RS_DI32 사용 설명서 V1.1

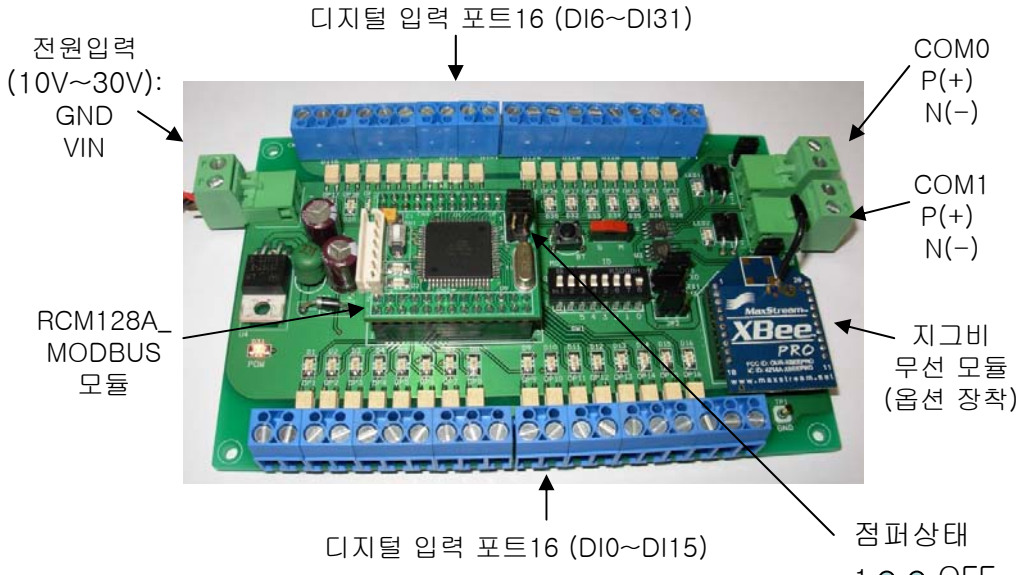
Flex. Com I/O Series

작성 일자: 2008.7.2

RealSYS

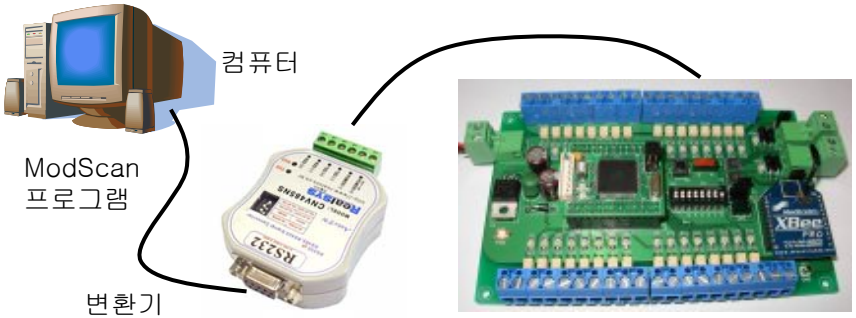
RS_DI32 주요 특징:

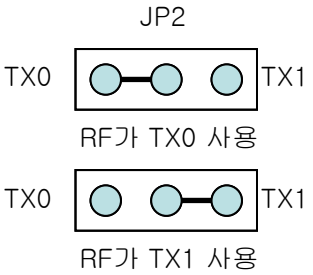
- RCM128A_MODBUS 모듈 장착
- 포토 커플러 절연 입력: 32개
- RS485 통신 포트: 2개 (2개의 마스터 또는 2중화 가능)
  지그비 무선 통신 포트 장착 가능 (COM0/COM1 선택)
- 전원 입력 : DC10V ~ 30V
- 설정 DIP 스위치(8), M/S 설정(1), 버튼(1)
- 표시 LED: 입력(32), 전원(1), 통신(2)
- 보드 크기: 가로 132.7mm x 세로 86.5mm



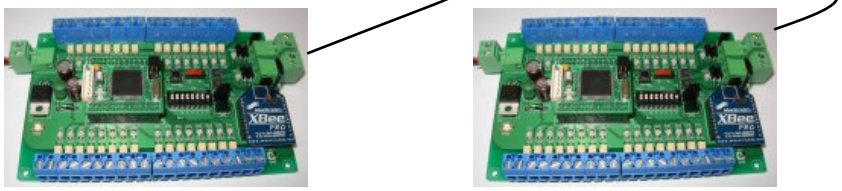
*** 컴퓨터와 연결 ***

컴퓨터에서 출력을 제어하거나 입력 상태를 읽을 수 있음

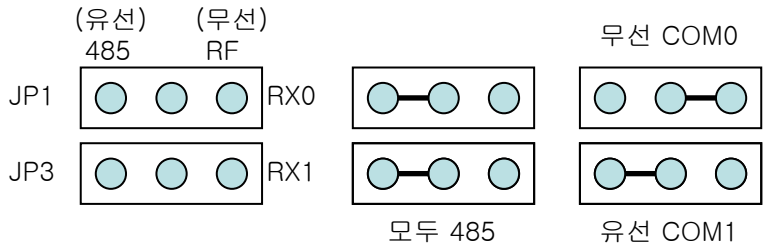




*** 1:1 연결 방법 ***



(RS_DI32와 RS_DO32를 한 쌍으로 사용)



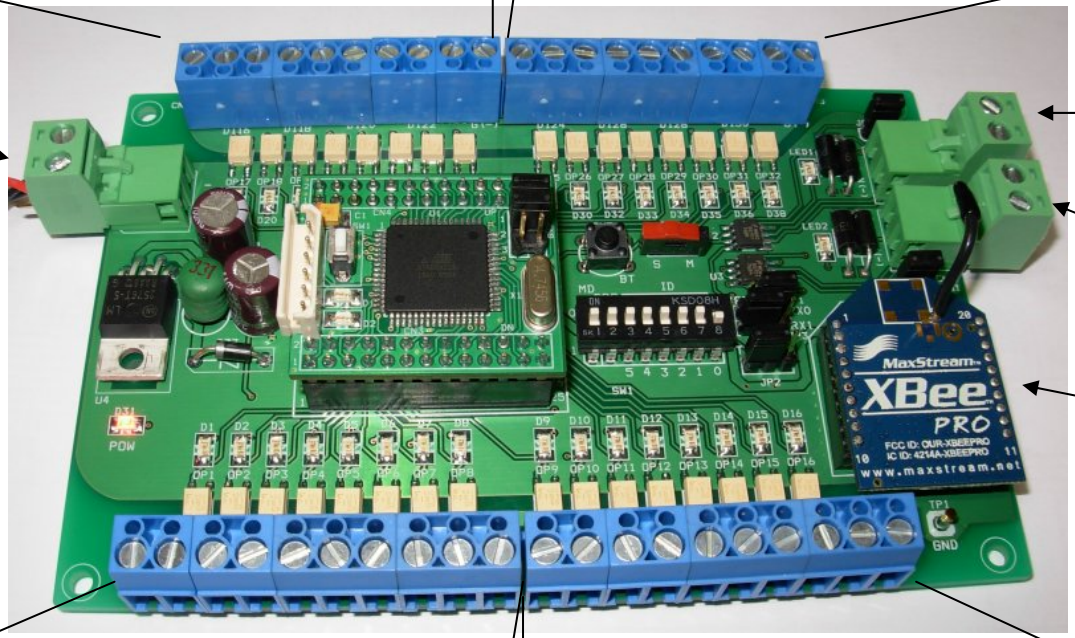
리얼시스(RealSYS) www.realsys.co.kr Tel:031) 342-3000 Fax:031) 343-0003

RS_DI32 보드 입출력 핀

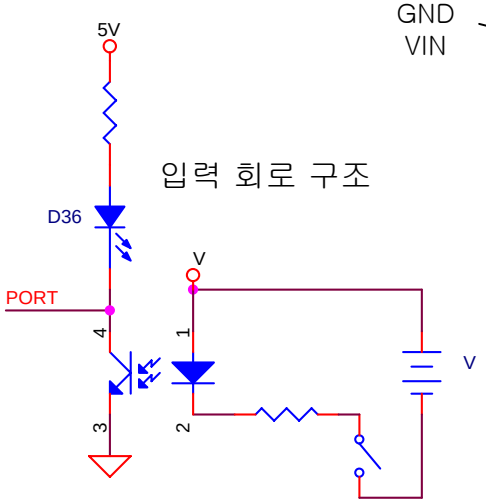
	Digital Input								
V (+)	DI 16	DI 17	DI 18	DI 19	DI 20	DI 21	DI 22	DI 23	G (−)

	Digital Input								
V (+)	DI 24	DI 25	DI 26	DI 27	DI 28	DI 29	DI 30	DI 31	G (−)

전원입력
(10V~30V):
GND
VIN



입력 회로 구조



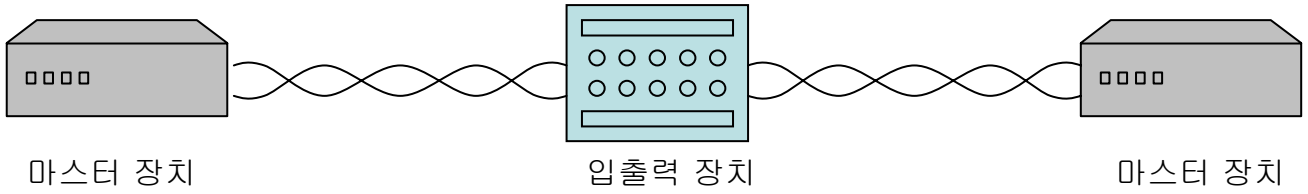
	Digital Input								
V (+)	DI 0	DI 1	DI 2	DI 3	DI 4	DI 5	DI 6	DI 7	G (−)

	Digital Input								
V (+)	DI 8	DI 9	DI 10	DI 11	DI 12	DI 13	DI 14	DI 15	G (−)

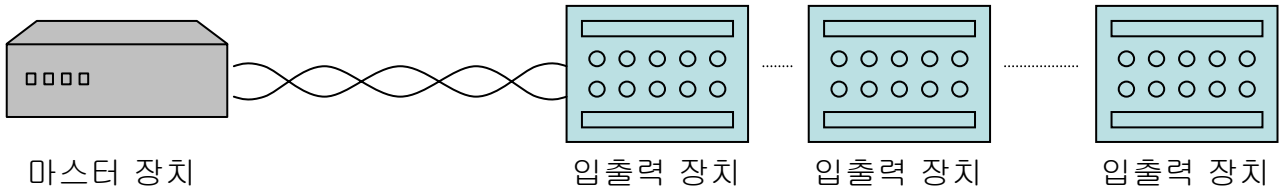
Flex Com I/O 통신 장치의 특징

<1> 2개의 통신 포트 사용: 2중화 또는 2개의 마스터 제어 가능

일반적으로 많이 사용하는 통신 입출력 장치는 하나의 통신 포트를 가지고 1개의 마스터 장치에서만 운영이 가능합니다. 선로의 이중화(1개의 통신 선이 고장 난 경우 바로 다른 하나로 대체) 또는 2곳의 마스터 장치에서도 제어가 가능하도록 2개의 통신 포트로 운영하도록 합니다.

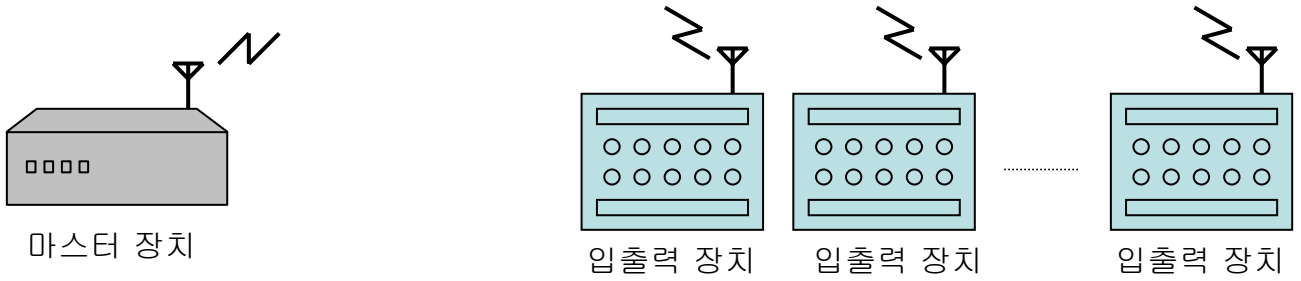


RS485 통신을 사용하여 다수의 장치 접속도 물론 가능



<2> 지그비를 사용한 무선 통신 제어 가능

유선 선로 설치가 곤란한 경우 바로 지그비 무선 통신 모듈을 장착하여 무선으로 원 결 제어가 가능합니다.



<3> 산업용 통신 프로토콜로 가장 많이 사용하는 모드버스 RTU 및 ASCII 프로토콜 사용

통신 장치는 통신 프로토콜이 필요한데, 본 입출력 장치에서는 산업용 통신에 가장 많이 사용하는
모드 버스 RTU 및 ASCII 프로토콜을 사용합니다
모드버스 규약에 대한 부분은 별도로 다룰 예정이며, DIP 스위치를 사용하여 간단하게 변경이 가능합니다.

통신 속도:
위: 9600 bps
아래: 38400 bps

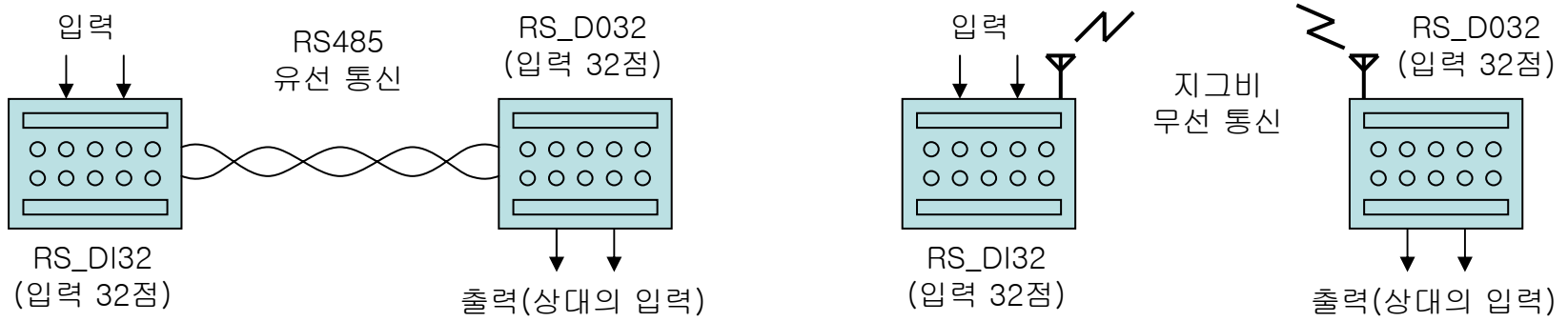
모드:
위: RTU
아래: ASCII

ID:
6비트 64대 구분
모드버스에서는 0번은 특정 ID로 사용하지 않고 Broadcast(동시에 명령)
동작으로 사용하므로 개별 제어 시는 설정하지 않음
(본 제작 모듈에서는 1대 1 전송 모드로 사용)

참조:
8비트 DIP 스위치를 사용하여 1개는 모드, 1개는 통신 속도, 6개는
ID 설정으로 사용하였지만, 사용자의 특정 요구에 따라 변경은 가능합니다.
예를 들어 통신 속도 종류를 많게 하거나 또는 고정하거나,
또는 ID를 더 많게 하거나 등의 변경도 가능 합니다.

<4> 별도의 마스터 장치 없이 1대 1 전송 기능

일반 통신 모듈은 반드시 마스터 장치가 있어야 통신이 가능하지만 본 장치는 유선 또는 무선으로 1:1 통신이 가능합니다.
RS_DI32 장치(마스터 설정)의 32점 입력이 RS_DO32 출력 장치 출력으로 전달합니다.

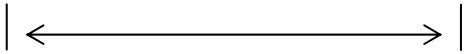


모드 버스 통신 프로토콜

비동기 통신은 1문자 단위로 통신이 됩니다. 일련의 정보를 전송하기 위해서 문자 하나 하나를 엮어서 프레임을 만드는데, 업체마다 개인마다 다양하게 구성이 가능합니다. 모드 버스 통신 프로토콜은 미국의 모디콘 사에서 처음 제안 사용한 방법으로 많은 업체에서 사용되고 있으며, 본 입출력 장치도 이를 기준으로 작성되었습니다.(비트 동작을 위해서 조금 변경 사용)
모드버스는 RTU 및 ACSII 두 가지 모드가 있는데, RTU는 ASCII 방식에 비해 2배정도 효과적입니다.
왜냐하면 0x00~0xFF 순수 값을 그대로 데이터로 사용하므로 송수신 시간이 적게 걸리며, 프레임의 시작과 끝을 공백시간으로 구분합니다. ASCII 모드는 시작 문자(':')와 끝 문자(CR LF)로 구분하고 데이터는 '0'~'9','A'~'F" 코드를 사용하며, 문자 사이에 공백은 많이 있어도 관계가 없습니다.

MODBUS RTU 통신 프로토콜

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

MODBUS ASCII 통신 프로토콜

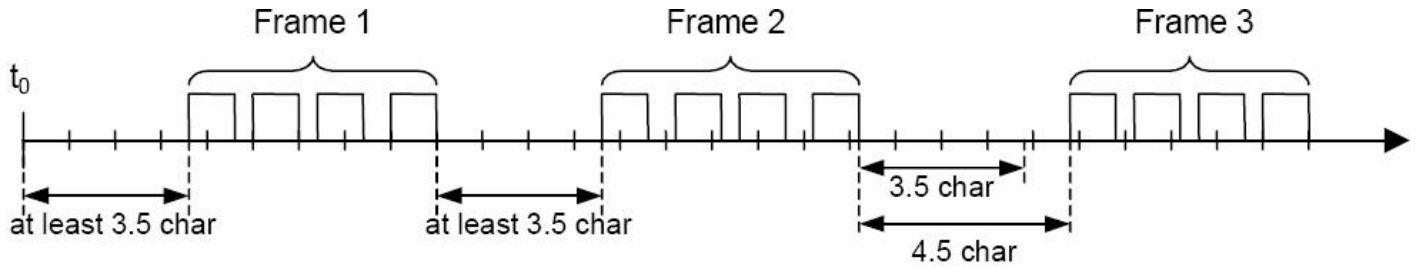
구분	시작	국번	기능코드	데이터	LRC	끝
값	':'	XX	XX	X...X	HI LO	CR LF
바이트	1	2	2	N	2	2

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

국번: 00 ~ 0xFF 까지 대응, 0번은 Broadcast 용도로 사용,
0번은 모든 슬래이브가 인식은 하지만 응답은 하지 않음.
따라서 국번은 1 ~ 247 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

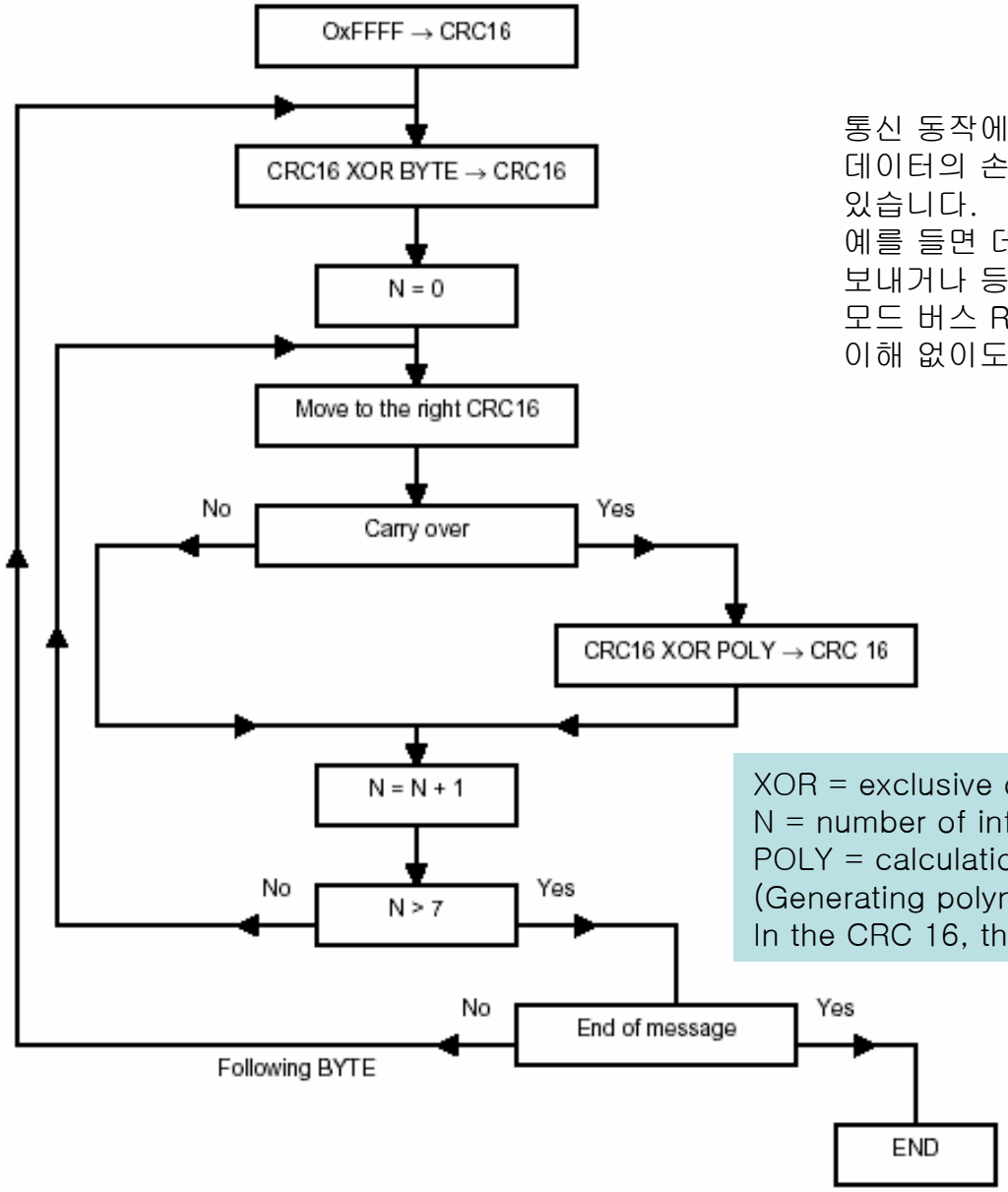
ASCII Mode

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**

리얼시스(RealSYS) www.realsys.co.kr Tel:031) 342-3000 Fax: 031) 343-0003



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, 구체적인 이해 없이도 다음 페이지의 함수를 호출하여 사용하시면 됩니다.

XOR = exclusive or
N = number of information bits
POLY = calculation polynomial of the CRC 16 = 1010 0000 0000 0001
(Generating polynomial = 1 + x2 + x 15 + x 16)
In the CRC 16, the 1st byte transmitted is the least significant one.

MODBUS RTU 통신 모드: CRC 계산 예
프레임: 01 01 00 00 00 03 CRC의 경우

```
#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);
}
```

모드 버스 통신 프로토콜: Fuction 코드

				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

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

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

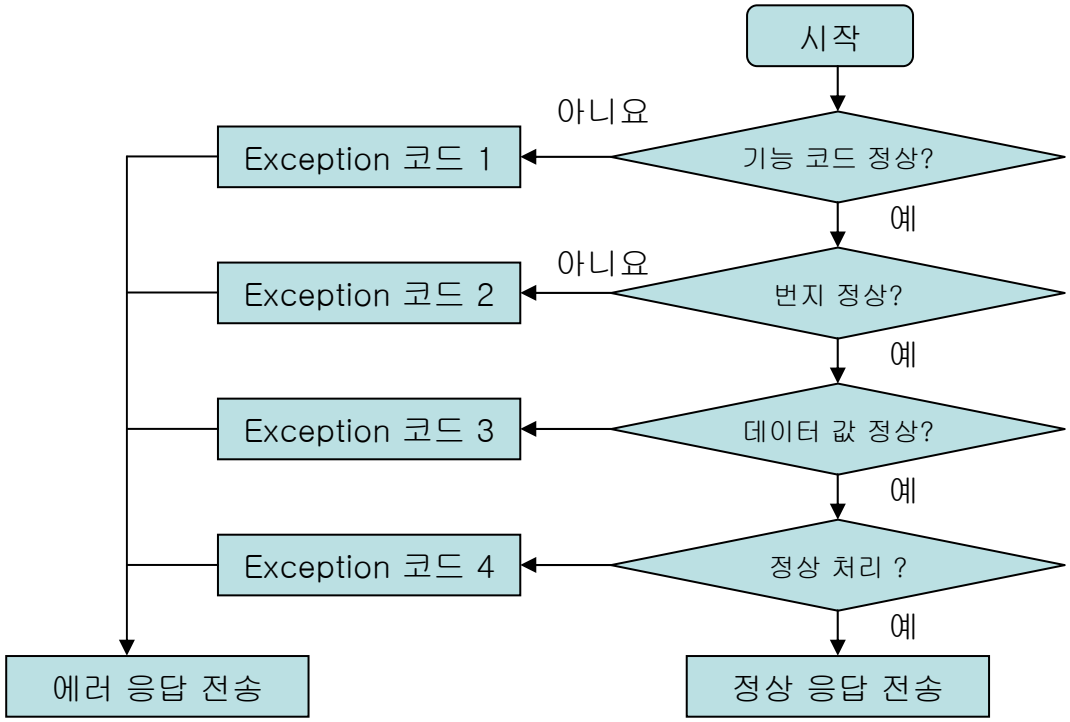
Error code	1 Byte	**0x86**
Exception code	1 Byte	01 or 02 or 03 or 04

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**

에러 Excetion 코드 처리 흐름도

통신 프레임에 에러가 발생하면 다음과 같은 순서에 의거 처리 됨.



01 : Illegal Function
02 : Illegal Data Address
03 : Illegal Data Value
04 : Slave Device Failure
05 : Acknowledge
06 : Slave Device Busy
08 : Memory Parity Error
0A : Gateway Path unavailable
0B : Gateway Target Device Failed to Respond

Read Holding Register (function code 0x03)

Inquiry:

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3a
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x03	0x03	"03"	0x30, 0x33
Starting address high	0x00	0x00	"00"	0x30, 0x30
Starting address low	0x00	0x00	"00"	0x30, 0x30
Number of points high	0x00	0x00	"00"	0x30, 0x30
Number of points low	0x02	0x02	"02"	0x30, 0x32
Error Check (LRC / CRC)	-	0xC4 0xA1	"F0"	0x46, 0x30
End of frame		t1-t2-t3	-	0xD, 0xA

Table 5.34: Example inquiry, Read Holding Register

Reply:

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3A
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x03	0x03	"03"	0x30, 0x33
Byte Count	0x04	0x04	"04"	0x30, 0x34
Data Hi (Register 0)	0x3F	0x3F	"3F"	0x33, 0x46
Data Lo (Register 0)	0xFB	0xFB	"FB"	0x46, 0x42
Data Hi (Register 1)	0x00	0x00	"00"	0x30, 0x30
Data Lo (Register 1)	0x00	0x00	"00"	0x30, 0x30
Error Check (LRC / CRC)	-	0x2D 0x61	"B4"	0x42, 0x34
End of frame	-	t1-t2-t3	-	0xD, 0xA

Table 5.35: Example reply, Read Holding Register

Read Input Register (Function code 0x04)

Inquiry:

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3a
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x04	0x04	"04"	0x30, 0x33
Starting address high	0x00	0x00	"00"	0x30, 0x30
Starting address low	0x00	0x00	"00"	0x30, 0x30
Number of points high	0x00	0x00	"00"	0x30, 0x30
Number of points low	0x02	0x02	"02"	0x30, 0x32
Error Check (LRC / CRC)	-	0xC4 0xA1	"F0"	0x46, 0x30
End of frame		t1-t2-t3	-	0xD, 0xA

Table 5.36: Example enquiry, Read Input Register

Reply:

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3A
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x04	0x04	"04"	0x30, 0x33
Byte Count	0x04	0x04	"04"	0x30, 0x34
Data Hi (Register 0)	0x3F	0x3F	"3F"	0x33, 0x46
Data Lo (Register 0)	0xFB	0xFB	"FB"	0x46, 0x42
Data Hi (Register 1)	0x00	0x00	"00"	0x30, 0x30
Data Lo (Register 1)	0x00	0x00	"00"	0x30, 0x30
Error Check (LRC / CRC)	-	0x2D 0x61	"B4"	0x42, 0x34
End of frame	-	t1-t2-t3	-	0xD, 0xA

Table 5.37: Example reply, Read Input Register

함수 예) Preset Single Register (Function code 0x06):

Inquiry:

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3a
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x06	0x06	"06"	0x30, 0x36
Register address high	0x00	0x00	"00"	0x30, 0x30
Register address low	0x00	0x00	"00"	0x30, 0x30
Preset data high	0x12	0x12	"12"	0x31, 0x32
Preset data low	0x34	0x34	"34"	0x33, 0x34
Error Check (LRC / CRC)	-	0x8C 0x17	"A9"	0x41, 0x39
End of frame		t1-t2-t3	-	0xD, 0xA

Table 5.40: Example inquiry, Preset Single Register

Reply:

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3a
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x06	0x06	"06"	0x30, 0x36
Register address high	0x00	0x00	"00"	0x30, 0x30
Register address low	0x00	0x00	"00"	0x30, 0x30
Preset data high	0x12	0x12	"12"	0x31, 0x32
Preset data low	0x34	0x34	"34"	0x33, 0x34
Error Check (LRC / CRC)	-	0x8C 0x17	"A9"	0x41, 0x39
End of frame		t1-t2-t3	-	0xD, 0xA

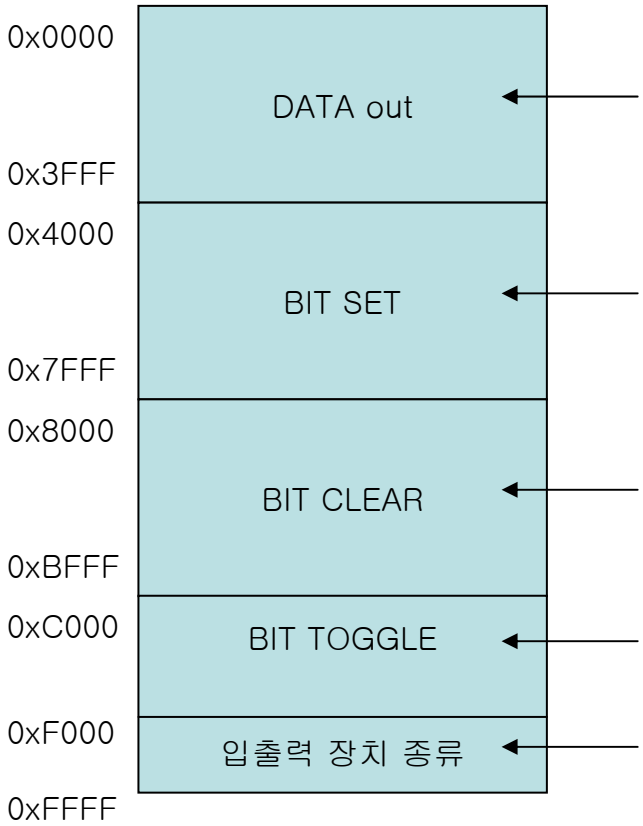
Table 5.41: Example reply, Preset Single Register

데이터 메모리 맵

모드 버스에서 번지 영역은 0x0000 ~ 0xFFFF(65535) 영역입니다.
본 통신형 입출력 장치는 데이터 량이 그리 많지 않고(예, 16비트 입출력의 경우 1개의 번지 사용) 사용하는 함수를 종류를 줄이기 위해 비트 크기 함수를 사용하지 않고 워드(16비트) 크기 함수(3, 4, 6, 16)을 사용하는데, 0번지에 데이터를 써넣으면 그대로 데이터가 출력 됩니다.
특정 비트만 세트,클리어,토글 동작을 위해서 아래와 같이 맵을 설정하였으며, 해당 번지에 비트를 1로 하면 해당 동작이 실행되고 0을 쓰면 아무 동작이 일어 나지 않음.
따라서 데이터를 그대로 출력하는 경우에는 0번지를 사용하고, 좀 더 편리한 비트 동작을 위해서 0x4000,0x8000,0xC000번지를 사용하시면 됩니다.

●메모리 맵

참조: 0x2000번지는 Broadcast 동작으로 Latch 된 입력 값 기억



이 영역에 데이터를 쓰면 그대로 출력 됨
예) 0x0000번지에 0xFFFF를 쓰면 0xFFFF 값이 그대로 출력 됨
0x0000 값을 쓰면 모든 비트가 0으로 출력 됨

이 영역에 데이터를 쓰면 비트 값이 1인 곳만 1로 SET 출력 됨
예) 0x4000번지에 0x0003를 쓰면 비트0과 1이 1로 출력
0x0000 값을 쓰면 변화 없음

이 영역에 데이터를 쓰면 비트 값이 1인 곳만 0으로 CLEAR 출력 됨
예) 0x8000번지에 0x8001를 쓰면 비트0과 15가 0로 출력
0x0000 값을 쓰면 변화 없음

이 영역에 데이터를 쓰면 비트 값이 토글 출력 됨
예) 0xC000 번지에 0x0003를 쓰면 비트0과 1가 반전 출력 됨
0x0000 값을 쓰면 변화 없음

특수 영역
예) 0xF000번지 값을 읽으면 장치 종류를 알 수 있음

장치 모드 별 입출력 개수

I/O 모드	디지털 입력	디지털 출력	아날로그 입력	아날로그 출력	PWM 출력
0 (RS_DIO16)	16	16			
1 (RS_DI32)		32			
2 (RS_DO32)	32				
3 (RS_AI8_DI24)	24		8		
4 (RS_AI8_PWM6)	10	8	8		6

장치 모드 별 메모리 맵

I/O 모드	0번지	1번지	2번지	3번지	4번지	5번지	6번지	7번지	8번지	9번지
0 (RS_DIO16)	DI16 DO16									
1 (RS_DI32)	DI32_L	DI32_H								
2 (RS_DO32)	DO32_L	DO32_H								
3 (RS_AI8_DI24)	DI24_L	DI24_H	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5	ADC6	ADC7
4 (RS_AI8_PWM6)	DI10 DO8	ADC0 PWM0	ADC1 PWM1	ADC2 PWM2	ADC3 PWM3	ADC4 PWM4	ADC5 PWM5	ADC6	ADC7	

메모리 맵 : 접속 장치 종류 및 버전 정보

현재 접속되어 있는 장치 종류 및 입출력 포트 상태를 알기 위해서 어드레스 0xf000 ~ 0xf005 내용을 읽어서 확인이 가능.

*장치 종류 및 버전(0xF000): 상위바이트(장치 종류), 하위바이트(버전 정보)

번지	기능	RS_DIO16 모드0	RS_DI32 모드1	RS_DO32 모드2	RS_AI8_DI24 모드3	RS_AI8_PWM6 모드4
0xF000	장치 종류 및 버전*	0x0001	0x0101	0x0201	0x0301	0x0401
0xF001	디지털 입력 수	16	32	0	24	10
0xF002	디지털 출력 수	16	0	32	0	8
0xF003	아날로그 입력 수	0	0	0	8	8
0xF004	아날로그 출력 수	0	0	0	0	
0xF005	PWM 출력 수	0	0	0	0	6