKING

1. **CRC (Cyclical Redundancy Checking) => RTU**
2. **LRC (Longitudinal Redundancy Checking)=>ASCII**

여러 가지 기능 코드가 있는데, 16비트 단위의 04(Read Input Register) 03(Read Holding Register), 06(Write Single Register) 16(Write Multiple Register) 기능을 주로 사용 함.

				Function Codes			
				code	Sub code	(hex)	Section
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02	6.2
		Internal Bits Or Physical coils	Read Coils	01		01	6.1
			Write Single Coil	05		05	6.5
			Write Multiple Coils	15		0F	6.11
	16 bits access	Physical Input Registers	Read Input Register	04		04	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03	6.3
			Write Single Register	06		06	6.6
			Write Multiple Registers	16		10	6.12
			Read/Write Multiple Registers	23		17	6.17
			Mask Write Register	22		16	6.16
			Read FIFO queue	24		18	6.18
	File record access		Read File record	20		14	6.14
			Write File record	21		15	6.15
	Diagnostics		Read Exception status	07		07	6.7
			Diagnostic	08	00-18,20	08	6.8
			Get Com event counter	11		OB	6.9
			Get Com Event Log	12		0C	6.10
			Report Slave ID	17		11	6.13
			Read device Identification	43	14	2B	6.21
Other			Encapsulated Interface Transport	43	13,14	2B	6.19

December 28, 2006 http://www.Modbus-IDA.org 11/51

03 (0x03) Read Holding Registers

기능 코드3은 출력 데이터 값을 읽는 기능으로
데이터는 16비트 크기이고, 시작 번지와 개수로 입력하면
응답으로 해당번지부터 요구한 개수 만큼의 출력 데이터가 응답 됨.

Request

Function code	1 Byte	**0x03**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	**0x03**
Byte count	1 Byte	2 x **N***
Register value	**N*** x 2 Bytes	

***N** = Quantity of Registers

Error

Error code	1 Byte	**0x83**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read registers 108 – 110:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**03**	Function	**03**
Starting Address Hi	**00**	Byte Count	**06**
Starting Address Lo	**6B**	Register value Hi (108)	**02**
No. of Registers Hi	**00**	Register value Lo (108)	**2B**
No. of Registers Lo	**03**	Register value Hi (109)	**00**
		Register value Lo (109)	**00**
		Register value Hi (110)	**00**
		Register value Lo (110)	**64**

04 (0x04) Read Input Registers

기능 코드4는 입력 상태 값을 읽는 기능으로 데이터는 16비트 크기이고,
시작 번지와 개수로 입력하면 응답으로 해당번지부터 요구한 개수 만큼의
입력 데이터가 응답 됨.

Request

Function code	1 Byte	**0x04**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Input Registers	2 Bytes	0x0001 to 0x007D

Response

Function code	1 Byte	**0x04**
Byte count	1 Byte	2 x **N***
Input Registers	**N*** x 2 Bytes	

***N** = Quantity of Input Registers

Error

Error code	1 Byte	**0x84**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read input register 9:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**04**	Function	**04**
Starting Address Hi	**00**	Byte Count	**02**
Starting Address Lo	**08**	Input Reg. 9 Hi	**00**
Quantity of Input Reg. Hi	**00**	Input Reg. 9 Lo	**0A**
Quantity of Input Reg. Lo	**01**		

06 (0x06) Write Single Register

기능 코드6은 하나의 16비트 크기의 출력 값을 쓰는 기능으로
해당 번지와 데이터를 전송하면 같은 형태로 응답 함.

Request

Function code	1 Byte	**0x06**
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 to 0xFFFF

Response

Function code	1 Byte	**0x06**
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 to 0xFFFF

Error

Error code	1 Byte	**0x86**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to write register 2 to 00 03 hex:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	06	Function	06
Register Address Hi	00	Register Address Hi	00
Register Address Lo	01	Register Address Lo	01
Register Value Hi	00	Register Value Hi	00
Register Value Lo	03	Register Value Lo	03

16 (0x10) Write Multiple registers

기능 코드16은 다수의 16비트 크기의 출력 값을 쓰는 기능으로
시작 번지와 개수 및 여러 데이터를 전송하면 시작 번지와 개수로 응답 함.

Request

Function code	1 Byte	**0x10**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	0x0001 to 0x007B
Byte Count	1 Byte	2 x **N***
Registers Value	**N*** x 2 Bytes	value

***N** = Quantity of Registers

Response

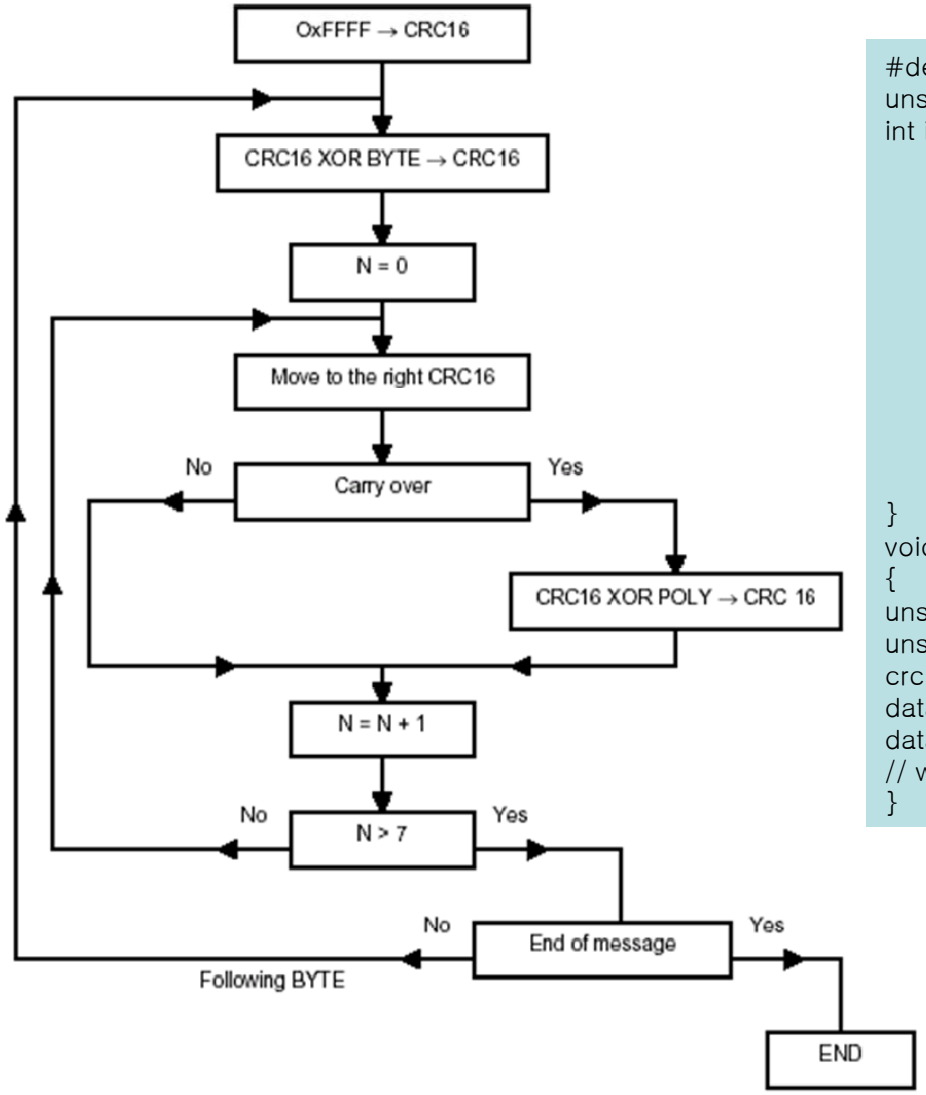
Function code	1 Byte	**0x10**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 123 (0x7B)

Error

Error code	1 Byte	**0x90**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to write two registers starting at 2 to 00 0A and 01 02 hex:

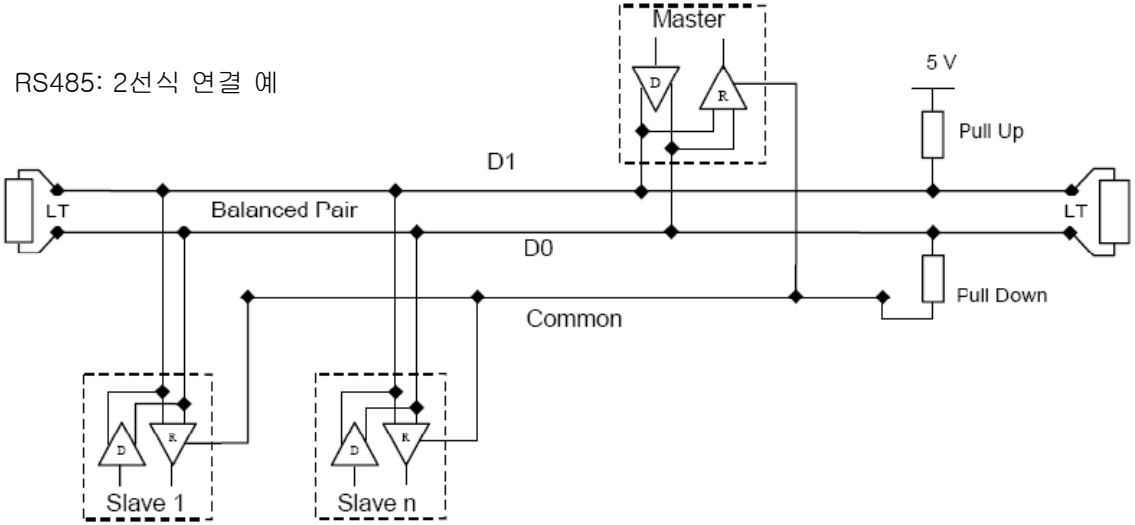
Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	10	Function	10
Starting Address Hi	00	Starting Address Hi	00
Starting Address Lo	01	Starting Address Lo	01
Quantity of Registers Hi	00	Quantity of Registers Hi	00
Quantity of Registers Lo	02	Quantity of Registers Lo	02
Byte Count	04		
Registers Value Hi	00		
Registers Value Lo	0A		
Registers Value Hi	01		
Registers Value Lo	02		




```
#define POLYNORMIAL 0xA001
unsigned short CRC16(unsigned char *puchMsg, int usDataLen){
int i;
                unsigned short crc, flag;
                crc = 0xffff;
                while(usDataLen--){
                        crc ^= *puchMsg++;
                        for (i=0; i<8; i++){
                                flag = crc & 0x0001;
                                crc >>= 1;
                                if(flag) crc ^= POLYNORMIAL;
                        }
                }
                return crc;
}
void main(void)
{
unsigned char data[30] = {0x01, 0x01, 0x00, 0x00, 0x00, 0x03, 0, 0};
unsigned short crc16;
crc16 = CRC16(data, 6);
data[6] = (unsigned char)((crc16>>8) & 0x00ff);
data[7] = (unsigned char)((crc16>>0) & 0x00ff);
// write_comm(data, 8);
}
```

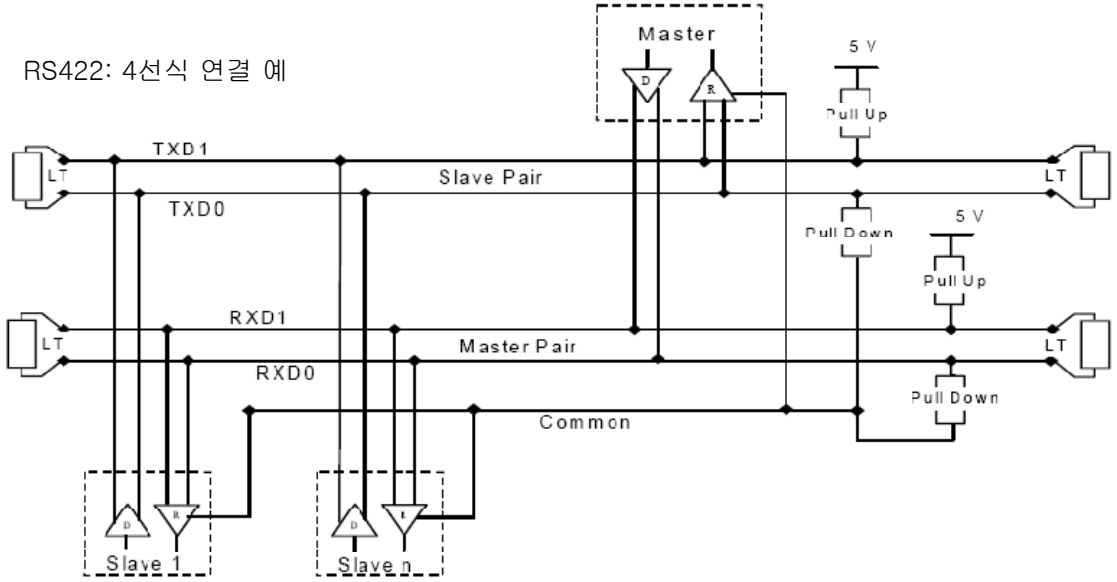
통신 동작에서 노이즈 등으로 데이터가 손상될 수 있으므로 데이터의 손상 여부를 체크 할 수 있는 방법에는 여러 가지가 있습니다.
예를 들면 데이터를 전부 더하여 보내거나 또는 XOR를 취하여 보내거나 등등..
모드 버스 RTU 모드에서는 CRC16 방법을 사용하는데, 위의 함수를 호출하여 사용하시면 됩니다.

RS485: 2선식 연결 예

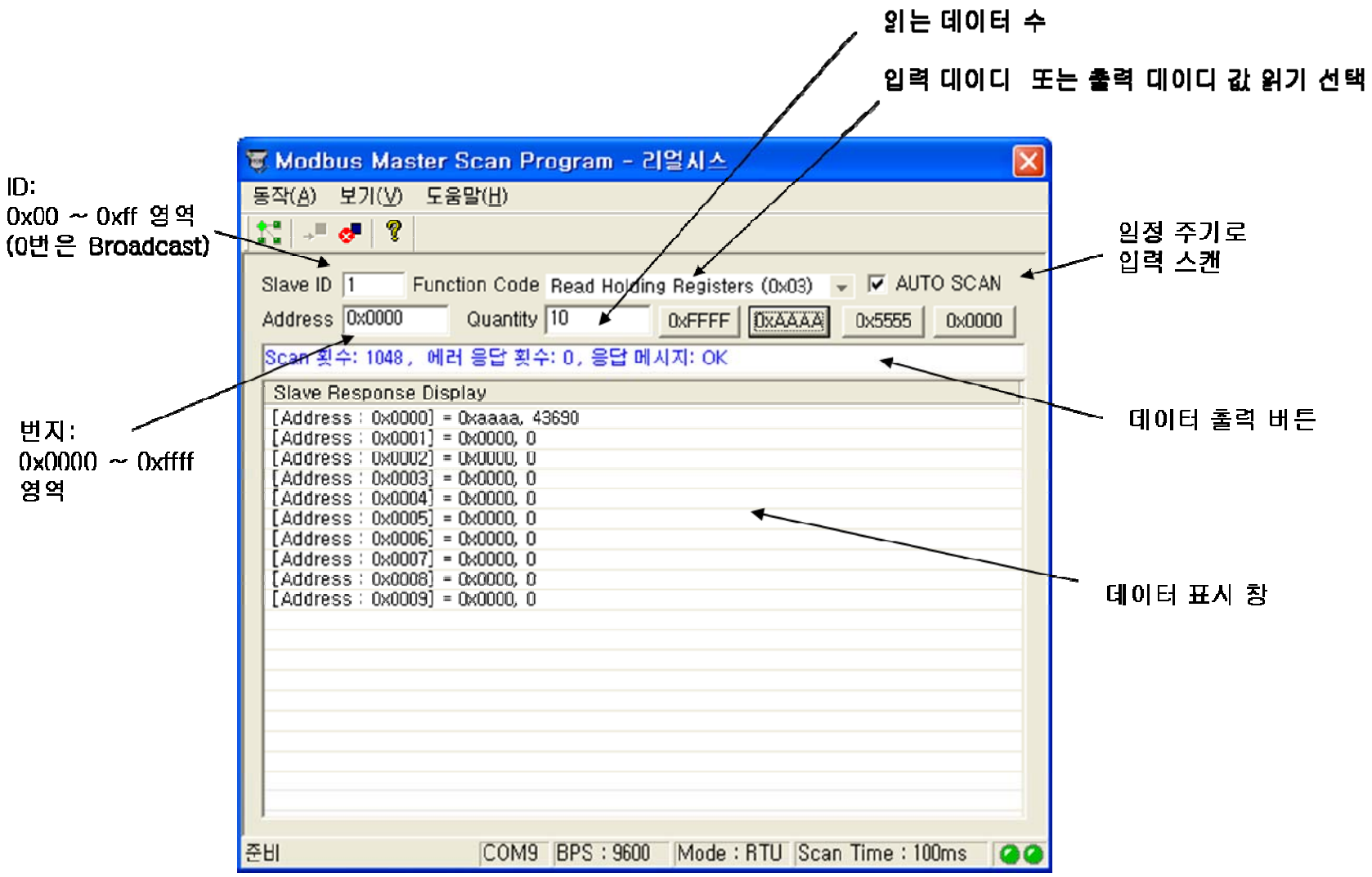


2선식 특징:
- 2개의 통신선 사용
- 마스터와 슬래이브가 대등 구조
- 2선로 양끝에 종단 저항 처리

RS422: 4선식 연결 예



4선식 특징:
- 4개의 통신선 사용
- 마스터의 송신이 모든 슬래이브의 수신으로
  슬래이브의 송신이 모아져서 마스터 수신으로
- 각각의 2선로에 종단 저항 처리



리얼시스(www.realsys.co.kr) 자료실에서 구할 수 있습니다.