MPINO STUDIO MANUAL

2022. 04. 07

- **사용 전에 안전을 위한 주의사항의 반드시 읽고 정확하게 사용하여 주십시오.**
- **사용설명서를 읽고 난 뒤에는 제품을 사용하는 사람이 항상 볼 수 있는 곳에 잘 보관하십시오.**

▣ 알림

본 사용설명서는 (주)아이로직스의 산업 범용 컨트롤러인 MPINO 제품군을 프로그램할 수 있는 소프트웨어입니다. 아두이노의 C언어와 PLC(Programable Logic Controller)의 Ladder언어를 동시에 사용할 수 있습니다.

개발자의 도움이 되고자 블로그를 운영하고 있습니다.
링크: https://blog.naver.com/ilogics

▣ 주의사항

본 사용설명서는 내용이 변경될 수 있으니, 항상 최신 버전을 확인해 주시기 바랍니다.

폐사의 제품을 사용하다 발생한 손해 및 손실에 대하여 폐사는 민,형사적 법적 책임이 없음을 명시합니다. 폐사의 제품을 사용하기 전에 컨트롤러를 적용할 제품과 단동 및 연동 테스트 진행하고, 인명 및 재산피해가 발생할 수 있는 시스템에는 2차 안전장치를 설치하여 사용해주시기 바랍니다.

▣ INDEX

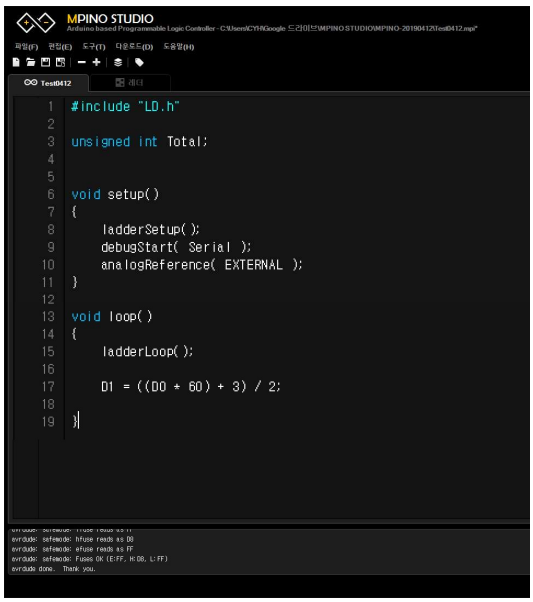
1. MPINO STUDIO

1.1. 개요

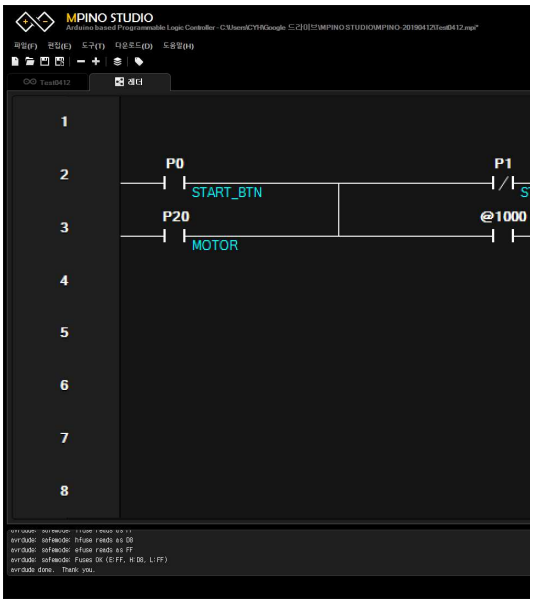
MPINO STUDIO는 MEGAPLC의 MEGA와 ARDUINO의 INO를 합성하여 작명되었습니다.

ARDUINO IDE에서 사용하였던 C언어를 호환하면서 PLC의 LADDER LOGIC을 사용하실 수 있습니다.

복잡한 산술 연산은 ARDUINO의 C언어를 사용하고, 단순한 I/O제어는 LADDER LOGIC을 사용하여 프로그램할 수 있는 장점을 가지고 있습니다.



〈아두이노 C언어〉



〈PLC LADDER LOGIC〉

1.2. MPINO STUDIO 설치

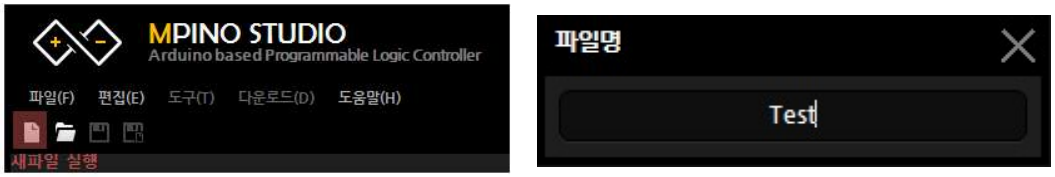
MPINO STUDIO는 (주)아이로직스 자료실에서 다운로드 받으실 수 있습니다.
쇼핑몰 : www.ilogics.co.kr
자료실 : http://www.ilogics.co.kr/page/07_view.php?idx=365&startPage=

MPINO STUDIO를 설치하고, 처음 실행할 때 플랫폼을 설치하므로, 다소 시간이 걸립니다. 이점 참고해 주세요.

1.3. MPINO STUDIO 시작

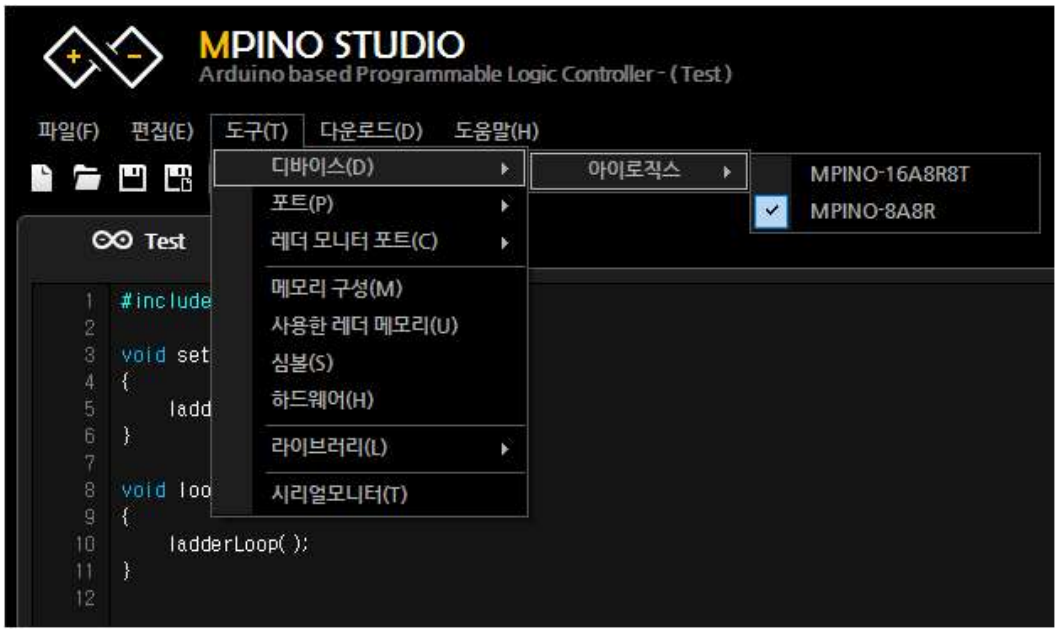
① 새파일 생성

새파일 아이콘을 실행하고, 프로젝트 명 등으로 파일명을 기입합니다.



② 디바이스 선택

도구 -> 디바이스 -> 아이로직스 -> MPINO 제품군을 선택합니다.



③ 포트 선택

도구 -> 포트 -> 포트번호를 선택합니다.

포트는 MPINO제품군의 다운로드 포트와 컴퓨터를 USB 케이블로 연결하면, 윈도우의 장치관리자의 “포트(COM&LPT)”에서 CP210x의 포트번호입니다.

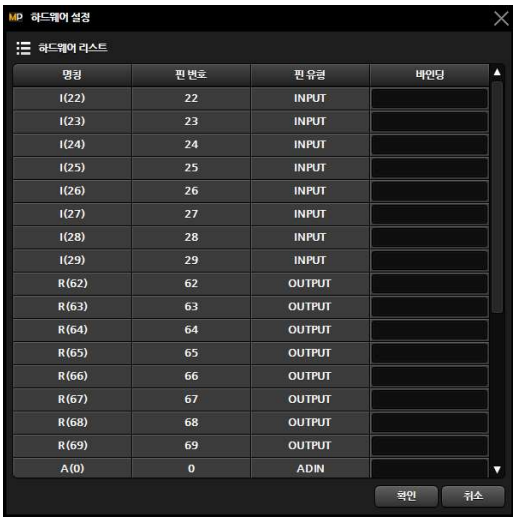
④ 하드웨어

LADDER LOGIC을 실행하기 위하여, 핀번호에 대한 포트번호를 바인딩 해야 합니다. 아두이노 C언어만을 사용하고자 할 경우에는 바인딩 하지 말아 주시기 바랍니다. 또한, 사용하지 않는 기능에 대해서는 바인딩 하지 않아도 됩니다.

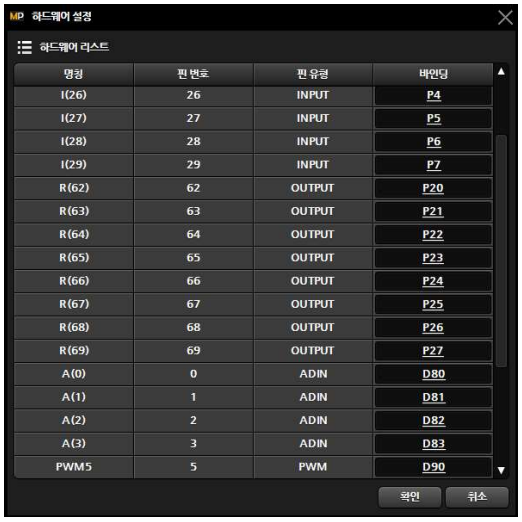
도구 -> 하드웨어를 실행하여, MPINO 제품군 터미널블럭에 새겨져 있는 명칭을 Px 또는 Dx로 LADDER LOGIC에서 사용하고자 하는 메모리로 바인딩 합니다.

디지털 입력, 트렌지스터 출력, 릴레이 출력은 Px로 바인딩 합니다. (P0, P1..)
아날로그 입력, 아날로그 출력, PWM, 고속카운터는 Dx로 바인딩 합니다. (D0, D1...)

Px는 비트 Dx는 워드단위로서, LADDER LOGIC에서 정해져 있는 메모리 이름입니다. 관련 내용은 LADDER LOGIC에서 자세히 확인할 수 있습니다.



[바인딩 이전]



[바인딩 이후]

⑤ 프로그램 작성 & 다운로드

프로그램을 작성한 이후, Ctrl + D 또는 상단 탭에서 다운로드 -> 다운로드를 실행하거나, 프로그램 작성창에서 마우스 우클릭 -> 다운로드를 실행하여 프로그램을 MPINO 제품으로 다운로드합니다.

프로그램 작성 등의 자세한 사항은 각 파트별 설명글에서 확인하실 수 있습니다.

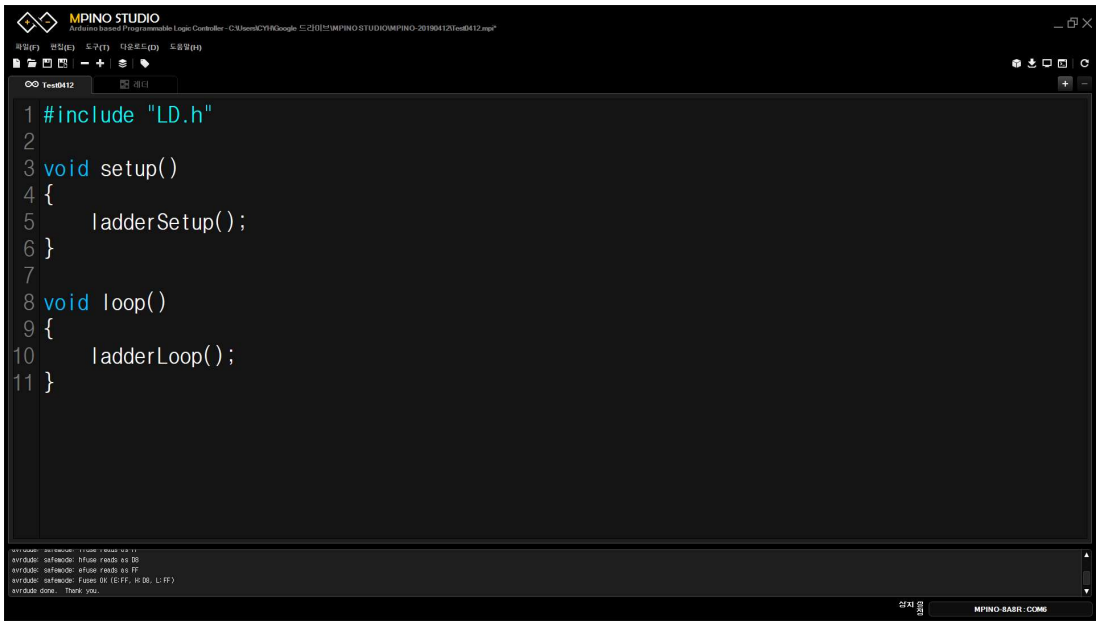
2. 아두이노 C언어

2.1. 기본 구조

아두이노의 C언어는 아래와 같이 void setup(){} 과 void loop() {}의 구조로 동작합니다. setup()함수와 loop()함수는 아래와 같은 기능을 수행합니다.

setup() : 프로그램 실행시 최초에 1회만 실행합니다.

loop() : setup()함수를 실행한 이후, 계속 실행합니다.



< 아두이노 C언어 작성창 >

MPINO STUDIO는 새파일로 생성시 위 그림과 같이 ladderSetup() 과 ladderLoop() 코드가 자동생성되고, 각각의 명령어는 아래와 같은 기능을 수행합니다.

- ladderSetup() : LADDER LOGIC의 I/O 설정등의 기본기능을 수행합니다.
- ladderLoop() : LADDER LOGIC의 프로그램을 호출하여 실행합니다.

LadderSetup()과 LadderLoop()를 실행하지 않으면, LADDER LOGIC이 동작하지 않으므로, 특별한 경우 이외에는 지우지 말고 사용하시기 바랍니다.

또한, debugStart(Serial)를 사용하여 LADDER LOGIC 모니터링 시스템을 사용하실 수 있습니다. 매개변수 Serial(0번채널), Serial1(1번채널), Serial2(2번채널), Serial3(3번채널)을 의미합니다.

- debugStart(Serial) : 다운로드 포트로 LADDER LOGIC 모니터링을 사용합니다.

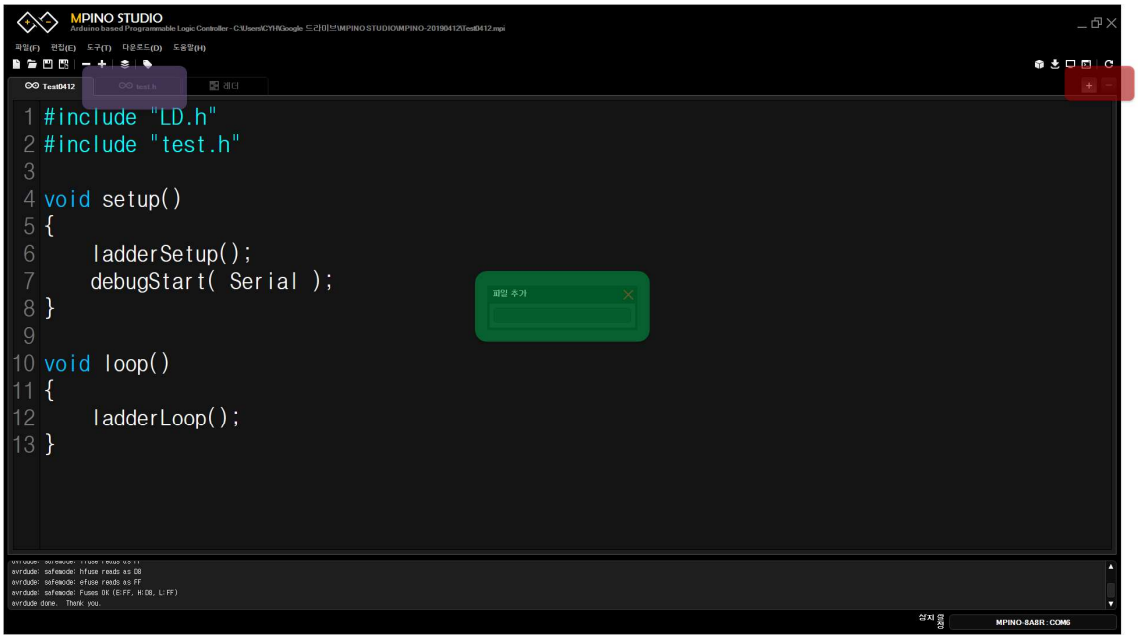
2.2. #include

아두이노는 다른 C컴파일러와 같이 "#include"를 사용하실 수 있습니다. “#include"를 사용하여 미리 정의하거나 정리해 놓은 함수들을 참조시킬 수 있습니다.

- #include "LD.h"

MPINO STUDIO의 LADDER LOGIC 함수들을 참조시키고 있습니다. MPINO STUDIO에서 새파일을 만들면 자동으로 생성되는 참조파일입니다. LADDER LOGIC의 명령어들이 정의되어 있습니다.

- include 추가 방법



적색부분의 “+”를 클릭하면, 초록색부분이 팝업됩니다. “이름”을 입력 후 엔터를 수행하면, 해당 이름으로 된 코딩탭이 보라색부분과 같이 생성됩니다. 이후, setup()문 위에 #include “이름”을 작성하고. 해당 탭에서 함수를 코딩할 수 있습니다.

적색부분의 “-”는 추가된 코딩탭을 삭제하는 버튼입니다. 삭제하고 싶은 코딩탭으로 이동한 후, “-”버튼을 클릭하여 삭제하실 수 있습니다.

2.3. 입/출력 모드 설정

MPINO STUDIO에서 디바이스를 선택함으로서, 자동으로 입/출력 설정이 됩니다. 때문에, 별도의 입/출력 설정이 필요 없지만 아래의 명령어로 변경이 가능합니다.

- **void pinMode(PIN, "INPUT"/"OUTPUT")**

디지털 입력 또는 출력모드로 설정하는 함수입니다.

pinMode(0, INPUT); 또는 pinMode(0, 0); 으로 0번핀을 입력모드로 설정
pimMode(62, OUTPUT); 또는 pinMode(62, 1); 으로 62번핀을 출력모드로 설정

2.4. 디지털 입력

- **bool digitalRead(PIN)**

디지털 입력포트의 상태를 읽어오는 함수입니다.

digitalRead(핀 번호)의 핀 번호는 MPINO 제품군의 디지털 입력포트 번호를 사용하시면 됩니다. 가령, 디지털 입력포트가 I(22)일 경우, 22를 사용합니다.

if (digitalRead(22) == 1) { 실행문; } I(22)에 입력이 On되면, 실행문을 실행.

2.5. 디지털 출력

- **void digitalWrite(PIN, 0/1)**

디지털 출력포트를 On 또는 Off 시키는 함수입니다.

digitalWrite(핀 번호, 상태)의 핀 번호는 MPINO 제품군의 디지털 출력포트 번호를 사용하시면 됩니다. 가령 릴레이 출력포트가 R(62)일 경우, 62를 사용합니다.

아래의 프로그램은 디지털입력 22번 포트의 상태가 On이면, 디지털출력 62번 포트를 On시키고, 22번 포트의 상태가 Off이면, 62번 포트를 Off 시킵니다.

```
#include "LD.h"

void setup()
{
        ladderSetup();
        debugStart( Serial );
}

void loop()
{
        if ( digitalRead( 22 ) == 1 ) digitalWrite( 62, 1 );
        else digitalWrite( 62, 0 );
    ladderLoop();
}
```

2.6. 아날로그 입력

• **unsigned int analogRead(Analog Pin)**
아날로그 입력포트로 입력된 전기신호를 디지털 수치로 읽어오는 명령입니다.

analogRead(0); 는 아날로그 입력포트 A(0)에 입력된 아날로그 전기신호를 0~1023의 디지털 수치로 읽어옵니다.

아래는 아날로그 입력포트 A(1)에 인가된 전기신호를 ADC1 변수에 저장하는 코딩입니다.

2.7. 아날로그 출력

• **void analogWrite(AO PIN, Value)**
아날로그 출력포트(AO PIN)에 0~65535 Range에서 Value의 값만큼 출력합니다.

analogWrite(6, 0); 는 아날로그 출력포트 AO(6)에 아날로그 출력 Range의 최소값을 출력합니다. (Ex. 0~5V일 경우, 0V)
analogWrite(7, 32767); 는 아날로그 출력포트 AO(7)에 아날로그 출력 Range의 절반 값을 출력합니다. (Ex. 0~5V일 경우, 2.5V)
analogWrite(6, 65535); 는 아날로그 출력포트 AO(6)에 아날로그 출력 Range의 최대 값을 출력합니다. (Ex. 0~5V일 경우, 5V)

MPINO STUDO의 하드웨어 설정에서 AO(Analog Output) 포트에 워드 메모리를 바인딩하면, 해당 메모리의 값을 참조하여 LADDER LOOP() 함수에서 자동으로 analogWrite() 함수를 실행하므로, analogWrite() 함수를 C언어 창에서 사용할 수 없습니다.
때문에, 바인딩한 워드 메모리를 아래와 같이 C언어 창에서 사용해야 합니다.

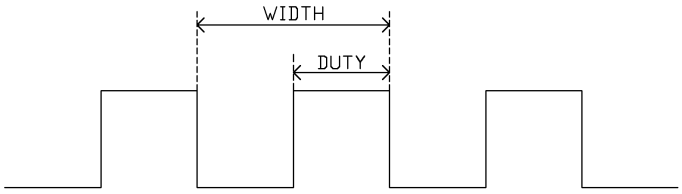
• D80 = 32767;

2.8. 고속펄스(PWM) 출력

• **void analogWrite(PWM PIN, Duty)**
PWM 펄스 출력을 실행하는 함수입니다. Duty는 65535(Width)를 초과할 수 없습니다.

analogWrite(5, 0); 는 PWM 출력포트 PWM(5)에 펄스를 출력을 하지 않습니다.
analogWrite(2, 32767); 는 PWM 출력포트 PWM(2)에 Duty가 32767이고 Width가 65535인 펄스를 계속적으로 출력합니다.

analogWrite(3, 65535); 는 PWM 출력포트 PWM(3)에 펄스를 출력 하지 않고, 5V를 출력합니다.



[PWM의 WIDTH와 DUTY]

MPINO STUDIO의 하드웨어 설정에서 워드 메모리를 바인딩하면, 해당 메모리의 값을 참조하여 LADDER LOOP()에서 자동으로 analogWrite()를 실행하므로, analogWrite()함수를 C언어창에서 사용할 수 없습니다.

때문에, 바인딩한 워드 메모리를 아래와 같이 C언어 창에서 사용해야 합니다.

- D80 = 32767;

2.9. 고속펄스 카운터

고속펄스 카운터는 고속으로 입력되는 펄스를 하드웨어적으로 카운트합니다.

LOOP() 문에서 고속으로 입력되는 펄스를 카운트하기 위해서는 LOOP()문의 스캔타임(모든 프로그램이 한번 동작하는 시간)의 2배가 필요합니다. 때문에, 하드웨어에서 별도로 카운트되는 아래의 함수를 사용해야 합니다.

- **unsigned long hcntRead(TCNT Ch)**

TCNT ch에 카운트된 펄스 개수를 읽어옵니다.

TCNT Ch은 MPINO 제품 고속카운터 포트에 적혀 있는 번호를 사용할 수 있습니다. TCNT4의 경우, 4를 사용하셔야 합니다.

- **void hcntReset(TCNT Ch)**

TCNT Ch의 카운트 값을 0으로 초기화 합니다.

```
#include "LD.h"

unsigned long HCNT0;
unsigned long HCNT1;

void setup()
{
    ladderSetup();
}

void loop()
{
        ladderLoop();
        HCNT0 = hcntRead( 1 );
        HCNT1 = hcntRead( 5 );
        if ( digitalRead( 22 ) == 1 ) hcntReset( 1 );
        if ( digitalRead( 23 ) == 1 ) hcntReset( 5 );
}
```

3. LADDER LOGIC

LADDER LOGIC은 PLC(Programable Logic Controller)에서 사용하는 프로그래밍 방식이며, 전기회로와 유사한 그래픽형식을 띄고 있습니다. 그래픽형식의 언어로서 직관적으로, 배우기 쉽고 프로그램의 로직이 눈에 쉽게 들어오고 모니터링 시스템으로 편리하게 디버깅을 할 수 있는 장점을 가지고 있는 언어방식입니다.

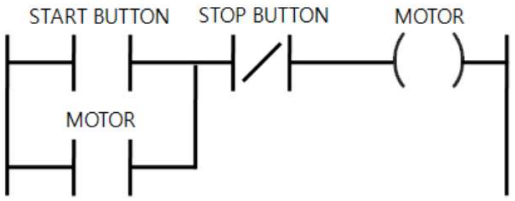
LADDER LOGIC은 아래와 같은 기호로 구성됩니다.

3.1. LADDER LOGIC 기호

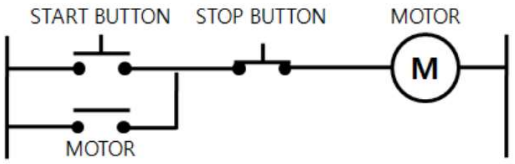
아이콘	이름	설 명	단축키
┤ ├	N.O 접점, A접점 (Normal Open)	평상시에 입력조건을 “LOW" 접점신호가 ON되면, 입력조건을 “HIGH"	F3
┤/├	N.C 접점, B접점 (Normal Close)	평상시에 입력조건을 “HIGH" 접점신호가 ON되면, 입력조건을 “LOW"	F4
——	가로 연결선	가로 방향으로 접점을 연결해주는 선	F5
|	세로 연결선	세로 방향으로 접점을 연결해주는 선	F6
-( )-	COIL	입력조건이 "HIGH“이면, “HIGH"를 출력 입력조건이 “LOW”이면, "LOW"를 출력	F7
-[ ]	FUNCTION	입력조건이 맞으면, 펑션을 실행	F8
-/-	NOT 접점	조건을 반전시키는 접점	F9
┌┘	Positive Edge 접점	입력조건이 “LOW"에서 ”HIGH"로 변할 때, 한 스캔동안만 연결되고 끊기는 접점	F11
└┐	Negative Edge 접점	입력조건이 “HIGH"에서 ”LOW"로 변할 때, 한 스캔동안만 연결되고 끊기는 접점	F12

3.2. LADDER LOGIC이란?

LADDER LOGIC은 LD 라고 약어로 쓰이기도 하는 PLC의 대표적인 프로그래밍 언어입니다. LD는 전기회로와 아주 유사한 형태를 띄고 있으며, 그래픽 형식의 언어로서 직관성이 좋고 모니터링 시스템을 통해 I/O의 디버깅이 용이하기 때문에 산업의 기계 및 공장자동화에서는 필수적으로 사용하고 있습니다.

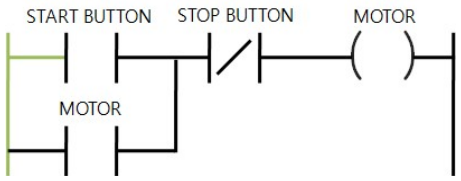
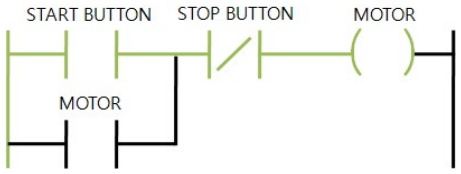
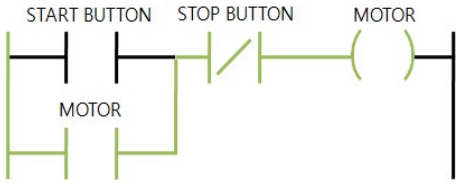
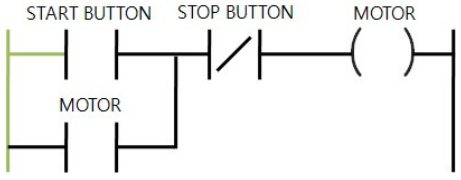


[LADDER LOGIC]



[전기 회로]

위 LD를 해석하면 아래와 같습니다. 아래의 설명글은 래더언어의 기본인 접점과 코일을 설명하면서 “자기유지회로”를 설명하고 있습니다.

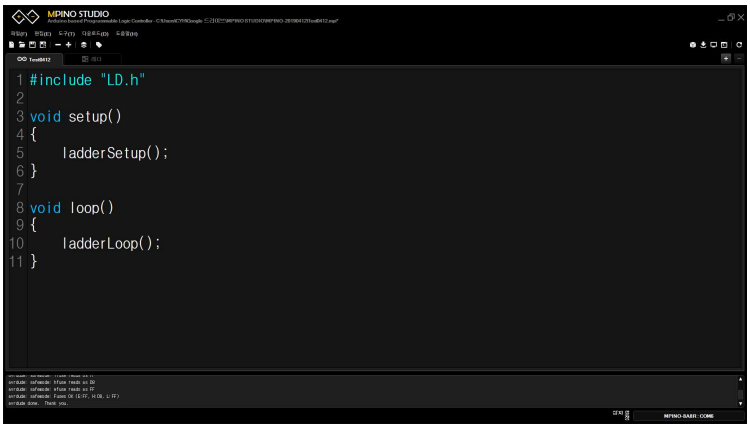
	"START" 버튼을 누르지 않고 있으면, “MOTOR" 코일은 ON조건을 성립하지 않으므로 OFF되고 있다.
	"START" 버튼을 누르면,"START" 버튼은 ON되고 이후에 N.C접점인 “STOP" 버튼은 버튼을 누르지 않을 때는 연결된 접점이므로, ON된다. 따라서 ”MOTOR" 코일은 ON된다.
	"START" 버튼을 누르다가 떼면, “START" 버튼은 OFF된다. 하지만, 그 전에"MOTOR" 코일이 ON되었을 때, 동시에 ”MOTOR" N.O접점이 ON되어, "MOTOR" 코일은 계속 ON되어진다. (자기유지회로)
	N.C접점인 “STOP" 버튼은 누르지 않을 때는 계속 ON되어 있다. 이후, 정지하고 싶을 때, 누르면 이어진 선이 끊어지면서 ”MOTOR"코일을 OFF시키며, 동시에 "MOTOR" N.O접점도 OFF되어진다.

3.3. LADDER LOGIC 실행

LADDER LOGIC 실행을 위해선 아래의 세가지 조건을 충족해야 합니다.

- #include "LD.h"
- ladderSetup()
- ladderLoop()

세가지 조건은 새파일로 프로젝트를 생성할 때, 기본적으로 생성됩니다.



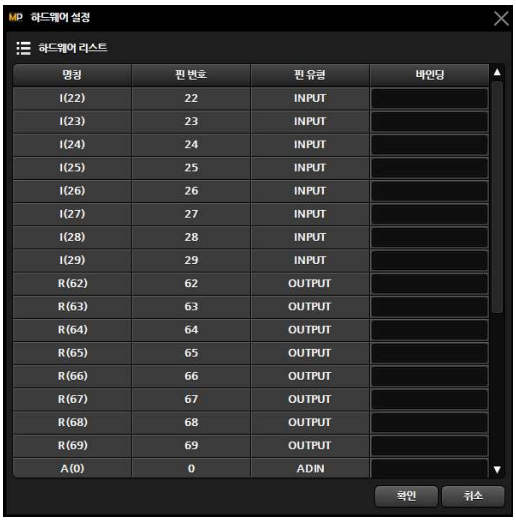
3.4. 포트번호 바인딩

LADDER LOGIC에서 PLC처럼 포트번호를 사용하기 위해서 도구 -> 하드웨어에서 바인딩 해야 합니다. 사용하지 않는 포트는 바인딩 하지 않아도 됩니다. 바인딩 하지 않으면, 해당 포트는 LADDER LOGIC에서 사용하지 않게 되며, 아두이노 C언어에서 제약없이 사용하실 수 있습니다.

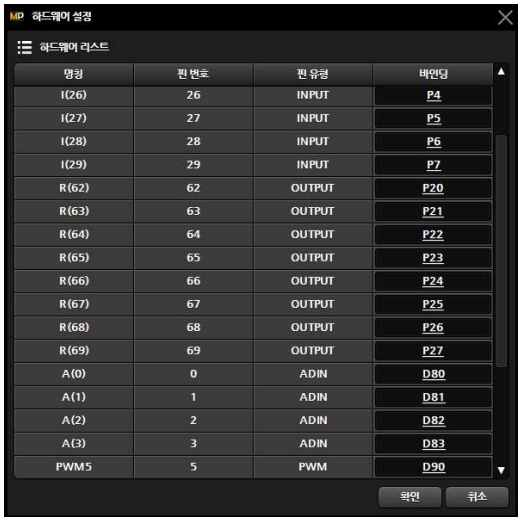
도구 -> 하드웨어를 실행하여, MPINO 제품군 터미널블럭에 새겨져 있는 명칭을 Px 또는 Dx로 LADDER LOGIC에서 사용하고자 하는 메모리로 바인딩 합니다.

디지털 입력, 트렌지스터 출력, 릴레이 출력은 Px로 바인딩 합니다. (P0, P1..)
아날로그 입력, 아날로그 출력, PWM, 고속카운터는 Dx로 바인딩 합니다. (D0, D1...)

Px는 비트 Dx는 워드단위로서, LADDER LOGIC에서 정해져 있는 메모리 이름입니다. 관련 내용은 LADDER LOGIC에서 자세히 확인할 수 있습니다.



[바인딩 이전]



[바인딩 이후]

3.5. N.O접점

Normal Open의 약자로서, A접점이라고도 불러집니다. 평상시에는 열려 있다가 접점에 매칭된 비트가 ON이 되면 닫히는 접점입니다.

N.O접점은 비트단위의 메모리를 매칭하여 사용할 수 있습니다.
가령 P0, M0, D0.0이 있습니다.

3.6. N.C접점

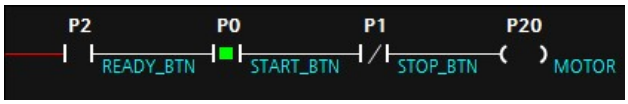
Normal Close의 약자로서, B접점이라고도 불러집니다. 평상시에는 닫혀 있다가 접점에 매칭된 비트가 ON이 되면 열리는 접점입니다.

N.C접점은 N.O접점과 똑같이 비트단위의 메모리를 매칭하여 사용할 수 있습니다.

3.7. 출력코일

코일 앞의 로직들의 논리합이 참일때 코일출력에 매칭된 비트가 On되며, 거짓일 때 코일출력에 매칭된 비트가 Off됩니다.

아래 로직은 디지털입력 P0포트에 전원을 인가하여 On시켰을 때의 모니터링 상태를 나타내었습니다.



위 로직에서 로직의 상태표는 아래와 같습니다.

버튼상태	P2 상태값	P0 상태값	P1 상태값	P20 상태값
모든버튼 0	0	0	1	0
시작버튼 1	0	1	1	0
준비버튼 1	1	0	1	0
준비버튼 1 시작버튼 1	1	1	1	1
모든버튼 1	1	1	0	0

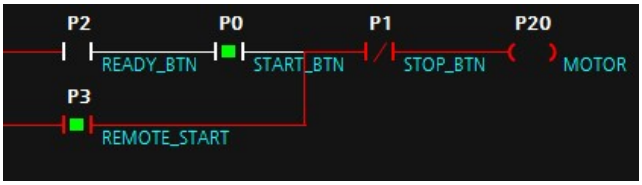
3.8. SET / RESET

출력코일은 조건에 따라 ON / OFF 가 됩니다. 조건이 성립하였을 때, 계속적으로 ON 시키고 싶을 경우와 RESET을 시키고 싶을 경우 아래와 같이 사용할 수 있습니다.

디지털입력 P0가 On이 되면, 디지털출력 P20이 계속적으로 On됩니다. 이후, 디지털입력 P1가 On되면, 디지털출력 P20은 Off되는 로직입니다.

3.9. OR접점

아래 로직은 OR접점인 “P3"을 추가하였을 때의 모니터링 상태를 나타내었습니다. P20이 ON이 되기 위해서는 P2와 P0 모두가 ON이거나, P3이 ON이 되어야 합니다.



자세한 사항은 아래 로직 상태표를 참고하세요.

버튼상태	P2 상태값	P0 상태값	P3 상태값	P1 상태값	P20 상태값
모든버튼 0	0	0	0	1	0
시작버튼 1	0	1	0	1	0
준비버튼 1	1	0	0	1	0
준비버튼 1 시작버튼 1	1	1	0	1	1
원격버튼 1	0	0	1	1	1
준비버튼 1 시작버튼 1 원격버튼 1 정지버튼 1	1	1	1	0	0

3.10. 비교문

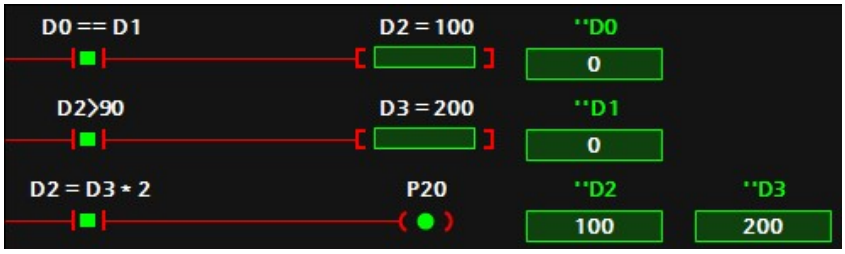
3.10.1. 부등호

비교문은 N.O접점과 N.C접점에서 아래와 같은 부등호를 사용할 수 있습니다.

부등호	예 시	설 명
==	x == y	x가 y가 같다
!=	x != y	x와 y가 다르다
<	x < y	x가 y보다 작다
>	x > y	x가 y보다 크다
<=	x <= y	x 가 y보다 작거나 같다

3.10.2. 워드 비교문

아래는 D0와 D1이 같으면, D2에 100의 상수값을 저장하고, D2가 90보다 크면, D3에 200의 상수값을 저장하는 로직입니다.



3.10.3. 더블워드 비교문

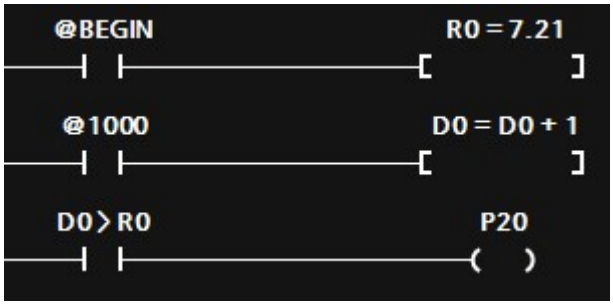
더블워드 메모리는 형변환을 하여 아래와 같이 비교할 수 있습니다.

더블워드 메모리는 대입과 비교문을 사용하기 위해서 위 로직과 같이 형변환을 사용해야 합니다. D2를 더블워드로 메모리로 형변환하면, D2와 D3 메모리 영역을 사용합니다.

3.10.4. 실수 비교문

실수와 워드(더블워드) 메모리와 비교할 경우, 실수값의 절대값과 비교합니다.

아래는 R0 실수 메모리에 7.21 실수값을 저장하고, D0 워드 메모리가 1초에 한번씩 1씩 증가하다가 D0가 8이 되었을 때 D0값이 R0의 절대값 7보다 크므로 디지털출력 P20이 ON되는 로직입니다.



3.11. 형변환

16Bit를 32Bit로 D0 메모리를 형변환 할 경우, D0와 D1 메모리 영역을 사용합니다.

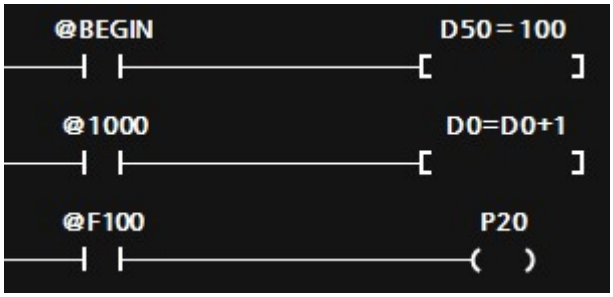
변환식	데이터 타입	Bits	사용 범위	예 시
UD(x)	unsigned long	32Bit	0 ~ 4,294,967,295	UD(D0)
UW(x)	unsigned int	16Bit	0 ~ 65,535	UW(D0)
WD(x)	int	16Bit	-32,768 ~ 32,767	WD(D0)
DW(x)	long	32bit	-2,147,483,648 ~ 2,147,483,647	DW(D0)
REAL(x)	float	32Bit	-3.4028235E+38~3.4028235E+38	REAL(D0)

3.12. 특수 릴레이

N.O접점과 N.C접점에 아래와 같은 특수 릴레이를 사용할 수 있습니다.

특수 릴레이	설 명
@10	10ms 마다 1번(1스캔동안) ON
@100	100ms 마다 1번(1스캔동안) ON
@1000	1000ms 마다 1번(1스캔동안) ON
@F10	10ms 마다 ON/OFF를 반복, 10ms ON->10ms OFF
@F100	100ms 마다 ON/OFF를 반복, 100ms ON->100ms OFF
@F1000	1000ms 마다 ON/OFF를 반복, 1000ms ON->1000ms OFF
@ON	항상 ON
@OFF	항상 OFF
@BEGIN	LADDER 처음 1번(1스캔동안) ON

아래는 처음에 한번만 D50에 100 상수값을 저장하고, 1초에 한번씩 D0를 1씩 증가시키며, 100ms마다 P20출력포트를 On/Off를 반복하는 로직입니다.



3.13. NOT

NOT접점은 로직의 상태값을 반전시키는 접점입니다. 로직 상태가 1일 경우 0로 만들고, 0일 경우 1로 만듭니다.

아래는 디지털입력 P0가 OFF일때 디지털출력 P20이 ON되고, 디지털 입력 P0가 ON일때 디지털출력 P20이 OFF되는 로직입니다. NOT접점에 의해 P0 입력에 의한 출력이 반전되어 동작됩니다.

3.14. RISING EDGE / FALLING EDGE

3.14.1. 상승검출접점

상승검출(RISING EDGE)는 로직 상태값이 0에서 1이 되었을 때, 1번(1스캔동안)만 ON되고 이후 OFF가 되는 접점입니다. 이후, 로직 상태값이 0에서 1이 되면 다시 1번만 동작됩니다.

아래는 디지털입력 P0가 On되면 한번만 "D0=D0+1" 펑션을 실행하여 D0가 1이 되는 로직입니다. 상승검출접점이 없다면, 디지털입력 P0가 On이 되어 있는 동안 계속적으로 "D0=D0+1"이 실행되어 D0는 계속적으로 1씩 증가될 것입니다.

3.14.2. 하강검출접점

하강검출(FALLING EDGE)는 로직 상태값이 1에서 0가 되었을 때, 1번(1스캔동안)만 ON되고 이후 OFF가 되는 접점입니다. 이후, 로직 상태값이 1에서 0이 되면 다시 1번만 동작합니다.

아래는 디지털입력 P0가 On되었다가 Off되었을 때 한번만 “D0=D0+1" 펑션을 실행하여 D0가 1이 되는 로직입니다.

3.15. 펑션블럭

펑션블럭은 비트단위 이상의 연산과 특수명령을 실행하기 위해서 사용합니다.

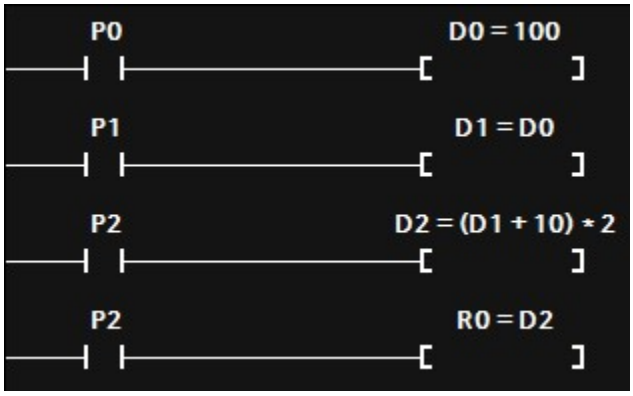
3.15.1. MOVE (메모리에 값 저장)

MOVE 명령어는 대입 연산자인 “=”를 사용해야 합니다.

아래는 디지털입력 P0가 ON되면 D0 워드 메모리에 100의 상수값을 저장하고, 디지털입력 P1이 ON되면 D1 워드 메모리에 D0 워드 메모리 값을 저장하는 로직입니다.

3.15.2. 산술연산

아래는 디지털입력 P0가 ON되면 D0는 100이 되고, 이후 P1이 ON되면 D1에 D0값인 100이 저장되고, 이후 P2가 ON되면 D2는 (100+10)*2의 값인 220이 저장되고, 이후 P2가 ON되면 R0 실수 메모리에는 220.0이 저장됩니다.



3.15.3. C언어 함수 호출

C언어 코드창에서 작성한 함수를 호출할 수 있습니다.

아래와 같이 C_CODE() 함수가 있을 경우,

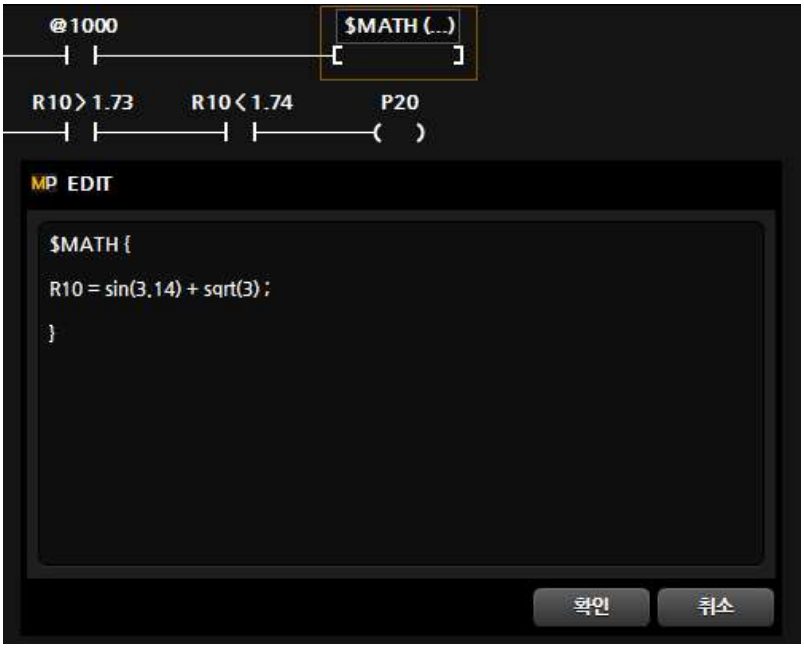
```
#include "LD.h"

void setup()
{
    ladderSetup();
}

void loop()
{
        ladderLoop();
}

void C_CODE()
{
        D50 = ( analogRead(0) / 1023.0 ) * 1000 ;
        if ( D50 > 500 ) digitalWrite( 62, 1 );
        else digitalWrite( 62, 0 );
}
```

아래와 같이 LADDER LOGIC에서 C_CODE()함수를 호출하여 실행할 수 있습니다.

3.15.4. C언어 함수 생성

펑션블럭에서 “$” + “함수명” + blank(Space) + { 실행문1.. 실행문2... .... }을 기입하여 사용할 수 있습니다. ex) $MATH { 실행문; }

아래는 디지털입력 P0가 On될때 R10 실수 메모리에 sin(3.14) + sqrt(3)을 계산하여 저장하고, R10 실수 메모리가 1.73보다 크고 1.74보다 작을 때 디지털출력 P20이 On되는 로직입니다.


@1000
$MATH (...)
R10 > 1.73
R10 < 1.74
P20
MP EDIT
$MATH {
R10 = sin(3.14) + sqrt(3) ;
}
확인
취소

4. 모드버스

MPINO 제품군은 시리얼 통신 전채널에 Modbus RTU Slave 모드를 지원합니다.

4.1. 모드버스 실행 명령어

ModbusRTU Slave는 아두이노의 setup() 문에서 “modbusStart()" 함수로 사용하실 수 있습니다.

- modbusStart(Serial, BaudRate, SlaveAddress)

Serial은 통신포트(Serial 0채널, Serial1 1채널, Serial2 2채널, Serial 3채널)을 BaudRate는 통신속도(4800~115,200)를 SlaveAddress는 주소값(1~255)을 입력할 수 있습니다.

4.2. 모드버스 어드레스

Modbus의 통신영역은 Mpino에서 지정된 메모리 영역만이 사용됩니다.

Modbus의 Start Address는 아래의 표를 참조하여 사용하실 수 있습니다.

	메모리	MPINO	HMI / SCADA
비트	P	0 ~ 127	10001 ~ 10128
	M	4096 ~ 6,553	14097 ~ 16554
워드	D	0 ~ 999	40001 ~ 41000
	T	1000 ~ 1999	41001 ~ 42000
	C	2000 ~ 2999	42001 ~ 43000
	WM	3000 ~ 3999	43001 ~ 44000
	R	4000 ~ 4999	44001 ~ 45000

상위 메모리는 LADDER LOGIC에서 사용하는 메모리이지만, C언어창에서도 사용할 수 있는 메모리 영역입니다.

C언어창에서 "D50 = 0x1234;"와 같이 코딩하여 통신 맵을 구성하실 수 있습니다.

4.3. 펑션코드의 종류

- 펑션코드 01 : Read Coil Status
  - 메모리를 BIT 단위로 읽습니다.
- 펑션코드 02 : Read Input Status
  - 메모리를 BIT 단위로 읽습니다. (펑션코드 02와 동일)
- 펑션코드 03 : Read Holding Registers
  - 메모리를 WORD 단위로 읽습니다.
- 펑션코드 04 : Read Input Registers
  - 메모리를 WORD 단위로 읽습니다. (펑션코드 03과 동일)
- 펑션코드 05 : Force Single Coil
  - 하나의 BIT 메모리를 On/Off 시킵니다.
- 펑션코드 06 : Preset Single Registers
  - 하나의 WORD 메모리값을 변경시킵니다.
- 펑션코드 15 : Force Multiple Coils
  - 여러 BIT 메모리를 On/Off 시킵니다.
- 펑션코드 16 : Preset Multiple Registers
  - 여러 WORD 메모리를 변경시킵니다.

4.4. CRC

- CRC는 Modbus RTU의 통신장애를 검출하기 위한 CheckSum입니다. CRC의 계산 방법은 아래의 주소에서 확인하실 수 있습니다.

www.lammertbies.nl/comm/info/crc-calculation.html

4.5. 에러처리

- Modbus 수신에 대한 문제가 발생했을 때, 응답신호를 보내지 않습니다. Master에서는 응답신호가 없을 시 재전송 등을 하여 주시기 바랍니다.

4.6. HMI / SCADA 시스템에서의 어드레스 값

HMI 또는 SCADA에서는 별도의 펑션코드를 선택하지 않고 어드레스에 펑션코드의 정보를 포함하고 있습니다. 즉, 40001 일 경우에는 펑션코드 4번에 시작 어드레스 0을 의미합니다. 10002일 경우에는 펑션코드 2번에 시작 어드레스 1을 의미합니다. 시스템에 따라 400001처럼 단위가 변경될 수도 있습니다.

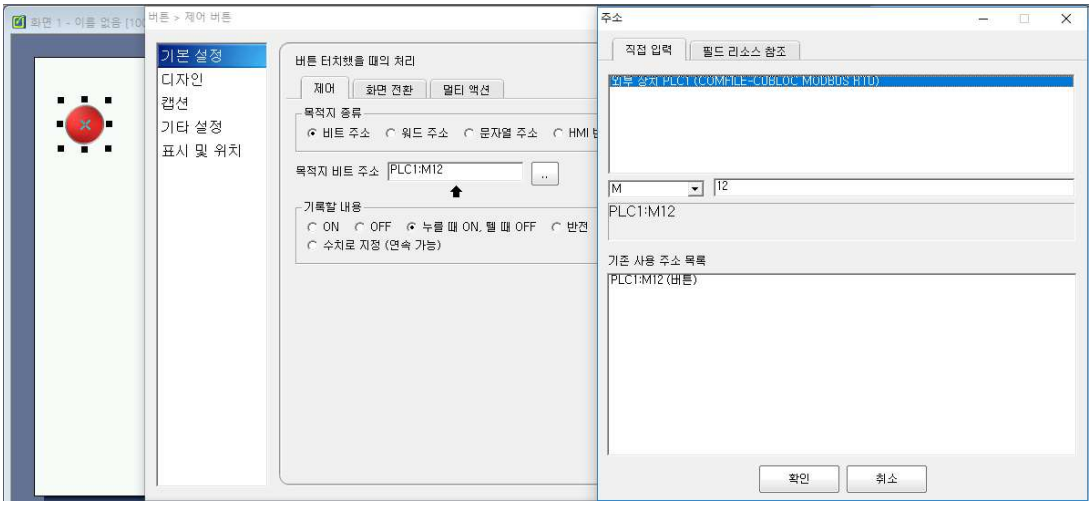
4.7. COMFILE HMI에서 사용법

프로젝트 생성시 또는 프로젝트 설정에서 제조사:COMFILE, 모델명 :CUBLOC MODBUS RTU를 선택합니다.

① 제어버튼

M12 BIT를 ON 시키는 예제입니다.

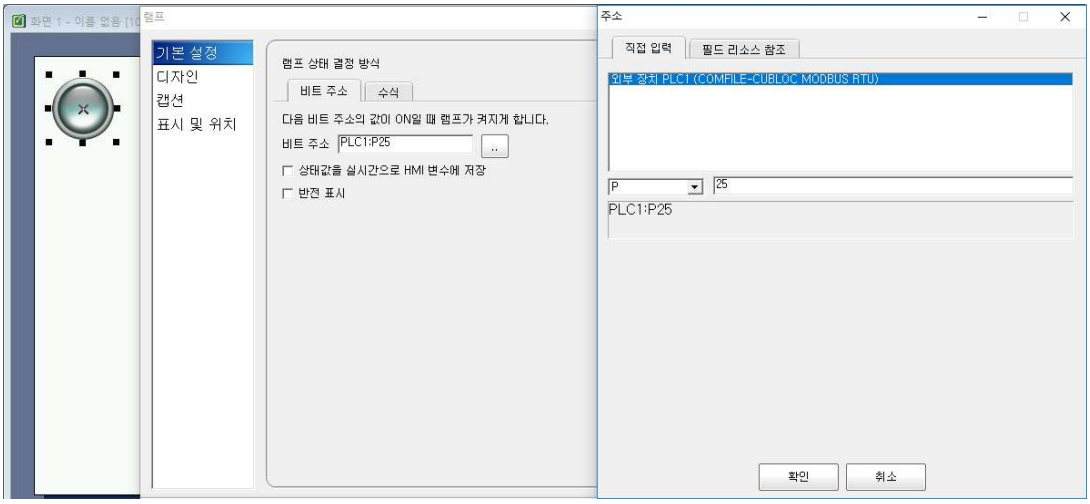
제어버튼 생성 -> 기본설정 -> 제어 -> 비트주소 -> 목적지 비트주소 -> M선택->12 기입



② 램프

P25 출력포트의 상태를 표시하는 예제입니다.

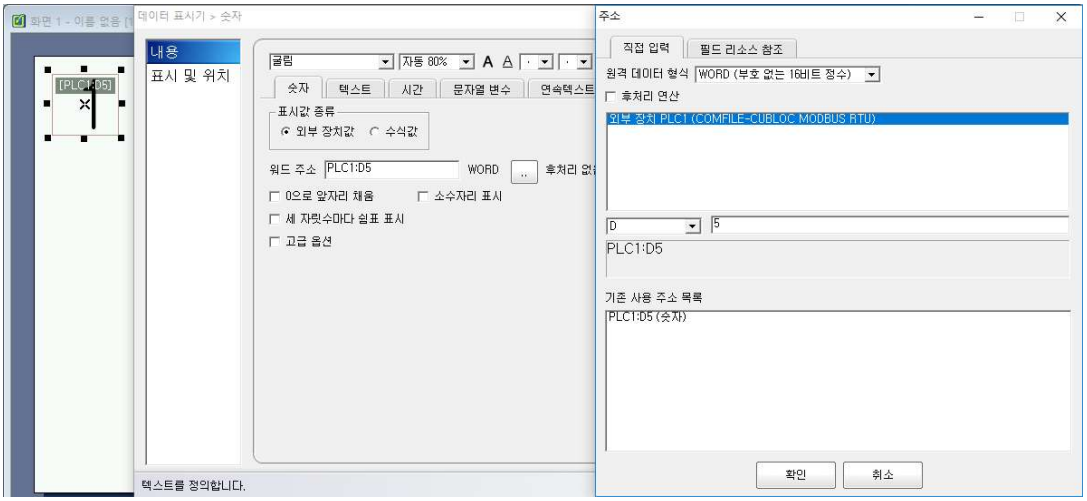
램프 생성 ->기본설정 -> 비트주소 -> P선택 -> 25기입



③ 숫자 표시

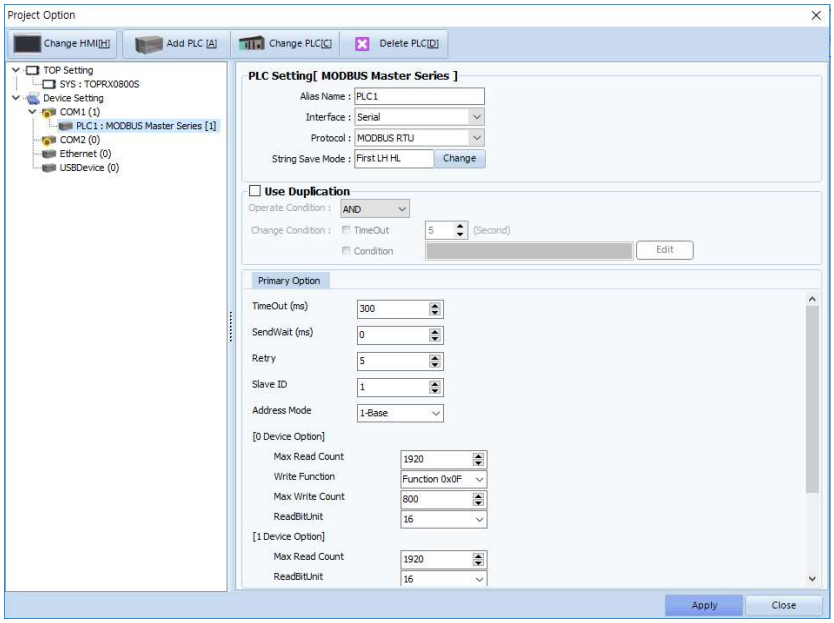
D5메모리 값을 표시하는 예제입니다.

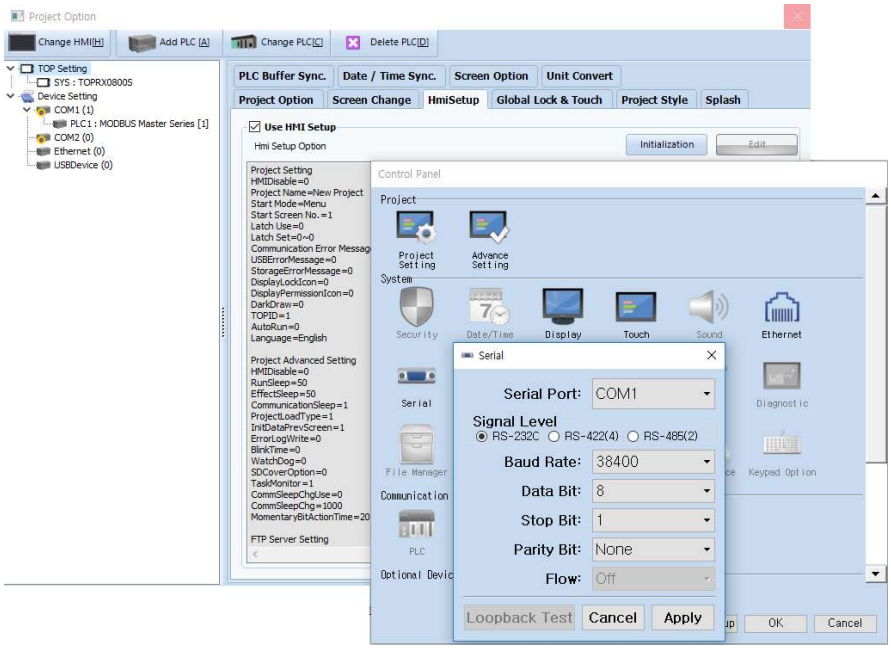
숫자 생성 -> 내용 -> 숫자 -> 외부장치값 -> 워드주소 -> D선택 -> 5기입



4.8. M2I에서 사용법

TOP Design Studio의 프로젝트를 생성하고, 프로젝트 속성에서 PLC를 MODBUS Master Series로 생성합니다. Protocaol을 MODBUS RTU로 선택한 이후, TOP Setting -> HmiSetup -> 체크 Use HMI Setup -> Edit -> Serial -> 통신속도 등을 설정합니다.

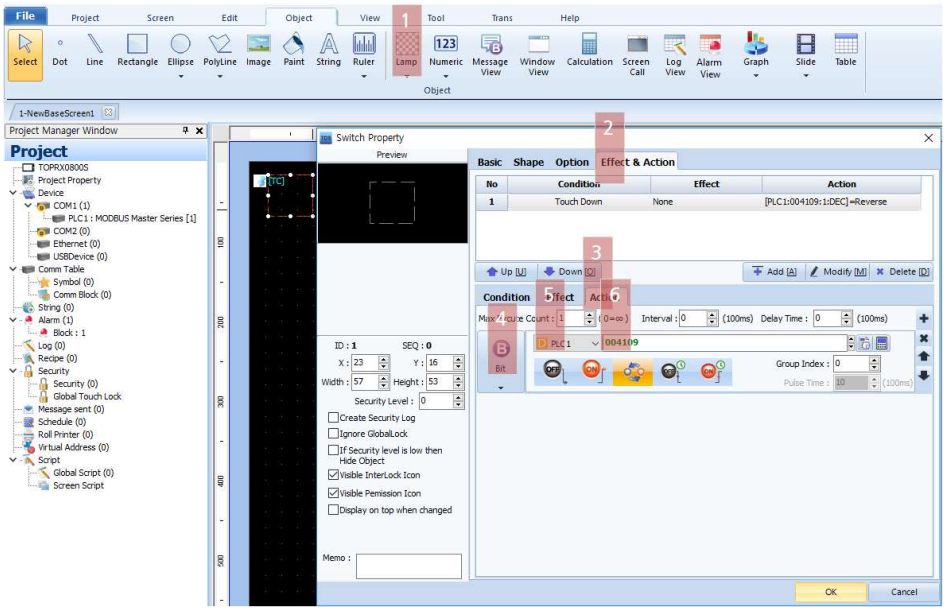




① 제어버튼

M12 BIT를 ON 시키는 예제입니다.
터치 생성 -> Effect&Action -> Action -> Bit -> PLC1 -> 004109 -> ADD
비트 읽기 쓰기는 는 M2I에서 000001 또는 100001부터 시작합니다. (비트시작주소)

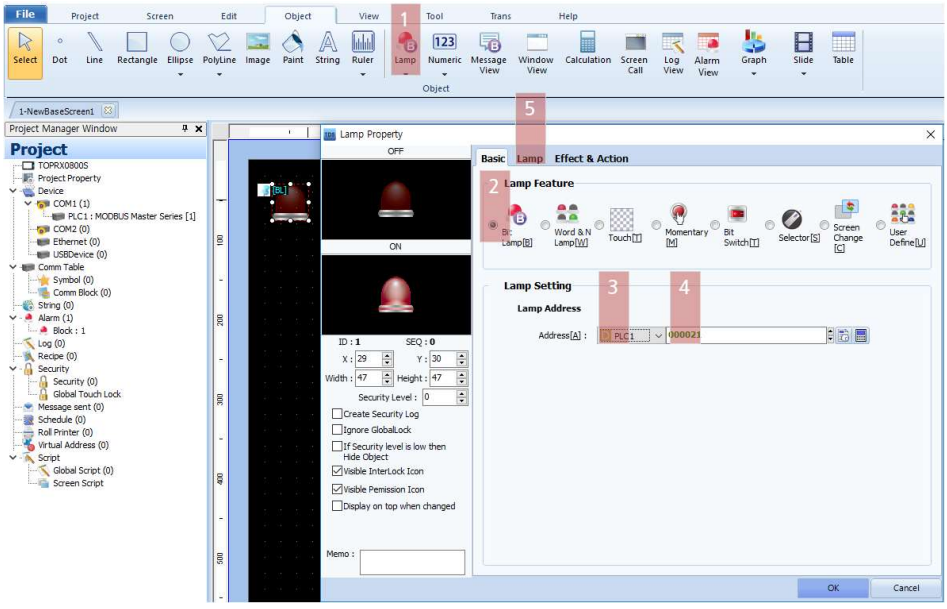
M12 : M메모리 시작주소(4096) + Bit(12) + 비트시작주소(000001) = 004109



② 램프

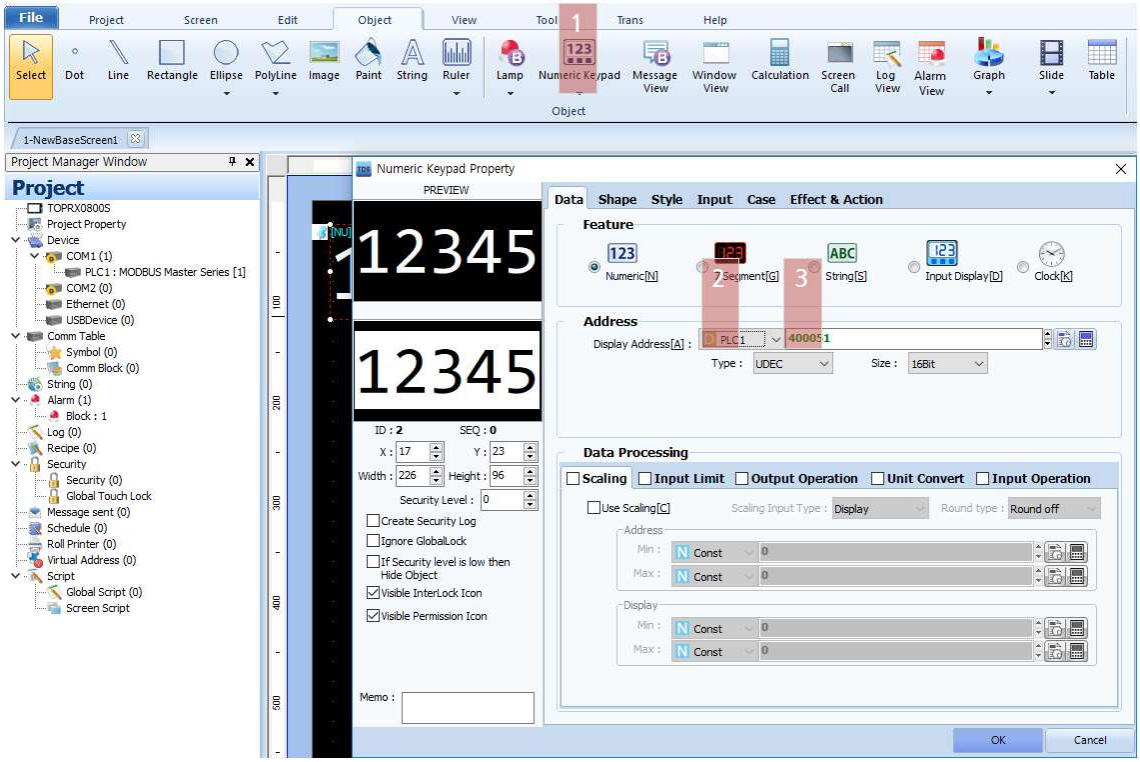
P20 출력포트 상태를 램프로 표시하는 예제입니다.
램프 생성 -> Bit Lamp -> PLC1 -> 000021 -> Lamp에서 이미지 선택

P20 : P메모리 어드레스(0) + Bit(20) + 비트시작주소(000001) = 000021



③ 램프

D50 메모리 값을 표시하고 키패드로 변경하는 예제입니다.
Numberic Keypad 생성 -> PLC1 -> 400051
워드 쓰기 읽기는 M2I에서 300001 또는 400001부터 시작합니다 (워드시작주소)
D50 : D메모리 어드레스(0) + Word(50) + 워드시작주소(400001) = 400051



4.9. 펑션코드 01 : READ COIL STATUS
4.10. 펑션코드 02 : READ INPUT STATUS

- 메모리를 BIT단위로 읽을 수 있는 펑션코드입니다.
- 아래는 2번 슬레이브 제품에서 M15 ~ M33을 읽어오는 예 입니다.

- Master에서 송신

패 킷 명	구 분	패 킷 (16진수)
슬레이브 어드레스		02
펑션코드		01
어드레스	HIGH BYTE	10
어드레스	LOW BYTE	0F
길이	HIGH BYTE	00
길이	LOW BYTE	1F
CRC	LOW BYTE	49
CRC	HIGH BYTE	32

- 어드레스 : M메모리 시작 어드레스(4069) + M15(15) = 1015 = 0x100F
- 길이 : 읽고자 하는 비트의 개수를 의미합니다.

- Slave로 부터 수신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		02
펑션코드		01
바이트 카운트		04
데이터	M15 ~ M22	01
데이터	M23 ~ M30	02
데이터	M31 ~ M32 + Dummy	03
CRC	LOW BYTE	9D
CRC	HIGH BYTE	AC

- 바이트 카운트 : M15~M33은 총 19개의 비트이며, 바이트로 환산하면 3바이트
- 데이터 : M15 ~ M22의 데이터 패킷은 아래와 같습니다.

BIT	M22	M21	M20	M19	M18	M17	M16	M15
BIN	0(MSB)	0	0	0	0	0	0	1(LSB)
HEX	0				1			

4.11. 펑션코드 03 : READ HOLDING REGISTERS
4.12. 펑션코드 04 : READ INPUT REGISTERS

- 메모리를 WORD단위로 읽을 수 있는 펑션코드입니다.
- 아래는 2번 슬레이브 제품에서 D32~D33를 읽어오는 예 입니다.

- Master에서 송신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		02
펑션코드		03
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	20
길이	HIGH BYTE	00
길이	LOW BYTE	02
CRC	LOW BYTE	C5
CRC	HIGH BYTE	F2

- 어드레스 : D메모리 시작 어드레스(0) + D32(32) = 32 = 0x0020
- 길이 : 읽고자 하는 데이터의 워드 개수를 의미합니다.

- Slave로 부터 수신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		02
펑션코드		03
바이트 카운트		04
데이터1 (D32)	HIGH BYTE	12
데이터1 (D32)	LOW BYTE	34
데이터2 (D33)	HIGH BYTE	56
데이터2 (D33)	LOW BYTE	78
CRC	LOW BYTE	B2
CRC	HIGH BYTE	07

- 바이트 카운트 : 응답하는 데이터의 바이트 개수를 의미합니다.
- 데이터 : D32의 데이터 패킷은 아래와 같습니다.

BYTE	D32 (HIGH BYTE)							
BIN	0(MSB)	0	0	1	0	0	1	0(LSB)
HEX	1				2			

BYTE	D32 (LOW BYTE)							
BIN	0(MSB)	0	1	1	0	1	0	0(LSB)
HEX	3				4			

4.13. 펑션코드 05 : FORCE SINGLE COIL

- 메모리중 하나의 BIT를 변경하기 위한 펑션코드입니다.
- 아래는 3번 슬레이브 제품의 P21 출력접점을 ON 시키는 예입니다.

- Master에서 송신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		03
펑션코드		05
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	15
데이터	FF: HIGH, 00: LOW	FF
데이터	00	00
CRC	LOW BYTE	9C
CRC	HIGH BYTE	1C

- 어드레스 : P메모리 시작 어드레스(0) + P21(21) = 21 = 0x0015
- 데이터 : 0xFF00는 HIGH로 변경하고 0x0000는 LOW로 변경시킵니다.

- Slave로 부터 수신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		03
펑션코드		05
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	15
데이터	HIGH BYTE	FF
데이터	LOW BYTE	00
CRC	LOW BYTE	9C
CRC	HIGH BYTE	1C

- Slave는 수신받은 데이터 패킷과 같은 데이터 패킷으로 응답합니다.

4.14. 펑션코드 06 : PRESET SINGLE REGISTERS

- 메모리중 하나의 WORD를 변경하기 위한 펑션코드입니다.
- 아래는 3번 슬레이드의 D16 메모리를 0x8520으로 변경하는 예입니다.

- Master에서 송신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		03
펑션코드		06
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	10
데이터	HIGH BYTE	85
데이터	LOW BYTE	20
CRC	LOW BYTE	EA
CRC	HIGH BYTE	A5

- 어드레스: D메모리 사작 어드레스(0) + D16(16) = 16 = 0x0010

- Slave로 부터 수신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		03
펑션코드		06
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	10
데이터	HIGH BYTE	85
데이터	LOW BYTE	20
CRC	LOW BYTE	EA
CRC	HIGH BYTE	A5

- Slave는 수신받은 데이터 패킷과 같은 데이터 패킷으로 응답합니다.

4.15. 펑션코드 15 : FORCE MULTIPLE COILS

- 메모리중 여러개의 BIT를 변경하기 위한 펑션코드입니다.
- 아래는 1번 슬레이브의 M27~M40의 BIT값을 변경하는 예입니다.

- Master에서 송신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		01
펑션코드		0F
어드레스	HIGH BYTE	10
어드레스	LOW BYTE	1B
길이	HIGH BYTE	00
길이	LOW BYTE	0E
바이트 카운트		02
데이터1	FIRST BYTE	8B
데이터2	SECOND BYTE	35
CRC	LOW BYTE	50
CRC	HIGH BYTE	C5

- 어드레스: M메모리 시작 어드레스(4096) + M27(27) = 4123 = 0x101B
- 길이: M27~M40의 비트개수 = 14 = 0x0E
- 바이트 카운트: 14비트의 바이트개수 = 2
- 데이터: M27부터 시작하는 데이터로서 아래의 표를 참고해 주세요

Bit	M40	M39	M38	M37	M36	M35	M34	M33	M32	M31	M30	M29	M28	M27
Bin	1	1	0	1	0	1	1	0	0	0	1	0	1	1
Hex	3		5				8				B			
	SECOND BYTE						FIRST BYTE							

- Slave로 부터 수신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		01
펑션코드		0F
어드레스	HIGH BYTE	10
어드레스	LOW BYTE	1B
길이	HIGH BYTE	00
길이	LOW BYTE	0E
CRC	LOW BYTE	A0
CRC	HIGH BYTE	C8

4.16. 펑션코드 16 : PRESET MULTIPLE REGISTERS

- 메모리중 여러개의 WORD를 변경하기 위한 펑션코드입니다.
- 아래는 1번 슬레이브 어드레스 제품의 D22~D24을 0x1234,0x5678,0x1245로 변경하는 예입니다.

- Master에서 송신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		01
펑션코드		10
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	16
길이	HIGH BYTE	00
길이	LOW BYTE	03
바이트 카운트		06
데이터1 (D22)	HIGH BYTE	12
데이터1 (D22)	LOW BYTE	34
데이터2 (D23)	HIGH BYTE	56
데이터2 (D23)	LOW BYTE	78
데이터3 (D24)	HIGH BYTE	12
데이터3 (D24)	LOW BYTE	45
CRC	LOW BYTE	E9
CRC	HIGH BYTE	7E

- 어드레스: D메모리 시작 어드레스(0) + D22(22) = 22 = 0x0016
- 길이: D메모리 개수 = 3
- 바이트 카운트 : 수신받을 바이트 개수 = 6

- Slave로 부터 수신

패킷명	구분	패킷 (16진수)
슬레이브 어드레스		01
펑션코드		10
어드레스	HIGH BYTE	00
어드레스	LOW BYTE	16
길이	HIGH BYTE	00
길이	LOW BYTE	03
CRC	LOW BYTE	61
CRC	HIGH BYTE	CC