ThermoBlock Series
Communication Protocol
V1.0

목 차

1. 개 요

- 본 문서는 ㈜디웰전자 의 ThermoBlock series 의 통신 프로토콜 통합 문서 입니다.
- 본 문서의 프로토콜은 ThermoBlock 제품군에만 적용됩니다.
- ThermoBlock 은 디지털 통신 방식 3 가지를 지원합니다. (I2C, UART, RS485)
- 한 모델이 동시에 여러 통신방식을 지원하지 않습니다.
  - I2C 통신 모델은 I2C로만 통신이 가능합니다.
  - UART 통신 모델은 UART(3.3V TTL)로만 통신이 가능합니다.
  - RS-485 통신 모델은 RS-485로만 통신이 가능합니다.(MODBUS RTU 지원)
- 본 문서 전체를 볼 필요는 없습니다.
  목차를 참고하여 사용하는 모델의 프로토콜 방식 부분만 발췌독 하기 바랍니다.

2. 회로 연결시 주의사항

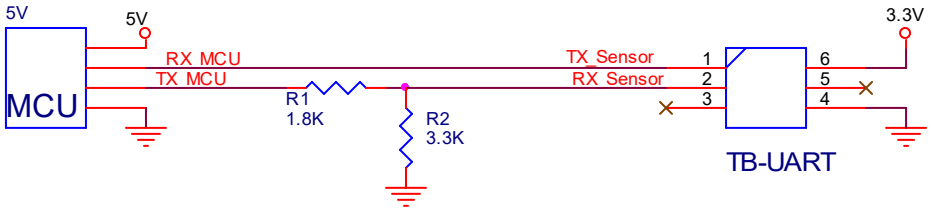
- UART
  - UART와 RS-232는 전압 레벨이 다르기 때문에 직결하면 안됩니다. 주의바랍니다.
  - 통신포트의 IO 레벨은 3.3V입니다. 5V tolerant 미지원.
  - 5V 로 동작하는 컨트롤러와 연결할 경우 레벨 시프터 또는 저항 분압 회로를 통해 전압을 맞춰 사용하십시오.

- I2C
  - 반드시 SCL, SDA 통신포트에 풀업 저항(Typ. 1.5㏀)을 연결해야 합니다.
  - 풀업 저항값은 통신 거리, 주파수, 전압에 따라 다르며, 1㏀~10㏀ 정도에서 선택하십시오.
  - 통신이 간헐적으로 실패한다면, 통신 거리와 풀업 저항 수치를 낮추기 바랍니다.
  - 통신 거리는 정해진 값은 없습니다만, 가급적 on-board로 적용하시길 바랍니다.
    (가급적 30cm는 넘지 않도록 하세요.)
  - 통신 포트는 반드시 오픈 드레인(open drain) 방식으로 사용해야 합니다. 관련 설정은 각 컨트롤러의 포트 설정을 참고하시기 바랍니다.(push-pull 방식 사용 금지)
  - 통신포트 레벨이 12V인 컨트롤러와는 추가 회로 없이 직접 연결이 불가능 합니다.

- RS-485
  - TB-485 의 포트(D+,D-)에는 직렬로 10옴의 저항만 연결돼 있으며, Bias 및 종단저항 그리고 보호 회로 등은 적용돼 있지 않습니다. 일반적인 연결 회로는 본 문서에 나와 있으나, 사용 환경/ 현장에 맞는 보호 회로 설계가 필요할 수 도 있습니다. 각 회사별 회로 설계 규정을 따르시기 바랍니다.

3. 연결 회로도

- UART
  - 5V 컨트롤러와 연결(저항 분압 회로)

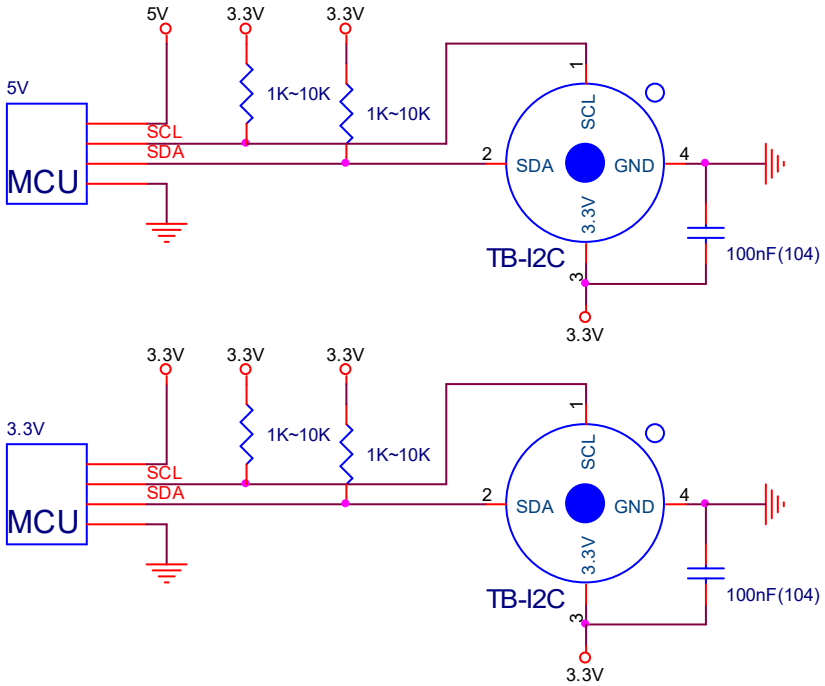


※ 5V MCU측의 RX 포트는 항상 입력으로 설정해야 하며, 출력 및 포트제어를 하면 절대 안됩니다. ThermoBlock 의 TX 통신 포트가 전압차로 인해 손상 될 수 있습니다.

  - 3.3V 컨트롤러와 연결

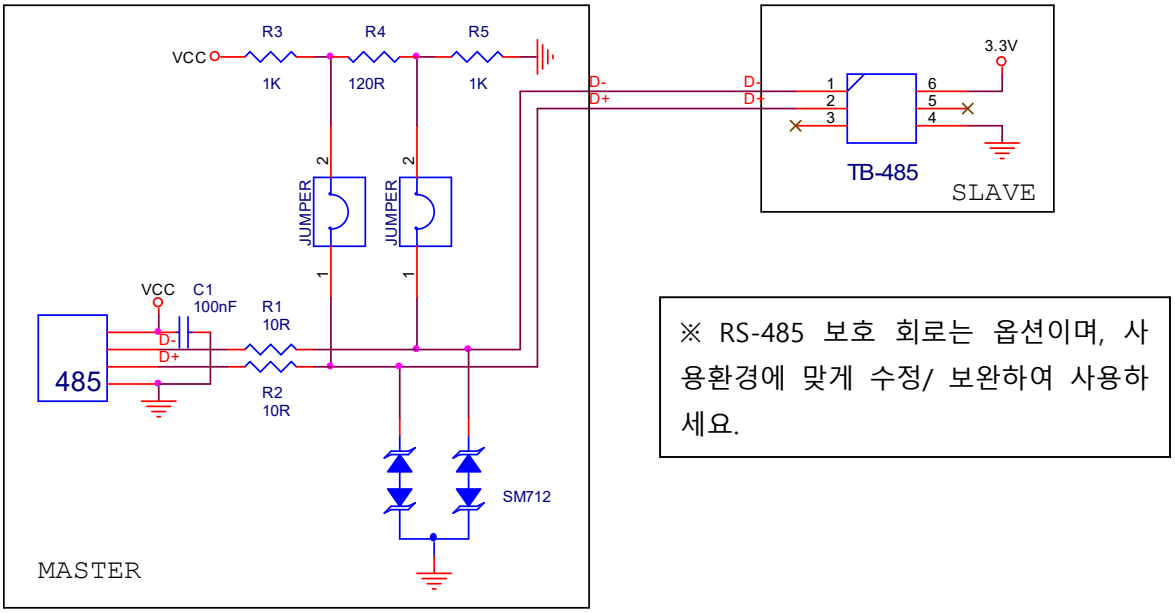
※ UART 는 1:1 통신만 가능합니다.

- I2C
  - 5V 및 3.3 V 컨트롤러와 1:1 연결



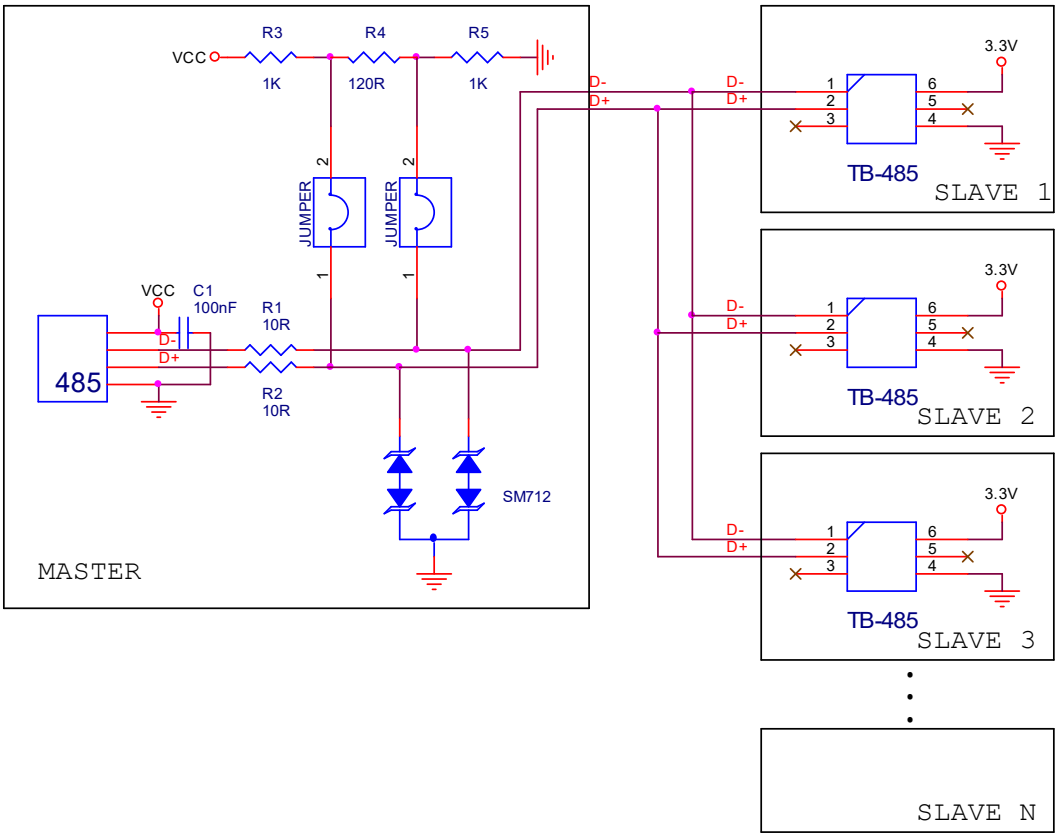
※ 캐패시터(100nF)는 센서와 최대한 가깝게 배치하십시오.

- RS-485
  - 1:1 연결



※ RS-485 보호 회로는 옵션이며, 사용환경에 맞게 수정/ 보완하여 사용하세요.

  - 1:N 연결



4. 통신 프로토콜

4.1. UART

4.1.1. 통신 규격

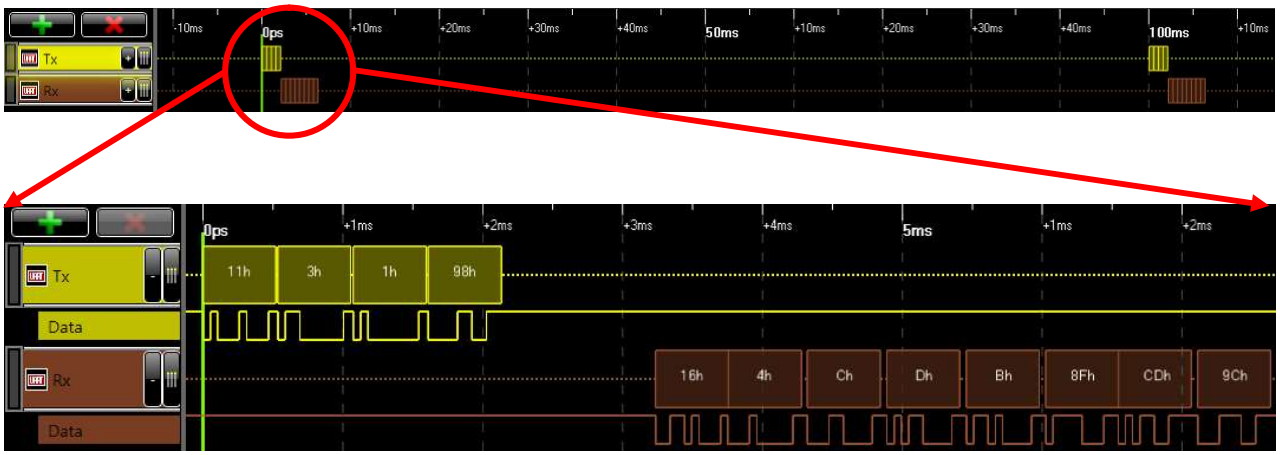
통신속도(bps)	DATA bit	Parity	Stop bit	Flow control
19200	8	none	1	no

4.1.2. 온도 데이터 읽기

※ Request 주기 : 반드시 최소 100ms 이상

※ 전원 공급 후 첫 Request 시간 : 최소 200 ms 이후

※ Request 후 첫 Byte 응답 시간 : 1~3ms 이후



Request (Master → TB-UART)			Response (TB-UART → Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
Request 1	0x11	17	START	0x16	22
Request 2	0x03	3	FUNCTION	0x04	4
Request 3	0x01	1	Data Value 1 Hi	대상온도	
Request 4	0x98	152	Data Value 1 Lo		
			Data Value 2 Hi	센서온도	
			Data Value 2 Lo		
			CS	Checksum	
			END	0x9C	156
4 Byte Request			8 Byte Response		

※ CS (checksum) 연산 방법.

CS = START + FUNCTION + Data Value 총 합에서 하위 1바이트만 사용합니다.

예) 0x16 + 0x04 + 0xF8 + 0x2B + 0x27 + 0x11 = 0x175

0x175 에서 하위 1바이트는 0x75 입니다. 따라서 CS 는 0x75가 됩니다.

4.1.3. TB-UART 온도 계산 방법(필독)

온도 데이터 변수의 Data type은 4byte signed 변수로 선언해야 합니다.

데이터는 2byte이지만 4byte를 사용하는 이유는 380.00℃까지 표현이 불가능하기 때문입니다. 380℃일 경우 데이터는 38,000 이 됩니다. 하지만 2byte로 선언을 하면 최상위 비트가 1이기 때문에 380도 가 아닌 영하의 온도가 됩니다. 문제는 실제 영하의 온도도 최상위 비트가 1이기 때문에 구분하기가 어렵게 됩니다. 따라서 이를 방지하기 위해 4byte signed 변수를 선언하는 것이며, 아래 계산식을 참고하시어 적용하시기 바랍니다.

만약 응답 데이터가 아래와 같은 경우.

응답한 데이터	HEX	DEC
Data Value 1 Hi	0xF82B	63,531
Data Value 1 Lo		
Data Value 2 Hi	0x2711	10,001
Data Value 2 Lo		

영상 온도/ 영하온도 판별은 데이터 값 40,000(dec)을 기준으로 합니다.

▶ Data Value 1 는 63,531 >40,000 이므로 최종 온도에서 65,536을 빼줘야 합니다.

63,531-65,536 = -2005. 결과값에 100을 나누면 최종 온도입니다.

-2005 / 100 = -20.05℃ (이 단계에서 float 형변환)

▶ Data Value 2 는 10,001 < 40,000 이므로 그대로 사용하면 됩니다.

10,001 / 100 = 100.01℃ (이 단계에서 float 형변환)

4.1.4. 방사율 읽기

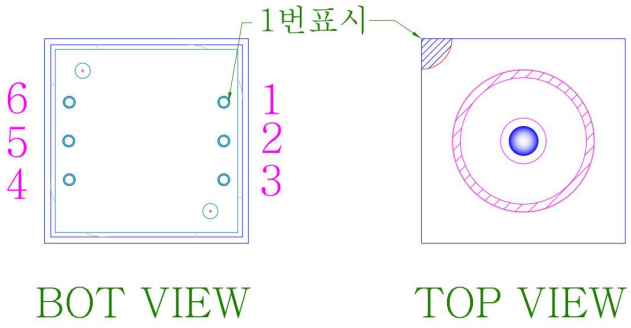
Request (Master → TB-UART)			Response (TB-UART → Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
Request 1	0x11	17	START	0x16	22
Request 2	0x03	3	FUNCTION	0x01	1
Request 3	0x05	5	Data Value	방사율	
Request 4	0x98	152	CS	Checksum	
			END	0x9C	156
4 Byte Request			5 Byte Response		

4.1.5. 방사율 쓰기

Request (Master → TB-UART)			Response (TB-UART → Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
Request 1	0x11	17	START	0x16	22
Request 2	0x06	6	FUNCTION	0x01	1
Request 3 (방사율)	0x0A ~ 0x64	10~100	Data Value	방사율	
Request 4	0x98	152	CS	Checksum	
			END	0x9C	156
4 Byte Request			5 Byte Response		

※ 방사율(default 0.97) 은 0.1~ 1.0 까지만 가능합니다.
쓰고자 하는 값에 100을 곱한 값을 전송하면 됩니다.
예) 0.95로 변경하고자 하는 경우 : 0.95 * 100 = 95
1.00 으로 변경하고자 하는 경우 : 1.00 * 100 = 100

4.1.6. Pin Description



Pin Number	통신 방식
	UART
1	RX
2	TX
3	-
4	GND
5	-
6	3.3V

※ 위 핀 번호는 UART 모델에만 적용됩니다.

4.1.7. 프로그래머 주의 사항.

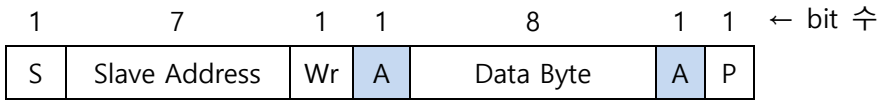
※ 방사율 변경 메시지는 메인 함수의 반복문에 넣지 마십시오.
방사율 변경 후에는 반드시 제품의 전원을 리셋하십시오.
※ 제품 전원 공급 후 최소 200ms 이후에 통신을 시작하십시오.

4.2. I2C

4.2.1. 통신 규격

※ I2C Slave Address : 0x3A (7bit), SCL frequency : 최대 100Khz

※ Symbol description

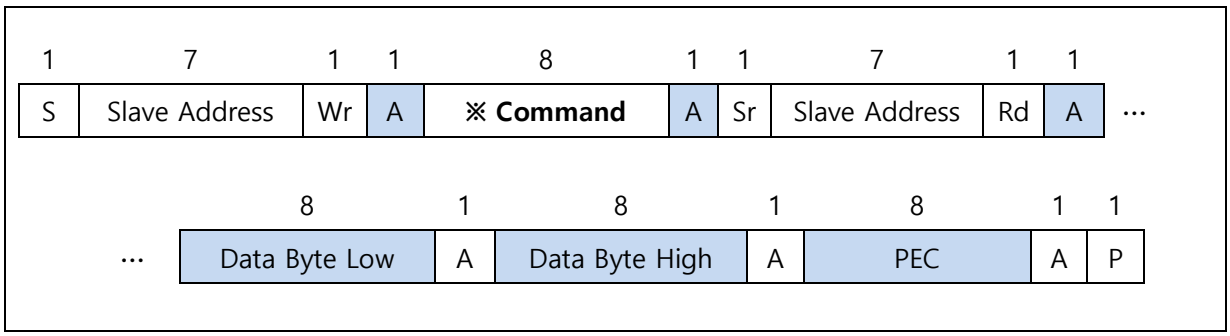


S	Start Condition
Sr	Repeated Start Condition
Rd	Read (bit value of 1)
Wr	Write (bit value of 0)
A	Acknowledge (this bit can be 0 for ACK and 1 for Nack)
S	Stop condition
PEC	Packet Error Code (※Note 1)
□	Master-to-Slave
■	Slave-to-Master

※ Note 1

PEC 는 통신 데이터에 오류가 없는지 확인하는 byte 입니다. (Data byte의 3번째 byte)
(CRC-8 with polynomial X8+X2+X1+1)
자세한 PEC 연산 과정은 아두이노 예제 코드를 확인하세요.(쇼핑몰 다운로드)

4.2.2. 데이터 Read 포맷

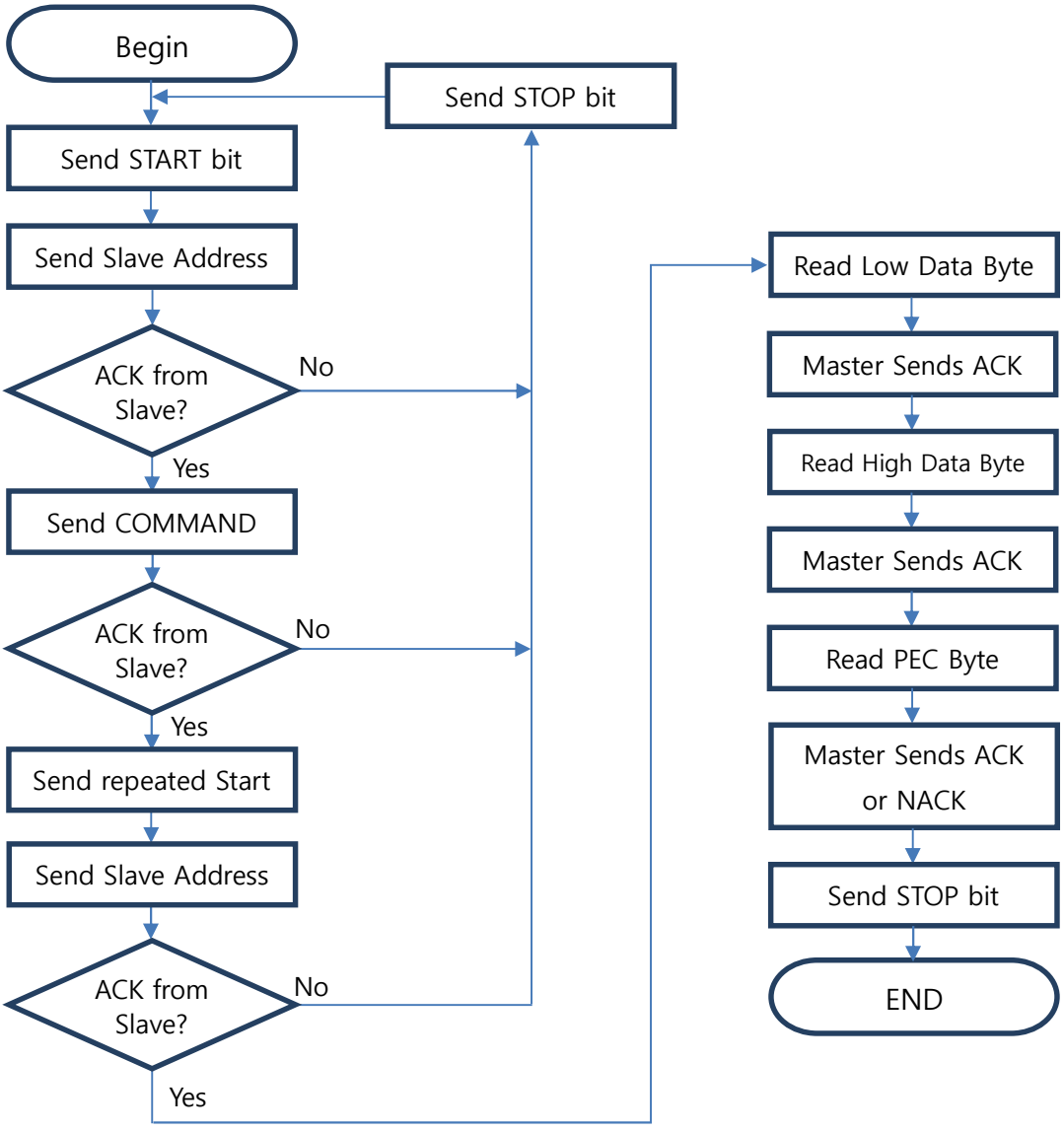


※ Command 정의

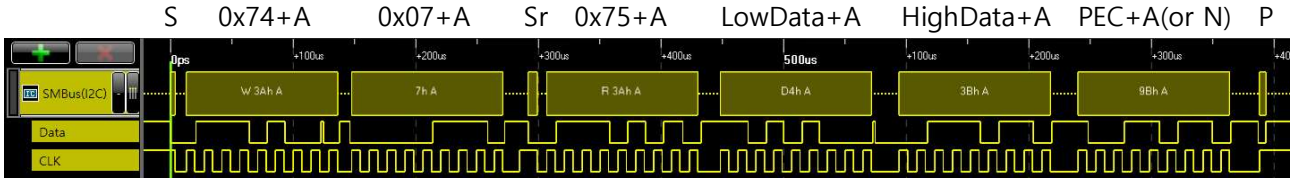
Name	**Command**	remarks
센서온도	0x06	-
대상온도	0x07	-
방사율	0x24	

※ 전원 공급/ 리셋 이후 첫 명령은 최소 200ms 이후에 시작하여야 합니다.

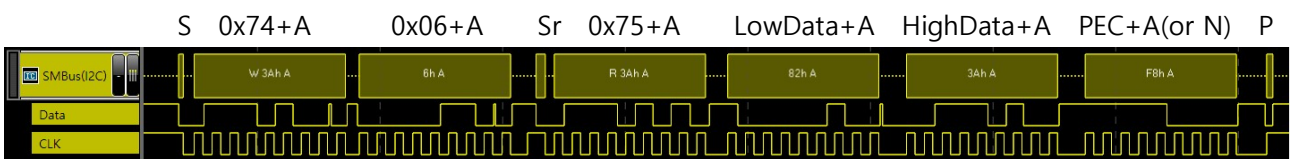
※ Read operation flow diagram:



※ 대상 온도 읽어오기 스코프 파형



※ 센서 온도 읽어오기 스코프 파형



4.2.2.1. I2C 온도 데이터 읽기

※ 온도 읽어오기 시퀀스

0. Power ON & 센서 초기화 시간(**200ms**)
1. Send Start bit
2. Send 0x74 : [Slave Address(0x3A by default) + Wr(0)]
3. ACK from Slave
4. Send Command : 0x07 (대상온도) [센서 온도일 경우 0x06]
5. ACK from Slave
6. Send repeated Start
7. Send 0x75 : [Slave Address(0x3A) + Rd(1)]
8. ACK from slave
9. Read Data Byte Low (Send ACK)
10. Read Data Byte High (Send ACK)
11. Read PEC (Send ACK or NACK)
12. Send Stop bit
13. Wait >**100ms** & repeat 1~13

4.2.2.2. I2C 온도 데이터 계산하기

Raw 데이터에 0.02 를 곱하면 절대온도(Kelvin)를 의미합니다.

※ 대상 온도 계산

대상 온도는 2Byte 의 크기를 갖습니다. 이중에서 온도 데이터는 0~14 bit 까지이며,
최상위 비트 15는 Error flag를 의미합니다. 0x7FFF(hex) = 0 111 1111 1111 1111 (2진)
Flag data

Error flag 가 1일 경우 : 데이터를 사용하지 마십시오. 0일 경우에만 온도계산을 수행하세요.

예제) Raw Data : 0x3B99 (15,257) 일 경우
절대온도(K) = 15,257 x 0.02 = 305.14 (K)
섭씨온도(℃) = 절대온도(K) - 273.15 = 305.14 - 273.15 = 31.99℃

※ 센서 온도 계산

센서 온도 역시 2Byte 의 크기이며 대상온도와 계산 과정이 동일합니다.

예제) Raw Data : 0x3365(13,157)
절대온도(K) = 13,157 x 0.02 = 263.14 (K)
섭씨온도(℃) = 절대온도(K) - 273.15 = 263.14 - 273.15 = -10.01℃

4.2.2.3. 방사율 읽기

※ 방사율 읽어오기 시퀀스

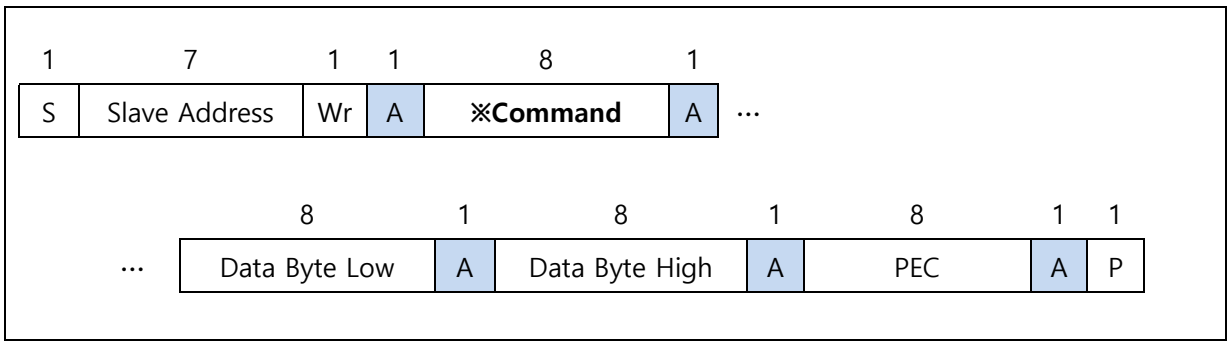
0. Power ON & 센서 초기화 시간(**200ms**)
1. Send Start bit
2. Send 0x74 : [Slave Address(0x3A by default) + Wr(0)]
3. ACK from Slave
4. Send Command : 0x24 (방사율)
5. ACK from Slave
6. Send repeated Start
7. Send 0x75 : [Slave Address(0x3A) + Rd(1)]
8. ACK from slave
9. Read Data Byte Low (Send ACK)
10. Read Data Byte High (Send ACK)
11. Read PEC (Send ACK or NACK)
12. Send Stop bit
13. End

4.2.2.4. 방사율 계산

읽어온 데이터가 0xF850 일 경우, 0xF850(hex) = 63,568(dec)

방사율 = (63,568 + 1) /65,536 = 0.9699... ≒ 0.97 (셋째 자리 반올림)

4.2.3. 데이터 Write 포맷



※ Command 정의

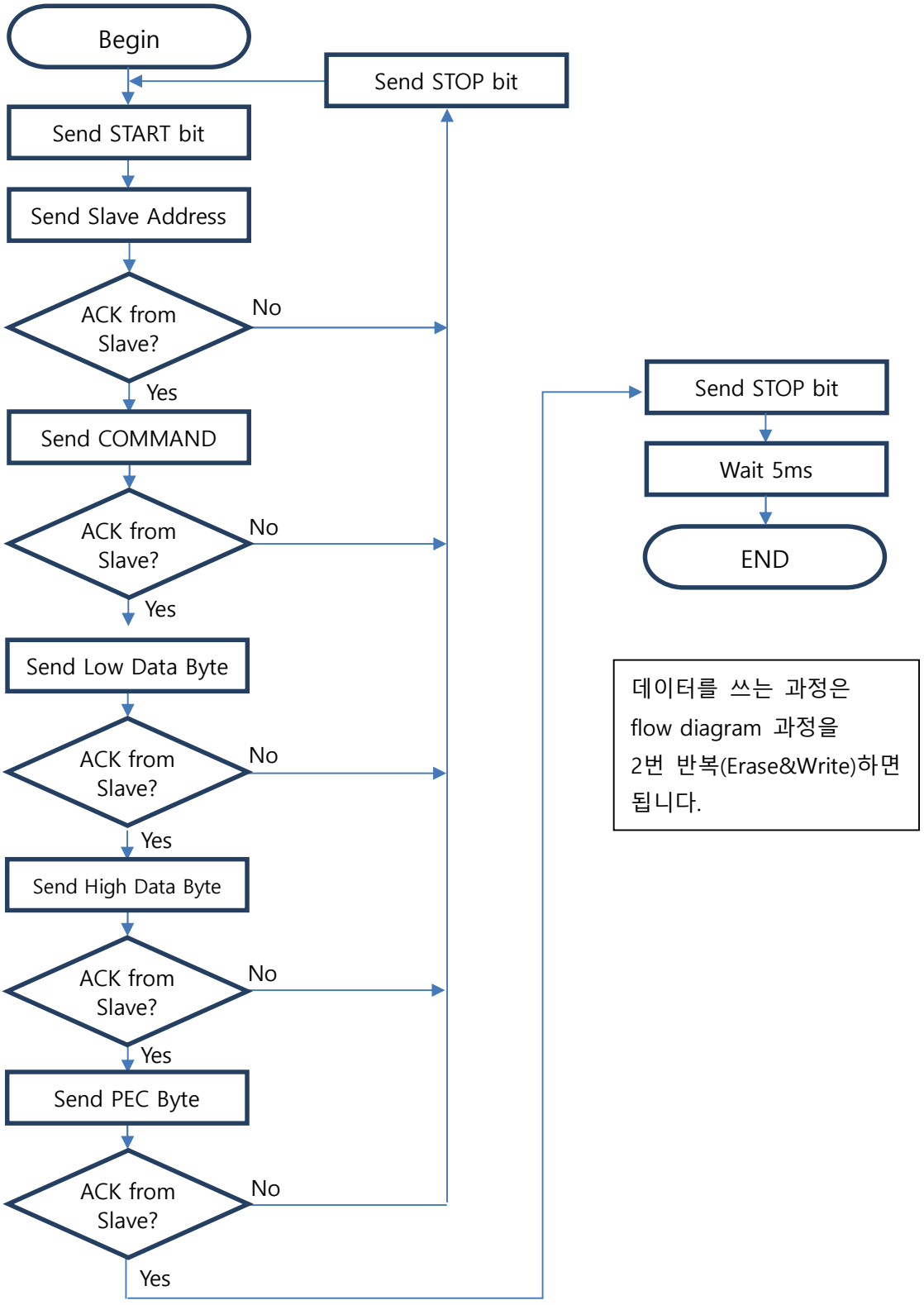
Name	**Command**	remarks
방사율	0x24	-
Address	0x2E	-

※ 전원 공급/ 리셋 이후 첫 명령은 최소 200ms 이후에 시작하여야 합니다.

※ 데이터 쓰기는 반복적으로 이루어 지면 안됩니다. 필요할 경우에만 한번 수행하세요.

※ 반드시 데이터를 변경한 후에는 전원 리셋을 해야 합니다.

※ Write operation flow diagram:



4.2.3.1. I2C Address 변경하기

.

※ Address 변경 시퀀스 <Byte 전송 후 ACK 문구는 생략합니다>

Erasing :

1. Send START bit
2. Send 0x74 : [Slave Address(0x3A by default) + Wr(0)]
3. Send Command : 0x2E (고정)
4. Send Data Byte Low : 0x00 (항상 0)
5. Send Data Byte High : 0x00 (항상 0)
6. Send PEC : 0x05 (데이터에 따라 값은 바뀝니다.)
7. Send STOP bit
8. Wait 5ms(중요) - EEPROM write 시간을 기다려야 합니다.

Writing : (Slave Address 0x4C로 변경)

9. Send START bit
10. Send 0x74 : [Slave Address(0x3A by default) + Wr(0)]
11. Send Command : 0x2E(고정)
12. Send Data Byte Low : 0x4C (변경할 주소)
13. Send Data Byte High : 0x00 (항상 0)
14. Send PEC : 0xA2 (데이터에 따라 값은 바뀝니다)
15. Send STOP bit
16. Wait 5ms(중요) - EEPROM write 시간을 기다려야 합니다.
17. 센서 전원 Reset(중요)

※ Address를 달리하여 최대 127개의 센서를 동시에 연결하여 통신이 가능합니다.
※ 센서의 default Address는 0x3A이며, 어드레스 변경시 센서는 마스터와 1:1로 연결해야 합니다.
※ 변경된 Address 의 관리에 주의하십시오. Address 를 잊은 경우 Global Address(0) 를 통해 확인이 가능합니다.(단, 센서와 마스터 1:1로 연결하셔야 합니다. 아두이노 예제코드 참고)
※ 변경할 Address 는 7bit를 넘지 않도록 하세요. 1~127. (0x01~0x7F)

4.2.3.2. I2C 방사율 변경하기

0.97로 변경하고자 하는 경우 아래의 계산 방식을 따릅니다.

$(0.97 \times 2^{16}) - 1 = 63568.92$ (소수점 이하는 버립니다.)

따라서 63568(dec) = 0xF850(hex) 이 변경할 데이터 입니다.

※ 방사율 변경 시퀀스 <Byte 전송 후 ACK 문구는 생략합니다>

※ 변경 후에는 반드시 전원을 리셋하여 센서를 다시 구동해야 합니다.

< Byte 전송 후 ACK 문구는 생략합니다. >

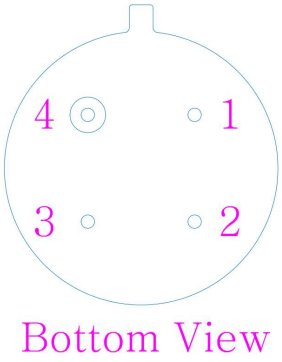
Erasing :

1. Send START bit
2. Send 0x74 : [Slave Address(0x3A by default) + Wr(0)]
3. Send Command : 0x24 (고정)
4. Send Data Byte Low : 0x00 (항상 0)
5. Send Data Byte High : 0x00 (항상 0)
6. Send PEC : 0x05 (데이터에 따라 값은 바뀝니다.)
7. Send STOP bit
8. Wait 5ms(중요) - EEPROM write 시간을 기다려야 합니다.

Writing : (Slave Address 0x4C로 변경)

9. Send START bit
10. Send 0x74 : [Slave Address(0x3A by default) + Wr(0)]
11. Send Command : 0x24(고정)
12. Send Data Byte Low : 0x50 (Low byte)
13. Send Data Byte High : 0xF8 (High byte)
14. Send PEC : 0x68 (데이터에 따라 값은 바뀝니다)
15. Send STOP bit
16. Wait 5ms(중요) - EEPROM write 시간을 기다려야 합니다.
17. 센서 전원 Reset(중요)

4.2.4. Pin Description



Bottom View

Pin Number	통신 방식
	I2C
1	SCL
2	SDA
3	3.3V
4	GND

4.2.5. TB-I2C 프로그래머 주의 사항.

※ 데이터 변경(Write) 메시지는 메인 함수의 반복문에 넣거나, 전원 공급시 실행하도록 코딩 하면 절대 안됩니다.

※ 데이터 변경 후 전원 리셋시, 마스터(MCU) 전원은 유지한 상태로 센서의 전원만 차단할 경우, 리셋이 안될 수 있습니다. SCL, SDA 포트가 풀업 저항을 통해 High level로 유지할 경우 그렇습니다. 전체 시스템 전원을 차단하면 센서 리셋이 이루어 집니다.

※ 제품 전원 공급 후 최소 200ms 이후에 통신을 시작하십시오.

※ 내부 온도 update 주기는 10Hz 입니다. 따라서 최소 100ms 주기로 온도를 읽으면 됩니다.

※ SCL frequency 는 Max. 100Khz 입니다.

※ 통신이 간헐적으로 잘 안 된다면, 다음 몇 가지 사항을 점검하십시오.

1. 풀업 저항이 너무 크거나 작은 것은 아닌지 체크.
2. 통신선 길이가 너무 길면 안됩니다.
3. 통신주파수가 50k~100Khz 를 벗어나는지 체크

4.3. RS-485

4.3.1. 통신 규격

※ Half-duplex, RS485 Multi-Drop

※ Modbus 485-RTU Protocol 지원

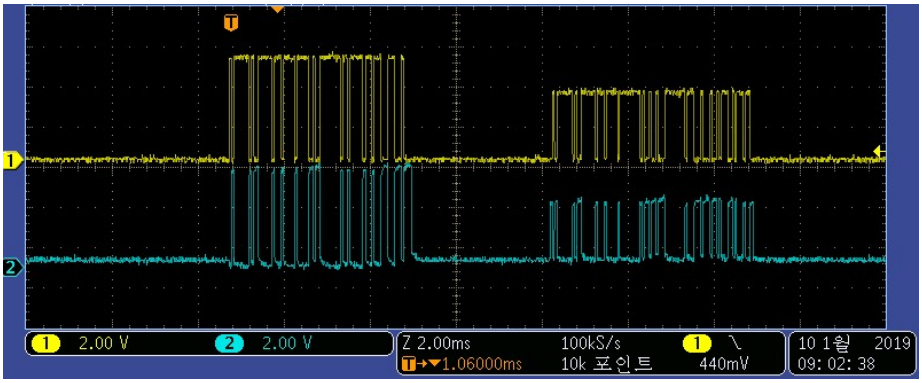
통신속도(bps)	DATA bit	Stop bit
19200	8	1

4.3.2. 온도 데이터 읽기

4.3.2.1. Modbus register(Modbus Function 03)

※ Request 주기 : 반드시 최소 100ms 이상

※ 전원 공급 후 첫 Request 시간 : 최소 200 ms 이후



※ 위 파형은 5V용 485 칩(Master) 과 TB-485(3.3V 485)간의 통신 파형입니다.
회로에 Bias 저항 및 종단 저항이 적용되지 않은 파형입니다.

Request (Master → TB-485)			Response (TB-485→ Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
ID(default 0x01)	0x01~0xc8	1~200	ID(default 0x01)	0x01~0xc8	1~200
Function	0x03	3	Function	0x03	3
Starting Address Hi	0x04	1200	Byte Count	0x04	4
Starting Address Lo	0xB0		Data Value 1 Hi	대상온도	
No. of Data Hi	0x00	2	Data Value 1 Lo		
No. of Data Lo	0x02		Data Value 2 Hi	센서온도	
CRC (19p~20p 참고)	데이터에 따라 바뀜.		Data Value 2 Lo		
CRC			CRC	데이터에 따라 바뀜.	
			CRC		
8 Byte Request			9 Byte Response		

※ No. of Data는 항상 2로 요청해야 합니다.

4.3.2.2. TB-485 온도 계산 방법(필독)

온도 데이터 변수의 Data type은 4byte signed 변수로 선언해야 합니다.
데이터는 2byte이지만 4byte를 사용하는 이유는 380.00℃까지 표현이 불가능하기 때문입니다. 380℃일 경우 데이터는 38,000 이 됩니다. 하지만 2byte로 선언을 하면 최상위 비트가 1이기 때문에 380도 가 아닌 영하의 온도가 됩니다. 문제는 실제 영하의 온도도 최상위 비트가 1이기 때문에 구분하기가 어렵게 됩니다. 따라서 이를 방지하기 위해 4byte signed 변수를 선언하는 것이며, 아래 계산식을 참고하시어 적용하시기 바랍니다.

만약 응답 데이터가 아래와 같은 경우.

응답한 데이터	HEX	DEC
Data Value 1 Hi	0xF82B	63,531
Data Value 1 Lo		
Data Value 2 Hi	0x2711	10,001
Data Value 2 Lo		

영상 온도/ 영하온도 판별은 데이터 값 40,000(dec)을 기준으로 합니다.

▶ Data Value 1 는 63,531 >40,000 이므로 최종 온도에서 65,536을 빼줘야 합니다.
63,531-65,536 = -2005. 결과값에 100을 나누면 최종 온도입니다.
-2005 / 100 = -20.05℃ (이 단계에서 float 형변환)

▶ Data Value 2 는 10,001 < 40,000 이므로 그대로 사용하면 됩니다.
10,001 / 100 = 100.01℃ (이 단계에서 float 형변환)

4.3.3. ID 변경하기 (반드시 마스터와 1:1 로 연결한 상태에서만 진행해야 합니다.)

4.3.2.1. Modbus register(Modbus Function 06)

Request (Master → TB-485)			Response (TB-485 → Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
ID	0xFF	255	ID	0xFF	255
Function	0x06	6	Function	0x06	6
Register Address Hi	0x03	1000	Register Address Hi	0x03	1000
Register Address Lo	0xE8		Register Address Lo	0xE8	
Register Value Hi	0x00	1~200	Register Value Hi	0x00	**1~200**
Register Value Lo	변경할 ID		Register Value Lo	**변경한 ID**	
CRC			CRC		
CRC					
8 Byte Request			7 Byte Response		

변경할 ID 는 1~200 까지 설정이 가능합니다. 범위를 초과하는 ID 를 설정하면, ID 는 1 로 초기화 됩니다. Default ID 는 1 이기 때문에 1:1 사용시에는 변경할 필요가 없습니다.

4.3.3. 방사율 읽기

4.3.2.1. Modbus register(Modbus Function 04)

Request (Master → TB-485)			Response (TB-485 → Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
ID	0x01~0xc8	1~200	ID	0x01~0xc8	1~200
Function	0x04	4	Function	0x04	4
Starting Address Hi	0x03	800	Byte Count	0x02	2
Starting Address Lo	0x20		방사율 Hi	0x00	0
Quantity of Input Reg. Hi	0x00	1	**방사율 Lo**	**0x0A~0x64**	**10~100**
Quantity of Input Reg. Lo	0x01		CRC		
CRC			CRC		
CRC					
8 Byte Request			7 Byte Response		

방사율은 default 0.97이며 100을 곱한 값이 전송됩니다.
만약, 방사율 Lo값이 97이면 0.97을 의미합니다.

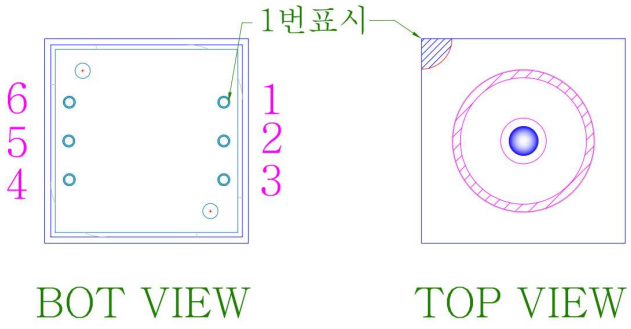
4.3.4. 방사율 변경

4.3.4.1. Modbus register(Modbus Function 06)

Request (Master → TB-485)			Response (TB-485 → Master)		
Field Name	Hex	DEC	Field Name	Hex	DEC
ID	0x01~0xc8	1~200	ID	0x01~0xc8	1~200
Function	0x06	6	Function	0x06	6
Register Address Hi	0x03	800	Register Address Hi	0x03	800
Register Address Lo	0x20		Register Address Lo	0x20	0
Register Value Hi	0x00	10~100	Register Value Hi	0x00	10~100
Register Value Lo	**방사율**		Register Value Lo	방사율	
CRC			CRC		
CRC			CRC		
8 Byte Request			8 Byte Response		

"방사율" 은 10~100 까지 변경이 가능합니다. (실제 방사율 0.1 ~ 1.0 에 100을 곱한 값)

4.3.5. Pin Description



Pin Number	통신 방식
	RS-485
1	D-
2	D+
3	-
4	GND
5	-
6	3.3V

※ 위 핀 번호는 TB-485 에만 적용됩니다.

4.3.6. 프로그래머 주의 사항.

※ 방사율 변경 메시지는 메인 함수의 반복문에 넣지 마십시오.
방사율 변경이 필요할 때 한번만 실행하면 됩니다. 전원을 리셋 해도 값은 유지됩니다.

※ 제품 전원 공급 후 최소 200ms 이후에 통신을 시작하십시오.

※ 485 통신chip은 메시지를 요청할 경우를 제외하고는 항상 입력 상태로 유지하십시오.

※ 모드 버스(Modbus)가 무언인가요?

생산 현장에서는 여러 장비들이 존재합니다. 모든 장비들을 한 회사에서 만들었다면 문제가 없겠지만 현실은 그렇지 않습니다. 장비도 다르고 환경도 다른 제 각각인 상황이 발생합니다. 따라서 일련의 약속을 통해 장비들을 제어 하는 통신 방법이 요구되었으며, 이런 통신 방법을 필드버스(Fieldbus) 라고 정의합니다. 필드버스에는 여러 기술들이 있습니다. 익히 알려진 CAN, Profibus, EtherCAT 등이 있으며, 본 TB-485 에 적용된 모드버스(Modbus) 또한 표준으로 정의돼 있습니다.

RS-485 통신을 사용하며, 마스터/ 슬레이브로 구분되어 연결된 모든 장비의 정보를 모니터링 할 수 있습니다.(환경에 따라 동시 연결 장비 수는 제한됩니다) 마스터는 한대만 가능하며, 질의 응답 구조로 해당하는 장비만 응답을 하게 됩니다. 또한 오픈 프로토콜이기 때문에 무료로 사용이 가능합니다.

(아래 CRC 는 RS-485 모드버스 프로토콜에만 적용됩니다.)

※ CRC란?

CRC(Cyclic Redundancy Check) 는 데이터 통신시 전송 데이터에 오류가 있는지 확인 하기 위해 체크값을 결정하는 방식입니다. 모든 Request/ Response 메시지에 2byte 크기의 CRC 값이 붙어서 전송됩니다. 전송된 데이터 값으로 연산한 CRC 값과 수신된 CRC 값을 비교하여 오류 여부를 검증합니다. Polynomial 값은 0xA001 이지만, 아래 예제 코드는 비트단위 연산 부하를 줄이기 위해 CRC 테이블을 활용하는 예제 입니다. 요즘 시스템들은 메모리가 작지 않기 때문에 테이블 적용을 추천합니다.

※ CRC 계산 예제 코드.

```
const uint16_t TableCRC16[256]={
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241, 0xC601, 0x06C0,
    0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440, 0xCC01, 0x0CC0, 0x0D80, 0xCD41,
    0x0F00, 0xCFC1, 0xCE81, 0x0E40, 0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0,
    0x0880, 0xC841, 0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41, 0x1400, 0xD4C1,
    0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641, 0xD201, 0x12C0, 0x1380, 0xD341,
    0x1100, 0xD1C1, 0xD081, 0x1040, 0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1,
    0xF281, 0x3240, 0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41, 0xFA01, 0x3AC0,
    0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840, 0x2800, 0xE8C1, 0xE981, 0x2940,
    0xEB01, 0x2BC0, 0x2A80, 0xEA41, 0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1,
    0xEC81, 0x2C40, 0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041, 0xA001, 0x60C0,
    0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240, 0x6600, 0xA6C1, 0xA781, 0x6740,
    0xA501, 0x65C0, 0x6480, 0xA441, 0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0,
    0x6E80, 0xAE41, 0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41, 0xBE01, 0x7EC0,
    0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40, 0xB401, 0x74C0, 0x7580, 0xB541,
    0x7700, 0xB7C1, 0xB681, 0x7640, 0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0,
    0x7080, 0xB041, 0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440, 0x9C01, 0x5CC0,
    0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40, 0x5A00, 0x9AC1, 0x9B81, 0x5B40,
    0x9901, 0x59C0, 0x5880, 0x9841, 0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1,
    0x8A81, 0x4A40, 0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641, 0x8201, 0x42C0,
    0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040
};

uint16_t CalcCRC16(unsigned char *pdata, uint16_t DataLen)
{
    uint16_t AccumCRC16 = 0xFFFF;
    unsigned char i,j;
    for(j=0; j<DataLen; j++)
    {
        i = (AccumCRC16 ^ *(pdata++) ) & 0xFF;
        AccumCRC16 = ((AccumCRC16>>8) ^ TableCRC16[i]) & 0xFFFF;
    }
    return AccumCRC16;
}
```

※ 온도 읽는 명령 CRC 구하는 프로세스.(TB-485 제품)

온도 읽는 명령의 CRC 값을 구해보도록 하겠습니다.

먼저 데이터를 전송할 적당한 크기의 전역 변수를 선언합니다.

```
uint8_t RequestData[8];          // 전송할 배열
uint16_t crc;                    // 연산한 crc값 임시 저장 변수
```

"4.3.2 온도데이터 읽기"를 참고하여 메인 함수에서 배열 요소에 값을 할당합니다.

```
RequestData[0] = 0x01;
RequestData[1] = 0x03;
RequestData[2] = 0x04;
RequestData[3] = 0xB0;
RequestData[4] = 0x00;
RequestData[5] = 0x02;
```

여기까지가 보낼 데이터이며, 여기에 CRC 2byte를 계산하여 추가해야 합니다.
앞페이지에서의 CalcCRC16 함수를 사용합니다.

```
crc = CalcCRC16(RequestData, 6);    // RequestData 배열 0~5 까지, 6 바이트 데이터를 연산합니다.

RequestData[6] = (unsigned char)((crc >> 0) &0x00FF);    // CRC 값의 하위 바이트
RequestData[7] = (unsigned char)((crc >> 8) &0x00FF);    // CRC 값의 상위 바이트
```

연산을 거친 crc 값은 아래와 같습니다.
RequestData[6] 변수에는 0xC4 값이 저장됩니다.
RequestData[7] 변수에는 0xDC 값이 저장됩니다.

CRC를 포함한 전송할 데이터 준비는 끝났으며, RequestData 배열 0~7 의 값을 전송하면 됩니다.
수신 측에서는 (RequestData[6], RequestData[7] 의 데이터) 와 (수신받은 데이터의 crc 연산값)을 비교한 후, 일치하면 응답, 일치하지 않으면 응답을 하지 않습니다.

반대로 마스터는 TB-485로부터 수신된 데이터를 위와 같은 연산을 거쳐, 데이터에 오류가 없는지 검증한 후 온도 데이터를 활용하면 됩니다.

※ TB-485 온도 요청 데이터(CRC 포함) 예제

ID1번 : 0x01, 0x03, 0x04, 0xB0, 0x00, 0x02, (0xC4), (0xDC)
ID2번 : 0x02, 0x03, 0x04, 0xB0, 0x00, 0x02, (0xC4), (0xEF)
ID3번 : 0x03, 0x03, 0x04, 0xB0, 0x00, 0x02, (0xC5), (0x3E)

Additional Information

- manufacturer : Diwell Electronics Co., Ltd. <(주)디웰전자>
- Homepage : www.diwell.com
- shopping mall : www.diwellshop.com
- Phone : +82-70-8235-0820
- Fax : +82-31-429-0821
- Technical support : expoeb2@diwell.com, dsjeong@diwell.com

Revision History

Version	Date	Description
1.0	2019-05-13	First version is released.