32 비트 C 프로그래머블 산업용 콘트롤러

MOACON™

사용 설명서

VERSION 2.0

www.comfilewiki.co.kr 에 가시면 모아콘 라이브러리 참조표가 있습니다.
알파벳순으로 정리되어 있어 프로그래밍하실 때 참조하시기 편리하도록 되어 있습니다.

COMFILE TECHNOLOGY

컴파일 테크놀로지㈜

등록상표

WINDOWS 는 Microsoft Corporation 의 등록상표입니다.
CUBLOC 은 Comfile Technology 의 등록상표입니다.
MOACON 은 Comfile Technology 의 등록상표입니다.
기타 다른 상표는 해당회사의 등록상표입니다.

알림

본 설명서의 내용은 사전 통보 없이 변경될 수 있습니다. 본 제품의 기능은 성능 개선을 위하여 사전 통보 없이 변경될 수 있습니다. 본 제품을 이 자료에서 설명한 용도 외에서 사용할 경우, 폐사에서는 어떠한 책임도 지지 않으므로 주의하시기 바랍니다. 본 제품은 컴파일 테크놀로지의 고유 기술을 사용하여 개발된 제품으로 특허법, 저작권법에 의한 보호를 받고 있습니다. 따라서 본 제품 (제품에 대한 아이디어 및 설명서 및 기타 포함)의 어떠한 부분도 복사되거나 변경, 유사 제품 및 복제품을 생산할 수 없으며 이를 어길 시 법적 책임을 지게 됩니다.

주의사항

본 제품을 사용하시다가 생긴 손해 및 손실, 간접적인 손상, 재산상의 손실, 부상또는 사망에 대하여 저희 컴파일 테크놀로지 주식회사는 어떠한 법적 책임이 없음을 명시하는 바입니다. 본 제품 사용에 따른 2 차적인 결과에 대해서는 소비자 여러분에게 책임이 있음을 명시합니다. 이에 대한 동의가 없다면 본 제품 사용을 중단하여 주시기바랍니다.

본 제품은 특허출원 제품입니다. 유사 아이디어의 제품은 법적 책임을 지게되므로 유의하시기 바랍니다.
본 제품은 성능향상 및 보완을 위해서, 소비자분들에게 사전공지 없이 기능첨삭을 할 수 있습니다.

생명지원정책

본 제품을 생명유지관련 (의료지원, 수명 지원 및 유지장치) 시스템에 적용을 금지하여주시기 바랍니다.

머릿말

모아콘은 "산업용 C 언어 콘트롤러" (Industrial C-Programmable Controller) 입니다.

모아콘은 고성능 ARM 프로세서를 메인 칩으로 채택하여 빠른 속도로 데이터를 처리할 수 있으며, C 언어를 기본언어로 채택하였습니다. 또한, 필요한 I/O 모듈을 조합하여, 유저 여러분이 원하는 구성으로 I/O 조합을 맞추실 수 있습니다.

모아콘에서 사용하는 C 언어는 여러분이 생각하시는 것만큼 어렵지 않습니다. 초보자가 이해하기 어려운 개념 (객체지향, 포인터등)을 사용하지 않고, 기본적인 문법만으로도 사용할 수 있도록 하였습니다.

본 사용설명서 **"부록 1. C 언어"에서 설명하고 있는 30 여 페이지 분량의 C 언어 문법만 이해하시면, 모아콘을 사용하는데 아무 지장이 없습니다.**

모아콘의 실제기능은 본 사용설명서에서 설명하고 있는 라이브러리 함수로부터 나오는 것입니다. C 언어는 이 함수를 조합할 수 있는 도구 정도로 생각하시면 됩니다.

또한, 다운로드 및 디버깅을 하기위한 별도의 툴이 필요없고, 단지 USB 케이블 만으로 사용이 가능합니다. 개발에 필요한 통합개발 환경 소프트웨어는 저희 회사의 홈페이지를 통해 무료로 제공합니다.

컴파일 테크놀로지㈜에서 축적해온 임베디드 콘트롤러 개발 기술이 함축되어 있는 모아콘으로 여러분들이 생각하시는 시스템을 단기간안에 개발하실 수 있습니다. 많은 이용바랍니다.

컴파일 테크놀로지 주식회사

자주 묻는 질문과 대답

Q. 레더로직은 지원하지 않습니까?

레더로직은 지원하지 않습니다.레더로직을 원하시면 CUBLOC 을 선택하세요.

Q. 다운로드된 프로그램을 다시 업로드 할 수 있습니까?

불가능합니다. 불법복제를 막기위해 업로드 기능을 넣지 않았습니다.

Q. 모아콘을 사용하기 위해 어떤 소프트웨어가 필요합니까?

모아콘 스튜디오 하나만 있으면 됩니다. 모아콘 스튜디오에는 컴파일러와 에디터가 모두 포함되어 있습니다. 모아콘 스튜디오는 www.comfile.co.kr 에서 무료로 다운받을 수 있습니다.

Q. 기존 제품인 큐블록을 사용중입니다. 큐블록과 비교해볼 때 무엇이 다릅니까?

큐블록은 레더로직과 베이직언어를 동시에 사용할 수 있는 "산업용 콘트롤러"입니다. 인터프리터 방식이므로 모아콘에 비해 처리속도가 느리다는 단점이 있습니다.
코어모듈은 반도체형이므로 PCB 에 장착가능합니다. 대량생산시 유리합니다.

모아콘은 32 비트 ARM 프로세서를 채용하였으며, C 언어를 사용합니다. 인터프리터방식이 아닌 컴파일방식이므로 **실행속도가 월등히 빠릅니다.**
모듈형이므로 필요한 조합을 임의대로 구성할 수 있습니다.

Q. 최대 몇 개의 디지털 I/O 포인트를 연결할 수 있습니까?

기본 DIO 모듈과 확장 DIO 모듈을 모두 합치면 256 포인트입니다.

*기본 DIO : 48 점
*확장 디지털 입력 : 128 점
*확장 릴레이 출력 : 80 점

Q. 멀티테스킹을 지원합니까?

멀티테스킹을 지원하지 않습니다. 대신 타임 이벤트를 이용한 백그라운드 처리가 가능합니다.

Q. 맨처음 어떤 제품을 구입하면 됩니까?

최초 구매시에는 "START PACK"을 구입하시면 됩니다. 기본적으로 필요한 모든 구성이 포함되어 있습니다. 나중에 I/O 모듈만 추가로 구입하시면 됩니다.

START PACK 구성품 LIST

- 모아콘 CPU 모듈 : DP-CPU500
- 디지털 입력 모듈 8 점: CF-DIDC8
- 릴레이 출력 모듈 8 점 : CF-DORL8
- 10 슬롯보드
- 24V DC 아답터
- 배선케이블 / 파워케이블
- USB 다운로드 케이블
- 인쇄된 매뉴얼

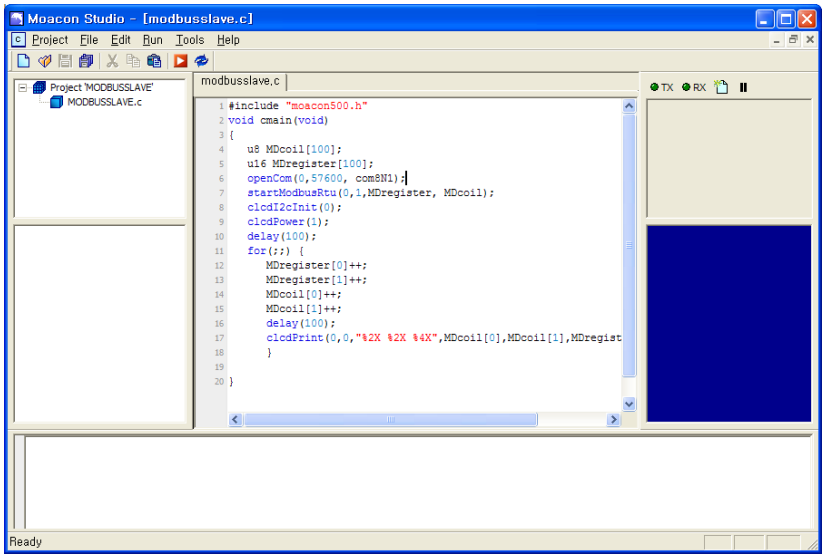
차 례

제 1 장
MOACON
개 요

개발환경 MOACON STUDIO

MOACON 개발에 필요한 통합개발환경 소프트웨어인 MOACON STUDIO 는 저희 회사 홈페이지 (www.comfile.co.kr) 자료실 에서 무료로 다운로드 받은 후 사용하실 수 있습니다.

MOACON STUDIO 는 C 컴파일러와 소스 에디터, 디버그 터미널이 통합된 소프트웨어 입니다. USB 케이블을 사용해서 PC 와 MOACON CPU 모듈을 연결하시고, 통합개발환경 MOACON STUDIO 에서 소스를 작성하십시오.



그리고  실행 아이콘을 클릭하기만 하면, 방금 작성한 프로그램이 MOACON CPU 모듈로 다운로드 되어 실행됩니다.

다운로드된 프로그램은 MOACON CPU 모듈에 있는 플레쉬 메모리에 저장되어, 전원이 없는 상태에서도 계속 유지됩니다. 따라서 개발이 끝나면 USB 케이블을 제거하시고 바로 사용하실 수 있습니다.

전원을 다시 ON 하게 되면, 가장 최근에 다운로드 된 프로그램이 실행됩니다. 추후 언제든지 새로운 프로그램을 다시 다운로드 하실 수 있습니다.

CPU 모듈

CPU 모듈은 가장 중요한 핵심적인 모듈입니다.

다음은 CPU 모듈의 일반적인 특징입니다. (DP-CPU500 기준)

- 32 비트 ARM 프로세서 (ARM CORTEX)
- 클록스피드 : 72MHz
- 프로그램 메모리 : 512K BYTE 플레쉬
- 데이터메모리 : 64K BYTE SRAM
- 비휘발성 메모리 : 32K BYTE FRAM
- 리얼타임 클록과 밧데리 내장
- RS232 통신포트내장
- LCD 와 7 SEGMENT 제어포트 포함
- 24V 전원 입력

I/O 모듈

크게 디지털 입출력 모듈, 아날로그 모듈, 특수기능 모듈로 나누어 집니다.

DIO 모듈 (디지털 입출력 모듈)

모델명	모듈종류	설명	사용전압 / 전류
CF-DOSI8	DC 싱크 출력 8 점	8 개의 DC 출력 (싱크방식)	DC 3.3V ~ 27V 1A
CF-DOSO8	DC 소스 출력 8 점	8 개의 DC 출력 (소스방식)	DC 12V ~ 24V 1A
CF-DORL8	RELAY 출력 8 점	8 개의 릴레이 출력	DC 6 ~ 27V 4A AC 6 ~ 240V 4A
CF-DIDC8	DC 입력 8 점	8 개의 DC 입력 (12V~24V)	DC 12V ~ 24V
I2-EDI16	확장 디지털 입력 16 점	16 개의 DC 입력 (12V~24V)	DC 12V ~ 24V
RS-EDOR8	확장 릴레이 출력 8 점	8 개의 릴레이 출력	DC 6 ~ 27V 4A AC 6 ~ 240V 4A

아날로그 모듈

모델명	모듈종류	설명	사양
RS-SADIN6	고속 AD 입력 6 채널	12 비트 AD 변환 6 점	0~5V, 0~20mA
RS-ADIN4	AD 입력 4 채널	13.3 비트 AD 변환 4 점	0~10V, 1~5V, 4~20mA
RS-HADIN4	고해상도 AD 입력 4 채널	16.6 비트 AD 변환 4 점	0~10V, 1~5V, 4~20mA
RS-THRT4	온도입력 4 채널	PT100 옴 온도센서 4 점	-100 ~ 500 도
RS-DAOUT2	DA 전압 출력 2 채널	16 비트 DA 변환 2 점	0~10V, 0~5V
RS-DAOUT2B	DA 전류 출력 2 채널	16 비트 DA 변환 2 점	0~20mA, 4~20mA

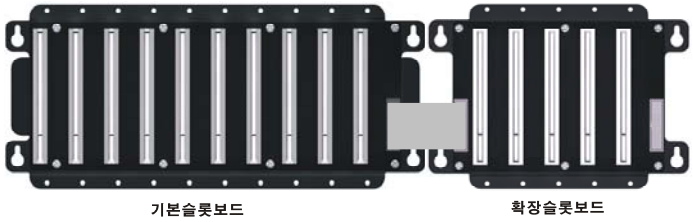
특수기능 모듈

모델명	모듈종류	설명
DP-COMM2	통신모듈	2 개의 RS232 또는 1 개의 RS232, 1 개의 RS485 포트
DP-HCNT	고속카운터모듈	고속카운터입력 2 채널 또는 엔코더입력 2 채널 그리고 PWM 출력 8 채널
DP-ETHER	이더넷 통신모듈	이더넷통신 포트
RS-MOT2	2 축 모션제어 모듈	스탭모터 제어용 펄스출력 2 축

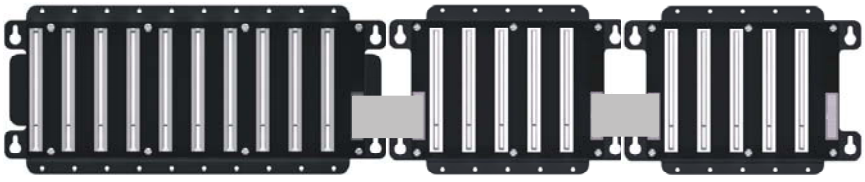
슬롯보드

기본적으로 10 슬롯보드를 사용합니다.

10 슬롯보드로 모자란다면, 다음 그림처럼 확장케이블을 사용해서 5 슬롯보드를 연결한다음 필요한 모듈을 추가로 장착할 수 있습니다.

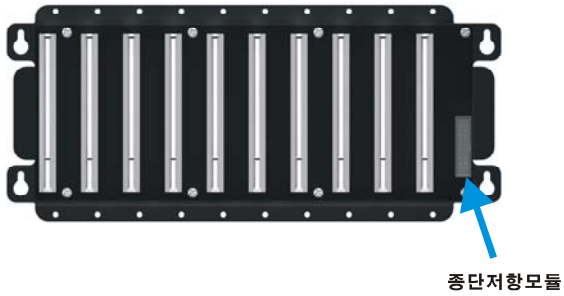


10 슬롯보드는 "기본슬롯보드", 추가된 슬롯보드는 "확장슬롯보드"라고 부릅니다.
15 개 슬롯으로도 모자란다면, 5 슬롯보드를 또 추가하실 수 있습니다. 한 시스템에서 최대로 사용할 수 있는 모듈의 개수는 20 개입니다.

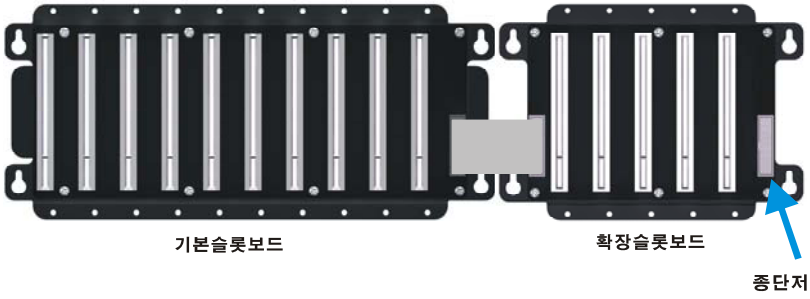


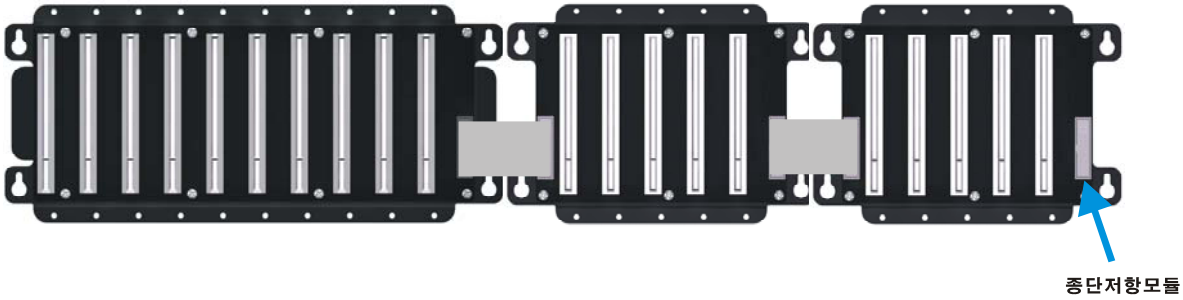
종단저항 모듈 위치

10 슬롯보드는 기본적으로 "종단저항"모듈이 꼽혀 있습니다.



확장 슬롯을 연결하려면, 종단저항 모듈을 제거한 뒤, 케이블을 연결하고 확장슬롯의 오른쪽 빈 소켓에 종단저항모듈을 장착하여 주십시오.



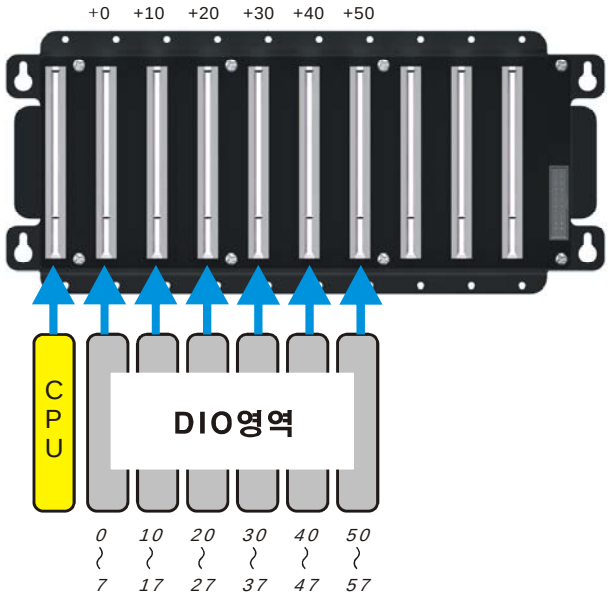


참고로 모델명이 RS 로 시작하는 아날로그모듈과 모션제어모듈은 내부 RS485 네트워크에 연결되는 모듈입니다. 본 종단저항 모듈은 내부 RS-485 네트워크를 위한 것입니다.

DIO 모듈 장착위치

모델명	모듈종류	동시장착 가능수
CF-DOSI8	DC 싱크 출력 8 점	6 개
CF-DOSO8	DC 소스 출력 8 점	6 개
CF-DORL8	RELAY 출력 8 점	6 개
CF-DIDC8	DC 입력 8 점	6 개

CPU 모듈은 가장 왼쪽에 장착하십시오. DIO(디지털 입출력)모듈은 CPU 바로 옆 6 개의 슬롯중 아무곳에나 장착할 수 있습니다. 이 영역을 "DIO 영역"이라고 부릅니다. 이 중 어떤 위치에 장착하느냐에 따라서 I/O 번호가 결정됩니다.



즉, +0 위치에 장착하면 0 부터 7 까지의 I/O 번호를 할당받게 됩니다. +30 위치에 장착하면 30 부터 37 까지의 I/O 번호를 할당받게 됩니다. (I/O 번호는 모두 10 진수입니다.)

주의 : 전원이 인가된 상태에서 모듈을 빼거나 장착하지 마십시오.

다음은 CPU 모듈과 DIO 모듈을 함께 장착한 사진입니다.

[CPU 모듈과 DIO 모듈 6 개]

같은 DIO 모듈을 여러 개 중복사용할 수 있습니다.

[릴레이 출력모듈 3 개를 장착한 모습]

이처럼, 여러분이 필요한 I/O 조합을 6 개의 슬롯안에서 임의대로 구성하실 수 있습니다.

확장 DIO 모듈 장착위치

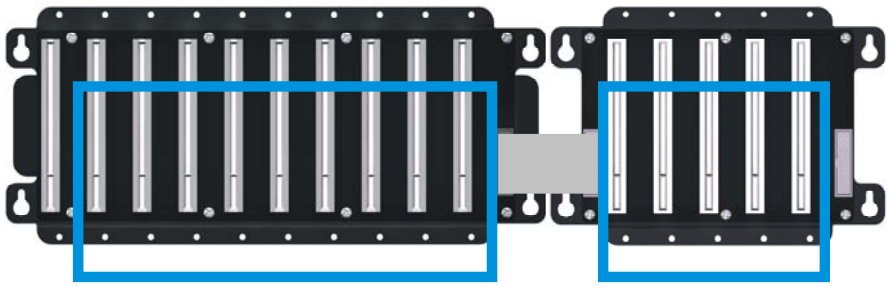
모델명	모듈종류	동시장착 가능수
I2-EDI16	확장 디지털 입력모듈 16 점	8 개
RS-EDOR8	확장 릴레이 출력모듈 8 점	10 개

모아콘에서는 총 48 개의 디지털 I/O 포트를 사용할 수 있습니다. 48 개의 디지털 I/O 만으로 부족하다면 확장 DIO 모듈을 사용하십시오.

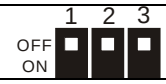
DC 입력을 확장할 수 있는 I2-EDI16 모듈은 한 시스템에서 최대 8 개까지 장착하실 수 있습니다. (최대 128 포인트)

릴레이 출력을 확장할 수 있는 RS-EDOR8 모듈은 한 시스템에서 최대 10 까지 장착할 수 있습니다. (최대 80 포인트)

확장 DIO 모듈은 기본 슬롯보드 또는 확장슬롯 보드중 CPU 모듈위치를 제외한 나머지 위치에 장착할 수 있습니다.



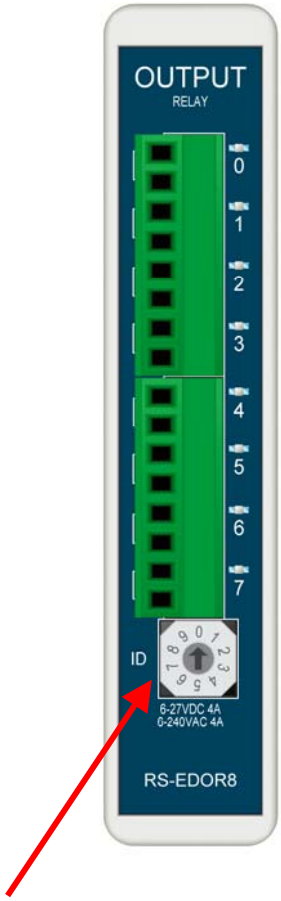
확장 디지털 입력모듈 (I2-EDI16)의 포트번호는 모듈하단에 있는 딥스위치로 결정합니다.

딥 스위치 위치	블록번호	포트 번호 (16 진)
	0	00 번부터 0F 번
	1	10 번부터 1F 번
	2	20 번부터 2F 번
	3	30 번부터 3F 번
	4	40 번부터 4F 번
	5	50 번부터 5F 번
	6	60 번부터 6F 번
	7	70 번부터 7F 번

주의 : 기본 DI 모듈을 우선적으로 사용하시고, 부족한 경우에 확장 DI 모듈을 사용하시는 것이 실행 속도측면에서 유리합니다. (확장 DI 모듈은 내부 I2C 버스를 사용하므로 기본 DI 모듈에 비해 반응속도가 느립니다.)

주의 : 전원이 인가된 상태에서 모듈을 빼거나 장착하지 마십시오.

확장 릴레이 출력 모듈 (RS-EDOR8)은 전면부의 ID 번호를 가지고 포트번호를 결정합니다.

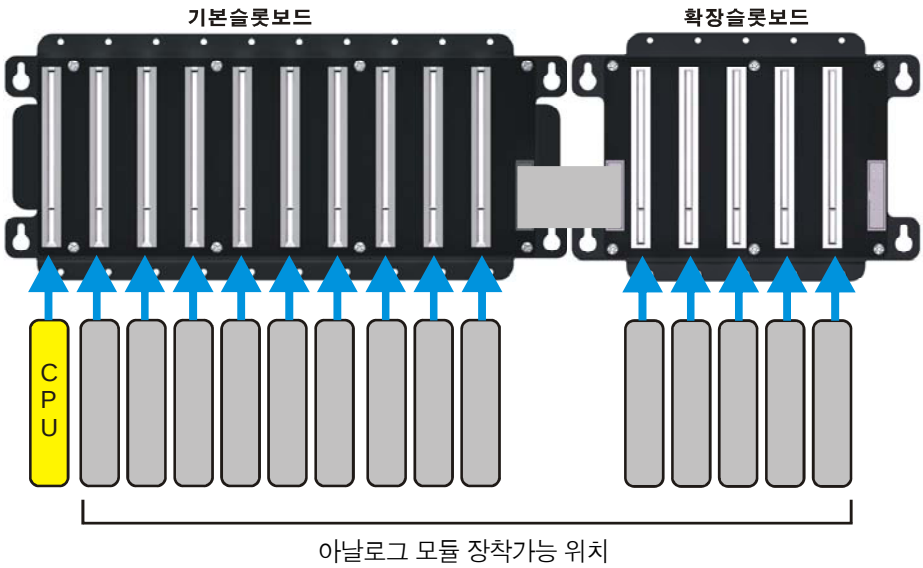


ID 번호	출력포트번호 (10 진)
0	0 부터 7
1	10 부터 17
2	20 부터 27
3	30 부터 37
4	40 부터 47
5	50 부터 57
6	60 부터 67
7	70 부터 77
8	80 부터 87
9	90 부터 97

아날로그 모듈 장착위치

모델명	모듈종류	동시장착 가능 수
RS-SADIN6	고속 AD 입력 6 채널	10 개
RS-ADIN4	AD 입력 4 채널	10 개
RS-HADIN4	고해상도 AD 입력 4 채널	10 개
RS-THRT4	온도입력 4 채널	10 개
RS-DAOUT2	DA 전압 출력 2 채널	10 개
RS-DAOUT2B	DA 전류 출력 2 채널	10 개

아날로그 모듈은, CPU 모듈위치를 제외한 나머지 위치중, 아무 위치에나 장착할 수 있습니다.



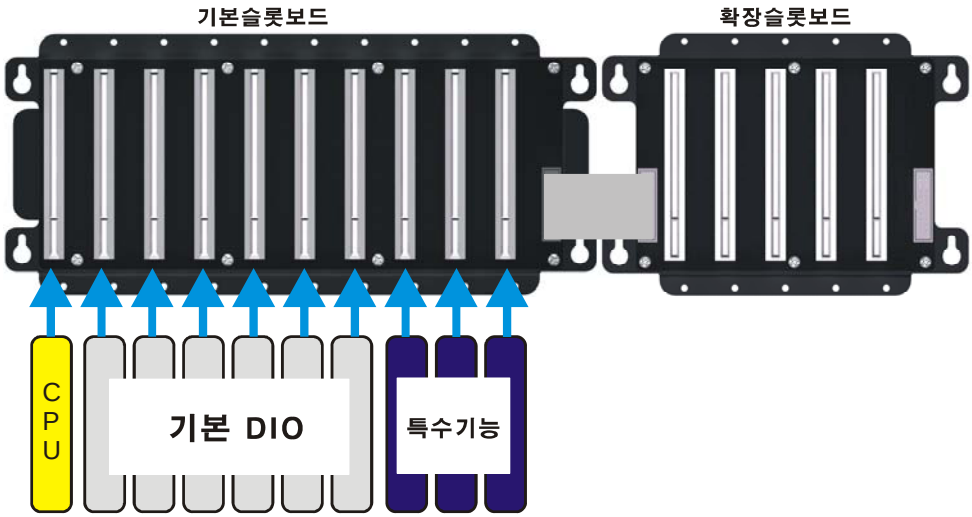
한 종류의 아날로그 모듈을 최대 10 개까지 동시에 사용하실 수 있습니다. 이때 각각의 아날로그모듈의 ID 번호 (전면부에 위치한 로터리 스위치)는 서로 다른번호로 셋팅해주어야 합니다.

주의 : 전원이 인가된 상태에서 모듈을 빼거나 장착하지 마십시오.

특수기능모듈 장착위치

모델명	모듈종류	동시장착 가능 수
DP-COMM2	통신모듈	1 개
DP-HCNT	고속카운터모듈	1 개
DP-ETHER	이더넷 통신모듈	1 개

위 3 가지 특수 기능 모듈은 "기본 슬롯보드"에만 장착할 수 있으며, 종류별로 1 개만 사용할 수 있습니다.



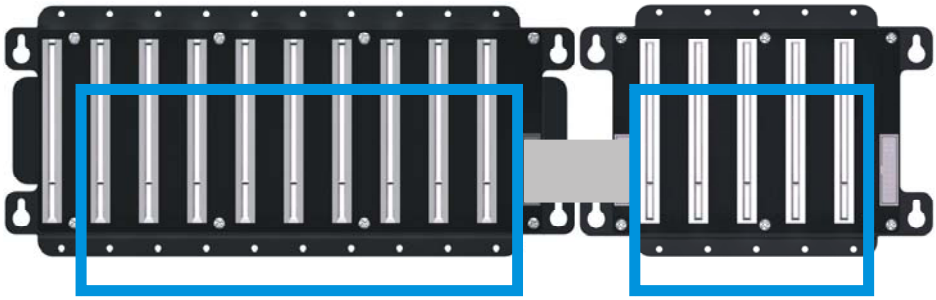
위의 그림처럼 "기본 슬롯보드"의 남는부분에 특수기능 모듈을 장착하십시오. 확장 슬롯보드쪽에는 장착할 수 없습니다.

주의 : 전원이 인가된 상태에서 모듈을 빼거나 장착하지 마십시오.

모션제어 모듈 장착위치

모델명	모듈종류	동시장착 가능 수
RS-MOT2	2 축 모션제어 모듈	10 개

모션 콘트롤 모듈은 CPU 모듈위치를 제외한 나머지 위치중 아무 곳에나 장착할 수 있으며, 동시에 10 개까지 사용가능합니다.



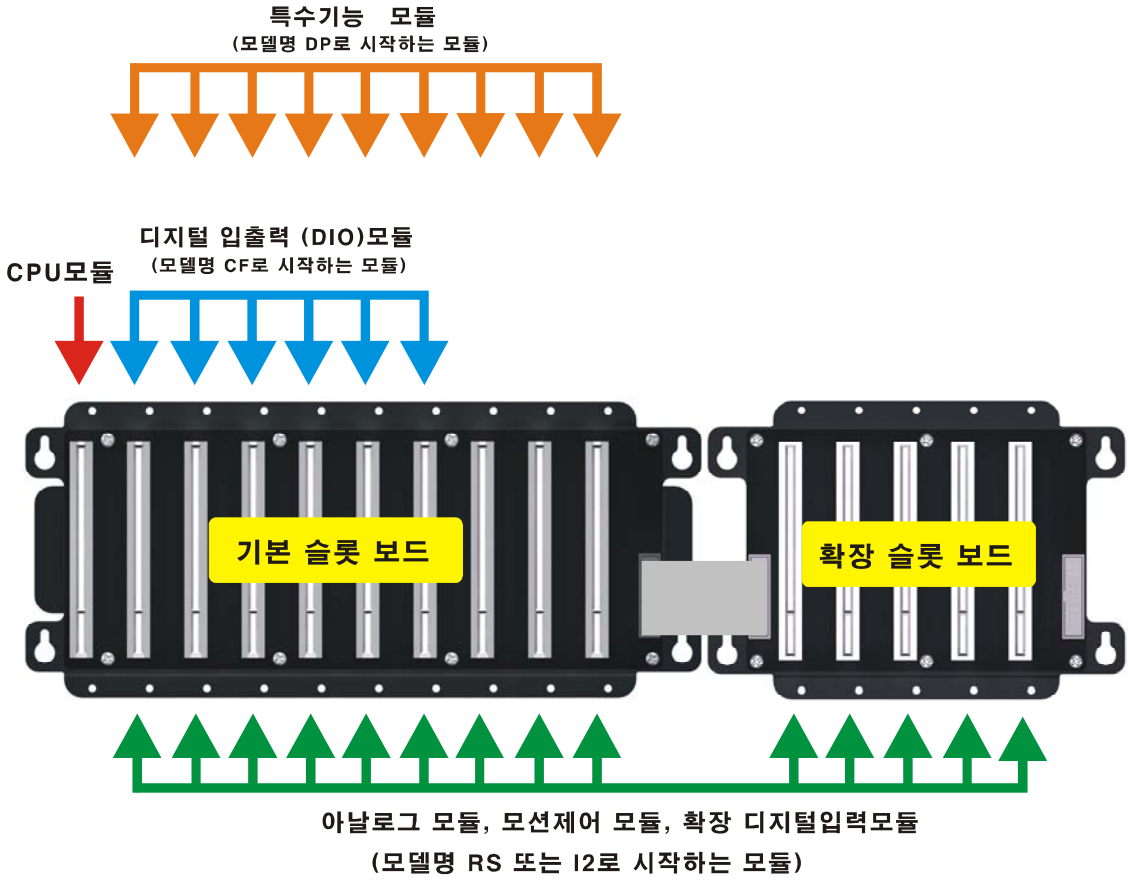
모션제어 모듈은 최대 10 개까지 동시 장착이 가능합니다. 모듈당 2 축을 콘트롤할 수 있으므로, 최대 20 개의 스탭모터를 동시제어할 수 있습니다.

주의 : 전원이 인가된 상태에서 모듈을 빼거나 장착하지 마십시오.

주의 : 모션 제어모듈의 최대출력 주파수는 50KHz 입니다. 그 이상은 출력이 불가능합니다.

모듈별 장착위치 총정리

다음은 모듈별 장착위치를 총정리한 그림입니다.



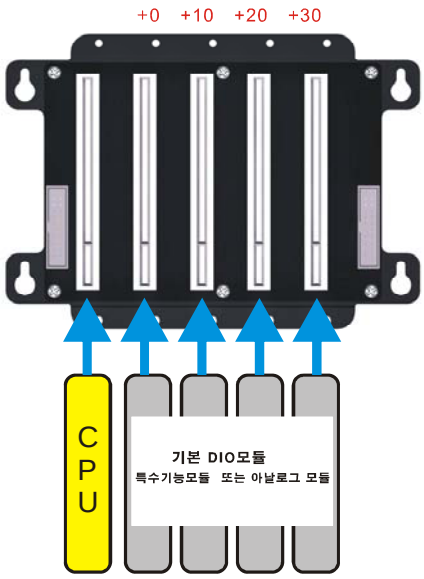
모델명만 보고도 어느 위치에 장착해야 되는지 알 수 있습니다. 예를들어 DP 로 시작하는 통신모듈 (DP-COMM2), 고속카운터 모듈(DP-HCNT), 이더넷 모듈(DP-ETHER)는 기본슬롯보드중, CPU 모듈 위치를 제외한 곳에 모든 위치에 장착가능합니다.

*잘못된 위치에 장착하였을 경우, MOACON STUDIO 상에서 별도의 에러멧세지가 표시되지 않으므로 주의하시기 바랍니다.

작은 시스템 구성

만약 총 5 개의 모듈만으로도 시스템 구성이 가능하다면, 확장용 5 슬롯보드를 기본슬롯 보드로 사용하실 수 있습니다.

5 슬롯보드에서 CPU 모듈위치를 제외한 나머지 4 슬롯에 DIO 모듈 (+0, +10, +20, +30) 또는 아날로그모듈 및 특수기능모듈을 장착하시면 됩니다.



다음은 5 슬롯 보드의 사용예 입니다.

[CPU 모듈과 3 개의 DIO, 그리고 통신모듈]

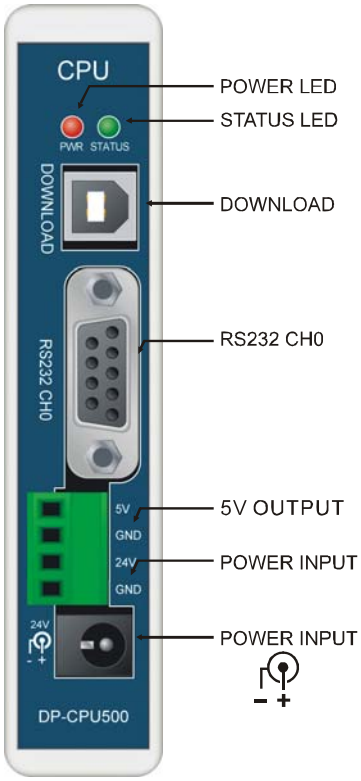
[CPU 모듈과 HCNT 모듈, 통신모듈]

제 2 장

설치방법

MOACON CPU 모듈

MOACON CPU 모듈에 대한 설명입니다.



POWER LED :전원이 인가되면 점등됩니다.

STATUS LED : 프로그램상에서 ON /OFF 가능합니다.

DOWNLOAD : PC 와 USB 케이블로 연결합니다. 유저 프로그램을 다운로드하거나 디버깅하기 위해 접속하는 포트입니다.

RS232 CH0 : RS232 채널 0 입니다.

5V OUTPUT : 내부에서 생성된 5V 를 출력해주는 단자입니다. 이 출력을 다른 기기의 전원으로 사용할 경우에는 반드시 0.5A 이내로만 사용하시기 바랍니다.

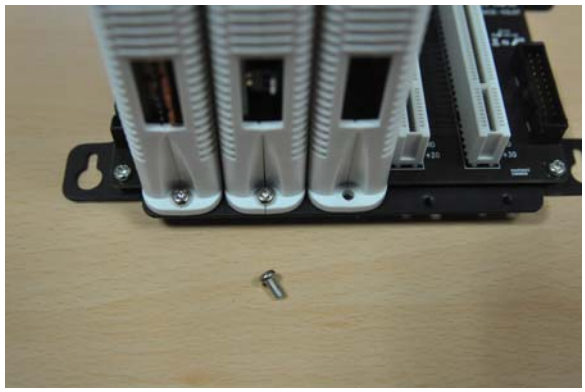
POWER INPUT : 24V 전원을 입력해주는 단자입니다. 둘 중 한 곳에 24V 를 인가하십시오.

주의 : 전원이 인가된 상태에서 모듈을 빼거나 장착하지 마십시오.

DP-CPU500 세부사양	
입력전압	24VDC
전압 입력가능 범위	18 – 28VDC
RTC 용 밧데리 교환주기	10 년
동작온도	0 도~55 도씨
보관온도	-20 도 ~70 도씨

설치

모든 모듈은 나사를 사용해서 슬롯보드와 확실하게 고정해야합니다. 진동 및 충격으로 인해 접촉불량이 나는 것을 막기위해서 입니다. 나사를 고정하지 않은상태로 운전할 경우, 진동으로 인해 예상치못한 결과가 초래될 수 있습니다.

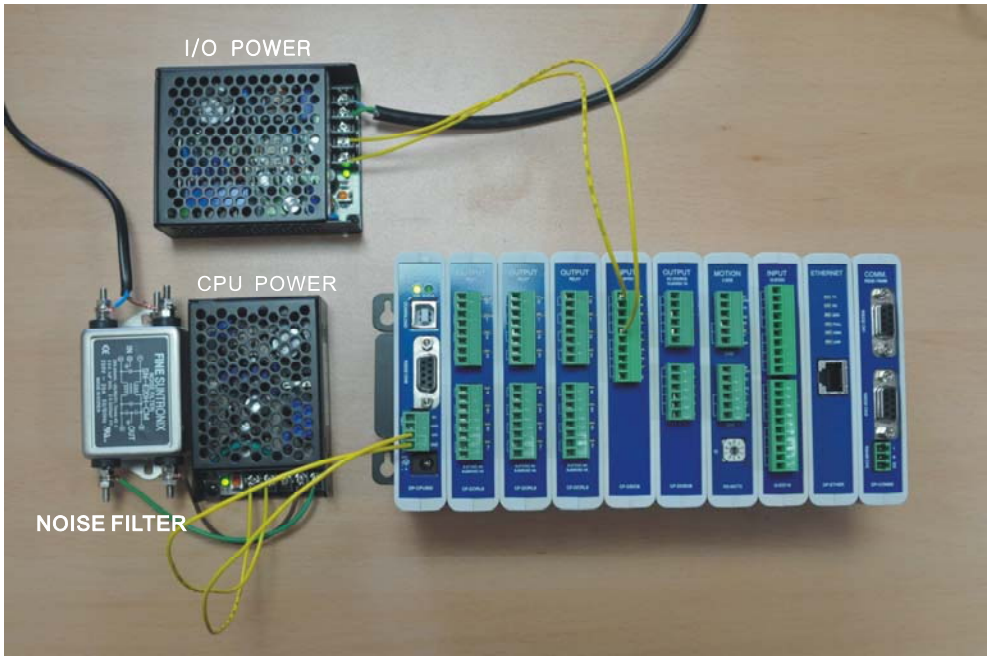


다음과 같은 장소에는 설치를 삼가해주시기 바랍니다.

- 주위온도가 55 도를 넘는 곳
- 주위습도가 30 ~ 65%RH 범위를 넘는 곳, 결로가 맺히는 곳
- 온도의 변화가 심한곳, 이로인해 결로가 발생하는 곳
- 진동 및 충격이 심한장소
- 직접 일광이 닿는 장소
- 히터, 대형저항기등 열이 발생하는 장치 주변
- 트랜스와 같이 고압을 다루는 부품 주변

전원연결

전원은 다음과 같이 2 개의 DC24V 전원공급장치를 이용하시기 바랍니다. 하나는 MOACON 메인과 모듈구동에 필요한 전원이고 다른 하나는 I/O 구동을 위한 전원입니다.

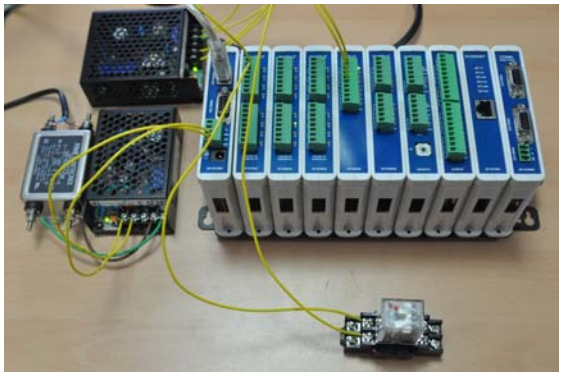


MOACON 콘트롤러 구동에 필요한 제어전원과 I/O 구동에 필요한 전원을 분리시켜, 보다 안정된 동작을 구현하기 위해서입니다. 이렇게 한다면 I/O 전원쪽에 노이즈가 발생하거나, 쇼트가 발생해도 콘트롤러 전원에는 영향을 주지 않습니다.

MOACON 쪽 파워에는 "노이즈 필터"를 통과한 A/C 입력을 연결해주는 것이 좋습니다.

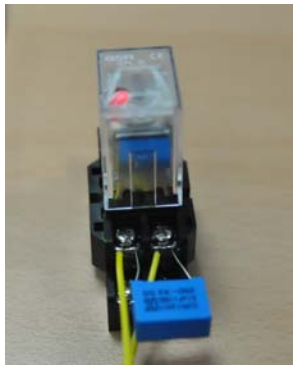
스파크 킬러

릴레이 출력모듈에 용량이 큰제품을 사용하실 때에는 큰부하를 감당할 수 있는 릴레이를 추가하여 사용하시기 바랍니다.



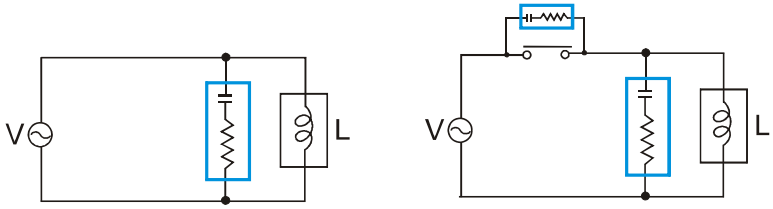
추가된 릴레이가 ON /OFF 할때, 코일에 의한 인덕턴스 때문에 큰 에너지의 펄스 역전압이 발생합니다. 이를 반드시 제거해주어야, 전체 시스템이 안정적으로 동작하게 되며, 릴레이의 수명도 늘어나게 됩니다.

이를 제거하기 위한 "스파크 킬러 (또는 서지킬러)" 라는 제품이 있습니다. 다음 그림처럼 스파크 발생원과 가까운 위치에 병렬로 연결합니다.

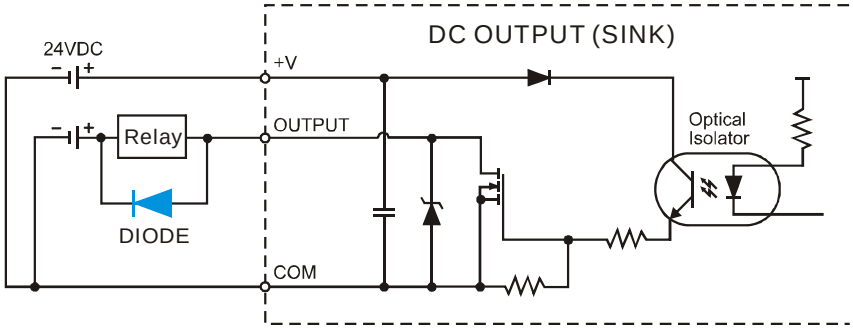


마그네트 접촉기(CONTACTOR) 를 사용할 때에는 아래 그림과 같이 시판제품인 서지 킬러를 반드시 부착하여 주십시오.

교류회로에서의 스파크 킬러 부착위치입니다.



직류회로에서는 스파크 제거를 위해 다이오드를 사용합니다. 모아콘 DC 출력모듈에 릴레이를 연결하는 경우 다이오드를 반드시 연결하여 주십시오. 다이오드는 1N4148 이 적당합니다.



***많은 유저분들이 간과하고 있는 부분입니다. 비교적 적은 비용으로 가장 효과가 좋은 출력 노이즈 제거방법입니다.**

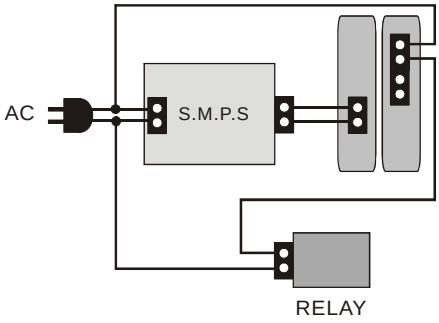
***유도성 부하는 1 초 ON/ 1 초 OFF 보다 더 빠른 간격으로 ON/OFF 하지 마십시오.**

릴레이 전원 배선방법

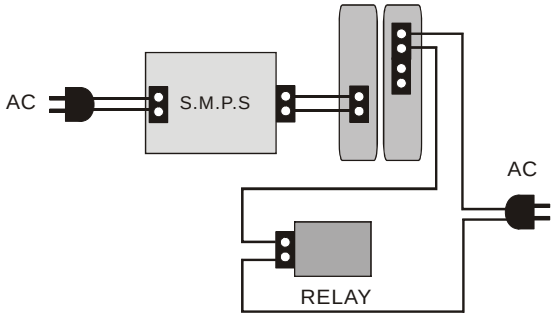
중계 릴레이를 구동하기 위한 전원선은 콘트롤러용 SMPS 전원입력부에서 연결하지 마십시오. 220V 릴레이가 ON 되는 시점에서 발생되는 노이즈가 SMPS 로 전달되어 시스템 운영에 영향을 주게 됩니다.

릴레이 (또는 마그네트 CONTACTOR)용 220V 는 콘트롤러 전원과 별도의 220V 콘센트로부터 연결된 선을 이용하셔야 합니다.

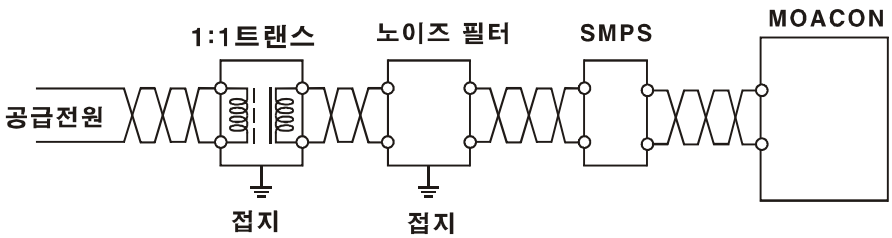
잘못된 사례



좋은 예



가능하면 모아콘의 전원 입력부에 실드 트랜스 (절연 1:1 트랜스)를 사용하시기 바랍니다.

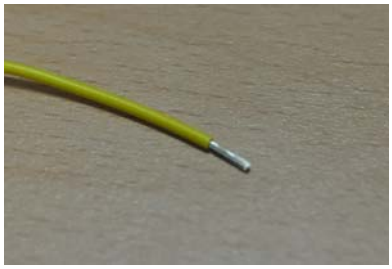


*** 시스템 구성에 있어서, 전원 배선은 가장 중요한 요소입니다. 전원선을 통해 유입되는 노이즈를 막아주지 않으면, 안정적인 동작을 보증할 수 없습니다.**

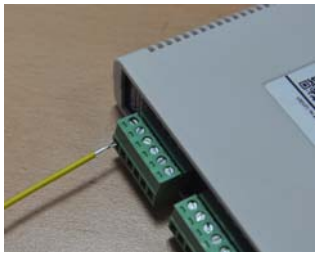
와이어 연결

터미널 블록에 와이어를 연결하는 방법을 설명드리겠습니다. 입출력 신호용 와이어선은 PVC 피복으로 되어 있는 1.8mm ~ 2mm 정도 두께의 선이 적당합니다. (AWG20 규격, www.comfile.co.kr 에서 구입가능)

와이어선의 끝을 0.6 ~ 0.8 mm 정도 탈피시킨뒤, 직접 터미널 블록에 삽입한뒤 드라이버로 꽉 조이시면 됩니다. 와이어선 끝을 납땜하는 것은 오히려 좋지 않습니다.



좋은예



나쁜예

이때, 위 사진처럼 벗겨진부분이 터미널 블록 바깥으로 노출되지 않아야 됩니다.
전원선은 이보다 두꺼운 AWG16 ~ AWG12 사이 규격의 와이어선을 사용하시기 바랍니다.

제어반내 배선방법

노이즈를 줄이기 위해서는 배선이 무엇보다도 중요합니다. 다음과 같은 규칙을 지켜준다면 노이즈를 줄일수 있습니다.

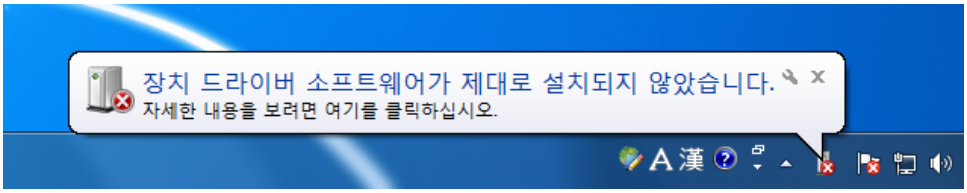
1. **전원선은 가능한 한 굵은 선**을 사용하고, 트위스트 할 것
2. 트랜스 2 차측은 트위스트로 하고, 모아콘과 최단거리로 배선할 것
3. SMPS 의 접지는 반드시 연결할 것.
4. **AC 입출력 선과 DC 입출력선을 하나의 덕트안에 혼용하지 말 것**
5. 신호선과 동력선은 별도의 덕트에 설치하고, 가능한한 20cm 이상 떨어뜨릴 것
6. 입력 신호선과 출력신호선도 가능하면 따로따로 덕트를 통해 배선할 것
7. **전자 계폐기, 접촉기, 파워 릴레이등은 가능한한 콘트롤러와 멀리 배치할 것**

특히, 전원공급장치의 용량을 확인하고, 그 사용범위안에서 부하를 연결하는 것이 중요합니다. 전원공급장치에 과부하가 발생하면, 발열이 되면서 결국은 시스템이 멈추게 됩니다.

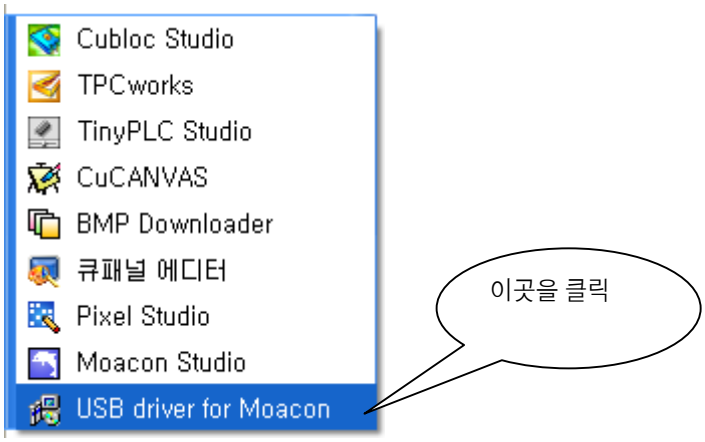
여름철 제어반 내부의 온도상승으로 인해, 시스템이 오동작하는 것을 막기위해, 가능한한 팬을 설치하여 공기가 흐르도록 해주는 것이 좋습니다.

USB 드라이버 설치

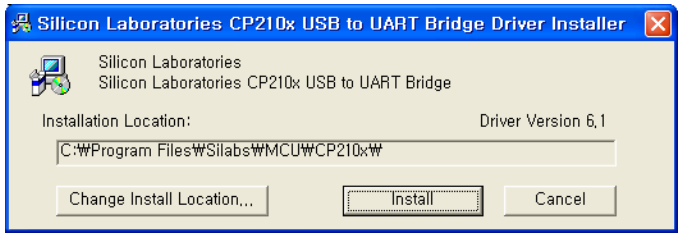
USB 케이블을 모아콘 CPU 모듈에 연결하십시오. USB 드라이버가 설치되어 있지 않은 PC 에서는 다음과 같은 멧시지가 화면하단에 표시됩니다.



이 멧세지가 표시되었다면, USB 케이블을 잠시 뽑은후, <시작>→ <모든프로그램>→ <Comfile Tools>에 있는 "USB driver for Moacon"을 실행하십시오.

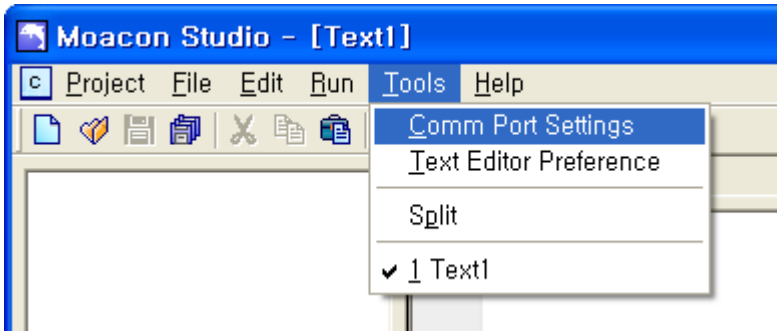


다음과 같은 창이 표시됩니다.

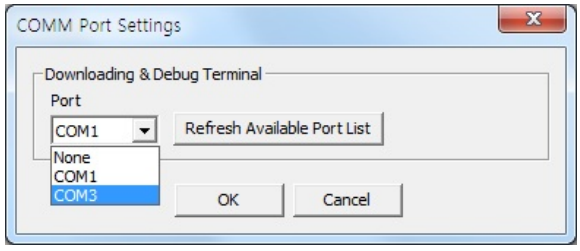


<Install>버튼을 누르시고 지시에 따라서 인스톨을 진행하십시오. 설치가 모두 끝나고 난뒤, USB 케이블을 다시 연결하면, 화면 하단에 성공적으로 설치되었다는 멧세지가 표시됩니다.

모아콘 스튜디오를 실행하십시오. 모아콘 스튜디오의 Tools 메뉴에서 Comm Port Settings 을 클릭하세요.

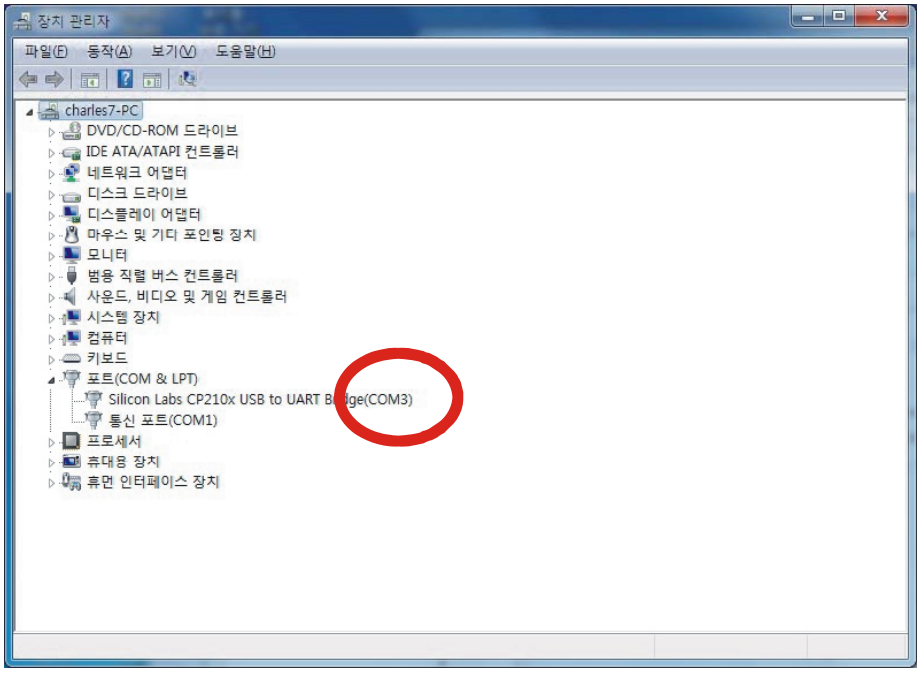


다음과 같은 다이얼로그 박스가 표시됩니다.



Port 드롭다운 리스트를 펼쳐보면, 사용가능한 COM 포트가 모두 표시됩니다. 이 중 여러분이 방금 USB 드라이버를 설치한 COM 포트로 선택하십시오.

만약 몇번 COM 포트에 USB 드라이버가 할당되어 있는지 모르신다면, 제어판의 "시스템"→ "하드웨어"의 장치관리자에서 "포트(COM 및 LPT)" 에서 확인하실 수 있습니다.



여러분이 사용하시는 PC 에서는 다른 COM 포트번호에 할당되어질 수 있습니다.

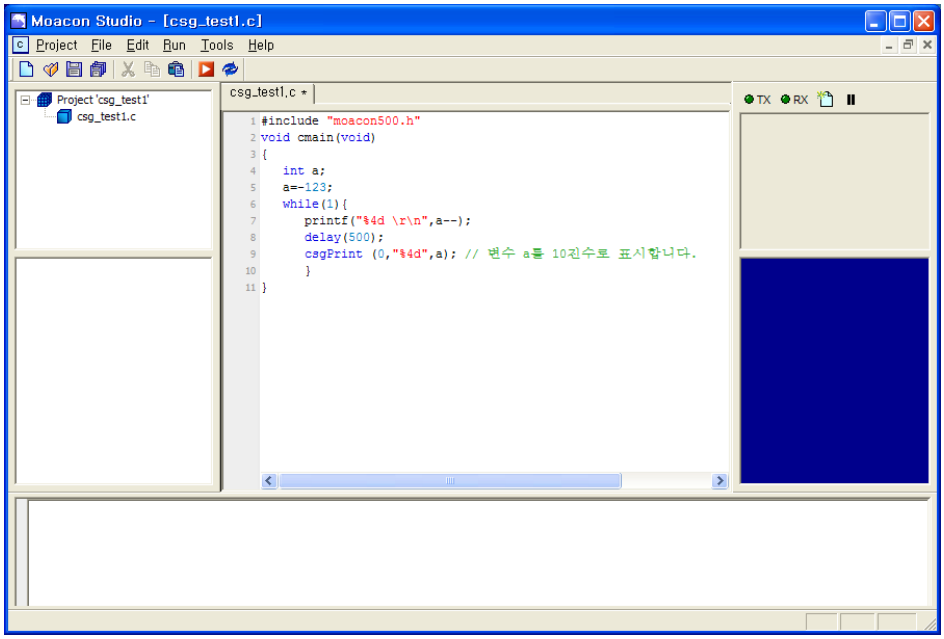
Silicon Labs CP210x USB to UART Bridge 가 표시된 COM 포트를 모아콘스튜디오에서 선택해주어야 합니다.

제 3 장
개발환경

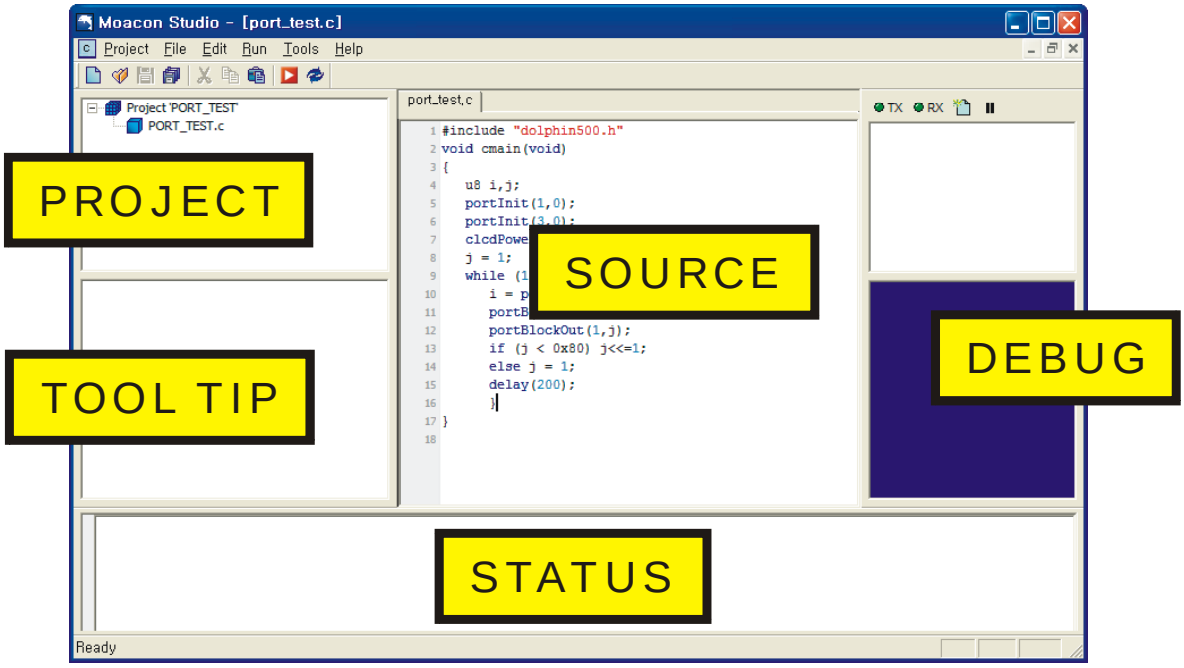
MOACON STUDIO

MOACON 를 사용하기 위해서는 MOACON STUDIO 를 PC 에 설치해야 합니다.
www.comfile.co.kr 에서 "MOACON 자료실" 에서 MOACON STUDIO 최신 버전을 다운로드 받으세요.

설치를 마치고 나면 바탕화면 혹은 "시작→ 모든프로그램→Comfile Tools" 아래에 MOACON STUDIO 을 실행시키십시오.



MOACON STUDIO 의 화면구성은 아래와 같습니다.



MOACON 에서 프로젝트를 기본단위로 해서 프로그램을 작성합니다. 하나의 프로젝트는 여러 개의 소스를 포함할 수 있습니다.

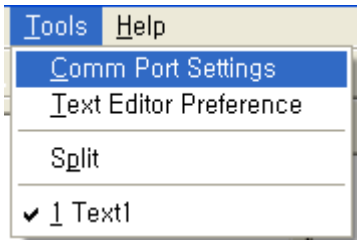
"PROJECT"라고 표시된 곳에 프로젝트 상황이 표시됩니다. "SOURCE"라고 표시된곳에 소스 프로그램을 입력합니다. 화면 상단의 빨간색 실행 아이콘을 클릭하면 <소스저장 → 컴파일 → 다운로드 → 실행>이 진행되는데, 이때 "STATUS "라고 표시된 곳에 컴파일 결과가 표시됩니다.

"DEBUG"라고 표시된 곳이 "디버그 터미널"입니다. 이곳을 통해 실행중 변수값등을 모니터링 할 수 있습니다.

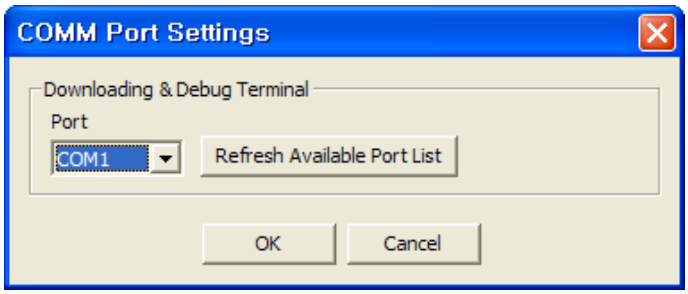
"TOOL TIP"이라고 표시된 곳에서는 라이브러리의 사용법이 요약된 형태로 표시되어, 사용설명서를 일일히 찾는 불편함을 덜어줍니다.

COMM 포트선택

우선 MOACON STUDIO 에서 COMM PORT 를 설정해주어야 합니다. 앞서 USB 드라이버 설치에 의해 추가된 COMM 포트를 설정하여 주십시오.



TOOLS 메뉴에서 Comm Port Settings 를 선택하십시오.

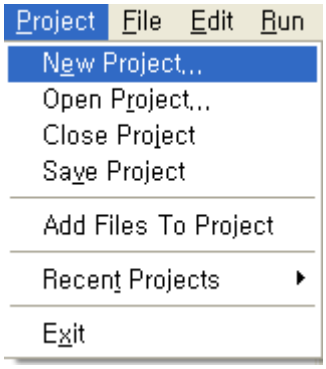


이곳에서 다운로드를 위한 포트를 선택하면 됩니다.

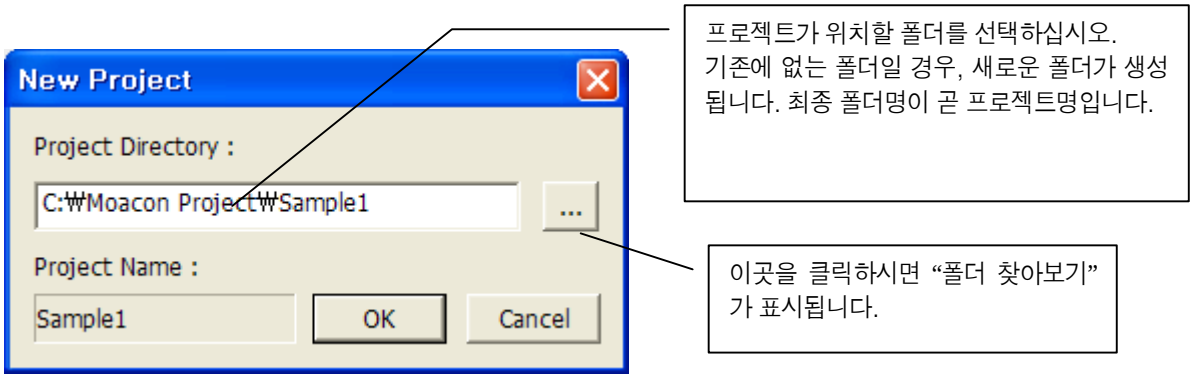
프로젝트 만들기

프로젝트란 여러 개의 소스파일 집합입니다. 소스파일 하나로도 프로젝트를 구성할 수 있습니다만, 프로그램이 길어지면, 여러 개의 파일로 나누어 작성하는 것이 훨씬 효율적인 방법입니다. 이를 위해서 "프로젝트"라는 관리방법이 필요한 것입니다.

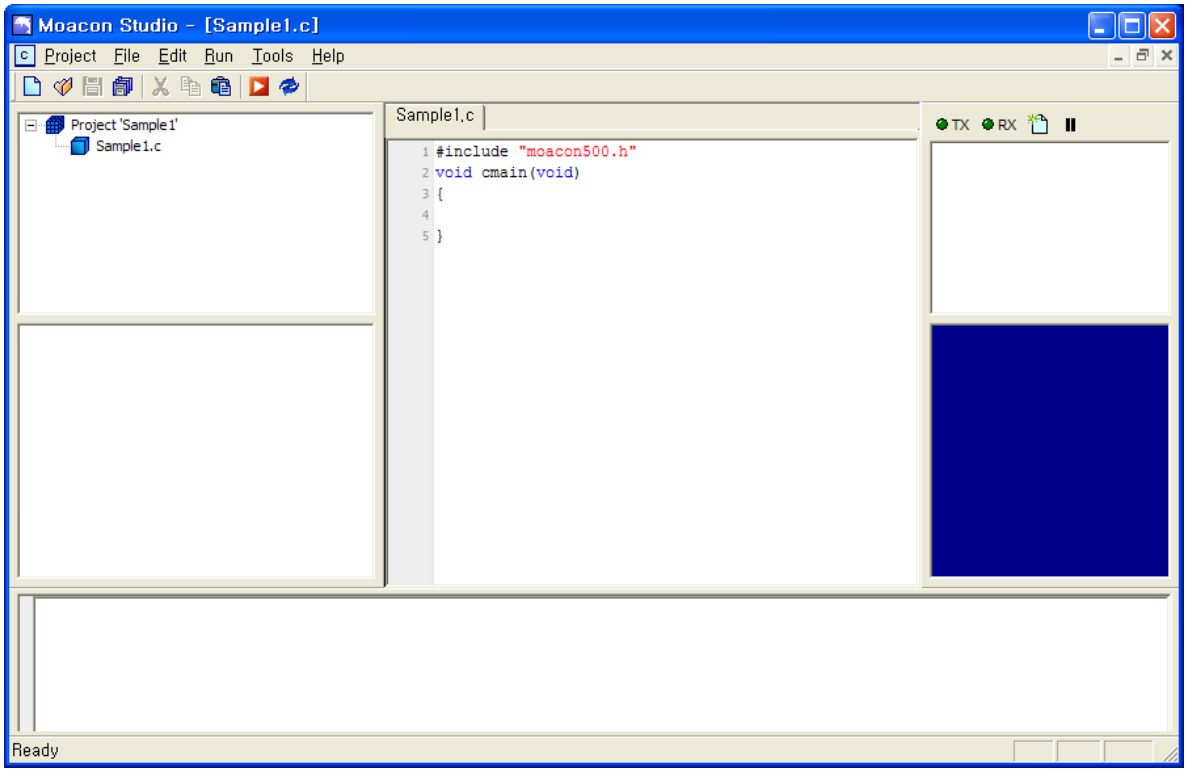
MOACON STUDIO 에서 프로젝트를 만드려면 먼저 Project 메뉴의 New Project 메뉴를 선택하십시오.



그러면 다음과 같은 박스가 표시됩니다.



OK 를 누르면 화면은 다음과 같은 상태가 됩니다.



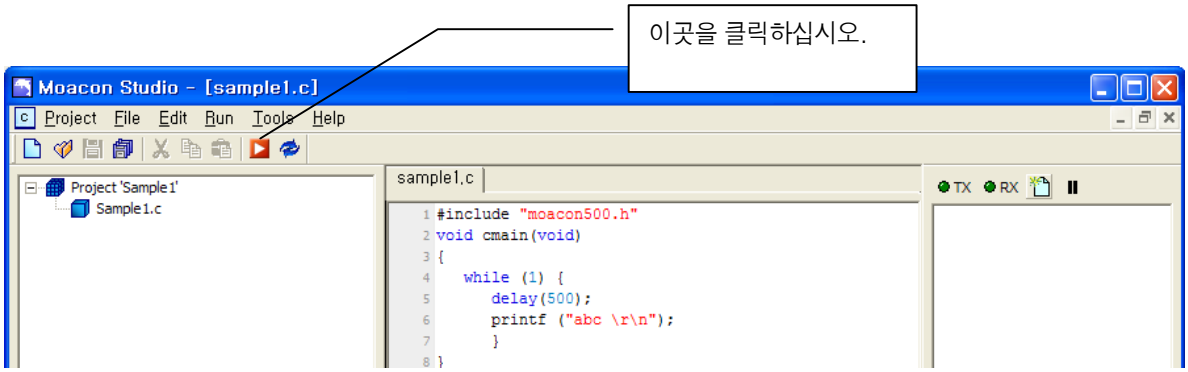
프로젝트와 최초 소스 파일 (프로젝트명.c)이 생성됩니다. 최초 소스파일에는 기본적인 소스가 들어있습니다.

최초 프로그램 다운로드 및 실행

아주 간단한 프로그램을 작성하여 다운로드 해보겠습니다. 앞서 만든 프로젝트를 그대로 사용하도록 하겠습니다.

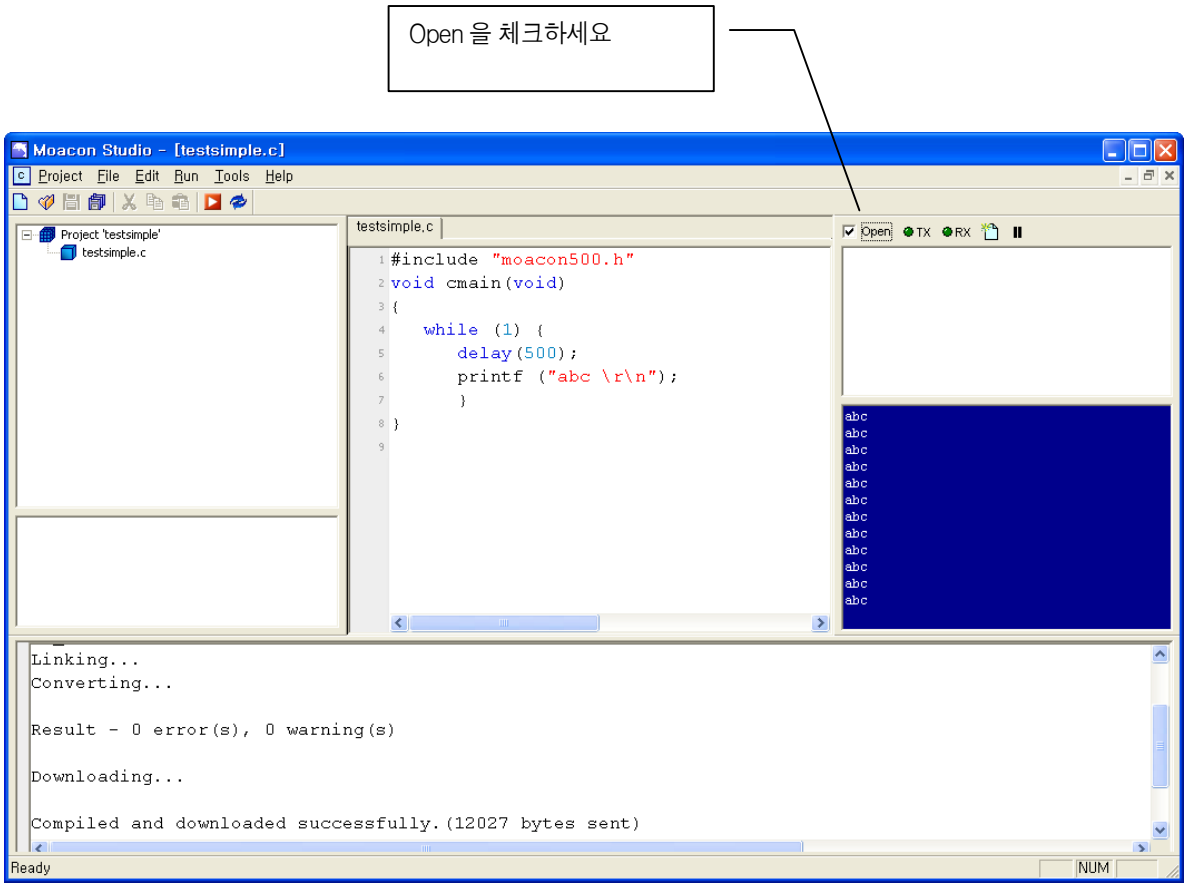
```
#include "moacon500.h"
void cmain(void)
{
  while (1) {
        delay(500);
        printf ("abc \r\n");
        }
}
```

디버그터미널에 0.5 초 간격으로 abc 를 표시하는 프로그램입니다. 실행버튼을 누르기 전에, USB 케이블 접속상태와 전원상태를 확인하십시오.



자 이제, 상단 툴바의  "RUN" 아이콘을 클릭하거나, 단축기 CTRL+R 을 누르면 "컴파일 → 다운로드 → 실행" 이 차례대로 진행됩니다.

이 과정에서 문제가 발생되지 않는되면, 프로그램은 이상없이 수행됩니다.



디버그 창위에 있는 Open 체크박스를 체크표시가 되도록 하십시오. 이렇게 하면 Debug 창이 Comm 포트와 연결됩니다.

실행결과는 MOACON STUDIO 오른쪽에 있는 "디버그 터미널"에서 보실 수 있습니다. 0.5 초 간격으로 abc 라는 멧세지가 표시된다면 성공입니다.

잠깐만…

일반적인 C 언어와는 다르게 MOACON 에서는 main 함수가 아닌 cmain 함수로부터 프로그램이 시작됩니다.

Printf 함수는 실행결과를 Debug 터미널 창에 표시해줍니다. Printf("abc \r\n"); 에서 \r\n 은 복귀개행 (carrage return) 을 실행해주는 특수문자입니다. 표시위치를 다음줄의 맨앞칸으로 이동시켜줍니다. (역슬레쉬는 키보드에 ₩으로 표기되어 있으므로 주의바랍니다. 슬레쉬가 아닙니다.)

실행이 안될 경우

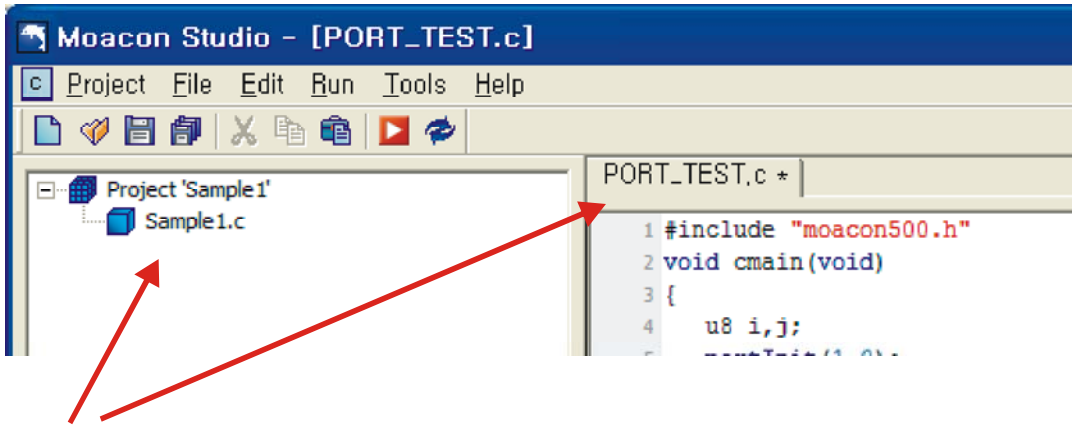
앞서 프로그램이 실행이 안될 경우 다음과 같은 사항들을 체크해보세요.

1. MOACON CPU 모듈에 전원이 제대로 인가 되어있습니까? CPU 모듈에 있는 Power LED 에 ON 되어 있는지 확인하세요.

2. COMM 포트셋업을 했습니까? USB 드라이버가 할당한 COM 포트를 선택해주어야 합니다.

3. 컴파일 결과에 오류가 발생되었습니까? 오타로 인해 컴파일 결과 에러가 발생되었다면 다운로드되지 않습니다. 문제를 수정하신뒤 다시 실행시켜보세요.

4. 프로젝트의 파일구성을 확인해보십시오. 엉뚱한 파일이 오픈되어 있지 않습니까?

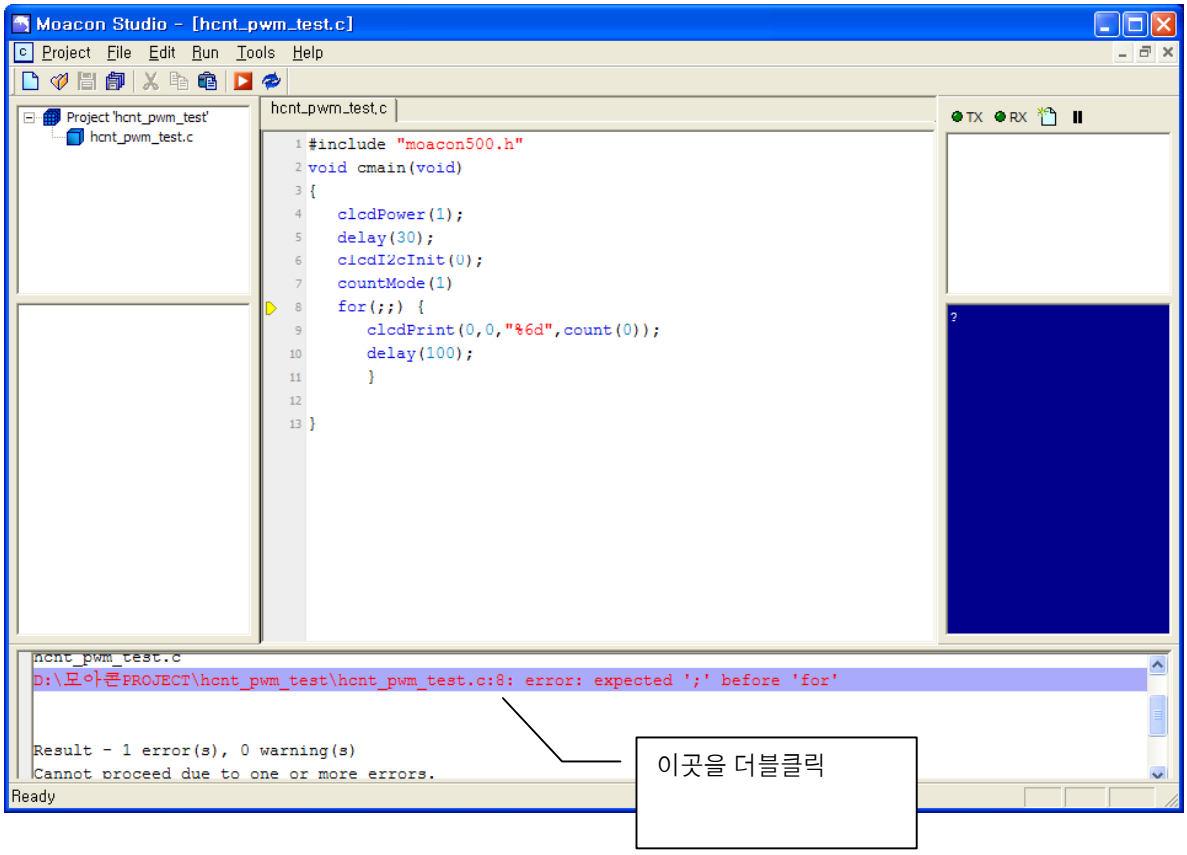


유저는 화면에 오픈된 파일을 고치고 있는데, 컴파일 및 다운로드 되는 파일은 프로젝트에 연결된 파일만 다운로드되는 상황이 발생할 수 있습니다. 주로 프로젝트 개념을 이해하지 못하고 원하는 파일만 열어서 수정한뒤, 다운로드 하는 분들이 자주 격는 일입니다.

오류 지점 알아내기

만약, 소스중 문법에러가 있다면 Status 창에 빨간색으로 표시됩니다. 에러가 발생된 지점을 알고 싶다면, 해당 에러멧세지를 더블클릭 하시면 됩니다.

에러가 발생된 지점으로 커서를 이동시켜 줍니다.



디바이스 선언

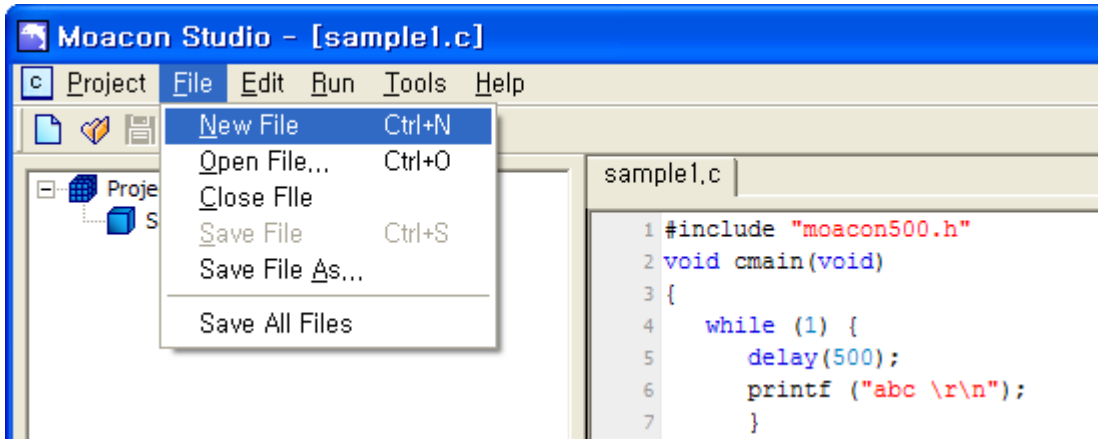
MOACON CPU 모듈중 어떤 모델을 대상으로 하는 지, 소스의 가장 첫머리에 #include 문을 사용하여 적어주어야 합니다. 아래 예제 프로그램은 DP-CPU 500 모듈을 사용하고 있으므로 #include “moacon500.h” 문장을 삽입하였습니다.

```
#include "moacon500.h"
void cmain(void)
{
  while (1) {
        delay(500);
        printf ("abc \r\n");
        }
}
```

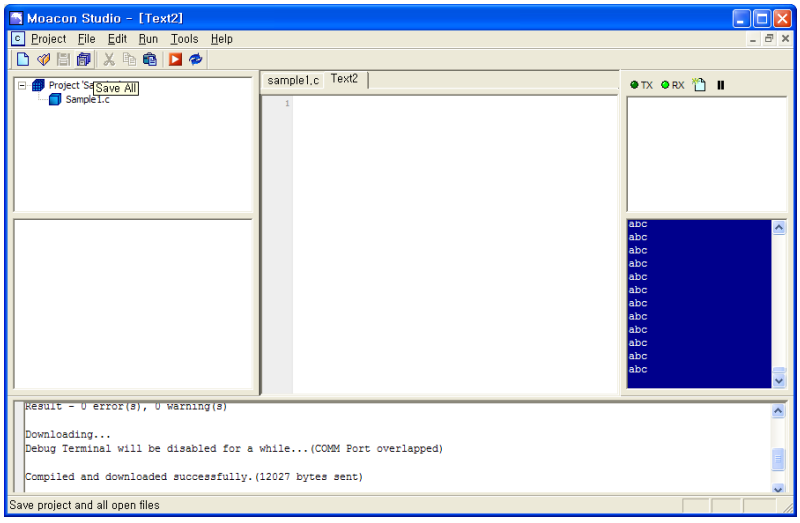
하나의 프로젝트 아래 여러 개의 소스가 있다면, 모든 소스의 첫 부분에 #include "moacon500.h"를 써주어야 합니다.

프로젝트에 소스추가

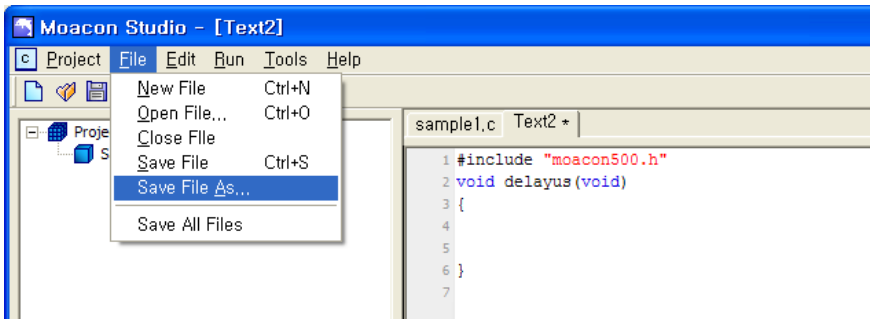
하나의 프로젝트에서 소스를 추가하는 방법을 설명하겠습니다. File 메뉴에서 New File 을 선택하십시오.



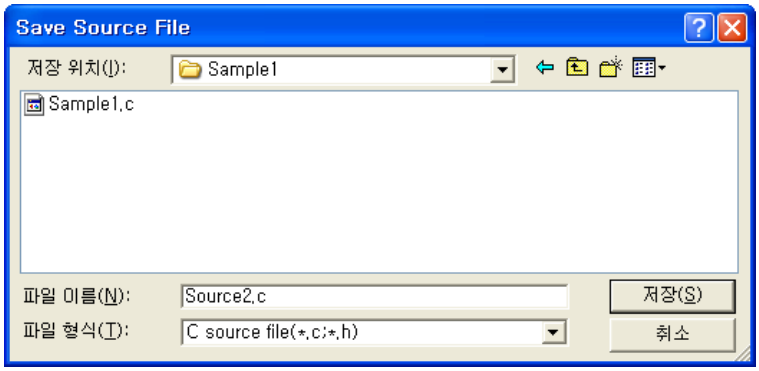
다음과 같이 text2 라는 새로운 소스탭이 생성됩니다. text 뒤에번호는 상황에 따라 바뀔수 있습니다.



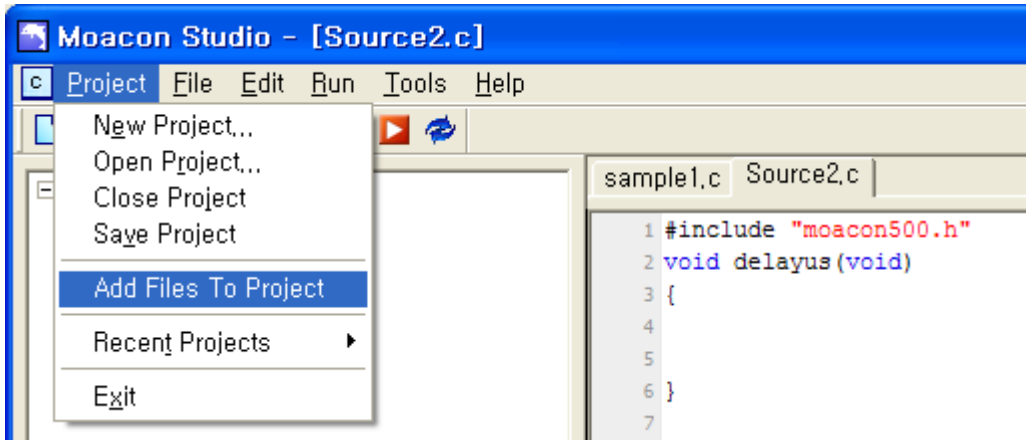
이곳에 필요한 소스를 작성하신뒤, File 메뉴의 Save As… 를 선택하십시오.



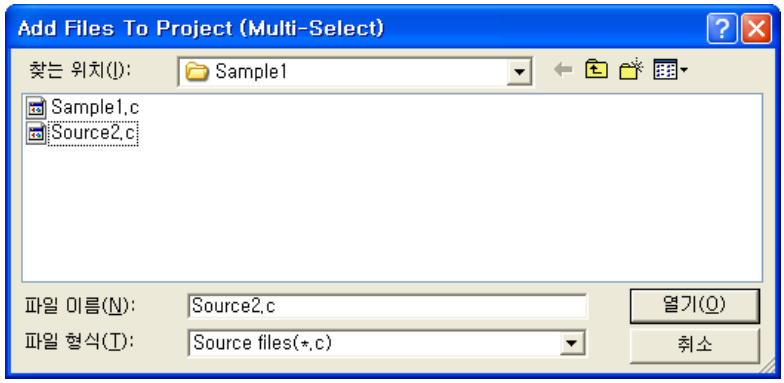
먼저 프로젝트가 생성되었던 폴더를 찾아가, 다른이름 (여기에서는 Source2.c)으로 저장합니다.



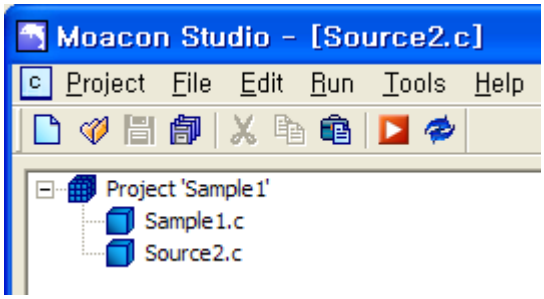
그리고, Project 메뉴에 있는 Add Files To Project 를 선택하십시오.



여기에서 방금전에 저장했던 파일을 선택하시면 됩니다.



프로젝트 창으로 보면 Source2.c 가 프로젝트안에 포함되어 있는 것을 보실 수 있습니다.



 "RUN" 아이콘을 클릭하면 프로젝트아래 있는 모든 소스파일이 컴파일 된 뒤, 링크되어 다운로드됩니다.

소스 프로그램이 길어진다면, 위와 같은 방법으로 여러 개의 파일로 소스를 나누어 작성하시는 것이 좋습니다.

변수형

MOACON C 언어에서 사용하는 변수형은 다음과 같습니다.

char	부호있는 8 비트 정수
unsigned char	부호없는 8 비트 정수
short	부호있는 16 비트 정수
unsigned short	부호없는 16 비트 정수
int	부호있는 32 비트 정수
unsigned int	부호없는 32 비트 정수
long	부호있는 32 비트 정수
unsigned long	부호없는 32 비트 정수
long long	부호있는 64 비트 정수
float	32 비트 실수 (IEEE 단일 정밀도)
double	64 비트 실수 (IEEE 두배 정밀도)

다른 C 언어와 다소 차이점이 있을 수 있으므로, 눈여겨 봐두시기 바랍니다. MOACON 에서 int 는 long 과 동일하게 취급됩니다.

다소 긴 변수형 명칭을 좀더 간편하게 사용할 수 있도록 , 다음과 같이 MOACON OS 에서 사전정의 해두었습니다.

```
#define   u8  unsigned char         //8 비트 크기의 부호없는 정수형
#define   u16  unsigned short       //16 비트 크기의 부호없는 정수형
#define   u32  unsigned long        //32 비트 크기의 부호없는 정수형
```

유저여러분은 특별한 선언없이 u8, u16, u32 와 같은 요약된 변수형을 소스프로그램에서 곧바로 사용하실 수 있습니다.

```
void cmain(void)
{
  u8 i;       // 변수 I 를 8 비트 부호없는 정수형으로 선언
  u32 li;     // 변수 li 를 32 비트 부호없는 정수형으로 선언
```

예약어

변수명을 작명하실 때, 유의해야될 사항이 있습니다. 이미 MOACON 의 라이브러리 함수로 사용중인 명칭이나, C 언어의 예약어등은 변수 또는 함수명으로 사용하실 수 없습니다.

```
void cmain(void)
{
  u8 portInit;     // 라이브러리명으로 사용중인 이름입니다.
  u32 for;         // C 언어의 예약어입니다.
```

컴파일 도중 이름과 관련된 에러가 발생되면, 다른 이름으로 바꾸어 사용하시기 바랍니다.

16 진수

C 언어에서 16 진수를 표기할 때에는 맨앞에 0X 를 붙여줍니다. 예를들어 16 진수 FF 는 0XFF 로 표기합니다.

BASIC 등 다른언어와 16 진수 표기법이 다르므로 주의하셔야합니다.

```
void cmain(void)
{
  u8 valueA = 0x123 // 16 진수 123 을 변수 valueA 에 할당합니다.
```

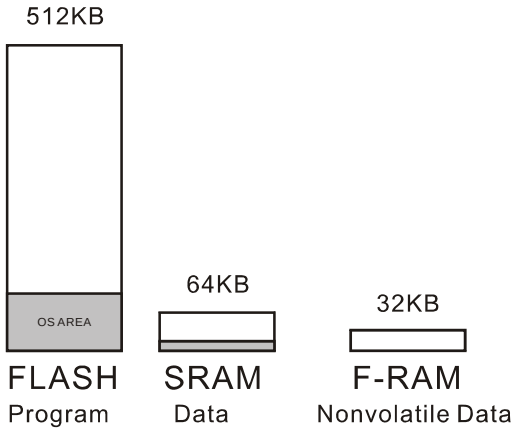
대,소문자

C 언어는 대,소문자를 엄격히 구분합니다. comfile 과 comFile 은 서로 다르게 인식합니다. 따라서 소스를 작성하실 때 대소문자를 유의해서 입력하여 주십시오.

메모리

MOACON 에 있는 512K BYTE 의 플레쉬 메모리는 프로그램을 저장하는 곳입니다. 유저가 512KB 전부를 사용할 수 있는 것은 아닙니다. MOACON OS 가 일부분을 사용하게 되는데 대략 20KB 내외입니다. OS 의 용량은 버전에 따라서 변동될 수 있습니다.

마찬가지로 데이터 메모리도 대략 10KB 정도 내외에서 OS 가 사용하고 있으며, 어떤 라이브러리를 사용하였느냐에 따라 사용량이 변화됩니다.



메모리는 C 컴파일러에 의해서 자동 관리되고 있으므로, 유저여러분께서 특별히 신경쓰실 부분은 없습니다. 용량을 초과하는 코드를 작성할 경우 Out of memory 에러가 발생됩니다.

MOACON 에는 비휘발성 메모리인 FRAM 도 내장되어 있으며, FRAM 관련 라이브러리를 사용해서 억세스합니다. FRAM 저장된 데이터는 전원이 없어도 계속 유지됩니다.

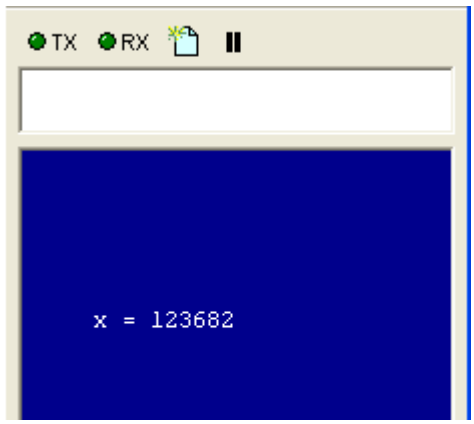
MOACON 에서는 별도의 밧데리 백업기능을 지원하지 않으므로, 정전시에도 유지되어야 하는 값은 FRAM 에 보관해두었다가, 파워온 초기화 부분에서 읽어오는 방법으로 사용하시기 바랍니다.

Debug 터미널

MOACON 에서는 기본적인 디버깅 방법으로 디버그 터미널을 사용합니다. 즉, 프로그램 실행중 변수값의 변화상태를 디버그터미널에 표시하는 방법으로 디버깅을 수행합니다.

소스프로그램중 printf 함수를 삽입하면, 실행 결과는 MOACON STUDIO 에 있는 디버그 터미널에 표시됩니다.

```
#include "moacon500.h"
void cmain(void)
{
   int x=123678;
   debugCls();
   while(1) {
         delay(500);
         debugLocate(5,5);
         printf ("value x = %d",x++);
   }
}
```



디버그 터미널을 콘트롤하기 위한 추가적인 함수도 있습니다.

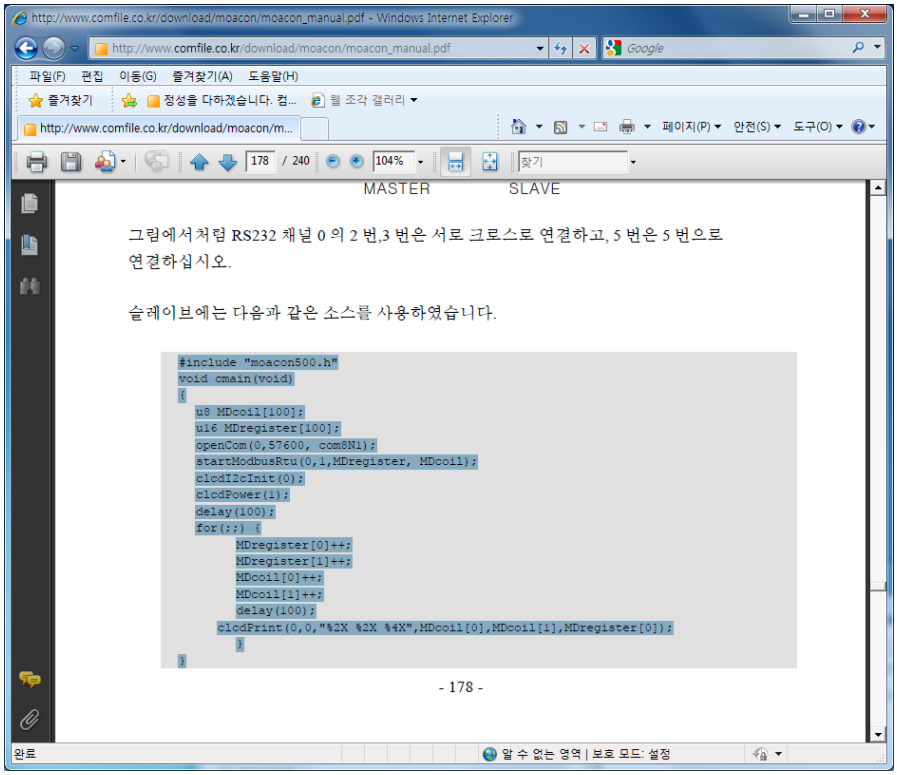
debugCls( );	// 디버그 터미널을 클리어합니다.
debugLocate(x, y);	// 표시위치를 지정합니다.(x, y 는 0 부터 시작)
debugPut(ch);	// 디버그 터미널로 1 바이트(ch)를 전송합니다.

printf 함수의 구체적인 사용법은 본 사용설명서의 뒷부분 "부록 1. C 언어"를 참조하시기 바랍니다.

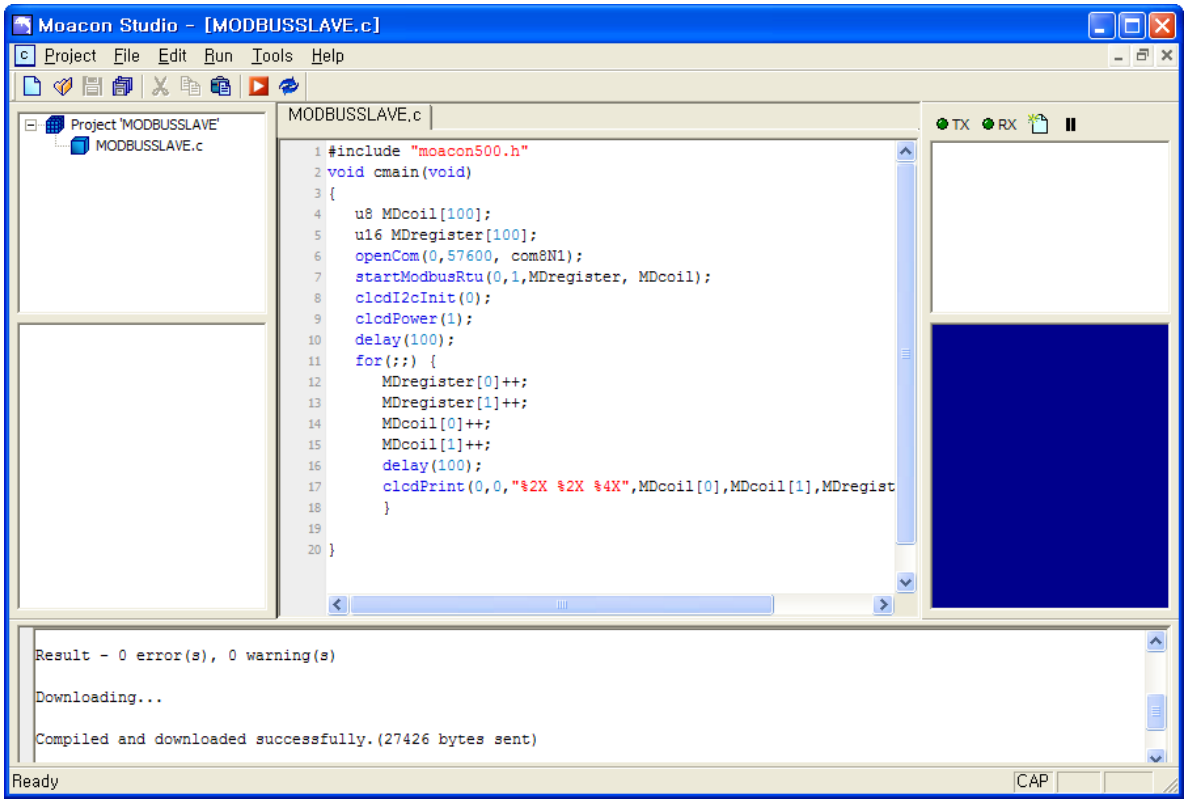
TIPS

본 메뉴얼에는 다양한 예제 소스가 수록되어 있습니다. 이 예제 소스를 여러분의 MOACON Studio 로 카피하여 실행시켜 보실 수 있습니다.

본 메뉴얼을 브라우저에서 오픈한 상태로 텍스트의 일부분을 선택 및 복사가 가능합니다. 원하시는 소스파일을 선택한뒤 ctrl + C (카피)하십시오. 그리고 여러분의 MOACON Studio 에 ctrl+V (붙여넣기)하시면 됩니다.



메뉴얼에 기록된 소스가 MOACON Studio 에 그대로 복사되었습니다.

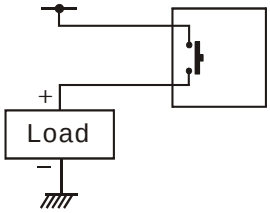
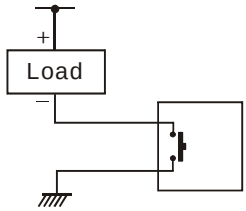


제 4 장

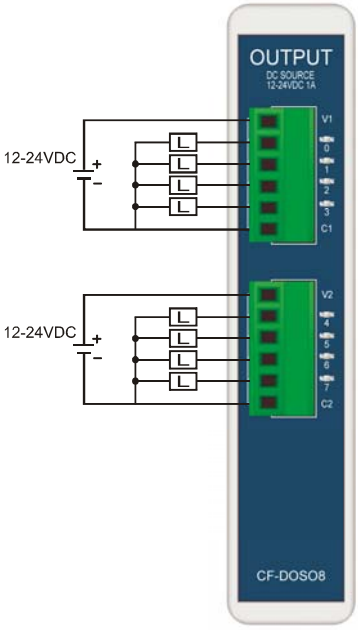
디지털 I/O 모듈

디지털 출력 모듈

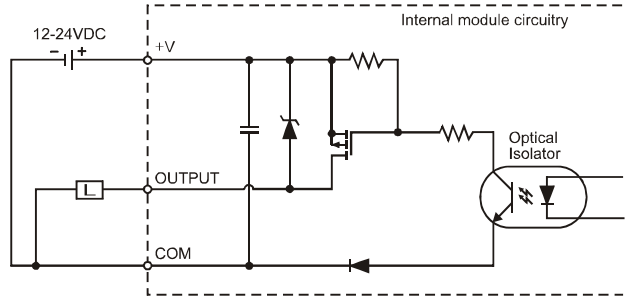
두종류의 DC 출력모듈이 있습니다.

SOURCE 출력	SINK 출력
턴 온시 전원의 플러스측이 ON 됩니다. 부하의 다른 한쪽 끝은 전원의 마이너스측과 항상 연결되어 있습니다.	턴 온시 전원의 마이너스측이 ON 됩니다. 부하의 다른 한쪽 끝은 전원의 플러스측과 항상 연결되어 있습니다.
	

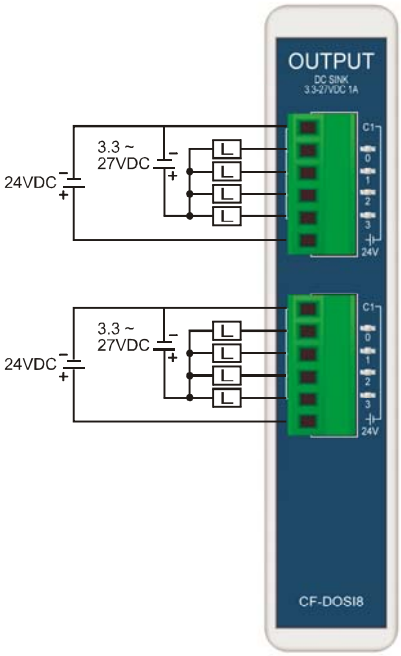
DC 소스 출력 8 점 모듈



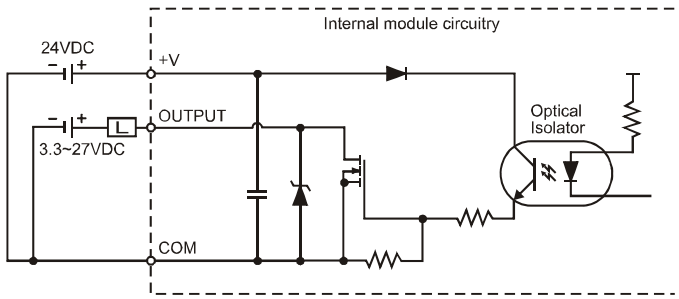
CF-DOSO8 출력 사양	
점수	8 점
오퍼레이팅 전압	12 – 24VDC
출력전압 범위	10 – 30VDC
최대 출력 전류	1A / 점, 4A / COMMON
최소 출력 전류	0.5mA
최대 On/Off 가능 주기	1KHz (초당 1000 번)
상태 LED	턴온시 점등
COMMON 단자	2 점 (분리되어 있음)



DC 싱크 출력 8 점 모듈



CF-DOSI8 출력 사양	
점수	8 점
오퍼레이팅 전압	3.3 – 27 VDC
출력전압 범위	3 – 30VDC
최대 출력 전류	1A / 점, 4A / COMMON
최소 출력 전류	0.5mA
최대 On/Off 가능 주기	1KHz (초당 1000 번)
상태 LED	턴온시 점등
COMMON 단자	2 점 (분리되어 있음)

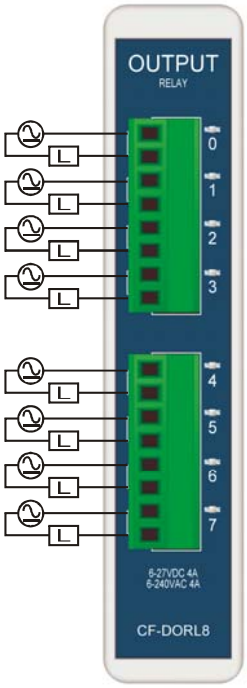


DC-SINK 모듈은 2 개의 전원을 사용합니다. DC24V 는 내부 FET 구동을 위한 전원입니다.

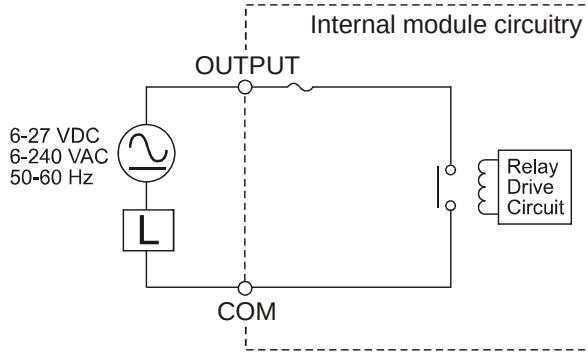
부하에 걸리는 전압은 3.3V~27VDC 까지 사용가능합니다. DC-SOURCE 출력 모듈에 비해서 사용가능한 전압의 범위가 넓다는 장점이 있습니다.

DC 출력모듈은 최대 1KHz 의 펄스를 출력할 수 있습니다. 초당 1000 번 ON / OFF 가능하다는 뜻입니다.

RELAY 출력 8 점 모듈



CF-DORL8 출력 사양	
점수	8 점
오퍼레이팅 전압	6-27VDC / 6-240VAC
출력전압 범위	5-30VDC / 5-264 VAC
최대 전류	4 A / 점, 4A / COMMON
최소 전류	100 mA @ 5VDC
최대 On/Off 가능 주기	25Hz (초당 25 번)
상태 LED	접점이 On 되었을때 점등
COMMON 단자	8 점 (개별 COMMON)



채널당 4A 의 부하를 on/off 할 수 있는 릴레이 출력 모듈입니다. AC 240V 까지 사용가능합니다. 4A 보다 큰 용량의 부하를 구동할 경우에는 반드시 고용량의 릴레이 또는 접촉기를 추가로 연결하여 사용하십시오.

동작주파수가 높은 (On / Off 가 빈번히 일어나는)경우에는 DC 출력모듈을 사용하시는 편이 좋습니다. 릴레이 접점은 기계식이므로 장시간 사용했을 경우 마모되므로, 교체해주어야 합니다. (교체는 모듈단위로만 됩니다. 릴레이 한 개만 교체할 수 없습니다.)

디지털 출력 모듈관련 라이브러리

portInit

void portInit (u8 portBlockNumber, u8 mode)

portBlockNumber : 포트 블록 번호 (0 ~ 5)

mode : 0 또는 1 (0=Output, 1=Input)

모아콘의 기본 DIO 포트는 입출력 설정을 바꿀 수 있습니다. 최초 파워온 상태에서 모두 입력상태로 되어 있습니다. 따라서 출력 모듈을 사용하기 위해서는 해당 블록을 출력상태로 바꾸어야 합니다.

모아콘에서는 블록단위로 입출력상태를 바꿀 수 있습니다. 하나의 블록에 8 개의 I/O 포인트가 포함되어 있습니다.

블록번호	0 블록 (+0)	1 블록(+10)	2 블록(+20)	3 블록(+30)	4 블록(+40)	5 블록(+50)
포트 (10 진)	0 부터 7	10 부터 17	20 부터 27	30 부터 37	40 부터 47	50 부터 57

mode 를 1 로 하면 해당 블록이 입력상태가 됩니다. 0 으로 하면 해당블록이 출력상태가 됩니다. Output 의 첫글자인 O 와 비슷한 0 은 출력, Input 의 첫글자 I 와 비슷한 1 은 입력으로 기억하시면 쉽습니다.

```
portInit(0,1);      // 0 번블록 (0-7)을 입력(Input)상태로 만듭니다.
portInit(1,0);      // 1 번블록 (10-17)을 출력(Output)상태로 만듭니다.
```

위 사진처럼 3 개의 입력모듈과 3 개의 출력모듈이 사용되었다면 다음과 같이 프로그램 하십시오.

```
portInit(0,0);     // 0 번블록을 출력(Output)상태로 만듭니다.
portInit(1,0);     // 1 번블록을 출력(Output)상태로 만듭니다.
portInit(2,0);     // 2 번블록을 출력(Output)상태로 만듭니다.
portInit(3,1);     // 3 번블록을 입력(Input)상태로 만듭니다.
portInit(4,1);     // 4 번블록을 입력(Input)상태로 만듭니다.
portInit(5,1);     // 5 번블록을 입력(Input)상태로 만듭니다.
```

파워온 후 모든 포트가 입력상태이므로 3,4,5 번 블록은 따로 정의하지 않으셔도 좋습니다.

```
portInit(0,0);     // 0 번블록을 출력(Output)상태로 만듭니다.
portInit(1,0);     // 1 번블록을 출력(Output)상태로 만듭니다.
portInit(2,0);     // 2 번블록을 출력(Output)상태로 만듭니다.
```

portOut

void portOut (u16 portNumber, u8 value)

portNumber : 포트번호 (0 ~ 57, 8 이나 9 로 끝나는 수는 사용 불가)
value : 0 또는 1

portNumber 로 지정된 포트를 On 또는 Off 상태로 만듭니다. Value 가 1 이면 On 이 되고, 0 이면 Off 가 됩니다. 해당포트는 사전에 출력모드로 설정되어 있어야 합니다.

```
portOut(20,1);      // 20 번 포트를 On 상태로 만듭니다.
portOut(20,0);      // 20 번 포트를 Off 상태로 만듭니다.
```

portBlockOut

void portBlockOut (u8 portBlockNumber, u8 value)

portBlockNumber : 포트 블록 번호 (0 부터 5 사이값)
value : 0 부터 255 사이의 값

portBlockNumber 로 지정된 블록에 출력포트 8 점을 한꺼번에 변경하는 함수입니다. Value 에 있는 1 BYTE 값을 해당 블록으로 출력합니다.

블록번호	0 블록 (+0)	1 블록(+10)	2 블록(+20)	3 블록(+30)	4 블록(+40)	5 블록(+50)
포트 (10 진)	0 부터 7	10 부터 17	20 부터 27	30 부터 37	40 부터 47	50 부터 57

Bit0 이 낮은번호, Bit7 이 높은번호의 I/O 포트에 해당됩니다. 블록 1 에 출력하는 경우 비트구성은 다음과 같습니다.

위치	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
포트	17	16	15	14	13	12	11	10

portOff

void portOff (u16 portNumber)

portNumber : 포트번호

portNumber 로 지정된 출력 포트를 Off 상태로 만듭니다.

```
portOff(20);        // 20 번 포트를 Off 상태로 만듭니다.
```

portOn

void portOn (u16 portNumber)

portNumber : 포트번호

portNumber 로 지정된 포트를 On 상태로 만듭니다.

```
portOn(20);         // 20 번 포트를 On 상태로 만듭니다.
```

portReverse

void portReverse (u16 portNumber)

portNumber : 포트번호

portNumber 로 지정된 포트의 상태를 반전시킵니다. 이전 상태가 Off 였다면 On 으로 만들고, On 이였다면 Off 로 만듭니다.

```
portReverse(20);          // 20 번 포트를 이전상태와 반대로 만듭니다.
```

portOutStat

u8 portOutStat (u16 portNumber)

portNumber : 포트번호

출력포트의 상태를 읽어옵니다. 해당포트는 사전에 출력모드로 설정되어 있어야 합니다.

```
I=portOutStat(20);        // 20 번 포트가 출력중인 상태를 읽어서 변수 i 에 저장합니다.
```

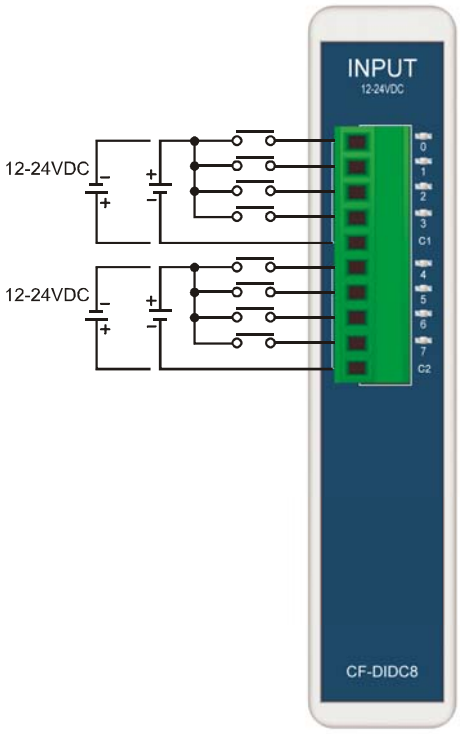
여기서 잠깐!

라이브러리 호출시 파라메터의 개수가 모자라거나 초과할 경우, 별도의 컴파일 에러가 발생하지 않으므로 이점 주의하시기 바랍니다.

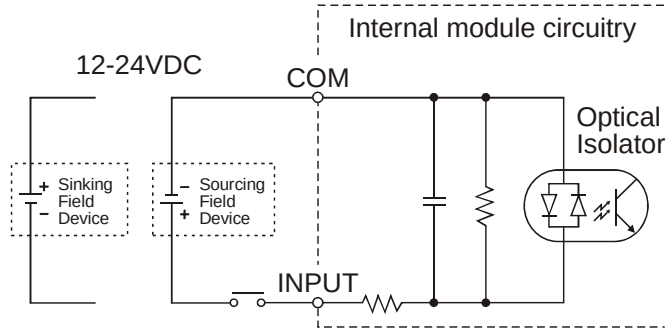
```
portBlockOut (0);  // 두개의 파라메터가 필요하지만, 하나만 써도 별도의 에러가 발생하지 않음.
portBlockOut (0,1,2);  // 파라메터가 초과되는 경우에도 에러가 발생하지 않음.
```

디지털 입력 모듈

DC 입력 8 점 모듈

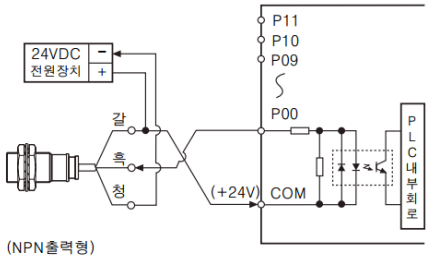


CF-DIDC8 입력 사양	
점수	8 점
오퍼레이팅 전압	12 –24VDC
입력전압 범위	11V – 26VDC
입력 전류	약 5mA @ 24VDC
ON/ OFF 응답속도	3mS
입력 임피던스	4.7K 오옴 @ 24VDC
OFF 인식 레벨	< 6.2VDC
ON 인식 레벨	> 7.2VDC



양방향 전원을 인식할 수 있는 DC12V ~24V 입력 모듈입니다.

근접센서와는 다음과 같이 결선하세요. 2 선식은 사용하실 수 없습니다. 3 선식으로 하세요.



디지털 입력 모듈 관련 라이브러리

portIn

u8 portIn (u16 portNumber)

portNumber : 포트번호

portNumber 로 지정된 포트의 상태를 읽어옵니다.결과 값은 0 또는 1 이 됩니다.
On 상태일경우에는 1 을 반환하고, Off 상태일 경우에는 0 을 반환합니다.

```
I = portIn(1);      // 1 번 포트상태를 읽어서 변수 I 에 저장합니다.
```

portBlockIn

u8 portBlockIn (u8 portBlockNumber)

portBlockNumber : 포트 블록번호 (0 부터 5 사이값)

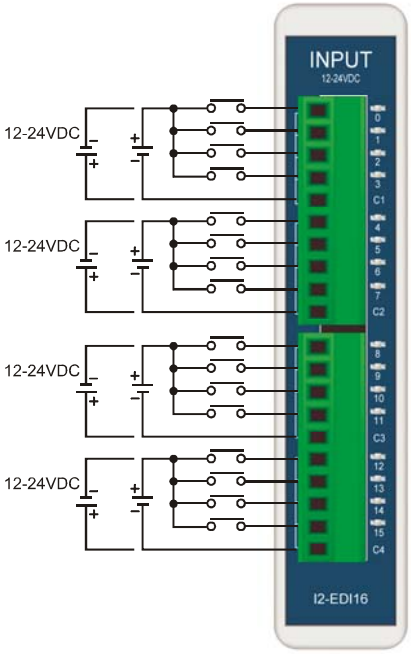
portBlockNumber 로 지정된 블록으로부터 포트 8 점의 상태를 모두 읽어 반환하는 함수 입니다. 결과값은 1 바이트값이 됩니다.

0 블록 (+0)	1 블록(+10)	2 블록(+20)	3 블록(+30)	4 블록(+40)	5 블록(+50)
0 부터 7	10 부터 17	20 부터 27	30 부터 37	40 부터 47	50 부터 57

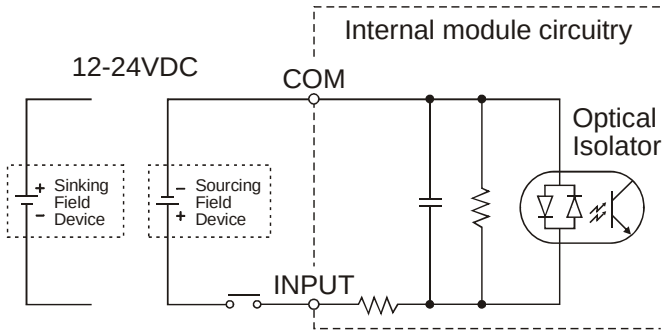
블록 0 을 읽었을경우, 포트 7 번이 MSB (최상위비트)에 위치하게 됩니다.

	MSB							LSB
위치	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
포트	7	6	5	4	3	2	1	0

확장 디지털 입력 16 점 모듈



I2-EDI16 출력 사양	
점수	16 점
오퍼레이팅 전압	12 –24VDC
입력전압 범위	11V – 26VDC
입력 전류	약 5mA @ 24VDC
ON/ OFF 응답속도	5 mS, 따라서 최소 5mS 간격으로 읽어야 합니다.
입력 임피던스	4.7K 오옴 @ 24VDC
OFF 인식 레벨	< 7.4VDC
ON 인식 레벨	> 7.5VDC



확장 DI 모듈도 양방향 전원을 인식할 수 있는 DC12V~ 24V 입력 모듈입니다. 포트번호는 모듈하단에 있는 딥스위치로 결정합니다. 한 시스템에서 중복된 딥스위치 설정이 없도록 주의하십시오.

딥스위치	블록	포트번호
1 2 3 OFF ON	0	0x00 ~0x0F
1 2 3 OFF ON	1	0x10 ~0x1F
1 2 3 OFF ON	2	0x20 ~0x2F
1 2 3 OFF ON	3	0x30 ~0x3F

딥스위치	블록	포트번호
1 2 3 OFF ON	4	0x40 ~0x4F
1 2 3 OFF ON	5	0x50 ~0x5F
1 2 3 OFF ON	6	0x60 ~0x6F
1 2 3 OFF ON	7	0x70 ~0x7F

확장 디지털 입력 모듈 관련 라이브러리

eportIn

u8 eportIn (u16 eportNumber)

eportNumber : 확장 포트번호 (16 진수 0 ~ 0X7F 까지 사용가능)

eportNumber 로 지정된 포트로부터 핀의 상태를 읽어옵니다.결과 값은 0 또는 1 이 됩니다. On 상태일경우에는 1 을 반환하고, Off 상태일 경우에는 0 을 반환합니다.

```
I = eportIn(0x10); // 0x10 번 포트상태를 읽어서 변수 I 에 저장합니다.
```

eportBlockIn

u16 eportBlockIn (u16 eportBlockNumber)

eportBlockNumber : 확장 포트 블록번호 (0 부터 7 까지 사용가능)

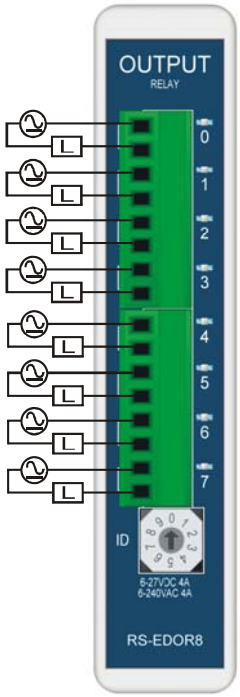
eportBlockNumber 로 지정된 블록으로부터 포트 16 점의 상태를 모두 읽어 반환하는 함수입니다. 결과값은 16 비트값이 됩니다. 포트 블록번호는 딥스위치로 결정됩니다.
(앞페이지의 표참조)

```
I = eportBlockIn(1);     // 0X10~0X1F 번 포트상태를 모두 읽어서 변수 I 에 저장합니다.
```

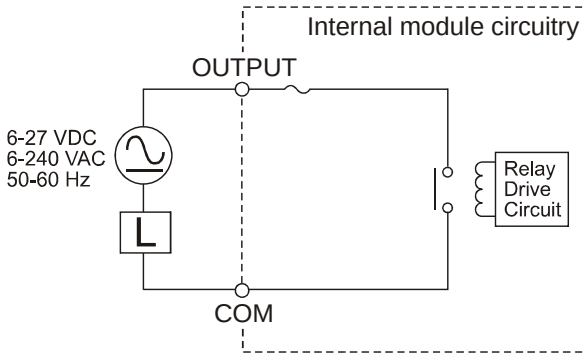
블록 1 을 읽었을 경우, 다음과 같은 16 비트값이 반환됩니다.

비트 (10 진수)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
포트 (16 진수)	1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10

확장 릴레이 출력 8 점 모듈



RS-EDOR8 출력 사양	
점수	8 점
오퍼레이팅 전압	6-27VDC / 6-240VAC
출력전압 범위	5-30VDC / 5-264 VAC
최대 전류	4 A / 점, 4A / COMMON
최소 전류	100 mA @ 5VDC
최대 On/Off 가능 주기	**10Hz (초당 10 번)**
상태 LED	접점이 On 되었을때 점등
COMMON 단자	8 점 (개별 COMMON)



전면부에는 ID 번호를 설정할 수 있는 스위치가 있습니다. 이 스위치상태에 따라 출력포트번호가 결정됩니다.

ID 번호	출력포트번호(10 진)
0	0 부터 7
1	10 부터 17
2	20 부터 27
3	30 부터 37
4	40 부터 47
5	50 부터 57
6	60 부터 67
7	70 부터 77
8	80 부터 87
9	90 부터 97

확장 릴레이 출력 모듈 관련 라이브러리

eRelay

void eRelay (u16 eportNumber, u8 value)

eportNumber : 확장 포트번호
value : 변경할 값 (0 또는 1)

eportNumber 로 지정된 릴레이를 On 또는 Off 합니다. value 값이 1 이면 On, 0 이면 Off 로 바꿉니다.

```
eRelay(0,1);  // 확장 0 번 릴레이를 On 합니다.
```

eRelayBlock

void eRelayBlock (u16 eRelayID, u8 value)

eRelayID : 확장 포트 ID 번호 (0 부터 9)
value : 변경할 값 (0 부터 255)

eRelayID 로 지정된 ID 번호의 8 개 릴레이상태를 value 에 저장된 값으로 한번에 변경합니다.

```
eRelayBlock(1,0xff);     // ID 어드레스 1 번 확장릴레이 모듈을 모두 On 합니다.
```

8 비트값의 0 번비트가 아랫쪽, 7 번비트가 위쪽에 해당합니다.

비트위치	7	6	5	4	3	2	1	0
포트 (ID 가 2 일때)	27	26	25	24	23	22	21	20

*확장 릴레이 출력모듈의 최대 억세스 가능 속도는 초당 10 번입니다. (10Hz)
***유도성 부하는 1 초 ON/ 1 초 OFF 보다 더 빠른 간격으로 ON/OFF 하지 마십시오. (1Hz)**

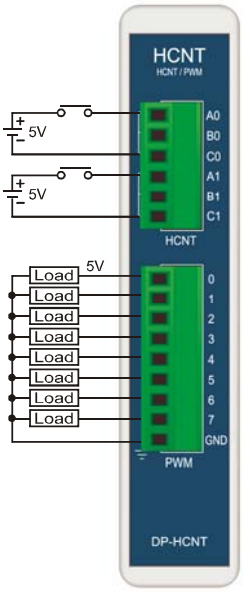
제 5 장

고속카운터모듈

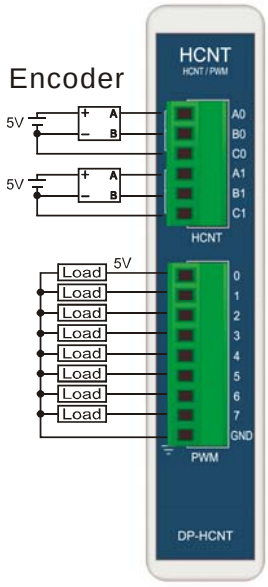
고속카운터 모듈

고속카운터 모듈을 사용하면 고속카운터입력 (또는 엔코더입력) , PWM 펄스출력이 가능합니다. 다른 종류의 입출력모듈과는 다르게, 5V 를 주로 사용하므로 주의를 요합니다.

고속펄스 입력시

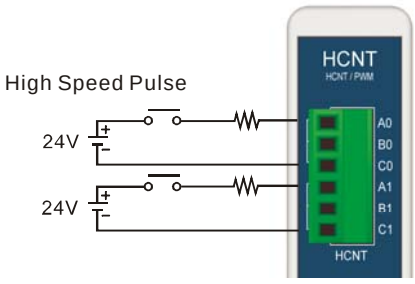


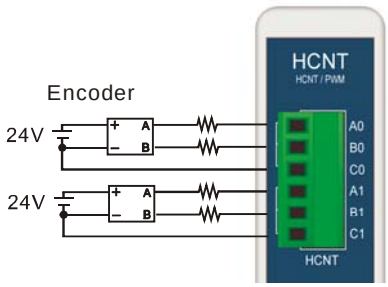
엔코더 입력시



고속카운터에 5V 를 입력하십시오. (2V 이상 입력시 High 로 인식합니다.)

고속카운터에 12V ~ 24V 전압을 입력하려면 2.2K (0.5W)오옴의 저항을 직렬로 연결하십시오.





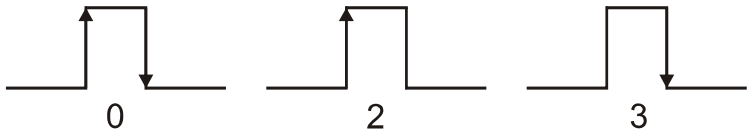
고속카운터 모듈 관련 라이브러리

countMode

void countMode (u8 cntChannel, u8 cntMode)

cntChannel : 고속카운트 채널번호 (0 또는 1)

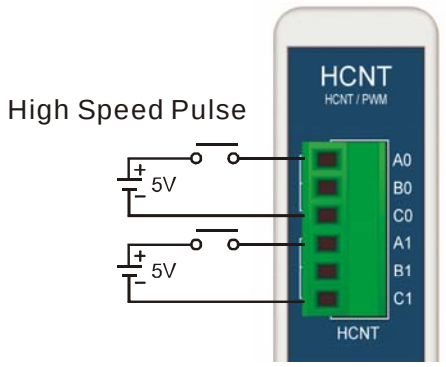
cntMode : 0= 단순 카운터 , 상승과 하강에지에서 모두 카운트업
1=엔코더 카운트,
2= 단순 카운터 , 상승에지에서 카운트업
3= 단순 카운터 , 하강에지에서 카운트업



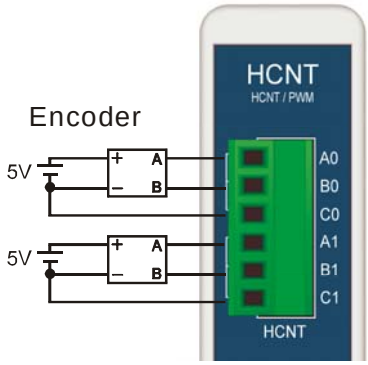
cntMode 를 0 으로 하면 입력펄스의 상승에지와 하강에지에서 모두 증가하는 단순카운트 모드입니다.

cntMode 를 2 로 하면 상승에지에서만 증가, 3 으로하면 하강에지에서만 증가하는 단순카운트 모드입니다.

해당채널의 A 단자로 입력되는 펄스를 카운트합니다. 이 모드 사용시 B 단자는 아무 것도 연결하지 않습니다.

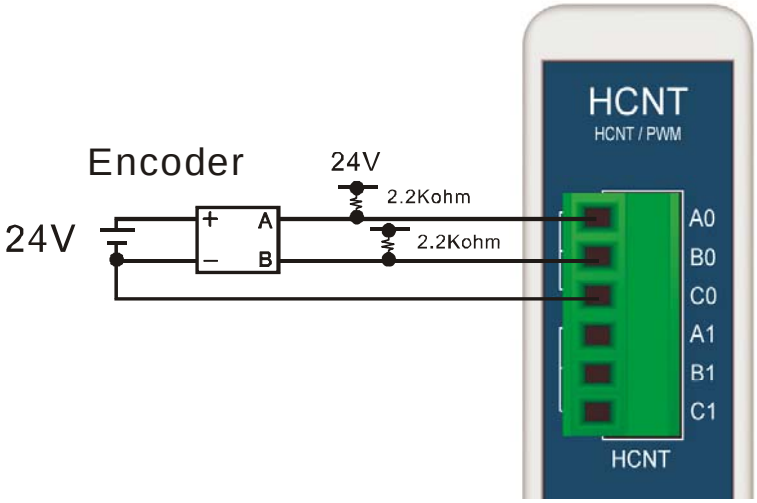


CntMode 를 1 로 하면 엔코더 카운트 모드입니다. 해당채널의 A,B 단자에 엔코더의 A, B 상을 연결하여 엔코더로부터 발생되는 펄스를 카운트업/ 다운합니다. 엔코더를 어느방향으로 돌리느냐에 따라서 값이 증가되거나 감소됩니다. CountMode 함수 사용시 기존 카운터값은 지워지고, 0 으로 초기화 됩니다.



5V 를 출력하는 라인드라이브 방식의 ENCODER 는 위와 같이 직접 연결하세요.

만약 오픈콜렉터 타입의 엔코더에 24V 전원을 사용한다면 2.2K 풀업저항을 붙이세요.



count

u16 count (u8 cntChannel)

cntChannel : 고속카운트 채널번호 (0 또는 1)

해당 카운트 채널에 입력된 펄스 수를 읽어오는 함수입니다. 2 개의 카운트 입력 채널은, 항상 들어오는 펄스 수를 카운트 하고 있습니다. 카운트 중인 값은 count 함수로 읽어올 수 있습니다.

즉, C 프로그램이 다른 부분을 실행하고 있을때에도 카운트채널은 항상 입력되는 펄스수를 카운트하고 있기 때문에, 언제 발생될 지 모르는 돌발적인 상황을 체크한다던가, 고속으로 입력되는 펄스를 놓치지 않고 카운트할 수 있습니다. 최대 65535 (16 비트)까지 카운트 할 수 있습니다.

```
countMode(0,0);  // 단순 카운트모드로 설정
cnt = count(0);  // 0 번 채널에 입력된 카운트값을 읽어옵니다.
```

CoutnMode 함수에서 엔코더 입력모드로 설정해 놓았다면, 엔코더의 위치값을 읽어옵니다.

```
countMode(0,1);  // 엔코더 카운트모드로 설정
cnt = count(0);  // 0 번 채널에 연결된 엔코더로부터 위치값을 읽어옵니다.
```

엔코더 위치값이 음수일때에는 2 의보수로 표현됩니다. 즉 –1 일 경우 0xffff 가 됩니다.
결과적으로 엔코더 위치값 사용범위는 –32768 에서 +32767 까지가 됩니다.

만약 엔코더를 조금만 돌렸는데도 변화량이 너무커서, 금방 사용범위를 초과할 것 같다면, countPrescaler 함수를 사용하여 분주비를 높게 설정하여 사용하시기 바랍니다.

Count 함수 실행시 기존 카운터값은 지워지지 않고 그대로 유지됩니다.

countPrescaler

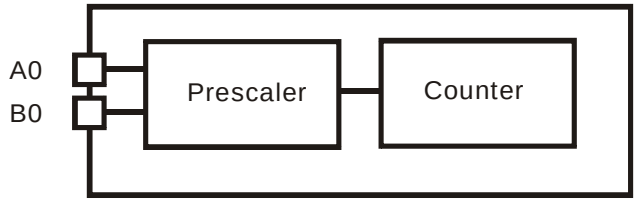
void countPrescaler (u8 cntChannel, u16 prescaleValue)

cntChannel : 고속카운트 채널번호 (0 또는 1)
prescaleValue : 분주비 (0~65535)

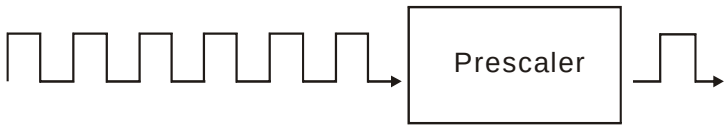
카운트입력에 대한 분주비를 결정할 수 있는 함수 입니다. 분주비 디폴트상태는 0 입니다.

```
countPrescaler(0, 1);
```

분주비를 1 로 하면 이후로부터 2 개의 펄스가 들어올 때 1 증가됩니다. 즉, 분주비에서 더하기 1 한 값이 실제 분주비가 됩니다.



분주비는 0 부터 65535 사이의 값을 자유롭게 사용할 수 있습니다. 5 로 설정하면 6 개의 펄스가 들어왔을 때 1 증가됩니다.



CountPrescaler 함수 사용시 기존 카운터값은 지워지고, 0 으로 초기화 됩니다.

countReset

void countReset (u8 cntChannel)

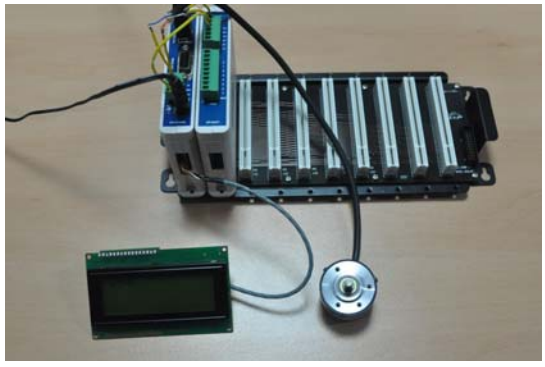
cntChannel : 고속카운트 채널번호 (0 또는 1)

해당 카운트 채널을 클리어시킵니다. 기존에 카운트하던 값들은 지우고 0 에서부터 새롭게 카운트를 진행할 때 사용합니다.

```
countPrescaler(0, 0);
countReset(0);      // 채널 0 클리어
delay(1000);
i = count(0);       // 1 초간 카운트된 값을 변수 i 에 저장합니다.
```

다음은 엔코더를 연결하여 테스트할 수 있는 소스 프로그램입니다.

```
#include "moacon500.h"
void cmain(void)
{
    clcdPower(1);
    delay(30);
    clcdI2cInit(0);
    countMode(0,1);
    pwm(0,100,500);
    pwm(1,100,500);
    pwm(2,100,500);
    pwm(3,100,500);
    pwm(4,100,500);
    pwm(5,100,500);
    pwm(6,100,500);
    pwm(7,100,500);
    for(;;) {
        clcdPrint(0,0,"%6d",count(0));
        delay(100);
        }
}
```

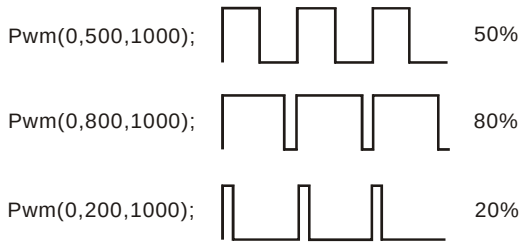


pwm

void pwm (u8 pwmChannel, u16 pwmDuty, u16 pwmWidth)

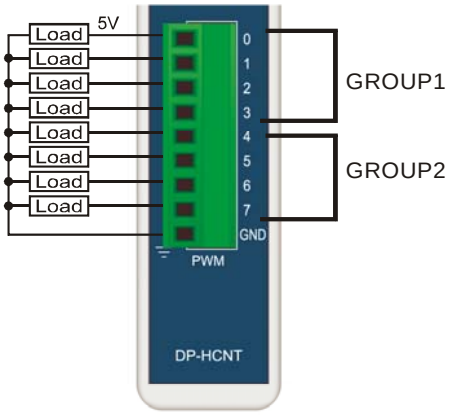
pwmChannel : PWM 채널번호 (0 ~ 7)
pwmDuty : 듀티비 (1 에서 주기값 사이의 값)
pwmWidth : 주기값 (1 에서 65535 사이값)

PWM 출력포트에는 5V 의 PWM 파형이 출력하는 함수입니다. 원하는 채널에 듀티비, 주기값등을 정해주면, 해당 듀티비를 가진 PWM 파형이 출력됩니다. 주기값은 최대 65535 까지 사용할 수 있으며, 이 범위 내에서 듀티비를 설정할 수 있습니다.



pwm 은 백그라운드에서 출력되도록 되어있으므로, 한번 발생된 pwm 파형은 pwmoff 함수가 수행되기 전까지 계속 출력됩니다.

MOACON 에는 총 8 개의 PWM 채널이 있으며, 이중 0 부터 3 까지는 주기값이 같아야 합니다. 마찬가지로 4 부터 7 까지도 주기값이 같아야 합니다. 즉, 두개의 PWM 그룹이 존재합니다. 하나의 그룹에서는 하나의 주기값만 사용가능합니다.




```
pwm(0, 10, 400);
pwm(1, 40, 400);
pwm(2, 200, 400);
pwm(3, 350, 400);

pwm(4, 1000, 1024);
pwm(5, 160, 1024);
pwm(6, 80, 1024);
pwm(7, 10, 1024);
```

PWM 파형의 주파수는 주기값(pwmwidth)에 의해서 결정됩니다. 주기값으로 부터 다음공식에 의해 실제주파수를 계산할 수 있습니다.

$$\text{출력주파수} = \frac{72000000}{\text{주기값}}$$

만약 주기값을 65535 로 했다면 1098Hz 의 파형이 출력됩니다.

```
Pwm (0, 30000, 65535 ) ;  // 1098Hz 의 pwm 파형이 출력됩니다.
```

주기값을 100 으로 했다면 720000 Hz 즉 720KHz 의 파형이 출력되어야 하지만, 실제로는 713KHz 정도가 출력됩니다. 주파수가 빠를수록 약간의 오차율을 가지고 있으므로 위의 공식으로 대략의 값을 알아낸뒤, 실제 정확한 주파수는 주파수측정기로 측정한뒤 사용하시기 바랍니다.

pwmoff

void pwmoff (u8 pwmChannel)

pwmChannel : PWM 채널번호 (0 ~ 7)

해당 채널의 PWM 파형을 중지시킵니다.

```
pwm(0,500,1000);
delay(1000);      // 1 초 딜레이
pwmoff(0);        // 1 초뒤 pwm 파형이 중지됩니다.
```

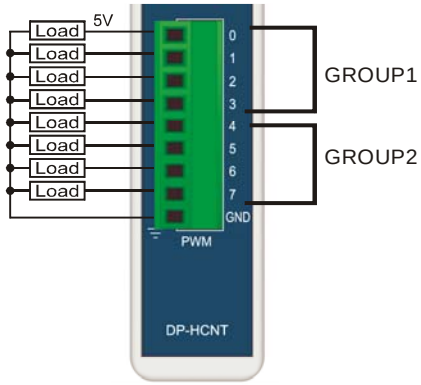
freqOut

void freqOut (u8 pwmChannel, u32 freqVal)

pwmChannel : PWM 채널번호(0 ~ 7)
freqVal : 주파수

원하는 주파수의 파형을 PWM 채널을 통해 출력하는 함수 입니다. PWM 채널을 그대로 사용할 수 있습니다. 단 하나의 그룹에서 하나의 채널만 사용할 수 있습니다.

MOACON 에서는 2 개의 PWM 그룹이 있습니다. 따라서 freqOut 함수로 2 개의 서로 다른 주파수를 가진 파형을 출력할 수 있습니다.



FreqOut (0, 440);

0 번채널에 440Hz 의 파형을 출력합니다.
나머지 1,2,3 번채널은 사용할수 없습니다.
(pwm 함수도 이 채널을 사용할 수 없습니다.)

FreqOut (5, 30000);

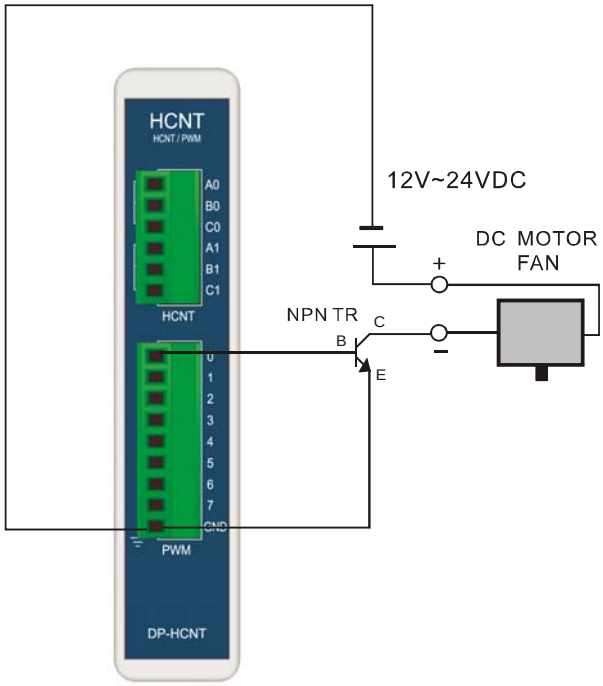
5 번채널에 30KHz 의 파형을 출력합니다.
나머지 4, 6, 7 번 채널은 사용할 수 없습니다.
(pwm 함수도 이 채널을 사용할 수 없습니다.)

freqOut 함수로 출력중인 파형을 중단시키고자할 때에도 pwmOff 함수를 사용합니다.

```
freqOut(0,435);  //0 번채널에 435Hz 의 파형이 출력됩니다.
delay(1000);      // 1 초 딜레이
pwmoff(0);        // 1 초뒤 pwm 파형이 중지됩니다.
```

PWM 출력 활용법

5V 의 PWM 출력으로는 매우 작은 전류만 흘릴 수 있기 때문에, 큰 전류를 소비하는 부하를 직접 구동하는 것은 불가능합니다. 아래 그림처럼 NPN TR 을 부착하면 큰 구동전류를 필요로하는 DC 모터나 팬모터의 속도를 제어할 수 있습니다.



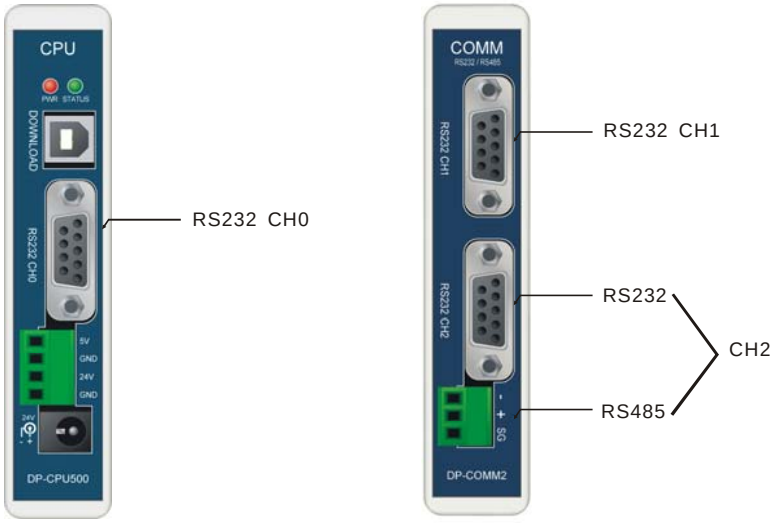
이 때, TR 은 모터의 용량을 감당할 수 있는 용량의 POWER TR (또는 MOSFET)을 사용해주어야 합니다. 만약 TR 에 발열이 있을 경우에 방열판을 부착하여야 과열로 인한 파손을 막을 수 있습니다.

제 6 장

통신모듈

통신모듈

모아콘은 최대 3 개의 시리얼(UART) 포트를 지원합니다. 그중 하나인 채널 0 은 CPU 모듈 전면부에 위치해 있습니다. (±12V 레벨의 RS232 신호를 송수신하는 포트입니다.)



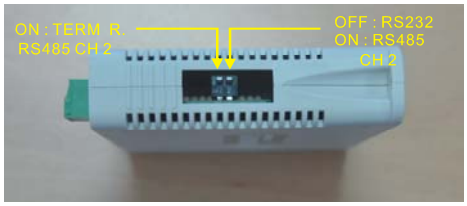
CPU 모듈 통신모듈

채널 0 의 핀 9 번은 UIF 에 전원을 공급하기 위한 5V 단자 출력단자 입니다. 이 5V 출력단자는 comPower 함수에 의해서 ON/OFF 할 수 있습니다.

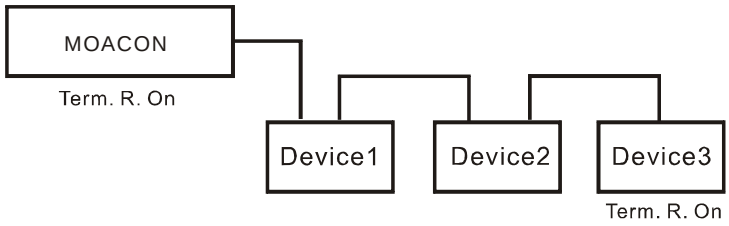
RS232 의 핀구성은 다음과 같습니다. 채널 1 과 채널 2 의 핀 9 번 (5V)는 항상 5V 를 출력합니다. (ON /OFF 불가능)

1
2(TXD)
3(RXD)
4
5(GND)
6
7
8
9 (5V)

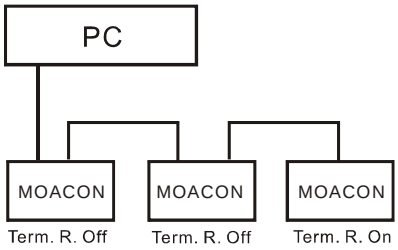
채널 1 와 채널 2 을 사용하기 위해서는 별도의 "통신모듈(DP-COMM2)"을 슬롯에 장착해야합니다. 채널 1 은 RS232 전용으로 되어 있고, 채널 2 는 하단의 딥스위치 설정에 따라서 RS232 / RS485 중 하나를 선택할 수 있습니다.



왼쪽 딥스위치를 ON 으로 하면 RS485 종단저항을 ON 하는 것입니다. 다음과 같이 모아콘이 MASTER 인경우에는 종단저항을 ON 하십시오.



다음과 같이 여러 개의 모아콘이 SLAVE 로 연결되어 있을때에는 가장 끝에 위치한 모아콘에서만 종단저항을 ON 하시면 됩니다.



일반적으로 RS485 사용시에는 송신 허가 (Transmit Enable) 를 따로 조정해주어야 하는데, 모아콘에서는 유저가 특별히 조정하지 않아도 내부적으로 알아서 처리해줍니다.
모아콘의 모든 RS232 포트는 ±12V 레벨의 신호로 송수신 하도록 되어 있습니다. 5V 레벨의 RS232 신호는 사용하지 않습니다.

통신모듈관련 라이브러리

openCom

void openCom (u8 comCh, u32 comBaud, u8 comMode)

comCh : 통신채널번호 (0 ~ 2)
comBaud : 보레이트
comMode : 통신 프로토콜

RS232 / RS485 통신 기능을 사용하기 전에 앞서 반드시 써주어야 함수입니다. 해당 채널을 어떤 보레이트 어떤 프로토콜을 가지고 OPEN 할 것인지 선언해 주는 역할을 수행합니다.

comBaud 에는 보레이트 값을 써줍니다. 일반적으로 자주 사용되는 보레이트는 다음과 같습니다. (1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200)

comMode 에는 C8N1 또는 C9E1 과 같은식으로 적어주어야 합니다. 각각의 문자열이 의미하는 바는 다음과 같습니다.

comMode	데이터 비트	패리티	스톱비트
C7E1	7	EVEN	1
C7O1	7	ODD	1
C8N1	8	NO	1
C8E1	8	EVEN	1
C8O1	8	ODD	1
C7E2	7	EVEN	2
C7O2	7	ODD	2
C8N2	8	NO	2
C8E2	8	EVEN	2
C8O2	8	ODD	2

*OS 버전 1.7 부터 이 표현을 사용하실 수 있습니다.

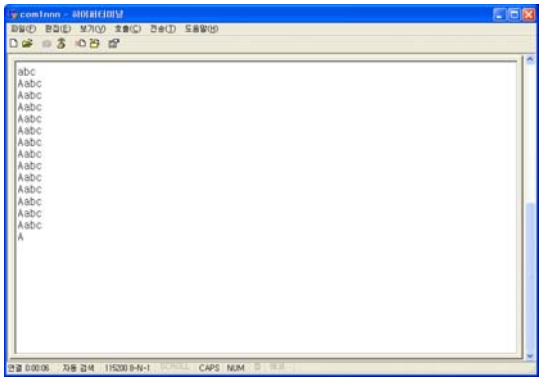
*7 비트 None 패리티는 지원하지 않습니다.

OpenCom 사용시 255 바이트의 수신버퍼가 할당되며, 해당버퍼는 클리어됩니다.
수신버퍼의 크기를 사용자가 결정할 수 없습니다.

다음은 openCom 을 사용한 예제 프로그램입니다.

```
#include "moacon500.h"
void cmain(void)
{
  openCom(0, 115200, C8N1);
  while (1) {
        delay(500);
        comPrint(0,"abc\r\n");
        comPut(0,0x41);
        }
}
```

MOACON 의 채널 0 포트를 PC 와 연결한후, 하이퍼 터미널과 같은 통신 프로그램으로 실행결과를 확인하실 수 있습니다.



comPut

void comPut (u8 comCh, u8 comChar)
comCh : 통신채널번호 (0 ~ 2)
comChar : 송신할캐릭터

한바이트를 송신하는 함수 입니다. comCh 에는 통신채널 번호, comChar 에는 1 바이트의 데이터 (ASCII 코드 또는 문자)를 적어줍니다.

```
#include "moacon500.h"
void cmain(void)
{
  openCom(1, 115200, C8N1);
  while (1) {
        delay(500);
        comPut(1,0x41);
        comPut(1,'B');
        }
}
```

comChar 에 숫자를 써넣으면 ASCII 코드로 인식됩니다. 문자를 직접 써넣을려면 위 예제 프로그램과 같이 [‘] 어포스트로피로 감싼 하나의 문자를 써넣어줍니다. 단 영문자 또는 숫자나 특수기호이여야 합니다. 한글, 한자와 같은 2 바이트형 문자는 사용할 수 없습니다.

이 함수는 단 1 바이트의 캐릭터만 송신할 수 있습니다. 여러 개의 문자를 한꺼번에 송신하고 싶다면 comPrint 함수를 사용하십시오.

comPrint

void comPrint (u8 comCh, char* comString)
comCh : 통신채널번호 (0 ~ 2)
comString : 송신할 문자열

UART 포트로 여러 개의 문자로 이루어진, 문자열을 송신하는 함수 입니다. comCh 에는 통신채널 번호, comString 에는 따옴표로 둘러 쌓여진 문자열을 적어줍니다.

```
#include "moacon500.h"
void cmain(void)
{
  openCom(0, 115200, C8N1);
  while (1) {
        delay(500);
        comPrint(0,"abc\r\n");
        }
}
```

위 예제 프로그램에서 "abc\r\n" 이 comPrint 함수에서 송신하는 문자열입니다. \r\n 은 복귀개행 코드입니다. 즉 표시행을 한줄 아래로 내려주는 특수코드입니다.

문자열안에는 printf 함수에서와 같은 %d, %x 와 같은 표현도 사용할 수 있습니다.

comPrint(0, "internal value is %d", comi);

변수 comi 의 값을 10 진형태로 바꾸어 문자열 안에 포함시켜 송신합니다. %포맷 문자 변환에 대한 사용법은 "부록 1. C 언어"에 나와있는 printf 함수 사용설명을 참조하시기 바랍니다.

잠깐만…

MOACON 시스템에서 UART 송신 함수 (comPut, comPrint)는 따로 버퍼를 사용하지 않고, 송신 데이터를 모두 송신할 때까지 대기합니다. 즉 해당 함수가 더 이상 송신할 데이터가 없을때까지 프로세스를 붙잡고 있게 됩니다.

comGet

short comGet (u8 comCh)

comCh : 통신채널번호 (0 ~ 2)

리턴값 : 정상 수행시 읽어온 데이터값 (0~0xff 사이값), 정상적으로 읽지 못한경우 –1 (0xffff)

수신버퍼에서 캐릭터 한바이트를 읽어오는 함수 입니다. comCh 에는 통신채널 번호를 적어줍니다.

통신포트로부터 수신된 바이트는 내부에 있는 255 바이트의 수신버퍼로 자동저장됩니다. comGet 함수로 가장 먼저 수신된 1 개의 바이트를 읽어옵니다.

comGet 함수를 사용하기위해서는 현재 수신된 데이터가 있는지, 사전에 조사해볼 필요가 있습니다. 바로 comLen 함수가 수신버퍼에 얼마 만큼의 데이터가 쌓여있는 알아 볼 수 있는 함수 입니다.

```
#include "moacon500.h"
void cmain(void)
{
  short chari;
  openCom(0, 115200, C8N1);
  while (1) {
        while (!comLen(0));  // 수신데이터가 있을때까지 대기
        chari = comGet(0);   // 한바이트를 수신해서
        comPut(0,chari);     // 그대로 송신합니다.
        }
}
```

위 예제 프로그램은 입력된 데이터를 그대로 발송하는 에코 프로그램입니다.

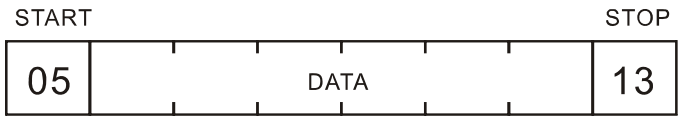
comGetInterval

u8 comGetInterval (u8 comCh)
comCh : 통신채널번호 (0 ~ 2)
리턴값 : 0 부터 0xff 사이의 값

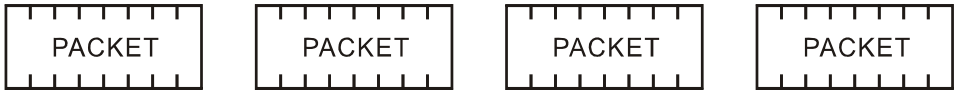
바로 이전에 수신된 데이터와의 간격을 알 수 있는 함수입니다. 이 함수가 왜 필요한지 이해하려면 패킷통신에 대해서 알아야 합니다.

RS232 통신에서는 패킷통신을 주로 사용합니다. 의미있는 여러 개의 데이터를 모아서 하나의 데이터집합을 구성합니다. 이것을 패킷이라고 부릅니다.

패킷을 보내거나 받을 때, 무엇보다도 중요한 것은 패킷의 시작과 끝을 판단하는 것입니다. 시작코드와 종료코드를 앞뒤로 감싸서 보내는 방법도 있습니다.이 방법은 시작코드와 종료코드를 DATA 필드에서 사용할 수 없다는 불편함이 있습니다.



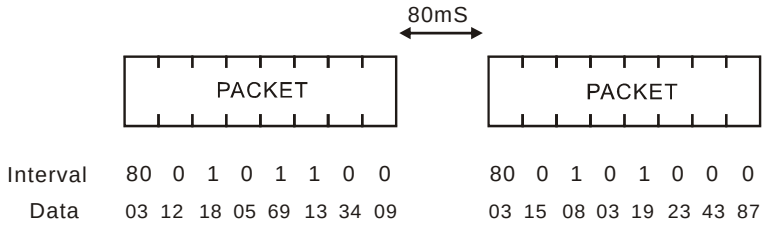
그래서 프레임과의 간격을 두어 구분하는 방법이 자주 쓰입니다. (MODBUS RTU 에서도 이 방법을 사용합니다.)



comGetInterval 함수는 바로 이런 패킷통신을 구현하기 위해서 필요한 함수입니다. 바로 앞에서 수신한 데이터와의 간격을 mS 단위로 알 수 있습니다. 연속해서 오는 데이터라면 간격이 매우 작은값이 됩니다.

바로 앞의 데이터와 간격이 벌어져 있다면, 큰 값이 들어 있습니다. 최대 255 까지 저장됩니다. 255 밀리초 이상의 큰 간격은 더 이상 측정하지 않습니다.

이 정보를 가지고 프레임의 시작과 끝을 구분할 수 있습니다.



가까운 데이터의 간격은 5 미만의 작은 값으로 랜덤하게 나올수 있습니다. (보레이트에 따라 달라질 수 있음)

```
// 수신되는 데이터의 인터벌 정보와 수신값을 LCD 상에 표시
#include "moacon500.h"
void cmain(void)
{
   int i=0;
   clcdI2cInit(0);
   clcdPower(1);
  delay(100);
  clcdCsr(0);
  openCom(1,115200,C8N1);

   while(1) {
     if (comLen(1) > 0) {
       clcdPrint(0,0,"%2X",comGetInterval(1));
       clcdPrint(5,0,"%2X",comGet(1));
       }
}
```

comGet 함수를 수행하기 전에 comGetInterval 함수를 사용하십시오. ComGetInterval 함수는 바로 다음에 comGet 함수로 읽어낼 데이터의 Interval 정보를 알려줍니다.

주의사항 : 이 방법은 MOACON 과 타기기 사이에 어떠한 버퍼링 장치가 없어야 사용가능합니다. 중간에 다른 콘트롤러 (컨버터류 : 예를들면 USB-TO-RS232 컨버터)가 개입하게 된다면, 프레임 간 간격이 재조정될 수 있기 때문입니다.

comLen

u16 comLen (u8 comCh)
comCh : 통신채널번호 (0 ~ 2)

수신버퍼에 쌓여있는 데이터수를 반환해주는 함수입니다. 아무 데이터도 수신되어 있지 않다면 0 을 반환합니다.

잠깐만…

만약 수신버퍼가 꽉찰때까지 comGet 함수를 사용해서 데이터를 읽어내지 않는다면, 그 이후 수신된 데이터는 버퍼에 저장되지 않습니다. 따라서 UART 수신 기능을 사용할때에는 항상 수신되는 데이터처리에 주의를 기울여야 합니다.

comFlush

void comFlush (u8 comCh)
comCh : 통신채널번호 (0 ~ 2)

수신버퍼를 깨끗이 청소해주는 함수입니다.

```
#include "moacon500.h"
void cmain(void)
{
  short chari;
  openCom(0, 115200, C8N1);
  while (1) {
        while (comLen(0) < 5);
        chari = comGet(0);
        if (chari == 0x41) comFlush(0);
        else comPut(0,chari);
        }
}
```

위 예제프로그램은 수신된 데이터중 '대문자 A'가 있다면, 수신버퍼를 클리어해줍니다.

comGets

short comGets(u8 comch, u8 * dest, u16 length)

comCh : 통신채널번호 (0 ~ 2)
dest : 저장할곳의 주소
length : 읽어올 데이터 개수
리턴값 : 정상적으로 읽었을 경우 –1, 읽지 못했을 경우 0

수신버퍼에서 여러 개의 데이터를 읽어오는 함수 입니다. comCh 에는 통신채널 번호를 써주고, dest 에는 저장할 배열의 포인터를 적어줍니다. Length 에는 읽어올 데이터의 바이트수를 적어줍니다.

이 함수를 실행하기 전에 comLen 을 사용해서 읽어올 데이터 개수만큼 버퍼에 있는지 확인한후 본 함수를 실행시켜 주십시오. 정상적으로 읽었을 경우에는 –1 을 리턴하고, 읽지 못했을 경우에는 0 을 리턴합니다.

```
u8 MbcoilBuffer[20];
if (comLen(0)>5) {
      res = comGets(0,MBcoilBuffer,5);
      }
clcdPrint(0,0,"%2x%2x%2x%2x%2x",MBcoilBuffer[0],MBcoilBuffer[1],
  MBcoilBuffer[2],MBcoilBuffer[3],MBcoilBuffer[4]);
```

comPuts

void comPuts(u8 comch, u8 * dest, u16 length)

comCh : 통신채널번호 (0 ~ 2)

dest :전송할 데이터가 저장된곳의 주소

length : 보낼 데이터 개수

한꺼번에 여러 개의 데이터를 송신하는 함수 입니다. comCh 에는 통신채널 번호를 적어주고, dest 에는 송신데이터가 저장된 배열의 주소를 기입합니다. 끝으로 length 에 보낼 데이터 바이트 개수를 적어주십시오.

```
u8 MbcoilBuffer[20];
if (comLen(0)>5) {
      res = comGets(0,MBcoilBuffer,5);  // 채널 0 에서 읽어온 데이터를
      comPuts(1,MBcoilBuffer,5);  // 채널 1 로 보냅니다.
      }
```

수신버퍼 이벤트 함수

수신버퍼에 데이터가 있을때 이벤트를 발생시켜주는 기능입니다.

*수신버퍼를 비우지 않는다면 계속해서 이벤트가 발생됩니다. comGet, comGets 함수로 데이터를 읽어내야만 수신버퍼가 비워집니다. (또는 comFlush 로도 수신버퍼를 비울수 있습니다.)

*이 이벤트는 1mS 간격으로 수신버퍼를 체크합니다. 따라서 수신버퍼에 최근 1mS 사이에 수신된 1 개이상의 데이터가 들어가 있을 수 있습니다. 따라서 comLen 함수로 수신된 데이터의 개수를 확인하고, 수신된 데이터를 모두 읽어서 처리하는 식으로 코드를 작성하시기 바랍니다.

startCom0Event

void startCom0Event ( )

통신채널 0 번의 수신 이벤트를 시작합니다.

startCom1Event

void startCom1Event ( )

통신채널 1 번의 수신 이벤트를 시작합니다.

startCom2Event

void startCom2Event ( )

통신채널 2 번의 수신 이벤트를 시작합니다.

com0Event

void com0Event ( )

채널 0 수신버퍼에 데이터가 있을때, 이 함수가 실행됩니다.

```
#include "moacon500.h"

void cmain(void)
{
  openCom(0,115200,C8N1);   // 채널 0 오픈
  startCom0Event();          // 채널 0 수신인터럽트 개시
  while(1) { } // 무한루프
}

void com0Event(void)     // 채널 0 에 수신된 데이터가 있을때 이곳이 호출됩니다.
{
  while (comLen(0)>0)         // 버퍼에 있는 모든 내용을 출력합니다.
        comPut(0,comGet(0));
}
```

com1Event

void com1Event ( )

채널 1 수신버퍼에 데이터가 있을때, 이 함수가 실행됩니다.

com2Event

void com2Event ( )

채널 2 수신버퍼에 데이터가 있을때, 이 함수가 실행됩니다.

stopCom0Event

void stopCom0Event ( )

통신채널 0 번의 수신 이벤트를 종료합니다.

stopCom1Event

void stopCom1Event ( )

통신채널 1 번의 수신 이벤트를 종료합니다.

stopCom2Event

void stopCom2Event ( )

통신채널 2 번의 수신 이벤트를 종료합니다.

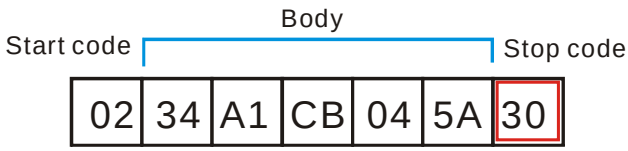
수신버퍼 검사이벤트 함수

특정 코드가 수신되면 이벤트를 발생시켜주는 기능입니다.

startCom0UntilEvent

void startCom0UntilEvent (u8 untilCode)
untilCode : 종료코드 (0 부터 0xff 사이의 값)

통신채널 0 번의 수신 검사 이벤트를 시작합니다. 수신버퍼에 UntilCode 로 지정한 코드가 수신되면 이벤트가 발생됩니다. 코드는 0 부터 0xff 사이의 아무 값이나 사용할 수 있습니다.



위 그림처럼 통신 패킷이 구성되어 있을 경우, stop code 인 0x30 이 수신되면 이벤트가 발생됩니다. 이벤트 처리 함수에서 수신버퍼에 있는 내용을 모두 읽어서 start code 와 stop code 사이에 있는 본체(Body)를 찾을 수 있습니다.

startCom1UntilEvent

void startCom1UntilEvent (u8 untilCode)
untilCode : 종료코드 (0 부터 0xff 사이의 값)

통신채널 1 번의 수신 검사 이벤트를 시작합니다.

startCom2UntilEvent

void startCom2UntilEvent (u8 untilCode)
untilCode : 종료코드 (0 부터 0xff 사이의 값)

통신채널 2 번의 수신 검사 이벤트를 시작합니다.

com0UntilEvent

void com0UntilEvent ( )

채널 0 에서 startCom0UntilEvent 함수에서 지정된 코드가 수신되었을 때, 이 함수가 실행됩니다.

```
#include "moacon500.h"

void cmain(void)
{
  startCom0UntilEvent(0x30);  // 채널 0 수신검사 이벤트 시작, 0x30 을 찾으면 이벤트실행
  openCom(0,115200,C8N1);
  while(1) {       }
}

void com0UntilEvent(void)  // 채널 0 수신버퍼에서 0x30 을 찾으면 이 함수가 실행됨.
{
  while (comLen(0)>0)         // 버퍼에 있는 모든 내용을 출력합니다.
        comPut(0,comGet(0));
}
```

com1UntilEvent

void com1UntilEvent ( )

채널 1 에서 startCom1UntilEvent 함수에서 지정된 코드가 수신되었을 때, 이 함수가 실행됩니다.

com2UntilEvent

void com2UntilEvent ( )

채널 2 에서 startCom2UntilEvent 함수에서 지정된 코드가 수신되었을 때, 이 함수가 실행됩니다.

stopCom0UntilEvent

void stopCom0UntilEvent ( )

통신채널 0 번의 수신 이벤트를 종료합니다.

stopCom1UntilEvent

void stopCom1UntilEvent ( )

통신채널 1 번의 수신 이벤트를 종료합니다.

stopCom2UntilEvent

void stopCom2UntilEvent ( )

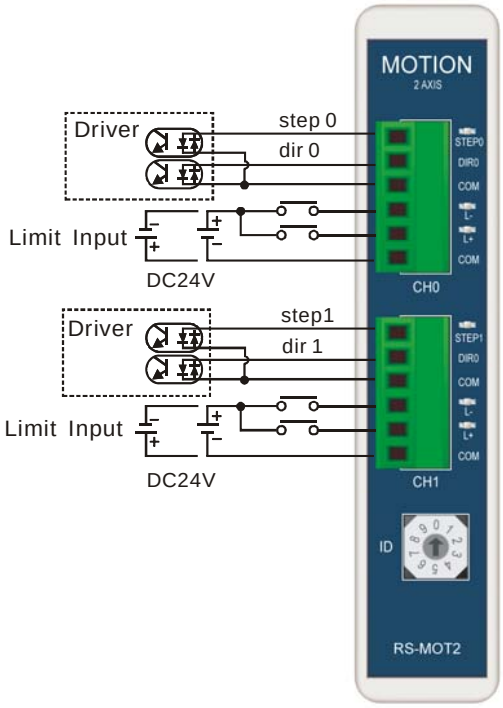
통신채널 2 번의 수신 이벤트를 종료합니다.

제 7 장

모션제어모듈

2 축 모션 제어 모듈

스탭 모터제어를 위한 2 축 펄스 출력 모듈입니다.



STEP 과 DIR 신호는 DC5V 신호로 출력됩니다. (출력전류 5mA)

STEP 은 모터의 회전수를 결정짓는 펄스신호입니다. 이 신호로 모터를 직접 구동할 수는 없습니다. 이 신호를 모터드라이버로 보내는 것입니다.

DIR 은 모터의 회전방향을 결정짓는 신호입니다. 이 신호 역시 모터 드라이버로 전송해야 합니다. (5V 는 CW, 0V 는 CCW)

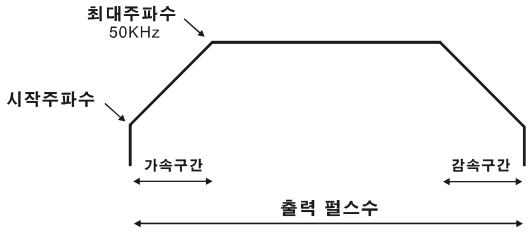
본 모듈의 LIMIT 입력은 비상시 모터 STOP 을 위한 입력입니다.

일반적인 LIMIT 입력은 별도 입력모듈을 사용하여 입력받으시기 바랍니다.

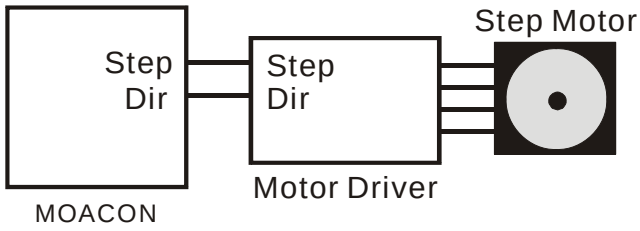
본 모듈의 LIMIT 신호에 24VDC 가 입력되면, 모터는 그 즉시 STOP 되며, 이 신호가 OFF 되기 전까지 동작하지 않습니다. (소프트웨어적으로 무시할 수 없습니다.)

모듈 앞부분에 ID 를 조절하는 선택스위치가 있습니다. 0 부터 9 까지 숫자중 하나를 선택하십시오. 모션 콘트롤 모듈은 최대 10 개까지 장착할 수 있습니다.

모션 콘트롤모듈 하나당 2 축의 모터를 동시제어할 수 있으므로, 총 20 개의 모터를 연결하실 수 있습니다. 모션제어 모듈에서는 최대 50KHz 까지 가감속 펄스를 출력할 수 있습니다.



본 제품으로 스탭 모터를 구동하기 위해서는 별도의 스탭 모터드라이버가 필요합니다.

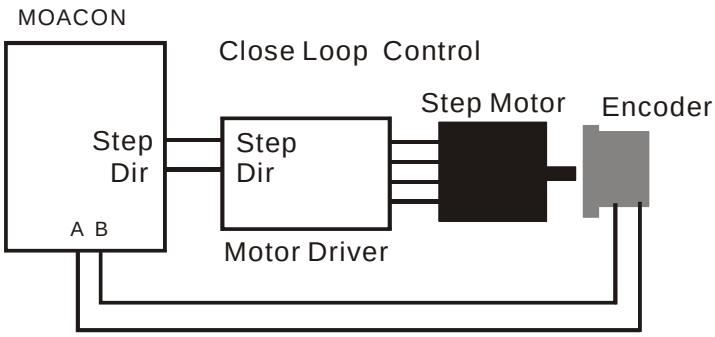


본 제품에서 직선/원호 보간기능은 지원하지 않습니다.

모터제어방식

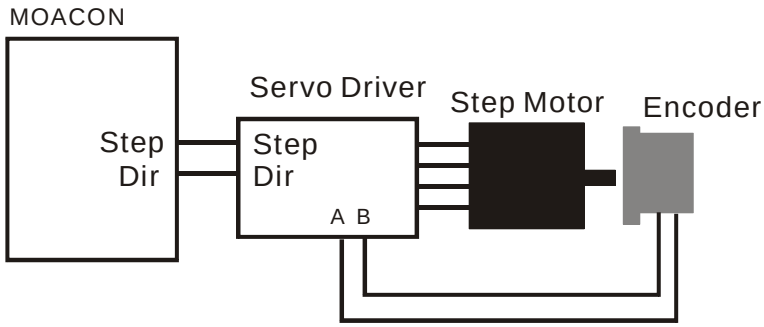
모터제어방식에는 오픈루프 (Open-loop)와 클로즈 루프(Close-Loop)방식이 있습니다. 오픈 루프방식은 피드백을 받지 않는 방식입니다. 따라서 모터가 원하는 회전수만큼 회전했는지 여부를 알 수 없습니다.

클로즈 루프 방식은 엔코더를 이용해 피드백을 받는 방식입니다. 따라서 모터가 원하는 회전수만큼 회전했는지 여부를 확인할 수 있습니다.



클로즈 루프를 구현하기 위해서 모아콘의 "고속카운터 모듈"을 사용할 수 있습니다만, 이 방식은 권장하지 않습니다. 모아콘이 다른일은 하지 않고, 모터의 움직임만 감시할 수는 없기 때문입니다.

따라서, 클로즈 루프방식으로 되어있는 (서보 방식) 스탭 모터 드라이버 사용을 추천합니다.



Ezi-Servo 제품을 추천합니다. (www.fastech.co.kr)

Ezi-Servo 드라이버가 자동적으로 위치보정을 해주므로, 모아콘에서는 엔코더입력을 받을 필요가 없습니다. (아래 모델은 Ezi-Servo st 입니다.)

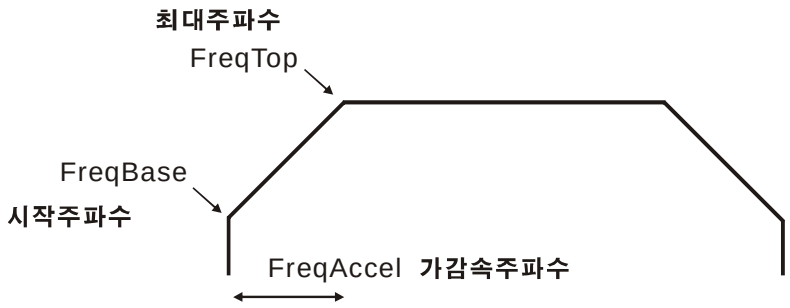
2 축 모션 제어 모듈 관련 라이브러리

motorSetup

void motorSetup (u8 motionID, u8 stCh, u32 freqBase, u32 freqTop, u32 stepAccel)

motionID : 모션제어 모듈의 ID (0 부터 9 까지)
stCh : 채널번호 (0 또는 1 만 사용가능)
freqBase: 시작 주파수 (최소 280 부터 사용가능)
freqTop: 최대 주파수 (최대 50,000 까지 사용가능)
freqAccel :가감속 주파수

모터의 움직임에 대한 기본 설정치 (시작주파수, 최대주파수, 가감속 주파수)를 셋팅하는 함수입니다. 이 함수는 셋팅만 바꿀뿐 아무 동작도 하지 않는 함수입니다.



motorMove

void motorMove (u8 motionID, u8 stCh, u32 position)

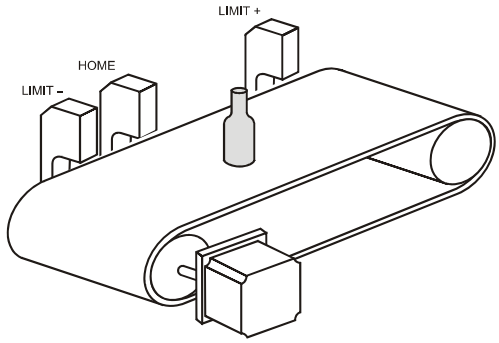
motionID : 모션제어 모듈의 ID (0 부터 9 까지)
stCh : 채널번호 (0 또는 1 만 사용가능)
position: 새로운 위치

모터를 실질적으로 움직이게 해주는 함수입니다. Step 포트로 펄스가 출력되고, dir 포트로 방향신호가 출력됩니다.
현재위치가 0 일경우, 새로운 위치를 1000 으로 지정한다면 그 만큼 회전하게 됩니다.
현재위치를 0 으로 만들기 위해서는 setMotorPos 함수를 사용하시면 됩니다.

setMotorPos

void setMotorPos (u8 motionID, u8 stChannel , u32 position)
motionID : 모션제어 모듈의 ID (0 부터 9 까지)
stChannel : 채널번호
position : 위치

모터의 위치를 세팅하는 함수입니다. 보통은 시스템이 시작할 때 원점위치로 모터를 회전시켜 원점을 잡는 작업을 하게 됩니다. (잉크젯 프린터의 헤드를 생각하시면 이해가 쉽습니다.) 원점의 근접센서에서 모터이동체가 원점까지 도달했는지 여부를 판단한다음, 이 지점의 위치를 0 으로 셋팅하십시오. 그리고 그 다음부터 motorMove 함수를 이용하여 원하는 위치로 이동하는 방법으로 사용합니다. 이 함수는 Power On 후 최소한 한번은 실행해주어야 합니다.



getMotorPos

u32 getMotorPos (u8 motionID, u8 stChannel)
motionID : 모션제어 모듈의 ID (0 부터 9 까지)
stChannel : 채널번호

모터의 위치를 읽어오는 함수입니다.

motorStop

void motorStop (u8 motionID, u8 stChannel)
motionID : 모션제어 모듈의 ID (0 부터 9 까지)
stChannel : 채널번호

출력중인 펄스를 강제로 종료시키는 함수입니다.

motorStat

u32 motorStat (u8 motionID, u8 stChannel)
motionID : 모션제어 모듈의 ID (0 부터 9 까지)
stChannel : 채널번호

현재, 출력중인 펄스의 개수를 알려주는 함수 입니다. 펄스출력이 모두 끝났다면 0 을 반환합니다.

MotorMove 함수를 실행시키면, 펄스출력은 모션제어모듈에서 알아서 출력합니다. CPU 가 펄스출력이 종료될때까지 기다리지 않습니다. 따라서 언제 펄스출력이 끝났는지 알아보기 위해서 motorStat 함수를 사용해야 합니다.

모션제어 예제 프로그램입니다.

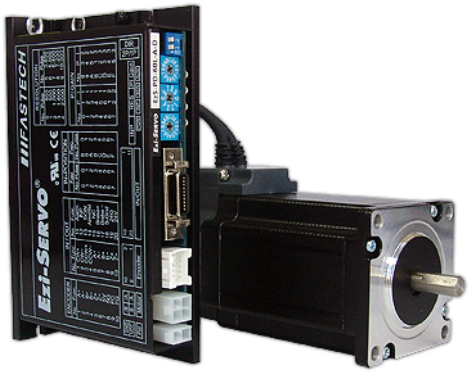
```
#include "moacon500.h"
void statprint(u32 i)
{
  u32 j;
  clcdPrint(0,0,"STAT:%09d ",i);
  delay(20);
  j = getMotorPos(0,0);
  clcdPrint(0,1,"POS: %09d ",j);
  delay(20);
}

void cmain(void)
{
  int i=0;
  clcdI2cInit(0);
   clcdPower(1);  // lcd 의 Power 를 On
  delay(100);
  clcdCls();
  clcdCsr(0);
   delay(2000); // 드라이버가 초기화될때까지 잠시 대기 (2 초)
   setMotorPos(0,0,0); // 현재위치를 0 으로 셋팅
   while(1) {
     motorSetup(0,0,500,50000,50000); // 모터초기정보 셋팅 (50KHz 출력)
     motorMove(0,0,1640000); // 1640000 위치로 이동
     while (i=motorStat(0,0)) {
       statprint(i);           // 이동하는 동안 정보를 lcd 상에 표시
       }
     delay(500);    // 이동이 종료됨
     i=motorStat(0,0);
```

```
    statprint(i);
    delay(3000);
    motorMove(0,0,0);  // 원래위치로 다시 이동
    while (i=motorStat(0,0)) {
      statprint(i);
      }
    delay(500);
    i=motorStat(0,0);
    statprint(i);
    delay(3000);
    }
}
```

이 프로그램은 스탭모터를 회전시키면서 회전상태를 CLCD 상에 회전수 (STAT)와 위치(POS)를 표시합니다.

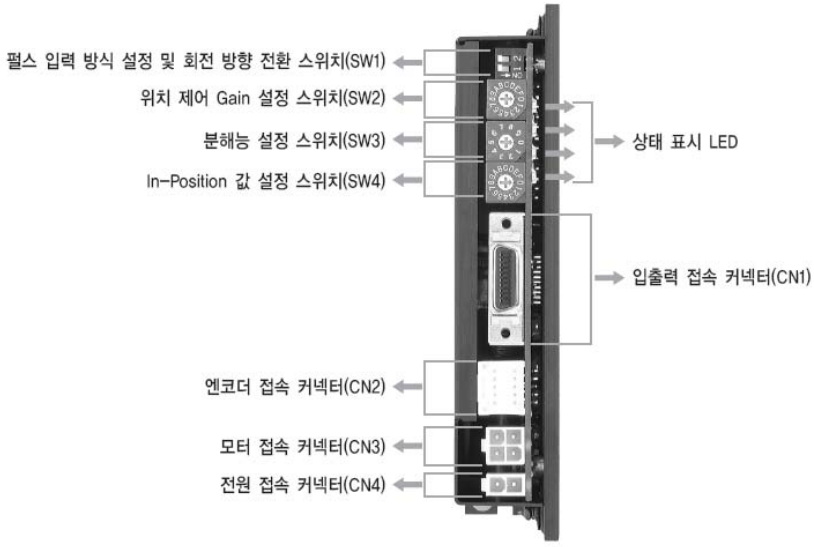
드라이버와 결선예

Ezi-Servo 제품군은 클로즈 루프 방식의 스태핑 서보 모터입니다. 스탭 모터와 드라이버가 한세트로 구성되어 있습니다.

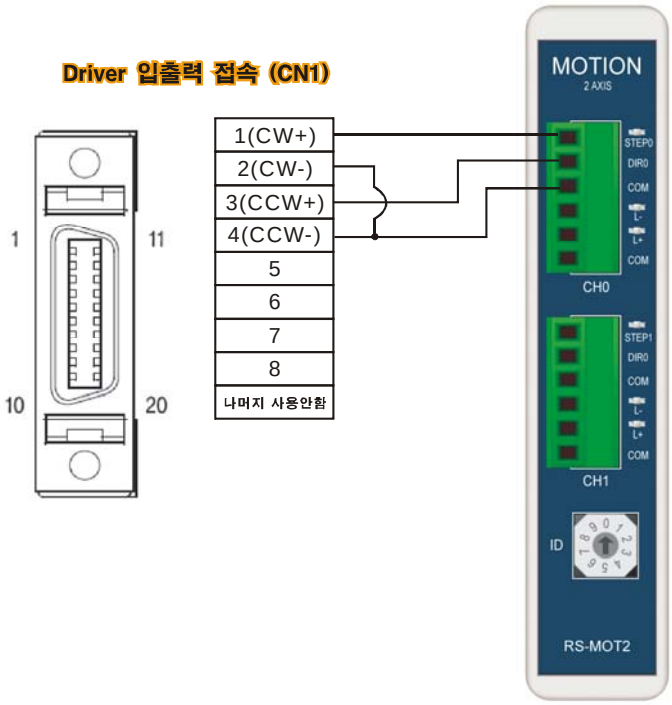


스탭 모터에 엔코더가 부착되어 있으며, 별도로 엔코더를 설치해야되는 불편함이 없습니다. 드라이버에서 엔코더의 상태를 읽어와 현재위치를 항상 감시하고 있으므로, 탈조가 발생하지 않고, 정확한 위치를 보장해줍니다.

모아콘을 사용하기 위해서는 드라이버의 SW1 의 1 번 스위치를 ON (1 펄스 방식)으로 하십시오.



입출력 접속커넥터 (CN1)중 1,2,3,4 번 핀을 모아콘 모션콘트롤러과 아래 그림과 같이 결선하십시오. (20 핀중 나머지 핀은 연결할 필요가 없습니다.)



전원 접속 커넥터 (CN4)에는 24V 를 인가하시고, 모터접속 커넥터 (CN3)는 모터와 접속하시고, 엔코더 접속 커넥터(CN2)는 엔코더와 접속하십시오.

제 8 장

아날로그모듈

AD 입력 모듈

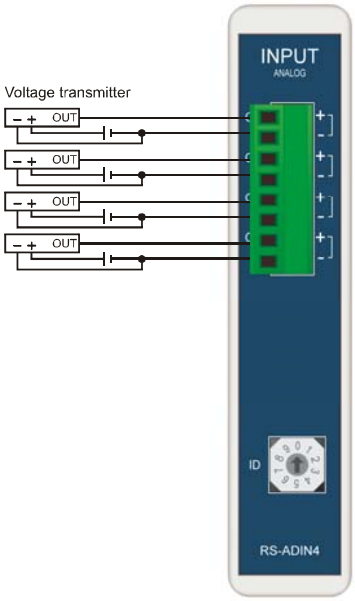
3 종류의 AD 입력 모듈이 있습니다.

모델명	출력값	해상도	정밀도 (오차율) (온도 25 도 시)
RS-SADIN6	0 부터 4095 사이의 값	12 비트	±1%
RS-ADIN4	0 부터 10,000 사이의 값	13.3 비트	±0.1%
RS-HADIN4	0 부터 100,000 사이의 값	16.6 비트	±0.1%

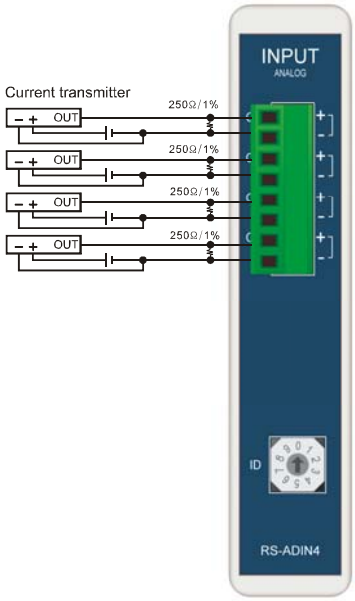
RS-ADIN4 , RS-HADIN4 사양

전류 4~20mA 연결시에는 Dip 스위치를 1~5V 모드로 설정하고, 250 오옴 (1% 1W)저항을 +입력과 –입력 사이에 연결해주어야 합니다.

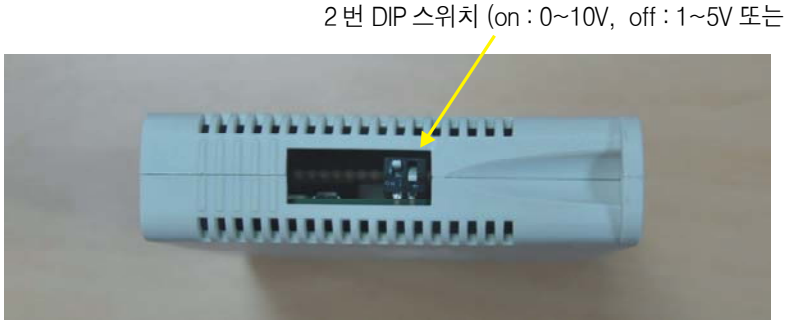
전압 입력시 결선도



전류 입력시 결선도



2 번 DIP 스위치를 조정하면 입력전압의 범위를 변경할 수 있습니다. (1 번 DIP 스위치는 사용하지 않습니다.)

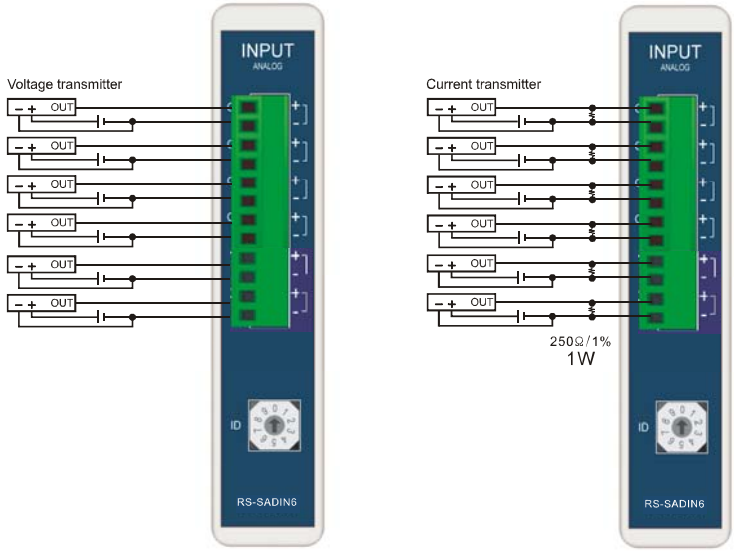


항목	설명
사용온도	-10 도 ~ 50 도 (단, 결빙되어 있지 않은 상태)
사용습도	35 ~ 85%RH
입력저항	590K 오옴
최대 입력가능 전압	1~5V 모드시 : 0.5 ~ 5.5V (이 범위초과시 파손위험있음) 0~10V 모드시: -0.5 ~ 10.5V (이 범위 초과시 파손위험있음)
통신방식	RS485
1 채널 변환속도	RS-ADIN4 는 30 밀리초, RS-HADIN4 는 240 밀리초
절연방식	비절연방식

AD 입력모듈은 비절연방식이므로 외부로부터 과전압 /과전류가 유입되지 않도록 주의하셔야 합니다. 만약 과전압 / 과전류 유입된다면, 제품에 손상이 가해질 수 있습니다.

RS-SADIN6 사양

RS-SADIN6 는 12 비트 고속 AD 입력 모듈입니다. AD 변환속도는 다른 AD 모듈에 비해서 속도는 빠르지만 정밀도는 다소 떨어집니다.



전류 0~20mA 연결시에는 250 오옴 (1% 1W)저항을 +입력과 –입력 사이에 연결해주어야 합니다. 별도의 딥스위치는 없습니다.

6 개 채널을 모두 읽어오는데 6 밀리초정도가 소요됩니다.
RS-SADIN6 는 DIP 스위치를 조정이 필요없습니다. 기본적으로 0 부터 5V 까지 입력가능하며, 0~20mA 입력시는 250 옴 (1%오차)저항을 +와 –사이에 연결하면됩니다.

항목	설명
사용온도	-10 도 ~ 50 도 (단, 결빙되어 있지 않은 상태)
사용습도	35 ~ 85%RH
0~5V 입력저항	450K 오옴
0~20mA 입력저항	250 오옴 (입력단자에 250 오옴 1% 1W 저항 연결요망)
최대 입력가능 전압	0~7V (이 범위 초과시 파손위험있음)
통신방식	RS485
6 채널 변환속도	6 밀리초
절연방식	비절연방식

AD 입력모듈 관련 라이브러리

getAdc

int getAdc (u8 adcId, u8 adcCh)
adcId : ADC 모듈의 ID (0 부터 9 사이의 값)
adcCh : 읽어올 채널 (1 부터 4 사이의 값)

getAdc 는 RS-ADIN4 를 위한 함수입니다. adcId 는 AD 입력모듈 전면부에 있는 ID 라고 적힌 로터리스위치를 뜻합니다. AD 입력모듈에는 4 개의 입력채널이 있으며 이중 측정하고자 하는 채널을 adcCh 에 써줍니다. ID 번호가 틀린경우, 혹은 모듈과 통신중 에러가 발생한 경우에는 –1 을 리턴합니다.

2 번 DIP 스위치	출력값	예외의 값
ON (0 ~ 10V)	0 부터 10,000 사이의 값	입력 open 시 310~ 320 사이값 (10 진수). 실제로도 310~320 사이 값이 나올수 있으므로,결과적으로 이 모드에선 입력오픈체크가 불가능.
OFF (1 ~ 5V)	0 부터 10,000 사이의 값	0.8V 미만의 값일경우 11,111 (10 진수) 5.2V 이상의 값일 경우 55,555 (10 진수) 입력 open 시 11,111 (10 진수)

getHadc

int getHadc (u8 adcId, u8 adcCh)
adcId : ADC 모듈의 ID (0 부터 9 사이의 값)
adcCh : 읽어올 채널 (1 부터 4 사이의 값)

getHAdc 는 RS-HADIN4 를 위한 함수입니다. ID 번호가 틀린경우, 혹은 모듈과 통신중 에러가 발생한 경우에는 –1 을 리턴합니다.

2 번 DIP 스위치	출력값	예외의 값
ON (0 ~ 10V)	0 부터 100,000 사이의 값	입력 open 시 3100~ 3200 사이값 (10 진수) 실제로도 3100~3200 사이 값이 나올수 있으므로,결과적으로 이 모드에선 입력오픈체크가 불가능.
OFF (1 ~ 5V)	0 부터 100,000 사이의 값	0.8V 미만의 값일경우 111,111 (10 진수) 5.2V 이상의 값일 경우 555,555 (10 진수) 입력 open 시 111,111 (10 진수)

```
#include "moacon500.h"
void cmain(void)
{
  int i;
  short j;
  clcdI2cInit(0); // 슬레이브 어드레스는 0 으로 합니다.
  clcdPower(1);  // lcd 의 Power 를 On
  delay(100); // clcd 기동시간 대기
  clcdCls();
  clcdCsr(0);
   delay(900); // ADC 모듈 최초 샘플링시간 대기
   comPower(1); // RS232 파워를 On 한뒤 여기서 5V 를 공급받습니다.
   while(1) {
     i = getAdc(2,1);
     clcdPrint(0,0,"ADC:%06d",i);

     delay(100);
     }
 }
```

getSadc

int getSadc (u8 adcId, int * array)
adcId : ADC 모듈의 ID (0 부터 9 사이의 값)
array : 결과를 저장할 배열의 이름

getSAdc 는 RS-SADIN6 를 위한 함수입니다.

GetSadc 는 앞에서 설명한 getAdc, getHadc 와 사용법이 다릅니다. 6 개의 채널값을 한번에 모두 읽어와서 지정된 배열에 저장합니다. 이 배열은 사전에 선언해두어야합니다. Int 형으로 6 개의 공간이 필요합니다.

```
int sAdcData[6];
aj = getSadc(0,sAdcData);
```

ID 번호가 틀린경우, 혹은 모듈과 통신중 에러가 발생한 경우에는 –1 을 리턴하고, 성공적으로 수행했을경우에는 0 을 리턴합니다.

```
#include "moacon500.h"
void cmain(void)
{
  int sAdcData[6];
  u16 aj;
  while (1) {
        aj = getSadc(0,sAdcData);
        delay(500);
        printf("sadin ch1 = %x \r\n",sAdcData[0]);
        printf("sadin ch2 = %x \r\n",sAdcData[1]);
        printf("sadin ch3 = %x \r\n",sAdcData[2]);
        printf("sadin ch4 = %x \r\n",sAdcData[3]);
        printf("sadin ch5 = %x \r\n",sAdcData[4]);
        printf("sadin ch6 = %x \r\n",sAdcData[5]);
        }
}
```

위의 예제의 경우, 채널 1 의 값은 배열 sAdcData[0] 에 저장됩니다.
채널 6 은 sAdcData[5]에 저장됩니다.

AD 변환속도

RS-ADIN4 모델의 경우 채널당 샘플링시간은 30 밀리초입니다. 4 개 채널을 모두 샘플링하는데 걸리는 시간은 120 밀리초입니다. 따라서 120 밀리초보다 빠른 간격으로 getAdc 함수를 수행하는 것은 무의미한 일입니다. 같은 값만 계속 반환하게 됩니다.

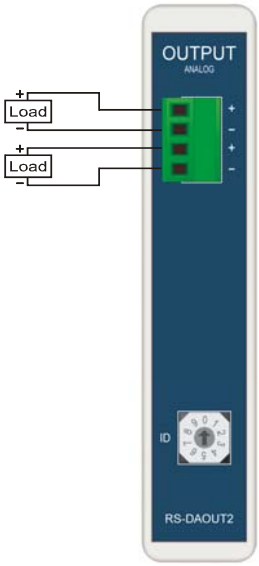
RS-HADIN4 모델의 경우 채널당 240 밀리초, 4 개 채널을 모두 샘플링하는데 걸리는 시간은 960 밀리초 입니다.

RS-SADIN6 모델의 샘플링시간은 채널당 수십 마이크로초단위로 매우 빠릅니다,그리고 한번에 6 개의 데이터를 모두 읽어오기 때문에 보다 빠르게 데이터를 수집할 수 있습니다. getSadc 함수 수행시간이 약 6 밀리초입니다. SADIN6 는 샘플링 시간을 기다릴 필요가 없습니다.

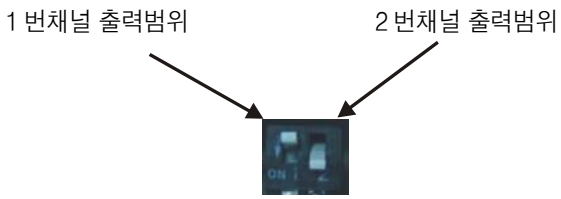
-RS-ADIN4 는 파워온후 120 밀리초 이후에 사용해야합니다.
-RS-HADIN4 는 파워온후 약 1 초뒤에 사용해야 합니다.
-RS-SADIN6 는 파워온후 즉시 사용가능합니다..

전압 DA 출력모듈

2 채널, 0~5V 또는 0~10V 출력이 가능한 D/A 컨버터 모듈입니다. 하단 딥스위치를 조정하여 출력범위를 결정합니다.



딥스위치 설정



Off : 0 부터 10V 출력

On : 0 부터 5V 출력

RS-DAOUT2 상세사양

항목	설명
아날로그 출력	전압 : DC 0~5V 또는 0~10V (부하저항 1K 옴 이상)
사용온도	-10 도 ~ 50 도 (단, 결빙되어 있지 않은 상태)
사용습도	35 ~ 85%RH
출력 분해능	0 부터 60000 까지 가능
통신방식	RS485
최대 변환속도	0 에서 60000 까지 변환속도 600mS
절연방식	비절연방식

전압 DA 출력모듈 관련 라이브러리

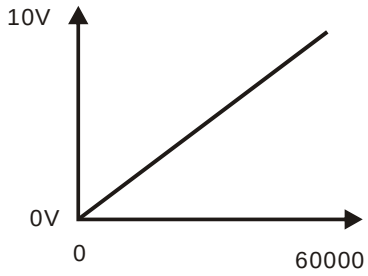
dacOut

void dacOut (u8 dacId, u8 dacCh, u16 daValue)
dacId : DAC 모듈의 ID (0 부터 9 사이의 값)
dacCh : 출력할 채널 (1 또는 2)
daValue : 출력할 DA 값 (0 부터 60000)

dacId 는 DA 출력모듈 전면부에 있는 ID 라고 적힌 로터리 스위치를 뜻합니다. 0 부터 9 중 하나를 선택합니다. DA 출력모듈에는 2 개의 채널이 있으며 이중 하나를 선택하여 dacCh 에 써줍니다.

10 개의 DA 모듈을 하나의 시스템에서 사용할 수 있습니다. 따라서 총 20 개의 DA 출력이 가능합니다.

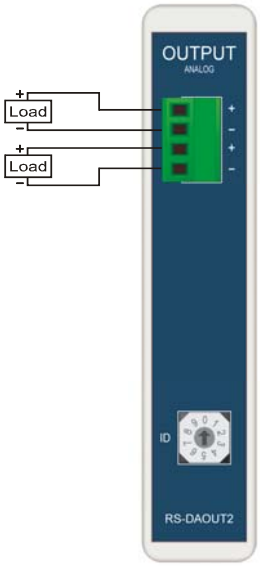
출력값은 0 부터 60000 사이의 값이 0 ~ 10V (또는 0~5V)로 출력됩니다.



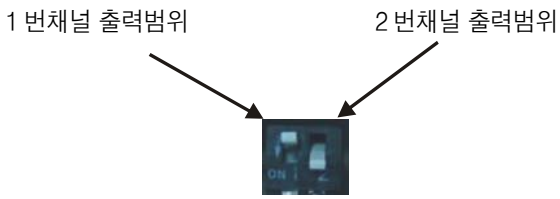
부하저항은 반드시 1K 오옴 이상을 사용해야 합니다.

전류 DA 출력모듈

2 채널, 0~20mA 또는 4~20mA 출력이 가능한 D/A 컨버터 모듈입니다. 하단 딥스위치를 조정하여 출력범위를 결정합니다.



딥스위치 설정



Off : 4~20mA 출력

On : 0~20mA 출력

RS-DAOUT2B 상세사양

항목	설명
아날로그 출력	전류 : 0~20mA 또는 4~20mA (부하저항 600 옴 이하) 전류가 모듈로부터 출력됩니다.
사용온도	-10 도 ~ 50 도 (단, 결빙되어 있지 않은 상태)
사용습도	35 ~ 85%RH
출력 분해능	0 부터 60000 까지 가능
통신방식	RS485
최대 변환속도	0 에서 60000 까지 변환속도 600mS
절연방식	비절연방식

***주의 : 1 번채널과 2 번채널의 마이너스 단자를 서로 쇼트시키지 마십시오. 각각의 채널을 별개로 사용하시기 바랍니다.**

전류 DA 출력모듈 관련 라이브러리

dacOut2

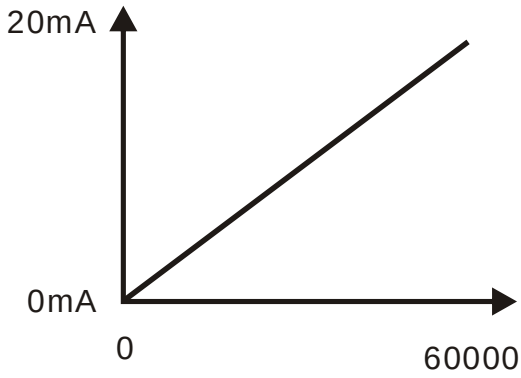
void dacOut2 (u8 dacId, u8 dacCh, u16 daValue)

dacId : DAC 모듈의 ID (0 부터 9 사이의 값)

dacCh : 출력할 채널 (1 또는 2)

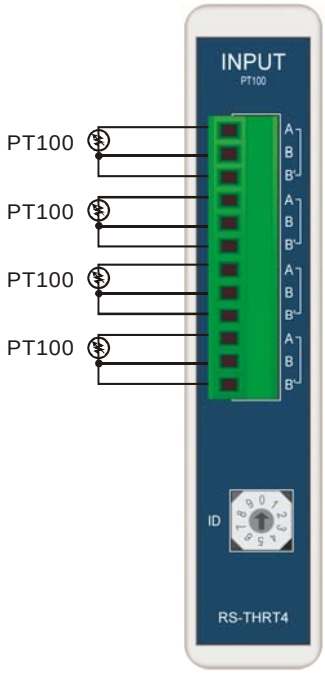
daValue : 출력할 DA 값 (0 부터 60000)

출력값은 0 부터 60000 사이의 값이 4~20mA (또는 0~20mA)로 출력됩니다.



온도입력모듈

PT100 옴 온도센서를 입력받을 수 있는 모듈입니다. 측정범위는 –100 도에서 500 도이며, ±0.5%의 정밀도를 가지고 있습니다.



RS-THRT4 상세사양

항목	설명
사용온도	–10 도 ~ 50 도 (단, 결빙되어 있지 않은 상태)
사용습도	35 ~ 85%RH
샘플링주기	200 밀리초 /1 채널
입력	RTD 센서 (PT100)
허용선로저항	선당 10 옴 이하 (단, 3 선간 저항은 동일할 것) *온도센서모듈부터 PT100 센서 지점까지의 연결선의 저항치
측정정도	±0.5% of F.S
통신방식	RS485
1 패킷 통신속도	3mS
절연방식	비절연방식

온도입력모듈 관련 라이브러리

getTemp

int getTemp (u8 tempId, u8 tempCh)
tempId : 온도모듈의 ID (0 부터 9 사이의 값)
tempCh : 읽어올 채널 (1 부터 4 사이의 값)

tempId 는 온도입력모듈 전면부에 있는 ID 라고 적힌 로터리 스위치를 의미합니다. tempCh 는 모듈에 있는 4 개 채널중 측정하고자하는 하나의 채널을 뜻합니다. 본 함수는 해당 채널에 연결된 PT100 온도 센서로부터 읽어온 –1000 부터 5000 사이의 값을 반환합니다.

원래는 소수점 밑으로 내려가야될 1 자릿수가 포함되어 있습니다. 즉, 20.5 도는 205 로 반환됩니다. 나누기 10 을 해야 실제 온도값이 됩니다. 소수점 아래 자릿수는 10 으로 나눈 나머지값으로 알 수 있습니다.

```
j = getTemp(2,1);
k = j / 10;   //온도값 정수부
m = j % 10;   //온도값 소수부
```

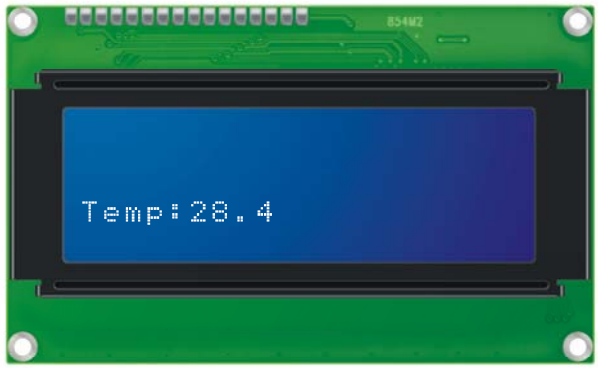
이렇게 하는 이유는 실수형 변수를 사용하지 않고 정수형 변수만으로 처리하기 위해서입니다. GetTemp 함수는 단지 소수점 아래 1 자리만 필요로 하므로, 10.2 의 경우 102 로 반환합니다. (정수형변수가 실행속도면에서 유리하기 때문입니다.)

만약 센서가 연결되어 있지 않거나, 접촉불량일 경우, 또는 측정범위를 넘어선 경우에는 다음과 같은 값이 반환됩니다.

예외상황 (10 진수)	설명
20001 ~ 20003	모듈과 통신중 에러가 발생된경우
20004	모듈이 없거나, 해당 ID 가 없는 경우
20005	센서가 연결되어 있지 않거나, 단선된 경우
55555	+영역을 초과한 경우
11111	–영역을 초과한 경우

```
#include "moacon500.h"
void cmain(void)
{
  int i;
  short j;
  clcdI2cInit(0); // 슬레이브 어드레스는 0 으로 합니다.
  clcdPower(1);  // lcd 의 Power 를 On
  delay(100); // clcd 기동시간 대기
  clcdCls();
  clcdCsr(0);
   delay(800); // 온도센서모듈 최초 샘플링시간 대기
   while(1) {

    j = getTemp(2,1);
    if (j > 9000)
      clcdPrint (0,1,"Temp Err:%d",j); // 에러상황 (단선, over range 등)
    else
      clcdPrint(0,2,"Temp:%d.%d",j/10,j%10);  // 100 번지에서 읽은값을 print

    delay(100);
    }
}
```

위 프로그램을 실행하면 LCD 에 온도값이 표시됩니다.

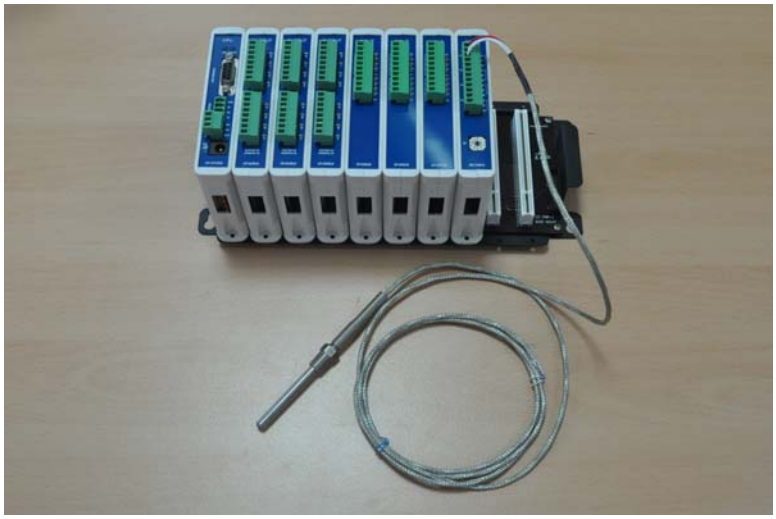


온도센서의 채널별 샘플링 주기는 200 밀리 초입니다. 온도센서 모듈은 항상 온도센싱을 하고 있다가 getTemp 함수가 실행되면, 측정해 놓은 값을 보내줍니다.

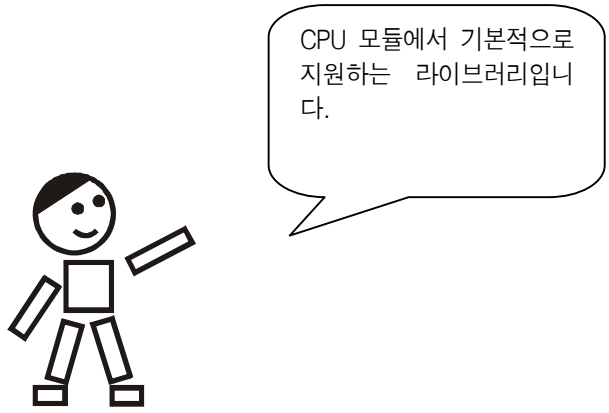
온도센서 모듈은 4 개의 채널을 가지고 있으므로 모든 채널을 한번 스캔하는데 800mS , 즉 0.8 초가 필요합니다. 따라서 0.8 초보다 빠른 주기로 getTemp 함수를 실행시키는 것은 의미없는 불필요한 동작입니다. GetTemp 함수의 실행시간은 대략 3mS 입니다.

최초 파워온시 모든 채널을 한번 스캔하는 데 걸리는 시간동안은 getTemp 함수를 실행하지 말아야 합니다. 아직 온도센싱을 하지 않았기 때문에 엉뚱한 값이 읽혀지게 됩니다.
파워온후 0.8 초 이후에 getTemp 함수를 사용하십시오.

다음은 PT100 센서를 온도모듈에 연결한 모습입니다.



제 9 장
시스템
라이브러리



시간지연함수

delay

void delay (u32 interval)

interval : 지연시간 (밀리초 단위)

일정시간을 지연시켜주는 함수 입니다. Interval 은 ms (밀리초)단위입니다. 즉 10 을 적어주면 10mS 를 딜레이합니다.

```
while (1) {              // 무한루프
   portOut(20,1);        // 20 번 포트를 High 상태로 만듭니다.
   delay(100);           // 100mS 지연
   portOut(20,0);        // 20 번 포트를 Low 상태로 만듭니다.
   delay(100);
   }
```

*delay 에서 대기하는 시간은 다소 오차가 있습니다. 정밀한 시간 측정과 같은 용도에는 적합하지 않습니다.

1 밀리 초보다 더 작은 시간을 DELAY 하는 함수는 따로 준비된 것이 없지만, 아래 함수를 추가하여 사용하시기 바랍니다.

```
void waitTime(vu32 nCountus)
{
    for (;nCountus > 0;nCountus--);
}
```

아무일도 하지 않고 for 루프를 일정시간 반복수행하면서 시간을 보내는 방식입니다.

상태 LED 함수

statusLed

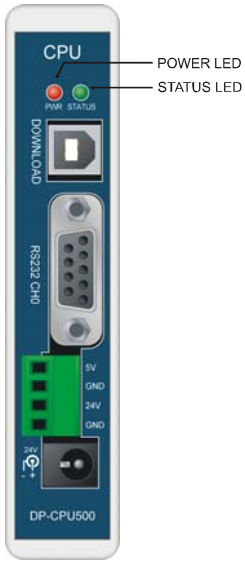
void statusLed (u8 onoff)

onoff : 0=off, 1=on

CPU 모듈 전면부에 있는 STATUS LED 를 제어하는 함수 입니다. 최초 전원 인가상태에서는 OFF 로 되어 있습니다.

STATUS LED 는 유저가 임의대로 사용하실 수 있는 LED 입니다. 동작 상태를 나타내는 용도로 사용하실 수 있습니다.

```
statusLed(1);      // STATUS LED 를 On 시킵니다.
```



리얼타임 관련 함수

MOACON 에는 정밀도가 높은 RTC 칩이 내장되어 있습니다. 이 칩은 온도센서를 포함하고 있어 온도변화에 따른 오차를 줄여줍니다. 또한 건전지가 내장되어 있어, 전원이 없는 상황에서도 시간이 계속 갱신됩니다. 건전지 교환주기는 10 년입니다. 교환시기가 되면, 모아콘 CPU 모듈을 분해한뒤 내부에 RTC 용 건전지(CR2032)를 교환해주시기 바랍니다.

다음은 RTC 칩의 어드레스별 데이터 정보입니다.

rtcAdr	Value	Range	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	Second	0 to 59		2nd digit place			1st digit place			
1	Minute	0 to 59		2nd digit place			1st digit place			
2	Hour	0 to 23			2nd digit place		1st digit place			
3	Day	1 to 7						1st digit place		
4	Date	1 to 31			2nd digit place		1st digit place			
5	Month	1 to 12				2nd digit	1st digit place			
6	Year	00 to 99	2nd digit place				1st digit place			

데이터는 BCD 코드형태로 기록됩니다. 즉 상위 4 비트와 하위 4 비트에 숫자 한자릿수씩 위치합니다. 요일의 경우 숫자로 기록되며, 다음과 같은 의미로 사용됩니다.

Sunday	1
Monday	2
Tuesday	3
Wednesday	4
Thursday	5
Friday	6
Saturday	7

Hour (시간)은 0 부터 23 까지 변화되므로 12 보다 클경우 오후, 작을경우 오전으로 판단하시면 됩니다.

*모아콘에 사용된 RTC 칩은 일반적인 RTC 칩보다는 정밀하지만, 실제 시간과 100% 일치는 보장할 수 없습니다. 이점 유의하시기 바랍니다.

rtcRead

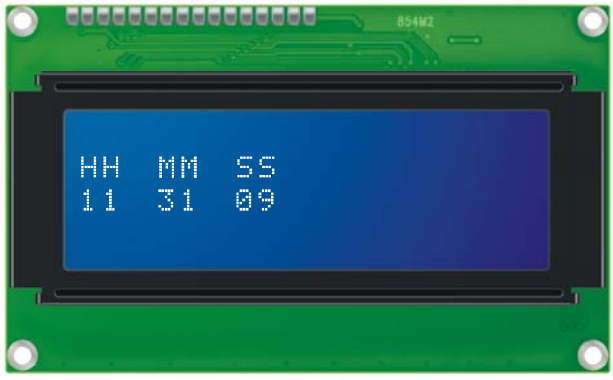
u8 rtcRead(u8 rtcAdr)

rtcAdr : 어드레스 (0 부터 6 까지 사용가능)

지정한 어드레스로부터 정보를 읽어옵니다.

다음 프로그램을 실행시키면 현재 시간 (시,분,초)를 clcd 화면상에 표시합니다.

```
#include "moacon500.h"
void cmain(void)
{
  int i=0;
  clcdI2cInit(0); // 슬레이브 어드레스는 0 으로 합니다.
  clcdPower(1);  // lcd 의 Power 를 On
   delay(100); // clcd 기동시간 대기
  clcdCls();
  clcdCsr(0);
  clcdPrint(0,1,"HH  MM  SS");
   while(1) {
    clcdPrint(0,2,"%02X",rtcRead(2));
    clcdPrint(4,2,"%02X",rtcRead(1));
    clcdPrint(8,2,"%02X",rtcRead(0));
    delay(500);
    }
 }
```



rtcWrite

void rtcWrite (u8 rtcAdr, u8 rtcData)

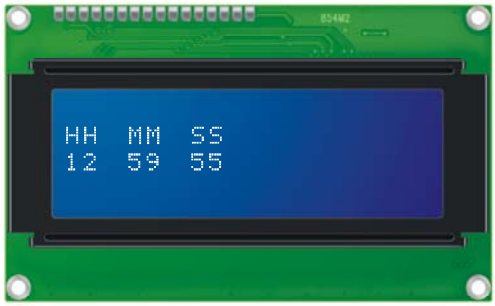
rtcAdr : 어드레스 (0 부터 6 까지 사용가능)

rtcData : 데이터

지정한 어드레스에 데이터를 기록합니다.

다음 프로그램을 실행시키면 12 시 59 분 55 초부터 표시를 시작합니다.

```
#include "moacon500.h"
void cmain(void)
{
  int i=0;
  clcdI2cInit(0); // 슬레이브 어드레스는 0 으로 합니다.
  clcdPower(1);  // lcd 의 Power 를 On
   delay(100); // clcd 기동시간 대기
  clcdCls();
  clcdCsr(0);
  clcdPrint(0,1,"HH  MM  SS");
  rtcWrite(0,0x55); // sec write
  rtcWrite(1,0x59); // min write
  rtcWrite(2,0x12); // hour write
   while(1) {
     clcdPrint(0,2,"%02X",rtcRead(2));
     clcdPrint(4,2,"%02X",rtcRead(1));
     clcdPrint(8,2,"%02X",rtcRead(0));
     delay(500);
     }

}
```



FRAM 관련 함수

FRAM 은 EEPROM 의 단점을 개선한 비휘발성 메모리입니다. FRAM 에 저장된 데이터는 전원이 없어도 보존됩니다. 모아콘에는 32KB 의 FRAM 이 내장되어 있습니다.

32KB 에서의 사용가능 어드레스 : 0 ~ 0x7FFF

framWrite

void framWrite (u16 fAdr, u8 fData)

fAdr : 어드레스 (16 진수 0x7fff 이내의 값)
fData : 데이터 (8 비트)

FRAM 의 특정번지에 1 바이트 데이터를 기록하는 함수입니다.

framRead

u8 framRead (u16 fAdr)

fAdr : 어드레스 (16 진수 0x7fff 이내의 값)

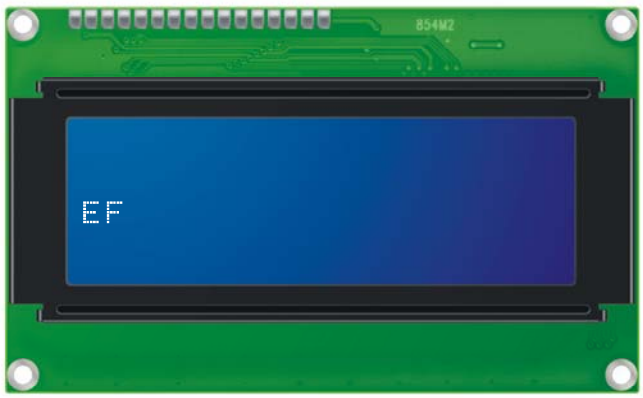
FRAM 의 특정번지에서 1 바이트 데이터를 읽어오는 함수입니다.

다음 소스프로그램은 fram 의 100 번지에 1 씩증가하는 데이터를 기록하고, clcd 상에 표시해주는 프로그램입니다.

이 프로그램을 실행시키면 clcd 상에 증가되는 숫자가 보입니다. 전원을 끄고, 다시키면 좀전에 증가되던 수치에서 계속 증가됩니다. Fram 이 전원이 없는상태에서도 데이터값을 유지하고 있기 때문입니다.

```
#include "moacon500.h"
void cmain(void)
{
  u8 i;
  clcdI2cInit(0); // 슬레이브 어드레스는 0 으로 합니다.
  clcdPower(1);  // lcd 의 Power 를 On
  delay(100); // clcd 기동시간 대기
  clcdCls();
```

```
   clcdCsr(0);
   while(1) {
     i = framRead(100);
     clcdPrint(0,2,"%02X",i);  // 100 번지에서 읽은값을 print
     framWrite(100,++i);
     delay(500); // fram 기록을 기다리기위한 대기시간이 아닙니다.
                 // Lcd 표시를 0.5 초마다 하기위한 딜레이입니다.
                 // framWrite 는 별도의 대기시간이 필요하지 않습니다.
     }

}
```



잠깐만..

갑작스러운 정전에 대비해서 중요한 값을 보관해야 될 필요성이 간혹 발생합니다. 이를 위해서 전통적으로 사용하는 방법이 밧데리 백업이나 EEPROM 을 사용하는 방법이 있습니다.

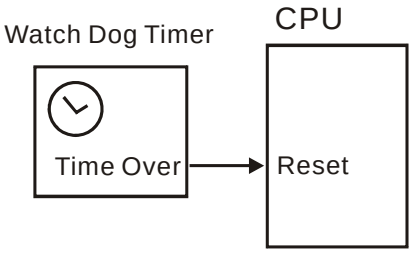
밧데리 백업은 별도의 밧데리를 사용해서 내부 SRAM 을 정전시에도 보존시켜주는 방법인데, 교체시기를 알리기 위해, 밧데리 잔량을 체크해서 외부로 알려주어야 하고, 운영시 신경을 써주어야하는 부담이 있습니다.

EEPROM 은 1 바이트 라이팅할때마다 수밀리초씩 대기시간이 발생하고, 사용 횟수도 백만회 정도로 제한되어 있습니다. 이에 반해 FRAM 은 밧데리 교체가 필요없이 한번 저장된 데이터는 10 년동안 지워지지 않습니다. 그리고 라이팅시 대기시간도 전혀없으며 사용 횟수 또한 제한이 없습니다.

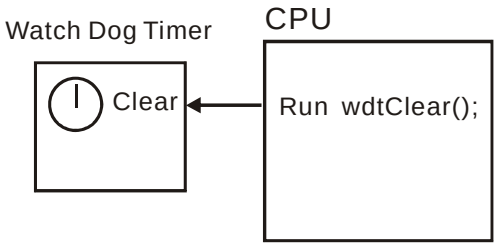
와치독 타이머

와치독 타이머는 프로그램이 아무일도 하지 않는 멀펑션 (mulfunction)에 빠지는 것을 막기위해 사용합니다.

와치독타이머는 외부에서 CPU 를 감시하도록 설계되어 있습니다. 0 부터 증가하기 시작해서 Time Over 되면 메인 CPU 를 리셋 시킵니다.



프로그램이 정상적으로 수행되기 위해서는, 와치독 타이머가 Time Over 가 되기전에 클리어 시켜 주어야 합니다.



만약 프로그램이 엉뚱한 곳으로 빠져서 탈출하지 못한다면, 그 곳에는 와치독 클리어 명령이 없으므로, 와치독 타이머가 Time Over 가 되어 메인 CPU 를 리셋시켜줍니다.

소스의 가장 첫부분에는 와치독 타이머를 사용할 것인지의 여부를 결정짓는 wdtOn( )함수가 필요합니다.

wdtOn

void wdtOn (u8 interval)

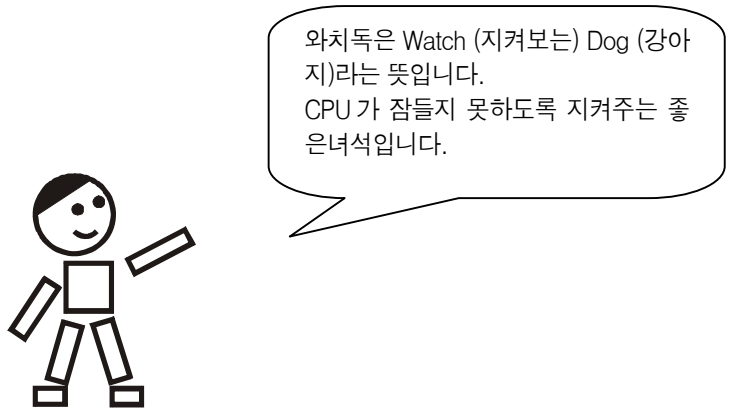
interval : 와치독 타이머의 리셋주기 (0 ~ 6 사이의 값)

wdtOn 함수를 사용하기 전에는 와치독 타이머가 동작하지 않습니다. WdtOn 함수는 와치독 타이머를 On 시켜주면서, 동시에 동작주기도 함께 결정짓습니다.

Interval 값	타임아웃 주기
0	약 0.4 초
1	약 0.8 초
2	약 1.6 초
3	약 3.2 초
4	약 6.5 초
5	약 13 초
6	약 26 초

* 타임아웃 시간은 주변 온도에따라 변동될 수 있습니다. (±5%)

wdtOn(2); 이 함수를 소스의 가장 첫머리에 써넣어주면, 와치독 타이머는 1.6 초주기로 메인칩을 리셋시키려고 합니다. 그전에 와치독 타이머를 클리어해주어야 메인칩이 리셋되는 것을 막을 수 있습니다.



wdtClear

void wdtClear (void)

wdtClear( )함수를 실행시키면 와치독타이머가 0 으로 리셋됩니다. 처음부터 다시 카운트업을 시작합니다.

와치독 타이머 사용시에는 Time Over 주기보다 빠르게 와치독 타이머를 클리어해주어야 합니다. 그렇게 하지 않으면 프로그램은 주기적으로 리셋됩니다.

```
#include "moacon500.h"
void cmain(void)
{
   wdtOn(2); //1.6 초간격으로 와치독 타이머 동작개시
   int i=0;
   printf ("Reset\r\n");
   while (1) {
         wdtClear(); // 와치독 타이머 클리어 (리셋방지)
         printf ("comfile %d\r\n",i++);
         delay(200);
         }
}
```

메인 루틴뿐만 아니라, 시간을 소비하는 모든 루틴, 함수마다 wdtClear( ) 함수를 넣어줘야 합니다.

주기적으로 wdtClear( ) 를 실행시킬 자신이 없다면 와치독 타이머를 사용하지 않는 것이 좋습니다. 정상적인 수행상태에서 프로그램이 주기적으로 리셋된다면 와치독 타이머 클리어를 제대로 수행하지 못한 경우입니다.

제 10 장

이벤트

타이머 이벤트 관련 함수

타이머 이벤트는 주기적으로 타이머 이벤트 함수를 실행시켜주는 기능입니다. 최소 1mS 간격부터 65 초 간격까지 사용가능합니다.

startTimerEvent

void startTimerEvent (u16 interval)

interval : 타이머 이벤트 간격 (mS 단위, 1 부터 65535 까지 사용가능 ; =65.5 초)

이 함수를 사용하여 타이머 이벤트를 시작합니다. Interval 은 이벤트를 발생시킬 시간 간격을 의미합니다. 1 일경우 1mS 마다 timerEvent 함수를 실행시킵니다.
10 일경우에는 10mS 마다 timerEvent 함수를 실행시킵니다.

이 기능은 매우 유용하게 쓰입니다. 백그라운드에서 I/O 처리를 해준다거나, 타이밍을 측정하기 위해 사용하기도 합니다. 마치 여러 개의 프로세서가 동시에 돌아가는 것처럼 보이도록 하는 멀티테스킹 처리도, 타이머 이벤트를 이용해서 처리할 수 있습니다.

타이머 이벤트 사용시 주의사항은, 이벤트 처리루틴이 **반드시 타이머 이벤트 발생주기보다 빠르게 실행을 마쳐야 한다는 것입니다.** 예를 들어 이벤트는 10mS 마다 발생하는데 이벤트 처리루틴이 100mS 이상 걸린다면, 이 프로그램은 계속해서 이벤트 루틴만 실행하게 되고, 메인루틴은 실행하지 못하게 됩니다.

timerEvent

void timerEvent (void)

타이머 이벤트 처리를 담당하는 함수입니다. startTimerEvent 에서 설정한 시간간격으로 timerEvent 함수를 실행시킵니다.

```
#include "moacon500.h"
int tm;
#define delaytime 250
void cmain(void)
{
  short i,j;
  startTimerEvent(100);  // 타이머 이벤트를 기동시킵니다.
  portInit(6,0);
   while (1) {
         portOff(61);
         delay(delaytime);
         portOn(61);
         delay(delaytime);
         }
}

// 100mS 마다 아래 함수를 실행합니다.

void timerEvent(void)
{
     printf("comfile tech %d \r\n",tm++);
}
```

stopTimerEvent

void stopTimerEvent (void)

타이머 이벤트를 영구히 종료합니다.

disableTimerEvent

void enableTimerEvent (void)

타이머 이벤트 실행을 임시 중단합니다. 내부적으로 시간카운트는 계속 진행됩니다. 타이머 이벤트 함수 실행만 하지 않는 것입니다.

```
#include "moacon500.h"
int tm;
void cmain(void)
{
  tm = 0;
  startTimerEvent(100);
  portInit(3,0);
  while(1)
  {
    portOn(30);
    delay(200);
    portOff(30);
    delay(200);
    if (portIn(0) == 1)
      disableTimerEvent();  // 입력이 들어오면 타이머 이벤트를 임시로 중단합니다.
    else
      enableTimerEvent();
  }
}

void timerEvent(void)
{
  portReverse(31);
}
```

enableTimerEvent

void enableTimerEvent (void)

임시로 중단했던 타이머 이벤트를 다시 사용가능한 상태로 만듭니다.

특정 함수실행중 타이머 이벤트 실행을 허용하고 싶지 않다면, 해당함수를 disableTimerEvent 와 enableTimerEvent 로 감싸면 됩니다.

```
disableTimerEvent();  // 입력이 들어오면 타이머 이벤트를 임시로 중단합니다.
AdSampleInput(); // 이 함수 실행중 타이머 이벤트가 발생되며 안됩니다.
enableTimerEvent(); // 다시 타이머 이벤트를 사용가능하도록 만듭니다.
```

외부 인터럽트 이벤트 관련 함수

startExtIntEvent

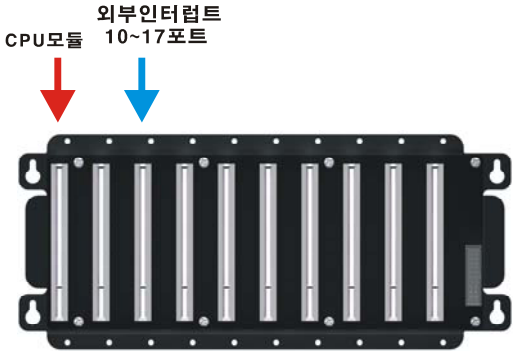
void startExtIntEvent (u8 extIntPort, u8 extIntKind)

extIntPort : 외부 인터럽트를 받을 포트 (10 부터 17 포트까지 사용가능)
extIntKind : 0= ON 시점, 1=OFF 시점, 2=ON/OFF 시점 모두

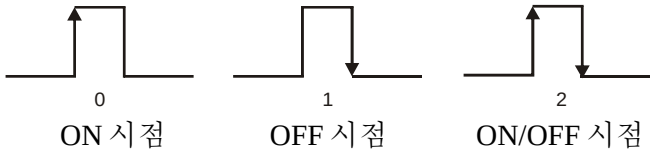
이 함수를 사용하여 특정포트의 외부 인터럽트 이벤트를 시작합니다. 외부 인터럽트는 돌발적인 입력에 즉각적으로 대처하기 위해서 사용합니다.

포트 10 번부터 17 번을 외부 인터럽트 입력으로 사용할 수 있습니다.

따라서, 이 기능을 사용하기 위해서는 "기본 DIO 모듈"중 입력모듈을 +10 슬롯에 장착해야 합니다.



ExtIntKind 값을 어떻게 정하느냐에 따라서 이벤트를 검출하는 위치가 결정됩니다.



이벤트가 발생되면 extIntEvent 함수가 실행됩니다.

stopExtIntEvent

void stopExtIntEvent (u8 extIntPort)

extIntPort : 외부 인터럽트 발생을 종료시킬 포트 (10 부터 17 포트까지 사용가능)

해당 포트의 외부 인터럽트 이벤트를 종료합니다.

extIntEvent

void extIntEvent(u8 extIntPort)

extIntPort : 외부 인터럽트가 발생된 포트번호 (10 부터 17 포트까지 사용가능)

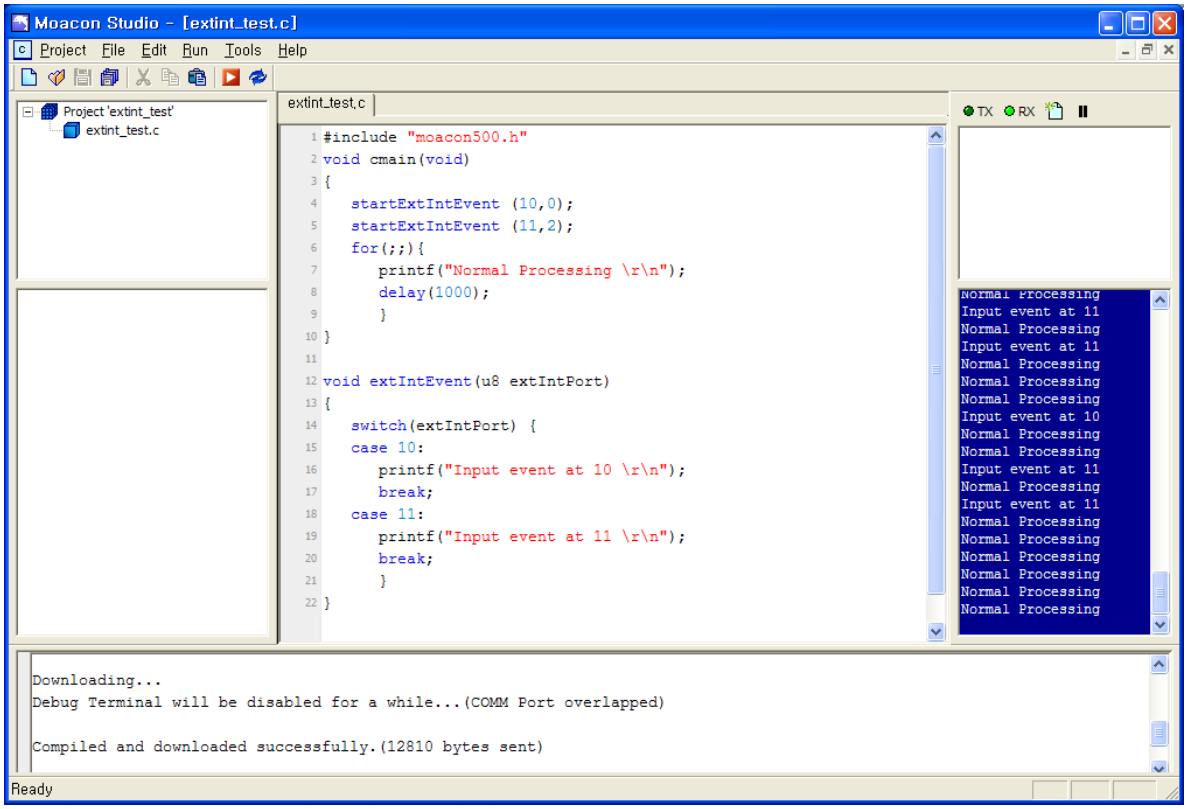
이벤트가 발생되면 extIntEvent 함수가 실행됩니다. 이때 인터럽트 입력이 발생된 포트번호를 인수로 전달받게 됩니다.

다음은 외부 인터럽트 테스트 소스입니다. 포트 10 과 11 에 외부 인터럽트 입력시 디버그창에 표시를 해줍니다.

```
#include "moacon500.h"
void cmain(void)
{
  startExtIntEvent (10,0); // on 시점
  startExtIntEvent (11,2); // on/off 시점
  for(;;){
        printf("Normal Processing \r\n");
        delay(1000);
        }
}

void extIntEvent(u8 extIntPort)
{
  switch(extIntPort) {
  case 10:
        printf("Input event at 10 \r\n");
        break;
  case 11:
        printf("Input event at 11 \r\n");
        break;
        }
}
```

이 소스를 실행시키고 포트 10 번을 ON 할때마다 디버그 창에 “Input event at 10”라고 표시됩니다.

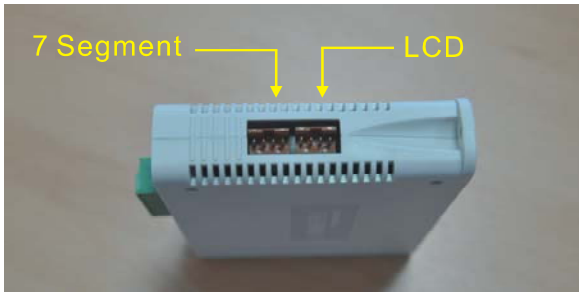
포트 11 번을 ON 또는 OFF 할때마다 디버그 창에 “Input event at 11”이라고 표시됩니다.



제 11 장
디스플레이
라이브러리

디스플레이 연결

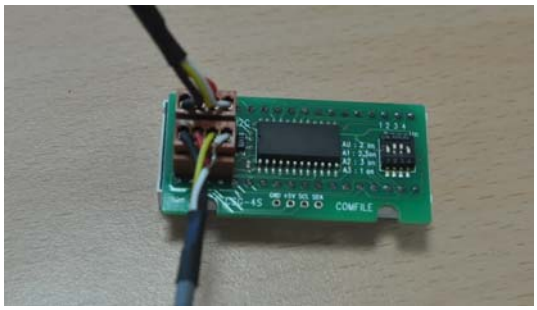
CPU 모듈 하단에는 캐릭터 LCD 모듈(CLCD 시리즈)연결을 위한 포트와 7 SEGMENT 모듈(CSG 시리즈)연결을 위한 포트가 있습니다.



간단하게 동작상황이나 특정정보를 표시하고 싶다면 주로 LCD 나 7 SEGMENT 를 사용합니다. LCD 는 1 개만 연결가능합니다. 저희 회사 홈페이지 (www.comfile.co.kr) 에서 디스플레이 관련제품군중 CLCD 시리즈중 하나를 사용하실 수 있습니다.

*모아콘은 새로운 펌웨어 (2011 년 1 월 출시이후 버전)의 CLCD 모듈과 호환됩니다. 이전 버젼의 CLCD 모듈을 가지고 계신분들은 펌웨어 업그레이드를 받으신후 모아콘과 함께 사용하시기 바랍니다. (업그레이드 문의 02-711-2592)
만약 구형 CLCD 모듈을 그대로 사용하고 싶으시다면, clcdPrint 명령대신에 clcdPrint2 명령어를 사용하십시오.

7SEGMENT 모듈은 최대 4 개까지 다음과 같은 방법으로 연결할 수 있습니다. 이때 각각의 Slave Address 는 서로 다르게 Dip 스위치를 조정해주어야 합니다.



저희 회사 홈페이지 (www.comfile.co.kr) 에서 디스플레이 관련제품군중 CSG 시리즈중 하나를 사용하실 수 있습니다.

CSG 와 CLCD 모듈은 케이블을 최대 3 미터 이내에서 사용하시기 바랍니다.

CSG/ CLCD 용 1 미터 케이블은 www.comfile.co.kr 홈페이지에서 구입가능합니다.

CLCD 모듈 관련 라이브러리

캐릭터 LCD 인 CLCD 시리즈를 구동할 수 있는 라이브러리입니다.

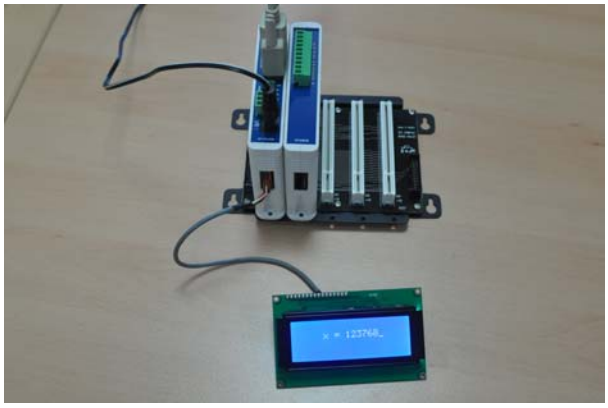
clcdI2cInit

void clcdI2cInit (u8 clcdadr)

clcdAdr : CLCD 모듈의 슬레이브 어드레스 (0 부터 7 까지 사용가능)

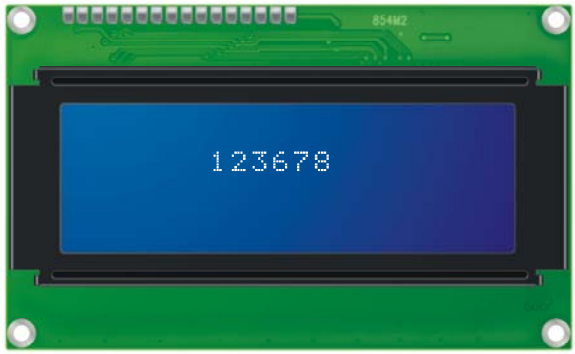
clcd 관련함수의 출력이 I2C 포트로 향하도록 해주는 선언함수입니다. 슬레이브 어드레스는 CLCD 모듈 뒷면에 DIP 스위치로 선택하도록 되어 있습니다. MOACON CPU 모듈에는 LCD 를 위한 포트가 별도로 마련되어 있으며 다음과 같이 선언하면 clcd 를 사용할 수 있는 상태가 됩니다.

```
clcdI2cInit(0); // 슬레이브 어드레스는 0 으로 합니다.
clcdPower(1); // clcd 에 전원을 공급합니다.
Delay(100);  // 100 밀리초 대기, 파워온 리셋 시간
```



다음은 CLCD 에 변수값을 표시하는 예제 프로그램입니다.

```
#include "moacon500.h"
void cmain(void)
{
   clcdI2cInit(0);
   clcdPower(1);
   delay(100); //기동시간대기
   int x=123678;
   clcdCls();
   while(1) {
         delay(500);
         clcdPrint (5,1, "%d",x++);
   }
}
```



clcdCls

void clcdCls (void)

CLCD 를 초기화하고, 모든 화면을 지웁니다. Clcd 모듈이 초기화하는 시간을 기다려준뒤 (약 100 밀리초) 수행을 마칩니다.

clcdCsr

void clcdCsr (u8 onOff)

onOff : 0=OFF, 1=ON

CLCD 의 커서를 On 또는 Off 하는 함수입니다. 파라메터가 1 이면 On, 0 이면 Off 상태가 됩니다. 초기화상태에서는 커서가 표시되는 On 상태입니다.

clcdPrint

void clcdPrint (u8 cx, u8 cy, string…)

cx : x 좌표 (0 부터 19 까지 사용가능)

cy : y 좌표 (0 부터 3 까지 사용가능)

CLCD 상에 문자를 표시하는 함수입니다. 우선, 표시할 위치 (x, y 좌표)를 정하고, 뒤에 표시할 문자열을 기술합니다. 문자열작성법은 printf 함수와 동일합니다.
*좌표를 반드시 적어주어야 합니다.

```
while(1){
     if (comLen(0)) {
       clcdPrint(0,1,"%03d",comGet(0));
       }
    statusLed(1);
    delay(100);
    statusLed(0);
    delay(100);
}
```

clcdLocate

void clcdLocate (u8 cx, u8 cy)

cx : x 좌표 (0 부터 19 까지 사용가능)

cy : y 좌표 (0 부터 3 까지 사용가능)

CLCD 상의 커서위치를 변경하고자 할 때 사용하는 함수입니다.

clcdBlit

void clcdBlit (u8 onOff)

onOff : 0=OFF, 1=ON

CLCD 의 백라이트를 On 또는 Off 하는 함수입니다. 파라메터가 1 이면 On, 0 이면 Off 상태가 됩니다.

clcdCmd

void clcdCmd (u8 ccmd)

ccmd : 1 바이트 데이터

CLCD 로 8 비트 데이터 하나를 전송시킵니다. 명령 코멘드 또는 특수 코드등을 전송할 때 사용합니다.

clcdPower

void clcdPower (u8 onoff)

onoff : 1=On, 0=Off

모아콘의 CLCD 연결포트 5V 전원을 ON / Off 할 수 있습니다. 디폴트 상태에서는 Off 입니다.

간혹 노이즈의 영향으로 LCD 화면상에 잘못된 표시가 나올 수 있습니다. 본 함수를 사용하면 CLCD 모듈의 전원을 껏다 켤수 있으므로, 잘못된 표시를 모두 지우고 처음부터 다시 표시를 시작할 수 있습니다. 장시간 운용되는 장비에서는 주기적으로 Off / On 을 해주는 것이 좋습니다. 꺼진 상태에서 적어도 1 초정도의 대기시간이 필요합니다.

```
clcdPower(0);
delay(1000); //꺼진상태에서 적어도 1 초간 대기해야 됩니다.
clcdPower(1);
delay(100); //기동시간대기
```

clcdPower 실행직후 CLCD 에 다른 코멘드를 보내면, clcd 가 이를 수행할 수 없습니다. 왜냐하면 파워온 초기화 작업중이기 때문입니다. 약 100 밀리초를 기다린후 다른 코멘드를 보내주어야 합니다.

```
  int i=0;
  clcdI2cInit(0);
  clcdPower(1);
  delay(100);  // Clcd 가 파워온 리셋하는 시간동안 어떤코멘드도 받아들일 수 없습니다.
```

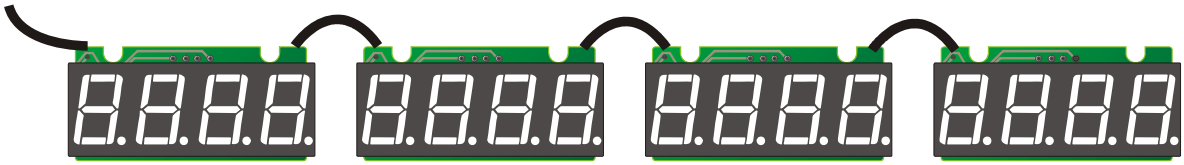
주의사항

타이머 이벤트 루틴 안에서 CLCD 에 표시하는 함수를 사용하실 수 없습니다. 메인루프에서 CLCD 화면에 데이터를 표시하고 있는 중간에 이벤트가 발생할 수 있는데, 이때 표시위치를 다른곳으로 바꾸어서 캐릭터를 표시하게되면, 이벤트 복귀후 엉뚱한 위치에 글자가 표시될 수 있기 때문입니다.

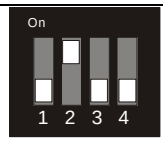
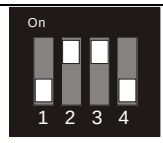
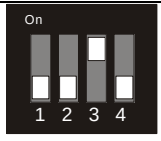
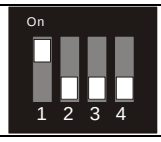
```
#include "moacon500.h"
void cmain(void)
{
  int i=0;
  clcdI2cInit(0);
  clcdPower(1);
  delay(100);
  startTimerEvent(100);
  while(1) {
        clcdPrint(1,3,"%05d",i++);
        clcdPrint(2,2,"Comfile");
        }
}

void timerEvent(void)
{
  clcdPrint(5,0,"Event Print");  // 이벤트 루틴안에서 표시위치를 바꿉니다.
}
```

CSG 모듈 관련 라이브러리

세븐세그먼트 모듈인 CSG 시리즈를 구동할 수 있는 라이브러리입니다. CSG 모듈 뒷편에는 슬레이브 어드레스를 선택할 수 있는 DIP 스위치가 있습니다. 여러 개의 CSG 모듈을 동일선상에 연결한뒤 슬레이브 어드레스만 각각 틀리게 하는 방법으로, 최대 4 개의 CSG 를 제어할 수 있습니다.



CSG 모듈 뒷면의 슬레이브 어드레스 Dip 스위치 조정방법입니다.

어드레스 0	2 번만 On	
어드레스 1	2,3 번 On	
어드레스 2	3 번만 On	
어드레스 3	1 번만 On	

csgPrint

void csgPrint (u8 csgSlaveAdr, String…)

csgSlaveAdr : CSG 모듈의 슬레이브 어드레스 (0 부터 3 까지 사용가능)
csgData : 표시할 데이터값 (0x30 – 0x39, 0x41 – 0x4f 만 사용가능)

CSG 상에 숫자및 문자를 표시하는 함수입니다. 다음과 같이 다양한 사용방법이 있습니다.

```
a=123;
csgPrint (0,"%4d",a); // 변수 a 를 10 진수로 표시합니다.
```

```
a=123;
csgPrint (0,"%04d",a); // 변수 a 를 10 진수로 표시합니다. 앞의 공백은 0 으로 채웁니다.
```

```
a=-123;
csgPrint (0,"%4d",a); // 음수일 경우 –부호도 함께 표시됩니다..
```

이 함수는 CSG 모듈의 표시위치 0 부터 표시를 시작합니다. 표시위치번호는 왼쪽부터 위치 0, 1, 2, 3 입니다.

csgPrintDot

void csgPrintDot (u8 csgSlaveAdr, u8 dot0, u8 dot1, u8 dot2, u8 dot3, String…)

csgSlaveAdr : CSG 모듈의 슬레이브 어드레스 (0 부터 3 까지 사용가능)
dot0 : 0 위치 도트 표시여부 (1=표시, 0=표시안함)
dot1 : 1 위치 도트 표시여부 (1=표시, 0=표시안함)
dot2 : 2 위치 도트 표시여부 (1=표시, 0=표시안함)
dot3 : 3 위치 도트 표시여부 (1=표시, 0=표시안함)
csgData : 표시할 데이터값 (0x30 – 0x39, 0x41 – 0x4f 만 사용가능)

도트와 함께 숫자및 문자를 표시하는 함수입니다. 도트를 표시하고 싶은 위치에 1 을 적어주면 해당위치에 도트가 표시됩니다. 다음과 같이 다양한 사용방법이 있습니다.

```
csgPrintDot (0, 1,0,0,0,"%4d",a); // 변수 a 를 10 진수로 표시합니다. 0 번위치에 도트표시
csgPrintDot (0, 0,1,0,0,"%04d",a); // 앞의 공백은 0 으로 채운 10 진수, 1 번위치에 도트표시
csgPrintDot (0, 0,0,1,0,"%4X",a); // 변수 a 를 16 진수로 표시합니다. 2 번 위치에 도트표시
csgPrintDot (0, 0,0,0,1,"%04X",a); // 앞의 공백은 0 으로 채운 16 진수, 3 번위치에 도트표시
```

이 함수는 CSG 모듈의 표시위치 0 부터 표시를 시작합니다.

```
a=123;
csgPrintDot (0, 1,0,0,0,"%4d",a); // 변수 a 를 10 진수로 표시합니다. 0 번위치에 도트표시
```

```
a=123;
csgPrintDot (0, 0,0,1,0,"%4X",a); // 변수 a 를 16 진수로 표시합니다. 2 번 위치에 도트표시
```

csgNput

void csgNput (u8 csgSlaveAdr, u8 csgDigit, u8 csgData)

csgSlaveAdr : CSG 모듈의 슬레이브 어드레스 (0 부터 3 까지 사용가능)
csgDigit : 위치 (0 부터 3 사이의 값)
csgData : 표시할 데이터값 (0x30 – 0x39, 0x41 – 0x4f 만 사용가능)

CSG 모듈의 4 자리중 원하는 자리에 특정 ASCII 코드를 표시합니다. MSB(최상위 비트)를 1 로 만들면 Dot 가 함께 On 됩니다.

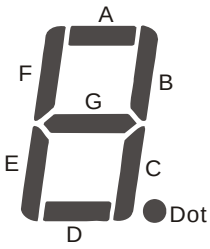
```
csgNput (0, 0, 0x42);  // 0 번위치에 b (아스키코드 0x42)표시
```

csgXput

void csgXput (u8 csgSlaveAdr, u8 csgDigit, u8 csgData)

csgSlaveAdr : CSG 모듈의 슬레이브 어드레스 (0 부터 3 까지 사용가능)
csgDigit : 위치 (0 부터 3 사이의 값)
csgData : 표시할 데이터값 (아무값이나 가능)

CSG 모듈의 4 자리중 원하는 자리에 특정 LED 만을 On 합니다. 8 개의 LED 를 각각 ON /OFF 할 수 있습니다. CsgNput 으로 표시할 수 없는 모양을 표현하고 싶을 때 사용하십시오.



Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Dot	G	F	E	D	C	B	A

```
csgXput (0, 1, 0x55);  // 1 번 위치에 01010101 점등
```

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Dot	G	F	E	D	C	B	A
	On		On		On		On

제 12 장

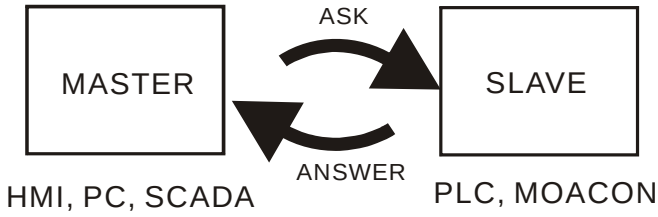
MODBUS RTU

MODBUS RTU

MODBUS 는 산업용 필드버스 분야에서 자주 사용되는 통신 프로토콜입니다. 모아콘에서는 MODBUS RTU 를 위한 라이브러리를 지원하고 있습니다.

MODBUS RTU 통신규격에는 마스터와 슬레이브의 역할이 나누어져 있습니다.

- 마스터는 물어보는 쪽입니다.
- 슬레이브는 마스터가 물어보는 데이터에 대해 응답을 해줍니다.



주로 마스터에는 산업용터치 HMI 기기, 또는 PC 와 같은 상위 기기가 위치합니다. 그리고 슬레이브에는 모아콘이나 PLC 등이 위치합니다.

슬레이브는 상위기기에서 원하는 동작만하는 수동적인 위치에 있습니다. 반면 마스터쪽에서는 원하는 데이터를 읽어오거나, 원하는 데이터를 기입하는등 적극적으로 슬레이브 기기를 다루어 주어야 합니다.

MODBUS RTU 슬레이브

startModbusRtu

void startModbusRtu(u8 comCh, u8 modSlaveadr, u16 * modbusBufferRegister,
u8 * modbusBufferCoil)

comCh : 통신 채널 (0 또는 1 또는 2 사용가능)
modSlaveadr : 모드버스의 슬레이브 어드레스
modbusBufferRegister : 레지스터 버퍼 시작번지
modbusBufferCoil : 코일 버퍼 시작번지

MODBUS RTU 슬레이브의 시작을 알리는 함수입니다. openCom 함수 바로 아래에 작성해 줍니다.

```
static u8 MDcoil[10];
static u16 MDregister[10];
openCom(2,38400, C8N1);
startModbusRtu(2,1,MDregister, MDcoil);
```

comCh 위치에 사용할 채널을 기입합니다. 모아콘은 총 3 개의 통신포트를 사용할 수 있습니다. **이 중 한 곳에서만 MODBUS RTU 슬레이브 기능을 활성화 시킬수 있습니다.** 3 채널 모두 동시에 사용은 불가능합니다.

modSlaveadr 에 슬레이브 어드레스를 기입합니다. 1 부터 255 사이 값을 사용할 수 있습니다. 0 번은 브로드케스트 모드로 할당되어 있으므로, 본 함수에서는 사용할 수 없습니다.

MODBUS RTU 를 사용하기 위해서는 16 비트 워드데이터를 저장하는 "레지스터 (Register)"영역과 1 비트 데이터를 저장하는 "코일(Coil)" 영역이 필요합니다. 데이터 교환을 위한 일종의 링크 영역이라고 생각하시면 됩니다.

이 공간을 위해 사전에 **"static 형 배열"**을 선언하고, 배열의 포인터를 함수에 넘겨줍니다.

1 비트 영역 (Coil)은 바이트형 배열로 선언하십시오.

배열의 1 번째 바이트에는 0 부터 7 번 비트가 포함되어 있습니다. 배열의 2 번째 바이트에는 그 다음 주소인 8 부터 15 번 비트가 포함되어 있습니다. 이런식으로 각각의 바이트에는 8 개의 비트가 배치되어 있는 것입니다.

	첫번째 바이트							
위치	7	6	5	4	3	2	1	0
주소	7	6	5	4	3	2	1	0

두번째 바이트							
7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8

세번째 바이트							
7	6	5	4	3	2	1	0
23	22	21	20	19	18	17	16

1 워드 영역 (Register)는 unsigned 16 비트 크기 배열로 선언하십시오.

위치	첫번째	두번째	세번째	네번째	다섯번째	…
주소	0	1	2	3	4	…

배열의 크기는 유저가 필요한 만큼 확보하실 수 있습니다. 너무 많이 확보하는 것 보다 필요한 만큼 확보하시는 것이 좋습니다. 전체 메모리 사용량이 그 만큼 줄어들기 때문입니다.

```
static u8 MDcoil[10];
static u16 MDregister[10];
openCom(2,38400, C8N1);
startModbusRtu(2,1,MDregister, MDcoil);
```

위 예제는 openCom 함수에서 38400 보레이트로 채널 2 를 오픈했기 때문에, MODBUS RTU 도 38400 으로 적용됩니다. MODBUS RTU 는 300 보레이트부터 사용가능합니다.

*모아콘에서는 MODBUS ASCII 를 지원하지 않습니다.

MODBUS RTU 펑션코드별 동작

MODBUS RTU 규격에는 여러가지 펑션코드가 있지만, 모아콘 SLAVE 모드에서는 아래와 같은 펑션코드만 지원합니다.

펑션코드 (10 진)	동작	설명
1	Read Coil Status	하나의 비트데이터 읽기
2	Read Input Status	하나의 비트데이터 읽기
3	Read Holding Registers	여러 개의 워드 읽기
4	Read Input Registers	여러 개의 워드 읽기
5	Force Single Coil	1 비트 쓰기
6	Preset Single Register	1 워드 쓰기
15	Force Multiple Coils	여러 개의 비트 쓰기
16	Preset Multiple Registers	여러 개의 워드 쓰기

MODBUS RTU 에서 Register 는 16 비트 저장공간을 의미합니다. MODBUS 규격에는 16 비트 저장공간도 여러종류가 있습니다. (Holding Register, Input Register)

모아콘에서는 따로 구분을 두지않고 하나의 16 비트 저장공간으로 통합하여 사용합니다.

즉 3 번 펑션코드나, 4 번 펑션코드를 사용해도 모두 같은 "워드 버퍼공간"에서 데이터를 리턴합니다.

1 비트 저장공간도 마찬가지입니다. Coil Status 와 Input Status 가 있지만 모아콘에서는 1,2 번 펑션코드 모두 같은 "1 비트 버퍼공간"에서 데이터를 리턴합니다.

펑션코드 01 : Read Coil Status

펑션코드 02 : Read Input Status

1 비트 상태를 읽어올 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 0~7 번을 읽어오는 예제입니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X01	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X08	1
에러체크	CRC	2

이에 대한 응답은 아래와 같습니다. Response:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X01	1
바이트 카운트	0X01	1
데이터 1	0X53	1
에러체크	CRC	2

비트정보 8 개를 읽어서 한 개의 바이트로 구성한다음 반환해줍니다..

펑션코드 03 : Read Holding Registers

펑션코드 04 : Read Input Registers

1 워드 데이터 상태를 읽어올 수 있는 펑션코드입니다. 다음은 슬레이브 3 번의 워드버퍼 어드레스 0~2 번을 읽어오는 예제입니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X03	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X03	1
에러체크	CRC	2

이에 대한 응답은 아래와 같습니다. 1 워드는 2 바이트이므로, 총 6 바이트의 데이터를 응답합니다.

Response:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X03	1
바이트 카운트	0X06	1
데이터 1 HI	0X03	1
데이터 1 LO	0XE8	1
데이터 2 HI	0X01	1
데이터 2 LO	0XF4	1
데이터 3 HI	0X05	1
데이터 3 LO	0X33	1
에러체크	CRC	2

펑션코드 05 : Force Single Coil

1 비트 WRITE 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 비트 저장 버퍼 어드레스 1 번을 ON 시키는 예제입니다. 데이터 필드에 ON 할 때에는 FF 00 을 OFF 할 때에는 00 00 을 보냅니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X05	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X01	1
데이터 HI	0XFF	1
데이터 LO	0X00	1
에러체크	CRC	2

이에 대한 응답은 아래와 같습니다.

Response:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X05	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X01	1
데이터 HI	0XFF	1
데이터 LO	0X00	1
에러체크	CRC	2

펑션코드 06 : Preset Single Registers

1 워드의 값을 변화시킬 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 워드 영역 버퍼 1 번지의 값을 변화 시키는 예제입니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X06	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X01	1
데이터 HI	0X12	1
데이터 LO	0X34	1
에러체크	CRC	2

이에 대한 응답은 아래와 같습니다.

Response:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X06	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X01	1
데이터 HI	0X12	1
데이터 LO	0X34	1
에러체크	CRC	2

펑션코드 15 : Force Multiple Coils

여러 개의 비트를 변화시킬 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 비트 버퍼 영역 어드레스 0~15 번을 변화시키는 예제입니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X0F	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X10	1
바이트 카운트	0X02	1
데이터 1	0XD1	1
데이터 2	0X05	1
에러체크	CRC	2

이에 대한 응답은 다음과 같습니다.

Response:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X0F	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X10	1
에러체크	CRC	2

펑션코드 16 : Preset Multiple Regs

여러 개의 워드 데이터 영역을 변화시킬 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 워드 버퍼 영역 어드레스 0~2 번을 변화시키는 예제입니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X10	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X03	1
바이트 카운트	0X06	1
데이터 1 HI	0XD1	1
데이터 1 LO	0X03	1
데이터 2 HI	0X0A	1
데이터 2 LO	0X12	1
데이터 3 HI	0X04	1
데이터 3 LO	0X05	1
에러체크	CRC	2

이에 대한 응답은 다음과 같습니다.

Response:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X10	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X03	1
에러체크	CRC	2

MODBUS RTU 마스터

모아콘을 마스터로 사용하려면 마스터용 라이브러리 함수를 사용해서 슬레이브쪽으로 데이터를 송신해주어야 합니다. 모아콘에서는 다음과 같은 마스터용 함수를 지원하고 있습니다. 이 함수는 보레이트 9600 이상에서만 사용가능합니다.

함수명	펑션코드	동작설명
RTU_readCoils	1,2	여러 개의 코일 읽기 (비트 읽기)
RTU_readRegs	3	여러 개의 레지스터 읽기 (워드 읽기)
RTU_readInRegs	4	여러 개의 Input 레지스터 읽기 (워드 읽기)
RTU_writeCoil	5	코일 1 비트 쓰기
RTU_writeReg	6	레지스터 1 워드 쓰기
RTU_writeCoils	15	여러 개의 코일 (8 비트)쓰기

MODBUS RTU 마스터 용 함수를 사용하기 위해서는 사전에 openCom 으로 해당 채널을 오픈해두어야 합니다.

```
#include "moacon500.h"
void cmain(void)
{
   int i=0,kk;
  u8 res;
  static u8 MBcoilBuffer[100];
  static u16 MBregisterBuffer[100];
   clcdI2cInit(0);
   clcdPower(1);
  delay(100);
  clcdCsr(0);
  openCom(0,57600,C8N1);  // 마스터로 사용할 채널을 오픈

   while(1) {
     delay(100);
     res = RTU_writeCoils(0,1,0,0x12); // MODBUS RTU 마스터 송신 (1 비트 쓰기)
     delay(100);
     res = RTU_readCoils(0,1,MBcoilBuffer,4,12); // 12 비트 읽기
      }
}
```

RTU_readCoils

short RTU_readCoils(u8 comCh, u8 slaveAdr, u8 * result, u16 targetAdr, u8 numberOfCoils)

comCh : 통신 채널 (0 또는 1 또는 2 사용가능)
slaveAdr: 슬레이브 주소
result : 결과를 저장할 배열의 주소
targetAdr : 읽어올 코일영역의 주소
numberOfCoils : 읽어올 코일의 개수
결과값 : -1 = 이상없이 수행됨,
0=타임아웃 에러,
1=데이터는 수신되었으나 잘못된 값이 포함되어 있음

슬레이브로부터 다수의 코일(비트 단위)정보를 읽어올 수 있는 함수 입니다. 읽어온 코일정보는 바이트 단위로 조합되어, result 에서 지정한 (바이트형) 배열위치에 처음부터 (배열요소 0 부터) 저장됩니다.

```
u8 coilBuffer[100];
res = RTU_readCoils(0,1,coilBuffer,4,12);
```

RTU_readRegs

short RTU_readRegs(u8 comCh, u8 slaveAdr, u16 * result, u16 targetAdr, u8 numberOfWord)

comCh : 통신채널 (0 또는 1 또는 2 사용가능)
slaveAdr: 슬레이브 주소
result : 결과를 저장할 배열의 주소
targetAdr : 읽어올 레지스터 영역의 주소
numberOfWord : 읽어올 레지스터의 개수
결과값 : -1 = 이상없이 수행됨,
0=타임아웃 에러,
1=데이터는 수신되었으나 잘못된 값이 포함되어 있음

슬레이브로부터 여러 개의 레지스터 (워드 단위)정보를 읽어올 수 있는 함수입니다. result 로 지정한 배열에 워드 단위로 정보를 기록합니다. 펑션코드는 3 번을 사용합니다.

```
u8 registerBuffer[100];
res = RTU_readRegs0,1,registerBuffer,0,2);
```

RTU_readInRegs

short RTU_readInRegs(u8 comCh, u8 slaveAdr, u16 * result, u16 targetAdr, u8 numberOfWord)

comCh : 통신채널 (0 또는 1 또는 2 사용가능)
slaveAdr: 슬레이브 주소
result : 결과를 저장할 배열의 주소
targetAdr : 읽어올 Input 레지스터 영역의 주소
numberOfWord : 읽어올 Input 레지스터의 개수
결과값 : -1 = 이상없이 수행됨,
0=타임아웃 에러,
1=데이터는 수신되었으나 잘못된 값이 포함되어 있음

RTU_readRegs 와 동일한 기능을 수행하는 함수입니다. 단 펑션코드를 4 번을 사용합니다. 일부 기기에서는 펑션코드의 종류에 따라 다른 값을 리턴하는 경우도 있습니다. 모아콘이 슬레이브일 경우, 3 번과 4 번 펑션코드는 모두 같은 영역입니다.

RTU_writeCoil

short RTU_writeCoil(u8 comCh, u8 slaveAdr, u16 targetAdr, u8 value)

comCh : 통신채널 (0 또는 1 또는 2 사용가능)
slaveAdr: 슬레이브 주소
targetAdr : 저장할 코일 영역의 주소
value : 저장할 값 (0 또는 1)
결과값 : -1 = 이상없이 수행됨,
0=타임아웃 에러,
1=데이터는 수신되었으나 잘못된 값이 포함되어 있음

슬레이브의 특정 코일 영역에 1 비트를 Write 하는 함수입니다.

```
res = RTU_writeCoil(0,1,0,1);  //슬레이브 코일영역 0 번지에 1 을 WRITE 합니다.
```

RTU_writeReg

short RTU_writeReg(u8 comCh, u8 slaveAdr, u16 targetAdr, u16 value)

comCh : 통신채널 (0 또는 1 또는 2 사용가능)
slaveAdr: 슬레이브 주소
targetAdr : 저장할 레지스터 영역의 주소
value : 저장할 1 워드 값 (16 비트)
결과값 : -1 = 이상없이 수행됨,
0=타임아웃 에러,
1=데이터는 수신되었으나 잘못된 값이 포함되어 있음

슬레이브의 특정 레지스터 영역에 1 워드를 Write 하는 함수입니다.

```
res = RTU_writeReg(0,1,0,0x12); //슬레이브 레지스터 영역 0 번지에 0X12 를 WRITE 합니다.
```

RTU_writeCoils

short RTU_writeCoils(u8 comCh, u8 slaveAdr, u16 targetAdr, u8 value)

comCh : 통신채널 (0 또는 1 또는 2 사용가능)
slaveAdr: 슬레이브 주소
targetAdr : 저장할 코일 영역의 주소 (8 의 배수단위)
value : 저장할 1 바이트(8 비트)
결과값 : -1 = 이상없이 수행됨,
0=타임아웃 에러,
1=데이터는 수신되었으나 잘못된 값이 포함되어 있음

슬레이브의 특정 코일영역에 8 비트의 데이터를 한꺼번에 WRITE 하는 함수입니다. targetAdr 은 반드시 8 의 배수로 적어주어야 합니다. (0, 8, 16, 24, 32…). value 는 기록할 1 바이트 값을 써줍니다.

```
res = RTU_writeCoils(0,1,0,0x12); // 슬레이브 코일영역 0 부터 7 위치에 0x12 를 기록합니다.
```

본래, 펑션코드 15 번은 여러 개의 비트를 WRITE 할 수 있지만, 이 함수에서는 8 비트만 WRITE 할 수 있습니다.

getCrc

u16 getCrc(u8 * targetArray, u16 datalength)

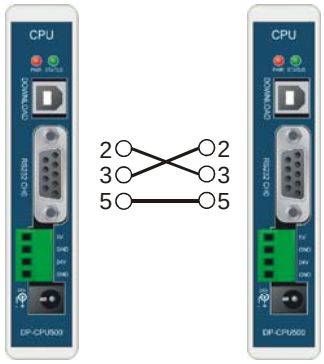
targetArray : 데이터가 저장된 배열명 (배열의 명칭만 적어줘야 합니다.)
dataLength : CRC16 계산할 데이터 개수

MODBUS RTU 마스터 송신관련 함수를 직접 만들어 사용하실 분들을 위해 CRC16 계산 함수를 준비해 두었습니다. 특정 배열에 있는 데이터중 원하는 개수만큼 CRC16 값을 계산해줍니다. 이때 배열은 반드시 바이트형 배열을 사용해야 합니다. 결과는 16 비트 정수값으로 리턴해줍니다.

```
u8 modbusTxdata[8],i,blen,j,totalnum;
 u16 crc,tadr;
modbusTxdata[0] = slaveAdr;  // 송신해야 될 값들을 모두 배열에 넣습니다.
modbusTxdata[1] = 1;
modbusTxdata[2] = targetAdr >> 8;
modbusTxdata[3] = targetAdr;
modbusTxdata[4] = numberOfCoils >> 8;
modbusTxdata[5] = numberOfCoils;
crc = getCrc(modbusTxdata,6);  // 배열에 있는 데이터를 가지고 CRC16 값을 계산합니다.
```

모아콘끼리 상호 통신

MODBUS RTU 를 이용해서 모아콘끼리 상호 통신하실 수 있습니다. 한쪽을 슬레이브로 만들고, 다른 한쪽에서 마스터 함수를 이용하는 방식입니다.



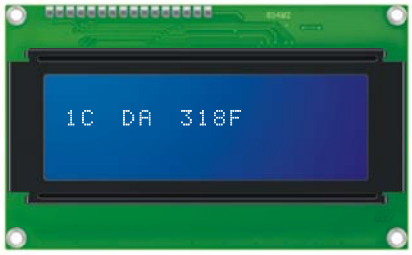
그림에서처럼 RS232 채널 0 의 2 번,3 번은 서로 교차 연결하고, 5 번은 5 번끼리 연결하십시오.

슬레이브에는 다음과 같은 소스를 사용하였습니다.

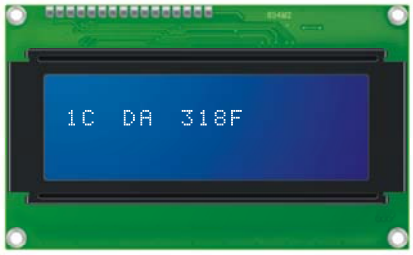
```
#include "moacon500.h"
void cmain(void)
{
   static u8 MDcoil[100];
   static u16 MDregister[100];
   openCom(0,57600, C8N1);
   startModbusRtu(0,1,MDregister, MDcoil);
   clcdI2cInit(0);
   clcdPower(1);
   delay(100);
   for(;;) {
         MDregister[0]++;
         MDregister[1]++;
         MDcoil[0]++;
         MDcoil[1]++;
         delay(100);
      clcdPrint(0,0,"%2X %2X %4X",MDcoil[0],MDcoil[1],MDregister[0]);
         }
}
```

마스터에는 다음과 같은 소스를 사용하였습니다.

```
#include "moacon500.h"

void cmain(void)

{
  short res;
  int errtime=0,errtime2=0;
  static u8 MBcoilBuffer[4];
  static u16 MBregisterBuffer[4];
   clcdI2cInit(0);
   clcdPower(1);
  delay(100);
  clcdCls();
  delay(100);
  clcdCsr(0);
  openCom(0,57600,C8N1);

   while(1) {
     res = RTU_readCoils(0,1,MBcoilBuffer,0,16);
     if (res != -1) {
       clcdPrint(0,2,"Error FC1 %d #%d",res,++errtime);
     }
     delay(200);
     res = RTU_readInRegs(0,1,MBregisterBuffer,0,1);
     if (res != -1) clcdPrint(0,3,"Error FC3 %d #%d",res,++errtime2);
     delay(200);
     clcdPrint(1,1,"%2X %2X %4X",MBcoilBuffer[0],MBcoilBuffer[1],MBregisterBuffer[0]);
         }
}
```



MODBUS RTU SLAVE



MODBUS RTU MASTER

마스터에서 슬레이브의 정보를 읽어와서 LCD 상에 표시합니다. 결과적으로 양쪽 LCD 창에는 같은 정보가 표시됩니다.

HMI / SCADA 에서 사용하는 어드레스

다음은 HMI / SCADA 소프트웨어등에서 사용하고 있는 어드레스 체계입니다.

포인트 타입	범위
Coil	1–999
Input Status	10001 – 19999
Input Register	30001 – 39999
Holding Register	40001 – 49999

HMI /SCADA 소프트웨어에서 40001 번지를 사용했다면, 펑션코드는 3 으로 어드레스는 0 으로해서 모아콘에 도착합니다.

Query:

필드명	*DATA*	*바이트 수*
슬레이브 어드레스	0X03	1
펑션코드	0X04	1
시작어드레스 HI	0X00	1
시작어드레스 LO	0X00	1
길이 HI	0X00	1
길이 LO	0X03	1
에러체크	CRC	2

30001 을 사용하면 펑션코드는 4, 그리고 어드레스는 0 이되어 모아콘에 도착합니다.

* HMI /SCADA 프로그램에 따라서 다를 수 있습니다.

제 13 장
이더넷 통신

ETHERNET 통신 모듈

모아콘에서 이더넷 통신기능을 사용하시려면, 이더넷 통신모듈을 추가로 장착하신뒤, 전용 라이브러리를 사용하시면 됩니다.

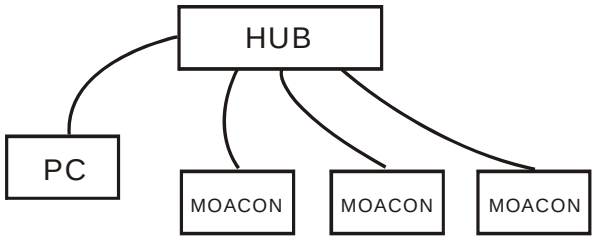
LED 설명

TX : 송신
RX : 수신
ERR : 데이터 충돌 (IP 충돌)
FULL : 전이중통신 (Full duplex)
100M : 100M 접속시 On
LINK : 연결활성화시 On

모아콘의 이더넷 통신모듈은 TCP 통신을 지원합니다. TCP 통신은 서버 / 클라이언트 방식으로 운용됩니다. UDP 통신 및 기타 다른 통신방식은 지원하지 않습니다.

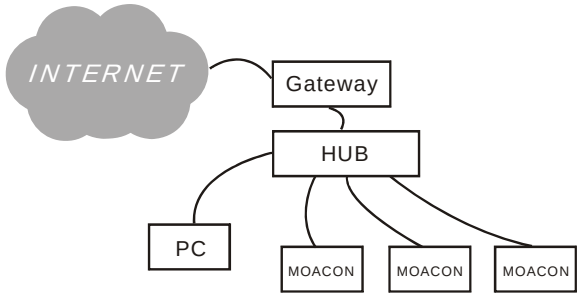
네트워크 구성예

1. 로컬 네트워크만 구성된 경우



하나의 허브에 여러 개의 모아콘과 PC 를 연결하여 서로 통신하는 경우입니다. 이때 반드시 허브는 (인터넷 연결기능이 없는) 스위칭허브를 사용해야 합니다. 인터넷공유기는 사용하실수 없습니다.
이 경우 외부 인터넷망과 접속이 없으므로, 각각의 모아콘은 임의대로 MAC 어드레스와 IP 주소를 부여할 수 있습니다. 단 로컬네트워크안에서 같은번호가 중복되지 않도록만 신경써주시면 됩니다.

2. 인터넷에 연결된 경우.



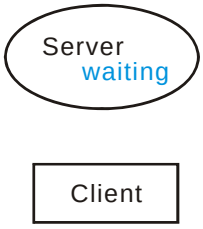
게이트웨이 (인터넷 공유기)를 통해 인터넷에 연결된 네트워크망에 모아콘을 접속한 경우입니다. 이때 각각의 모아콘은 IEEE 에서 부여된, 고유한 MAC 번호를 가지고 있어야하며, IP 주소 또한 로컬네크워크 아래에서 고유한 번호를 부여해주어야합니다.
이더넷 통신 모듈에는 제품고유의 MAC 어드레스가 스티커로 부착되어 있으므로, 이것을 입력하시면 됩니다.

TCP 서버 클라이언트 통신

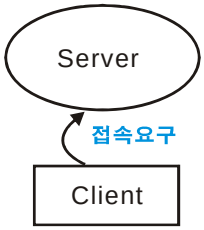
이더넷 통신모듈을 사용하기 위해서는 서버 / 클라이언트 통신방식에 대한 이해가 필요합니다.

한쪽은 서버상태, 다른한쪽은 클라이언트상태로 있다가, 클라이언트에서 서버로 접속요구를 한뒤, 접속이 되면 데이터를 주고받는 방식입니다. 단계별로 통신이 이루어지는 과정을 살펴보겠습니다.

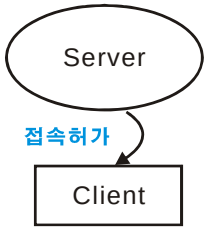
단계 1.
서버는 대기상태가 됩니다.



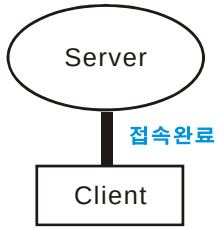
단계 2.
클라이언트에서 접속요구를 합니다.



단계 3.
서버에서 접속요구에 허가를 해줍니다.

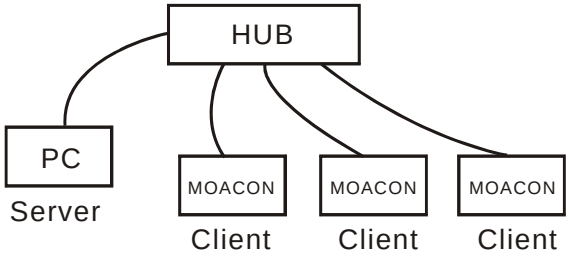


단계 4.
서버와 클라이언트가 접속되었습니다.



접속이 완료된 다음에는 서로 데이터를 주고 받을 수 있습니다. 통신을 하기위해서는 계속 접속상태가 유지되어야 된다는 점이 특징입니다.

모아콘이 클라이언트가 되고, PC 가 서버가되어 네트워크를 구성한 예입니다.



*TCP 통신은 1:1 통신만 가능합니다. 동시에 여러 개의 기기에 데이터를 보낼수 없습니다.

이더넷 통신모듈 관련 라이브러리

netBegin

void netBegin(u8 * GatewayIP, u8 * SubnetMask, u8 * MacAdr, u8 * DeviceIp)

GatewayIP : 게이트웨이의 IP 주소
SubnetMask : 서브넷 마스크
MacAdr : MAC 어드레스
DeviceIP : 디바이스 IP 주소

사용방법은 다음과 같습니다. 필요한 파라메터 배열선언을 먼저 한뒤 netBegin 함수로는 각 배열의 포인터만 전달해주는 것입니다.

```
u8 GatewayIP[]={192,168,0,1};
u8 SubnetMask[]={255,255,255,0};
u8 MacAdr[]={0,0,34,53,12,0};
u8 DeviceIp[]={192,168,0,12};

netBegin(GatewayIP, SubnetMask, MacAdr, DeviceIp);
```

DeviceIP 는 본 제품의 IP 어드레스를 의미합니다. 게이트웨이와 접속된 경우에는 게이트웨이에서 DHCP 라는 기능을 사용해서, IP 를 자동적으로 할당해줍니다. (로컬 네트워크의 IP 어드레스가 중복되지 않도록 관리해주는 것입니다.) 하지만 본 제품에서는 DHCP 에서 할당해주는 IP 를 받아주는 프로토콜 처리 기능이 내장되어 있지 않으므로, DHCP 기능을 사용할 수 없습니다.

따라서 DeviceIP 는 네트워크상에서 다른 기기와 중복되지 않는 번호로 유저여러분이 직접 할당해주셔야 합니다.

socketOpen

u8 socketOpen (u8 socket, u16 port)

socket : 소켓번호 (0 부터 3 사이 값)
port : 포트번호

이더넷 통신을 사용하기 위해서는 socketOpen 함수를 사용해서 원하는 Socket 을 오픈해주어야 합니다. 본 제품에서는 4 개의 소켓을 지원합니다.

```
socketOpen(0, 82); // 0 번 소켓의 82 번 포트를 오픈합니다.
```

포트번호는 0 부터 65535 중 하나의 값을 사용할 수 있습니다. 이중 특정포트는 이미 용도가 정해져 있습니다. 예를들어 포트 80 은 HTTP (WEB)을 위한 포트입니다. 광역망에 접속하지 않고 로컬네트워크만으로 데이터를 통신하는 경우에는 아무 포트번호나 사용할 수 있지만, 광역망에 접속하는 경우라면, 이미 용도가 정해진 포트를 제외한, 나머지 포트중 하나를 사용하셔야만 합니다.

소켓오픈이 성공하면 1 을 반환하고, 실패하면 0 을 반환합니다.

socketClose

void socketClose(u8 socket)

socket : 소켓번호 (0 부터 3 사이 값)

소켓을 Close 하는 함수입니다. 해당 소켓이 오픈되어있는지 여부가 확실하지않은 경우에도 사용하실 수 있습니다.

여기서 잠깐

모아콘 이더넷통신모듈에는 각 소켓별로 수신 2K 바이트, 송신 2K 바이트의 버퍼가 내장되어 있습니다. 소켓이 4 개이므로 총 16K 바이트의 버퍼가 내장되어있습니다. 이 메모리 영역은 CPU 모듈의 64K 데이터 메모리와 별도의 영역입니다. 따라서, 이더넷 통신기능 때문에 메인 메모리 공간이 줄어들지 않을까하는 염려는 하실 필요가 없습니다.

listen

u8 listen(u8 socket)

socket : 소켓번호 (0 부터 3 사이 값)

해당 소켓을 서버모드로 만드는 함수입니다.

connect

u8 connect (u8 socket, u8 * destIP, u16 port)

socket : 소켓번호 (0 부터 3 사이 값)
destIP : 접속할곳의 IP 주소
port : 접속할 포트번호

해당 소켓을 클라이언트 모드로 만드는 함수입니다. destIP 에는 접속할 곳의 IP 주소를 적어줍니다.

```
u8 GatewayIP[]={192,168,0,1};
u8 SubnetMask[]={255,255,255,0};
u8 MacAdr[]={0,0,34,53,12,0};
u8 DeviceIp[]={192,168,0,12};
u8 destIP[]={192,168,0,2};

netBegin(GatewayIP, SubnetMask, MacAdr, DeviceIp);

socketOpen(0,82);
connect(0,destIP,5000);
```

disConnect

u8 disConnect (u8 socket)

socket : 소켓번호 (0 부터 3 사이 값)

해당 소켓을 연결해제하는 함수입니다. 해제성공시 1 을 반환, 실패시 0 을 반환합니다.

```
disConnect(0);
```

netStatus

u8 netStatus (u8 socket)

socket : 소켓번호 (0 부터 3 사이 값)

해당소켓의 상태를 알려주는 함수입니다. 결과값은 다음중 하나가 됩니다.

결과값	define 상수	설명
00	SOCK_CLOSED	socketClose 함수가 성공적으로 수행된경우, 또는 접속이 종료된 경우, 시간이 초과된 경우입니다.
0x13	SOCK_INIT	socketOpen 함수에 의해 소켓이 오픈된 경우입니다. 이후 listen 이나 connect 함수를 수행하면 다른상태로 바뀝니다.
0x 14	SOCK_LISTEN	listen 함수를 수행한 상태입니다. TCP 서버 상태입니다. 이후 클라이언트에서 접속시도를한뒤 성공하면 SOCK_ESTABLISHED 로 전환됩니다.
0x 17	SOCK_ESTABLISHED	서버와 클라이언트가 서로 연결된 상태입니다. 즉 통신가능한 상태입니다.
0x1C	SOCK_CLOSE_WAIT	접속종료 요구를 받은 상태입니다. 즉, 접속종료요구를 받았지만 아직 접속종료가 되지 않은 상태입니다.

숫자대신 define 상수를 써서 프로그램을 작성하시는 것이 소스를 이해하는데 도움을 줍니다.

```
while (1) {
    switch(netStatus(0)) {
    case SOCK_ESTABLISHED:
        //
        //
break;
}
```

netSend

u16 netSend(u8 socket, u8 * buf, u16 len)

socket : 소켓번호 (0 부터 3 사이 값)
buf :송신할 데이터의 포인터
len : 송신할 데이터의 바이트수

TCP 모드에서 사용하는 데이터 송신 함수입니다. netStatus 함수로 읽어본 상태가 SOCK_ESTABLISHED 일 경우에만 사용할 수 있습니다. 보낼 데이터를 사전에 배열에 저장해둔뒤 배열의 이름 (포인터)를 가지고 본 함수를 호출하면 해당 배열의 len 으로 지정한 바이트수만큼의 데이터가 송신됩니다.

송신이 성공하면 1, 실패하면 0 을 반환합니다.

```
u8 data1[] = "comfile";
netSend(0,data1, 7);
```

netPrint

void netPrint(u8 socket, u8 * string)

socket : 소켓번호 (0 부터 3 사이 값)
string : 보낼 문자열의 포인트

printf 와 같은 형식으로 데이터를 보낼수 있는 함수입니다.

```
netPrint(0,"Comfile%D",kk++);
```

netTxFree

u16 netTxFree(u8 socket)

socket : 소켓번호 (0 부터 3 사이 값)

netSend 나 netPrint 함수가 실행되면 보내고자 하는 데이터는 송신버퍼에 저장됩니다. 해당 함수에서 송신이 끝날때까지 대기하지 않습니다. 이후 송신과정은 이더넷 모듈이 알아서 처리합니다.

만약, 데이터송신이 다 끝났는지 알고싶다면, netTxFree 함수를 이용해서 송신버퍼의 남아있는 용량을 체크하면됩니다. 최초상태에서는 2048 입니다. (2K 의 송신버퍼를 가지고 있기 때문입니다.)

즉, netTxFree 의 결과값이 2048 이라면, 더 이상 송신할 데이터가 없다는 뜻입니다.

netRecv

u16 netRecv(u8 socket, u8 * buf, u16 len)

socket : 소켓번호 (0 부터 3 사이 값)
buf :수신한 데이터를 저장할 곳의 포인터
len : 수신할 데이터의 바이트수

이더넷 데이터 수신 함수입니다. netStatus 함수로 읽어본 상태가 SOCK_ESTABLISHED 일 경우에만 사용할 수 있습니다.

수신할 데이터의 개수를 적어주어야 하므로, 본 함수를 사용하기 전에 수신버퍼에 데이터가 도착해 있는지를 살펴보아야 합니다. 이를 위한 netRxLen 함수가 준비되어 있습니다.

성공하면 수신한데이터의 개수를 반환하고, 실패하면 –1 (0xffff)을 반환합니다.

netRxLen

u16 netRxLen(u8 socket)

socket : 소켓번호 (0 부터 3 사이 값)

수신데이터버퍼에 쌓여잇는 데이터의 개수를 반환합니다. 수신된 데이터가 없다면 0 을 반환합니다. 수신버퍼가 2KB 이므로 최대 2048 바이트만 수신버퍼에 넣을수 있습니다. 따라서 수신버퍼가 가득 차기전에 netRecv 함수를 이용해서 데이터를 읽어내야 합니다.

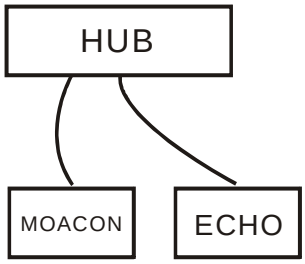
수신버퍼가 가득찬 이후에 수신되는 데이터는 더 이상 저장할 곳이 없어서, 버려지게 됩니다.

여기서 잠깐…

모아콘에서 이더넷 데이터 수신이벤트는 지원하지 않습니다. 타이머 이벤트를 사용해서 주기적으로 수신 버퍼에 데이터가 있는지를 조사하여, 처리하는 방법을 사용하십시오.

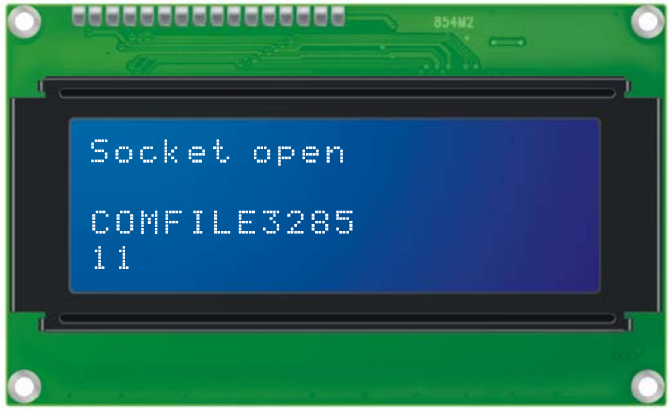
테스트 프로그램

다음은 TCP 루프백 테스트 프로그램입니다. 아래그림처럼 네트워크상에 TCP 에코처리를 해주는 별도의 디바이스가 있어야 테스트가 가능합니다. 이 프로그램은 에코디바이스로 TCP 데이터를 송신하고, 같은 데이터가 되돌아 오는지를 확인하여, LCD 상에 표시해주는 프로그램입니다. 에코디바이스의 IP 어드레스는 192.168.0.1 로 되어 있습니다.




```
#include "moacon500.h"
void cmain(void)
{
  u8 i=0x35,j;
  u16 k,kk=0,length1;
  clcdPower(1);
  clcdI2cInit(0);

  u8 GatewayIP[]={192,168,0,1};
  u8 SubnetMask[]={255,255,255,0};
  u8 MacAdr[]={0,0,34,53,12,0};
  u8 DeviceIp[]={192,168,0,12};
  u8 destIP[]={192,168,0,2};

  char data1[] = "Comfile";
  char data2[20];

  netBegin(GatewayIP, SubnetMask, MacAdr, DeviceIp);

  socketOpen(0,1000);
  connect(0,destIP,5000);

  while (1) {
    switch(netStatus(0)) {
    case SOCK_ESTABLISHED:
      netPrint(0,"Comfile%D%C",kk++,0);
      delay(20);
      length1 = netRxLen(0);
```

```
        clcdLocate(0,2);
        netRecv(0,data2,length1);
        data2[--length1]=0;
        clcdPrint(0,2,data2);
        clcdPrint(0,3,"Rx=%d",length1);
        delay(200);
        break;
      case SOCK_CLOSE_WAIT:
        disConnect(0);
        clcdPrint(0,0,"Socket disconnect");
        delay(200);
        break;
          case SOCK_CLOSED:
        clcdPrint(0,0,"Socket Closed    ");
        delay(200);
        if (connect(0,destIP,5000) != 0 ) {
          clcdPrint(0,0,"Socket open      ");
          }
        break;
      }
    }
  }
```



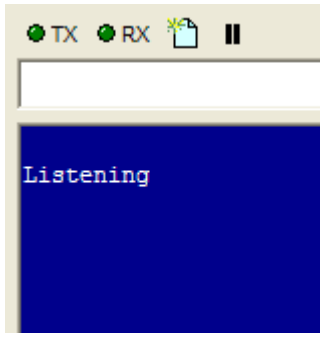
웹서버 프로그램

다음은 모아콘과 이더넷모듈을 사용해서 간단한 웹서버를 구현한 프로그램입니다.

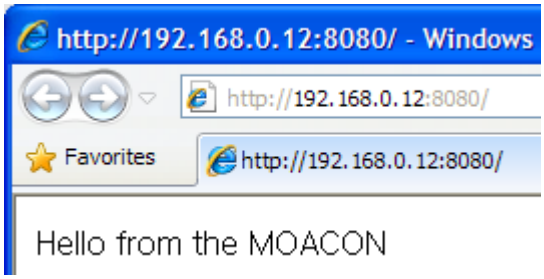
```
#include "moacon500.h"
#include <string.h>

void cmain(void)
{
 //Configure the network settings
 u8 GatewayIP[]={192,168,0,1};
 u8 SubnetMask[]={255,255,255,0};
 u8 MacAdr[]={0,0,34,53,12,0};
 u8 DeviceIp[]={192,168,0,12};
 netBegin(GatewayIP, SubnetMask, MacAdr, DeviceIp);

 //Use socket 0
 u8 socket = 0;

 //Keep track of the connection status
 u8 currentStatus = 0xFF;
 u8 lastStatus = 0xFF;

 while (1)   //Run forever
 {
    currentStatus = netStatus(socket);
    if (currentStatus != lastStatus)        //If connection status changes
    {
       lastStatus = currentStatus;

       switch(currentStatus)
       {
          case SOCK_INIT:        //If not listening, start listening
            printf("Init\r\n");
            socketOpen(socket,8080);
            listen(socket);
            break;

          case SOCK_CLOSED:      //If closed, start listening
            printf("Closed\r\n");
            socketOpen(socket,8080);
```

```
            listen(socket);
            break;

          case SOCK_ESTABLISHED: //If connection establed, respond
            printf("Established\r\n");

            netPrint(socket, "HTTP/1.0 200 OK\r\n");
            netPrint(socket, "Content-Type: text\r\n");

            char* hello = "Hello from the MOACON";
            netPrint(socket, "Content-Length: %d\r\n\r\n", strlen(hello));
            netPrint(socket, "%s", hello);

            break;

        case SOCK_LISTEN:             //If listening, just wait
            printf("Listening\r\n");
            break;

        case SOCK_CLOSE_WAIT:         //If client disconnects, disconnect
            printf("Closing\r\n");
            disConnect(socket);
            break;

        default:
            break;
       }

       delay(10);              //Wait for 10 milliseconds
     }
  }
}
```

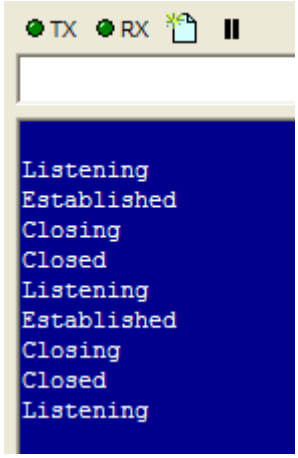
웹서버가 준비되면, 디버그창에 다음과 같은 멧세지가 표시됩니다.



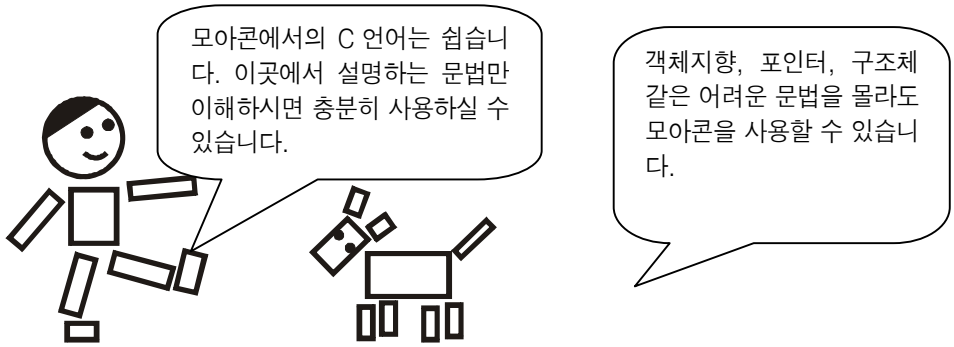
브라우저에서 아래와 같이 해당 ip 주소와 8080 포트를 입력하십시오. 브라우저에는 모아콘에서 보낸 문자열이 표시됩니다.



디버그 창에는 웹서버의 상태가 표시되도록 하였습니다.



부록 1.
C 언어


모아콘에서의 C 언어는 쉽습니다. 이곳에서 설명하는 문법만 이해하시면 충분히 사용하실 수 있습니다.
객체지향, 포인터, 구조체 같은 어려운 문법을 몰라도 모아콘을 사용할 수 있습니다.

1. C 언어의 기본 구성

C 언어의 기본 특징을 살펴보도록 하겠습니다.

1.1 세미콜론

C 언어에서는 문장 맨끝에 반드시 세미콜론을 붙여줍니다.

```
sum = a + b;
```

세미콜론으로 문장이 끝납니다.

1.2 블록

여러 개의 문장은 하나의 블록으로 만들 수 있습니다. 블록은 만드는 방법은 아래와 같이 { } 기호로 묶으면 됩니다. 통상적으로 블록앞에는 C 언어의 어떤 예약어가 붙습니다.

C 의 예약어

```
while (1)
{
    a = 5;
    sum = a + 4;
}
```

1.3 함수

함수는 블록의 앞에 예약어 대신 함수명을 사용합니다. **함수는 C 언어를 구성하는 가장 기본적인 단위가 됩니다.** 함수 여러 개가 모여서 하나의 C 프로그램이 완성됩니다.

Main 프로그램도 main 이라는 함수로 되어 있습니다.

```
main ()
{
    int sum, a;
    a = 5;
    sum = a + 4;
}
```

MOACON 에서는 main 함수 대신 cmain 이라는 함수를 메인함수로 사용합니다.

```
void cmain (void)
{
    int a, b;
    a = 5;
    b = a + 4;
    delay_ms(100);
}
```

cmain 함수

다른함수 호출

cmain 함수에서는 다른 함수를 호출하는 방법으로 여러 개의 함수를 실행시킬 수 있습니다.

2. 유저정의 함수

C 프로그램에서는 여러가지 함수를 유저가 임의대로 정의해서 사용할 수 있습니다.

```
int sum(int a, int b)
{
    return a + b;
}

void cmain (void)
{
    int a, b;
    a = 5;
    b = 4;
    b = sum (a, b);
}
```

유저가 정의한 sum 함수

앞에서 정의한 sum 함수를 호출 b = a + b 와 같은 효과

위의 소스는 두개의 인수를 받아 더한뒤 결과를 리턴하는 sum 함수를 구현해본 것입니다. 이처럼 C 프로그램에서는 지금까지 존재하지 않았던 함수를 유저가 정의해서, C 의 기본 함수처럼 호출할 수 있습니다

3. 주석을 붙이는 방법

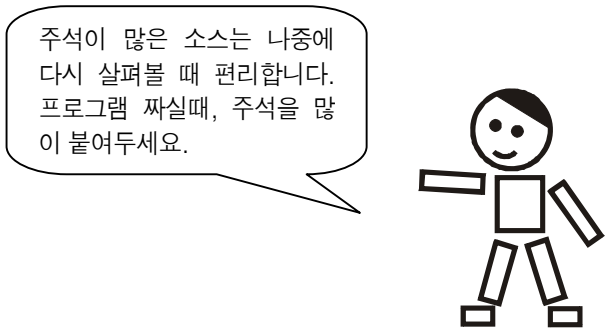
주석은 프로그램의 이해를 돕기 위해서 작성하는 부분입니다. C 언어에서 주석을 붙이는 방법은 2 가지가 있습니다.

```
/*
       메인 프로그램
*/


    void cmain (void)
    {
        int a, b;
        a = 5;
        b = 4;
        b = sum (a, b);  // sum 함수를 호출
    }
```

슬레쉬 두개를 붙여서 주석을 작성

위의 소스에서 처럼 /* 과 */을 사용해서 작성하는 방법과 //를 사용해서 작성하는 방법이 있습니다. "/*"과 "*/"로 둘러쌓여 있는 것은 전부 주석으로 간주되며 번역되지 않고, 무시됩니다.

//는 행 뒤에 주석을 붙일 때 사용합니다.



4. 변수

변수란 그 값을 자유롭게 변경할 수 있는 기억장소를 의미합니다. C 언어에서 모든 변수는 형(type)을 가지고 있습니다. 형(type)이란 변수의 성격을 말하는데, 정수형 변수에는 정수형 수치만 넣을 수 있으며, 실수형 변수에는 실수형 수치만 넣을 수 있습니다.

4.1 변수의 대입

C 언어에서 어떤 변수에 임의의 값을 넣기 위해서는 X = 10 과 같이 대입 기호 = (Eaual)을 사용합니다. 수학적인 의미는 X 와 10 이 같다는 뜻이지만, C 언어에서는 변수 X 에 10 을 대입한다는 뜻이 됩니다.

```
좌변 (변수) = 우변        //    우변의 수를 좌변에 대입
```

이때 우변은 변수또는 수식이라도 상관없지만, 좌변은 반드시 변수이여야 합니다.

4.2 변수의 형

다음은 변수의 형과 설명을 요약한 표입니다.

형	비트수	설명
char	8 비트	부호있는 바이트
short	16 비트	부호있는 하프워드
int	32 비트	부호있는 32 비트 정수
long	32 비트	부호있는 32 비트 정수
float	32 비트	IEEE 단일 정밀도
double	64 비트	IEEE 두배 정밀도
long long	64 비트	부호있는 64 비트 정수

1 워드는 32 비트, 하프워드 (halfword)는 16 비트를 의미합니다. 부호없는 변수를 선언할 때에는 앞에 unsigned 를 붙여줍니다.

4.3 변수의 선언

변수의 형(type)이 정해지면 컴파일러에게 알려주어야 합니다. 이것을 변수의 선언이라고 합니다. 변수선언의 일반적인 서식은 다음과 같습니다.

형(type) 변수명, 변수명……;

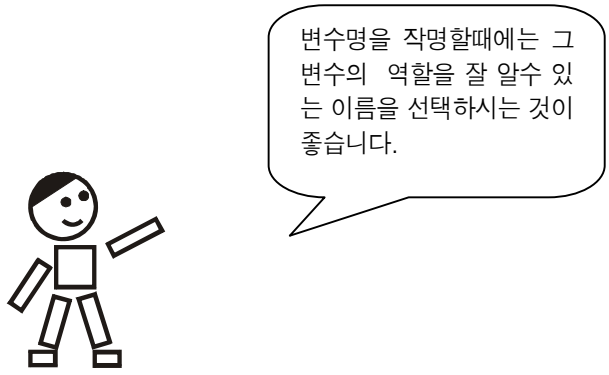
형(type)은 앞에서 설명한 형명중 하나를 사용하고, 변수명에는 여러분이 붙인 고유의 이름을 적어주면 됩니다. 예를들면 다음과 같습니다.

```
int  M, J, K;   // M, J, K 를 정수형으로 선언합니다.
```

변수명은 최대길이는 제한이 없지만 적당한 선에서 의미있는 단어를 사용하는 것이 좋습니다. 너무 길면 기억하기 어렵고, 너무 짧으면 그 의미를 파악하기 어렵습니다.

단, 숫자로 시작하는 단어나, C 또는 OS 에서 이미 사용하고 있는 예약어는 변수로 사용할 수 없습니다. 대소문자를 엄격히 구분하므로, 대소문자 사용에 주의를 기울여 주십시오.
C 언어는 대부분 소문자로 작성하고, 일부분에서만 대문자를 사용합니다.

```
int  123abc; // 숫자로 시작하는 문자는 변수명으로 쓸 수 없습니다.
int  for;    // 예약어는 변수명으로 쓸 수 없습니다.
```



5. 상수

상수란, 소스 작성시 그 값이 정해져 있는 수를 말합니다.

```
x = 1;          // 1 이 상수입니다. 대입식에 이용
x = y + 5;      // 5 가 상수입니다. 수식에 이용
```

상수는 const 명령을 사용해서 어떤 문자로 바꾸어서 사용할 수도 있습니다. 이때, 상수의 형도 함께 정의해주어야 합니다.

```
const  int baud_value = 208;
```

이 문장이 정의된 이후부터는 baud_value 가 상수 208 을 대신하게 됩니다. 이러한 상수선언은 실전 프로그램에서 유용하게 쓰이고 있습니다. 바뀔 수도 있는 어떤값을 소스프로그램 여러곳에 숫자로 작성해놓게 되면, 나중에 그 숫자가 바뀌었을 때, 일일이 찾아서 다시 바꿔주어야하는 불편함이 있기 때문입니다. 이 과정에서 놓치는 부분도 발생할 수 있으므로 에러의 원인이 되기도 합니다. 프로그램 맨앞에서 상수정의를 해놓았을 경우에는 상수정의부만 바꾸고 다시 컴파일하면 완벽하게 상수의 값을 대치할 수 있습니다.

5.1 상수배열

아래와 같이 배열과 같은 형태로 여러 개의 상수를 정의하는 방법도 있습니다. 이와 같은 형태를 "상수배열"이라고 합니다.

```
const char port_bit[8] = {1,2,4,8,32,1,2,4};
```

port_bit[0]의 값은 1 이되고, port_bit[7]의 값은 4 가 됩니다.

상수배열은 배열처럼 사용하지만 그 내용을 바꿀수는 없습니다. 읽기전용 배열이라고 생각하시면 됩니다.

6. 진수표현; 2 진, 10 진 16 진수

C 언어에서 2 진, 10 진, 16 진수를 표현하는 방법에 대하여 알아보겠습니다. 숫자를 아무 표시없이 그냥 사용할 경우에는 10 진수로 인식합니다. 숫자 앞에 0x 를 붙이면 16 진수로 인식합니다.

```
10,  20,  100        // 10 진수
0xab, 0x100, 0x1ff   // 16 진수
```

2 진수, 10 진수, 16 진수의 관계를 표로 정리하면 다음과 같습니다. 표에서 알 수 있듯이 2 진수 4 자리는 16 진수 한자릿수로 일대일 변환할 수 있습니다.

2 진수	10 진수	16 진수
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

참고 : C 언어에서는 2 진수를 소스상에 직접 사용할 수 없으므로, 16 진수로 바꾸어 써주어야 합니다.

```
Binval = 0b10100101;       // 이런식의 표현이 불가능합니다.
Binval = 0xa5;             // 16 진으로 바꾸어 써주어야 합니다.
```

7. 연산자

C 언어 특징중에 하나로 연산자가 풍부하다는 것을 들 수 있습니다. C 언어는 기본적인 연산자 이외에도 여러가지 변칙적인 연산자를 사용할 수 있기 때문에, 연산자에 대한 이해가 필수적입니다.

연산자란, 식중에서 변수와 상수 (이것을 피연산자라고 합니다.)에 대해 어떠한 연산을 할 것인가를 지시하는 것입니다. C 에서는 다음과 같은 연산자가 있습니다.

연산자의 명칭	예
산술 연산자	a + b
증가 연산자	a++
감소 연산자	a--
관계 연산자	a > b
논리 연산자	!a
비트 연산자	~a
대입 연산자	a = b
조건 연산자	(a > 0) ? a : -a
콤마 연산자	for (i = 1, j = 1 ; ……);
어드레스 연산자	&a

7.1 연산자의 우선순위

연산자가 여러 개 등장했을 때, 어떠한 연산자부터 연산할 것인지는 연산자의 우선순위에 의해서 결정됩니다. C 언어에서의 연산자의 우선순위는 다음과 같습니다.

연산자명	순위	연산자	결합방향
괄호 연산자	1	( ) [ ]	→
구조체 연산자	1	.	→
구조체 포인터 연산자	1	->	→
부호 연산자	2	+ -	←
논리 부정 연산자	2	!	←
1 의 보수 연산자	2	~	←
증감 연산자	2	++ --	←
캐스트 연산자	2	(형명)	←
간접지정 연산자	2	*	←
번지 연산자	2	&	←
sizeof 연산자	2	sizeof	←
산술 연산자	3	* / %	→
산술 연산자	4	+ -	→
쉬프트 연산자	5	<< >>	→
관계 연산자	6	< <= >= >	→
상등 연산자	7	== !=	→
비트 and 연산자	8	&	→
비트 xor 연산자	9	^	→
비트 or 연산자	10	|	→
논리곱 연산자	11	&&	→
논리합 연산자	12	||	→
조건 연산자	13	? :	←
대입 연산자	14	=	←
대입 연산자	15	+=, -=, *=, /=…	←
쉼표 연산자	16	,	←

이 표에는 결합방향이라는 것이 있습니다. 결합방향은 같은 우선순위를 가진 연산자들이 나란히 배열되었을 때, 어디서부터 연산하는가를 나타내는 것입니다.

```
A = 20 + 40 + 100;
```

위와 같은 식에서 연산자 +는 결합방향이 → 이므로, 맨먼저 20 + 40 이 행해지고, 그 연산의 답인 60 과 60 + 100 이 나중에 수행됩니다.

대입연산자와 복합연산자

일반 적인 대입식의 경우에는 아래와 같이 간단하게 표현할 수 있습니다.

```
A = 100;
```

C 에서는 다음과 같이 표기하는 것도 가능합니다.

```
A = B = 100;
```

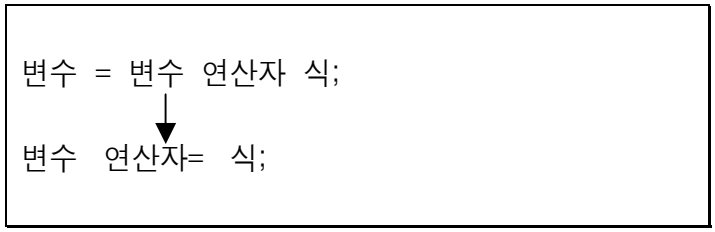
이럴 경우 A 와 B 가 모두 100 이 됩니다.

```
A = A + 100;
```

A 에 100 을 더하는 이런 식은 복합 연산자를 써서 다음과 같이 표현할 수 있습니다.

```
A += 100;       // A = A + 100 과 같은 의미
```

결과를 저장할 변수와 처음 등장하는 피 연산자가 동일 변수일 경우 복합연산자를 써서 식을 간단하게 표현할 수 있습니다.



7.2 관계 연산자

if 나 while 등의 제어문에서 자주 사용되는 것이 관계연산자 입니다. 관계연산자에는 다음과 같은 것이 있습니다.

연산자	의미
==	같다
!=	같지 않다
<	작다
<=	작거나 같다
>	크다
>=	크거나 같다

관계연산자에 의한 연산결과는 <참>또는 <거짓>의 2 종류만 있습니다. C 언어에서는 <참>일 경우에는 0 이 아닌수, <거짓>일 경우에는 0 으로 표현합니다. 따라서 결과가 0 인가 아닌가만을 가지고 참,거짓을 판단할 수도 있습니다.

참 = 0 이 아닌수, 거짓 = 0

다른 언어를 사용해본 경험자는 연산자 사용에 주의를 해야 합니다. BASIC 등에서는 "같지 않다"는 <>로 표현하지만, C 언어에서는 !=로 표현하고, "같다"는 ==로 표현하는 것에 주의해야 합니다.

C 초보자가 흔히 범하는 실수

If 문에서 어떤수가 1 인지 비교할 경우 if (a == 1) 로 작성해야 하는데, 초보자의 경우에는 if (a = 1)으로 작성하는 경우가 종종 있습니다. 이 경우에는 변수 a 에 1 을 넣어버리기 때문에 결과는 엉뚱하게 나타나지만, 컴파일시 에러가 발생하지 않으므로 비교적 찾기 힘든 에러에 속합니다. 따라서 If 문을 사용할때에는 항상 주의를 기울여야 합니다.

7.3 ++i 와 i++의 차이점

증가 연산자 ++는 1 을 더하고, 감소 연산자 --는 1 을 감소하는 기능을 가지고 있습니다.

```
++i      // i 를 1 증가합니다.
--i      // i 를 1 감소합니다.
```

증가 연산자 (또는 감소 연산자)가 변수명 앞에 오는 경우에는 먼저 증가 (감소)한 후에 다음 연산에 사용됩니다.

```
B = 0
A =  ++B * 10
결과 : A = 10,  B = 1
```

위식의 결과는 10 이됩니다. B 의 값이 먼저 증가되어 1 이 된후에 10 과 곱해졌기 때문입니다. 하지만 아래와 같은 경우를 보면…

```
B = 0
A =  B++ * 10
결과 : A = 0,  B  = 1
```

결과는 0 이 됩니다. B 가 증가 되지 않은 상태에서 10 과 곱해진후, 나중에 B 가 1 이된 것입니다.

7.4 비트 연산

비트 연산이란 두 수사이에 AND 나 OR 연산 처럼, 비트 대 비트의 논리연산 및 쉬프트 연산을 통칭하는 말입니다. 비트연산을 표로 정리하면 다음과 같습니다.

비트 연산자	명칭	설명 및 사용예
~	1 의 보수	모든 비트를 반전처리 합니다. ~(10101111) → 01010000
&	AND 연산	AND 연산은 특정비트를 남기는 작업 (마스크)을 하기위해서 주로 사용합니다. 0xab & 0x0f → 0xb
|	OR 연산	OR 연산은 특정비트를 SET 하기 위해서 사용합니다. 00001010 | 11110000 → 11111010
^	XOR 연산	XOR 연산은 특정비트를 반전시키기 위해서 사용합니다. 00001010 ^ 00001111 → 00000101 두번째 값의 1 로 된 부분만 반전처리되었음을 알 수 있습니다.
<<	좌측 쉬프트 연산	쉬프트 연산자는 우변에 있는 피연산자에서 지정한 수만큼 좌변 피연산자의 비트를 좌측으로 쉬프트합니다. 왼쪽으로 밀려나가는 비트는 소멸되며, 오른쪽으로 밀려 들어오는 비트는 무조건 0 이 됩니다. A = 10100011 일경우 A = A << 3 하면 A 는 00011000 이 됩니다.
>>	우측 쉬프트 연산	쉬프트 연산자는 우변에 있는 피연산자에서 지정한 수만큼 좌변 피연산자의 비트를 우측으로 쉬프트합니다. 오른쪽으로 밀려나가는 비트는 소멸되며, 왼쪽으로 밀려 들어오는 비트는 무조건 0 이 됩니다. A = 10100011 일경우 A = A >> 3 하면 A 는 00010100 이 됩니다.

7.5 산술 연산자

산술 연산에서 사용하는 수식을 산술식이라고 부릅니다. 산술 연산자중 사칙연산자는 +, -, *, /의 네가지가 있으며 나머지 연산자는 %를 사용합니다.

연산자명	연산
+	덧셈
–	뺄셈
*	곱셈
/	나눗셈
%	나머지

나머지 연산자는 정수형에서만 사용할 수 있는 연산자 입니다. float 와 같은 실수형 변수에는 % 연산자를 사용할 수 없습니다.

나눗셈 연산자의 경우, 정수끼리 나눈 결과가 실수 (소수점 아래 수가 있는 경우)가 되어도 소수점 아래가 잘린 정수값이 됩니다. 예를 들어

5 / 4 는 1.25 이지만 결과는 1 이됩니다.

소수점 아래의 값을 취급하고자 할때에는 실수형 (float 형)으로 정의된 변수를 사용해서 실수형 상수와 연산해 주어야 합니다.

```
float   a;
a = 5.0;
a = a / 4.0;       // a 에는 1.25 가 저장됩니다.
```

8. 제어문과 루프

C 언어에서의 제어문은 모두 9 개가 있으며 선택문, 순환문, 점프문으로 구분됩니다.

선택문 : if-else 문, switch 문
순환문 : while 문, do-while 문, for 문
점프문 : break 문, continue 문, goto 문, return 문

8.1 if 문

C 에서의 if 문 조건식은 반드시 괄호로 묶어야 합니다. 다음은 if 문 사용예입니다.

가장 단순한 if 문	`If (조건식) 문장;`

블록을 사용한 if 문	`if (조건식) {`
	`  문장;`
	`  }`

간단한 if-else 문	`if (조건식) 문장;` `  else 문장;`

블록을 사용한 if-else 문	`if (조건식) {`
	`  문장;`
	`  }`
	`else {`
	`  문장;`
	`  }`

중첩된 if-else 문	`If (조건식)`
	`  if (조건식) 문장;`
	`  else 문장;`
	`else 문장;`

if-else if-else 문	`if (조건식) 문장;`
	`else if (조건식) 문장;`
	`else 문장;`

조건식 내부에는 <, >, ==등의 관계연산자가 사용되는 것이 일반적이고, 수식이나 대입식도 사용될 수 있으며, 결과가 0 이면 거짓, 0 이 아니면 참으로 인식하게 됩니다.

```
if (a = b + 10) k = 0;
  else k = 1;
```

위의 경우 b+10 의 결과 값을 a 에 저장하고, 결과가 0 이 아니면 k = 0 이 실행됩니다. 결과가 0 일 경우에는 거짓이 되므로 k = 1 이 실행됩니다.

여러 개의 조건이 논리합(and)으로 결합될 경우에는 다음과 같이 작성합니다.

```
If ((a > 10) && (a < 30)) k = 1;
```

a 의 값이 10 보다 크고 30 보다 작을 경우에만 k = 1 이 수행됩니다. 논리합(or)의 경우에는 &&대신 ||을 사용합니다.

8.3 switch case 문

여러 개의 if-elseif-else 문을 대신할 수 있는 명령이 바로 switch case 문입니다. switch case 문은 여러 개의 선택사항중에서 하나를 골라서 실행할 경우 유용하게 사용할 수 있습니다.

```
switch (식)
{
    case 상수 1:
          문장 1;
          break;
    case 상수 2:
          문장 2;
          break;
    case 상수 3:
          문장 3;
          break;
    default:
          문장 4;
}
```

switch 의 괄호안의 식과 case 문에 있는 상수을 비교해서 일치할 경우, 일치한 상수가 있는 문장을 수행합니다. 예를 들어 (식)과 상수 2 가 일치할 경우 문장 2 를 수행합니다. 각각의 문장의 끝에는 반드시 break 를 작성해 주어야 합니다. Break 를 만다면 switch 블록밖으로 제어를 옮기게 됩니다. 만약 break 가 없다면 아래있는 case 문을 또 실행하게 됩니다.

만약 switch 의 괄호안의 식과 어떠한 case 의 상수도 일치하지 않는다면 default 에 있는 문장이 수행됩니다.

```
switch (a) {
case 0:
     b = 0;
     break;
case 1:
     b = 12;
     break;
case 2:
     b = 23;
     break;
default:
     b = 50;
     break;
     }
```

맨끝에 있는 break 문은 생략해도 무방

위의 예는 a 의 값이 0 이면 b = 0 을 넣고, a 가 1 이면 b = 12 를 넣고, a 가 2 이면 b =23 을 넣고, 그 이외의 값이면 b = 50 을 넣습니다.

한 개 이상의 값을 동시에 만족시킬 경우에 대해서는 아래와 같이 작성합니다.

```
switch (a) {
case 0:
     b = 0;
     break;
case 1:
     b = 12;
     break;
case 2:
case 3:
     b = 23;
     break;
default:
     b = 50;
     break;
     }
```

2 또는 3 일때 b =23 을 넣습니다.

8.4 for 문

여러가지 순환문중에서 가장 사용 빈도수가 높은 것이 바로 for 문입니다. C 의 for 문은 여러가지 형태로 변형할 수 있기 때문에 거의 모든 루프구조에 등장합니다. 다음은 for 문의 일반적인 사용법입니다.

```
for (초기식 ; 조건식 ; 증감식 )  문장 ;

for (초기식 ; 조건식 ; 증감식 ) {
  문장;
  }
```

초기식에는 루프 제어 변수에 초기값을 대입하는 대입식을 기술합니다. 초기식을 생략할 수도 있습니다. 조건식에는 루프 제어 변수의 조건 범위를 검사하는 논리식을 기술합니다. 이 조건을 만족할때까지만 루프가 수행됩니다. 끝으로 증감식에는 루프 제어 변수의 증감을 처리합니다. 다음은 전형적인 사용 예입니다.

```
for (i = 0; i < 10; i++) {
     문장;                  // 0 부터 9 까지 1 씩 증가
     }

for (i = 10; i > 0; i--) {
     문장;                  // 10 부터 1 까지 1 씩 감소
     }

for (i = 0; i <= 99; i += 2) {
     문장;                  // 0 부터 99 까지 2 씩 증가
     }


i = 0;
for (; i <= 99; i += 2) {
     문장;
     }
```

초기식을 생략할수도 있음

초기식, 비교식, 증감식을 모두 생략하면 무한루프가 됩니다.

```
for (;;) {
    문장;
    }
```

무한 루프에서 탈출하려면 break 명령을 사용합니다.

```
i = 0;
for (;;) {
    if (i > 10) break;     // i 가 10 보다 크면 루프 탈출
    i++;
    }
```

쉼표 연산자를 사용해서 여러 개의 인수를 취급할 수도 있습니다.

```
for (i = 0, j = 0; i < 9; i++, j++) {
    문장;
    }
```

변수 i 와 변수 j 를 모두 0 으로 만든뒤, 두 변수 모두 1 씩 증가시켜가면서 루프를 수행할 수 있습니다.

8.5 while 과 do-while 순환문

while 문의 서식은 다음과 같습니다.

```
while (식)
{
문장;
}
```

while 문은 for 문과는 달리 식이 하나밖에 없습니다. 이 식은 for 문에서의 조건식에 해당하는 것으로, 반복의 조건을 나타내고 있습니다. 이 식이 참일 동안은 블록내의 문장을 반복 실행합니다.

```
while(1) {

    문장;
    }
```

이렇게 한다면 무한루프가 됩니다.

do-while 문은 조건식이 뒤에오는 while 문이라고 생각하면 됩니다.

```
do
{
  문장;
} while (식);
```

while 문과의 차이점은 일단 문장이 한번은 수행된뒤에 식의 참,거짓여부를 판단하는 것입니다. 바꿔말하면 while 문에서 식이 거짓일 경우. 블록내의 문장이 단 한번도 실행되지 않게 되지만, do-while 문의 경우는 식이 거짓이라도 한번은 문장이 수행되는 것입니다.

8.6 break 와 continue 점프문

루프의 조건 검사를 무시하고 루프를 즉시 빠져나가야 할 경우가 종종 있습니다. 이 경우 break 문을 사용하면 루프를 무조건 벗어날 수 있습니다. break 문은 while, for, do-while, switch 등의 4 가지 루프에서 가장 안쪽 루프 하나만 벗어나게 해주는 명령입니다.

switch 문에서 break 명령은 필수적이지만, 나머지 루프에서는 선택적으로 사용할 수 있습니다.

```
do {
    if (조건) break;
} while (1);
```

여기서 조건과 break 문은 무한루프를 빠져나가는 유일한 탈출구를 제공하고 있습니다.

C 언어에는 break 와 비슷한 기능을 하는 제어문이 하나 더 있습니다. Continue 라는 명령인데, contine 문은 break 문과 마찬가지로 아직 실행하지 않은 부분을 건너뛴다는 점에서는 동일한데, break 문은 그냥 루프를 빠져나오는데 반해, continue 문은 루프의 조건식을 검사하는 부분으로 다시 되돌아간다는 점에서 틀린 명령입니다.

```
for (i = 0; i < 100; i++) {
    if (i % 2 == 0) continue;     // I 가 짝수이면 다시 윗줄로..
    j = j + 4;
}
```

9. 함수

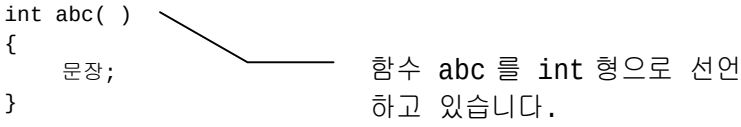
함수는 프로그램 내에서 어떤 특정한 작업을 전담할 수 있도록 독립적으로 만들어진 하나의 단위를 말합니다. 즉, 함수는 일종의 규격화된 서브루틴이라고 할 수 있습니다.

함수는 마치 블랙박스와 같이, 함수 내부의 구체적인 동작원리를 몰라도, 매개변수 (파라메터)를 주고 콜하면, 리턴값을 돌려주는 구조를 취하고 있습니다. 이런 구조는 체계적이고 구조적인 프로그램을 작성하는 데 있어서 많은 도움이 됩니다.

C 언어는 철저한 함수위주의 언어라고 할 수 있습니다. 가장 기본적은 main 함수 조차 함수의 형식으로 되어 있으며, printf 등의 기본명령도 함수의 형태를 취하고 있습니다.

9.1 함수의 형

함수란 계산의 결과값을 구하는 것이므로, 당연히 결과에 적합한 형을 가지고 있습니다. Int 나 char 등의 형을 지정하거나, 또는 특별한 형을 지정하지 않을 경우 void 형이라는 (아무것도 없다는 뜻)것도 있습니다.



9.2 함수 정의

C 에서 프로그램을 짠다는 것은 곧 유저 함수를 만든다는 것을 의미할 정도로 함수를 만들어나가는 것은 중요한 일입니다. 함수는 다음과 같은 형식으로 되어 있습니다.

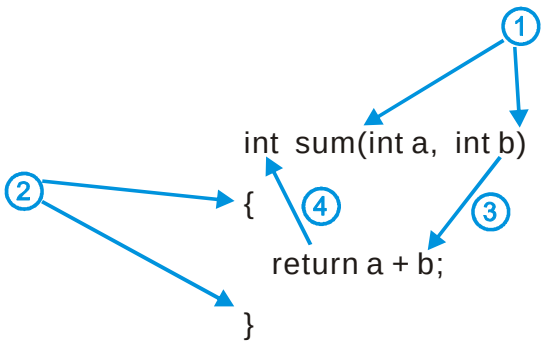
```
함수의형  함수명 ( 가인수열 )
{
  함수 본체
  [return 리턴값]
}
```

함수의 형이란 함수의 결과값의 형을 의미합니다. 함수명은 유저가 정의하는 이름입니다. 최대 256 문자까지 사용할 수 있지만, 보통은 16 자 내외에서 사용합니다. (변수명 작성규칙을 그대로 적용받게 됩니다. 예약어는 사용할 수 없고, 숫자로 시작되는 문자열은 사용할 수없습니다.)

가인수열은 함수 호출시 전달할 파라메터를 정의한 부분입니다. 가인수열에는 인수의 형과 가인수명을 함께 적어주어야 합니다.

Return 문은 void 형 함수의 경우에는 등장하지 않지만, int 나 long 형 함수와 같이 어떤 값을 리턴하는 함수에서는 반드시 등장해야 합니다. Return 명령을 만나면 함수의 실행이 종료되고 리턴값을 반환하게 됩니다. 이때 리턴값의 형은 함수의 형과 일치해야 합니다.

함수정의한 예를 보도록 하겠습니다.



- 1 번 : 함수명 뒤에는 반드시 괄호가 필요합니다.
- 2 번 : 대괄호는 함수의 내용 전체를 감싸줍니다.
- 3 번 : 함수의 인수는 형(type)과 함께 나열합니다. 인수는 함수내에서만 사용합니다.
- 4 번 : return 하는 값은 함수의 형(type)과 동일한 형을 리턴해야 합니다.

다음의 예처럼 main 함수 보다 아래에 유저 정의함수가 위치해 있을 경우에는 함수 선언이 필요합니다.

```
int sum(int a, int b);

void cmain (void)
{
        int a, b;
        a = 5;
        b = 4;
        b = sum (a, b);
}

int sum(int a, int b)
{
        return a + b;
}
```

맨뒤에 세미콜론을 붙이면 함수 선언이 됩니다.

Sum 함수가 뒤에 있어도, 함수선언이 있으므로, 에러가 발생하지 않습니다.

함수 선언이 필요한 이유는 함수의 형이나 파라메터 형식등을 다른 함수들에게 알리는 역할을 하기 때문입니다. 함수 선언이 없다면, 함수가 제대로 사용되었는지를 알 수 없기 때문에 에러가 발생됩니다.

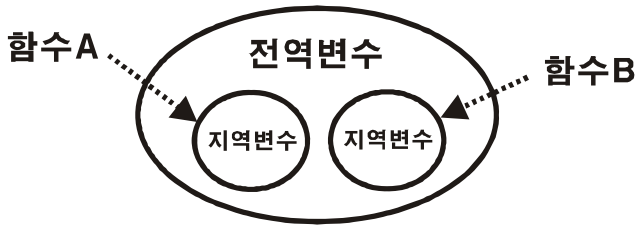
10. 함수에 변수를 전달

C 언어에서 함수로의 데이터 (변수)전달은 "값을 전달하는 방식(Call by value)"이 채택되고 있습니다. C 프로그램은 함수의 집합이므로, 각 함수사이에는 어떠한 값을 주고 받아야 하는데, 이때 전달되는 값을 인수(Parameter)라고 부르고, 돌아오는 값을 리턴값 (Return Value)라고 합니다. 인수의 전달방법에 대해서 구체적으로 설명하기 전에 지역변수와 전역변수에 대해서 알아보겠습니다.

10.1 지역변수와 전역변수

전역변수란 함수 바깥에서 선언한 변수이며, 모든 함수에서 이용할 수 있는 변수를 말합니다. 지역변수는 함수 내에서 선언한 변수이며, 그 함수내에서만 사용할 수 있는 변수를 말합니다.

```
int a;                  // 전역변수 a 를 선언
void func1 ( )
{
    int b;              // 지역변수 b 를 선언
    a = 0;
    b = 0;
}
```



10.2 전역변수에 의한 인수전달

전역변수에 있는 어떤값을 다른 함수에 전달하고자 할때에는 특별한 조치를 취하지 않아도 됩니다. 전역변수에 어떤 값을 넣어놓고, 해당 함수에서 그 값을 처리할 수 있기 때문입니다. 언뜻 보면 편리한 것 같지만, 함수 사이에 독립성을 해치고, 전체 프로그램을 모호하게 만들기 때문에, 꼭 필요한 것이 아니라면 전역변수 사용을 자제하는 것이 좋습니다.

10.3 지역변수의 내용을 전달

지역 변수는 함수내에서만 통용되는 변수이기 때문에, 이 값을 다른 함수에 전달하기 위해서는 파라메터를 사용해서 전달해야 합니다. 파라메터를 받는 함수에서 아무리 값을 변경해도 본래의 변수에는 영향을 주지 않기 때문에 함수의 독립성이 보장됩니다.

```
int sum(int a, int b)
{
    return a + b;
}

void cmain (void)
{
    int a, b;
    a = 5;
    b = 4;
    b = sum (a, b);
}
```

지역변수 a 와 b 의 값을 전달

11. 배열

배열이란 동일한 형을 가진 데이터(변수)들의 모임이라고 할 수 있습니다. C 언어에서는 다음과 같은 문법으로 배열을 선언합니다.

```
배열요소형  배열명 [배열크기];
```

```
int power [11];    // power 라는 1 차원배열 11 개 요소를 선언
```

위의 예에서 배열 크기를 11 로 지정했을 경우, 배열 첨자는 0 부터 시작해서 10 까지 사용가능합니다. C 에서는 배열 첨자가 항상 0 부터 시작됩니다. 다음 배열을 정의하고 배열요소를 모두 0 으로 클리어 시키는 예제 프로그램입니다.

```
void cmain (void)
{
    int i;
    int array[3];
    for (i=0; i<3; i++)
          array[i] = 0;
}
```

다음과 같이 사용하면 배열 정의와 동시에 초기값을 넣을수도 있습니다.

```
int array[3] = {11, 33, 123};
```

첨자가 1 개 이상인 경우를 다차원 배열이라고 합니다. 다음은 첨자가 2 개인 2 차원 배열의 선언예입니다.

```
int month[3][2] = {{10,10}, {20,20}, {30,30}};
```

배열이름만 사용했을 경우 그 배열을 가르키는 포인터가 됩니다.

```
int arrayOne[100];
startConvert(arrayOne); // 이 함수로 배열의 번지만 전송합니다.
```

11.1 문자 배열

베이직에서는 문자열 변수가 따로 있어서 문자열을 보관하는 것이 가능했지만, C 언어에서는 문자열 변수라는 것이 존재하지 않습니다. C 언어에서는 문자를 배열의 형태로 처리하는 방식을 사용합니다.

문자 배열은 8 비트의 문자 데이터를 저장할 수 있는 char 형으로 선언해야 합니다. (문자 하나는 8 비트의 ASCII 코드로 표현됩니다.) 문자열을 저장하기 위해서는 문자열의 크기에 맞는 배열을 미리 준비해 두어야 합니다. 다음은 문자 배열의 선언과 동시에 초기화를 하는 예입니다.

```
char str[6] = “testa”;
```

이렇게 하면, 각각의 배열 요소에는 다음과 같은 값이 들어 있습니다.

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]
t	e	s	t	a	0(null)

일반적인 배열과 다른 점이 있다면 가장 뒷부분에 문자열의 끝을 의미하는 NULL 문자가 자동적으로 추가된다는 점입니다. 이처럼 문자 배열에는 항상 문자열의 끝을 의미하는 NULL 문자가 따라다니게 됩니다. 따라서 문자배열의 크기는 실제 취급할 문자열보다 +1 된 수로 선언되어야 합니다.

만약 문자열의 크기를 정확히 모른다면 다음처럼 선언하는 것도 가능합니다.

```
char str[] = “comfile technology”;
```

12. 포인터

포인터는 곧 번지를 의미합니다. 더 정확히 말하자면 <변수의 내용이 보관되어 있는 곳의 어드레스>입니다.

```
int *a;
```

어떤 변수를 포인터 변수로 선언할때에는 위와 같이 선언합니다.

```
a = 0x10;
*a = 0;
```

라고 하면 0x10 번지에 0 을 넣으라는 의미입니다. a 만 따로 썼을때는 해당변수의 번지를 의미합니다.

여기서 잠깐!

```
int *a;
```

포인터 선언문에서 * (아스트리크 / 별표라고도 읽음)는 "포인터선언자" 역할입니다.
변수 a 를 포인터 변수로 선언하겠다는 뜻입니다.

```
*a = 0;
```

하지만, 수식에서 쓰이는 * 는 "단항연산자"입니다.

단항연산자 *의 피연산자는 언제나 포인터(번지값) 이어야 합니다. 단항연산자 *가 하는 일은 그 포인터가 가리키고 있는 번지에 저장되어 있는 내용을 지시하는 것입니다. 이곳에 저장하거나 읽어오라는 뜻입니다.

```
*a = 0;  // 포인터 a 가 가르키는 번지에 0 을 넣습니다.
b = *a;  // 포인터 a 가 가르키는 번지에 들어있는 값을 읽어서 변수 b 에 저장합니다.
```

12.1 포인터와 문자열

문자열은 포인터를 사용하면 간단하게 조작할 수 있습니다.

```
char *ps;
ps = "TEST";
```

"TEST"와 널문자는 메모리상에 보관되고, 선두 번지가 포인터 변수인 ps 에 대입됩니다. 즉, ps = "TEST"가 의미하는 것은 문자열 "TEST"의 선두번지를 ps 에 할당하라는 뜻이며, 문자열 "TEST"와 널문자는 ps 가 가르키는 번지부터 5 바이트에 저장됩니다.
여기에서 *ps 는 "T"를 가르키게 됩니다. 마치 문자배열에서의 첫번째 요소를 참조한 것과 같은 결과입니다. 다른 번지는 다음과 같은 수식으로 참조 할 수 있습니다.

*ps = "T"
*(ps + 1) = "E"
*(ps + 2) = "S"
*(ps + 3) = "T"
*(ps + 4) = NULL

일반적인 변수의 주소 참조

포인터 변수를 선언하지 않고, 일반적인 변수의 주소를 알아낼 수 있는 방법도 있습니다. 바로 &연산자를 사용하는 방법인데, &연산자는 해당 변수의 저장 번지를 나타냅니다.

```
int *a;
int data;
a = &data;      // data 의 번지를 포인터번지로 강제할당합니다.
*a = 0;         // 해당 번지를 0 으로 클리어합니다.
```

여기서 잠깐!

모아콘에서는 포인터를 쓰지 않아도 프로그램 작성이 가능합니다. 포인터는 소스코드를 난해하게 만들어, 추후 독해하는데 장애가 됩니다. 뿐만 아니라 이식할 때, 타 시스템과 호환성에서도 문제가 되어, 최근에는 쓰지 않는 추세입니다. 여러분들은 포인터가 대충 어떤것인지만 알아두고, 소스에서는 가급적 사용하지 마시기 바랍니다.

13. printf 출력

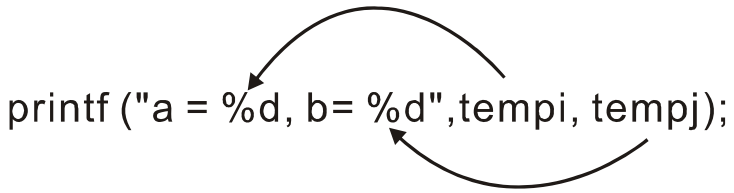
printf 는 C 언의 대표적인 표준 출력 라이브러리 함수입니다. 모아콘에서는 디버그용도로 사용합니다. Printf 실행결과는 모아콘스튜디오의 디버그창에 표시됩니다.

printf 단순하게 " "로 둘러쌓인 문자열을 출력할수도 있고, 어떠한 수치를 일정한 포맷으로 바꾸어 출력할 수도 있습니다.

포맷 문자열은 %로 시작되는데, 변수 a 의 값을 10 진으로 표시하려면 다음과 같이 하면 됩니다.

```
printf("a = %d", a);
```

여러 개의 포맷문자열을 포함하고 있는 경우에는 뒤에 나열된 변수(또는 상수)가 차례대로 적용됩니다.



여기에서 %d 가 10 진 출력을 위한 포맷 문자열입니다. 이외에도 다음 표와 같이 여러가지 포맷 문자열이 있습니다.

변환문자	변환형식	표시예
%d	정수치를 부호있는 10 진수로 표시	100 -10
%u	정수치를 부호없는 10 진수로 표시	100 12
%x	정수치를 부호있는 16 진수로 표시	Ab 12ab
%X	정수치를 부호있는 16 진수로 표시	AB 12AB
%c	1 바이트의 캐릭터	
%f	소수	0.123534
%s	널 종료문자를 만날때까지 출력	Korea
%e	부동소수점 표기법에 의한 출력	7.3458485e+07
%%	%문자 자체를 출력	%

%4d 와 같이 %와 d 사이에 숫자를 넣으면, 출력 칸수를 지정할 수도 있습니다. 맨앞에 0 을 붙이면 남는 공백을 0 으로 채웁니다. 다음은 printf 사용예입니다.

```
int x=345;
float y=34.564;
printf("%10d₩r₩n",x); /*x 를 10 자리에 맞추어 출력한다. */
printf("%-10d₩r₩n",x); /*x 를 10 자리에 맞추어 출력하고 출력방향을 왼쪽에 맞춘다. */
printf("%010d₩r₩n",x); /*x 를 10 자리에 맞추어 출력하고 남는 공백을 0 으로 채운다. */
printf("%.2f₩r₩n",y); /*y 를 소수점 이하 2 자리로 출력한다. */
```

%다음에 –를 넣으면 왼쪽 자리맞춤으로 표시됩니다. 주로 출력칸수와 함께 사용합니다.
%-10d (정수 d 가 1989 일경우) 결과는 1989bbbbbb (공백을 b 로 표시)가 됩니다.

*실수표시를 위한 %e 와 %f 의 경우, 전체자릿수 제한 표시가 적용되지 않습니다. (%10f 로 해도 전체 표시자릿수를 10 자리로 제한할 수 없습니다. 소수값 그 자체가 표시됩니다.
단 %.2f 와 같이 소수점 아래로 표시제한은 할 수 있습니다.)

위에 나열한 포맷문자열은 기타 다른 print 함수에서도 사용가능합니다. (clcdPrint, comPrint, netPrint, csgPrint 등)

```
while(1) {
      clcdPrint(1,0,"%d",i++);
      clcdPrint(1,1,"%.2f",j = j * 1.23);
      clcdPrint(3,2,"%6X",i);
      clcdPrint(4,3,"Technology");
      clcdPrint(10,0,"%2x %2x",rtcRead(0),rtcRead(1));
      delay(200);
      }
```

14. 프리 프로세서

프리 프로세서는 컴파일러가 컴파일을 수행하기에 앞서서 사전에 수행하는 변환작업을 의미합니다.

#include 는 지정한 파일을 소스내용에 포함시키는 역할을 수행합니다.
프로그램을 컴파일하기 전에 컴파일러가 알아야할 여러가지 정보 (예를들면, 라이브러리 정보, 디바이스 상황등)가 있는 파일을 포함시켜줍니다.

```
#include <파일명>
```

파일명만 적어줄 경우에는 소스가 저장된 폴더에 해당파일이 있어야 합니다. 그렇지 않은경우에는 Full path name 을 적어주어야 합니다.

```
#include <c:\program file\stdio.h>
```

#define 은 매크로정의를 위한 프리 프로세서 입니다.

```
#define  문자열 1 문자열 2
```

소스프로그램중 문자열 1 이 나오면 문자열 2 로 대치하는 역할을 수행합니다. 이 교환은 프로그램내의 모든 문자열에 대해서 무조건적으로 수행됩니다.

```
#define pie 3.141592
```

이후 소스에서 pie 는 3.141592 로 바뀌어 번역됩니다.

15. 변수참조

소스를 여러 개의 파일로 나누어 작업할 경우, 다른 소스에서 선언한 전역 변수를 참조할 필요성이 생기게 됩니다.

<소스 a>

```
int x=345;      // 최초 선언

void cmain(void) {

}
```

<소스 b>

```
extern int x;   // 같은 변수명으로 선언하되, 앞에 extern 을 붙입니다.

void functionB(void) {

}
```

소스 a 에서 변수 x 를 전역변수로 선언하고, 소스 b 에서 이것을 참조하고 싶다면, 소스 b 에도 변수 x 를 선언해주어야 합니다.

단, extern 을 앞에 붙여주면, 새로운 번지에 할당하지 않고, 기존 선언된 변수와 같은 영역을 공유합니다.

16. 함수 참조

하나의 프로젝트안에서 여러 개로 나누어진 소스에서 만들어진 함수는 기본적으로 서로 참조가 가능합니다.

단, 함수가 사용되기 전에 어떤형태 (반환값, 함수명, 매개변수등)로 으로 선언되었는지 사전에 알려줄 필요가 있습니다.

```
u16 sumAB(u16 x, u16 y);  // 다른 소스에서 선언된 함수를 쓸 수 있도록
                          // 함수의 형태를 컴파일러에 알려줌.

void functionB(void) {
  u16 result;
  result = sumAB(10,20);
}
```

C 에 대한 설명은 여기까지입니다. 더 자세한 사항을 알고 싶은신 분은 시중서점에서 판매하는 "ANSI C 언어 기초"관련서적을 참고하시기 바랍니다.

부록 2.
예제소스

예제 1 : 입력과 출력 연결

입력이 하나 들어오면, 출력을 On 하는 기본적인 예제 프로그램입니다.

본 예제 프로그램을 실행시키기 위해서, 다음과 같이 슬롯 0 번에는 입력 모듈 (CF-DIDC8 과 입력시뮬레이터 사용)을 슬롯 1 번에는 릴레이 출력 (CF-DORL8)을 연결하였습니다.

입력 하나가 들어오면 출력이 On

```
#include "moacon500.h"
void cmain(void)
{
  portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.
  while(1) { // 무한루프
        if (portIn(0)==1) portOn(10);
              else portOff(10);
        }
}
```

0 번 입력포트에 24V 가 입력되면 10 번 릴레이가 On 됩니다.

입력 2 개가 모두 들어오면 출력이 On

```
#include "moacon500.h"
void cmain(void)
{
  portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.
  while(1) { // 무한루프
        if ((portIn(0)==1) && (portIn(1)==1)) portOn(10);
              else portOff(10);
        }
}
```

0 번과 1 번 입력포트에 24V 가 입력되면 10 번 릴레이가 On 됩니다.
둘중 하나라도 입력이 안들어오면 출력은 Off 됩니다. 비교연산자가 AND 조건으로 되어 있습니다.

입력 2 개중 하나만 들어와도 출력이 On

```
#include "moacon500.h"
void cmain(void)
{
  portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.
  while(1) { // 무한루프
        if ((portIn(0)==1) || (portIn(1)==1)) portOn(10);
              else portOff(10);
        }
}
```

0 번과 1 번 입력포트중 하나에 24V 가 입력되면 10 번 릴레이가 On 됩니다.
둘중 하나라도 입력 들어오면 출력은 On 됩니다. 비교연산자가 OR 조건으로 되어 있습니다.

이와 같이 C 언어의 조건 연산자를 사용해서 여러 개의 입력상황의 조건을 검사하여 출력과 연동되도록 프로그램할 수 있습니다.

예제 2 : 타임루프 프로그래밍 기법 소개

타임루프 프로그래밍 기법을 알고 계시면, 모아콘으로 프로그램을 짜는데 많은 도움이 됩니다. 이 기법은 모아콘 뿐만 아니라 C 언어를 사용하는 다른 MCU 에서도 응용하실 수 있습니다. "타임루프 프로그래밍 기법"이라는 용어는 저희회사에서 작명한 것이므로 일반적인 명칭이 아닙니다.

여러분이 일반적으로 C 언어로 프로그래밍을 짜다보면, 반복적으로 처리해야될 일이 있을때 아래 처럼 루프를 구성하게 됩니다.

```
while (1) {
   proc_rtn1();
   proc_rtn2();
   }
```

while(1) 문으로 이 루프를 감싸게되면, 처리상황에 따라서 루프실행간격이 일정하지 않게됩니다.

이 루프가 항상 일정한 시간간격으로 실행되게 하는것이 "타임 루프" 프로그래밍의 핵심입니다.

과연 이렇게해서 무슨 이득이 있을까요?

바로 시간관리를 할 수 있다는 장점이 생기게 됩니다.

설명을 위해 다음과같은 프로그램을 작성해보도록 하겠습니다.

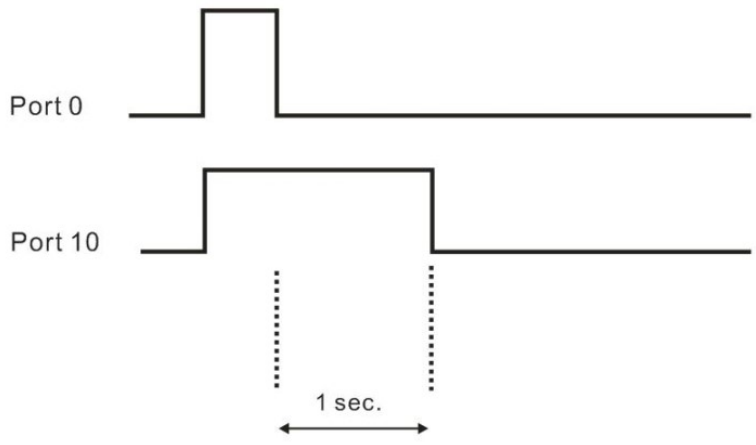
포트 0 에 입력이 들어오면 포트 10 을 On 하고,
포트 1 에 입력이 들어오면 포트 11 을 On 한다.

아주 단순한 프로그램입니다. 아래처럼 작성해 보았습니다.

```
#include "moacon500.h"
int mainTm=0;
```

```
void processA(void);
void processB(void);

void cmain(void)
{
 portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.

 while(1) { // 무한루프
 processA();
 processB();
  }
}

void processA(void)
{
 if (portIn(0) == 1)
  portOn(10);
 else
  portOff(10);
}

void processB(void)
{
 if (portIn(1) == 1)
  portOn(11);
 else
  portOff(11);
}
```

이 상황에서 아래 그림처럼, 입력조건이 잠깐 들어와도 출력은 1 초동안 On 상태를 유지하도록 프로그램해보겠습니다.



초보자라면 이런식으로 딜레이가 들어간 프로그램을 생각하셨겠죠?

```
portOn(10);
Delay(1000);
PortOff(10);
```

이렇게 한다면 delay 하는 1 초동안 다른 입력을 못받게 됩니다.

그럼 어떻게 할까요?

본격적으로 설명 드리기에 앞서, 모아콘에서 타이머 이벤트를 사용하는 방법부터 알아보겠습니다.

모아콘에서 아래 프로그램을 실행시키면 0.1 초마다 타이머 이벤트가 실행됩니다.

```
#include "moacon500.h"
int mainTm=0;
void cmain(void)
{
  portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.
  startTimerEvent(100); // 타이머 이벤트의 시작
  while(1) { // 무한루프
    }
}

// 0.1 초마다 이곳으로 옵니다.
void timerEvent(void)
{
  mainTm++;
  printf("mainTm value is %d \r\n",mainTm);
}
```

위 프로그램을 실행시키면 모아콘 스튜디오의 디버그창에 0.1 초 마다 증가되는 숫자를 보실 수 있습니다.

**

이제 "타임 루프 프로그래밍 기법"이 들어간 소스를 보여드리겠습니다.

프로세스를 입력과 출력으로 나누고, processInput 에서는 입력이 들어오면 timerA 또는 timerB 만 건드립니다.

출력프로세스인 processOutput 에선 timerA, timerB 가 0 이 아닐 경우에만 해당 출력을 On 시킵니다.

timerA, timerB 는 0.1 초마다 감소되는 타이머입니다. 0 이될때까지 감소시킵니다.

```
#include "moacon500.h"
int mainTm=0;
int timerA=0;
int timerB=0;

void cmain(void)
{
 portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.
 startTimerEvent(100); // 타이머 이벤트의 시작

 while(1) { }  // 무한루프, 여기서 남는시간을 소비
}

void processInput(void)        //   입력 처리
{
 if (portIn(0) == 1) timerA = 10;
 if (portIn(1) == 1) timerB = 10;
}

void processOutput(void)       // 출력처리
{
 if (timerA > 0)
  portOn(10);
 else
  portOff(10);

 if (timerB > 0)
  portOn(11);
 else
  portOff(11);
}

void timer(void)          // 메인타이머 처리
{
 mainTm++;
 if (timerA > 0) timerA--;
 if (timerB > 0) timerB--;
}

// 0.1 초마다 이곳으로 옵니다.

void timerEvent(void)    // 메인루틴
```

```
{
 timer();
 processInput();
 processOutput();
}
```

이렇게하면, 입력이 들어온뒤 입력이 꺼지더라도, 1 초간은 출력을 On 상태로 유지하게 됩니다.

공장자동화 분야에서 입력조건이 잠깐 살아도, 어느정도 출력을 유지해주어야 하는일은 매우 빈번하게 나오는 상황입니다.

이 골격을 유지하면서, 아무리 입력조건과 출력조건이 복잡해진다고 해도, 정확히 타이밍을 지켜가면서 출력이 나옵니다. 1 초간 출력을 유지하는 사이에 들어오는 신호를 놓치는 일도 없습니다.

이 기법의 장점은 노이즈에도 강한 프로그램이 된다는 것입니다. 모든 입력조건을 0.1 초마다 반복적으로 검사를하고 처리를 하니까, 노이즈가 잠깐들어와서 잘못된 출력결과가 나온다 하더라도, 0.1 초 뒤에는 다시 정상적인 출력이 나가게 됩니다.

"타임루프 프로그래밍"을 위해서는 2 가지의 중요한 규칙을 지켜주어야 합니다.

1. 함수의 처리시간이 기준시간을 초과해선 안됩니다. 0.1 초안에 모든처리가 끝나야합니다.

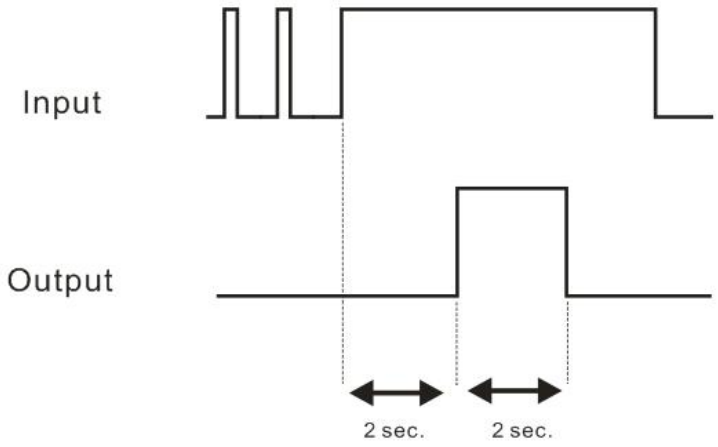
2. 함수내부에서 프로세싱을 잡아두면 안됩니다.

이게 무슨 말이냐면, 함수안에 delay 같은 시간을 끄는 함수나, while (1)같은 무한루프가 있어서는 안된다는 것입니다.

예제 3 : 타임루프 프로그래밍 기법 응용

산업현장에서 흔히 등장하는 노이즈 (채터링) 제거후 입력받기 예제 프로그램입니다.

아래 그림처럼 최초 2 초간 신호가 유지되어야 비로소 출력을 On 하는 상황입니다. 출력도 2 초간 On 하고 꺼져야 합니다. (실제로는 2 초까지 대기할 필요는 없습니다.
0.5 초~1 초미만이면 노이즈제거에 충분합니다.)



이렇게 되면, 짧은 노이즈가 유입되어도 출력이 나오지 않게됩니다. 입력이 하나라면, 구지 타임루프 프로그래밍 기법을 사용하지 않아도 됩니다.

하지만, 여러개 입력이 동시에 들어올 경우, 이 기법이 아니면 딱히 방법이 없습니다.

```
#include "moacon500.h"
int timerA = 0;
int timerB = 0;

void cmain(void)
{
  portInit(1,0); // 1 번 블록을 출력상태로 만듭니다.
  startTimerEvent(100); // 타이머 이벤트의 시작
```

```
  while(1) {} // 무한루프
}

void processInput(void)
{
   if (portIn(0) == 0) timerA = 40;
  if (portIn(1) == 0) timerB = 40;
}

void processOutput(void)
{
  if ((timerA > 0) && (timerA <20)) // 0 < timerA <20 일때 On
   portOn(10);
  else
   portOff(10);

  if ((timerB > 0) && (timerB <20)) // 0 < timerB <20 일때 On
   portOn(11);
  else
   portOff(11);
}

void timer(void)
{
  if (timerA > 0) timerA--;    // 0 보다 크면 1 감소
  if (timerB > 0) timerB--;
}

// 0.1 초마다 이곳으로 옵니다.

void timerEvent(void)
{
  timer();
  processInput();
  processOutput();
}
```

입력스위치를 짧게 On, Off 를 반복하면 출력이 On 되지 않습니다.

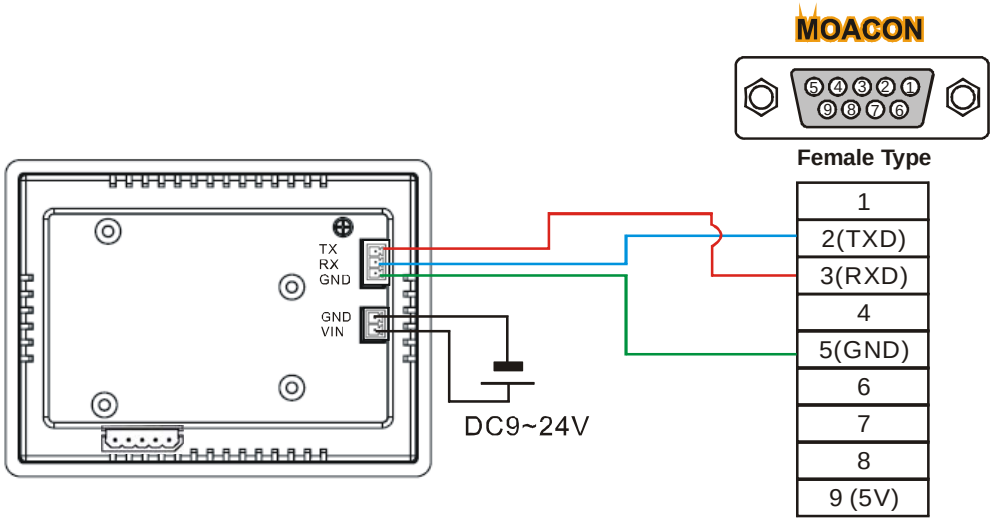
입력을 2 초이상 On 하고 있어야, 출력이 On 됩니다.

http://cubloc.blog.me/220069635656 이곳에서 동작 동영상을 보실 수 있습니다.

응용예제 1: UIF5K 와 모아콘 연결

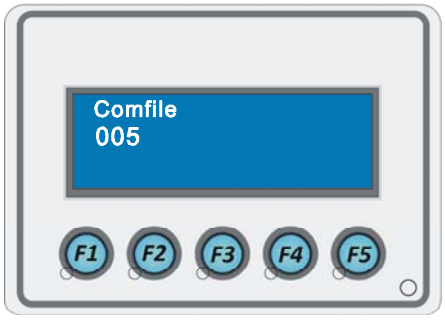
UIF-5K 는 LCD 디스플레이와 5 개의 버튼을 갖춘 유저인터페이스용 패널입니다. UIF-5K 는 RS232C 로 통신하도록 되어 있습니다. 모아콘의 RS232C 채널 0 을 통해서 UIF-5K 와 연결하실 수 있습니다.

다음과 같이 UIF-5K 와 MSB 시리즈를 연결하여 주십시오. UIF-5K 에는 9V~24VDC 전원을 별도로 연결해야 합니다.



다음 소스를 실행시키면, UIF-5K 의 LCD 화면에 Comfile 이라는 문자가 표시됩니다. 그리고 버튼을 누르면 해당 버튼의 스캔코드가 LCD 화면에 표시됩니다.

```
#include "moacon500.h"
void cmain(void)
{
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
  openCom(0,115200,C8N1);   // RS232 포트 0 번 오픈 (송신은 표시, 수신은 키입력)
  clcdPrint(0,0,"Comfile");  // Comfile 이라는 문자를 LCD 상에 표시합니다.

  while(1){
  if (comLen(0)) {  // UIF5k 의 아무키나 누르면 한바이트가 수신됩니다.
        clcdPrint(0,1,"%03d",comGet(0));  //입력된 키 스캔코드값을 화면에 표시합니다.
      }
        statusLed(1);  // STATUS LED 를 깜박거리게 합니다. (동작유무 확인용)
        delay(100);
        statusLed(0);
        delay(100);
  }

}
```



위 소스는 메인루프안에서 항상 키입력이 발생했는지를 검사하는 방식이기 때문에 다소 불편합니다. 다음 소스는 키입력이 발생했을때만 keyInProc( )함수를 수행하도록 한 것입니다. 타이머이벤트를 이용해서 0.5 초마다 키입력 유무를 검사하도록 한 것입니다.

```
#include "moacon500.h"
u8 keyPushed=0;
void cmain(void)
{
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
  delay(500);
  openCom(0,115200,C8N1);   // RS232 포트 0 번 오픈 (송신은 표시, 수신은 키입력)
```

```
   comPut(0,27); // beep sound on
   comPut(0,90);
   comPut(0,1);
   clcdPrint(0,0,"Comfile");  // Comfile 이라는 문자를 LCD 상에 표시합니다.
   startTimerEvent(500); // 타이머 이벤트의 시작
   while(1) { // 무한루프
         statusLed(1);  // STATUS LED 를 깜박거리게 합니다. (동작유무 확인용)
         delay(100);
         statusLed(0);
         delay(100);
         }
}

// 키입력 있을때만 이곳을 실행

void keyInProc(void)
{
   switch(keyPushed) {
   case 1:
         clcdPrint(0,1,"Key F1 pushed");
         break;
   case 2:
         clcdPrint(0,1,"Key F2 pushed");
         break;
   case 3:
         clcdPrint(0,1,"Key F3 pushed");
         break;
   case 4:
         clcdPrint(0,1,"Key F4 pushed");
         break;
   case 5:
         clcdPrint(0,1,"Key F5 pushed");
   }
}

// 0.5 초마다 이곳으로 옵니다.

void timerEvent(void)
{
   if (comLen(0)) {  // UIF5k 의 아무키나 누르면 한바이트가 수신됩니다.
         keyPushed = comGet(0);
         keyInProc();
       }
}
```

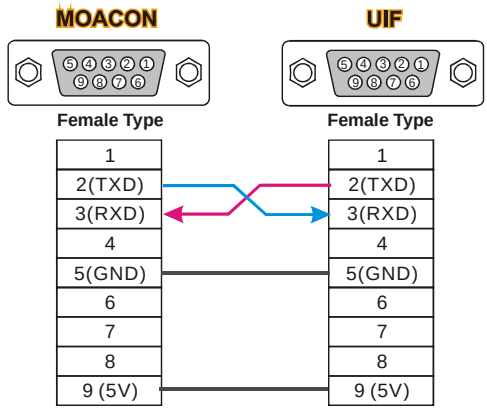
응용예제 2: UIF420A 와 모아콘 연결

다음은 UIF-420A 모듈과 연결하기 위한 라이브러리입니다. UIF-420A 는 LCD 디스플레이와 키입력을 한번에 해결할 수 있는 유저인터페이스용 제품입니다.

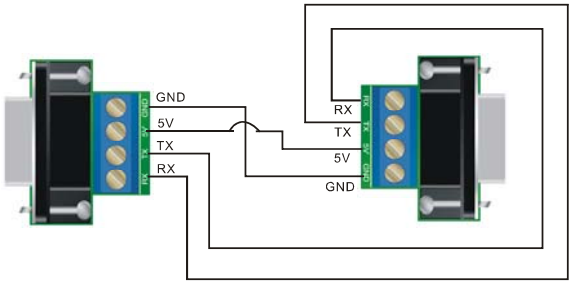
MOACON CPU 모듈의 RS232 채널 0 과 UIF-420A 의 RS232 포트를 연결합니다.

UIF-420A 에는 별도의 전원을 연결하지 않고, RS232 채널 0 의 5V 단자로부터 전원을 공급받습니다.

RX, TX 가 교차되어 있는 크로스 케이블을 사용해서 모아콘 CPU 모듈과 UIF 를 연결하십시오.



크로스 케이블이 없다면 "RS232 터미널블록 콘텍" (www.comfile.co.kr 에서 구입가능)을 사용해서 다음과 같이 연결하십시오.



딥 스위치는 다음 사진과 같이 선택합니다. (RS232 모드, 보레이트 57600)

UIF-420A 을 제어하기 위한 간단한 프로그램입니다. 아무키나 누르면 해당키의 스캔코드값이 UIF-420A 화면에 표시됩니다.

```
#include "moacon500.h"
void cmain(void)
{
  int i=0;
  u8 j;
  comPower(1);      // UIF-420A 에 전원을 공급합니다.

  delay(1000);       // UIF-420A 가 부팅할때까지 기다려줍니다.
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
  openCom(0,57600,C8N1);   // RS232 포트 0 번 오픈 (송신은 표시, 수신은 키입력)
  clcdPrint(0,0,"Comfile");  // Comfile 이라는 문자를 LCD 상에 표시합니다.

  while(1){
  if (comLen(0)) {  // UIF-420A 의 아무키나 누르면 한바이트가 수신됩니다.
      clcdPrint(0,1,"%03d",comGet(0));    //입력된 키 스캔코드값을 화면에 표시합니다.
      }
    statusLed(1);  // STATUS LED 를 깜박거리게 합니다. (동작유무 확인용)
    delay(100);
    statusLed(0);
    delay(100);
  }
}
```

유저인터페이스관련 라이브러리

UIF-420A 에 내장된 LCD 모듈은 CLCD 모듈입니다. 따라서 CLCD 관련함수를 써서 디스플레이를 할 수 있습니다.

하지만, RS232 를 통해 연결되어 있으므로, CLCD 관련함수의 출력이 RS232 포트로 출력되도록 다음함수를 먼저 사용해주어야 합니다.

clcdUartInit

void clcdUartInit (u8 channel)

channel : CLCD 모듈과 연결될 RS232 채널 번호

clcd 관련함수의 출력이 RS23C 포트를 향하도록 해주는 선언함수입니다. 이후 CLCD 관련 함수의 출력은 모두 해당 RS232 채널로 출력됩니다. (clcdPrint, clcdClr, clcdCmd, clcdCsr, clcdBlit)

comPower

void comPower (u8 onoff)

onoff : 0=off, 1=On

채널 0 에 있는 5V 전원출력을 On 하거나 Off 할 수 있습니다. 다음 그림과 같이 UIF 시리즈와 연결하였을 경우, UIF 시리즈로 전원을 공급하거나 차단할 수 있습니다.

*모아콘은 2011 년 이후버젼의 UIF 시리즈와 호환됩니다. 이전에 구입하신 UIF 를 가지고 계신분들은 저희회사에 연락하셔서 펌웨어 업그레이드를 받으시기 바랍니다.

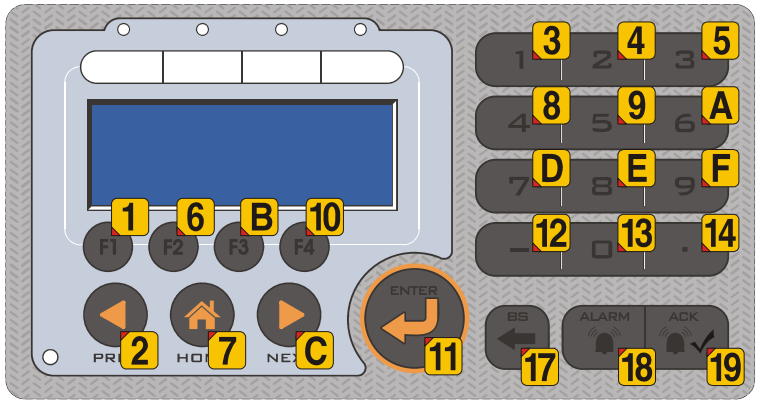
TIPS

UIF-420A 와 CLCD 모듈을 동시에 사용할 수도 있습니다. CLCD 모듈은 하단에 있는 I2C 포트에 연결하고, UIF-420A 는 RS232 포트에 연결한뒤, clcdUartInit 함수와 clcdI2cInit 함수를 써서 clcd 관련함수의 출력방향을 바꾸어 가면서 사용합니다.

```
#include "moacon500.h"
void cmain(void)
{
  int i=0;
  u8 j;

  clcdI2cInit(0); // 우선 i2c 포트의 clcd 로 할당.
  clcdPower(1);  // lcd 의 Power 를 On
  clcdCls();   // 이 함수는 i2c 포트로 출력됨
  clcdCsr(0);  // 이 함수는 i2c 포트로 출력됨

  comPower(1);  // uif-420a 의 파워를 on
  delay(1000);    // 초기화 시간 대기
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
  openCom(0,57600,C8N1);
  clcdPrint(0,0,"Comfile"); // 이 함수는 rs232 포트로 출력됨

  while(1){
    clcdUartInit(0); // clcd 를 RS232 로 할당
        if (comLen(0)) {
      clcdPrint(0,1,"%03d",comGet(0)); // 이 함수는 rs232 포트로 출력됨
      }
    statusLed(1);
    delay(100);
    statusLed(0);
    delay(100);

    clcdI2cInit(0); // clcd 를 i2c 포트로 할당
    clcdPrint(0,0,"comfile %03d",i++); // 이 함수는 i2c 포트로 출력됨

  }
}
```

UIF-420A 의 키보드 읽어오기

UIF-420A 의 키 스캔코드는 다음과 같습니다.

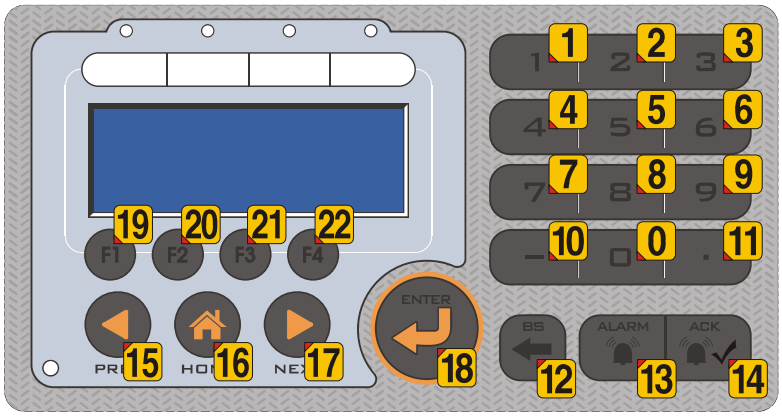


[본래의 키 스캔코드]

다소, 불규칙한 조합으로 되어있어 그대로 쓰기에는 불편합니다. 그래서 이 조합을 테이블 변환하여 좀더 쓰기 편리한 조합으로 바꾸는 것이 좋습니다.

```
#include "moacon500.h"
void cmain(void)
{

  const char uifscancode[] = {0xff,19,15,1,2,3,20,16,4,5,6,21,
                  17,7,8,9,22,18,10,0,11,0xff,0xff,12,13,14};
  int i=0;
  u8 j;
  comPower(1);  // uif-420a 의 파워를 on
  delay(1000);    // 초기화 시간 대기
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
  openCom(0,57600,C8N1);
  clcdPrint(0,0,"Comfile"); // 이 함수는 rs232 포트로 출력됨
  while(1){
        if (comLen(0)) {
      j = comGet(0); // 스캔 코드 읽어오기
      j = uifscancode[j]; // 테이블 변환
      clcdPrint(0,1,"%03d",j);
      }
  }
}
```

그러면 다음과 같은 스캔코드로 변하게 됩니다.



[쓰기좋게 변환한 키 스캔코드]

숫자키패드는 0 에서 9 까지 해당 숫자와 동일한 스캔코드를 반환합니다. 그 이외의 키패드는 10 이상의 숫자를 반환하게 됩니다.

키 스캔코드를 2 바이트로 수신

좀더 안정된 동작을 위해서, 2 바이트로 키 스캔코드를 보내는 방법이 있습니다.
키스캔코드를 정상적인 값과 반전된 값을 보내도록 하여, 수신측에서 두 바이트를 비교해 본뒤 일치하지 않으면, 통신도중 노이즈가 발생된 것으로 판단하여, 무시하도록 하는 것입니다.

이를 위해 최초 초기화 과정에서 UIF-420A 로 27, 72 코멘드 (10 진수)를 보내면, 이후부터 입력되는 모든 키 코드값은 2 바이트로 보내줍니다.

정상적인 키스캔코드 1 바이트	반전된 1 바이트
예) 0x03	예) 0xfc

2 번째 수신된 1 바이트를 다시 반전하여, 앞에 수신된 바이트와 일치하는 검사합니다.

```
#include "moacon500.h"
void cmain(void)
{

  u8 uif420a_scancode[] = {0xff,19,15,1,2,3,20,16,4,5,6,21,17,7,
       8,9,22,18,10,0,11,0xff,0xff,12,13,14};

  int i=0;
  u8 j,jb;

  comPower(1);  // uif-420a 의 파워를 on
  delay(1000);    // 초기화 시간 대기
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
  openCom(0,57600,C8N1);
  comPut(0,27);
  comPut(0,72); //2 바이트 스캔코드 송신 허가
  clcdPrint(0,0,"Comfile"); // 이 함수는 rs232 포트로 출력됨
  while(1){
        if (comLen(0)>1) { // 수신된 갯수가 1 바이트 이상이면
      j = comGet(0); // 스캔 코드 읽어오기
      clcdPrint(0,2,"%03x",j);
      jb = comGet(0); // 2 번째 스캔 코드 읽어오기
      clcdPrint(5,2,"%03x",jb);
      jb = ~jb;
      if (j == jb) { // 2 개가 일치할경우에만
```

```
        j = uif420a_scancode[j]; // 테이블 변환
        clcdPrint(0,1,"%03d",j);
        }
      } // if
  } // while
}
```

UIF 응용함수 1: 초기화와 키입력

앞의 프로그램을 이용하여 두가지 서브함수를 만들어보았습니다. 이 서브함수가 필요하신 분은 아래 소스에서 카피하여 사용하시기 바랍니다.

uif420aInit

void uif420aInit (void)

UIF420A 를 사용하기 위한 초기화 작업을 수행합니다. RS232 채널 0 을 57600 보레이트로 오픈하고, 키입력시 2 바이트를 송신하는 모드로 셋팅합니다.

uifKeyin

u8 uifKeyin (void)

UIF 에 키입력이 있는지 조사하는 함수입니다. 키입력이 있으면 해당 키 (쓰기 좋게 변환된) 스캔코드값을 반환합니다. 키입력이 없는 경우에는 99 를 반환하고, 통신중 에러가 발생된 경우에는 98 을 반환합니다.

다음은 위 두 함수를 포함한 테스트 소스 프로그램입니다.

```
#include "moacon500.h"
u8 uifKeyin(void)
{
  u8 j, jb;
  const char uifscancode[] = {0xff,19,15,1,2,3,20,16,4,5,6,21,17,7,
       8,9,22,18,10,0,11,0xff,0xff,12,13,14};
  if (comLen(0)<2) return 99;  //아무 입력이 없으면 99 반환
  j = comGet(0); // 스캔 코드 읽어오기
  jb = ~comGet(0); // 2 번째 스캔 코드 읽어오기
  if (j == jb)  // 2 개가 일치할경우에만
    return uifscancode[j];
  else
     return 98; // 통신에러
}

void uif420aInit(void)
{
  comPower(1);  // uif-420a 의 파워를 on
  delay(1000);    // 초기화 시간 대기
  clcdUartInit(0); // clcd 를 RS232 0 번채널로 할당
```

```
  openCom(0,57600,C8N1);
  comPut(0,27);
  comPut(0,72); //2 바이트 스캔코드 송신 허가
}


void cmain(void)
{
  u8 i;
  uif420aInit();
  clcdPrint(0,0,"Comfile"); // 이 함수는 rs232 포트로 출력됨
  while(1){
    i = uifKeyin();
    if (i < 90)
      clcdPrint(0,1,"%03d",i);
  } // while
}
```

UIF 응용함수 2: 숫자입력

이번에는 UIF 에서 숫자를 입력받는 방법에 대해서 알아보겠습니다. 언뜻 간단해 보이는 이 작업도 문자열변환과 배열등을 이해해야 작성할 수 있습니다. 다음 소스에서 이과정을 처리하는 하나의 함수를 만들어 보았습니다.

uifGetValue

void uifGetValue (u8 ux, u8 uy)

UIF 의 특정위치에서 숫자를 입력받는 함수입니다. 숫자키를 누르면 차례대로 표시하고, BS 키를 누르면 뒤에서부터 지웁니다. 최종적으로 Enter 키를 누르면 입력한 숫자를 Integer 형으로 반환합니다.
이 함수는 수행이 끝날때까지 프로세서를 붙잡고 있습니다. 즉, 유저가 키입력을 마치고 Enter 키를 누르지 않는다면, 이 함수에서 빠져나오지 않고 머물러 있게 됩니다. 이점 유의하시기 바랍니다.

맨앞에 있는 #include <stdlib.h>는 atoi 함수를 사용하기 위해서 포함시킨 것입니다. atoi 는 문자열을 숫자로 바꾸어주는 C 언어 기본함수입니다.

```
#include "moacon500.h"
#include <stdlib.h>

// 이 함수는 앞에서 설명되었으므로 생략했습니다.
// 본 소스를 실행하기 위해서는 이 함수를 원형그대로 포함시키십시오.
u8 uifKeyin(void);
void uif420aInit(void);

//
// Enter 가 입력될때까지의 숫자를 입력받고, 정해진위치에 표시
// 결과는 integer 형으로 반환
//
int uifGetValue(u8 ux, u8 uy)
{
  char val[11]; //최대 10 자리수까지 입력받음
  u8 loc=0,gch;
  int res;
  clcdLocate(ux, uy); //커서 위치 조정
  while(1) {
       while((gch = uifKeyin())<90) { //키입력이 있으면
```

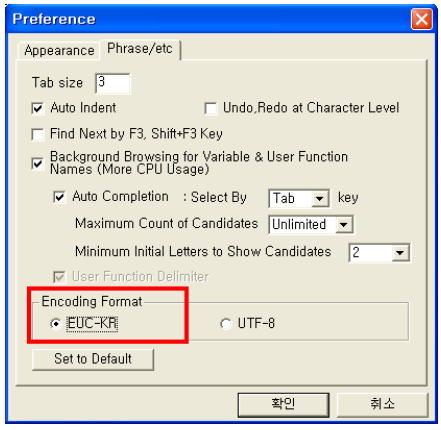
```
            if (gch < 10) { //숫자키만
                  if (loc < 9) {
                        val[loc] = 0x30 + gch; //ascii 코드로 저장
                        clcdPrint(ux,uy,"%c",val[loc]);
                        loc++; //10 자리로 제한
                        ux++;
                  } // if
            } // if
            if (gch==18) { // Enter 키 입력시
                  res = atoi(val); //문자열을 숫자로 변환
                  return res; // Enter
                  }
            if (gch==12) { //Bs 키 입력시
                  if (loc>0) {
                        loc--;
                        ux--;
                        clcdPrint(ux,uy," "); //지운자리에 공백을표시
                        clcdLocate(ux,uy); //커서위치도 조정
                        }
                  }
        } // while
   } //while
}

void cmain(void)
{
  u8 i;
  int valone;
  uif420aInit();
  clcdCsr(1); //커서를 On
  clcdPrint(0,0,"Input:"); // 이 함수는 rs232 포트로 출력됨
  valone = uifGetValue(6,0);
  clcdPrint(0,2,"Your Value:%d",valone);
  while(1){
  }
}
```

UIF 를 사용시, 여러가지 값을 유저가 직접 입력해야되는 경우가 많이 발생합니다. 이때 위 소스를 참고하셔서 프로그래밍하시기 바랍니다.

UIF416H 에서의 숫자입력

같은 프로그램을 UIF-416H 에서 실행시켜보았습니다. UIF-416H 는 한글표시가 가능한 UIF 제품군입니다. 사용법은 UIF-420A 와 거의비슷합니다.
그외에도 많은 기능 (선,원등의 그래픽표시, BMP 파일 다운로드)이 있는 제품입니다. UIF-416H 에 대한 자세한 설명은 www.comfile.co.kr 에서 해당제품의 사용설명서를 참고하시기 바랍니다.

UIF416H 는 KS 완성형 코드를 사용하므로 MOACON STUDIO 의 TOOLS 메뉴에서 EUC-KR 로 선택해야 합니다.
(Tools > Text editor preference > Phrase/etc > Encoding format 에서 선택가능)



```
#include "moacon500.h"
#include <stdlib.h>

void hlcdLocate(u8 cx, u8 cy)
{
   comPrint(0,"%cLO%c%c",0x1b,cx,cy);
}

u8 uifKeyin(void)
{
  u8 j, jb;
  const char uif_scancode[] = {0xff,19,15,1,2,3,20,16,4,5,6,21,17,7,
       8,9,22,18,10,0,11,0xff,0xff,12,13,14};
  if (comLen(0)<2) return 99;  //아무 입력이 없으면 99 반환
  j = comGet(0); // 스캔 코드 읽어오기
  jb = ~comGet(0); // 2 번째 스캔 코드 읽어오기
  if (j == jb)  // 2 개가 일치할경우에만
    return uif_scancode[j];
  else
     return 98; // 통신에러
}

void uif416hInit(void)
{
  comPower(1);  // uif-420a 의 파워를 on
  delay(1000);    // 초기화 시간 대기
  openCom(0,57600,C8N1);
}

//
// Enter 가 입력될때까지의 숫자를 입력받고, 정해진위치에 표시
// 결과는 integer 형으로 반환
//

int uifGetValue(u8 ux, u8 uy)
{
   char val[11]; //최대 10 자리수까지 입력받음
   u8 loc=0,gch;
   int res;
   hlcdLocate(ux, uy); //커서 위치 조정
   while(1) {
         while((gch = uifKeyin())<90) { //키입력이 있으면
               if (gch < 10) { //숫자키만
                     if (loc < 9) {
                           val[loc] = 0x30 + gch; //ascii 코드로 저장
                           hlcdLocate(ux,uy);
                           comPrint(0,"%c",val[loc]);
                           loc++; //10 자리로 제한
                           ux++;
```

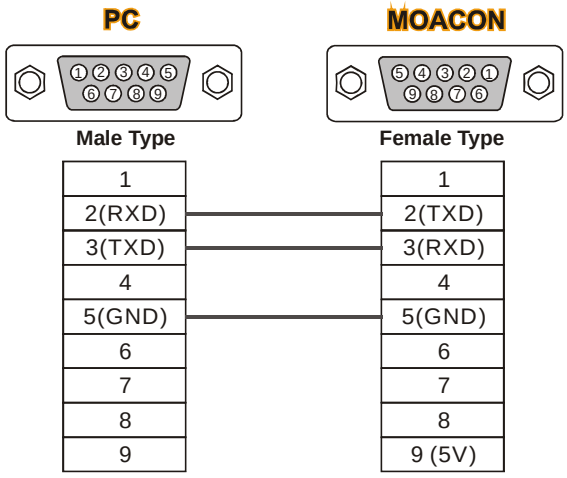
```
                  } // if
            } // if
            if (gch==18) {
                  res = atoi(val);
                  return res; // Enter
                  }
            if (gch==12) { //Bs
                  if (loc>0) {
                        loc--;
                        ux--;
                        hlcdLocate(ux,uy);
                        comPrint(0," ");
                        clcdLocate(ux,uy);
                        }
                  }
         } // while
   } //while
}



void cmain(void)
{
  u8 i;
  int valone;
  uif416hInit();
  hlcdLocate(0,0);
  comPrint(0,"입력:"); // 이 함수는 rs232 포트로 출력됨
  valone = uifGetValue(6,0);
  hlcdLocate(0,2);
  comPrint(0,"최종값:%d",valone);
  while(1){
  }
}
```

UIF-416H 는 별도의 코멘드를 보내지 않아도 2 바이트씩 스캔코드를 보내줍니다. 그리고 clcd 코멘드를 그대로 사용할 수 없어서, 문자열을 보내는 것은 comPrint 함수를 직접사용하였고, hlcdLocate 함수는 별도로 제작해 보았습니다.

```
void hlcdLocate(u8 cx, u8 cy)
{
   comPrint(0,"%cLO%c%c",0x1b,cx,cy);
}
```

응용예제 3: 모드버스 테스트

다음은 간단하게 모드버스동작유무를 테스트하는 방법입니다. 모아콘의 RS232C 채널 0 을 PC 와 연결하세요. (1:1 케이블 사용)



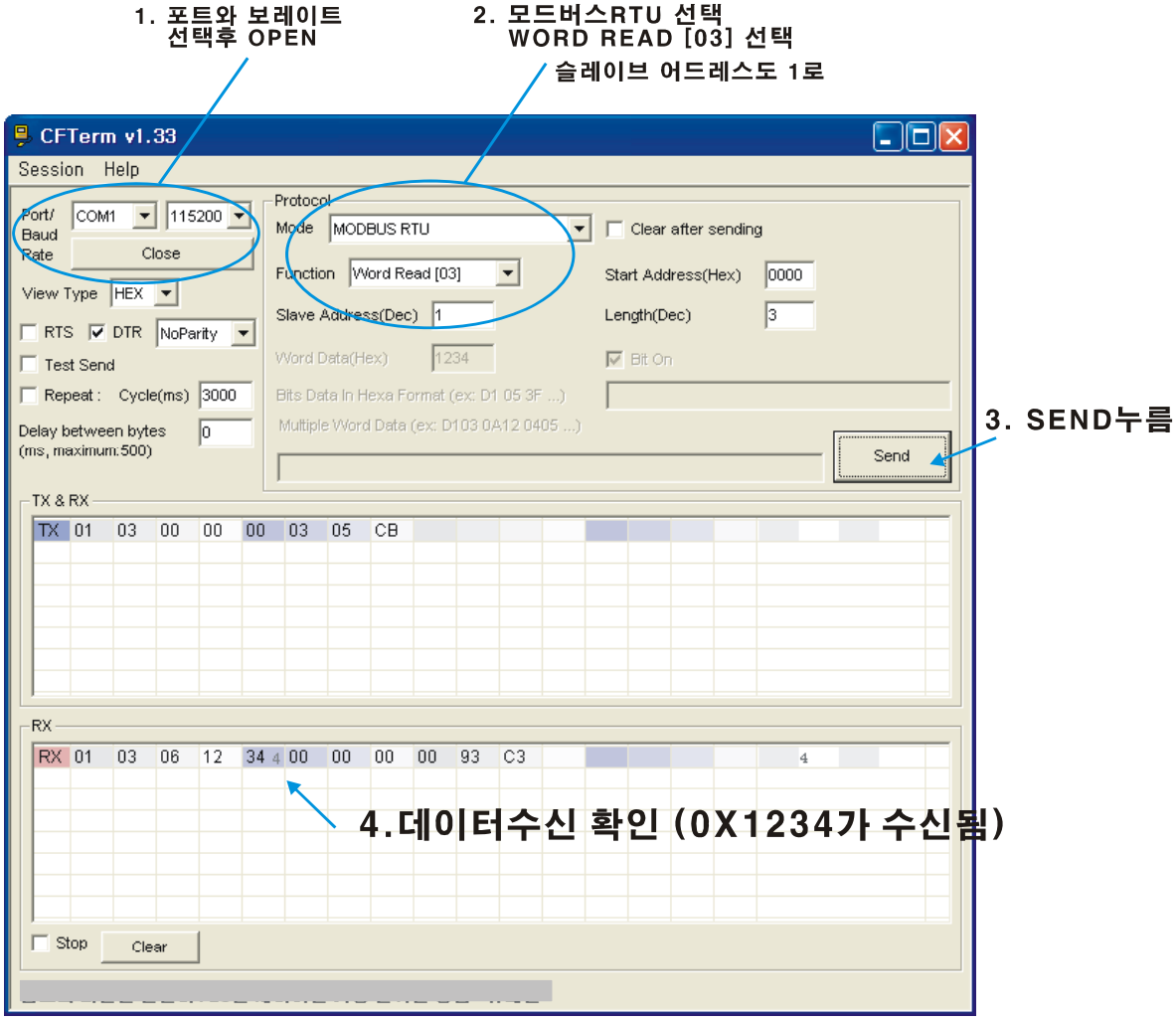
모아콘에는 다음 소스를 실행시키세요.

```
#include "moacon500.h"
void cmain(void)
{
  static u8 MBcoilBuffer[100];
  static u16 MBregisterBuffer[100];
  openCom(0,115200,C8N1);  // 슬레이브로 사용할 채널을 오픈
  startModbusRtu(0,1,MBregisterBuffer,MBcoilBuffer);
  MBregisterBuffer[0] = 0x1234;  // 0 번지에 0X1234 를 넣어둠
  while(1) {
    delay(500);
    statusLed(1);  // 동작유무 확인용 status LED 점멸
    delay(500);
    statusLed(0);
  }
}
```

모아콘을 MODBUS-RTU 슬레이브 상태로 동작하도록 만드는 소스입니다.

www.comfile.co.kr 자료실에서 다운로드받을 수 있는 CF-TERM 을 사용하여, 모아콘이 제대로 MODBUS-RTU 를 수행하고 있는지 테스트할 수 있습니다.

다음 번호에 적힌 순서대로 조작하여, 최종적으로 모아콘에서 응답이 있는지 확인하세요.



확인을 위해 소스에서 사전에 0 번지에 0X1234 를 넣어두었습니다. 이것을 WORD READ 코멘드로 읽어온 것입니다.

```
MBregisterBuffer[0] = 0x1234;  // 0 번지에 0X1234 를 넣어둠
```

별첨: ASCII 코드표

코드	문자
00H	NUL
01H	SOH
02H	STX
03H	ETX
04H	EOT
05H	ENQ
06H	ACK
07H	BEL
08H	BS
09H	HT
0AH	LF
0BH	VT
0CH	FF
0DH	CR
0EH	SO
0FH	SI
10H	DLE
11H	DC1
12H	DC2
13H	DC3
14H	DC4
15H	NAK
16H	SYN
17H	ETB
18H	CAN
19H	EM
1AH	SUB
1BH	ESC
1CH	FS
1DH	GS
1EH	RS
1FH	US

코드	문자
20H	SPACE
21H	!
22H	“
23H	#
24H	$
25H	%
26H	&
27H	‘
28H	(
29H	)
2AH	*
2BH	+
2CH	,
2DH	−
2EH	.
2FH	/
30H	0
31H	1
32H	2
33H	3
34H	4
35H	5
36H	6
37H	7
38H	8
39H	9
3AH	:
3BH	;
3CH	<
3DH	=
3EH	>
3FH	?

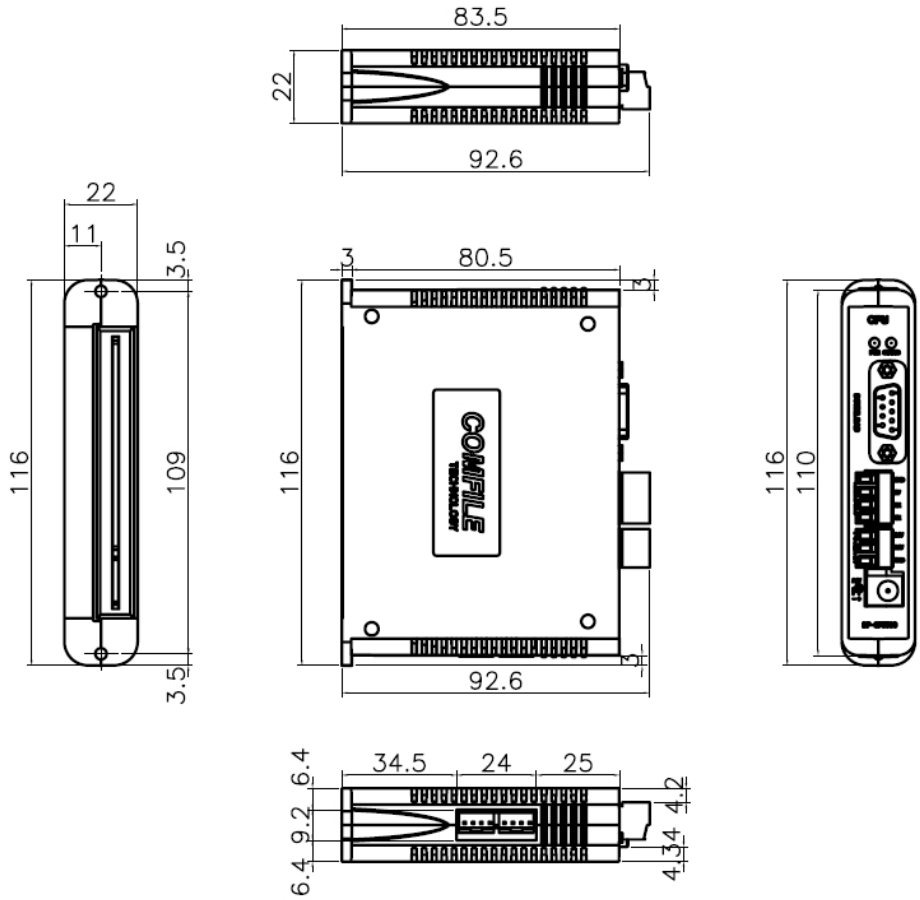
코드	문자
40H	@
41H	A
42H	B
43H	C
44H	D
45H	E
46H	F
47H	G
48H	H
49H	I
4AH	J
4BH	K
4CH	L
4DH	M
4EH	N
4FH	O
50H	P
51H	Q
52H	R
53H	S
54H	T
55H	U
56H	V
57H	W
58H	X
59H	Y
5AH	Z
5BH	[
5CH	₩
5DH	]
5EH	^
5FH	_

코드	문자
60H	`
61H	a
62H	b
63H	c
64H	d
65H	e
66H	f
67H	g
68H	h
69H	I
6AH	j
6BH	k
6CH	l
6DH	m
6EH	n
6FH	o
70H	p
71H	q
72H	r
73H	s
74H	t
75H	u
76H	v
77H	w
78H	x
79H	y
7AH	z
7BH	{
7CH	|
7DH	}
7EH	~
7FH	DEL

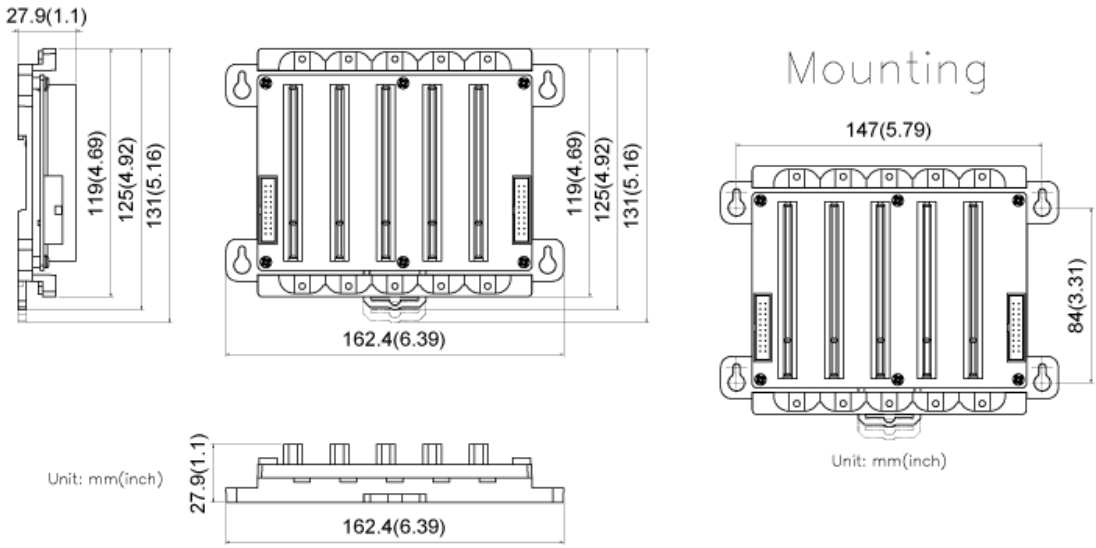
제품 외곽 사이즈

(단위 : mm)

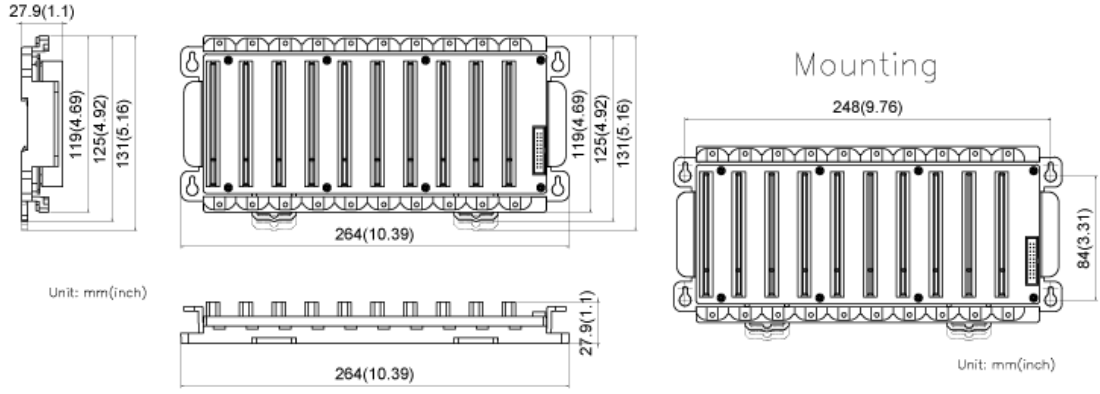
-모아콘모듈



-5 슬롯 보드



-10 슬롯 보드



<끝>