320 x 240 모노 터치 콘트롤러

CT172X/C

BASIC 언어중심

사용설명서 Version :200808

COMFILE
TECHNOLOGY

컴파일 테크놀로지 주식회사
www.comfile.co.kr

등록상표

WINDOWS 는 Microsoft Corporation 의 등록상표입니다.
CUBLOC, CUTOUCH 는 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 저작권법에 의한 보호를 받고 있습니다. 따라서 본 제품 (제품에 대한 아이디어 및 설명서 및 기타 포함)의 어떠한 부분도 사전에 폐사와의 문서 동의 없이 복사되거나 변경, 재 생산할 수 없으며 또한 다른 언어로도 번역될 수 없습니다.

주의사항

인쇄된 설명서는 인쇄된 시점에서는 최신 버젼이지만, 인쇄된 후 시간이 경과된 뒤에 새로운 내용이 추가되거나, 기존 내용이 바뀔 가능성이 있습니다. 최신 버젼의 설명서는 항상 인터넷 홈페이지 (www.comfile.co.kr)에서 확인하시기 바랍니다.

본 제품 (CUBLOC 및 CT172X/C 그리고, 부속장치 일체) 를 사용하시다가 생긴 손해 및 손실에 대하여 저희 컴파일 테크놀로지 주식회사는 어떠한 책임도 없음을 명시하는 바입니다.

본 제품을 사용하기 이전에 반드시 본 사용설명서를 읽어본 뒤 사용하시기 바랍니다. 본 사용설명서를 충분히 읽어보지 않은 상태로 본 제품을 사용하는 것으로 인해 발생된 피해에도 저희 회사에서는 어떠한 책임도 없음을 명시합니다.

인쇄 내역

2008 년 08 월 배포함

CUTOUCH 제품 분류

다음과 같은 종류의 CUTOUCH 가 출시되어 있습니다. (2008 년 4 월기준)

모델명	CT1720	CT1721	CT1721C	CT2400
스크린	5.7 인치 모노	5.7 인치 모노	5.7 인치 모노	7 인치 칼라
그래픽엔진	GHLCD3224	GHLCD3224	GHLCD3224	인텔리 LCD
코어모듈	CB290	CB290	CB290	CB405
프로그램메모리	80KB	80KB	80KB	200KB
데이터메모리	28KB	28KB	28KB	110KB
I/O 사양				
입력전용	32 (TTL)	32 (DC24V)	32 (DC24V)	20 (DC24V)
출력전용	32 (TTL)	32 (NPN TR)	32 (NPN TR)	12 (NPN TR) 4 (RELAY)
A/D 입력겸용	8	8	8	NONE
고속카운터 입력겸용	2	2	2	2
기타 I/O	8	8	8	PWM 3
총합계	82 개	82 개	82 개	41 개

CT1720 에 ADDON 보드가 장착된 모델이 CT1721 입니다.

CT1721C 는 CT1721 에서 소폭 업그레이드 된 모델입니다. CT1721C 에서 추가변경된 사항은 다음과 같습니다.

- 뒷면케이스가 바뀌었습니다.
- RS232 케이블을 연결하는 각도가 변경되었습니다.
- RTC 칩이 바뀌었습니다. (정확도 증가)
- RTC 밧데리 백업 시간이 증가되었습니다.
- 데이터 메모리 밧데리 백업이 제외되었습니다. (추가로 밧데리 연결시에만 백업가능)
- 통신 채널 1 에 RS485 기능이 추가되었습니다. 점퍼스위치로 RS232 또는 RS485 로 선택. (송신허가 포트는 15 번)
- BMP 파일 다운로드가 가능합니다.
- 그외 나머지 부분은 기존 CT1721 과 동일합니다.

CT2400 은 칼라 큐터치 제품입니다. 이 제품은 인텔리 LCD (칼라, TFT)에 CB405 기반으로 된 제품입니다. 별도의 매뉴얼을 참조하시기 바랍니다. 참고적으로 CT172X/C 에서 작성된 소스프로그램은 CT2400 에서 실행되지 않습니다. 그래픽처리 명령어가 틀리기 때문입니다.

알려드립니다.

첫번째, 본 사용설명서에 수록된 예제 프로그램이 일부 PC 에서 실행되지 않는 경우에는 맨앞에 WAIT 명령어를 추가로 써넣어 주십시오.

수정전	수정후
Const Device = CT1720 Dim A As Integer For A=1 To 9 Debug "3 * " Debug Dec A Debug " = " Debug Dec 3*A,Cr Next	Const Device = CT1720 Wait 500 ‘ 약간 딜레이를 해줍니다. Dim A As Integer For A=1 To 9 Debug "3 * " Debug Dec A Debug " = " Debug Dec 3*A,Cr Next

PC 에서 DEBUG 창이 뜨기도 전에 CT172X/C 가 실행되는 것을 막기위한 조치입니다.

머리말

CT172X/C 는 터치패널과 그래픽 LCD, CUBLOC 콘트롤러를 일체형으로 만든 “자동제어용 터치스크린 컴퓨터”입니다. 최근에 터치스크린을 사용한 산업용기기 등이 증가하는 추세에 있습니다만, 터치스크린을 사용하기 위해서는 별도의 PLC 를 연결해야 하고, 복잡한 터치스크린 저작도구 사용법을 익혀야 하는 등의 불편함이 있었습니다. 또한, 고가의 터치스크린 가격은 원가상승의 주된 원인이 되어 왔습니다.

저희 컴파일 테크놀로지에서는 이러한 “터치스크린 + PLC’ 조합을 대체할 만한, 새로운 임베디드 콘트롤러로 CUBLOC 과 그래픽 LCD 를 결합한 CT172X/C 를 출시하게 되었습니다.

CT172X/C 는 지금껏 동종업계에서 볼 수 없었던 새로운 개념의 “비주얼 터치스크린 콘트롤러”입니다. 외관은 여러분이 지금까지 보아오던 터치스크린과 다를 바 없습니다만, CT172X/C 의 가장 큰 차이점은 BASIC 과 LADDER LOGIC 으로 프로그램을 작성할 수 있는 비주얼 콘트롤러라는 것입니다.

여러분은 BASIC 언어를 사용해서 LCD 화면에 그래픽과 한글, 영문, 숫자 등을 표시할 수 있으며, 터치스크린으로부터 유저가 입력한 좌표정보를 받을 수 있습니다. I/O 포트를 통하여 센서입력을 받거나 RELAY 등을 ON/OFF 할 수 있으며, AD 변환 DA 변환 등을 통하여 아날로그 신호도 처리할 수 있으며, RS232 를 통하여 외부기기와 데이터를 주고 받을 수 있습니다. LADDER LOGIC 으로는 기존 PLC 에서 가능한 시퀀스 제어 등을 처리할 수 있습니다.

CT172X/C 에는 플레쉬 메모리가 내장되어 있어, 유저가 작성한 BASIC 및 LADDER 프로그램을 기억하게 됩니다. PC 와는 SERIAL 포트를 통하여 다운로드 및 디버깅을 하게 되며, 프로그램이 작성이 끝난 뒤 PC 와의 접속을 해제하면 “스탠드 얼론”상태로 동작하게 됩니다.

CT172X/C 는 지금까지 여러분이 보아왔던 “터치스크린”과는 다른 개념과 방식으로 움직이는 새로운 제품입니다. 여러분이 만드시고자하는 제품을 좀더 저렴한 가격으로, 그리고 좀더 편리하고 빠르게 개발할 수 있도록 도와드릴것입니다.

컴파일 테크놀로지㈜

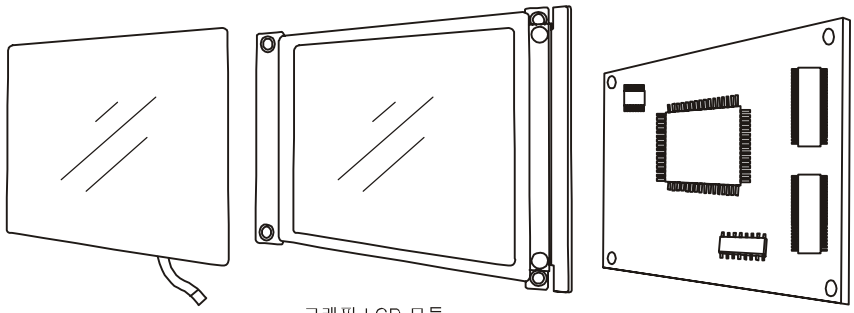
*** “ 레더로직” 을 사용하실 분들은 별도로 되어 있는 “ 사용자 설명서 Part2 레더중심 사용설명서” 를 참고하시기 바랍니다.**

차 례

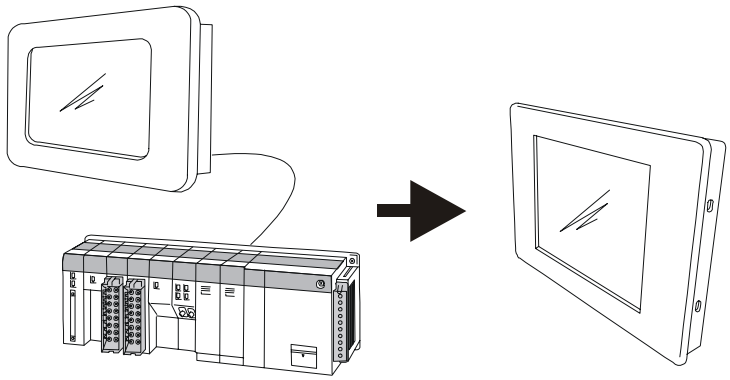
제 1 장
CT172X/C
의 기초

CT172X/C 의 개요

CT172X/C 는 CUBLOC (CB290) + 그래픽 LCD (GHB322C) + TOUCH PAD 를 일체형으로 만든 제품입니다. LINE, CIRCLE 과 같은 그래픽 전용명령어를 사용해서 그래픽 LCD 에 그림을 그릴 수 있으며, 터치지점을 읽어내는 명령어를 사용해서 유저가 터치한 지점을 알 수 있습니다.



CT172X/C 는 자동화 현장에서 흔히 볼 수 있는 "터치스크린"과 PLC 의 조합을 대체할 수 있는 새로운 방식입니다. 터치스크린+PLC 조합은 비용적인 부담이 상당하지만, CT172X/C 는 터치스크린과 PLC 을 한번에 해결할 수 있기 때문에 상대적으로 저렴한 비용으로 "터치스크린 콘트롤"을 구현할 수 있습니다.



기존 터치스크린과의 차이점

기존 제품은 터치스크린에서는 그래픽과 터치를 처리하고 별도의 PLC 에서 제어를 수행하도록 되어 있습니다. 이 방식은 원가상승의 부담이 있고, 서로간의 인터페이스를 위한 프로토콜 설정등의 작업을 추가로 구성해주어야 합니다.

그리고, CT172X/C 에서는 그래픽 및 터치입력 처리를 위해 BASIC 언어를 사용해야합니다. 기존 "터치스크린"에서는 프로그래밍 랭귀지를 사용하지 않고, 전용 그래픽 저작도구를 사용해서 화면을 구성해나가는 방식입니다만, CT172X/C 는 기본적으로 BASIC 언어를 사용해서 처음부터 끝까지 유저여러분께서 프로그램을 해주어야 합니다.

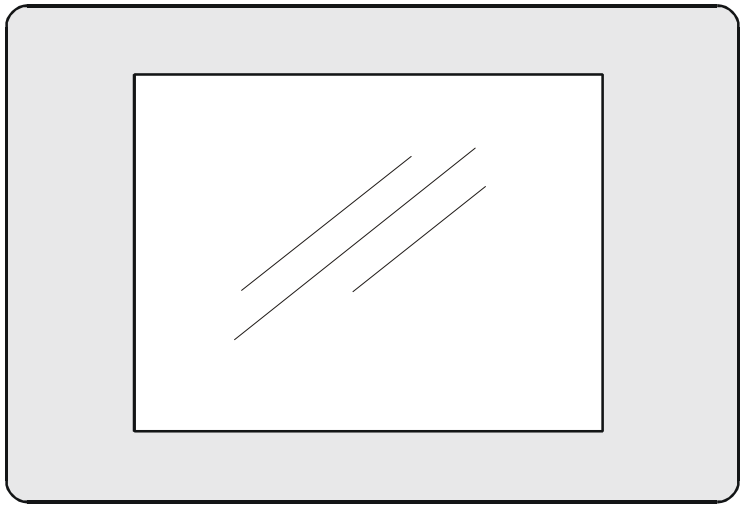
CT172X/C 에서는 LADDER LOGIC 도 지원합니다. 따라서 PLC (Programmable logic controller)에서 할 수 있는 일들을 처리할 수 있습니다. 입력포트와 출력포트를 내장하고 있어, 외부로부터의 센서입력이나, 릴레이제어등도 수행할 수 있습니다.

다음은 기존 터치스크린과 CT172X/C 의 차이점을 정리한 표입니다.

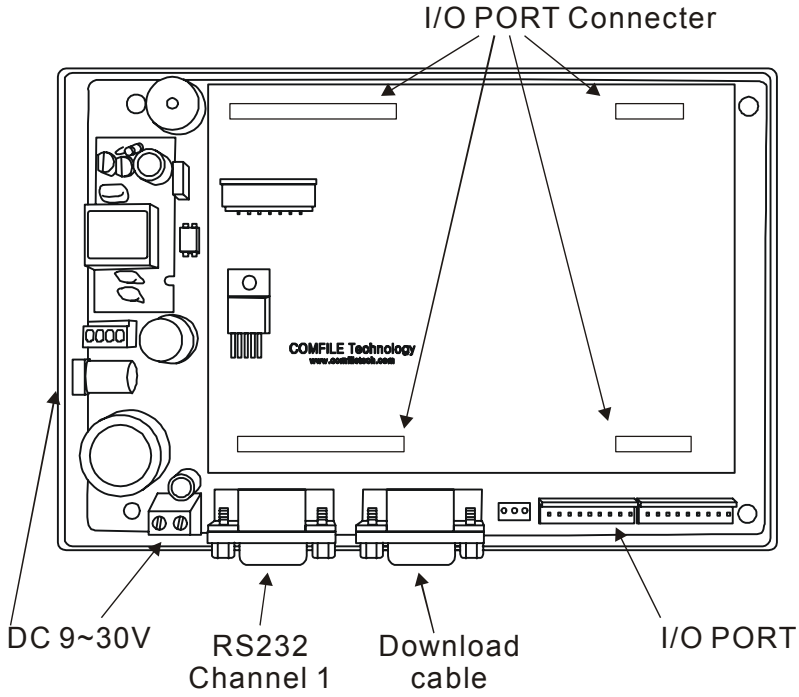
	CT172X/C	기존 터치스크린
콘트롤기능	있음 (LADDER LOGIC 사용)	없음
통신기능	있음 (RS232 포트 내장, 유저가 임의대로 데이터를 주고 받을 수 있음)	있음 (전용 프로토콜 사용)
그래픽 표현방식	BASIC 언어를 사용	전용 그래픽 툴을 사용
터치 입력방식	BASIC 언어를 사용	전용 툴에서 지원
PC 와의 접속	RS232	RS232
가격대	중저가	고가

CT172X/C 의 외형

앞면



뒷면

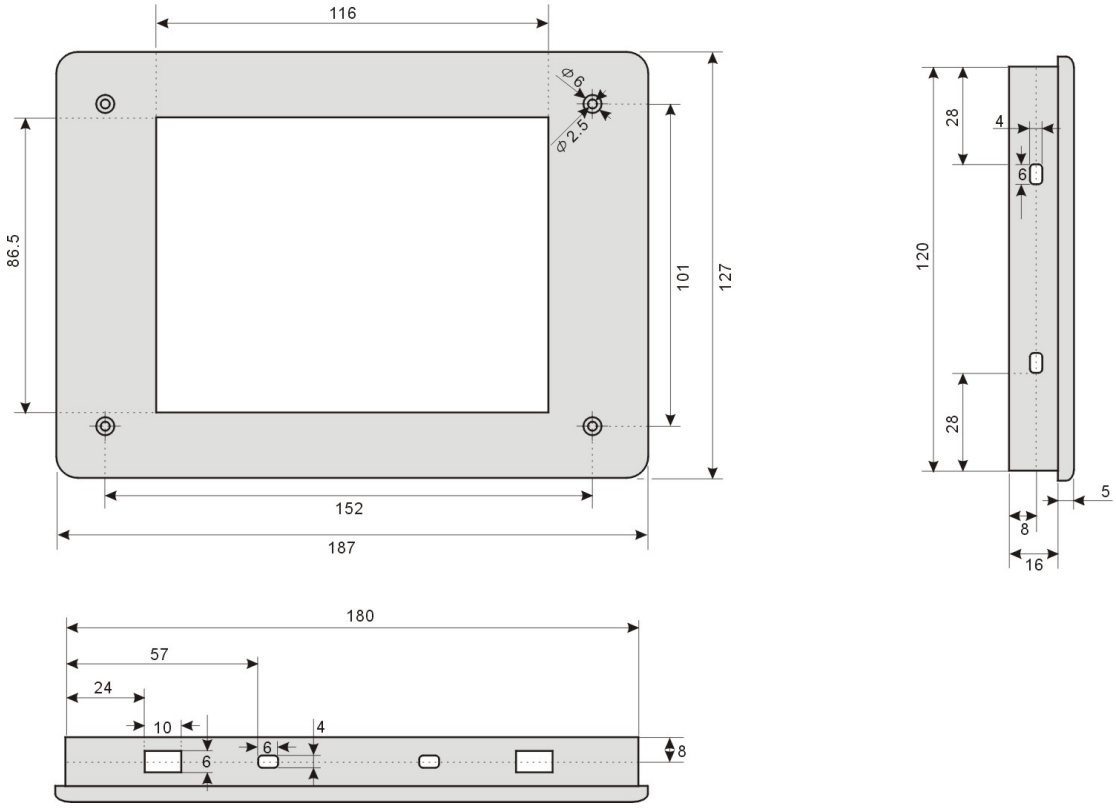


* 주의!!!!!!! 케이스 안에 DANGER 라고 적혀있는 인버터 부근에는 고전압이 흐르므로 감전의 위험이 있습니다. 주의해서 다루시기 바랍니다.

CT172X/C 사진

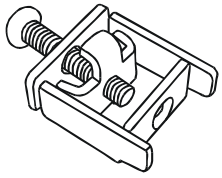
뒷면

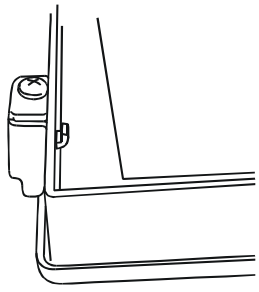
CT172X/C 외형치수



설치용 고정나사 사용법

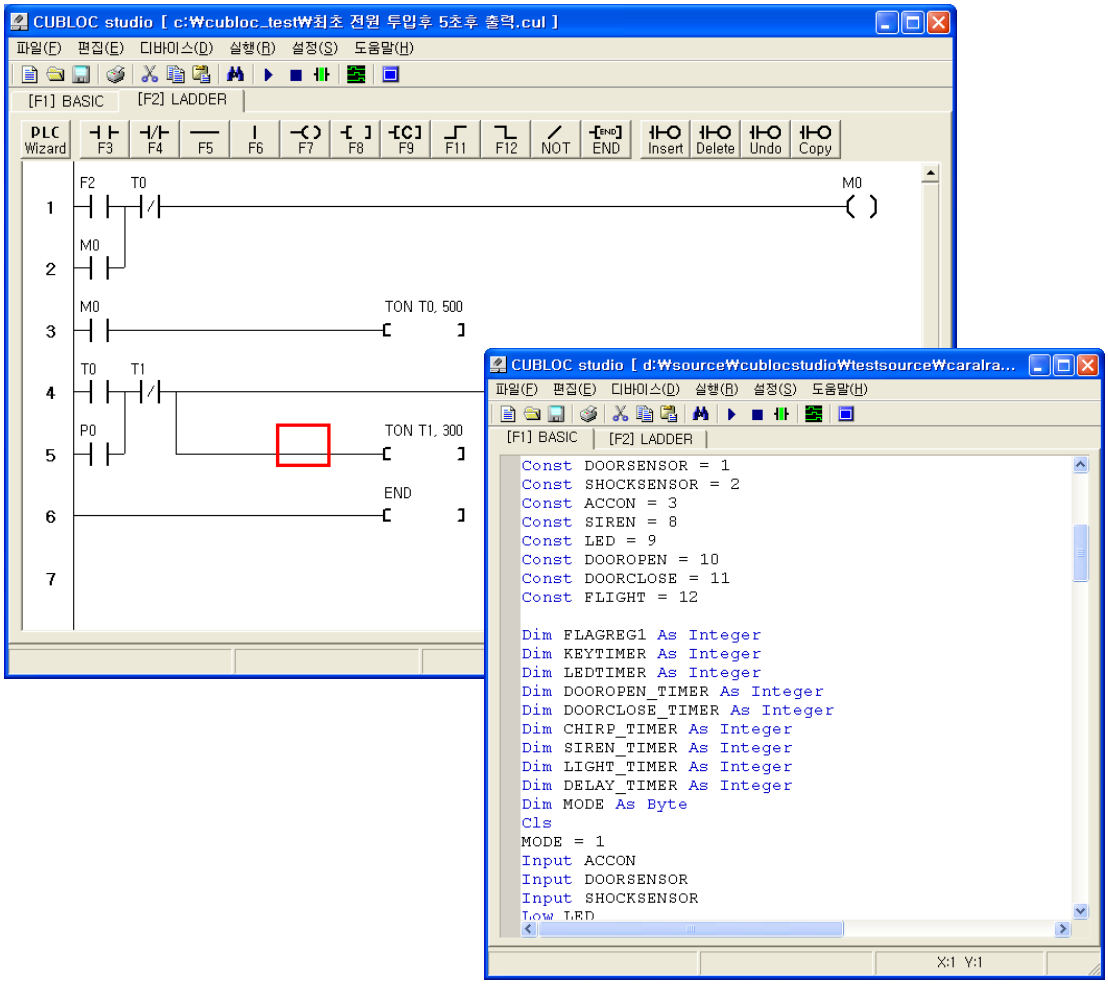
제품구입시, 큐터치를 전면판넬에 설치하기 위한 고정나사를 제공하고 있습니다. 아래 그림과 같이 고정용 장치를 조립하신 후 전면판넬에 장착하시기 바랍니다.





CUBLOC STUDIO

CT172X/C를 사용하는 데 필요한 통합개발환경 소프트웨어인 CUBLOC-STUDIO은 인터넷 홈페이지 www.comfile.co.kr을 통하여 무상 제공됩니다. 유저는 이 프로그램을 사용해서 BASIC과 LADDER프로그램을 작성하게 됩니다.



www.comfile.co.kr에서 최신 버전의 CUBLOC STUDIO 를 다운로드 받으세요. CUBLOC STUDIO는 기능향상을 위하여 예고 없이 업그레이드 될 수 있습니다.

*** CT172X/C 시리즈 사용시에는 CONST DEVICE 문에서 Const Device = CT1720 으로 하시면 됩니다.**

CT172X/C 의 I/O

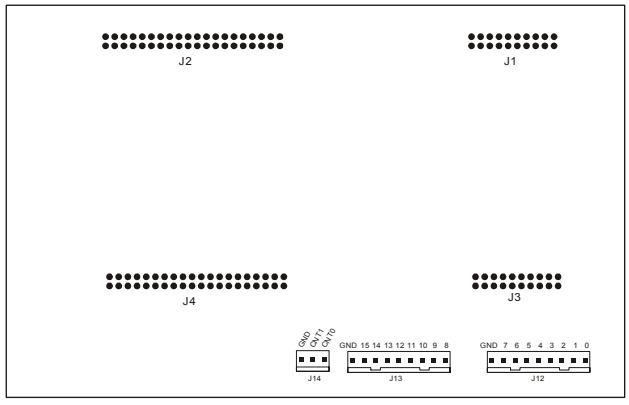
CT172X/C 는 최대 82 개의 I/O 포트를 사용할 수 있도록 되어 있습니다. 사용 가능한 포트 수는 향후, 모델에 따라 다소 차이가 있을 수 있습니다.

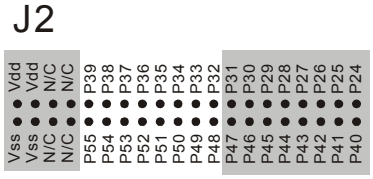
모델명	CT1720	CT1721
입력전용	32	32
출력전용	32	32
A/D 입력겸용	8	8
고속카운터 입력겸용	2	2
기타 I/O	8	8
총합계	82 개	82 개

*CT1720 과 CT1721 의 I/O 수는 동일합니다. 다만 CT1721 에는 DC24V 입력, NPN TR 출력을 위한 ADDON 보드가 기본적으로 장착되어 있습니다.

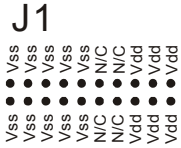
CT1720

CT1720 은 총 82 개의 I/O 포트를 지원합니다.

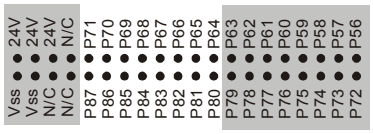




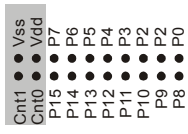
J2



J1



J4



J3

다음은 CT1720 에서 지원하는 I/O 기능을 정리한 표입니다.

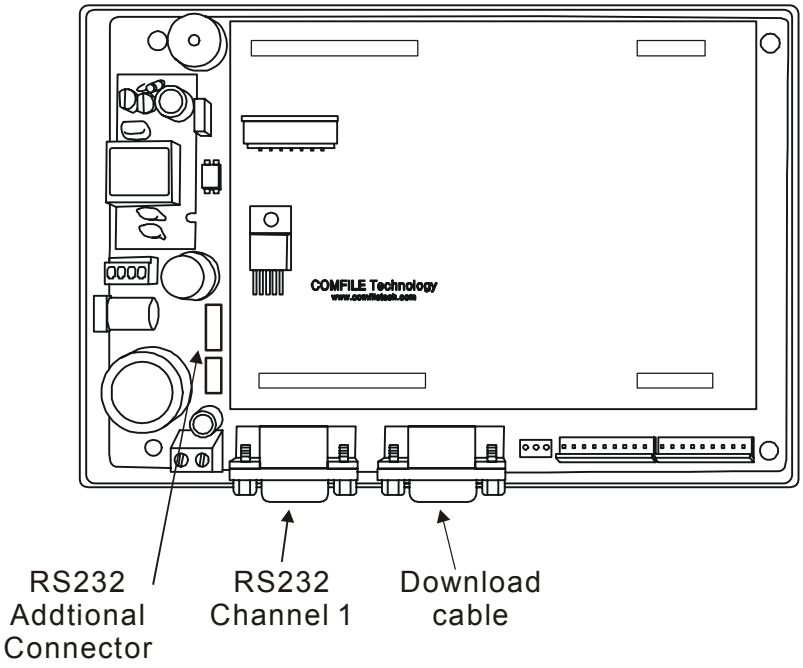
커넥터	이름	I/O	Port Bolck	설명
J12 (J3)	P0	I/O	Block 0	ADC0
	P1	I/O		ADC1
	P2	I/O		ADC2
	P3	I/O		ADC3
	P4	I/O		ADC4
	P5	I/O		ADC5
	P6	I/O		ADC6
	P7	I/O		ADC7
J13 (J3)	P8	I/O	Block 1	PWM0
	P9	I/O		PWM1
	P10	I/O		PWM2
	P11	I/O		PWM3
	P12	I/O		PWM4 / INT0
	P13	I/O		PWM5 / INT1
	P14	I/O		INT2
	P15	I/O		INT3 (RS485 송신허가)
J14	P16	I/O		HIGH COUNT INPUT 0
	P17	IN		HIGH COUNT INPUT 1
	P18	OUTPUT		내부적으로 BUZZER 에 연결되어 있음 (Ladder 측에서는 억세스 할 수 없음)
	P19~P23			사용안함
J2	P24~31	OUTPUT	Block 3	8 개의 Output 전용 포트
	P32~39	OUTPUT	Block 4	8 개의 Output 전용 포트
	P40~47	OUTPUT	Block 5	8 개의 Output 전용 포트
	P48~55	OUTPUT	Block 6	8 개의 Output 전용 포트
J4	P56~63	INPUT	Block 7	8 개의 Input 전용 포트
	P64~71	INPUT	Block 8	8 개의 Input 전용 포트
	P72~79	INPUT	Block 9	8 개의 Input 전용 포트
	P80~87	INPUT	Block 10	8 개의 Input 전용 포트

N/C (No Connection)은 사용하지 않는 핀입니다.

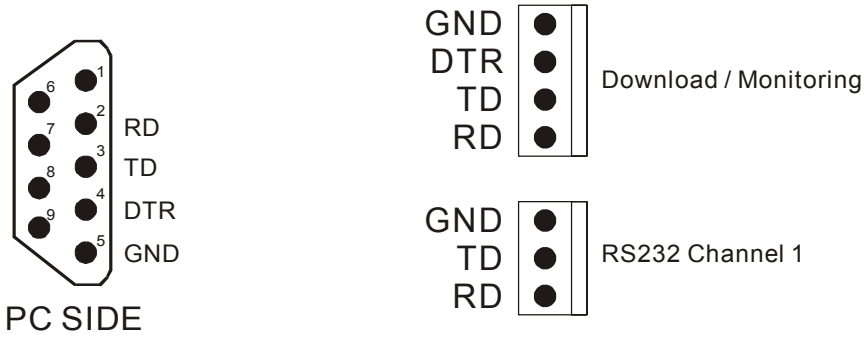
CT1720 의 I/O 포트는 5V 의 소신호를 취급하는 핀이므로 24V 입력 신호 또는 릴레이를 사용하려면 별도의 추가보드 (I/O ADD-ON BOARD) 또는 추가회로를 구성하여 사용해야 합니다.

CT1720 출력전용 포트를 사용하기 위해서는 Set Outonly On 이라는 명령어를 사용해야 합니다. 이 명령을 사용하기 전, 모든 출력전용 포트의 상태는 High-Z (하이 임피던스)상태이며, Set Outonly on 명령 실행 뒤, 출력상태로 바뀌게 됩니다. 따라서 이 명령을 수행하기에 앞서, 해당 포트의 상태를 미리 결정하시는 것이 좋습니다.

```
Byteout 3,0
Set Outonly on
```



Download 용은 4 핀 커넥터로 되어 있고, RS232 채널 1 용은 3 핀 커넥터로 되어 있으며 핀 배치는 다음 그림과 같습니다. 아래 PC 쪽의 RS232 핀 이름을 참조하시어 같은 이름을 가진 핀끼리 연결해 주십시오.



* CT1721C 에서는 RS232 Additional Connector 가 제거되었습니다.

릴레이 표현

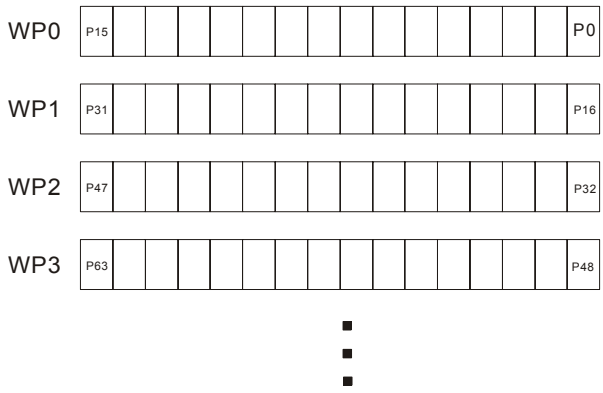
CT172X/C 에서 사용할 수 있는 릴레이를 정리한 표입니다.

릴레이 명칭	범위	단위	기능
입출력 릴레이 P	P0~P127	1 비트	외부 기기와의 인터페이스
내부릴레이 M	M0~M1023	1 비트	내부 상태의 보존
특수기능 릴레이 F	F0~F127	1 비트	시스템 상태
타이머 T	T0~T255	16 비트 (1 워드)	타이머용
카운터 C	C0~C255	16 비트 (1 워드)	카운터용
스텝제어 S	S0~S15	256 스탭(1 바이트)	스텝제어용
데이터 영역 D	D0~511	16 비트 (1 워드)	데이터보관

P,M,F 는 비트단위로 되어 있고, T,C,D 는 워드단위로 되어 있습니다. 비트 영역으로 되어 있는 P, M, F 를 워드단위로 억세스 하려면 WP, WM, WF 을 사용해야 합니다.

릴레이 명칭	범위	단위	기능
WP	WP0~7	16 비트 (1 워드)	P 영역의 워드단위 억세스
WM	WM0~WM63	16 비트 (1 워드)	M 영역의 워드단위 억세스
WF	WF0~WF7	16 비트 (1 워드)	F 영역의 워드단위 억세스

WP0 는 P0~P15 까지의 내용을 담고 있으며, P0 이 가장 아래쪽(LSB)에 P15 가 가장 위쪽(MSB)에 위치합니다. WMOV 등과 같은 응용명령어 군에서 사용하면 편리합니다.



밧데리 백업

CT172X/C 의 데이터 메모리는 밧데리 백업이 되어 있어, 전원이 없어도 내용이 유지됩니다. 만약 백업이 필요 없다면, 프로그램 초기화 부분에서 클리어 해주어야 합니다. BASIC 의 경우 RAMCLEAR 명령을 사용해서 전체 데이터 메모리를 일괄적으로 클리어 할 수 있으며, 각각의 변수를 개별적으로 클리어 할 수도 있습니다.

```
Const Device = CT1720
Dim TX1 As Integer, TY1 As Integer
TX1 = 0
TY1 = 0    ' 개별 클리어
RAMCLEAR   ' BASIC 의 모든 RAM 영역을 클리어
```

LADDER 의 경우 P 영역을 제외한 나머지 S, M, C, T, D 영역만 밧데리에 의한 백업이 됩니다. P 영역은 전원 ON 시 0 으로 클리어 됩니다. 백업을 원하지 않는 일부 구간이 있다면, 다음과 같이 프로그램으로 전원 On 시에 밧데리 백업상태를 클리어 합니다.

```
Const Device = CT1720
Dim I As Integer
For I=0 to 32          ' 백업을 원치 않는 일부 구간만 클리어
      M(I) = 0
Next
Set Ladder On
```

대부분의 PLC 는 KEEP 메모리를 지원하고 있습니다. KEEP 메모리는 순간 정전 시 그 값을 보존해 주는 기능을 하며, 시퀀스 제어의 필수적 요소입니다. CT172X/C 에서도 KEEP 기능을 지원하기 위해 밧데리를 내장하고 있습니다. 기본적으로 전체 릴레이영역에서 KEEP 기능을 사용할 수 있으며, KEEP 을 원하지 않는 일부 구간은 강제적으로 초기화 시켜주는 명령을 추가합니다.

*** CT172X/C 에 내장된 슈퍼콘덴서는 순간정전 시 데이터를 보전하는 용도로 밖에 사용할 수 없습니다. 좀더 오랜 시간 데이터를 보존하고 싶다면, 별도의 밧데리를 추가로 연결하여 주시기 바랍니다.**

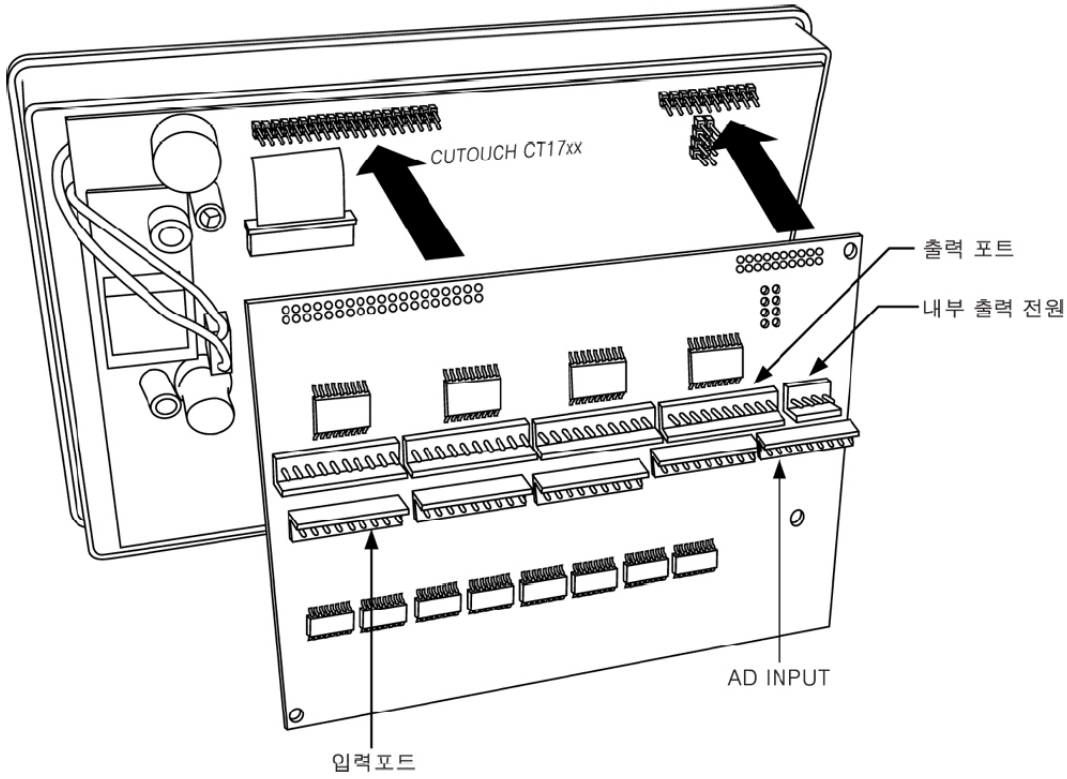
*** CT1721C 에서는 데이터 메모리 밧데리 백업기능이 제외되었습니다. 데이터를 정전 상태에서 유지하고 싶다면, EEPROM 을 사용하거나, 별도의 밧데리를 연결하여 주십시오.**

타이머와 카운터의 KEEP 사용

KEEP 타이머는 전원 OFF 전 상황에서 멈추고 있다가, 전원이 ON 되면 이전 값부터 계속 타이머가 진행됩니다. 타이머의 경우 CTU, CTD 명령대신 KCTU, KCTD 명령을 사용해야 KEEP 기능을 사용할 수 있습니다. 자세한 내용은 “LADDER LOGIC 사용설명서”중에서 KCTU, KCTD 명령 해설 부분을 참조하여 주시기 바랍니다.

ADD-ON BOARD

애드온 보드는 CT1720 에 추가하여 사용할 수 있는 보드입니다. DC24V 입력을 받을 수 있는 포토커플러와 NPN TR 출력 포트가 내장되어 있어, 산업현장에 있는 센서나 릴레이등을 직접 연결할 수 있습니다.



ADD-ON보드에 대한 자세한 규격/핀아웃은 별도의 설명서 (www.comfile.co.kr에서 다운로드 가능)를 참조하시기 바랍니다.

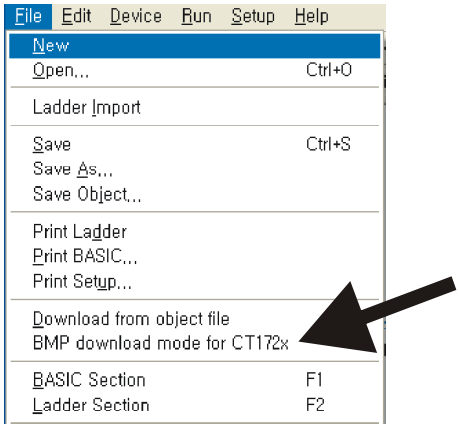
*** CT1721 모델은 CT1720 에 단순히 ADDON BOARD 가 추가된 모델입니다. 따라서 ADDON 보드가 필요하신 분들은 처음부터 CT1721 을 선택하시면, 별도로 장착해야하는 번거로움을 줄일 수 있습니다.**

BMP 다운로드

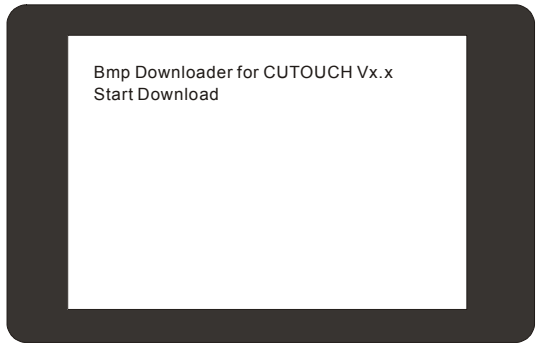
BMP 다운로드 기능을 활용하면, CT172X/C 에 BMP 이미지를 저장해 두었다가 화면에 표시할 수 있습니다. CT1721C 모델부터는 이 기능을 사용할 수 있습니다. 기존에 CT172X/C 를 구매하셨던 고객분들도 펌웨어를 업그레이드하신다면 이 기능을 사용할 수 있습니다.

다만, CT172X/C 의 그래픽칩 펌웨어를 업그레이드 해야하기 때문에, 제품을 직접 가지고 본사방문 (또는 택배)하셔야 업그레이드가 가능합니다. 자세한 업그레이드방법 문의는 본사 02-711-2592 로 문의바랍니다.

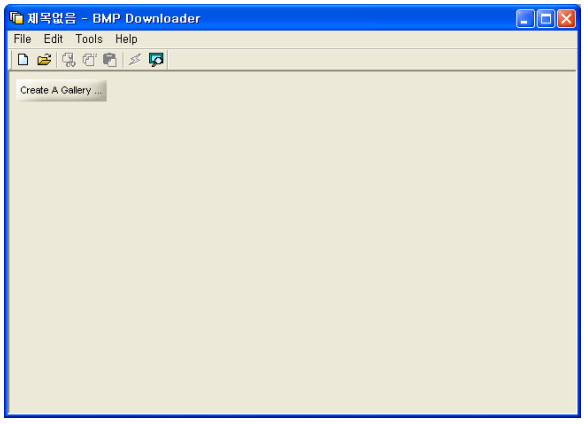
CUBLOC STUDIO 2.5.A 이후버젼에서 FILE 메뉴를 보시면 BMP download mode for CT172X/C 라는 메뉴를 보실 수 있습니다.



CT172X/C 가 PC 와 연결된 상태에서 이 메뉴를 선택하시면, CT172X/C 화면에 "BMP Downloader for CT172X/C"라는 멧세지가 표시됩니다.



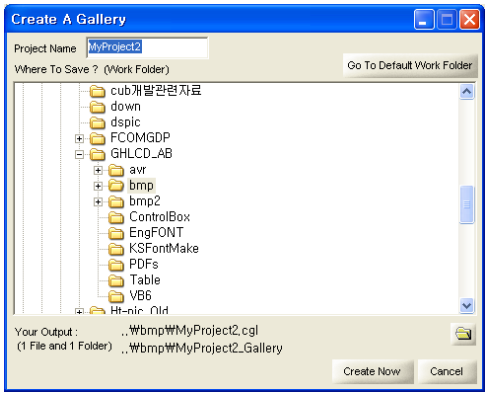
이때 별도의 프로그램인 BMP downloader를 실행시키십시오. BMP download는 www.comfile.co.kr에서 구하실 수 있습니다.



처음 사용하시는 거라면, File 메뉴에 있는 New Gallery 를 선택하세요.

Gallery 라는 개념은 일종의 파일의 집합을 말합니다.

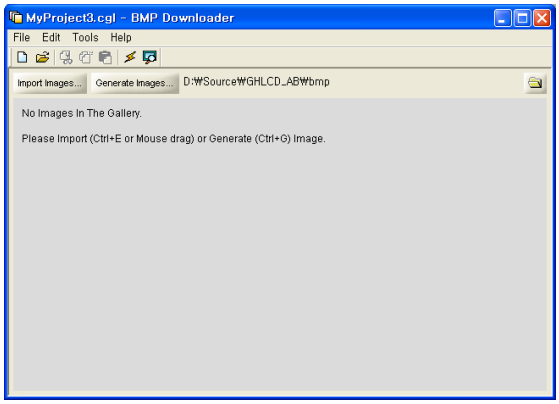
Gallery 가 없는상태에서는 일단 New Gallery 를 이용해서 한번만 생성해주시면 이후에 계속 사용가능합니다.



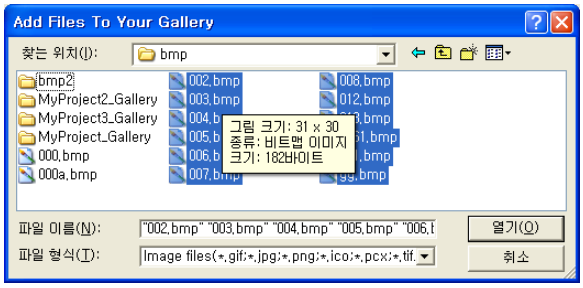
이 화면에서 BMP 파일이 존재하는 폴더를 선택하시면 됩니다.

CT172X/C 에 저장가능 한 BMP 파일은 반드시 “흑백 비트맵모드”로 저장된 것이여야 합니다.

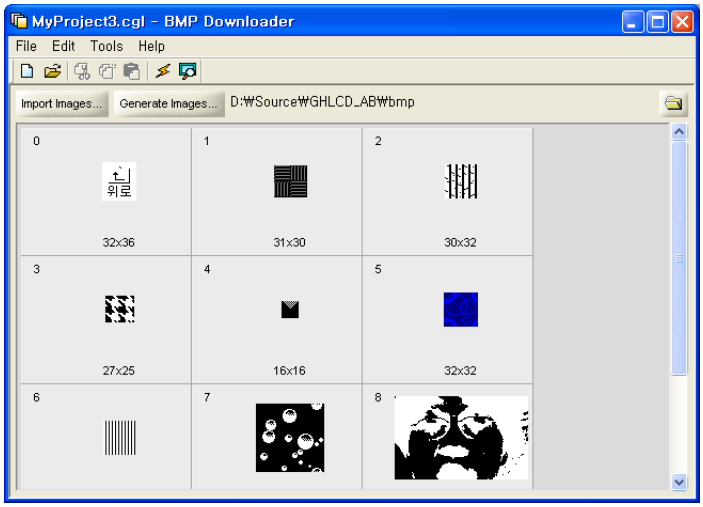
사전에 미리 다운로드할 BMP 파일을 하나에 폴더에 모아놓은뒤 BMP Downloader 를 실행하는 것이 좋습니다.



Gallery 가 생성되면 그 다음에는 화면상단에 있는 Import Image 를 선택하십시오.



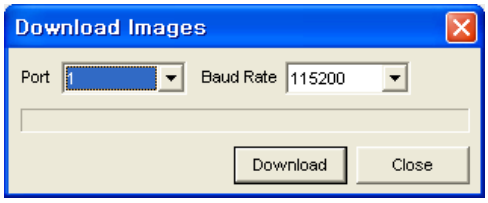
Bmp 들이 저장된 폴더로 가서 Gallery 에 등록할 파일을 선택한뒤 “열기”를 누르면 등록이 완료됩니다.

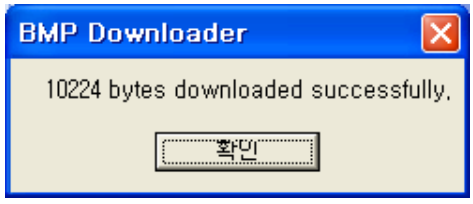


다음과 같은 상태가 되면 이제 다운로드를 시도합니다.

화면상단에 있는 아이콘중 번개모양을 클릭 하시거나, Tools 메뉴에 있는 Download Image 를 선택하십시오.

연결된 Comport 를 선택하면됩니다. 보레이트는 115200 으로 하십시오. Download 보턴을 누르면 다운로드가 시작됩니다.





다운로드가 끝나면, CT172X/C 화면상에서 다운로드된 BMP 이미지를 볼 수 있습니다.

이후에 똑 같은 BMP 파일들을 다른 CT172X/C 에 다운로드하고 싶다면, 앞서 저장한 Gallery 를 오픈하면됩니다. Gallery 는 BMP 파일의 위치를 저장하고 있는 “프로젝트 파일”입니다.

BMP 이미지 사용법

BMP 파일을 사용하면 LINE, CIRCLE 등의 명령어로 구현할 수 없었던, 다양한 그림들을 표현할 수 있습니다. PC 상에서 별도의 그래픽 툴 (포토샵등)을 이용하여 그림을 그린 다음 반드시 "흑백 비트맵 모드"로 저장하십시오.

그리고 사이즈는 최대 320 X 240 으로 해야됩니다. 그보다 작은 사이즈도 가능합니다.

CT172X/C 에는 최대 104.448 바이트의 BMP 파일 저장공간이 있습니다. 이것은 320 X 240 사이즈 (약 9600 바이트 소요됨)로 10 개정도 저장할 수 있는 크기입니다.

다운로드된 BMP 이미지는 CT172X/C 상의 별도의 FLASH 공간에 저장됩니다. 프로그램 메모리나 데이터메모리 (EEPROM 등) 을 사용하지 않으므로, 기존 메모리에서 손해보는 부분은 없습니다.

이미 저장된 BMP 이미지를 화면상으로 불러내기 위해서는 BMP 명령을 사용합니다.

BMP

BMP x, y, imageNO, layer

```
X : x축 좌표 (0 to 320)
y : y축 좌표 (0 to 240)
imageNO : BMP 파일의 인덱스 번호
layer : 표시할 레이어
```

X,Y 의 위치에 BMP 파일번호 imageNO 를 표시합니다. Layer 는 표시할 레이어를 뜻합니다. 보통은 레이어 2 (그래픽면)에 표시합니다.
ImageNO 는 BMP Downloader 에서 BMP 파일을 배치한 순서대로 부여됩니다. 따라서 파일명을 000.bmp, 001.bmp 와 같은 식으로 작명한다면, 좀더 쉽게 사용가능합니다. 왜냐하면 파일명의 이름으로 정렬되기 때문에, 실제 파일명과 ImageNO 가 일치합니다.

다음은 CT172X/C 에서 저장된 BMP 이미지를 차례대로 볼 수 있는 소스프로그램입니다.

```
Const Device = CT1720
Dim I As Integer, J As Integer
Dim TX1 As Integer, TY1 As Integer
Dim CN As Integer
CN=Eeread(0,2)
If CN < 400 Or CN > 600 Then
    CN = 540
Endif
Contrast CN
'
'   BMP BROWSER
'
Set Pad 0,4,5
```

```
On Pad Gosub GetPUSHSCREEN
I=0
Gosub DRAWSCREEN
Do
Loop

GetPUSHSCREEN:
TX1 = Getpad(2)
TY1 = Getpad(2)
If Menucheck(0,TX1,TY1) = 1 Then
    Menureverse 0
    Wait 100
    Menureverse 0
    Decr I
    Gosub DRAWSCREEN
Endif
If Menucheck(1,TX1,TY1) = 1 Then
    Menureverse 1
    Wait 100
    Menureverse 1
    Incr I
    Gosub DRAWSCREEN
Endif

If Menucheck(2,TX1,TY1) = 1 Then
    Menureverse 2
    High 18
    Wait 20
    Low 18
    CN = CN - 5
    Contrast CN
    Eewrite 0,CN,2
    Wait 100
    Menureverse 2
Endif
If Menucheck(3,TX1,TY1) = 1 Then
    Menureverse 3
    High 18
    Wait 20
    Low 18
    CN = CN + 5
    Contrast CN
    Eewrite 0,CN,2
    Wait 100
    Menureverse 3
Endif

Return
```

```
DRAWSCREEN:
High 18
Wait 20
Low 18
Cls
Menuset 0,2,8,16,87,44
Menutitle 0,23,5,"DECR"
Menuset 1,2,96,16,176,44
Menutitle 1,23,5,"INCR"
Menuset 2,2,200,16,240,44
Menutitle 2,12,5,"<<"
Menuset 3,2,250,16,290,44
Menutitle 3,12,5,">>"
Print "BMP IMAGE # : ",Dec I
Locate 26,0
Print "CONTRAST"
Bmp 0,48,I,2
Return
```

MEMO

제 2 장
메뉴 시스템 라이브러리

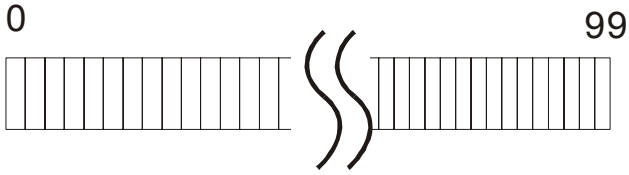
MEMU 관련 명령어는 CT172X/C 시리즈에서만 사용가능합니다. CT2400 이후 시리즈에서는 사용할 수 없습니다.

메뉴 시스템 라이브러리

CT172X/C 화면상에 메뉴를 표시하고 터치입력을 받기 위한 명령어 모음입니다. 본 라이브러리를 사용하면 아래와 같은 화면을 쉽게 구성하고 운영할 수 있게 됩니다.

MENU 관련 명령군

CT172X/C 에는 총 100 개의 MENU 보턴을 기억할 수 있는 방이 있습니다. 각각의 방에 MENUSET 명령으로 MENU 의 영역과 스타일을 정의해 놓습니다. 그리고 MENUTITLE 명령으로 MENU 의 제목을 표시하고, 터치입력이 들어오면, MENUCHECK 명령으로 눌려진 MENU 를 찾아내는 방법으로 프로그램을 작성합니다.



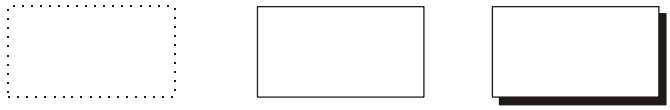
각 보턴의 상황을 기록해 놓는 100 개의 방은 MENUSET 명령에 의해서 언제든지 다시 셋팅할 수 있습니다. 즉, 한 화면에서 최대 100 개의 보턴을 입력 받을 수 있고, 다음 화면에서 보턴의 상태를 다시 셋팅하여 사용할 수 있습니다.

Menuset

MENUSET index, style, x1, y1, x2, y2

```
Index : 메뉴인덱스 번호
Style : 보턴의 모양; 0=표시없음, 1=박스, 2=그림자박스
X1,y1,x2,y2 : 메뉴 보턴의 위치정보
```

Index 에는 0~99 사이의 값을 적어줍니다. Style 은 보턴의 모양입니다. 0 으로 하면 아무것도 표시되지 않습니다. 1 은 보통박스의 모양이며, 2 는 그림자가 있는 박스의 모양입니다.



x1,y1, x2, y2 를 대각선으로 하는 박스가 보턴의 크기가 됩니다. 이 명령이 실행되면 화면의 일부 영역이 보턴영역으로 설정되고, 스타일의 종류에 따라 보턴의 모양이 표시됩니다.

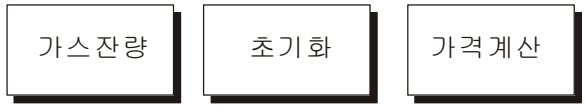
Menutitle

MENUTITLE index, x, y, string

```
Index : 메뉴인덱스 번호
X,y : 메뉴버턴의 왼쪽 위 기준점에서 x,y 위치
string : 메뉴의 제목
```

Menuset 명령으로는 박스만 표시되기 때문에 어떤 메뉴인지 알 수 없습니다. Menutitle 명령으로 Menuset 명령으로 선언한 메뉴영역에 제목을 표시합니다.

```
Menutitle 0,13,13,"가스잔량"
Menutitle 1,16,13,"초기화"
Menutitle 2,13,13,"가격계산"
```



Menucheck( )

Variable = MENUCHECK(index, touchx, touchy)

```
Variable : 결과를 저장할 정수형 변수 (선택되었으면 1, 아니면 0)
Index : 메뉴인덱스 번호
Touchx : 터치입력지점의 x 좌표
Touchy : 터치입력지점의 y 좌표
```

어떤 메뉴가 선택되었는지 알아낼 수 있는 명령어입니다. Touchx, Touchy 에는 유저가 터치패드에 입력한 지점의 좌표를 넣고, Index 에는 Menuset 에서 선언한 범위내의 값을 적어줍니다.

Index 에 지정한 메뉴가 선택되었다면 1 을 반환합니다. 선택되지 않았으면 0 을 반환합니다.

```
If Menucheck(0,TX1,TY1) = 1 Then
       Menureverse 0
       Beep 18,180
End If
```

Menu( )

Variable = MENU(index, pos)

```
Variable : 결과를 저장할 정수형 변수 (선택되었으면 1, 아니면 0)
Index : 메뉴인덱스 번호
pos : 위치 (0=x1, 1=y1, 2=x2, 3=y2)
```

Menuset 명령에 의해 셋팅된 값을 다시 읽어낼 필요가 있을 때, 사용하는 함수입니다. 해당인덱스에서 pos 값에 따라 읽어옵니다. 0 이면 x1, 1 이면 y1, 2 이면 x2, 3 이면 y2 의 좌표 값을 읽어옵니다. 마치 CT172X/C 내부에 있는 MENU 영역을 2 차원배열로 억세스 하는 것과 비슷합니다.

```
If Menu(0,1) < 100 THEN ' 메뉴 보턴 0 번의 Y1 위치가 100 보다 작으면
```

Menureverse

MENUREVERSE index

```
Index : 메뉴인덱스 번호
```

선택된 메뉴 박스를 반전시켜 주는 명령입니다. 어떤 메뉴가 선택되었는지 표현하기 위해 사용합니다.

터치 패드 입력방법

CT172X/C 에 있는 터치패드로부터 유저 입력이 있었을 경우, 어떤 지점을 눌렀는지 알아내는 명령어에 대하여 설명하겠습니다. 이를 위하여 SET PAD, ON PAD, GETPAD 명령 등을 사용합니다.
PAD 명령군은 외부로부터 어떤 입력을 받기 위한 CUBLOC BASIC 의 기본 기능입니다. 주로 발생시점을 예측할 수 없는 신호, 예를 들면 키보드입력 등에서 사용합니다.

터치패드입력도 발생시점을 예측할 수 없기 때문에 ON PAD 인터럽트를 사용해서 입력을 받아들입니다. 다음은 터치패드를 사용하고 있는 기본적인 샘플 프로그램입니다.

```
    '
    '   DEMO FOR GHTB
    '
    Const Device = CT1720
    Dim TX1 As Integer, TY1 As Integer
    Set Pad 0,4,5               '← (1) Touch PAD 입력 활성화
    On Pad Gosub abc            '← (2) 인터럽트 선언
    Do
    Loop
abc:
    TX1 = Getpad(2)             '← (3) 인터럽트 서비스 루틴
    TY1 = Getpad(2)
    Circlefill TX1,TY1,10       '← (4) 터치 지점에 원표시
    Return
```

(1) SET PAD 0, 4, 5 : 명령에 의해 PAD 명령 입력이 활성화 됩니다. (명령형식: SET PAD mode, packet size, buffer size). CT172X/C 에는 터치패드입력을 감지하여 SPI 신호를 발생시켜주는 별도의 "터치콘트롤러"가 내장되어 있습니다. 이 "터치 콘트롤러"에서 발생시키는 신호는 mode =0 에 해당합니다. (MSB 우선, clk 상승에지에서 샘플링) 입력패킷은 4 바이트 (X, Y 가 각각 2 바이트씩)로 되어 있습니다. 버퍼사이즈는 4 보다 하나 큰 5 바이트로 설정하였습니다.

(2) ON Pad Gosub ABC : 이 명령은 PAD 인터럽트 선언문입니다. PAD 입력이 발생하면 ABC 라는 라벨로 점프합니다.

(3) 인터럽트 서비스 루틴입니다. PAD 입력이 발생되면 이곳을 실행하게 됩니다. Getpad 명령으로 버퍼에 수신된 데이터를 읽어옵니다. 첫 2 바이트는 x 축 좌표, 뒤에 2 바이트는 y 축 좌표입니다.

(4) 해당위치에 원을 표시합니다.

이 프로그램을 실행시키면 터치입력지점에 원을 표시하는 동작을 하게 됩니다. 이 샘플프로그램을 기본골격으로 하여 CT172X/C 응용프로그램을 작성하시기 바랍니다.
*** 터치 입력 시 동시에 두 지점을 누르지 마십시오. 엉뚱한 지점을 누른 것처럼 판단될 수 있습니다.**

다음은 MENU 명령과 ON PAD 명령 활용법을 쉽게 알 수 있도록 만든 예제 프로그램입니다. 보턴을 누르면 삑 소리가 나면서 보턴이 반전됩니다.

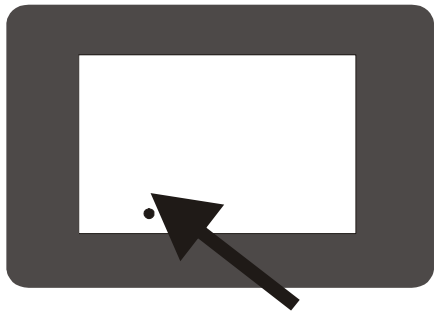
```
    '
    '   DEMO FOR GHTB
    '
    Const Device = CT1720
    Dim TX1 As Integer, TY1 As Integer
    Dim k As Long
    Contrast 550          ' 이 수치를 변경하면 화면의 Constast 가 조정됨.
    Set Pad 0,4,5
    On Pad Gosub abc
    Menuset 0,2,8,16,87,63
    Menutitle 0,13,13,"시  작"
    Menuset 1,2,96,16,176,63
    Menutitle 1,13,13,"종  료"
    Menuset 2,2,184,16,264,63
    Menutitle 2,13,13,"재시작"
    Low 18
    Do
    Loop
abc:
    TX1 = Getpad(2)
    TY1 = Getpad(2)
    Circlefill TX1,TY1,10
    If Menucheck(0,TX1,TY1) = 1 Then
         Menureverse 0
         Pulsout 18,300   '키터치음 발생, 부저가 연결되어 있음
    End If
    If Menucheck(1,TX1,TY1) = 1 Then
         Menureverse 1
         Pulsout 18,300
    End If
    If Menucheck(2,TX1,TY1) = 1 Then
         Menureverse 2
         Pulsout 18,300
    End If
    Return
```

터치 패드 입력보정

터치지점과 실제좌표가 안맞는 경우가 간혹 발생합니다. 이런 경우에는 다음과 같은 방법으로 터치지점을 조정할 수 있습니다.

```
Const Device = CT1720
Dim TX1 As Integer, TY1 As Integer
Contrast 500
Set Pad 0,4,5
On Pad Gosub GETTOUCH
Do
Loop

GETTOUCH:
TX1 = Getpad(2)
TY1 = Getpad(2) * 1.075      ' ←----- 이 부분 주목하세요!
Circlefill TX1,TY1,5
Return
```

이 프로그램을 실행시키면 터치한 지점에 작은 원을 표시합니다. 위 소스중 TY1 좌표를 받는 부분에 보정값을 곱해준 것을 확인할 수 있습니다. 터치한 지점을 읽은 값이 실제좌표와 차이가 나는 것을 이 와 같은 방법으로 보정하실 수 있습니다.



터치한 지점은 이곳인데, 점이 아랫부분에 찍힌 경우
TY1 = GETPAD(2) 결과에 1.075 를 곱해서
보정합니다.

MEMO

제 3 장
CT172X/C
샘플 프로그램

SAMPLE 1

화면상에 증가되는 숫자를 표시해보도록 하겠습니다. 이 곳에서 설명한 소스 파일은 CUBLOC STUDIO 설치 시 함께 설치됩니다.

<파일명 : CT001.CUL>

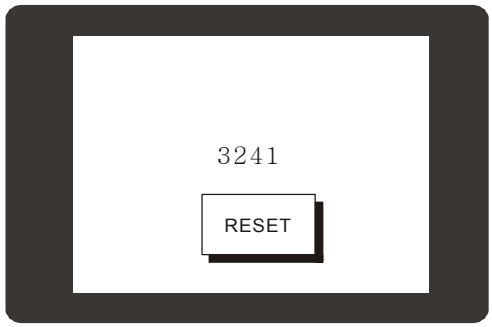
```
Const Device = Ct1720
Dim I As Integer
Contrast 550     ' LCD CONTRAST SETTING

Do
      Locate 15,6
      Print DEC5 I
      Incr I
      Delay 200
Loop
```

CONTRAST 뒤에 있는 숫자를 조정해서, 화면의 CONSTRAST 를 조정합니다. 이 부분을 적당한 값으로 조정해야 화면상에 숫자가 잘 보일 것입니다.

*모델에 따라서 CT172X/C 뒷면에 볼륨으로도 CONTRAST 를 조정할 수 있습니다. 이 경우, 뒷면볼륨과 CONTRAST 명령으로 함께 화면 밝기를 조정할 수 있습니다.

SAMPLE 2

다음은 화면상에 RESET 보턴을 그리고, 이 보턴이 터치되었을 때 증가되는 값을 0 으로 리셋하는 샘플 프로그램 입니다.



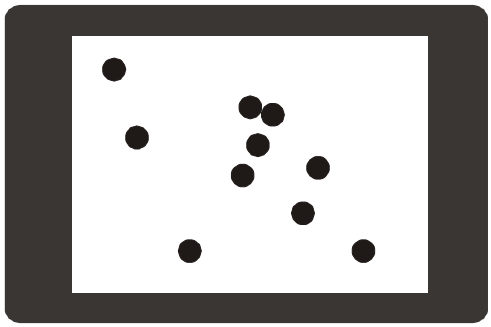
<파일명 : CT002.CUL>

```
    Const Device = Ct1720
    Dim I As Integer
    Dim TX1 As Integer, TY1 As Integer
    Contrast 550
    Set Pad 0,4,5
    On Pad Gosub GETTOUCH
    Menuset 0,2,120,155,195,200
    Menutitle 0,20,14,"RESET"

    Do
          Locate 15,6
          Print DEC5 I
          Incr I
          Delay 200
    Loop

GETTOUCH:
    TX1 = Getpad(2)
    TY1 = Getpad(2)
    If Menucheck(0,TX1,TY1) = 1 Then
          Pulsout 18,300
          I = 0
    End If
    Return
```

SET PAD 명령에 의해 터치입력이 활성화되고, ON PAD 명령에 의해 터치입력 시 점프할 라벨을 선언합니다. MENUSET 명령은 화면에 터치영역을 선언하는 명령이고, MENUTITLE 은 선언된 터치영역에 이름을 표시합니다.PULSEOUT 은 BEEP 사운드를 발생시키는 명령입니다.

SAMPLE 3

터치한 지점에 원을 표시하는 프로그램입니다.

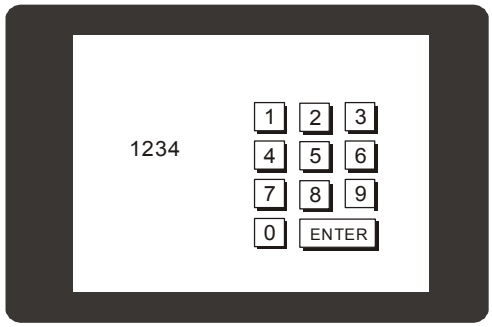


<파일명 : CT003.CUL>

```
    Const Device = Ct1720
    Dim TX1 As Integer, TY1 As Integer
    Contrast 550
    Set Pad 0,4,5
    On Pad Gosub GETTOUCH
    Do
    Loop

GETTOUCH:
    TX1 = Getpad(2)
    TY1 = Getpad(2)
    Circlefill TX1,TY1,10
    Pulsout 18,300
    Return
```

SAMPLE 4

숫자 키패드 모양의 화면을 표시하고, 입력한 값을 화면상에 표시하는 프로그램입니다.



<파일명 : CT004.CUL>

```
Const Device = Ct1720
Dim TX1 As Integer, TY1 As Integer
Dim I As Integer
Contrast 550
Set Pad 0,4,5
On Pad Gosub GETTOUCH
Menuset 0,2,165,50,195,75
Menutitle 0,11,4,"1"
Menuset 1,2,205,50,235,75
Menutitle 1,11,4,"2"
Menuset 2,2,245,50,275,75
Menutitle 2,11,4,"3"
Menuset 3,2,165,85,195,110
Menutitle 3,11,4,"4"
Menuset 4,2,205,85,235,110
Menutitle 4,11,4,"5"
Menuset 5,2,245,85,275,110
Menutitle 5,11,4,"6"
Menuset 6,2,165,120,195,145
Menutitle 6,11,4,"7"
Menuset 7,2,205,120,235,145
Menutitle 7,11,4,"8"
Menuset 8,2,245,120,275,145
Menutitle 8,11,4,"9"
Menuset 9,2,165,155,195,180
Menutitle 9,11,4,"0"
Menuset 10,2,205,155,275,180
Menutitle 10,17,4,"ENTER"

I =0
Do
Loop
```

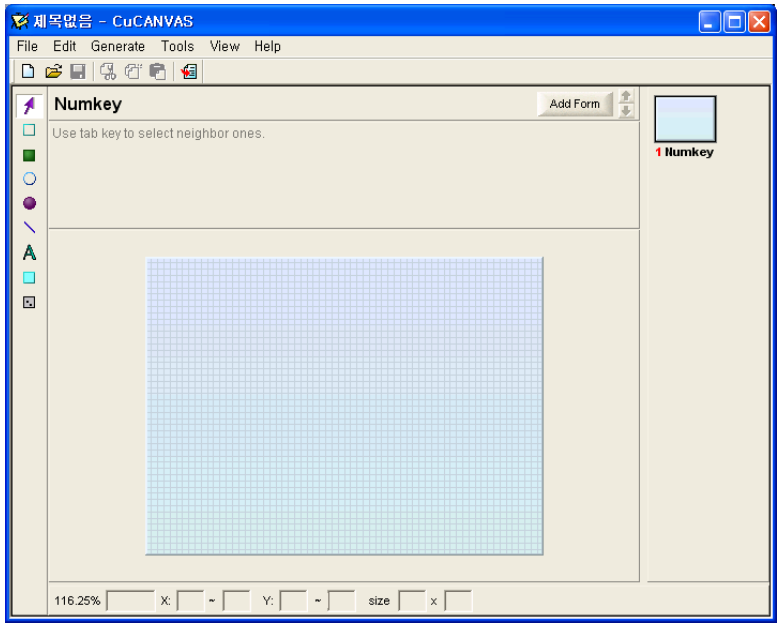
```
GETTOUCH:
    TX1 = Getpad(2)
    TY1 = Getpad(2)
    If Menucheck(0,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 1
          Pulsout 18,300
    Elseif Menucheck(1,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 2
          Pulsout 18,300
    Elseif Menucheck(2,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 3
          Pulsout 18,300
    Elseif Menucheck(3,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 4
          Pulsout 18,300
    Elseif Menucheck(4,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 5
          Pulsout 18,300
    Elseif Menucheck(5,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 6
          Pulsout 18,300
    Elseif Menucheck(6,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 7
          Pulsout 18,300
    Elseif Menucheck(7,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 8
          Pulsout 18,300
    Elseif Menucheck(8,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 9
          Pulsout 18,300
    Elseif Menucheck(9,TX1,TY1) = 1 Then
          I = I << 4
          Pulsout 18,300
    Elseif Menucheck(10,TX1,TY1) = 1 Then
          I = 0
          Pulsout 18,300
    End If
    Locate 3,3
    Print HEX4 I
    Return
```

최종결과인 I는 BCD 코드 형태로 값이 저장되어 있으므로, BCD2BIN 명령을 이용해서 일반수치로 바꾸어서 사용해야 합니다.

SAMPLE 5

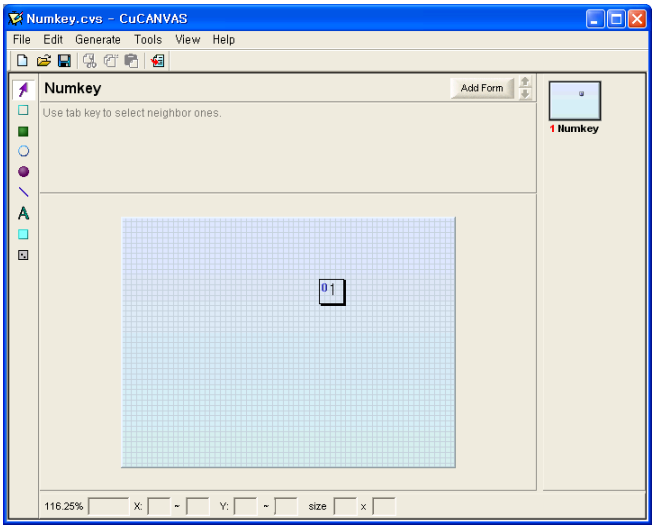
앞에서와 같은 화면을 일일이 좌표를 계산해서 입력해주어야 하는 것은 매우 힘든 일입니다. 이런 화면을 구성하기 위해서 CUCANVAS를 사용하면, 보다 편리하게 작업할 수 있습니다. (CUCANVAS는 FreeWare로 www.comfile.co.kr에서 무료로 다운로드 받으실 수 있습니다.)

CUCANVAS 를 실행시키고, 화면 오른쪽 위에 있는 ADD FORM 을 눌러서 화면이름 (여기서는 NUMKEY)을 입력하면 다음과 같은 빈 화면이 표시됩니다.

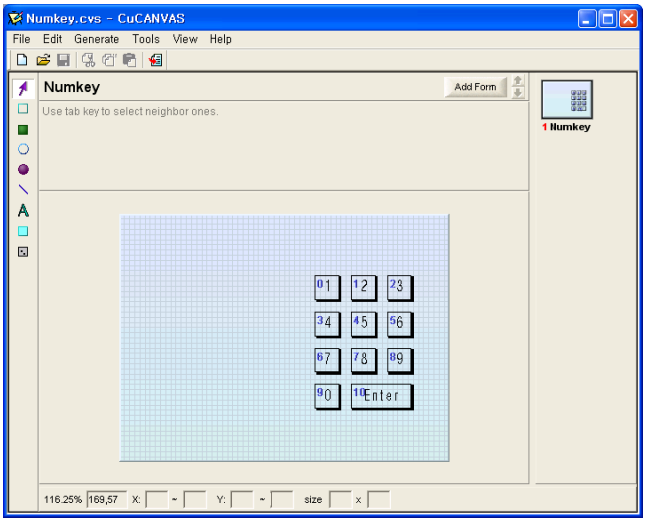


화면 왼쪽에 있는 툴 바를 보면 박스,원, 선 등을 그릴 수 있는 도구들이 있습니다. 이중 가장 아래에 위치한 박스가 바로 "메뉴 박스"를 그리는 툴입니다. 이 툴을 선택한 뒤 화면상에 조그마한 박스를 그립니다.

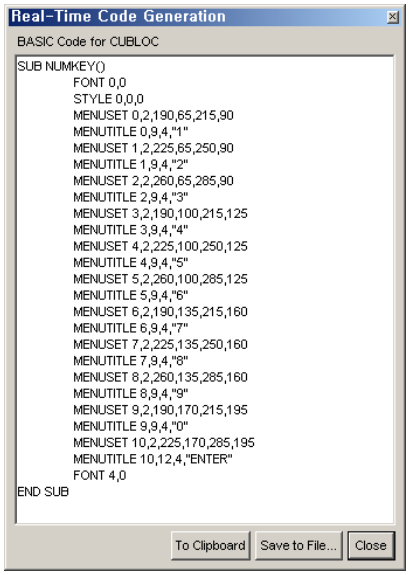
박스 상단의 0 이라는 표시는 0 번 메뉴라는 뜻입니다. 실제 LCD 화면상에는 표시되지 않습니다. 상단에 TITLE 에 1 이라고 입력하면 1 번 보턴이 완성됩니다.



아래와 같이 여러 개의 보턴을 그려서 화면을 완성합니다.

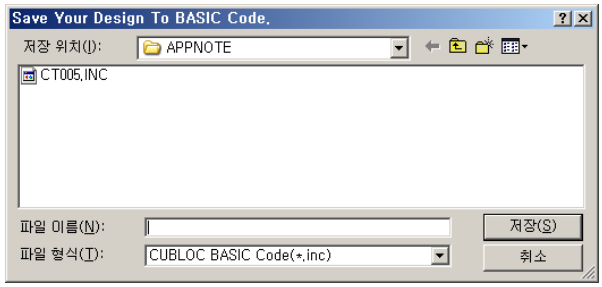


Generate 의 View Basi Code 메뉴를 선택하면 아래와 같이, 최종 소스가 생성됩니다. 이 최종 소스를 Cubloc Studio 상에 그대로 카피해서 붙여 넣게 되면, 위와 같은 화면이 표시됩니다. 이를 위해서 화면 하단의 to Clipboard 를 클릭한 뒤, Cubloc Studio 상에 원하는 위치에 커서를 놓은 뒤, Ctrl-V 키를 누릅니다.



이 방법은 화면을 수정할 때마다 일일이 Copy, Paste 를 반복해 주어야 하므로 매우 번거롭습니다. 보다 편리하게 CuCANVAS 를 사용하기 위해서 INCLUDE 문을 이용한 방법이 있습니다.

Generate 메뉴상에 Save BASIC code As... 메뉴를 선택하신 뒤 적당한 파일명을 선택하십시오.



그러면 다음부터 최종 생성된 BASIC 코드는 이 파일로 자동적으로 저장되게 됩니다. 이 파일을 BASIC 소스에서 INCLUDE 를 시켜주면, 일일이 COPY-PASTE 하지 않아도, 화면이 바뀔 때마다 변경된 상태를 그대로 반영할 수 있게 됩니다.
SAMPLE4 에서 작성한 프로그램을 이 방법을 사용하여 변경한 소스 프로그램입니다.

<파일명 : CT005.CUL>

```
    Const Device = Ct1720
    Dim TX1 As Integer, TY1 As Integer
    Dim I As Integer
    Contrast 550
    Set Pad 0,4,5
    On Pad Gosub GETTOUCH
    NUMKEY                ' INCLUDE된 SUB함수 실행
    I =0
    Do
    Loop

GETTOUCH:
    TX1 = Getpad(2)
    TY1 = Getpad(2)
    If Menucheck(0,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 1
          Pulsout 18,300
    Elseif Menucheck(1,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 2
          Pulsout 18,300
    Elseif Menucheck(2,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 3
          Pulsout 18,300
    Elseif Menucheck(3,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 4
          Pulsout 18,300
    Elseif Menucheck(4,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 5
          Pulsout 18,300
    Elseif Menucheck(5,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 6
          Pulsout 18,300
    Elseif Menucheck(6,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 7
          Pulsout 18,300
    Elseif Menucheck(7,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 8
          Pulsout 18,300
    Elseif Menucheck(8,TX1,TY1) = 1 Then
          I = I << 4
          I = I + 9
```

```
            Pulsout 18,300
      Elseif Menucheck(9,TX1,TY1) = 1 Then
            I = I << 4
            Pulsout 18,300
      Elseif Menucheck(10,TX1,TY1) = 1 Then
            I = 0
            Pulsout 18,300
      End If
      Locate 3,3
      Print HEX4 I

      Return

      End

#INCLUDE "CT005.INC"
```

앞 부분은 똑같고, 뒷부분에 END 명령 뒤에 #INCLUDE 명령이 있는 것만 차이점이 있습니다. 앞에서 NUMKEY 함수를 콜 하면, 화면상에 보턴이 표시됩니다.

이후, CuCANVAS 에서 화면을 고치고, CUBLOC STUDIO 에서 RUN 하면, 고친 화면이 그대로 반영되는 것을 확인할 수 있습니다.

제 4 장
CUBLOC STUDIO

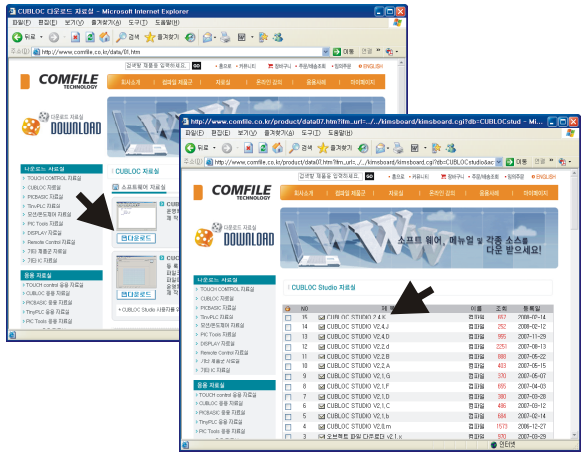
*CUBLOC STUDIO 는 CUTOUCH CT172X/C를 프로그래밍하기 위한 개발환경 소프트웨어이며 www.comfile.co.kr 에서 무료로 다운로드 가능합니다. 본 사용설명서에 있는 화면모습은 사용하고 있는 CUBLOC STUDIO의 버전에 따라 다소 차이가 있을 수 있습니다.

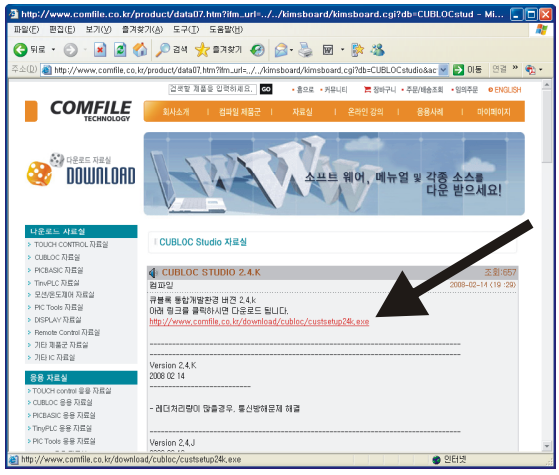
*레더 편집과 관련된 기능은 별도의 매뉴얼인 “레더로직 중심 사용설명서”에 언급하고 있습니다.

CUBLOC STUDIO 설치

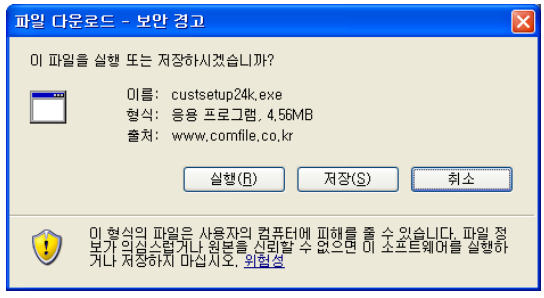
제품 구입시 제공되는 CD 로 설치할 수 있습니다. 가능한 한 인터넷 홈페이지 www.comfile.co.kr에서 다운로드 받아서 설치할 것을 권장합니다. CD보다는 홈페이지에서 가장 최근 버전의 CUBLOC STUDIO를 다운로드 받을 수 있기 때문입니다.

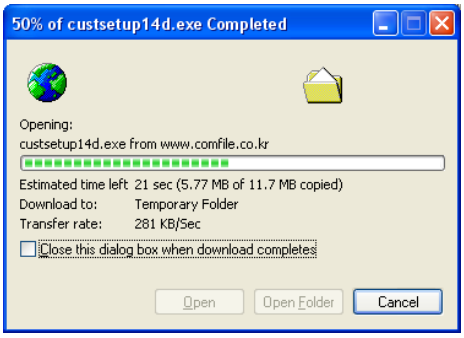
1. www.comfile.co.kr에서 다운로드 메뉴에서 cubloc을 선택하신 뒤, CUBLOC Studio 다운로드를 클릭.
2. 가장 최신버젼의 CUBLOC STUDIO 를 선택하세요. (화살표가 있는 곳을 따라가면서 클릭합니다.)



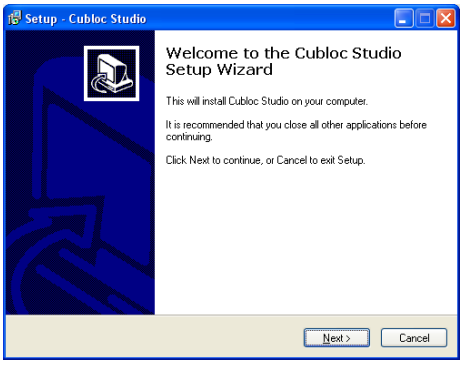


3. 화살표로 표시된 곳을 클릭하면 다운로드를 여부를 확인하는 박스가 표시됩니다. 이때 Open (실행)을 선택하세요.

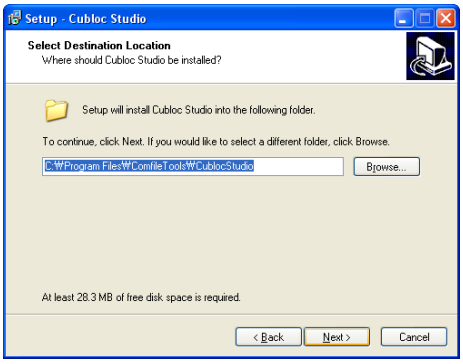




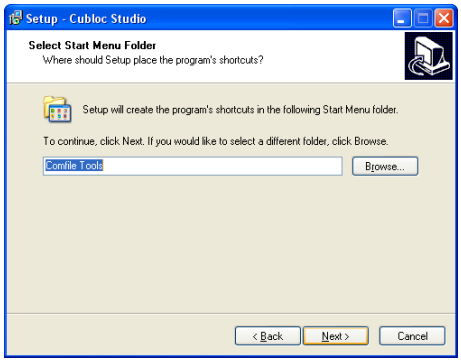
4. 다운로드가 진행중임을 알리는 바가 표시됩니다.



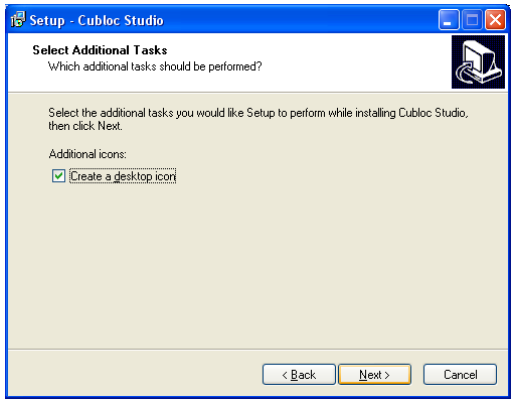
5. 다운로드가 다 끝나면 설치가 시작됩니다. Next 를 누르면 다음단계로 진행합니다.



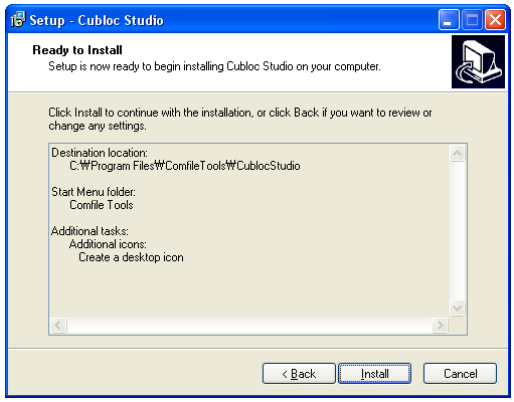
6. 설치할 폴더를 물어보는 다이얼로그 박스가 표시됩니다. 기본 상태는 Program Files 밑에 ComfileTools 라는 폴더 안에 저장되도록 되어 있습니다. 유저가 원하는 다른 폴더로 변경하는 것도 가능합니다.



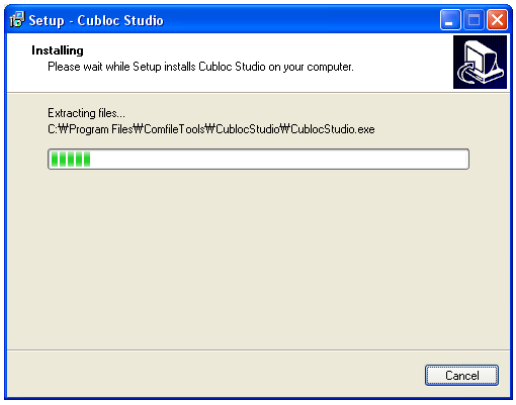
7. 시작(Start)메뉴에서 어떤 이름의 폴더를 사용할 것인지를 선택하는 다이얼로그 박스입니다. 이 상태에서는 그냥 Next 를 누르십시오.



8. 바탕화면에 아이콘을 만들 것인지 물어봅니다. 바탕화면에 CUBLOC STUDIO 아이콘을 만드는 것이 여러모로 편리하므로, 체크를 하도록 합니다.



9. 끝으로 지금까지의 선택사항을 한눈에 볼 수 있는 화면이 표시됩니다. 이상이 없으면 Install 을 누르면, 설치가 진행됩니다.

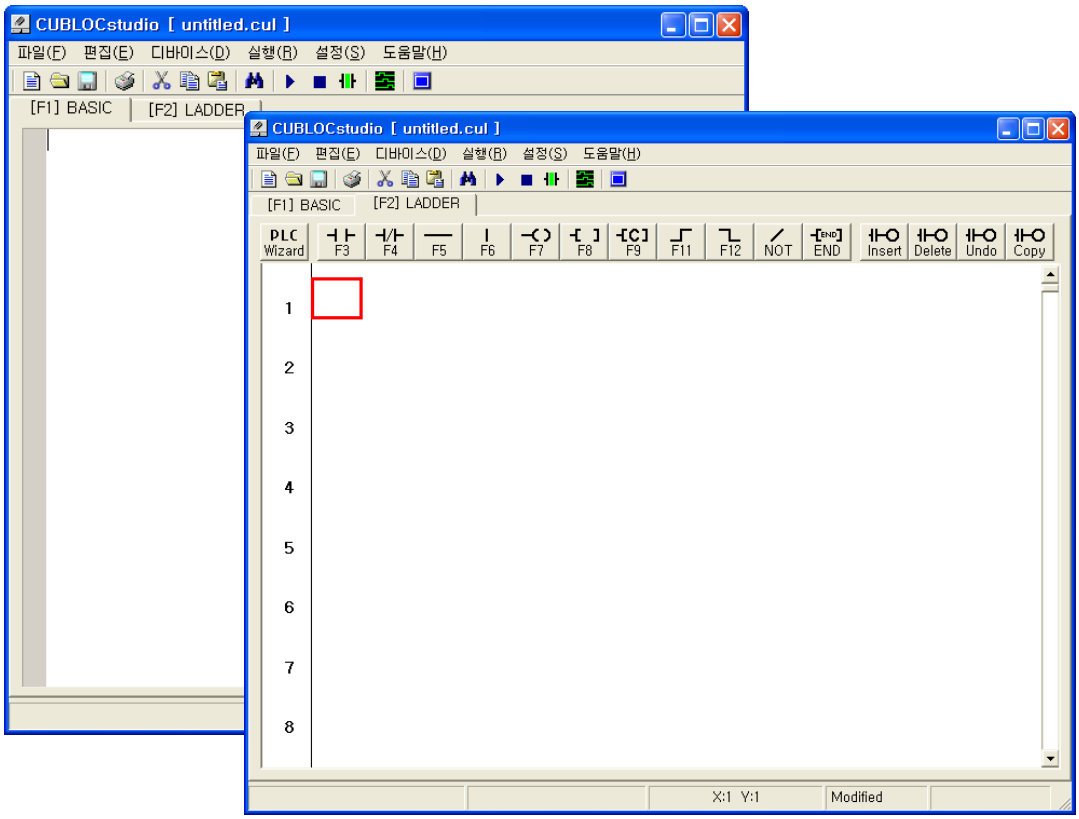


10. 설치가 진행됩니다.

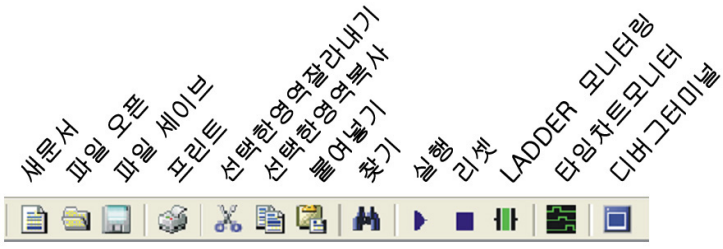
화면의 초록색 바가 끝까지 도달하면 설치가 완료됩니다.

CUBLOC STUDIO 사용법 기초

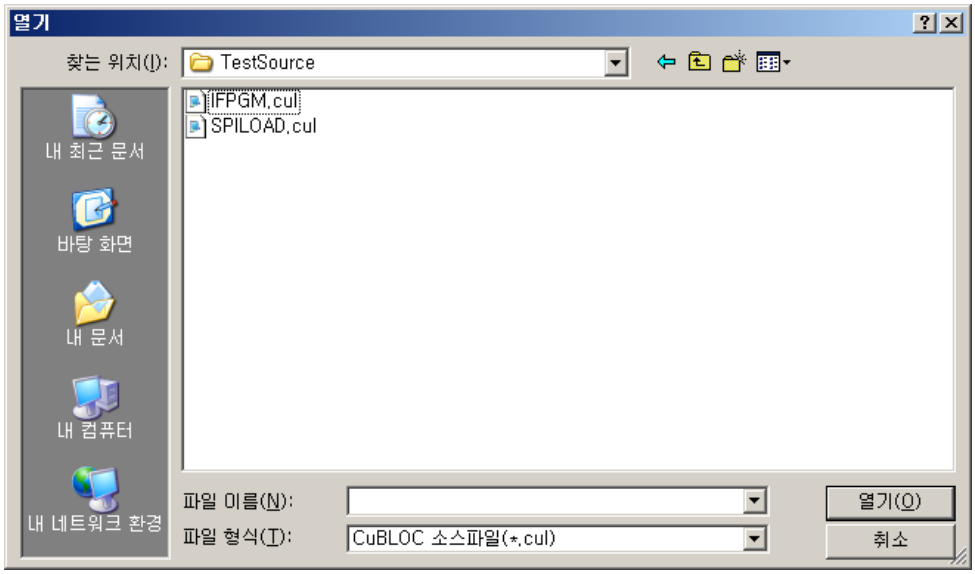
CUBLOC STUDIO 을 설치한 후 실행하면 다음과 같은 빈 화면이 표시됩니다.



처음에는 BASIC 프로그램을 작성할 수 있는 TEXT EDITOR 상태가 됩니다. F2 키를 누르면 LADDER LOGIC 을 편집할 수 있는 LADDER EDITOR 화면이 표시되고, F1 키를 누르면 다시 BASIC 프로그램을 편집할 수 있는 TEXT EDITOR 상태가 됩니다. 화면 상단의 툴 바를 이용하여 기본적인 조작을 할 수 있습니다. 가장 많이 사용하는 3 가지 보턴은 “파일오픈”, “파일세이브”, “실행” 보턴입니다.



소스파일은 확장자 .CUL 과 .CUB, 두 개의 파일로 저장됩니다. 따라서 파일을 따로 백업하거나, 이동할 때 반드시 두 개 파일을 같이 옮겨야 합니다. 나머지 .OBJ 파일등은 다운로드과정에서 저절로 생성되므로 따로 백업하지 않아도 됩니다.



파일 오픈 시에는 위의 그림처럼 .CUL 파일만 표시됩니다. (.CUB 파일을 표시하지 않습니다. 하지만 실제로는 같은 폴더에 들어 있습니다.) .CUL 파일을 오픈하면 자동적으로 같은 이름의 CUB 파일이 오픈됩니다.

유저가 작성한 소스는 PC 상에만 저장할 수 있습니다. "CUBLOC 모듈"에 저장했다가 나중에 다시 불러오는 **"소스 업로드"기능은 지원하지 않으므로, 주의하시기 바랍니다.** 소스를 잃어버렸을 경우, CUBLOC 모듈 에서 소스를 다시 불러올 수 있는 방법은 없으므로 소스프로그램의 보관 및 백업에 항상 신경 써 주시기 바랍니다.

오브젝트 다운로드

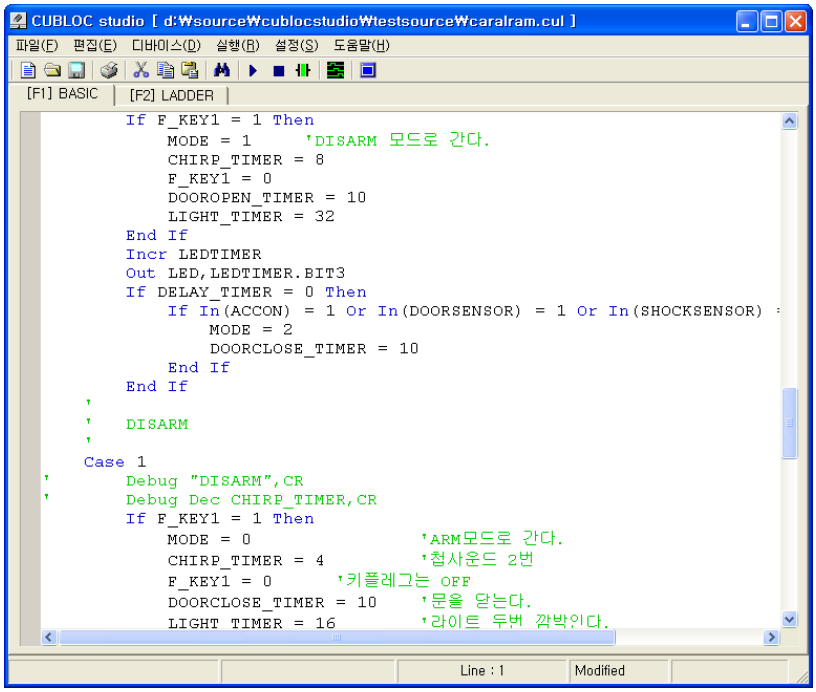
컴파일/다운로드 후 파일명.OBJ 파일이 자동적으로 생성됩니다. 이 파일을 File 메뉴에 있는 "오브젝트 다운로드"에서 선택하면, 큐블록으로 다운로드됩니다. 만약 여러분이 현장에 있는 실무자에게 소스공개 없이 실행파일만 전달하고 싶을 때에는 OBJ 파일과 CUBLOC STUDIO 만 카피해주면 됩니다. 이 파일은 소스를 오픈하거나, 수정할 수 없고, 단지 다운로드만 가능한 파일입니다.

메모

CUBLOC 모듈은 기본적으로 "코드 프로텍션"을 지원합니다. 다운로드 된 유저 프로그램을 읽어낼 수 없도록 설계되어 있습니다. 가령, 특수한 장비를 사용하여 반도체 칩 내부의 데이터를 읽어낸다 하더라도, 암호화되어 알 수 없는 숫자들만 읽혀질 뿐입니다.

RUN 을 누르면 (또는 단축키 CTRL-R) **저장-컴파일-다운로드-실행**이 됩니다. LADDER 와 BASIC 모두 RUN 보턴 하나로 컴파일 에서 실행까지 한번에 수행됩니다. 번역도중 에러가 발생되면 에러멧세지가 표시되고, 에러가 발생한 곳으로 커서가 이동합니다.

BASIC 은 다음과 같이 작성합니다. CUBLOC Text Editor 는 일반적인 에디터와 동일한 사용방법을 가지고 있으며 칼라링과 일부 특수 기능키를 지원합니다.



기능키	*동작설명*
CTRL-Z	UNDO
CTRL-O	OPEN
CTRL-S	SAVE
	*소스작성 중 자주 CTRL-S 키를 눌러 저장하는 습관을 들이는 것이 좋습니다.
CTRL-C	COPY
CTRL-X	CUT
CTRL-V	PASTE
CTRL-F	FIND (찾기)
CTRL-H	REPLACE (치환)
CTRL-HOME	문서의 맨 처음으로
CTRL-END	문서의 맨 끝으로

메뉴 설명

파일 (File) 메뉴

메뉴	*설명*
새로 만들기	새로운 파일을 작성하기 위해 BASIC 과 LADDER 영역을 모두 클리어 합니다.
열기	저장해 놓은 CUBLOC 파일을 불러옵니다.
레더 IMPORT	저장해 놓은 CUBLOC 파일 중 LADDER 만 읽어서, 현재 LADDER 편집영역의 커서가 있는 부분에 삽입시킵니다. 다른 파일에서 레더만 가져올 때 사용하는 메뉴입니다.
저장하기	편집중인 내용을 파일로 저장합니다.
다른 이름으로 저장	다른 이름으로 저장합니다.
오브젝트만 저장	소스형태로 저장하지 않고, 오브젝트형태로 저장합니다. 오브젝트로 저장한 파일은 소스로 변환되지 않으므로, 소스 유출을 방지할 수 있습니다. 오브젝트로 저장한 파일은 아래 "오브젝트 다운로드"에 의해서 다운로드만 가능할 뿐, 편집할 수 없습니다.
LADDER 인쇄	LADDER 영역에 있는 레더소스를 프린터로 출력합니다. 아래의 프린터 설정을 먼저 하신 후 사용하시기 바랍니다.
BASIC 인쇄	BASIC 소스를 프린터로 출력합니다. 프린터 설정을 한 뒤, 프린트하도록 되어 있습니다.
프린터 설정	LADDER 영역 프린트를 위한 프린터 설정 창을 띄웁니다.
오브젝트 다운로드	오브젝트 파일을 CUBLOC 모듈로 다운로드 합니다.
BMP download mode for CT172X/C	CT172X/C 를 위한 BMP 다운로드 기능을 제공합니다.
BASIC 편집모드로..	BASIC 편집모드로 이동합니다. F1 을 누른 것과 동일합니다.
LADDER 편집모드로..	LADDER 편집모드로 이동합니다. F2 를 누른 것과 동일합니다.
최근 편집파일	최근에 편집한 파일 4 개를 보여줍니다. 이중 하나를 선택하면, 오픈됩니다.
끝내기	CUBLOC STUDIO 를 종료합니다.

실행 (Run) 메뉴

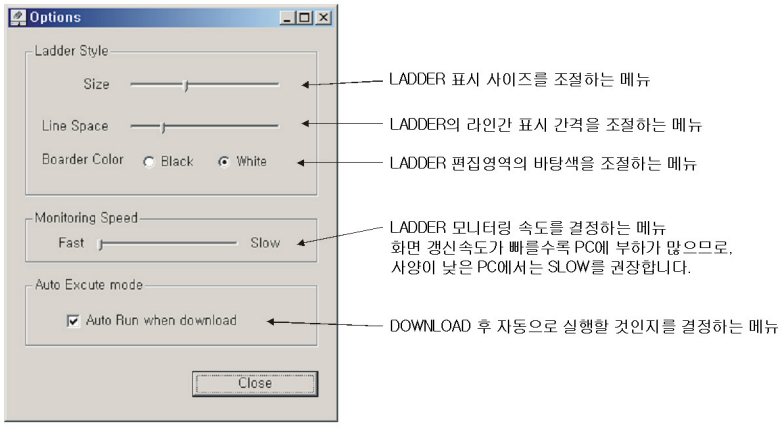
메뉴	*설명*
실행	BASIC 과 LADDER 를 모두 컴파일하고, 에러가 없으면 CUBLOC 모듈로 다운로드 후 실행합니다. 다운로드 후 자동으로 실행하는 것을 원치 않을 경우, 설정(Setup)의 Studio Option 에서 변경할 수 있습니다. *실행 시 "오브젝트"파일이 자동적으로 생성됩니다.
리셋	CUBLOC 모듈을 리셋 시킵니다.
LADDER 모니터링 시작	LADDER 모니터링을 시작합니다.
IntelliLCD Simulator	인텔리 LCD 시뮬레이터를 띄웁니다.
BASIC 디버그 터미널	BASIC 디버그 터미널을 표시합니다. 소스 중 DEBUG 명령이 있다면, RUN 후 자동으로 표시됩니다.
CUBLOC 플레쉬 메모리 지움	CUBLOC 플레쉬 메모리의 내용을 모두 클리어 합니다.
Write 가능 퓨즈 OFF	더 이상 다운로드할 수 없도록 퓨즈를 off 합니다. 이렇게 함으로써, 외부 노이즈충격으로 플레쉬메모리가 지워지는 것을 예방할 수 있습니다. 다시 다운로드 가능상태로 만들고 싶다면, 설정메뉴에 있는

	펌웨어 다운로드를 실행하십시오.
릴레이 사용현황 보기	(컴파일 후) 레더에서 사용한 릴레이를 보여줍니다.
문법검사	큐블록이 연결되지 않은 상태에서 문법상의 오류만 체크할 때 사용하는 메뉴입니다.

설정(Setup)메뉴

메뉴	*설명*
PLC 셋업 마법사	레더를 위한 베이직 소스 자동 생성 마법사
PC 인터페이스 설정	PC 와의 인터페이스를 위한 RS232 COM PORT 를 선택하는 메뉴입니다. COM 1 ~ 4 중 하나를 선택합니다.
에디터 환경 설정	BASIC 텍스트 에디터의 환경을 설정하는 메뉴입니다.
환경설정	CUBLOC Studio 의 세부상황을 선택할 수 있는 메뉴입니다.
영문메뉴로 전환	영문메뉴로 전환해줍니다. 영문메뉴 상태에서 이 자리에는 Use Korean menu 로 바뀝니다. Use Korean menu 를 선택하면 다시 한글 메뉴상태로 복귀됩니다. 한글 메뉴상태에서 모든 에러멧세지는 한글로 표시되고, 영문 메뉴상태에서 모든 에러멧세지는 영문으로 표시됩니다.
펌웨어 다운로드	CUBLOC CORE 의 펌웨어를 다운로드 합니다. 이 메뉴를 사용하지 않아도, RUN 시에 STUDIO 에서 펌웨어 업그레이드 여부를 판단하여, 질문 창을 띄워줍니다.

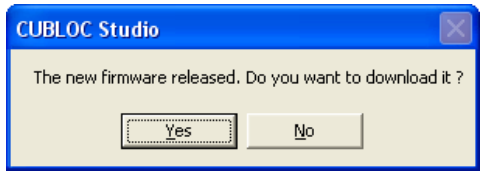
Studio 옵션 설정 창에 대한 자세한 설명입니다.



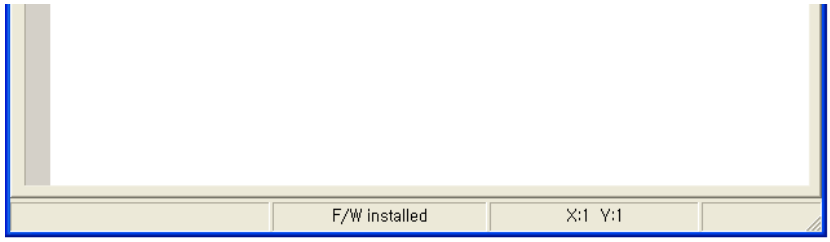
대부분 LADDER LOGIC 편집화면 과 관련된 내용입니다만 BASIC 사용자들도, 이중 AUTO EXCUTE MODE 에 대해서 알아둘 필요가 있습니다. 기계와 연결이 되어 있는 상황에서는 다운로드 후 자동으로 실행하도록 하면 기계동작에 무리가 발생할 가능성이 있습니다. 다운로드 후 자동실행 옵션을 OFF 하게 되면, CUBLOC 은 다운로드 후 정지상태로 머물러 있습니다. 이때 RESET 보턴을 누르거나, CUBLOC 의 전원을 OFF – ON 하게 되면 실행됩니다. 도움말 메뉴에는 CUBLOC 의 사용설명서와 오늘의 팁, 업그레이드 정보, CUBLOC Studio 의 현재 버전 등을 볼 수 있는 메뉴가 있습니다.

펌웨어 다운로드

펌웨어란 큐블록 코어모듈 내부에 저장되는 “시스템 코드”를 말합니다. 즉, 큐블록 코어모듈을 관장하고 운영하는 “Operating System”입니다. 큐블록을 최초 구입했을 때, 그리고 인터넷에서 새로운 버전의 큐블록 스튜디오를 다운받아서 설치한 뒤, 최초 실행했을 때 다음과 같은 멧세지가 표시되는 경우가 있습니다.



“새로운 펌웨어가 나왔습니다. 다운로드 하시겠습니까?”하고 물어보는 것입니다. “Yes”를 클릭하시면 다운로드가 진행되고, 다운로드가 끝나고 나면 화면하단에 “F/W installed”라고 표시됩니다.



펌웨어 다운로드를 유저가 직접 하실 수도 있습니다. “Setup”메뉴의 “Firmware download”를 선택하시면 됩니다. 큐블록코어모듈이 갑자기 다운로드가 안될 경우 펌웨어 다운로드를 하시면, 큐블록의 메모리를 전부 클리어 한 상태에서 펌웨어를 재기입하기 때문에, 이전 프로그램의 잘못으로 인한 “멀펑션”상태에서 벗어날 수 있습니다.

참고하세요!

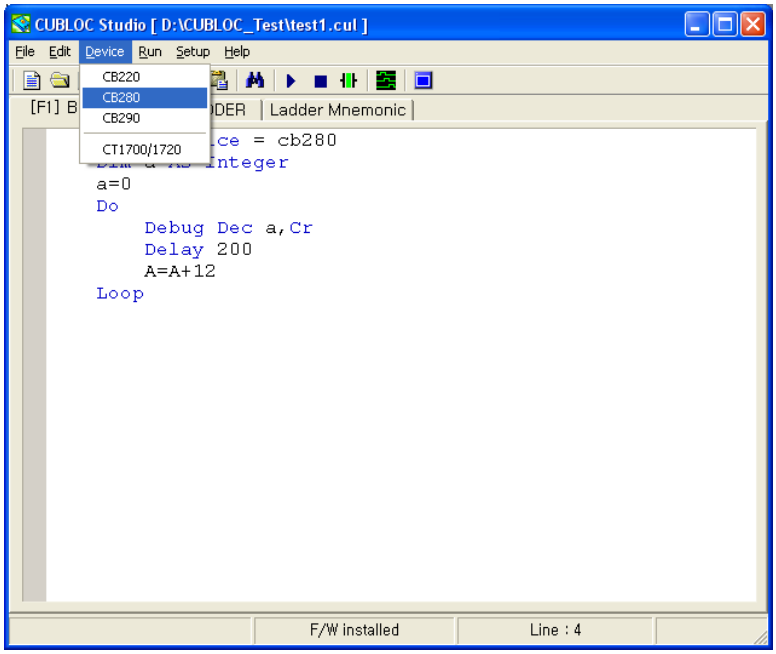
너무 빠른 DEBUG 반복수행으로 간혹 다운로드시 RS232 포트를 찾을수 없다고 에러멧세지가 나오는 경우가 발생합니다. 이 경우에도 펌웨어 다운로드를 수행시키면, 플레쉬 메모리가 모두 지워져서 다운로드 가능한 상태가 됩니다.

DO..LOOP 안에 DEBUG 명령어 하나만 두었을 경우에 너무 빠른 RS232 통신이 발생되므로, 반드시 DELAY 명령과 함께 사용하시기 바랍니다.

```
Do
    Debug Dec I,CR
    Delay 200
Loop
```

디바이스 선택

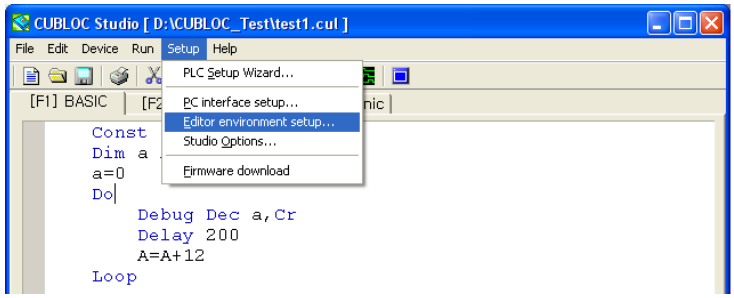
현재 사용하고 있는 큐블록 코어모듈을 BASIC 소스 상에서 선언해주기 위한 명령어가 Const Device = cb280 명령어 입니다. 이 명령은 소스를 작성할 때마다 매번 타이핑을 해주어야 하기 때문에 다소 귀찮을 때가 있습니다. Device 메뉴에서 현재 사용하고 있는 큐블록 모델명을 찾아서 선택하시면, 저절로 Const Device 문을 만들어 줍니다.

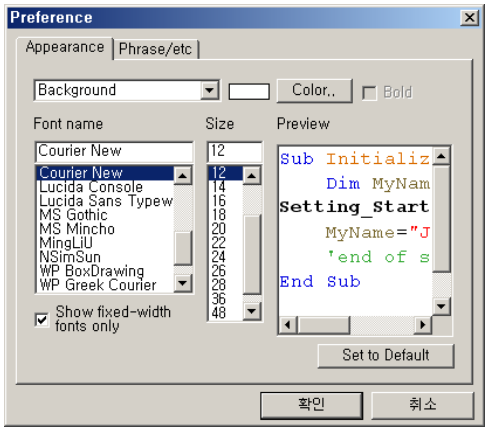


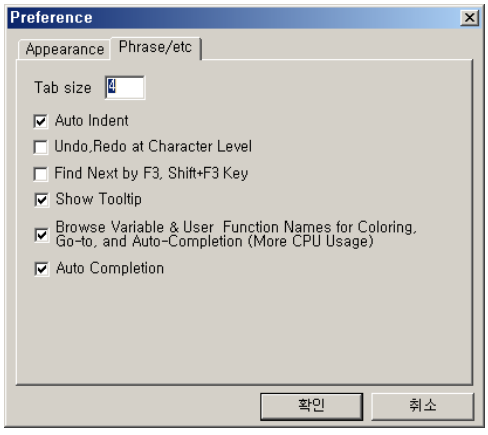
이미 Const Device 문이 소스상에 존재하고 있을 경우에는 그 문장을 찾아서, 뒷부분의 디바이스명만 고쳐줍니다.

에디터의 환경설정

Setup 메뉴의 Editor Environment Setup…을 선택하시면 “텍스트 에디터 환경설정”을 할 수 있습니다.







Appearance 탭

이 다이얼로그 박스에서 폰트의 종류와 크기를 선택할 수 있고, 배경화면의 색도 바꿀 수 있습니다.

Preview 에서 바뀔 폰트의 모양을 사전에 미리 확인해 볼 수 있습니다.

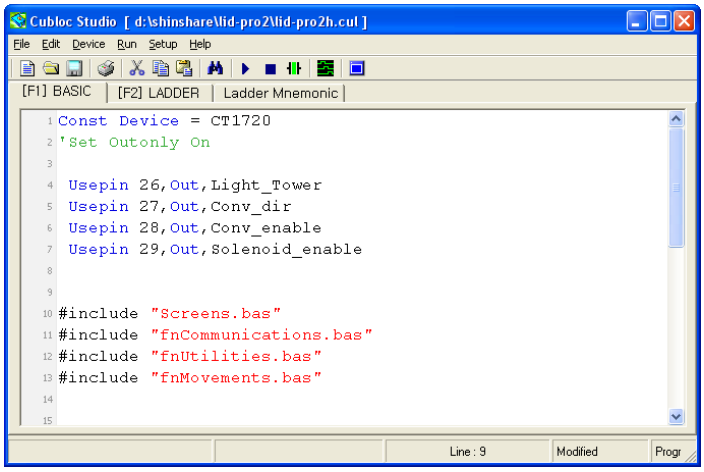
Background 라고 되어 있는 부분의 드롭박스를 변경하여 베이직예약어를 나타내는 컬러를 다른색으로 변경할 수 있습니다.

Phrase / etc 탭

-Tab size : 탭의 기본 사이즈를 입력합니다. (기본값 4)
-Auto Indent : 자동 들여쓰기 기능을 on/off 합니다.
-Undo..: 캐릭터 단위로 undo/ redo 를 수행합니다.
-Find Next…:찾기를 한 다음 F3 키를 누르면, 밑으로 찾기를 계속 수행하고, Shift-F3 을 누르면 위로 찾기를 수행합니다.
-Show Tooltip : 툴팁을 보이게 합니다.
-Browse…: 변수와 부프로그램의 컬러를 변하게 합니다.
-Auto Completion :자동완성 기능을 사용합니다.

긴 소스작성시 테크닉

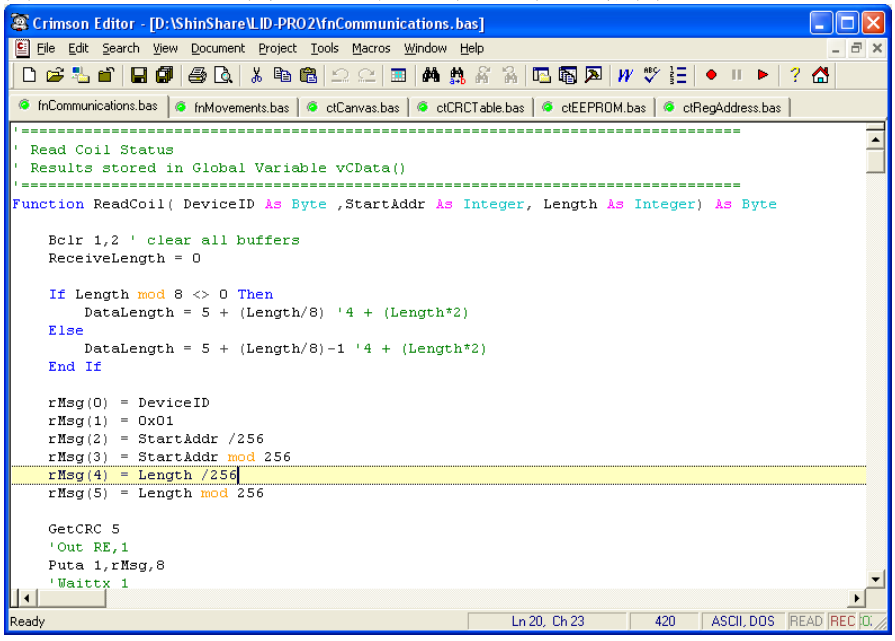
BASIC 소스가 길어질 경우에는 별도의 에디터를 이용해서 소스를 분리해서 작성해는 것이 여러가지로 편리합니다. 왜냐하면 큐블록 스튜디오에서는 하나의 파일만 에디트 할 수 있으므로, 소스가 길어질 경우 위 아래로 찾아다니면서 에디팅 하기가 불편하기 때문입니다.



큐블록 스튜디오 상에는 간단한 선언문만 쓰고, 밑에 #include 로 해당파일들을 불러옵니다.

오른쪽 화면을 보면 기능별로 여러 개의 파일로 나누어져 있습니다.

별도의 텍스트 에디터 (여러분이 즐겨쓰시던 에디터)로 여러 개의 파일을 동시에 오픈시켜놓고 에디팅하고, 반드시 저장한뒤에 CUBLOC STUDIO 에서 RUN 을 누르면 다운로드후 실행됩니다.



위에 있는 에디터는 프리웨어이며 http://www.crimsoneditor.com/에서 다운로드 받을 수 있습니다.

제 5 장
CUBLOC BASIC

CT172X/C 에서 사용하는 BASIC 을 편의상 “CUBLOC BASIC” 이라고 부릅니다. 이 BASIC 은 저희 컴파일 테크놀로지에서 개발한 BASIC 언어로, MICROSOFT 사의 QBASIC, 또는 VISUAL BASIC 문법과 매우 유사합니다.

저희회사에 제품인 CUBLOC, CUTOUCH CT172X/C, CT2400, CUMOTION 등에 모두 이 CUBLOC BASIC 언어를 사용하고 있습니다.

1. CUBLOC BASIC 의 기본구성

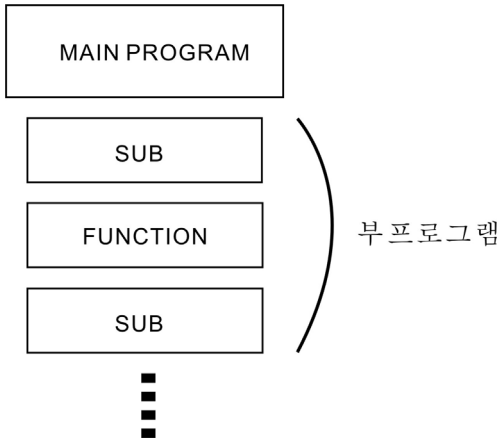
먼저 가장 간단한 형태의 CUBLOC BASIC 프로그램을 보여드리겠습니다.

```
Dim A As Byte
Do
    Byteout 0,A
    A=A+1
Loop
```

포트 0~7 번에 순차적으로 증가되는 값을 출력하는 간단한 프로그램입니다. 다음은 부 프로그램 선언문 FUNCTION 을 이용한 샘플 프로그램입니다.

```
Dim A As Byte
Do
    Byteout 0,A
    A=ADD_VALUE(A)
Loop
End

Function ADD_VALUE(B As Byte) As Byte
    ADD VALUE = B + 1
End Function
```

앞의 프로그램과 같은 동작을 하는 프로그램인데, A 의 값을 증가시키는 부분에 부 프로그램 FUNCTION 을 사용한 것입니다. 이처럼 CUBLOC BASIC 은 메인 프로그램과 여러 개의 부 프로그램으로 이루어집니다. END 명령어 앞에 있는 부분이 “메인 프로그램”이고, SUB, FUNCTION 명령으로 따로 정의한 부분이 “부 프로그램”입니다.



일반적으로 부 프로그램과 메인 프로그램을 가진 구성을 많이 사용합니다만, 간단한 프로그램의 경우 메인 프로그램만으로도 구성할 수 있습니다. 다음은 큐블록 베이직에서 일반적으로 적용되는 규칙입니다.

1.1 명령문 형식

문장이 너무 길어서 한 라인에 쓰기 어려운 경우에는 언더바(_)를 사용해서 여러 줄에 나누어 쓸 수 있습니다.

```
ST = “COMFILE TECHNOLOGY”
ST = “COMFILE _
      TECHNOLOGY”
```

1.2 코멘트

코멘트(주석문)은 어포스트로피(‘)를 사용해서 명령문의 뒷부분 혹은 행의 첫 부분에 사용할 수 있습니다. 코멘트는 컴파일시 번역하지 않습니다.

```
ADD_VALUE = B + 1   ‘B 의 값에 1 을 더합니다.(주석문)
```

1.3 대소문자 구분

CUBLOC 에서는 대소문자를 구분하지 않습니다.

```
Dim MOTOR As Integer
Dim motor As Integer
```

MOTOR 와 motor 는 같은 변수로 취급됩니다.

1.4 행의 분리

CUBLOC 에서는 콜론을 사용해서 반복적으로 명령을 기술할 수 없습니다.

```
A=1: B=1 : C=1   ‘이런 식으로 기술하면 에러가 발생합니다.

A=1          ‘반드시 행을 분리해서 작성하십시오.
B=1
C=1
```

2. 부 프로그램

부 프로그램에는 SUB 형과 FUNCTION 형이 있습니다. SUB 형은 반환 값이 없는 경우 사용하며, FUNCTION 형은 반환 값이 있는 경우 사용합니다.

```
Sub 부 프로그램명 (인수 As 데이터형[,인수 As 데이터형][,...])[As 반환값데이터형]

       [명령문]

       [Exit sub] '수행 중 빠져나올 경우
End Sub

Function 부 프로그램명 (인수 As 데이터형[,인수 As 데이터형][,...])[As 반환값데이터형]

       [명령문]

       [Exit Function] '수행 중 빠져나올 경우
End Function
```

2.1 FUNCTION 형 부 프로그램

어떤 부 프로그램에서 처리결과를 값(Value)로 돌려주고자 할 때, Function 형 부 프로그램을 사용합니다. 이 부 프로그램은 마치 함수처럼, 수식의 일부로 사용할 수 있습니다.

```
Dim K As Integer
K = SUMAB(100,200)   ' 부 프로그램을 호출 (수식의 일부로 사용됨)
End

Function  SUMAB(A AS INTEGER, B AS INTEGER) As Integer
    SUMAB = A + B
End Function
```

부 프로그램 내부에서 반환 값은 "함수명"에 저장하면 됩니다.

```
Function ADD VALUE(B As Byte) As Byte
    ADD_VALUE = B + 1  '부 프로그램명인 ADD_VALUE 에 리턴 값을 저장합니다.
End Function
```

Function 형 부 프로그램을 선언할 때에는 "반환 데이터형"도 함께 적어주어야 합니다. 생략 시에는 Long 형으로 지정됩니다.

```
Function ADD_VALUE(B As Byte) As Byte  ' Byte 형 값을 반환합니다.
End Function
```

2.2 Sub 형 부 프로그램

SUB 형으로 기술한 부 프로그램은 마치 명령어처럼 사용할 수 있습니다.

```
DELAYTIME 100        ' 부 프로그램을 호출
End

Sub  DELAYTIME(DL As Integer)
    Dim  K  As  Integer     ' K 를 지역변수로 선언
    For K=0 To DL
    Next
End Sub
```

Sub 형 부 프로그램을 사용할 때에는 뒤에 괄호를 붙이지 않고 사용합니다.

2.3 인수(Argument)

부 프로그램을 호출할 때 같이 넘겨주는 값이 바로 "인수"입니다. 아래의 경우에는 딜레이에 사용할 값을 인수로 전달하는 것입니다.

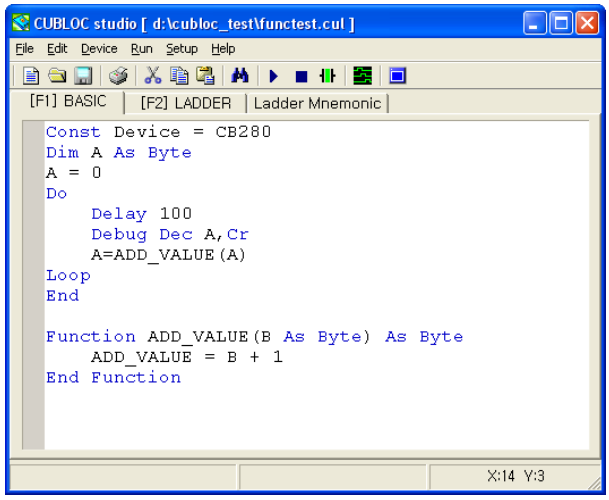
```
Sub  DELAYTIME(DL As Integer)
    Dim  K  As  Integer     ' K 를 지역변수로 선언
    For K=0 To DL
    Next
End Sub
```

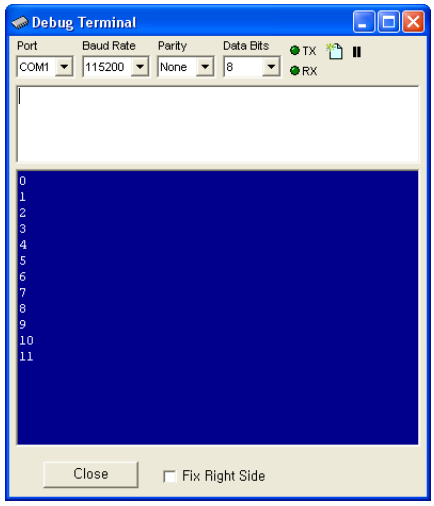
인수가 여러 개 필요한 경우에는 콤머(,)를 사용해서 계속 사용할 수 있습니다. 최대 128 개까지 사용 가능합니다.

```
Sub  DELAYTIME(DL As Integer, DL2 As Integer)
    Dim  K  As  Integer     ' K 를 지역변수로 선언
    For K=0 To DL
    Next
    For K=0 To DL2
    Next
End Sub
```

DEMO PROGRAM

부 프로그램을 사용하여, 기본적인 프로그램을 작성해 보았습니다.





이 프로그램을 실행시키면 화면에는 왼쪽 그림과 같은 "디버그 터미널"이 표시됩니다.

디버그 터미널은 큐블록의 실행상태를 PC 화면에서 볼 수 있도록 해주는 일종의 "디버깅 툴"입니다.

큐블록 베이직 명령어 Debug 가 실행되면, "디버그 터미널"에 결과가 표시됩니다.

이 프로그램의 경우 0 부터 1 씩 증가되는 값이 표시됩니다.

* 본 사용설명서상의 DEMO PROGRAM 은 큐블록코어모듈상에서 실험할 수 있도록 작성되었으나, 대부분 CT172X/C 에서 사용가능합니다. 소스맨앞에 있는 CONST DEVICE 문장을 CT1720 으로 바꾸어 사용하시기 바랍니다.

* DEBUG 결과가 화면상에 표시되지 않는다면 맨앞에 WAIT 500 이라는 명령어를 추가해 보십시오. CT172X/C 이 DEBUG 결과를 출력하기 전에 DEBUG 창이 PC 화면상에 띄어져 있어야 하지만, 일부 PC 에서는 동작속도가 느려서, 화면에 미처 DEBUG 창이 뜨기도전에 CT172X/C 이 결과를 송신해버리기 때문입니다.

2.4 부 프로그램 인수와 리턴 값의 사용상 제한

부 프로그램의 인수와 리턴 값으로는 모든 데이터형을 사용할 수 있습니다.

```
Dim A(10) As Integer

Function ABC(A AS Single) as Single  '실수형 인수와 반환 값
End Function

Function ABC(A AS String * 12) as String *12     '문자열 형 인수와 반환 값
End Funtion

Function ABC(A AS long)   'Long 형 인수와 반환 값
End Function              '함수형을 생략하면 디폴트인 Long 형으로 선언됨.
```

단, 인수로 배열전체를 주는 것은 불가능합니다.

```
Function ARRAYUSING(A(10) AS Integer)      '배열전체를 인수로 사용할 수 없습니다.
End Function
```

배열요소 중 하나를 인수로 전달하는 것은 가능합니다.

```
Dim b(10) as integer
K = ARRAYUSING(b(10)) 'b 배열의 10 번째 요소를 인수로 사용한 것입니다.

Function ARRAYUSING(A AS Integer)     as integer
End Function
```

부 프로그램에 사용되는 인수는 모두 "값에 의한 참조" (Call by value)입니다. "값에 의한 참조"란 부 프로그램을 호출할 때 인수의 값만 전달되는 것입니다. 부 프로그램 내부에서 전달받은 인수의 값을 변경하여도, 본래 인수의 값에는 영향을 미치지 않습니다.

```
Dim  A  As  Integer
Dim  K  As  Integer
A = 100
K = ADDATEN(A)
Debug Dec? A, Dec? K,CR  ' 결과 A 는 100, K 는 110
End

Sub ADDATEN(V As Integer)
    V = V + 10       ' V 의 값을 바꾸어도 A 의 값은 바뀌지 않는다.
    ADDATEN = V
End Sub
```

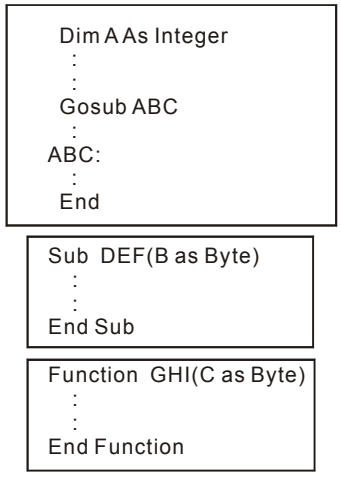
이와는 반대로, "주소에 의한 참조"방식이 있습니다. 인수전달 시, 변수의 내용 대신에 주소를 전달하는 방식이므로, 부 프로그램 내부에서 값을 바꾸면 본래의 값도 바뀌게 됩니다. 큐블록에서는 "주소에 의한 참조"방식을 지원하지 않습니다.

2.5 부 프로그램의 위치

부 프로그램은 메인 프로그램의 뒷부분에 작성해야 합니다. 메인 프로그램의 끝에는 메인 프로그램의 끝을 알리기 위해 END 명령어를 적어주어야 합니다.

```
Dim  A  As  Integer
LOOP1:
    A = A + 1
    Debug  DP(A),CR
    DELAYTIME
    Goto Loop1

    End                ' 메인 프로그램의 끝을 지정, 여기까지가 메인 이라는 뜻

Sub  DELAYTIME()
    Dim  K  As  Integer
    For K=0 To 10
    Next
End Sub
```

END 명령 이후에는 반드시 SUB, FUNCTION 등의 부 프로그램만 위치할 수 있습니다. GOSUB 에서 부르는 서브루틴도 Main 프로그램 내에 위치시켜야 합니다.



*** BASIC 에서는 부 프로그램과 메인 프로그램을 구분해 주기 위해 END 명령을 사용합니다. Ladder 에서는 프로그램의 맨 끝부분에 END 명령을 써주어야 합니다.**

TIPS

부프로그램을 소스앞부분에 작성하고 싶다면, 다음과 같이 GOTO 명령으로 SKIP 하도록 하는 방법이 있습니다.

```
    Const Device = CB280
    Dim  A  As  Integer

    Goto LOOP1

Sub  DELAYTIME()
    Dim  K  As  Integer
    For K=0 To 10
    Next
End Sub

LOOP1:
    A = A + 1
    Debug  DP(A),CR
    DELAYTIME
    Goto Loop1
```

이 경우 END 명령어가 필요없습니다. 주로 #include 안에 부프로그램까지 한꺼번에 넣고 싶을 때 사용하면 편리합니다.

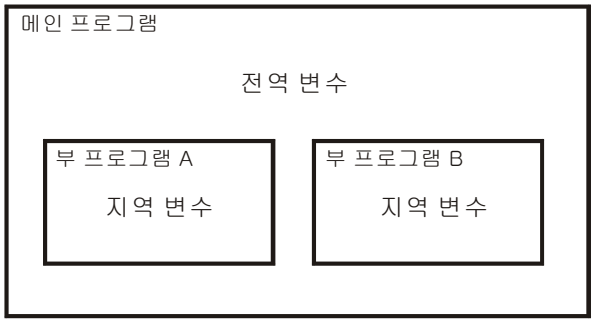
```
#include “CB280init.inc”

LOOP1:
    A = A + 1
    Debug  DP(A),CR
    DELAYTIME
    Goto Loop1
```

이렇게 소스를작성하고 “CB280init.inc” 파일을 다음과 같이 작성하실 수 있습니다.

```
    Const Device = CB280
    Dim  A  As  Integer

    Goto LOOP1

Sub  DELAYTIME()
    Dim  K  As  Integer
    For K=0 To 10
    Next
End Sub
```

3. 지역변수와 전역변수

부 프로그램 안에서 변수형을 선언한 경우에는 “지역변수”로 할당됩니다. 지역변수는 부 프로그램이 호출되면 생성되었다가, 호출이 끝나면 소멸되는 변수 입니다. 지역변수는 다른 부 프로그램에서 호출할 수 없습니다. 반면 메인 프로그램에서 선언된 변수는 “전역변수”입니다. 전역변수는 메인 프로그램뿐만 아니라 프로그램 전체의 모든 부 프로그램에서 사용할 수 있습니다.




```
Dim  A  As  Integer   ' A를 전역변수로 선언
LOOP1:
    A = A + 1
    Debug  Dp(A),CR   ' A의 값을 10진으로 화면 DEBUG창에 표시
    DELAYTIME         ' 부 프로그램 DELAYTIME을 호출
    Goto LOOP1
    End               ' 메인 프로그램의 끝을 지정, 여기까지가 메인 이라는 뜻

Sub  DELAYTIME()
    Dim  K  As  Integer     ' K를 지역변수로 선언
    For K=0 To 10
    Next
End Sub
```

위 프로그램의 경우 A가 전역변수, K가 지역변수가 됩니다. A는 전역변수이므로 전체프로그램에서 사용 가능하지만 K는 DELAYTIME 부 프로그램 안에서만 사용 가능합니다.
지역변수에는 모든 변수형을 사용할 수 있지만, 배열은 사용할 수 없습니다. 배열은 반드시 전역변수로만 사용할 수 있습니다.

*** 지역변수명과 전역변수명을 같은 이름으로 선언할 수 없습니다.**
*** 부프로그램의 인수도 지역변수로 인식되어 집니다.**

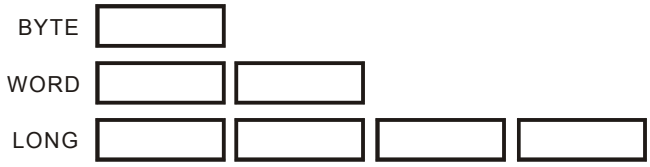
4. 변수형

CUBLOC BASIC 에서의 변수형은 모두 5 가지가 있습니다.

```
BYTE          8 비트 부호 없는 정수, 0~255
INTEGER       16 비트 부호 없는 정수, 0~65535
LONG          32 비트 부호 있는 정수, -2147483648~+2147483647
SINGLE        32 비트 실수, -3.402823E+38~3.402823E+38
STRING        문자열, 0 TO 127 byte
```

BYTE 형은 부호 없는 정수 형으로 8 비트(1 바이트)의 기억 공간을 차지합니다. 따라서 표현 가능한 범위는 0 부터 255 까지 입니다. INTEGER 형은 부호 없는 정수 형으로 16 비트(2 바이트)의 기억 공간을 차지하고, 표현 가능 범위는 0 부터 65535 입니다. LONG 형은 부호 있는 정수 형으로 32 비트 (4 바이트)의 기억공간을 차지하고, 표현범위는 –2,147,483,648 부터 2,147,483,647 입니다.

SINGLE 형은 32 비트 부호 있는 실수형 변수입니다. 표현 가능한 범위는 $-3.402823x10^{38}$ ~3.402823×10^{38} 입니다.



***음수 값을 저장할 수 있는 변수형은 LONG 형과 SINGLE 형 뿐입니다.**

변수를 사용하기 전에, DIM 명령을 이용해서 변수의 기억장소를 선언해 주어야 합니다.

```
DIM  A  AS  BYTE            'A 를 BYTE 형으로 선언합니다.
DIM  B  AS  INTEGER, C AS BYTE '콤마를 사용할 수 있습니다.
DIM  ST1  AS STRING * 12    '문자열변수는 최대바이트 수를 지정해줍니다.
DIM  ST2  AS STRING         '지정하지 않으면 디폴트값인 64 바이트가 됨.
DIM  AR(10) AS BYTE         '바이트형 배열로 선언합니다.
DIM  AK(10,20) AS INTEGER  '다차원배열도 가능합니다. (최대 8 차원까지)
DIM  ST(10) AS STRING*10    '문자열 배열의 선언

Debug a1,cr
```

위 프로그램을 실행시키면, 디버그 창에는 “Comfile Technology, Inc”라고 표시됩니다.

4.1 변수명 작성

변수 명은 영문자로 시작하는 문자를 사용합니다. 명령 또는 함수와 중복되는 이름도 사용할 수 없습니다. 한글을 변수로 사용하는 것도 가능합니다.

```
변수 명으로 적합한 것            : A, B0, I, J, TH, BF1, 압력, 수압
변수 명으로 적합하지 않은 것     : 23, 3A, INPUT, GOTO
```

CUBLOC BASIC 에서는 대문자만을 취급합니다. 소문자를 입력해도 무방하지만 컴파일시 대문자로 번역합니다. 따라서 대소문자를 혼용했을 경우 모두 대문자로 인식하므로 이점을 주의하시기 바랍니다. 예를 들어 CFB_loop 와 CFB_LOOP 는 같은 변수로 인식합니다.

4.2 VAR 명령어 (DIM 의 다른 표현)

VAR 명령도 DIM 명령과 마찬가지로 변수를 선언하는 명령입니다. VAR 명령을 사용하여 아래와 같은 방법으로도 변수선언을 할 수 있습니다.

```
A     VAR   BYTE              'A 를 BYTE 형으로 선언합니다.
ST1   VAR   STRING * 12       '문자열변수는 최대바이트 수를 지정해줍니다.
ST2   VAR   STRING            '지정하지 않으면 (디폴트)64 바이트가 됩니다.
AR    VAR   BYTE(10)          '바이트형 배열로 선언합니다.
AK    VAR   INTEGER(10,20)    '다차원배열도 가능합니다. (최대 8 차원까지)
ST    VAR   STRING *12 (10)   '문자열 배열의 선언
```

4.3 전역변수 메모리 공간

CB220, CB280 의 경우 최대 2048 바이트의 데이터 메모리를 사용할 수 있습니다만, 이 메모리를 전부 전역변수의 저장공간으로 사용할 수는 없습니다. 최소한 지역변수가 사용할 메모리공간과, DISPLAY, RS232 등에서 버퍼로 사용할 공간을 남겨두어야 하기 때문입니다. 디폴트 상태에서는 80 바이트를 제외한 1968 바이트만 사용할 수 있도록 되어 있습니다. 80 바이트는 DEBUG 명령을 위한 버퍼와 최소한의 지역변수를 위한 여유공간입니다. 부 프로그램을 사용할 경우 부 프로그램을 위한 150 바이트 정도의 공간이 더 필요합니다. 따라서 1800 바이트 정도를 전역변수 영역으로 사용하는 것이 안전합니다.

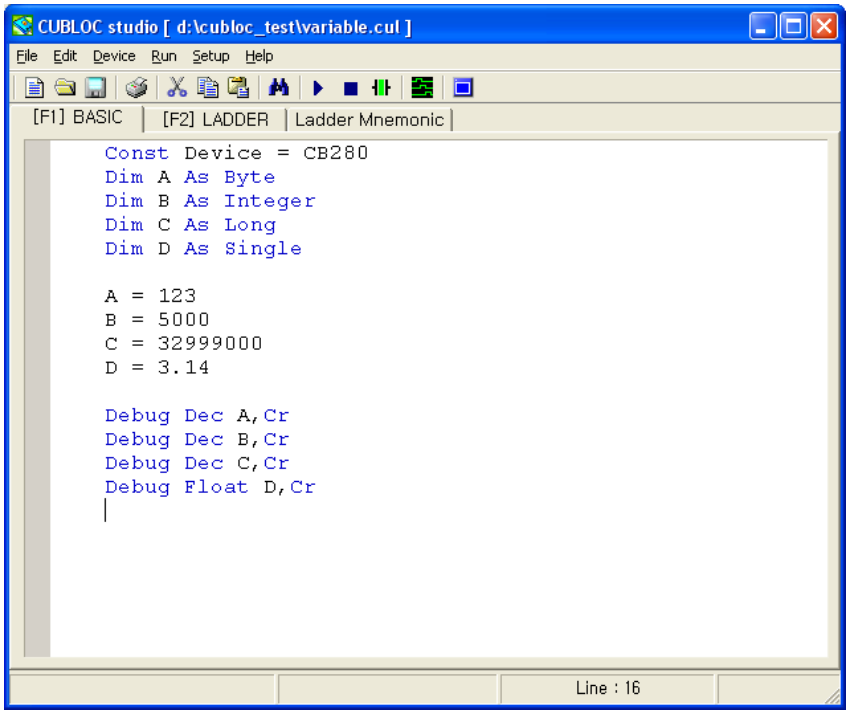
여기에 유저가 SET DISPLAY 명령이나 OPENCOM 명령 등으로 버퍼를 더 사용할 경우, 그만큼 전역변수로 사용할 수 있는 메모리 공간은 줄어들게 됩니다.

4.4 메모리 초기화

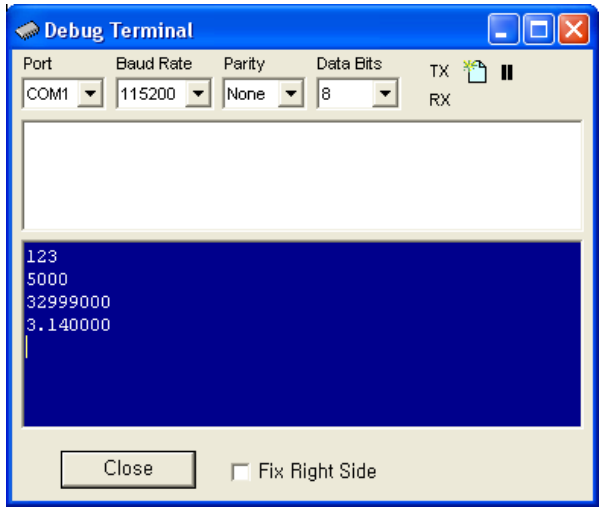
큐블록 BASIC 의 데이터 메모리는 파워 ON 시 자동적으로 클리어 되지 않습니다. 유저가 해당 변수에 0 을 기입하거나, Ramclear 명령을 써서 전체 메모리를 0 으로 클리어 해주어야 합니다. 밧데리로 데이터 메모리를 백업하는 모델에서는 전원 Off 후에도 메모리의 내용을 그대로 기억하고 있으므로, 전원이 다시 On 된 뒤에 따로 클리어 하지 않는다면, 이전 값을 그대로 유지합니다.

DEMO PROGRAM

전역변수를 선언하고 값을 저장한 뒤 표시하는 프로그램입니다.



이 소스를 입력하고 실행시키면, 다음과 같은 DEBUG 터미널이 표시됩니다.



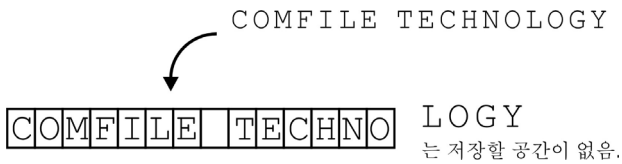
5. 문자열

문자열은 최대 127 바이트까지 사용할 수 있으며, 기억공간 확보를 위한 최대 바이트 수를 써주어야 합니다. 최대 바이트 수를 생략하면 디폴트로 64 바이트가 지정됩니다.

```
DIM  ST  AS  STRING * 14      ' 최대 14 바이트를 넘지 않는 문자열 변수
DIM  ST2  AS  STRING          ' 최대 64 바이트를 넘지 않는 문자열 변수
```

최대길이를 14 바이트로 했을 경우, 실제로는 문자열의 끝을 나타내는 NULL 값을 저장하기 위한 1 바이트의 영역이 더 확보되어, 15 바이트가 할당됩니다. 14 바이트의 공간에 "COMFILE TECHNOLOGY"와 같이 18 바이트 문자열을 저장하였을 경우, 나머지 4 바이트는 저장하지 못합니다. 다음은 문자열 변수의 사용 예입니다.

```
DIM  ST  AS  STRING * 14      ' 최대 14 바이트를 넘지 않는 문자열 변수
ST = "COMFILE TECHNOLOGY"     ' 14 바이트를 넘는 끝부분 "LOGY"는
                              ' 저장하지 못합니다.
```



이와 같이 문자열변수에서는 최대길이를 초과하는 데이터를 저장할 수 없으므로, 최대길이를 신중하게 결정해야 합니다.

CUBLOC BASIC 에서 문자열을 표현할 때에는 쌍따옴표 (")를 사용합니다. 문자열 내부에 쌍따옴표를 포함시킬 수 없으며, 어포스토로피(')도 포함시킬 수 없습니다.

```
ST = "COMFILE " TECHNOLOGY"  ' 문자열 내부에 쌍따옴표를 포함할 수 없습니다.
ST = "COMFILE ' TECHNOLOGY"  ' 문자열 내부에 어포스트로피를 포함할 수 없습니다.
```

쌍따옴표는 CHR(&H22)를 사용하는 방법으로 표현할 수 있습니다. 어포스트로피는 CHR(&H27)을 사용합니다.

```
PRINT CHR(&H22),"COMFILE " TECHNOLOGY",CHR(&H22) ' 쌍따옴표
PRINT CHR(&H27),"COMFILE " TECHNOLOGY",CHR(&H27) ' 어포스트로피
```

문자열을 연결하여 표시하고자 할 때에는 쉼표를 사용합니다. (+나 ;를 사용하면 에러가 발생합니다.)

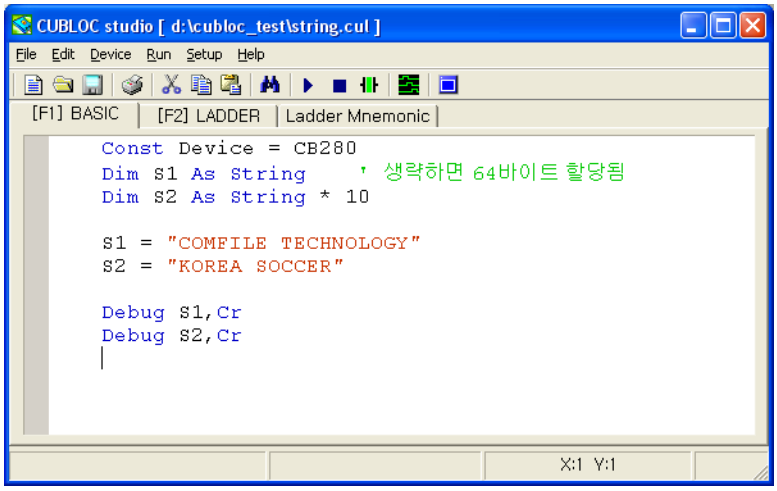
```
PRINT "ABC","DEF","GHI"   ' ABCDEFGHI 로 연결되어 표시됩니다.
```

줄 바꿈을 위하여 CR 을 같이 사용할 수 있습니다.

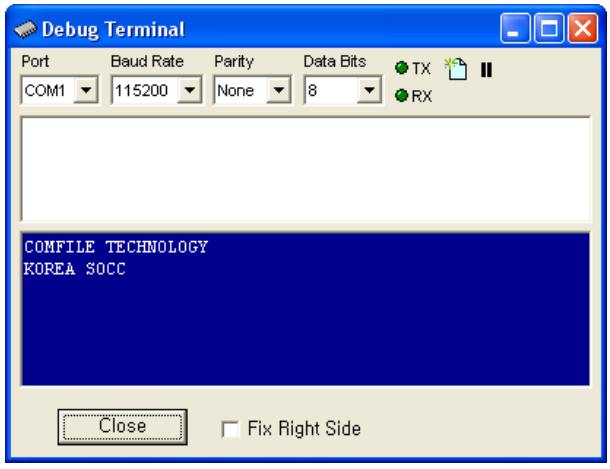
```
PRINT "KOREA",CR   ' KOREA 를 표시하고 줄 바꿈을 합니다.
```

DEMO PROGRAM

문자열 변수를 선언하고, 문자열 상수를 저장한 뒤 표시하는 프로그램입니다.



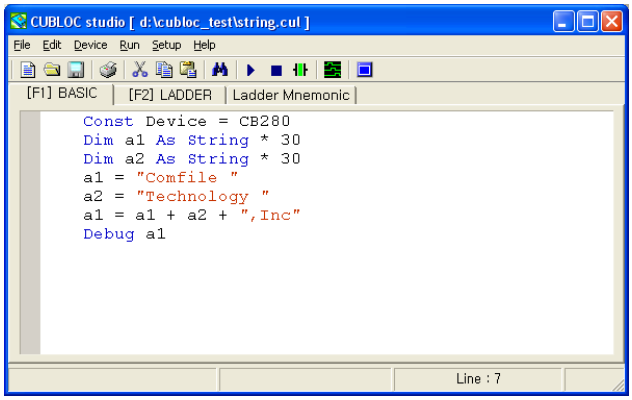
결과는 다음과 같습니다.

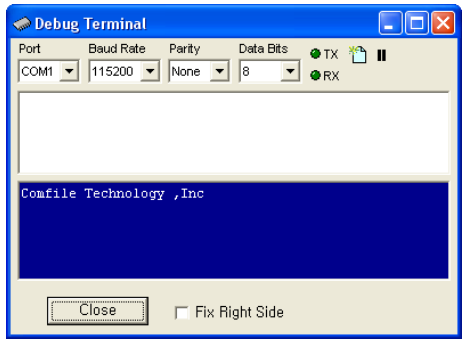


S1 은 64 바이트 길이를 갖는 문자열 변수이므로, “COMFILE TECHNOLOGY”가 모두 저장되어 있지만, S2 의 경우에는 10 바이트 크기이므로,”KOREA SOCCER”가 모두 저장되지 못하고, 뒤에 2 자리가 잘려 있는 것을 볼 수 있습니다.

5.1 문자열의 결합

두 개의 문자열을 결합하기 위해서 + 연산자를 사용합니다.





문자열의 구조

문자열데이터가 메모리에 저장될 때에는 ASCII 코드로 바뀌어 저장되고, 맨 뒤에는 NULL 문자 (ASCII 코드 0)가 저장됩니다. “ABCD”를 변수 ST1 에 저장하는 경우 다음과 같이 저장됩니다.

Dim ST1 as String * 8
ST1 = “ABCD”

캐릭터	A	B	C	D	NULL	X	X	X	X
ASCII	&H41	&H42	&H43	&H44	0				

Dim 선언에 의해 9 바이트의 빈 공간이 확보됩니다. 그리고 ST1 = “ABCD” 명령에 의해 앞에서부터 &H41, &H42, &H43, &H44 가 저장되고 맨 뒤에 0 이 저장됩니다.

Dim 선언에서는 8 바이트크기로 선언했는데, 9 바이트가 확보되는 이유는 맨뒤에 Null 을 저장할 수 있는 공간을 확보하기 위해서입니다. 예를들어 “ABCDEFGH”를 저장했을 때, 맨뒤에 NULL 까지 저장하려면 9 바이트가 필요하게 됩니다.

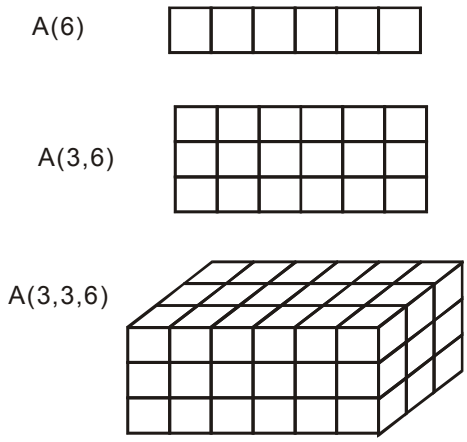
시리얼 통신이나 Debug 명령어에서 여러 개의 문자열을 차례대로 보낼때에는 콤마(,)를 사용합니다.
Debug “Comfile”, “ Technology”
즉, Debug “Comfile” + “ Technology” 와 같이 + 연산자로 처리하지 않아도, 콤마만으로도 간단히 처리됩니다.

6. 배열

CUBLOC BASIC 에서는 배열을 최대 8 차원 배열까지 선언할 수 있으며, 각 차원의 인자는 최대 65535 요소까지 사용할 수 있습니다.

```
DIM   A(20)  AS  BYTE              ' A 배열 20 개를 정의
DIM   B(200)  AS   INTEGER         ' 정수형 배열
DIM   C(200)  AS   LONG            ' 32 비트 정수형 배열
DIM   D(20,10)  AS   SINGLE        ' 실수형 배열을 2 차원배열로 선언
DIM   ST1(10) AS STRING * 12       ' 문자열 배열 선언
```

배열의 인수는 0 부터 시작됩니다. 따라서 20 요소로 정의하면 0 부터 19 까지 사용할 수 있습니다. 문자열 배열은 1 차원 배열만 사용가능 합니다.



다차원배열과 문자열 배열은 메모리를 많이 차지하게 되므로, 주의해서 사용해야 합니다. 선언한 배열의 총 바이트수가 최대 허용 메모리 수를 넘어서지 않도록 주의하시기 바랍니다.

```
DIM   ST1(10) AS STRING * 12      ' 이 경우 13 * 10 = 130 바이트 소요
DIM   D(20,10)  AS   SINGLE       ' 이 경우 4*10 * 20 = 800 바이트 소요
```

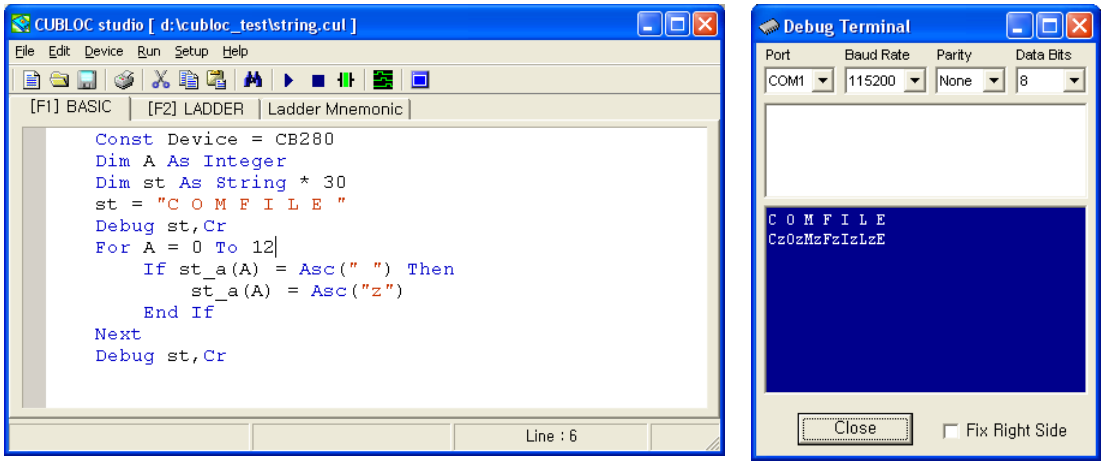
6.1 문자열 변수의 배열접근법

문자열 변수를 마치 1 차원 배열을 다루듯이 취급할 수 있다면 편리한 경우가 많이 있습니다. 예를 들어 문자열 변수 안에 있는 내용 중 단 한글자만 바꾼다던 지, 문자열 변수 안에 있는 내용 중 공백문자가 몇 개나 있는지 확인하고 싶을 때, 문자열 관련 함수로 프로그램을 짠다면 너무 번거롭게 됩니다.

큐블록에서는 문자열 변수를 "BYTE 형 1 차원 배열"처럼 접근할 수 있는 방법을 제공하고 있습니다. 여러분은 단지 문자열변수명 뒤에 "_A"를 붙여서 배열처럼 사용하시기만 하면 됩니다.

```
DIM ST1 AS STRING * 12      ' ST1_A 배열도 동시에 선언되는 것입니다.
ST1 = "123"
ST1_A(0) = ASC("A")         ' 문자열의 첫 번째 요소를 A 로 변경합니다.
```

DIM ST1 AS STRING * 12 에서 ST1_A 바이트형 1 차원 배열도 동시에 선언됩니다. 물론 문자열 변수 ST1 과 동일 메모리공간을 사용하기 때문에, ST1_A 를 사용해서 프로그램을 작성해도 ST1 을 사용하는 것과 같습니다. 다음은 문자열 st 에서 공백만 z 로 바꾸는 예제 프로그램입니다.



단, 아래와 같이 문자열 배열을 사용할 경우에는 이 기능을 지원하지 않습니다.

```
Dim st(10) As String * 3
```

7. 상수

소스 프로그램에서 자주 사용하는 숫자를 상수 명으로 선언해주면, 읽기도 편하고 나중에 수정하는 경우에도 매우 편리합니다. CONST 명령을 사용해서 상수를 선언할 수 있습니다.

CONST 명령을 사용할 때, 상수형을 같이 지정해 주는 경우와 그렇지 않은 경우가 있습니다.

사용 예)

```
CONST  PI  AS  SINGLE = 3.14159
CONST  WRTTIME  AS  BYTE  = 10
CONST  MSG1 AS STRING = “ACCESS PORT”
```

다음 예와 같이, 상수형을 지정하지 않으면, 컴파일러가 적당한 상수형으로 선언합니다.

사용 예)

```
CONST  PI = 3.14159        ‘SINGLE 형 상수로 선언합니다.
CONST  WRTTIME  = 10       ‘부호 없는 255 미만의 값이므로 BYTE 형으로 선언.
CONST  MYROOM  = 310       ‘부호 없는 255 이상의 값이므로 INTEGER 로 선언.
CONST  MSG1 = “ACCESS PORT” ‘문자열 상수로 선언합니다.
```

CON명령어 (CONST의 다른 표현)

CON 명령을 사용해서 상수를 선언하는 방법도 있습니다.

```
PI        CON   3.14159     ‘SINGLE 형 상수로 선언합니다.
WRTTIME   CON   10          ‘부호 없는 255 미만의 값이므로 BYTE 형으로 선언.
MYROOM    CON   310         ‘부호 없는 255 이상의 값이므로 INTEGER 로 선언.
MSG1      CON   “ACCESS PORT”     ‘문자열 상수로 선언합니다.
```

7.1 상수배열

상수란 프로그램 동작 중에도 변하지 않는 고정된 값을 말합니다. CUBLOC BASIC 에서는 상수 여러 개를 배열처럼 정의할 수 있는 기능이 있습니다. 이 기능은 주로 대용량의 데이터를 필요로 할 때 사용합니다. 상수 배열로 정의된 값들은 마치 배열처럼 프로그램상에서 참조할 수 있습니다. 상수배열을 정의하는 방법과 이용하는 방법은 다음과 같습니다.

```
CONST BYTE DATA1 = (31, 25, 102, 34, 1, 0, 0, 0, 0, 0, 65, 64, 34)
I = 0
A = DATA1(I)  ‘ 31 을 리턴합니다.
I = I + 1
A = DATA1(I)  ‘ 25 을 리턴합니다.

CONST BYTE DATA1 = (“CUBLOC SYSTEMS”)
```

BYTE 형에 문자열 데이터를 사용할 수도 있습니다. DATA1(0)을 읽으면 “C”에 해당하는 ASCII 코드가 읽혀집니다. DATA1(1)은 “U”에 해당하는 ASCII 코드가 읽혀집니다.

정수형 및 다른 데이터형의 상수배열도 사용 가능합니다.

```
CONST INTEGER DATA1 = (6000, 3000, 65500, 0, 3200)
CONST LONG DATA2 = (12345678, 356789, 165500, 0, 0)
CONST SINGLE DATA3 = (3.14, 0.12345, 1.5443, 0.0, 32.0)
```

다음과 같이 멀티라인 (여러 줄에 걸쳐서 데이터를 기술 하는 방법)도 가능합니다.

```
CONST BYTE DATA1 = (31, 25, 102, 34, 1, 0, 0, 0, 0, 0, 65, 64, 34,
                    12, 123, 94, 200, 0, 123, 44, 39, 120, 239,
                    132, 13, 34, 20, 101, 123, 44, 39, 12, 39)
```

언더바를 사용해서 멀티라인을 사용해도 무방합니다.

```
CONST BYTE DATA2 = (31, 25, 102, 34, 1, 0, 65, 64, 34,
                   101, 123, 44, 39, 12, 39)
```

문자열도 데이터로 사용할 수 있습니다. 문자열 상수배열은 다음과 같이 최대 가능 바이트 수를 적어주어야 합니다.

```
CONST STRING * 6 STRTBL = (“COMFILE”, “BASIC”, “ERROR”, “PICTURE”)
```

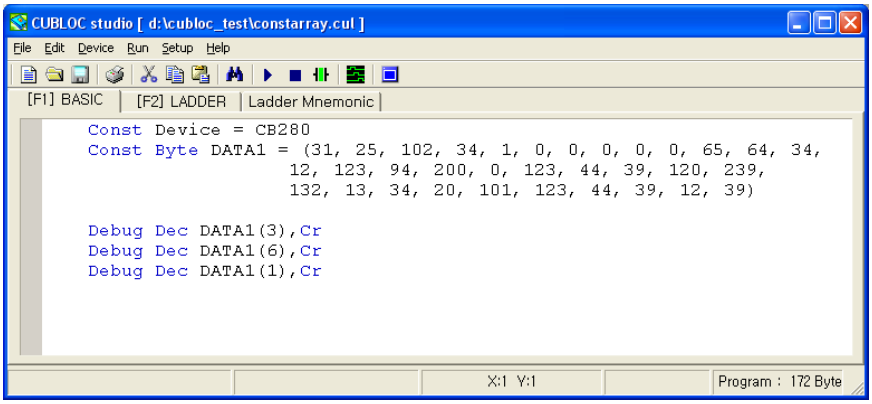
이 경우 각각의 요소는 모두 6 바이트씩 저장공간을 차지하게 됩니다. 만약 6 바이트보다 긴 문자열을 사용한 경우 뒷부분이 잘려서 기억되지 않게 됩니다. 6 바이트보다 작은 데이터는 문제없이 저장됩니다. 이후 소스에서 STRTBL(0)은 “COMFIL”, STRTBL(1)은 “BASIC”을 의미하게 됩니다.

상수배열은 1 차원배열만 선언 가능합니다. 다차원 배열형태로 사용할 수 없습니다. 상수배열로 정의된 데이터들은 **프로그램 메모리상에 다운로드 시 기록**됩니다. 바로 이점이 일반 배열과 틀린 점입니다. 온도테이블등과 같이 이미 결정된 값을 기록해 두었다가, 참조하는 목적으로 사용하시기 바랍니다. 프로그램 수행도중 변경될 필요가 있을 경우에는 일반 배열에 값을 저장해야 합니다.

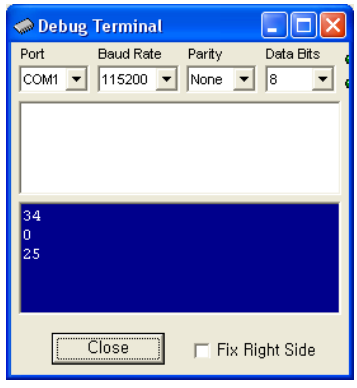
비교항목	배열	상수배열
보관장소	데이터 메모리 (SRAM)	프로그램 메모리 (FLASH)
값이 기록되는 시점	프로그램 실행 시	다운로드 시
실행 중 변경가능	가능함	불가능함
사용목적	실행 중 바뀔 가능성이 있는 데이터의 보관	실행 중 바뀌지 않는 상수 데이터 보관
전원 OFF 시	사라짐	보존됨

DEMO PROGRAM

상수배열을 사용한 예제 프로그램입니다.



실행결과는 다음과 같습니다.



8. 연산자

CUBLOC BASIC 에서 사용되는 산술연산자와 비교연산자, 논리연산자로 구분됩니다.

- 산술연산자 : 더하기 빼기와 같은 수학연산을 하는데 사용합니다.(+ ,-,*등)
- 비교연산자: IF, DO..WHILE 문 등에서 비교연산을 하는데 사용합니다. (<,>,=등)
- 논리연산자: 논리연산을 하는데 사용합니다. (AND,OR 등)

수식표현에서 여러 가지 연산자가 혼합되어 사용될 수 있습니다. 이런 경우 정해진 "연산자 우선순위"에 의해서 계산됩니다. 단, 괄호 안에 있는 수식표현은 어떤 것보다도 가장 먼저 계산됩니다.

연산자	설명	종류	우선순위
^	거듭제곱	산술연산	높음
*,/,MOD	곱하기,나누기,나머지	산술연산	
+,-	더하기,빼기	산술연산	
<<, >>	좌쉬프트,우쉬프트	비교연산	
<, >, <=, >=	작다,크다,같거나작다, 같거나크다	비교연산	
=, <>	같다, 다르다	비교연산	
AND, XOR, OR	AND,XOR,OR 연산	논리연산	낮음

문장 안에서의 연산식도 사용할 수 있습니다.

```
IF A+1 = 10 THEN GOTO ABC
```

문자열변수 및 상수간의 비교도 가능합니다. 문자열 비교에는 ASCII 코드 값이 사용됩니다.

```
DIM ST1 AS STRING * 12
DIM ST2 AS STRING * 12
ST1 = "COMFILE"
ST2 = "CUBLOC"
IF ST1=ST2 THEN ST2 = "OK"    'ST1 과 ST2 가 같은지 비교해 봅니다.
```

정수와 실수를 혼용하여 연산할 수 있습니다. 최종결과는 결과를 저장할 변수형에 따라 변환저장 됩니다. 즉, 최종연산결과를 정수형 변수에 저장하면 최종결과가 실수라도, 소수점 밑부분이 삭제되어 정수로 변환되어 저장됩니다.

```
DIM F1 AS SINGLE
DIM A AS LONG
F1=1.1234
A = F1 * 3.14   '최종결과가 3.525456 이지만, A 가 정수형변수이므로 3 이 저장.
```

단, 실수를 사용하는 연산식에서 사용하는 상수는 반드시.(포인트)를 포함한 실수형태로 적어주십시오.

```
F1 = 3.0/4.0          ' 3/4 를 실수형으로 3.0/4.0
F1 = 200.0 + FLOOR(A) * 12.0 + SQR(B)  '200 을 200.0 으로, 12 를 12.0 으로..
```

AND, XOR, OR 는 논리연산자로 사용되기도 하고, 비트연산자로 사용되기도 합니다.

```
IF A=1 AND B=1 THEN C=1 'A=1 이고 B=1 일때 ...(논리연산자로 사용)
IF A=1 OR B=1 THEN C=1  'A=1 이거나 B=1 일때...(논리연산자로 사용)

A = B AND &HF         '하위 4 비트만 남기고 상위 4 비트를 클리어.(비트연산자)
A = B XOR &HF         '하위 4 비트를 반전합니다.(비트연산자)
A = B OR &HF          '하위 4 비트를 무조건 1 로 만듭니다.(비트연산자)
```

8.1 연산기호

BASIC 에서는 수학에서 사용하는 연산기호와 다른 연산기호를 사용합니다.

Operator	수학식	Basic 언어	BASIC 에서 사용 예
덧셈	+	+	3+4+5, 6+A
뺄셈	−	−	10−3, 63−B
곱셈	X	*	2 * 4, A * 5
나누기	÷	/	1234/3, 3843/A
제곱	5^3	^	5^3, A^2
나머지연산	없음	mod	102 mod 3

나누기의 경우 슬레쉬(/)를 사용합니다. 다음과 같은 분수 식은 괄호와 슬레쉬를 사용한 식으로 열거할 수 있습니다.

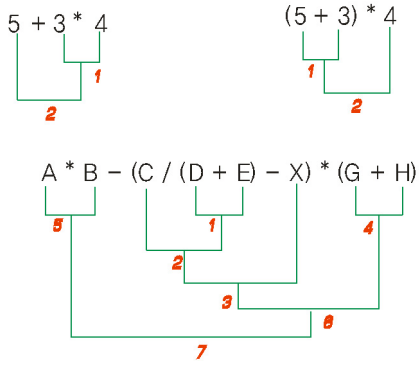
$$\frac{1}{2} \blacktriangleright 1/2 \qquad \frac{5}{3+4} \blacktriangleright 5/(3+4)$$

$$\frac{2+6}{3+4} \blacktriangleright (2+6)/(3+4)$$

8.1 연산 우선순위

하나의 식에서 연산자마다 우선순위를 가지고 있으며, 같은 우선순위에서는 왼쪽에서 오른쪽으로 연산을 진행합니다. 큐블록 BASIC 에서는 다음 순서대로 연산을 수행합니다.

1) 괄호 안의 수식
2) 음수부호 (–)
3) 제곱(^)
4) 곱셈,나눗셈, 나머지연산 (*, /, MOD)
5) 더하기, 빼기 (+ ,–)
6) 좌쉬프트, 우쉬프트 (<<, >>)



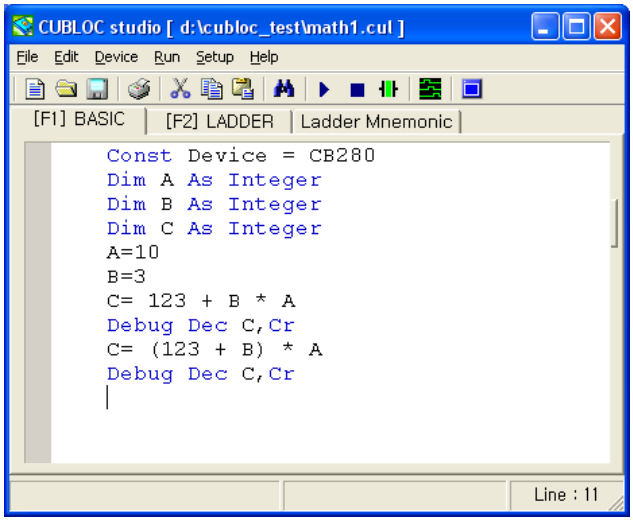
8.1 진법표현

CUBLOC BASIC 에서 진법에 따라 상수를 표현하는 방법은 다음과 같습니다. 16 진수는 전통적인 Basic 형식과 C 언어에서 사용하는 형식, 일부 어셈블러에서 사용하는 형식을 모두 지원합니다.

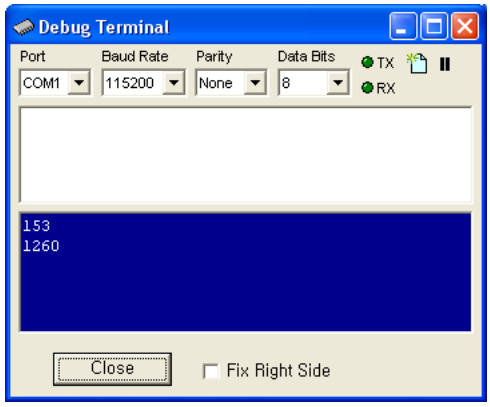
```
10 진수 :    10, 20, 32, 1234

16 진수 :    &HA, &H1234, &HABCD
             0xABCD, 0x1234          ← c 언어에서 사용하는 형식
             $1234, $ABCD            ← 일부 어셈블러에서 사용하는 형식

2 진수 :     &B10001010, &B10101,
             0b1001001, 0b1100
```

DEMO PROGRAM

연산 우선순위를 비교해 볼 수 있는 예제 프로그램입니다.



다음과 같은 결과가 나옵니다. 첫 번째 연산식에서는 B 와 A 를 먼저 연산한 뒤, 123 을 더하기 때문에 153 이 되고, 두 번째 연산식은 123 과 B 를 먼저 더하고, 나중에 A 와 곱하기 때문에 1260 이 됩니다.



8.2 단항연산자

더하기, 나누기 등과 같이 연산을 하기 위해 두 개의 수가 필요한 연산자는 이항연산자라고 부릅니다. 단항연산자는 하나의 수를 가지고 연산을 수행하는 연산자입니다. 대표적으로 마이너스부호가 있습니다. 마이너스 부호가 있는 숫자는 음수 값이 됩니다. CUBLOC BASIC 에서는 이외에도 여러 가지 단항연산자를 지원합니다.

– (마이너스 부호)

어떤 수를 음수(2 의 보수)로 만듭니다.

```
I = -10  '변수 I 에는 음수 10 이 저장됩니다.
```

NOT

어떤 수를 1 의 보수로 만듭니다.

```
I = not 1  '변수 I 에는 -2 가 저장됩니다.
```

NCD

0 에서 31 의 숫자를 앞에 NCD 를 붙이면, 해당 숫자의 비트를 1 로 하는 32 비트 정수 값을 리턴합니다.

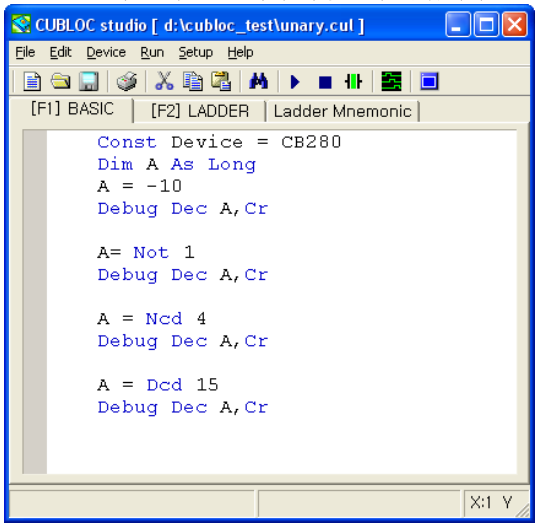
```
I = NCD 4  '결과는 0000000000010000  4 번째 비트가 1 이 된 것입니다.
```

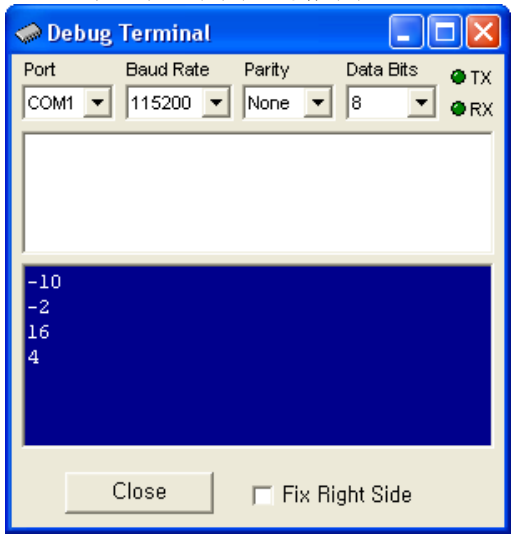
DCD

앞의 NCD 와 반대개념입니다. 어떤 숫자의 비트가 있는 곳의 위치를 알려줍니다. 단, 가장 최상위 비트를 기준으로 합니다. 예를 들어 &B00001111 의 경우 최상위 비트가 위치한 4 을 리턴합니다. 32 비트값에서 최상위는 32, 최하위는 1 이됩니다. 값이 0 일때는 0 을 리턴합니다.

```
I = DCD 15 '결과는 4 입니다.
```

* 단항연산자를 이항연산식에 사용할 수 없습니다. A = B NOT C 라고 하면 에러가 발생합니다.





8.3 실수 수학연산자

다음은 수학식을 계산하기 위한 연산자 입니다. 아래에 열거한 연산자를 사용할 때에는 실수형을 사용해야 정확한 값을 얻을 수 있습니다. 결과를 정수형 변수에 저장하면 소수점 이하는 잘려서 정수부분만 저장되므로 유의하시기 바랍니다.

SIN, COS, TAN

각각 사인, 코사인, 탄젠트 값을 반환합니다. 각도는 라디언 단위를 사용합니다.

```
A=SIN B          '사인 값을 반환합니다.
A=COS B          '코사인 값을 반환합니다.
A=TAN B          '탄젠트 값을 반환합니다.
```

ASIN, ACOS, ATAN

각각 Arc 사인, Arc 코사인, Arc 탄젠트 값을 반환합니다.

```
A=ASIN B         'Arc 사인 값을 반환합니다.
A=ACOS B         'Arc 코사인 값을 반환합니다.
A=ATAN B         'Arc 탄젠트 값을 반환합니다.
```

SINH, COSH, TANH

하이퍼 볼릭 사인, 하이퍼볼릭 코사인, 하이퍼볼릭 탄젠트 값을 반환합니다.

```
A=SINH B         '하이퍼볼릭 사인 값을 반환합니다.
A=COSH B         '하이퍼볼릭 코사인 값을 반환합니다.
A=TANH B         '하이퍼볼릭 탄젠트 값을 반환합니다.
```

SQR

제곱근 값을 반환합니다.

```
A=SQR B          '제곱근 값을 반환합니다.
```

EXP

자연대수 E^X 값을 반환합니다.

```
A=EXP B          '자연대수 E^x 값을 반환합니다.
```

LOG, LOG10

LOG 는 자연로그 값을, LOG10 은 상용로그 값을 반환합니다.

```
A=LOG B          '자연로그 값을 반환합니다.
A=LOG10 B  '상용로그 값을 반환합니다.
```

ABS

정수형 변수의 절대값을 반환합니다. 음수는 양수로 변환됩니다. 실수형은 FABS 를 사용하세요

```
A=ABS B          '절대값을 반환합니다.
```

FABS

실수형 변수의 절대값을 반환합니다. 음수는 양수로 변환됩니다. 정수형은 ABS 를 사용하세요

```
A=ABS B            '절대값을 반환합니다.
```

FLOOR

주어진 실수 값을 넘지 않는 정수를 반환합니다.

```
A=FLOOR B  'FLOOR 3.14 일 경우 3 을 반환합니다.
```

단항연산자는 괄호를 사용할 수도 있고, 사용하지 않을 수 도 있습니다.

```
A=FLOOR(B)
A=SQR(B)
```

8.4 증감연산자

증감연산자는 단항연산자의 변형된 형식으로 볼 수 있습니다. 프로그램을 작성하다 보면 어떤 변수 하나를 1 씩 증가시키거나, 1 씩 감소시키는 경우가 빈번히 발생하게 됩니다. 이럴 때 편리하게 쓸 수 있는 연산자입니다. 증감연산자는 정수만 사용할 수 있습니다.

INCR

변수의 내용을 1 증가시킵니다. 최대치에서 INCR 을 실행하게 되면 다시 0 이 됩니다.

```
INCR A      ‘변수 A 의 값을 1 증가 시킵니다.
```

DECR

변수의 내용을 1 감소시킵니다. 0 에서 DECR 을 실행하게 되면 최대치가 됩니다.

```
INCR A      ‘변수 A 의 값을 1 증가 시킵니다.
```

DTZERO

변수의 내용을 1 감소시킵니다. 만약 0 이 되면 더 이상 감소시키지 않습니다. Down to Zero 의 약자입니다. 어떤 값을 0 이 될 때까지 계속 감소시키고자 할 때 사용합니다.

```
DTZERO A    ‘변수 A 의 값을 1 감소 시킵니다.0 이 되면 더 이상 감소되지 않습니다.
```

UTMAX

변수의 내용을 1 증가시킵니다. 만약 최대치가 되면 더 이상 증가시키지 않습니다. Up to Max 의 약자입니다. 어떤 값을 최대치가 될 때까지 계속 증가시키고자 할 때 사용합니다. 최대치는 변수형에 따라 다릅니다. BYTE 형일 경우 255 가 최대치이고, INTEGER 형일 경우 65535 가 최대치입니다.

```
UTMAX A     ‘변수 A 의 값을 1 증가 시킵니다. 최대치가 되면 더 이상 증가되지 않습니다.



UTMAX A(3)          ‘일반배열의 요소
INCR _D(10)         ‘레더영역중 일부
```

9. 비트, 바이트 지정자

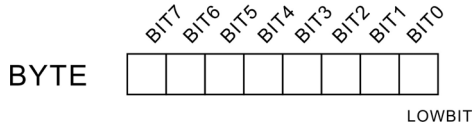
변수의 특정비트 및 니블, 바이트 등을 지정자를 사용해서 억세스 할 수 있습니다. 지정자는 도트(.)를 사용하며, 도트 뒤에 정해진 문자를 작성하면 됩니다.

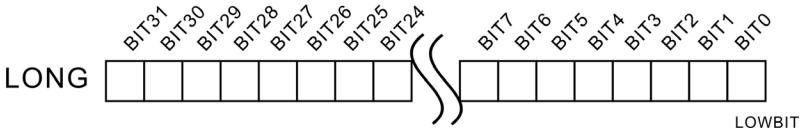
사용 예)

```
DIM  A  AS INTEGER
DIM  B  AS BYTE
B = A.BYTE1       ' 변수 A 의 BYTE1 를 변수 B 에 저장
A.LOWBYTE = &H12  ' 변수 A 의 LOWBYTE 에 &H12 를 저장
```

변수의 특정비트를 지정하기 위해서 사용하는 문자는 다음과 같습니다.

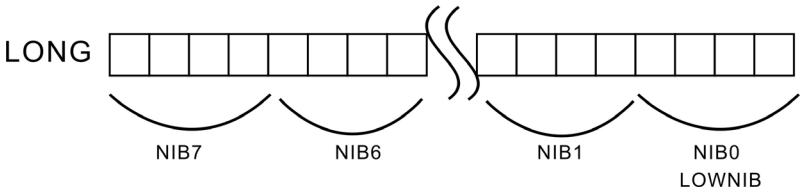
LOWBIT 변수의 bit0
BIT0~31 변수의 bit0~bit31
사용 예)A.BIT2 = 1 '변수 A 의 비트 2 를 1 로 만듦





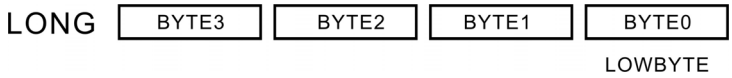
변수의 특정니블을 지정하기 위해서 사용하는 문자는 다음과 같습니다. 니블이란 4 비트를 의미합니다.

LOWNIB 변수의 NIBBLE 0
NIB0~7 변수의 NIBBLE 0~7
사용 예)A.NIB3 = 7 '변수 A 의 니블 3 을 7 로 만듦



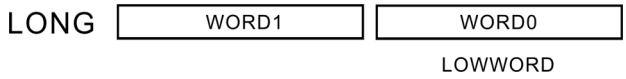
변수의 특정바이트를 지정하기 위해서 사용하는 문자는 다음과 같습니다.

LOWBYTE	변수의 BYTE 0
BYTE0	변수의 BYTE 0
BYTE1	변수의 BYTE 1
BYTE2	변수의 BYTE 2
BYTE3	변수의 BYTE 3



변수의 특정워드를 지정하기 위해서 사용하는 문자는 다음과 같습니다.

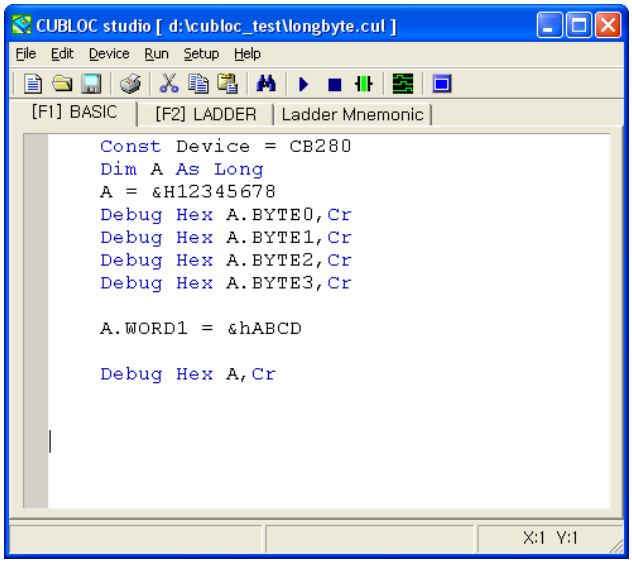
LOWWORD	변수의 WORD 0
WORD0	변수의 WORD 0
WORD1	변수의 WORD 1



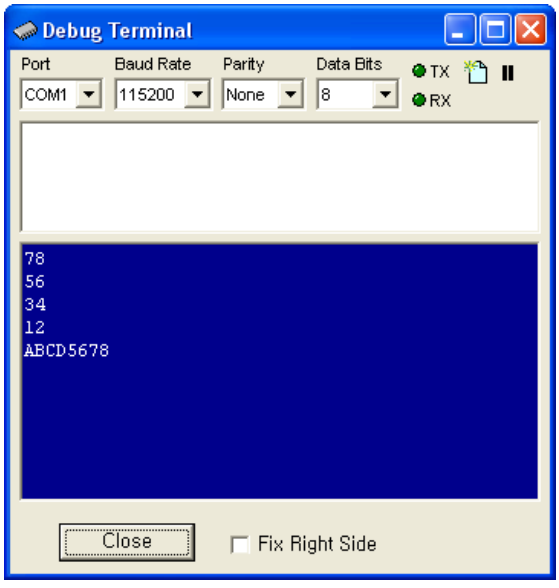
* HIGHWORD, HIGHBYTE, HIGHBIT, HIGHNIB 는 지원하지 않습니다.

DEMO PROGRAM

LONG 형 변수 A 를 선언하고, 어떤 수를 저장한 다음, 바이트 단위로 쪼개어 표시하는 예제 프로그램입니다. 두 번째로, 변수 A 의 상위 워드(16 비트)에 다른 값을 넣어서 표시합니다.



결과는 다음과 같습니다.



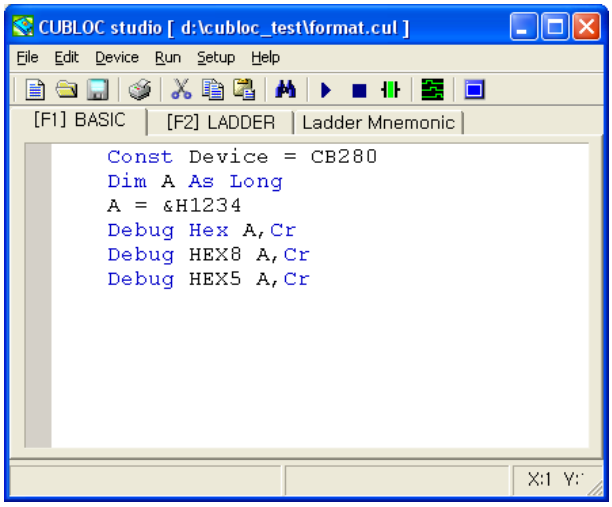
10. 형식변환자

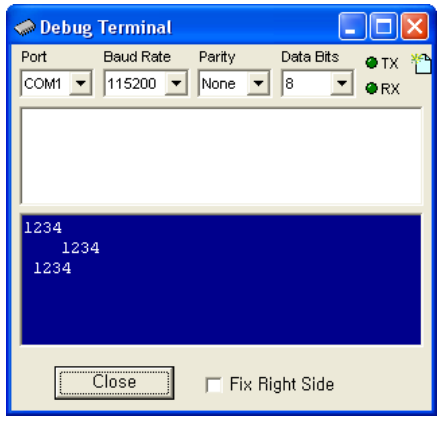
형식변환자는 변수의 내용을 표시할 때 십진수로 표시할 것인지, 16 진수로 표시할 것인지, 자릿수는 얼마로 할 것인지 등을 결정하는 문장입니다.

HEX

변수의 내용을 부호 없는 16 진수로 표시합니다. 바로 뒤에 숫자가 오는 경우에는 "표시하고자 하는 자릿수"를 의미합니다. 예를 들어 HEX8 은 8 자릿수 이내의 문자로 표시하라는 뜻입니다. HEX 에서 자릿수는 1~8 까지 사용 가능합니다.

```
DEBUG HEX A       'A 가 123ABC 일 경우 123ABC 가 그대로 표시됩니다.
DEBUG HEX8 A      'A 가 123ABC 일 경우 bb123ABC 로 표시,
                  '앞부분의 b 자리에 공백이 표시됨
DEBUG HEX5 A      'A 가 123ABC 일 경우 23ABC 로 표시, 앞부분이 잘립니다.
```





DEC

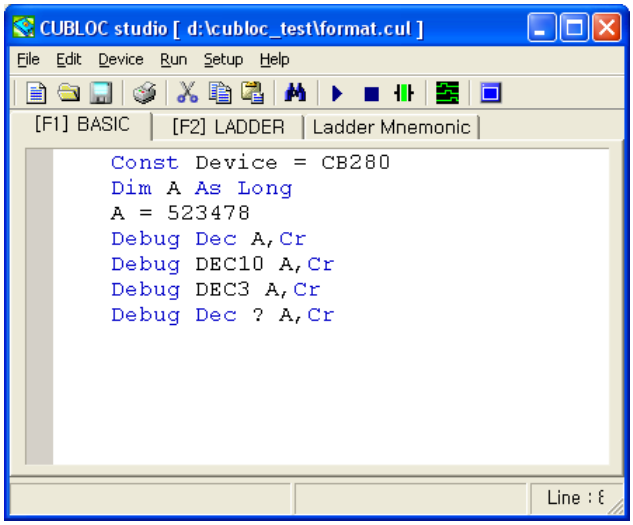
변수의 내용을 부호 있는 10 진수로 표시합니다. 바로 뒤에 숫자가 오는 경우에는 “표시하고자 하는 자릿수”를 의미합니다. DEC 에서 자릿수는 1~11 까지 사용 가능합니다.

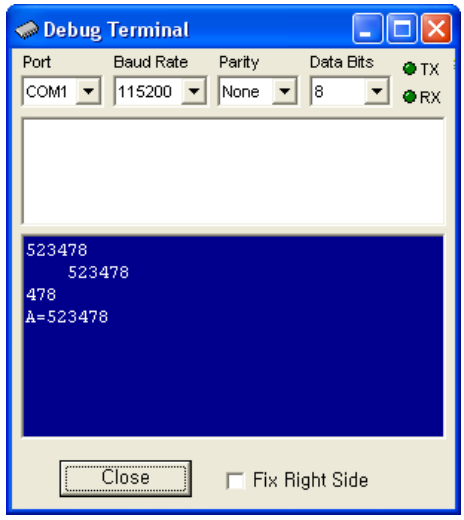
```
DEBUG DEC A       'A 가 1234 일 경우 1234 가 그대로 표시됩니다.
DEBUG DEC10 A     'A 가 1234 일 경우 bbbbbb1234 로 표시, 앞부분의 b 자리에
                  '공백이 표시됨
DEBUG DEC3 A      'A 가 1234 일 경우 234 로 표시, 앞부분이 잘립니다.
```

?

물음표(?)를 형식변환자 뒤에 바로 사용하게 되면, 표시할 변수명도 같이 표시해 줍니다. ?는 HEX, DEC 바로 뒤에서만 사용 가능합니다.

```
DEBUG DEC ? A     'A 가 1234 일 경우 “A=1234”라고 표시됩니다.
DEBUG HEX ? A     'A 가 ABCD 일 경우 “A=ABCD”라고 표시됩니다.
DEBUG HEX ? B     'B 가 부 프로그램 CONV 에 소속된 지역변수일 경우
                  '“B_@_CONV=ABCD”라고 표시됩니다. (CONV 에 있는 B 라는 뜻)
```

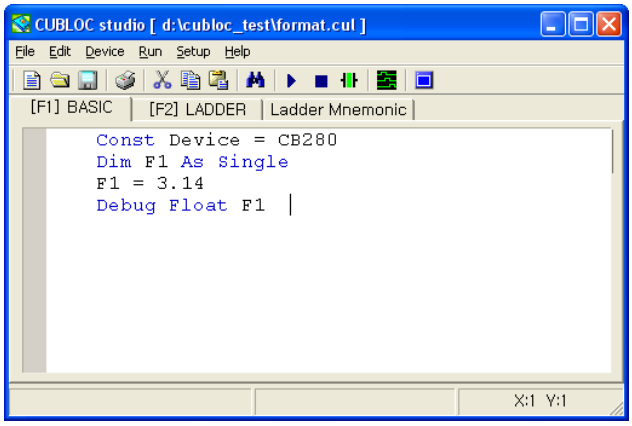


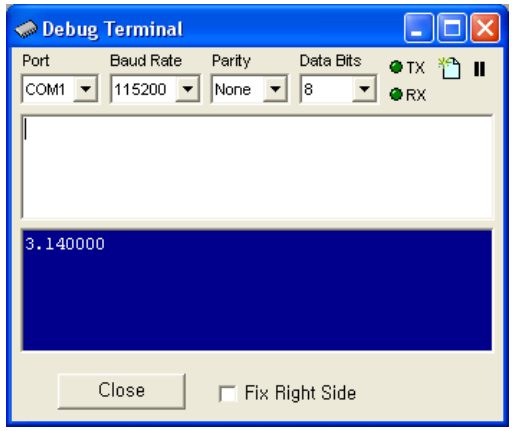


FLOAT

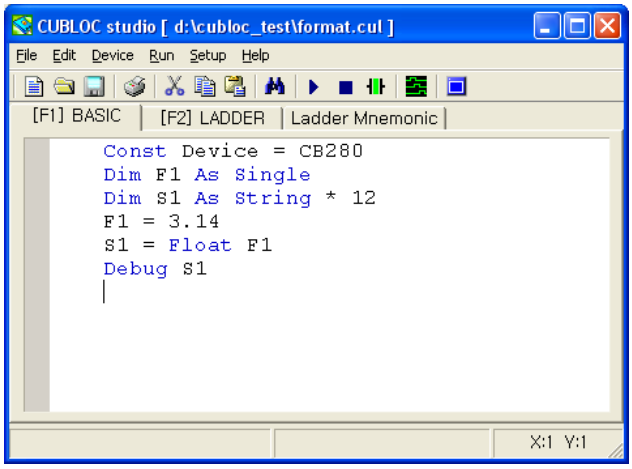
실수를 표시하고자 할 때 사용합니다. FLOAT 에서는 표시자릿수는 별도로 지정하지 않습니다.

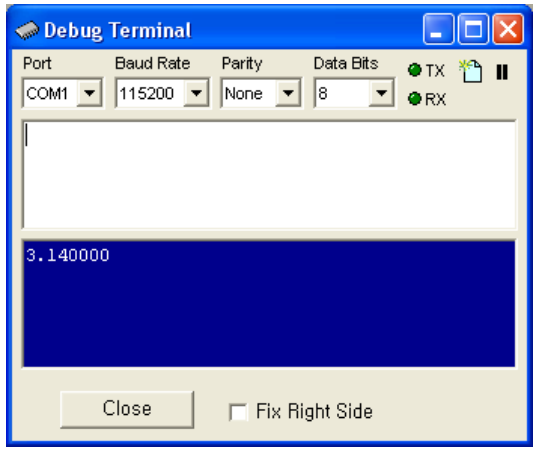
```
DIM F1 AS SINGLE
F1 = 3.14
DEBUG FLOAT F1          '3.140000 로 표시됩니다.
```





문자열 변수에 일단 저장했다가 표시하는 것도 가능합니다.

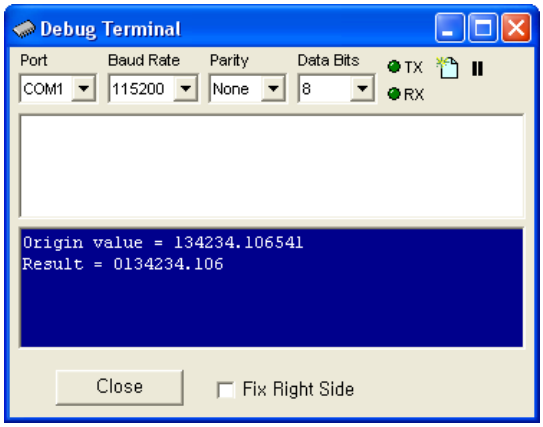




10.1 실수형 숫자의 Format 표시

실수를 표시하는 float 변환자의 경우, 자릿수를 지정할 수 없어서 불편한 경우가 있습니다. 소수부와 지수부의 자릿수를 지정해서 표시하려면, 아래와 같은 부 프로그램을 사용해서 표시할 수 있습니다.

```
        Const Device = CB280
        Dim f1 As Single
        f1 = 134234.1112
        Debug "Origin value = ",Float f1,Cr
        Debug "Result = ",FormatFloat(f1,7,3),Cr

        Do
        Loop
        End

Function FormatFloat(Tval As Single,Jdigit As Byte, Sdigit As Byte) As String * 30
        #define FrontChr &h30
        '               ^
        '             이 부분을 &h20 으로 바꾸면 앞에 오는 0 을 공백으로 바꿀 수 있음.
        Dim i As Byte
        Dim Tempst As String * 30
        Dim Res As String * 30
        Tempst = Float Tval
        For i=0 To Len(Tempst)-1
              If tempst a(i)=&h2e Then Exit For
        Next
        Res = String(FrontChr,29-i)+Left(Tempst,i)
        Res = Right(Res,Jdigit)+"."
        Tempst = Mid(Tempst,i+2,Len(Tempst)-i-1)
        Res = Res + Left(Tempst,sdigit)
        FormatFloat = Res
End Function
```



위 프로그램의 결과는 다음과 같습니다.

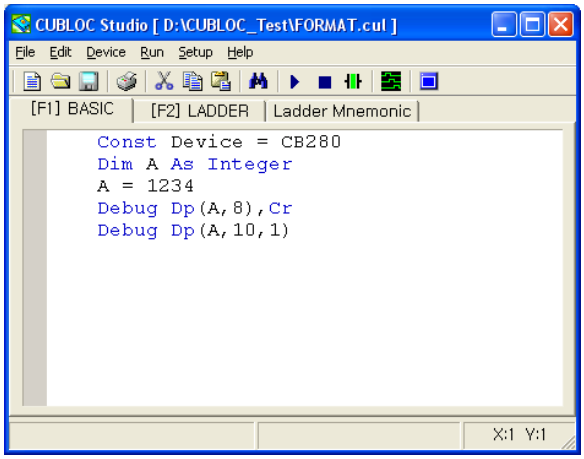
FormatFloat 함수는 문자열을 리턴하는 함수입니다. 실수와 함께, 정수부자릿수, 소수부자릿수를 넘겨주면, 포맷화된 문자열을 반환합니다. (정수부, 소수부의 합이 29 를 넘으면 안됩니다.)
앞의 오는 빈 자리는 0 으로 채웁니다. 만약 공백으로 채우고 싶다면, 소스에서 FrontChr 을 정의하는 부분을 &h20 으로 변경하면 됩니다.

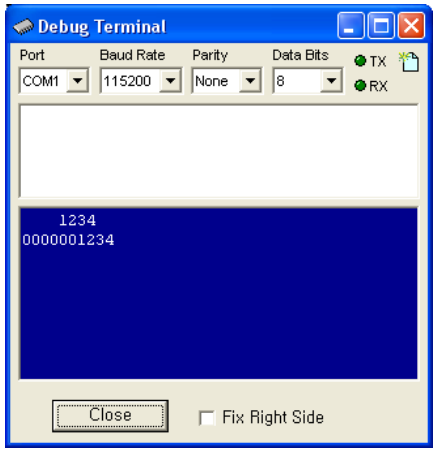
11. 문자열 처리함수

DP(Value, Length, Zero)

숫자를 10 진수 형식 문자열로 변환하는 함수입니다. Zero 위치에 1 을 쓰면, 앞부분 공백을 0 으로 채웁니다. Zero 를 생략하면 앞부분을 공백으로 처리합니다.

```
DEBUG DP(A,8)  '변수 A 를 10 진으로 표시합니다. 표시할 자릿수를 8 자리로 제한.
DEBUG DP(A,10,1)  'A 가 1234 일 경우 0000001234 로 표시합니다.
```

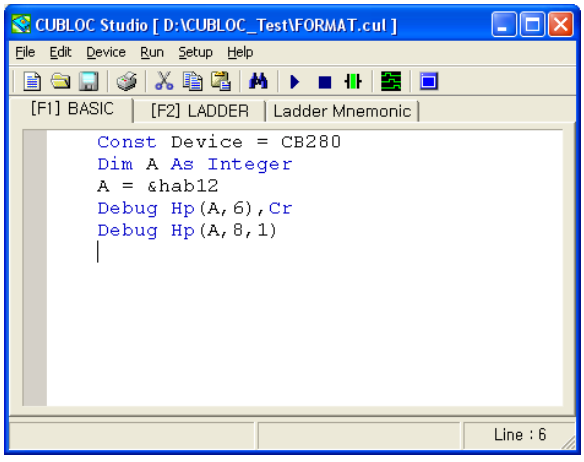




HP(Value, Length, Zero)

숫자를 16 진수 형식 문자열로 변환하는 함수입니다. Zero 위치에 1 을 쓰면, 앞부분 공백을 0 으로 채웁니다. Zero 를 생략하면 앞부분을 공백으로 처리합니다.

```
DEBUG HP(A,6)  '변수 A 를 16 진으로 표시. 표시할 자릿수를 6 자리로 제한합니다.
DEBUG HP(A,8,1)  'A 가 &H12 일 경우 00000012 로 표시합니다.
```

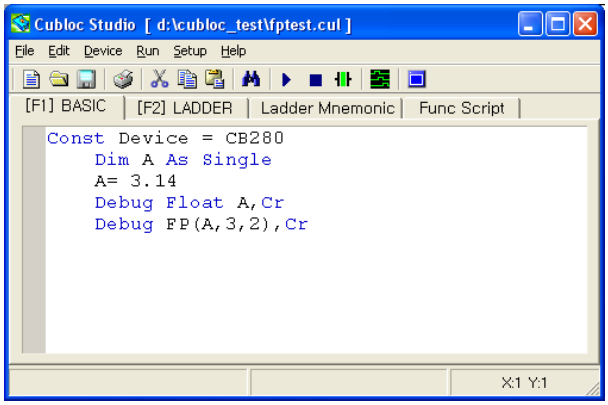


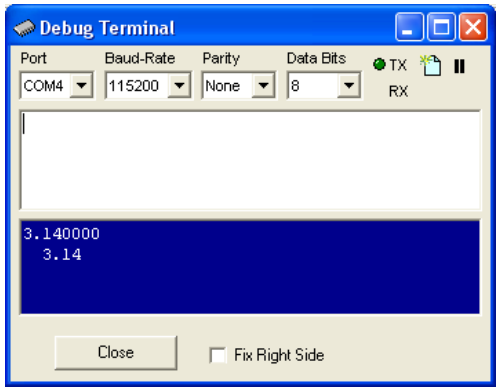
FP(Value, 정수부길이, 소수부길이)

실수형 변수를 일정한 포맷으로 변환하는 함수입니다. 실수를 문자열로 변환하면서 정수부의 길이와 소수부의 길이를 제한하여 변환합니다. 실수를 표시하는데 있어서 Float 와 FP 를 사용하는 방법이 있습니다.

```
Dim A as Single
A = 3.14
DEBUG Float A        ' 3.1400000 으로 모든 유효자릿수가 표시됩니다.
DEBUG FP(A,3,2)      '   3.14 로 표시되며, 유효자릿수가 모두 표시되지 않습니다.
```

실수를 FP 함수를 사용하여 표시하는 경우, 유효자릿수의 제한을 받게 됩니다. 그 결과 소수부에서 FLOAT 함수를 써서 표시한 결과와 다른 결과로 표시될 수 있습니다.





큐블록에서 실수는 IEEE724 포맷으로 저장됩니다. 이를 읽어오는 알고리즘에서 FLOAT 함수와 FP 함수가 다소 차이가 있습니다. 따라서 실제 저장되어있는 실수값이 같은 경우에도 FLOAT 로 표시한 값과, FP 로 표시한 값이 다르게 표시될 수 있습니다.

LEFT(문자열변수,자릿수)

왼쪽에서 자릿수만큼 잘라낸 문자열을 반환합니다. 자릿수는 반드시 상수를 사용해야 합니다.

```
DIM ST1 AS STRING * 12
ST1 = “CUBLOC”
DEBUG LEFT(ST1,4)  ‘CUBL 가 표시됩니다.
```

RIGHT(문자열변수,자릿수)

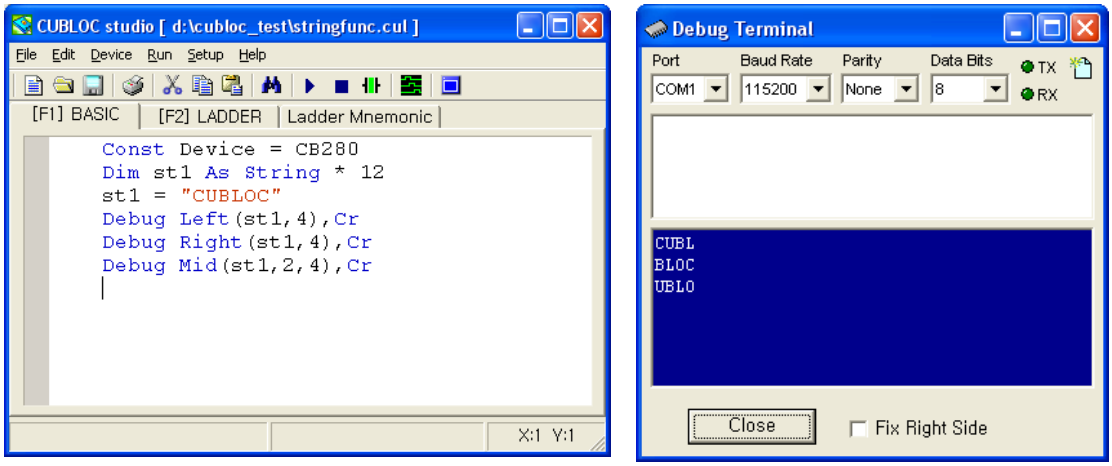
오른쪽에서 자릿수만큼 잘라낸 문자열을 반환합니다. 자릿수는 반드시 상수를 사용해야 합니다.

```
DIM ST1 AS STRING * 12
ST1 = “CUBLOC”
DEBUG RIGHT(ST1,4)  ‘BLOC 가 표시됩니다.
```

MID(문자열변수,위치,자릿수)

문자열의 중간에서 자릿수만큼 잘라낸 문자열을 반환합니다. 위치는 1 부터 시작하며, 문자열 최대자릿수를 초과할 수 없습니다. 자릿수는 반드시 상수를 사용해야 합니다.

```
DIM ST1 AS STRING * 12
ST1 = “CUBLOC”
DEBUG MID(ST1,2,4)  ‘UBLO 가 표시됩니다.
```



LEFT, RIGHT, MID 함수의 경우, 문자열 변수위치에 문자열 상수 (“AXBDF” 이런 식으로 표현된..)를 쓰면 동작하지 않습니다. 이 경우 특별한 에러멧세지 발생되지 않으므로 주의하시기 바랍니다.

```
DEBUG LEFT(“INTER”,4)  ‘문자열 변수 위치에 상수를 사용할 수 없습니다.
```

“자릿수”에 문자열의 최대길이를 넘어서는 수치를 사용하지 않도록 주의바랍니다. 이 경우에도 특별한 에러멧세지를 발생시키지는 않지만, 의도하지 않은 결과가 나올 수 있습니다.

LEN(문자열변수)

문자열변수에 들어있는 문자열의 길이를 반환합니다.

```
DIM ST1 AS STRING * 12
ST1 = “CUBLOC”
DEBUG DEC LEN(ST1)  ‘6 이 표시됩니다.
```

STRING(아스키코드,자릿수)

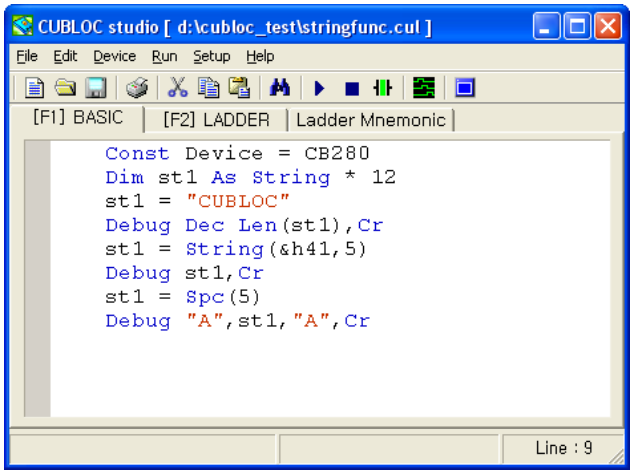
아스키코드에 해당하는 문자를 자릿수만큼의 크기를 가진 문자열로 만들어 반환합니다.

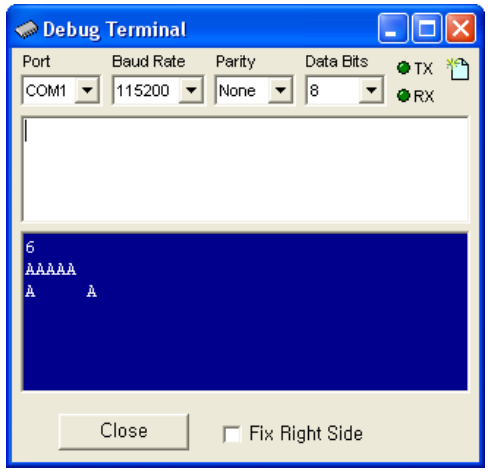
```
DIM ST1 AS STRING * 12
ST1 = STRING(&H41,5)
DEBUG ST1  ‘AAAAA 이 표시됩니다.&H41 은 A 의 아스키코드입니다.
```

SPC(자릿수)

공백문자를 자릿수만큼의 크기를 가진 문자열로 만들어 반환합니다.

```
DIM ST1 AS STRING * 12
ST1 = SPC(5)
DEBUG “A”,ST1,”A”  ‘AbbbbbA 이 표시됩니다.편의상 공백을 b 로 표기했습니다.
```





LTRIM(문자열변수)

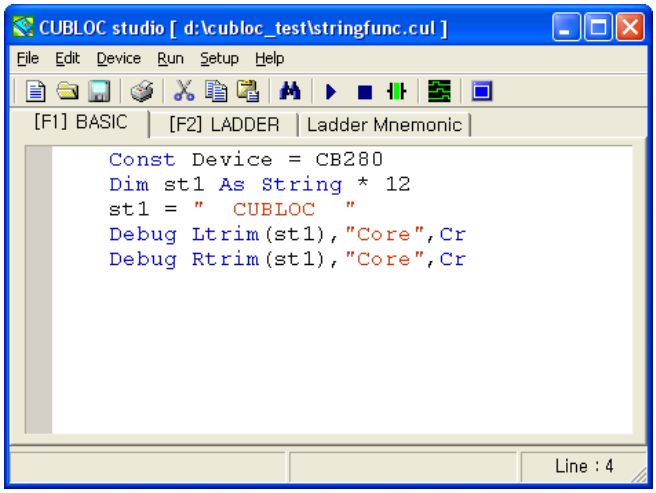
문자열의 왼쪽 공백을 제거하여 반환합니다.

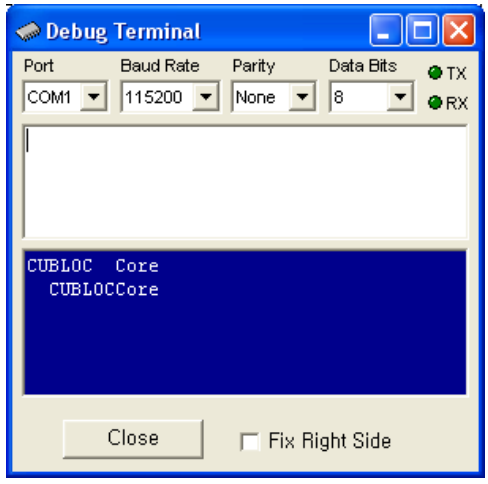
```
DIM ST1 AS STRING * 12
ST1 = “  COMFILE”
ST1 = LTRIM(ST1)
DEBUG “AAA”,ST1  ‘AAACOMFILE 이 표시됩니다.왼쪽 공백이 제거되었습니다.
```

RTRIM(문자열변수)

문자열의 오른쪽 공백을 제거하여 반환합니다.

```
DIM ST1 AS STRING * 12
ST1 = “COMFILE  ”
ST1 = RTRIM(ST1)
DEBUG ST1,”TECH”  ‘COMFILETECH 이 표시됩니다.오른쪽 공백이 제거되었습니다.
```





LTRIM, RTRIM 함수의 경우, 문자열 변수위치에 문자열 상수 (“AXBDF” 이런 식으로 표현된..)를 쓰면 동작하지 않습니다. 이 경우 특별한 에러멧세지 발생되지 않으므로 주의하시기 바랍니다.

```
DEBUG LTRIM(“  INTER”)  ‘문자열 변수 위치에 상수를 사용할 수 없습니다.
```

VAL(문자열변수)

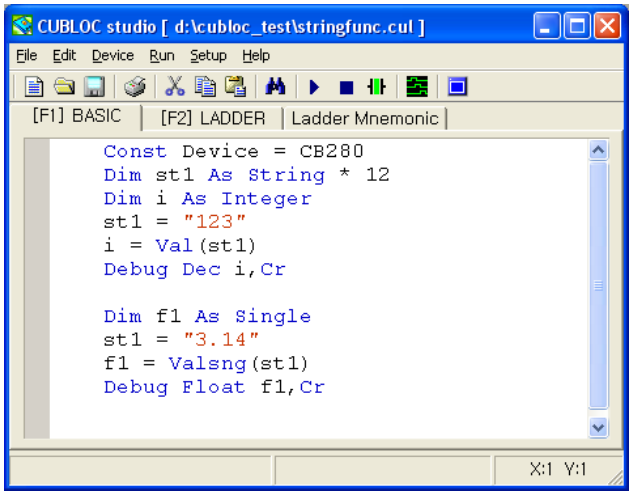
10 진수로 되어 있는 문자열의 내용을 정수형 숫자로 바꾸어 반환합니다.

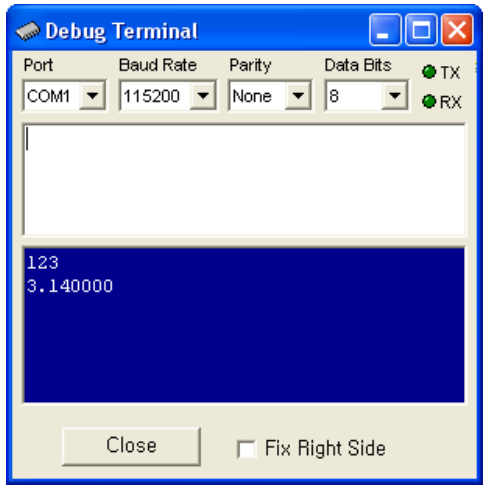
```
DIM ST1 AS STRING * 12
DIM I AS INTEGER
ST1 = “123”
I = VAL(ST1) ‘변수 I 에는 123 이 저장됩니다.
```

VALSNG(문자열변수)

소수점이 포함된 실수형으로 되어 있는 문자열의 내용을 실수형 숫자로 바꾸어 반환합니다.

```
DIM ST1 AS STRING * 12
DIM F AS SINGLE
ST1 = “3.14”
F = VALSNG(ST1) ‘변수 F 에는 3.14 가 저장됩니다.
```





VALHEX(문자열변수)

16 진수로 되어 있는 문자열의 내용을 정수형 숫자로 바꾸어 반환합니다.

```
DIM ST1 AS STRING * 12
DIM I AS LONG
ST1 = “ABCD123”
I = VALHEX(ST1) ‘변수 I 에는 &HABCD123 이 저장됩니다.
```

CHR(아스키코드)

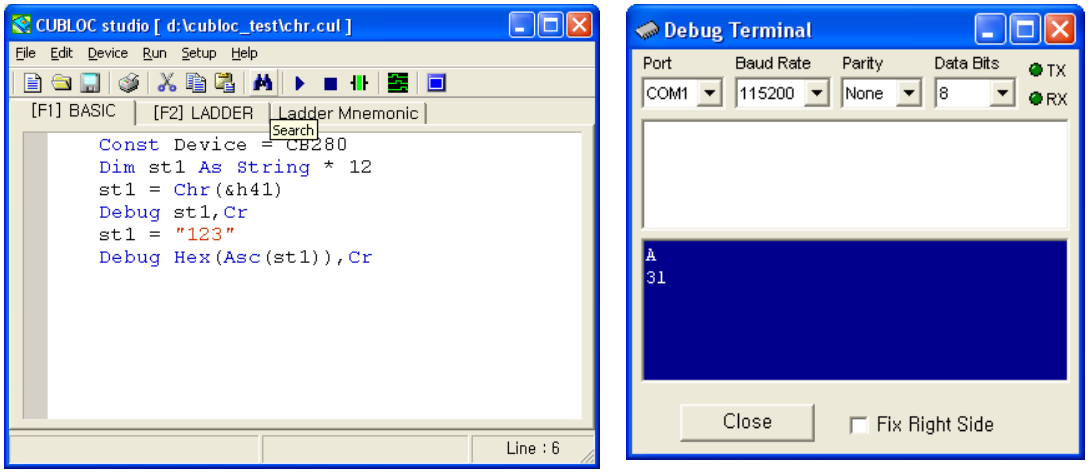
아스키코드에 해당하는 문자를 반환합니다.

```
DIM ST1 AS STRING * 12
ST1 = CHR(&H41)
DEBUG ST1   ' A 가 표시됩니다. 문자 A 의 아스키코드는 &H41 입니다.
```

ASC(문자열변수/상수)

문자열중 가장 처음에 위치한 문자의 아스키코드 값을 반환합니다.

```
DIM ST1 AS STRING * 12
DIM I AS INTEGER
ST1 = "123"
I = ASC(ST1) '변수 I 에는 &H31 이 저장됩니다. 문자 1 의 아스키코드는 &H31 입니다.
```



각 캐릭터에 대한 아스키 코드 표는 "부록"편에 수록해 놓았습니다.

* 화면상의 DEBUG 창에 내용표시가 잘 안된다면 소스 맨앞줄에 WAIT 500 이라는 명령어를 삽입해주세요.

12. 전처리기

CUBLOC BASIC 에서는 C 언어에서 사용 가능한 전처리기(Preprocessor)를 사용할 수 있습니다. #DEFINE 을 사용한 문자열 치환, #INCLUDE 를 사용한 파일삽입 등이 가능합니다.

#include " filename"

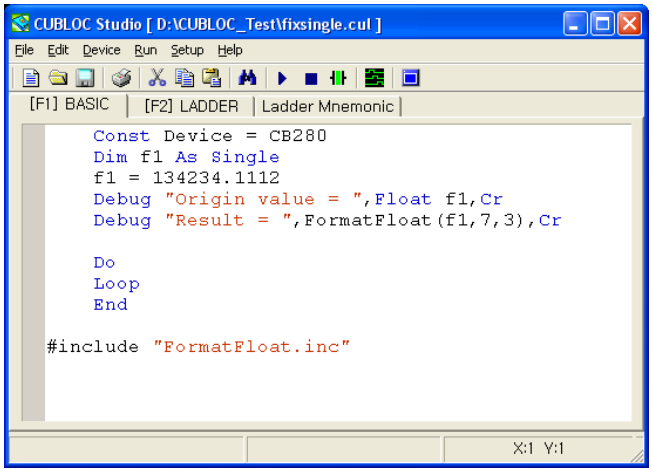
파일을 삽입합니다. 삽입된 파일은 반드시 베이직 소스파일이어야 합니다. 소스가 들어있는 폴더에 같이 들어있는 경우에는 아래와 같이 파일명만 적어줍니다.

```
#INCLUDE "MYLIB.cub"
```

만약, 다른 폴더에 들어있다면, Full Path name 을 적어주어야 합니다. 파일을 찾지 못하거나, 파일을 오픈할 수 없는 상태이면 에러가 발생됩니다.

```
#INCLUDE "c:\mysource\CUBLOC\lib\mylib.cub"
```

SUB, FUNCTION 과 같은 "부 프로그램"을 별도의 파일로 저장하여, INCLUDE 하는 경우, 반드시 END 문 뒤에 INCLUDE 를 써주어야 합니다.



"실수형 문자의 Format 표시" 함수를 FormatFloat.inc 라는 파일로 저장해 놓고, 그 위치에 #include 문을 사용한 경우입니다.

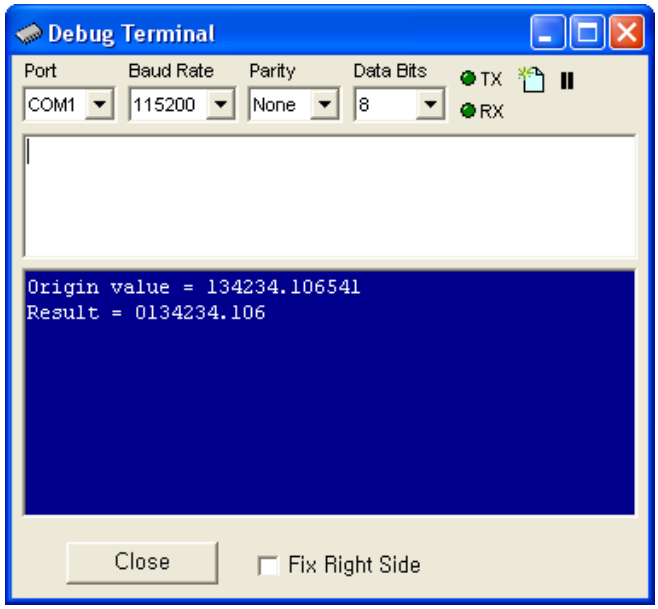
이미 완성된 부 프로그램 (더 이상 고칠 필요가 없다고 판단되는 ...)은 따로 파일로 만들어 놓은 뒤 #include 문을 사용하면, 소스 프로그램을 좀더 깔끔하게 정리할 수 있게 됩니다.

FormatFloat.Inc 파일의 내용은 다음과 같습니다.

```
Function FormatFloat(Tval As Single,Jdigit As Byte, Sdigit As Byte) As String * 30
        #define FrontChr &h30
        Dim i As Byte
        Dim Tempst As String * 30
        Dim Res As String * 30
        Tempst = Float Tval
        For i=0 To Len(Tempst)-1
              If tempst a(i)=&h2e Then Exit For
        Next
```

```
        Res = String(FrontChr,29-i)+Left(Tempst,i)
        Res = Right(Res,Jdigit)+"."
        Tempst = Mid(Tempst,i+2,Len(Tempst)-i-1)
        Res = Res + Left(Tempst,sdigit)
        FormatFloat = Res
End Function
```

결과는 다음과 같습니다.



#define name constants

컴파일 하기 전에 문자열을 치환해주는 전처리기 입니다. 언뜻 보면 CONST 상수 정의 명령과 유사해 보이지만, #define 은 문자열 자체를 치환해주기 때문에 좀더 광범위하게 응용할 수 있습니다.

```
#define motorport 4
low motorport
```

위의 예에서 컴파일 할 때, motorport 는 4 로 치환됩니다. 단순히 상수에 이름을 붙이는 용도라면 CONST 명령으로도 할 수 있습니다.

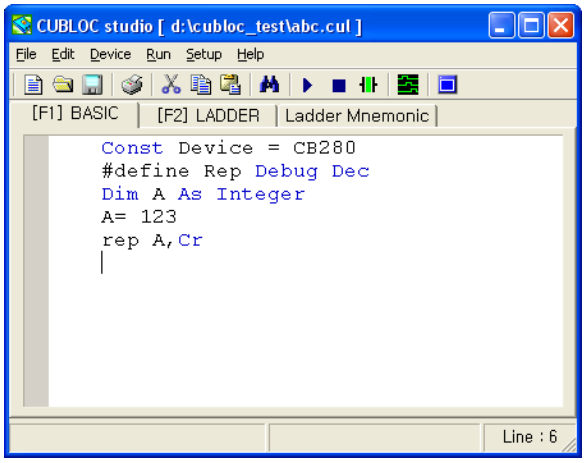
```
CONST motorport = 4
low motorport
```

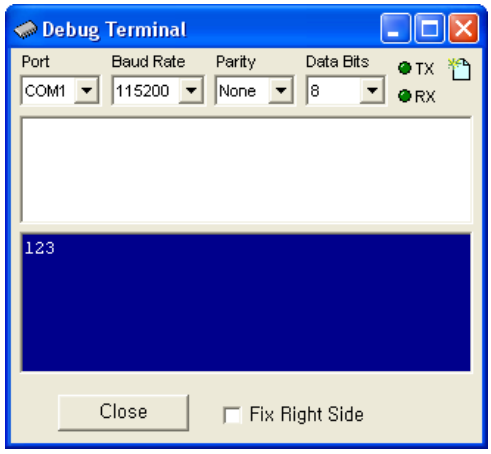
다음 예와 같이 상수 하나가 아닌 특정한 문자열 자체를 치환하고자 할 때 사용할 수 있습니다.

```
#define FLAGREG1 2
#define f_led FLAGREG1.BIT0
#define susik (4+i)*256
f_led = 1                          ' FLAGREG1 의 비트 0 을 1 로 만듭니다.
IF f_led = 1 then f_led = 0      '비트조작을 좀더 읽기 편하게 할 수 있습니다.
j = susik                        '수식도 치환하여 사용할 수 있습니다.
```

NOTE

#define 에서도 다른 BASIC 명령과 마찬가지로 대소문자를 구분하지 않고, 모든 문자를 대문자로 변환하여 처리합니다. 즉 #define ALPHA 0 와 #define alpha 0 는 같은 동작을 수행합니다.





Debug 명령으로 10 진수를 자주 표시한다면, 다음과 같이 새로운 명령을 정의할 수도 있습니다. Rep 라는 명령을 쓰면 Debug Dec 로 바꾸어주기 때문에, 타이핑 하는 수고를 덜 수도 있습니다. #define 명령이 컴파일 이전에 소스전체에서 Rep 라는 단어를 Debug Dec 로 바꾸어 놓기 때문입니다.

13. 조건 컴파일

조건 컴파일이란 말 그대로 어떤 조건에 따라 컴파일을 하거나, 하지 않을 수 있는 기능입니다. #if, #ifdef 등의 명령을 사용하여 소스 중 일부분의 컴파일 여부를 결정할 수 있습니다.

#if constant
#endif

CONST 명령으로 선언한 상수를 가지고 비교합니다. 다른 변수나 #DEFINE 으로 정의한 상수는 비교대상이 아닙니다. 비교한 결과가 참이면 #if…#endif 블록 안의 내용을 컴파일 합니다. 거짓이면 컴파일을 하지 않습니다.

```
CONST MODELNO = 4
#if MODELNO = 4
    LOW 4
#endif
```

위의 예를 보면, 왜 뻔한 내용을 가지고 컴파일 여부를 결정하기 위한 #if 문을 썼는지 궁금하신 분들도 있을 것입니다. 예를 들어 하나의 소스를 가지고 여러 개의 모델에 적용하기 위한 프로그램을 작성하고자 하는 경우에, 각각의 모델마다 소스를 따로 작성하는 것 보다, 조건 컴파일을 사용하면 하나의 소스만으로도 관리가 가능한 경우가 있습니다. 모델간의 사소한 차이를 조건컴파일 명령을 사용해서 추가 또는 삭제할 수 있기 때문입니다.

#elseif 나 #else 를 사용해서 보다 다양한 구조의 #if 문 블록을 구성할 수도 있습니다.

```
#if MODELNO = 0
    LOW 4
#elseif MODELNO = 1
    LOW 5
#elseif MODELNO = 2
    LOW 6
#else
    LOW 7
#endif
```

#if 문 에서 조건식은 반드시 하나만 작성해야 하며, 반드시 #if 문이 나오기 전에 CONST 문으로 정의된 상수만 사용할 수 있습니다.

#ifdef name
#endif

#if 문이 상수의 내용을 가지고 비교한다면, #ifdef 은 정의된 적이 있는 이름인지 여부를 가지고 비교하는 명령입니다. #define 이나 , 상수 명이나 변수명으로 선언한 적이 있는 이름을 적어준다면 조건결과는 참이 되며 #if…#endif 블록 안의 내용을 컴파일 합니다. 거짓이면 컴파일을 하지 않습니다.

```
#define  LOWMODEL 0
#ifdef LOWMODEL
    LOW 0
#endif
```

LOWMODEL 이라는 값이 정의된 적이 있으므로 조건식은 참이 되어 LOW 0 라는 명령이 컴파일 됩니다. #ifdef 도 #elseifdef 과 #else 를 사용하여 복문으로 구성할 수 있습니다.

```
#ifdef LOWMODEL
    LOW 0
#elseifdef HIGHMODEL
    HIGH 0
#else
    LOW 1
#endif
```

#ifndef name
#endif

#ifdef 명령과 정확히 반대작용을 하는 명령입니다. 주어진 이름이 정의된 적이 없으면 #if…#endif 블록 안의 내용을 컴파일 합니다. 정의된 적이 있다면 컴파일을 하지 않습니다.

```
#define  LOWMODEL 0
#ifndef LOWMODEL
    LOW 0
#endif
```

#ifndef 도 #elseifndef 과 #else 를 사용하여 복문으로 구성할 수 있습니다.

```
#ifndef LOWMODEL
    LOW 0
#elseifndef HIGHMODEL
    HIGH 0
#else
    LOW 1
#endif
```

다른 조건 문과 결합하여 사용하는 것도 가능합니다.

```
#if MODELNO = 0
    LOW 0
#elseifdef HIGHMODEL
    HIGH 0
#else
    LOW 1
#endif
```

14. 사용 디바이스 선언

여러 개의 CUBLOC 모델 중 어떤 디바이스를 사용하는지 컴파일러에게 알려주어야 합니다. 이를 위해서 CONST DEVICE 명령을 사용합니다.

```
CONST DEVICE = CB220   ' CB220 모델을 사용합니다.
```

이 명령은 반드시 소스프로그램의 가장 첫 번째 줄에 위치해야 합니다. 이 명령을 생략했을 때에는 초기값인 "CB220"모델이 선택됩니다.

```
CONST DEVICE = CB220         ' CB220 모델을 사용합니다.
CONST DEVICE = CB280         ' CB280 모델을 사용합니다.
```

#if 문에서 디바이스 종류 판별

조건 컴파일 명령인 #if 에서 디바이스의 종류를 판별하고 싶다면 다음과 같이 사용합니다.

```
CONST DEVICE = CB220
#If device = CB280
  Out 20,0
#else
  Out 20,1
#endif
```

위에서 Const Device=CB220 이라는 문장으로 device 라는 상수 명에 CB220 이라는 값을 할당한 것입니다. #if 에서 device 상수에 어떤 값이 있는 지를 체크해 볼 수 있습니다. device 상수 명은 예외적으로 문자열을 저장할 수 있습니다.

TIPS

베이직 소스중 어떤 한 블록을 모두 임시로 실행 금지시키고 싶다면, #IF 0...#ENDIF 를 사용한뒤 다운로드하시면 됩니다. 맨앞에 어포스트로피 (')를 써서 일일이 막아주는 방법보다 간편합니다.

```
Const Device = CT1720
Do
    Wait 200
    Stepaccel 0,8,10,4000,4400,30000
#If 0
    Do While Stepstat(0) > 0          ' 이 블록을 실행하시 않습니다.
    Loop                              ' 이 블록을 실행하시 않습니다.

#Endif
    Wait 1500
Loop
```

제 6 장
CUBLOC BASIC
흐름제어 명령문

IF..THEN..ELSEIF…ELSE..ENDIF

조건식의 참,거짓 여부에 따라 어떤 명령문을 수행할 것인지가 결정됩니다.

```
If  조건 1 Then [명령문블록 1]
      [명령문블록 2]
[Elseif 조건 2 Then
      [명령문블록 3]]
[Else
      [명령문블록 4]]
[End If]
```

유형 1

```
If  A<10  Then  B=1
```

유형 2

```
If  A<10  Then  B=1  Else  C=1
```

유형 3

```
If  A<10  Then
B=1
End If
```

'*복문구조 if 문일 경우에는 반드시,
'*Then 뒤에 명령을 기술 하지 않아야 합니다.

유형 4

```
If  A<10  Then
      B=1
Else
      C=1
End If
```

유형 5

```
If  A<10  Then
      B=1
Elseif A<20 Then
      C=1
End If
```

유형 6

```
If  A<10  Then
      B=1
Elseif A<20 Then
      C=1
Elseif A<40 Then
      C=2
Else
      D=1
End If
```

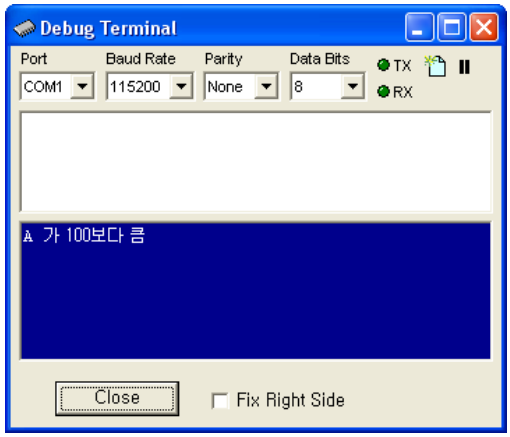
- 주의사항 : ELSEIF 에서 ELSE 와 IF 는 붙여서 작성하세요.
- EndIF 를 사용할 때, End 와 If 사이에 공백이 있어도 되고, 공백 없이 붙여서 사용해도 됩니다.
- 같은 변수 형끼리 비교해야 합니다. 문자열변수와 정수형 변수를 서로 비교할 수 없습니다.

DEMO PROGRAM

다음은 IF 문을 사용한 샘플 프로그램입니다.

```
Const Device = CB280
Dim A As Integer
A = 1000
If A<100 Then
      Debug "A 가 100 보다 작음",Cr
Else
      Debug "A 가 100 보다 큼",Cr
Endif
```

결과는 Debug 터미널에 다음과 같이 표시됩니다.



다음은 잘못된 IF 문 입니다.

```
IF A=0 THEN B=0
C=0
END IF
```

Then 뒤에 명령이 있습니다. 이 경우에는 복문 IF 문으로 번역되지 않습니다.

```
IF A=0 THEN
  IF B=0 THEN
    C=0
END IF
```

IF 문 블록 안에 다른 IF 문 블록이 있으므로, END IF 를 한번 더 써주어야 합니다.

```
Dim A as INTEGER
Dim B as String
IF A = B THEN A = 0
```

문자열은 문자열끼리만 비교할 수 있습니다.

SELECT..CASE

조건에 따라 여러 명령문 블록 중에서, 하나의 명령블록을 실행하는 명령입니다.

```
Select Case 수식
      [Case 값 [,값],...
            [명령문블록 1]]
      [Case 값 [,값],...
            [명령문블록 2]]
      [Case Else
            [명령문블록 3]]
End Select
```

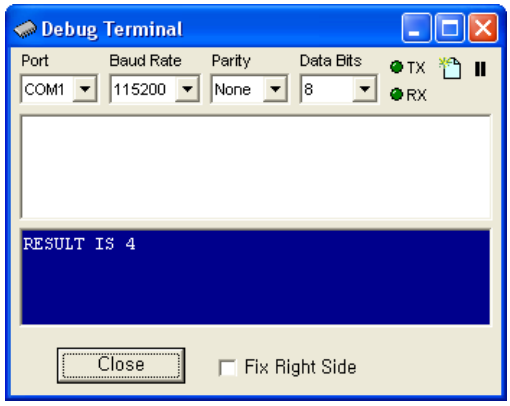
```
  Select Case A
      Case 1
            B = 0
      Case 2
            B = 2
      Case 3,4,5,6      '값이 여러 개일 때에는 콤마(,)로 분리합니다.
            B = 3
      Case Is < 1       '부등호를 사용할 때에는 IS 예약어를 사용합니다.
            B = 3
      Case Else         '그 외의 경우에는 ELSE 를 사용합니다.
            B = 4
  End Select
```

```
      Dim K As Integer
      Dim R As Integer
      K=Adin(0)         ' 0 번 채널에서 AD 입력
      Select Case K
      Case  Is < 10     '10 보다 작은 값일 경우
            R = 0
      Case Is < 40      '40 보다 작은 값일 경우
            R = 1
      Case Is < 80
            R = 2
      Case Is < 100
            R = 3
      Case Else
            R = 4
      Next K
```

DEMO PROGRAM

다음은 SELECT..CASE 문을 사용한 샘플 프로그램입니다.

```
Const Device = CB280
Dim A As Integer
A = 4
Select Case A
Case 0
      Debug "RESULT IS 0",Cr
Case 1
      Debug "RESULT IS 1",Cr
Case 2
      Debug "RESULT IS 2",Cr
Case 3
      Debug "RESULT IS 3",Cr
Case 4
      Debug "RESULT IS 4",Cr
Case 5
      Debug "RESULT IS 5",Cr
Case 6
      Debug "RESULT IS 6",Cr
End Select
```



DO..LOOP

DO..LOOP 문은 조건에 따라서 명령문을 계속 실행할 것인지의 여부를 결정합니다. DO WHILE 구조와 DO UNTIL 구조가 있으며, 빠져나올 때에는 EXIT DO 를 사용합니다.

```
Do
      명령문
Loop
```

```
Dim K As Integer
Do
    K=Adin(0) ' 0 번 채널에서 AD 입력
    Debug Dec K,Cr
    Delay 1000
Loop
```

위와 같이 조건 없이 DO..LOOP 문을 사용하면 무한루프가 됩니다. 무한루프를 탈출하기 위해서는 EXIT DO 또는 GOTO 명령을 사용합니다.

```
Do While [조건]
      명령문
      [Exit Do]
Loop

Do
      명령문
      [Exit Do]
Loop While [조건]
```

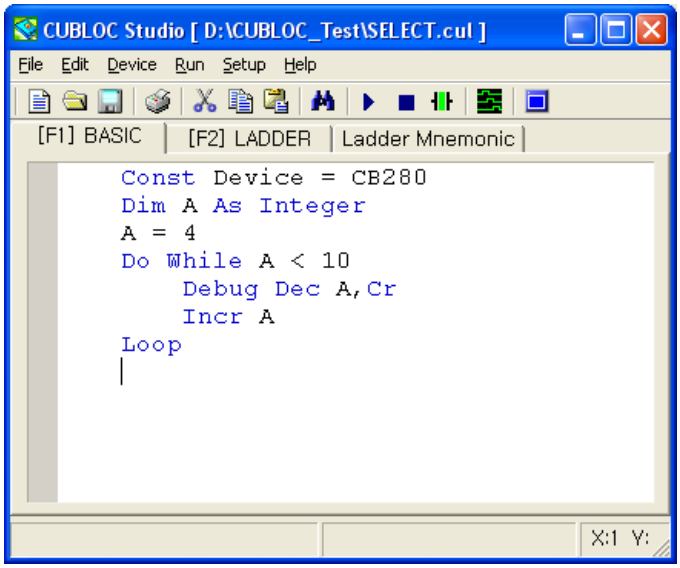
DO..WHILE 구조는 조건이 참인 동안 계속해서 명령문을 반복 실행합니다.

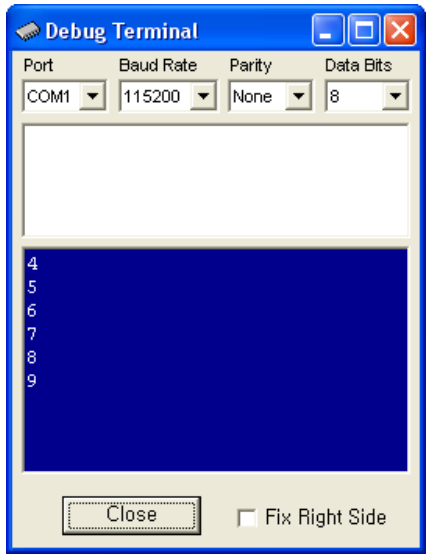
```
Do Until [조건]
      명령문
      [Exit Do]
Loop

Do
      명령문
      [Exit Do]
Loop Until [조건]
```

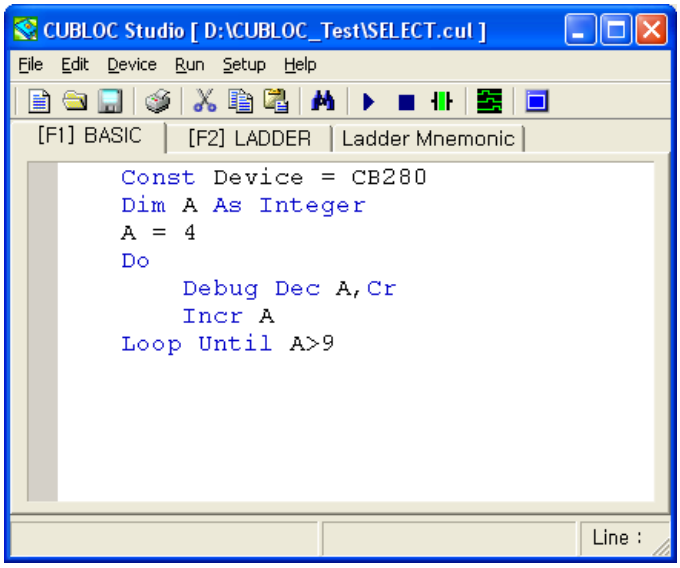
DO..UNTIL 구조는 조건이 거짓인 동안 계속해서 명령문을 반복실행 합니다.

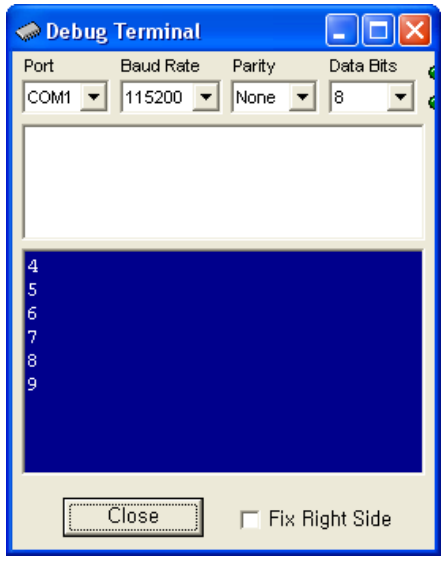
DEMO PROGRAM

다음은 DO..LOOP 문을 사용한 샘플 프로그램입니다.









FOR..NEXT

FOR..NEXT 문은 지정된 횟수만큼 명령문을 실행하는 명령어입니다.

```
For 카운터=시작 값 To 끝 값 [Step 증가분]
      명령문
      [Exit For]
Next
```

다음 예와 같이 STEP 명령이 생략된 경우에는 1 씩 증가합니다. STEP 명령을 음수 값으로 지정하는 경우에는 시작 값이 끝 값보다 커야 합니다.

```
Dim K As Long
For K=0 To 10
      Debug Dp(K),CR
Next

For K=10 To 0 Step -1        ' STEP 을 음수로 하면 감소합니다.
      Debug Dp(K),CR
Next
```

FOR 문을 수행하는 도중 탈출하고 싶다면, EXIT FOR 문을 사용합니다.

```
For K=0 To 10
      Debug Dp(K),CR
      If K=8 Then Exit For   ' EXIT FOR 를 만나면 FOR LOOP 를 탈출합니다.
Next
```

***NEXT 뒤에는 아무것도 적어주지 않습니다.**

FOR..NEXT 사용시 주의사항

FOR 문의 카운터 변수는 카운터 할 값의 범위를 충분히 커버할 수 있는 변수 형으로 선언해야 합니다. 예를 들어 255 까지 카운트 하는 경우에도 256 까지 카운트할 수 있는 INTEGER 형을 사용해야 합니다. 내부적으로 K 에 256 을 저장한 뒤 비교하기 때문입니다.

```
Dim K As Byte
For K=0 To 255
      Debug Dp(K),CR   'K 를 Byte 형으로 했을 경우 무한루프가 됩니다.
Next
```

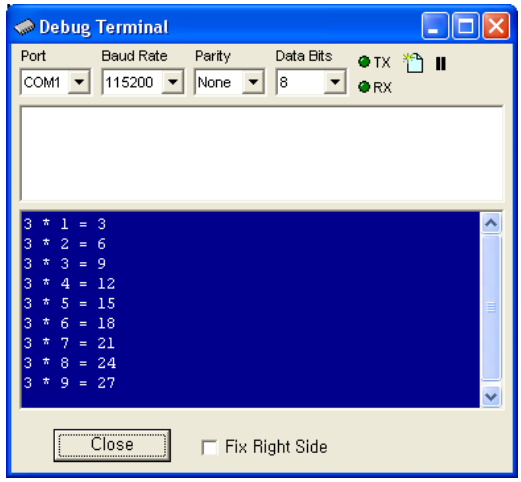
STEP -1 을 사용해서 스텝 다운하는 경우에도 음수 값을 저장할 수 있는 LONG 형 변수를 사용해야 합니다.

```
Dim LK As Long
For LK=255 To 0 Step -1      'LK 에 -1 를 저장한 뒤 비교하기 때문에 Long 형 사용
      Debug Dp(LK),CR
Next
```

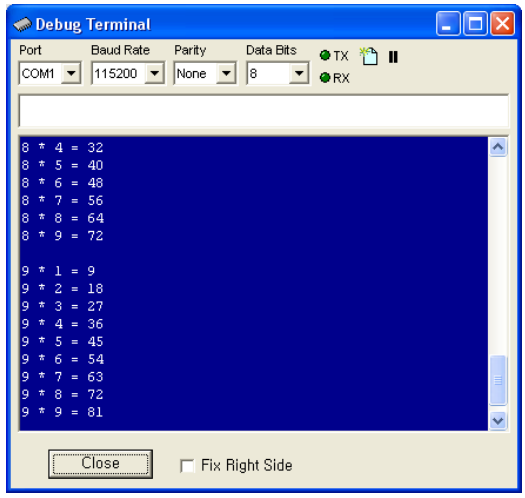
DEMO PROGRAM

다음은 FOR..NEXT 문을 사용한 샘플 프로그램입니다.

```
Const Device = CB280
Dim A As Integer
For A=1 To 9
      Debug "3 * "
      Debug Dec A
      Debug " = "
      Debug Dec 3*A,Cr
Next
```




```
Const Device = CB280
Dim A As Integer, B As Integer
For A=2 To 9
      For B=1 To 9
            Debug Dec A," * "
            Debug Dec B
            Debug " = "
            Debug Dec A*B,Cr
      Next
      Debug Cr
Next
```



For..Next 문으로 구구단을 표시하는 프로그램을 만들어 본 것입니다. 두 번째 소스는 2 단부터 9 단까지 모두 표시하는 프로그램입니다. For ..Next 루프 안에 또 다른 For..Next 루프를 넣어서 구현하였습니다.

GOTO

GOTO 문은 지정된 라벨로 무조건 분기하는 명령입니다. 프로그램을 구조적으로 작성하는 것을 방해하므로 잘 사용하지 않는 명령어입니다.

```
        If I = 2 Then
             Goto LAB1
        End If
LAB1:
        I = 3
```

GOSUB..RETURN

간단하게 서브루틴을 호출할 수 있는 명령입니다. GOSUB 로 점프 뒤 RETURN 명령을 만나면 호출했던 지점으로 복귀합니다.

```
        GOSUB ADD_VALUE
        :
ADD_VALUE:
        A=A+1
        RETURN
```

Label 명

GOTO, GOSUB 에서 점프할 곳을 가리키기 위해 Label 을 사용합니다. Label 은 아래와 같이 뒤에 콜론을 붙여서 행의 가장 왼쪽에 작성합니다.

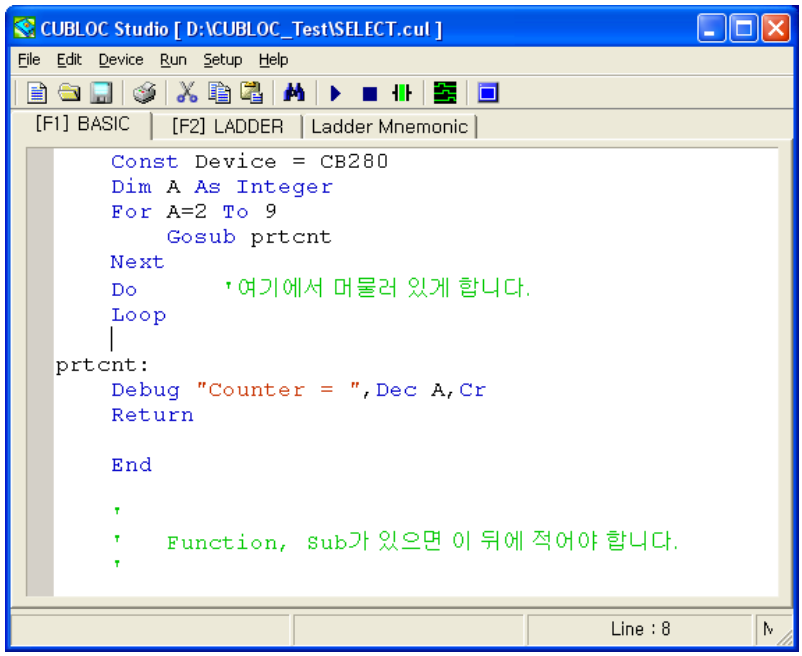
```
ADD_VALUE:
LINKPOINT:
```

Label 명은 예약어나 변수명으로 사용했던 이름을 사용할 수 없으며, 반드시 영문자(언더라인 포함)로 시작하는 255 자 이내의 문자로 작성해야 합니다. 16 자 내외가 적당하며, 한글을 라벨 명으로 사용할 수 있습니다. 다음은 잘못 사용된 Label 명 예입니다.

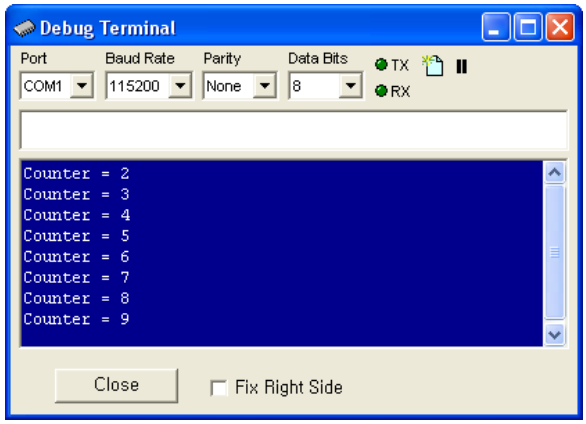
```
Ladder:          ‘예약어입니다.
123:             ‘숫자로 시작되었습니다.
Aboot 10:        ‘공백이 포함되어 있습니다.
```

DEMO PROGRAM

다음은 GOSUB, RETURN 문을 사용한 샘플 프로그램입니다.



Function, Sub 부 프로그램보다 간단한 형태의 서브루틴을 사용할 때 GOSUB 문을 사용합니다. Gosub 문을 사용할 때, Return 문 사용에 주의해야 합니다. Gosub 한 뒤 서브루틴 안에서 Return 문을 만나면, 호출했던 곳으로 되돌아가지만, Gosub 없이 Return 문을 만나면 "리셋"이 걸립니다. 스택오버플로우가 발생되었기 때문입니다. 따라서 프로그램 흐름상 Gosub 없는 Return 문이 발생하지 않도록 주의해야 합니다. "스택오버 플로우"는 실행 중 발생하는 논리적인 에러상황이므로, 컴파일 /다운로드 시 별도의 에러멧세지를 발생시킬 수 없습니다. 유저의 각별한 주의가 필요한 부분입니다.



MEMO

제 7 장
CUBLOC BASIC 라이브러리

CUBLOC BASIC 라이브러리 요약

디지털 입출력

INPUT	포트를 HIGH-Z 입력상태로 만듭니다.
OUTPUT	포트를 출력으로 설정합니다.
HIGH	포트를 HIGH 로 만듭니다.
LOW	포트를 LOW 로 만듭니다.
IN()	포트의 상태를 읽어옵니다.
BYTEIN()	포트 8 개를 한꺼번에 읽어옵니다.
OUT	포트의 상태를 HIGH 또는 LOW 로 바꿉니다.
BYTEOUT	포트 8 개를 한꺼번에 바꿉니다.
OUTSTAT()	포트가 출력중인 값을 읽어옵니다.
REVERSE	현재 출력 중인 값을 반전시킵니다.

아날로그 입출력

ADIN()	AD 변환 값을 읽어옵니다.
EADIN( )	OPAMP 를 사용한 AD 변환 값을 읽어옵니다.
PWM	PWM 파형을 출력합니다.
PWMOFF	PWM 파형 출력을 중단합니다.
FREQOUT	일정한 주파수의 파형을 출력합니다.

RS232C 명령

OPENCOM	RS232C 채널 오픈명령
GET	RS232C 수신 명령
GETSTR	문자열 수신 명령
GETA	1 차원 배열에 수신하는 명령
PUT	RS232C 송신 명령
PUTSTR	문자열 송신명령
PUTA	1 차원 배열의 내용을 송신하는 명령
SET UNTIL	UNTIL 조건을 선언합니다.
BCLR	RS232C 버퍼 초기화
BLEN ( )	RS232C 버퍼에 수신된 데이터 개수 리턴
BFREE( )	버퍼에 남아있는 공간의 크기를 리턴
WAITTX	송신버퍼가 비워질 때까지 대기

EEPROM

EEWRITE	EEPROM 의 특정 위치에 데이터를 기록합니다.
EEREAD	EEPROM 의 특정 위치에서 데이터를 읽어옵니다.

인터럽트

ON TIMER	주기적으로 특정 루틴을 수행합니다.
ON INT	외부 핀에서 에지 발생시 특정 루틴을 수행합니다.
ON RECV	RS232C 수신 인터럽트 발생시 특정루틴을 수행.
ON PAD	PAD 입력 시 특정루틴을 수행합니다.
ON LADDERINT	LADDER 로부터 인터럽트 발생

RTC 관련

TIME( )	RTC 로부터 시간정보를 읽어옵니다.
TIMESET	RTC 에 새로운 시간정보를 기입합니다.

기타

RND( )	난수를 발생시킵니다.
SYS( )	RS232 송수신 상태 등 (시스템 변수 관련)를 리턴합니다.
SET COUNT0	카운터채널 0 을 활성화시킵니다.
COUNT( )	카운터 값을 읽어옵니다.
COUNTRESET	카운터를 리셋 시킵니다.
SET PAD	PAD 입력을 선언합니다.
RAMCLEAR	RAM 을 모두 0 으로 클리어 합니다.
GETPAD()	PAD 입력을 받습니다.
PEEK()	데이터 메모리의 특정주소에서 직접 읽어옵니다.
POKE	데이터 메모리의 특정주소에 직접 기입합니다.
MEMADR()	특정변수의 주소 값을 반환합니다.
RESET	소프트 리셋

SPI, I2C 통신 관련

SHIFTIN	SPI 수신 명령
SHIFTOUT	SPI 송신 명령
I2CSTART	I2C START
I2CSTOP	I2C STOP
I2CWRITE	I2C WRITE 명령
I2CREAD	I2C READ 명령
I2CREADNA	ACK 발생없는 I2C READ

LADDER 관련

SET LADDER	LADDER 의 실행여부를 결정합니다.
USEPIN	LADDER 에서 사용할 I/O 핀의 상태를 결정합니다.
ALIAS	LADDER 에서 사용할 별명을 선언합니다.
LADDERSCAN	LADDER 를 강제로 한 스캔 실행합니다.

시스템 부 프로그램 라이브러리

DELAY	시간을 지연시킵니다.
UDELAY	작은 시간단위로 지연시킵니다.
PAUSE	DELAY 와 같은 기능입니다. 시간을 지연시킵니다.
KEYIN ( )	채터링이 제거된 키 입력을 받습니다. LOW-ACTIVE 입력
KEYINH( )	채터링이 제거된 키 입력을 받습니다. HIGH-ACTIVE 입력
BEEP	특정포트에서 키 터치 음을 발생합니다.
PULSE	일정시간 동안 펄스를 출력합니다.
TADIN ( )	평균값 AD 입력 명령입니다. 여러 번 읽어 평균을 구합니다.
KEYPAD	최대 4x4 의 키 매트릭스로부터 상태를 읽어옵니다.
EKEYPAD	최대 8x8 의 키 매트릭스로부터 상태를 읽어옵니다.
BCD2BIN	BCD 코드를 바이너리로 변환합니다.
BIN2BCD	바이너리를 BCD 코드로 변환합니다.

MENU 관련

MENUSET	메뉴의 기본정보를 셋팅합니다. (X,Y 좌표)
MENUTITLE	메뉴의 제목을 표시합니다.
MENUCHECK( )	주어진 좌표가 메뉴 안의 영역에 포함되는지 검사합니다.
MENUREVERSE	메뉴영역을 반전시킵니다.

주의사항

위 요약에는 모든 명령어가 표시되어 있지 않습니다. 대략 어떤 종류의 라이브러리가 제공되고 있는지만 확인하시기 바랍니다.

향후, CUBLOC BASIC 의 명령어 또는 라이브러리는 제품의 기능향상을 위하여 추가될 수 있습니다. 본 사용설명서에서 다루는 내용은 현재 버전을 기준으로 작성된 것입니다. 추후 발표될 다른 시리즈에서는 일부 명령어의 사용법에 약간의 차이가 있을 수 있습니다.

라이브러리의 종류

CUBLOC BASIC 에서 사용하는 명령은 IF, THEN, GOTO 와 같은 기본 명령어 이외에, OUT, IN 과 같은 라이브러리 명령어가 있습니다.

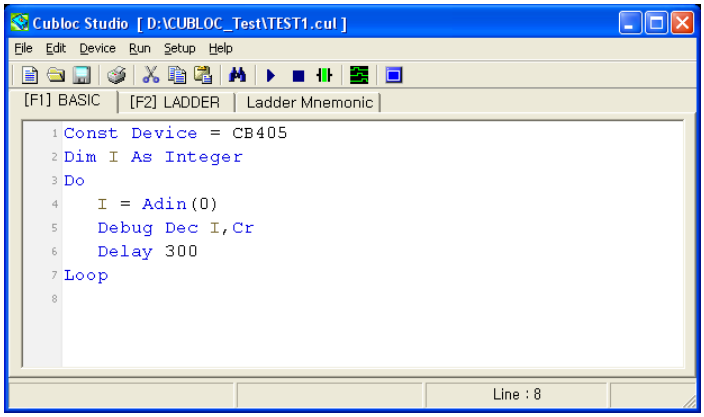
라이브러리에는 “명령문”형식과 “함수”형식이 있습니다.

명령문	: INPUT, OUTPUT
함수	: IN(), ADIN()

뒷부분에 괄호가 있으면, 수식의 일부로 사용할 수 있는 “함수”입니다. 뒷부분에 괄호가 없으면 수식에서 사용할 수 없는 “명령문”입니다.

```
INPUT 4          ‘ 4 번 포트를 입력으로 만듭니다.
A = ADIN(0)      ‘ 0 번 채널에서 AD 입력을 수행합니다.
```

에디터상에서 라이브러리와 예약어는 파란색으로 표시됩니다.



디지털 입출력

Input

INPUT Pin

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

특정 핀을 HIGH-Z (하이임피던스) 입력상태로 설정합니다. CUBLOC 모듈의 모든 I/O 핀은 파워 온 시 HIGH-Z 입력상태가 됩니다. INPUT 또는 OUTPUT 명령을 사용해서 I/O 핀의 입출력상태를 조정할 수 있습니다. INPUT 명령에 의해 해당포트는 HIGH-Z 상태가 됩니다. 하이 임피던스란 저항이 매우 높다는 뜻으로, HIGH 도 LOW 도 아닌 상태를 말합니다. 이 명령어는 출력전용 핀에는 사용할 수 없습니다.

```
INPUT 8      '8 번 포트를 HIGH-Z 입력상태로 만듭니다.
```

출력전용 핀은 Set Outonly On 명령어를 실행하기 전까지는 HIGH-Z 상태였다가, 명령실행후 Output 상태가 됩니다. 이때 미리 초기화해두지 않는다면 쓰레기값이 출력될수 있으므로, 사전에 초기화한뒤 Set Outonly On 을 실행하시기 바랍니다.

Output

OUTPUT Pin

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수 (모델 별로 차이가 있음)

특정 핀을 출력상태로 설정합니다. CUBLOC 모듈의 모든 I/O 핀은 파워 온 시 입력상태가 됩니다. INPUT 또는 OUTPUT 명령을 사용해서 I/O 핀의 입출력상태를 조정할 수 있습니다. 이 명령어는 입력전용 핀에는 사용할 수 없습니다.

```
OUTPUT 8     '8 번 포트를 출력상태로 만듭니다.
```

HIGH, LOW 명령을 써도 포트는 출력상태가 됩니다. OUTPUT 명령을 사용하면 HIGH, 또는 LOW 상태 중 어떤 상태로 될지 불확실하므로, OUTPUT 명령대신 HIGH, 또는 LOW 명령을 써서 출력상태로 만드는 것이 좋습니다.

```
LOW 8        '8 번 포트를 출력상태로 만들고 LOW 를 출력합니다.
```

사용하지 않는 포트는 입력상태로 두고, Gnd 와 연결하시거나, 출력 LOW 상태로 두어야 노이즈 유입을 방지할 수 있습니다.

In( )

Variable = IN(Pin)

Variable : 결과가 저장될 변수
Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

특정 핀의 상태를 읽어옵니다. CUBLOC 모듈의 I/O 핀 입력상태를 읽어서 지정한 변수에 저장하는 함수입니다. 이 명령을 실행하면 해당 포트를 자동적으로 입력상태로 만든 뒤 핀 상태를 읽어옵니다. 따라서 별도로 Input 명령을 사용해서 입력포트로 설정해 줄 필요가 없습니다. 이 명령은 출력전용핀에는 사용할 수 없습니다.

```
DIM A AS BYTE
A = IN(8)   '8 번 포트의 입력상태를 읽어서 변수 A 에 저장합니다.(0 또는 1 로 저장)
```

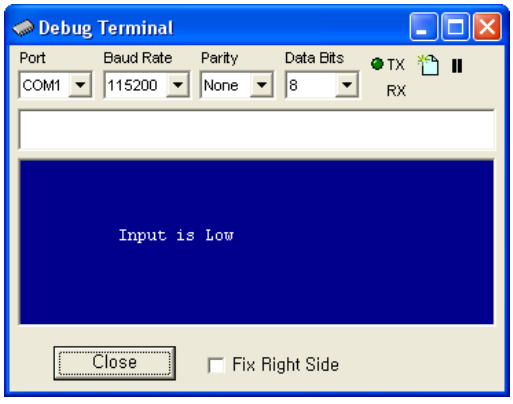
TIPS

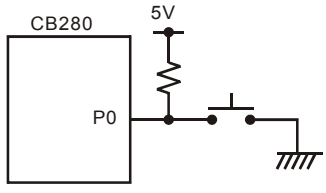
CUBLOC 의 대부분 I/O 포트는 양방향성입니다. 입력 또는 출력으로 선택할 수 있으며, 어떤 명령을 사용하느냐에 따라 입력 또는 출력 핀으로 설정할 수 있습니다. 전원 ON 시에 모든 I/O 포트는 HIGH-Z 입력상태가 됩니다.

DEMO PROGRAM

다음 샘플 프로그램을 실행시키면, P0 에 연결된 스위치 입력상태에 따라 Debug 터미널에 표시되는 내용이 변합니다.

```
Const Device = CB280
Dim A As Integer
Do
      A=In(0)
      Debug Goxy,10,3
      If A = 0 Then
            Debug "Input is Low "
      Else
            Debug "Input is high"
      Endif
Loop
```





Out

OUT Pin, Value

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

Value : I/O 핀에 출력할 값을 가지고 있는 변수 또는 상수

특정 핀을 출력으로 변경합니다. I/O 핀의 상태를 HIGH 또는 LOW 로 변경하는 명령입니다. 이 명령을 실행하면 해당 포트를 자동적으로 출력상태로 만듭니다. 따라서 별도로 Output 명령을 사용해서 입력포트로 설정해 줄 필요가 없습니다. 이 명령어는 입력전용핀에는 사용할 수 없습니다.

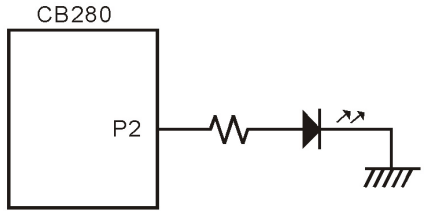
```
OUT 8,1     '8 번 포트를 HIGH 로 만듭니다. HIGH 8 과 같은 기능을 합니다.
OUT 8,0     '8 번 포트를 LOW 로 만듭니다. LOW 8 과 같은 기능을 합니다.
```

HIGH 상태에서는 5V 를 출력하고, LOW 상태에서는 0V 즉, GND 가 됩니다.

다음 샘플 프로그램을 실행시키면, P2 에 연결된 LED 가 일정한 시간간격으로 점멸합니다.

```
Const Device = CB280
Do
      Out 2,1
      Delay 100
      Out 2,0
      Delay 100
Loop
```

이 샘플 프로그램을 테스트 하기 위한, 회로도는 다음과 같습니다.



LED 와 직렬로 연결된 저항은 330 오옴이 적당합니다.

High

HIGH Pin

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

특정 핀 출력을 HIGH 상태로 만듭니다. 이 명령을 사용하여 특정 핀을 HIGH 로 만들면, 해당 포트는 OUTPUT 상태가 되고, 핀에서는 5V 가 출력됩니다. 이 명령어는 입력전용핀에는 사용할 수 없습니다.

```
OUTPUT 8    '8 번 포트를 출력상태로 만듭니다.
HIGH 8      '출력을 HIGH 로 만듭니다.
```

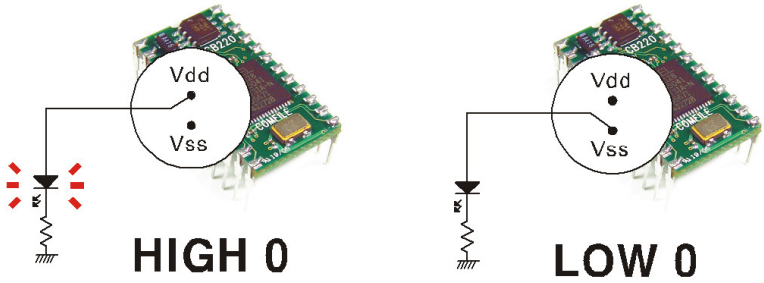
Low

LOW Pin

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

특정 핀 출력을 LOW 상태로 만듭니다. 이 명령을 사용하여 특정 핀을 LOW 로 만들면, 해당포트는 OUTPUT 상태가 되고, 해당 핀에서는 0V(GND)가 출력됩니다. 이 명령어는 입력전용핀에는 사용할 수 없습니다.

```
OUTPUT 8    '8 번 포트를 출력상태로 만듭니다.
LOW 8       '출력을 LOW 로 만듭니다.
```



다음 샘플 프로그램을 실행시키면, P2 에 연결된 LED 가 일정한 시간간격으로 점멸합니다.

```
Const Device = CB280
Do
      High 2
      Delay 100
      Low 2
      Delay 100
Loop
```

Byteout

BYTEOUT PortBlock, Value

PortBlock : 사용 가능한 포트블록을 가리키는 변수 또는 상수
Value : 포트블록에 출력할 변수 또는 상수 (0~255 사이의 값)

특정 포트블록에 원하는 값을 출력합니다. 8 개의 I/O 포트를 묶어서 하나의 포트블록이라고 합니다. 포트 0~7 번을 블록 0, 포트 8~15 를 블록 1 으로 정하고 있습니다. 모델 별로 사용 가능한 포트블록에 차이가 있습니다. BYTEOUT 명령을 사용하면 해당 포트블록은 모두 OUTPUT 상태로 설정되고, 원하는 값이 출력됩니다. 이 명령어는 입력전용핀에는 사용할 수 없습니다.

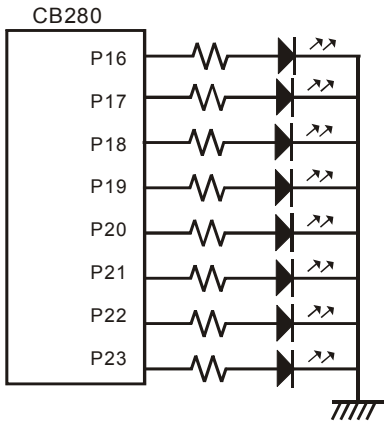
```
BYTEOUT 1,255          '1 번 포트블록에 255 를 출력합니다. 모두 HIGH 가 됩니다.
```

* 블록 0 에서 I/O 핀 1 번은 입력전용 핀입니다. 따라서 BYTEOUT 0 으로 하면, 핀 1 번은 Output 되지 않습니다.

DEMO PROGRAM

다음 샘플프로그램을 실행시키면, 포트블록 2 (P16 ~ P23) 에 연결된 LED 에 순차적으로 불이 점멸되는 것을 볼 수 있습니다.

```
Const Device = CB280
Dim A As Integer
Do
      Byteout 2,A
      Incr A
      Delay 100
Loop
```



Bytein( )

Variable = BYTEIN(PortBlock)

Variable : 결과가 저장될 변수

PortBlock : 사용 가능한 포트블록을 가리키는 변수 또는 상수

특정 포트블록에서 핀 상태를 읽어옵니다. 8 개의 I/O 포트를 묶어서 하나의 포트블록이라고 합니다. 포트 0~7 번을 블록 0, 포트 8~15 를 블록 1 으로 정하고 있습니다. 모델 별로 사용 가능한 포트블록에 차이가 있습니다. BYTEIN 함수를 사용하면 해당 포트블록은 모두 INPUT 상태로 설정됩니다. 읽어온 값은 지정된 변수에 저장됩니다. 이 명령어는 출력전용핀에는 사용할 수 없습니다.

```
DIM A AS BYTE
A = BYTEIN(0)          '0 번 포트블록에서 값을 읽어와 변수 A 에 저장합니다.
```

Outstat( )

Variable = OUTSTAT(Pin)

Variable : 결과가 저장될 변수

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

특정 포트에 출력중인 값을 읽어옵니다. 특정 I/O 포트에 출력중인 값을 읽어옵니다. 언뜻 보면 IN()함수와 비슷하지만, IN()함수는 핀의 상태를 읽어오고, Outstat()는 출력중인 상태를 읽어온다는 차이점이 있습니다. 대부분의 경우 출력중인 상태와 핀의 상태가 동일하지만, 과부하가 걸린 경우, high 를 출력 중이지만, 핀의 상태는 low 로 읽히는 경우가 생기게 됩니다. 이 경우, In()과 Outstat() 함수를 사용해서 과부하 여부를 판단할 수 있습니다. 이 명령어는 입력전용핀에는 사용할 수 없습니다.

```
DIM A AS BYTE
A = OUTSTAT(0)    '0 번 포트에 출력중인 값을 읽어서 변수 A 에 저장합니다.
```

Reverse

REVERSE Pin

Pin : 사용 가능한 I/O 핀을 가리키는 변수 또는 상수

특정 핀 출력을 반전합니다. 특정 핀에 출력중인 상태를 반대로 만듭니다. HIGH 를 출력 중이면 LOW 로 만들고, LOW 를 출력 중이면 HIGH 로 만듭니다. 이 명령어는 입력전용핀에는 사용할 수 없습니다.

```
OUTPUT 8    '8 번 포트를 출력상태로 만듭니다.
LOW 8       '출력을 LOW 로 만듭니다.
REVERSE 8   'HIGH 가 출력됩니다.
```

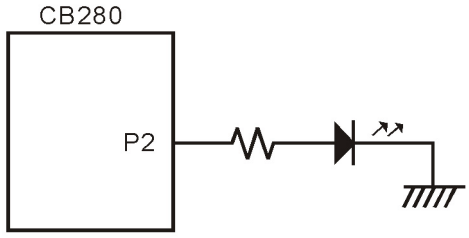
주의사항

PICBASIC 을 사용해보신 분들은 주의 깊게 보시기 바랍니다. PICBASIC 명령 중 TOGGLE 과 같은 동작을 하는 명령입니다. CUBLOC 에서는 TOGGLE 명령대신 REVERSE 명령을 사용하시기 바랍니다.

DEMO PROGRAM

다음 샘플 프로그램을 실행시키면, P2 에 연결된 LED 가 일정한 시간간격으로 점멸합니다.

```
Const Device = CB280
Low 2
Do
     Reverse 2
Loop
```



아날로그 입출력

CUBLOC 에는 10 비트 ADC 와 16 비트 PWM 을 내장하고 있습니다. 이 기능을 사용해서 외부의 아날로그 신호를 디지털수치로 변환하거나, 디지털 수치를 아날로그 신호로 변환하는 것이 가능합니다.

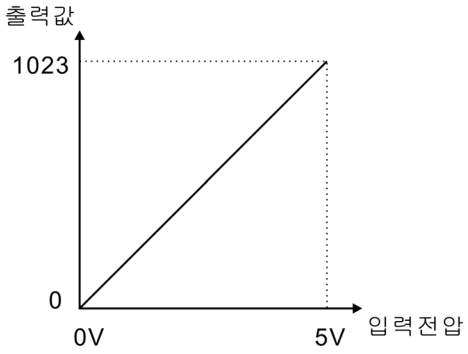
Adin( )

Variable = ADIN (Channel)

```
Variable : 결과가 저장될 변수
Channel : AD 입력 채널 (I/O 핀번호가 아님)
```

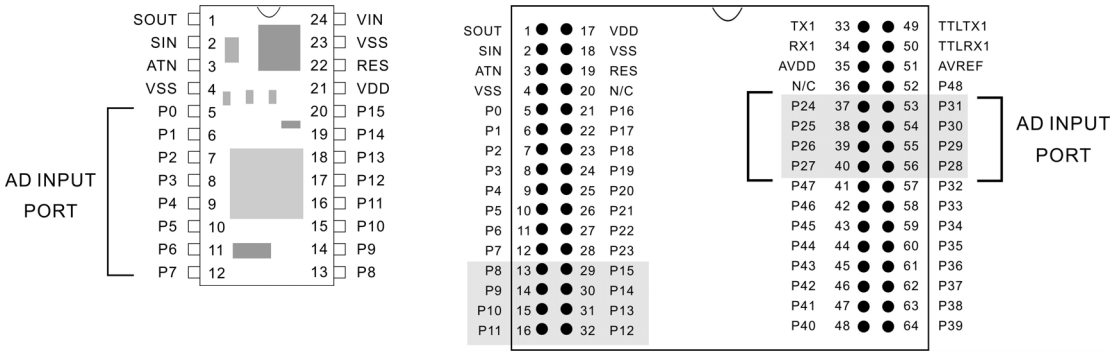
특정 포트에 입력중인 아날로그 값을 읽어서 지정한 변수에 저장합니다. CUBLOC 모델에 따라 사용할 수 있는 아날로그입력 포트가 다소 차이가 있습니다. CB280 / CB380 모델의 경우 포트 24~31 번 포트까지 8 개의 포트를 AD 입력포트로 사용할 수 있습니다. 괄호 안에는 포트번호가 아닌 AD 채널번호를 적어주고 해당 포트는 사용 전에 반드시 입력상태로 해두어야 합니다.

0~AVREF 사이의 전압을 인가하면 0~1023 사이의 값으로 변환합니다. AVREF 단자에는 2V~5V 까지 사용가능하며, 일반적으로 5V 를 연결해 줍니다. AVREF 에 3V 를 연결하면 0~3V 사이를 0~1023 으로 변환하게 됩니다. AVREF 에 5V 이상의 전압을 인가할 수는 없으므로 주의하시기 바랍니다. 일부 CUBLOC 모델의 경우 별도의 AVREF 단자가 나와있지 않은 경우가 있는데, 이 모델에서는 내부적으로 AVREF 단자가 5V 에 연결되어 있으므로, 반드시 0~5V 사이의 전압을 인가해야 합니다.




```
DIM A AS INTEGER
INPUT 24          ' 입력상태로 만든다.
A=ADIN(0)         '0 번 채널에서 A/D 변환을 해서 변수 A 에 저장
```

다음은 모델 별로 A/D 입력 가능 포트를 표시한 것입니다.



모델 별 A/D 채널과 포트와의 관계는 다음 표를 참조하시기 바랍니다.

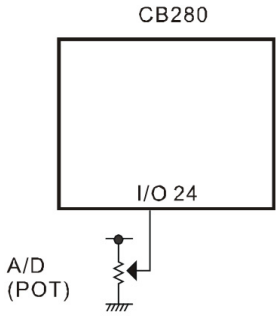
채널-모델명	CB220 CB320	CB280 CB380	CB290	CT172X/C	CB405
A/D channel 0	I/O 0	I/O 24	I/O 8	I/O 0	I/O 16
A/D channel 1	I/O 1	I/O 25	I/O 9	I/O 1	I/O 17
A/D channel 2	I/O 2	I/O 26	I/O 10	I/O 2	I/O 18
A/D channel 3	I/O 3	I/O 27	I/O 11	I/O 3	I/O 19
A/D channel 4	I/O 4	I/O 28	I/O 12	I/O 4	I/O 20
A/D channel 5	I/O 5	I/O 29	I/O 13	I/O 5	I/O 21
A/D channel 6	I/O 6	I/O 30	I/O 14	I/O 6	I/O 22
A/D channel 7	I/O 7	I/O 31	I/O 15	I/O 7	I/O 23
A/D channel 8					I/O 32
A/D channel 9					I/O 33
A/D channel 10					I/O 34
A/D channel 11					I/O 35
A/D channel 12					I/O 36
A/D channel 13					I/O 37
A/D channel 14					I/O 38
A/D channel 15					I/O 39

ADIN 명령은 포트의 상태를 단 한번만 읽어오는 명령이므로, 어느 한 순간의 값만을 측정할 수 있습니다. 이에 반해 TADIN 명령은 10 번을 읽어서, 그 평균값을 계산하여 보내주므로, 좀더 정밀한 측정이 가능합니다. 정밀한 측정을 원할 경우 ADIN 명령대신, TADIN 명령을 사용하시기 바랍니다. TADIN 명령은 "부 프로그램 라이브러리" 명령입니다.

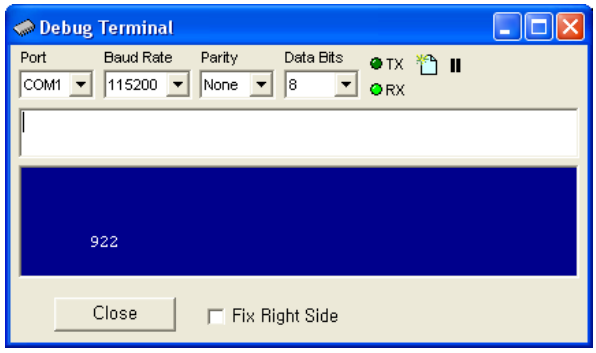
DEMO PROGRAM

다음 샘플프로그램을 실행시키고, 아래와 같은 회로를 구성하면, A/D 변환상태를 DEBUG 터미널로 확인할 수 있습니다.

```
Const Device = CB280
Dim A As Integer
Input 24
Do
      A=Adin(0)
      Debug Goxy,5,3
      Debug dec5 A
      Delay 200
Loop
```



볼륨(포텐션 미터)을 돌리면 디버그터미널에 표시된 숫자가 변하는 것을 볼 수 있습니다.

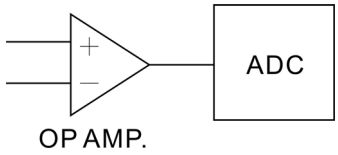


EAdin( )

Variable = EADIN (mux)

```
Variable : 결과가 저장될 변수
mux : AD 입력 핀과 조합을 결정하는 값
```

OPAMP 를 이용한 좀더 정밀한 A/D 변환을 필요로 할 때 사용하는 명령입니다. CUBLOC 는 A/D 컨버터의 앞쪽에 OPAMP 가 내장되어 있습니다. ADIN 명령을 사용할 때에는 OPAMP 를 사용하지 않고 곧바로 A/D 컨버터 입력을 받는 것입니다. EADIN 명령을 사용할 때에 비로소 OPAMP 를 사용한 입력을 받을 수 있습니다.



괄호 안에는 포트번호가 아닌 OPAMP 입력 단의 조합과 체배에 관한 내용을 써주어야 합니다. 아래 표를 보시고, 적절한 번호를 선택하여 적어주면 됩니다. 여기에서 A/D 입력 채널은 포트번호와는 다르다는 사실을 주의해야 합니다. 보통 CUBLOC 에는 8 개의 A/D 입력 채널이 있는데, 0 번부터 7 번까지의 채널이 존재하는 것입니다. A/D 채널이 어떤 포트에 연결되어 있는지는 모델마다 다소 차이가 있습니다. EADIN 명령에서 얘기하고 있는 A/D 채널 번호는 포트번호가 아닙니다. **그리고 EADIN 명령을 사용할 때에는 10 비트 해상도를 전부 지원하지 않습니다. 1 체배, 10 체배일 경우 8 비트, 200 체배일 경우 7 비트 해상도를 지원합니다.**

MUX	OPAMP 의 플러스 쪽 입력 A/D 채널	OPAMP 의 마이너스 쪽 입력 A/D 채널	체배	해상도
0	ADC0	ADC0	10	8 비트
1	ADC1	ADC0	10	8 비트
2	ADC0	ADC0	200	7 비트
3	ADC1	ADC0	200	7 비트
4	ADC2	ADC2	10	8 비트
5	ADC3	ADC2	10	8 비트
6	ADC2	ADC2	200	7 비트
7	ADC3	ADC2	200	7 비트
8	ADC0	ADC1	1	8 비트
9	ADC1	ADC1	1	8 비트
10	ADC2	ADC1	1	8 비트
11	ADC3	ADC1	1	8 비트
12	ADC4	ADC1	1	8 비트
13	ADC5	ADC1	1	8 비트
14	ADC6	ADC1	1	8 비트
15	ADC7	ADC1	1	8 비트
16	ADC0	ADC2	1	8 비트
17	ADC1	ADC2	1	8 비트
18	ADC2	ADC2	1	8 비트
19	ADC3	ADC2	1	8 비트
20	ADC4	ADC2	1	8 비트
21	ADC5	ADC2	1	8 비트

사용하고자 하는 포트는 EADIN 명령 사용 전에 입력상태로 만들어 주어야 합니다. 보통, 센서로부터 오는 Analog 량을 A/D 변환하는 경우에, OPAMP 를 사용해서 증폭하거나 노이즈를 필터링 하는 경우가 많습니다. 이런 경우 EADIN 명령을 사용하면 일부 부품을 줄이거나, 더 높은 정밀도를 가진 A/D 변환 결과를 얻을 수 있습니다.

```
DIM J AS LONG
INPUT 24              '해당 포트를 입력상태로(CB280 의 경우 24,25 포트사용)
INPUT    25
DO
    j = eadin(8)      ' 채널 0 과 1 에서 AD 변환, OPAMP 사용, 1 체배
    locate 0,0
    print hex5 J,cr   ' LCD 에 결과 표시
    delay 500         ' 약간의 딜레이
LOOP
END

SUB DELAY(DL AS INTEGER)
    DIM I AS INTEGER
    FOR I = 0 TO DL
    NEXT
END SUB
```

모델 별 A/D 채널과 포트와의 관계는 다음 표를 참조하시기 바랍니다.

채널-모델명	CB220 CB320	CB280 CB380	CB290	CT17X0	CB405
ADC0	I/O 0	I/O 24	I/O 8	I/O 0	I/O 16
ADC1	I/O 1	I/O 25	I/O 9	I/O 1	I/O 17
ADC2	I/O 2	I/O 26	I/O 10	I/O 2	I/O 18
ADC3	I/O 3	I/O 27	I/O 11	I/O 3	I/O 19
ADC4	I/O 4	I/O 28	I/O 12	I/O 4	I/O 20
ADC5	I/O 5	I/O 29	I/O 13	I/O 5	I/O 21
ADC6	I/O 6	I/O 30	I/O 14	I/O 6	I/O 22
ADC7	I/O 7	I/O 31	I/O 15	I/O 7	I/O 23

주의사항 : OPAMP 의 특성상, 전원전압중 사용할 수 없는 전압구간이 존재합니다. 5V 전원전압을 사용하므로, 대략 0.5V~4.5V 사이의 입력만 측정할 수 있습니다.

CB405 의 경우 채널 0~7 사이에서만 EADIN 명령을 사용할 수 있습니다.

Pwm

PWM Channel, Duty, Width

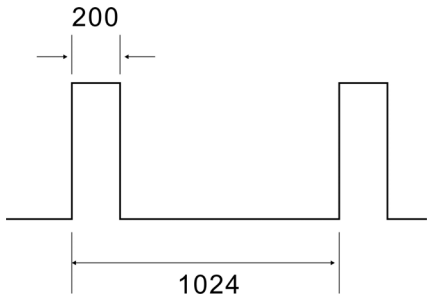
```
Channel : PWM 채널 (I/O 핀 번호가 아님)를 가리키는 변수 또는 상수
Duty : Duty 값 (변수 또는 상수), 이 값은 Width 보다 작아야 합니다.
Width : 한 주기의 최대 길이,최대 65535 까지 입력가능
```

원하는 채널에 PWM 파형을 출력합니다. PWM 명령에서는 포트번호가 아닌 채널번호를 사용하므로, 모델 별 채널에 따른 포트위치를 확인하시기 바랍니다. CB280/ CB380 의 경우 포트 5,6,7 에 PWM 채널 0,1,2 가 할당되어 있습니다. PWM 명령을 사용하기 전에 해당 포트를 반드시 OUTPUT 상태로 만들어 주어야 합니다.

Width 값에 따라 최대 16 비트 분해능을 갖는 PWM 파형을 만들어 냅니다. Width 값을 1024 로 하면 10 비트, 65535 로 하면 16 비트 분해능을 갖습니다. Duty 는 Width 값보다 작은 값으로 정합니다.

PWM 명령으로 CUBLOC 내부에 있는 PWM logic 이 작동을 개시하는 것이므로, CUBLOC 은 계속해서 다른 명령을 수행할 수 있습니다. 한번 PWM 명령이 실행되면, PWMOFF 명령을 만나기 전까지 계속 PWM 파형이 발생합니다.

PWM 출력으로 사용시, 해당 포트의 I/O 기능을 사용할 수 없게 됩니다. PWM 명령을 처음부터 사용하지 않았거나, PWMOFF 명령으로 사용을 정지한 뒤에는 I/O 포트로 사용할 수 있습니다.




```
LOW 5              '5 번 포트를 OUTPUT,LOW 상태로 만듦.
PWM 0,200,1024     '1024 주기안에서 200 만큼의 HIGH 를 출력하는 PWM 파형발생
```

PWM 0, 1, 2 번 채널의 주기(Width)는 같은 값을 사용해야 합니다. 내부적으로 하나의 타이머로 구동되기 때문입니다. PWM 3, 4, 5 도 주기(Width)는 하나의 값으로 통일해야 합니다. PWM 0, 1, 2 와 PWM 3, 4, 5 는 서로 달라도 상관없습니다.

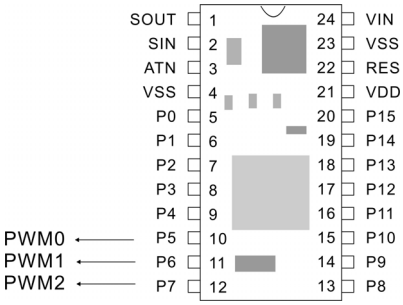
PWM 채널을 이용해서 고정 주파수를 발생시키려면 Freqout 명령을 사용하십시오.

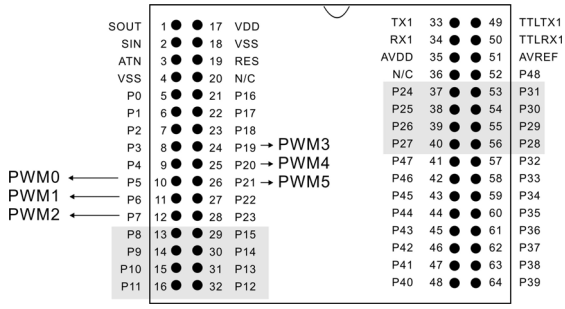
Pwmoff

PWMOFF Channel

Channel : PWM 채널 (I/O 핀 번호가 아님)를 가리키는 변수 또는 상수

출력중인 PWM 파형을 중단합니다. 다음은 모델 별로 사용할 수 있는 PWM 채널입니다.
CB220, CB320 의 경우 3 개의 PWM 채널을 사용할 수 있으며, 포트 5,6,7 에 할당되어 있습니다.





모델 별 PWM 채널과 포트와의 관계는 다음 표를 참조하시기 바랍니다.

채널-모델명	CB220 CB320	CB280 CB380	CB290	CT172X/C	CB405
PWM0	I/O 5	I/O 5	I/O 5	I/O 8	I/O 5
PWM1	I/O 6	I/O 6	I/O 6	I/O 9	I/O 6
PWM2	I/O 7	I/O 7	I/O 7	I/O 10	I/O 7
PWM3	I/O 19	I/O 19	I/O 89	I/O 11	I/O 27
PWM4	I/O 20	I/O 20	I/O 90	I/O 12	I/O 28
PWM5	I/O 21	I/O 21	I/O 91	I/O 13	I/O 29
PWM6					I/O 11
PWM7					I/O 12
PWM8					I/O 13
PWM9					I/O 51
PWM10					I/O 52
PWM11					I/O 53

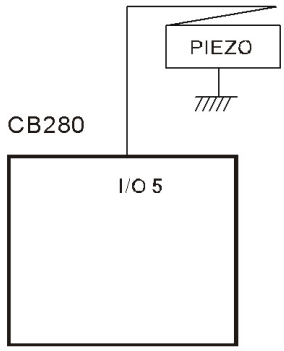
위 표에서 빈칸은 해당채널이 없음을 의미합니다. CB405 의경우 12 채널을 가지고 있습니다.

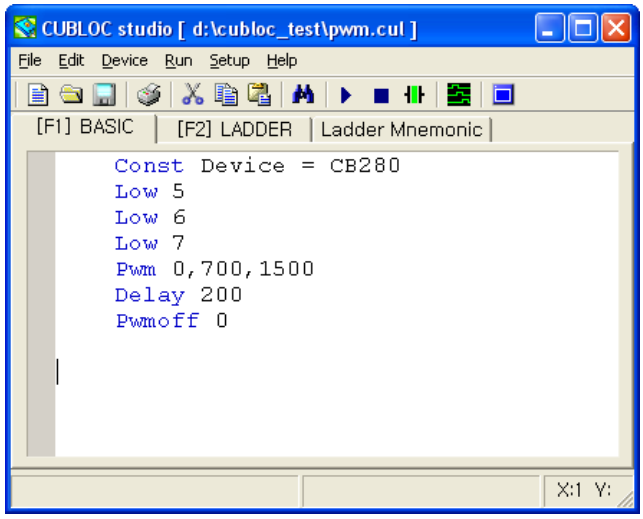
FREQOUT 명령으로 발생중인 파형을 중단할 때에도 PWMOFF 명령을 사용합니다. PWMOFF 명령과 INPUT 명령을 함께 사용한다면, PWM 출력과 동시에 해당핀을 입력상태로 만들수 있습니다.

```
Pwmoff 0
Input 5
```

DEMO PROGRAM

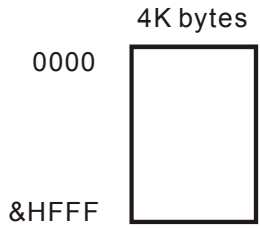
아래 회로처럼 포트 5 를 PIEZO 에 연결한 뒤 다음 샘플프로그램을 실행시키면, 짧은 삑~소리를 들을 수 있습니다.





EEPROM 억세스

CUBLOC 에 내장되어 있는 EEPROM 에 데이터를 기록하거나 읽어올 때 사용하는 명령어 입니다.



EEPROM Memory

Eeread( )

Variable = EEREAD (Address, ByteLength)

```
Variable : 결과가 저장될 변수
Address : 번지 (0~4095)
ByteLength : 읽어올 바이트 수를 가리키는 상수 또는 변수
```

EEPROM 의 지정한 Address 에서 데이터를 읽어옵니다. EEPROM 은 전원이 꺼져도 그 내용이 사라지지 않는 비 휘발성 메모리이기 때문에, 유저 세팅 값과 같이 전원과 상관없이 값을 유지해야 하는 경우에 사용합니다. ByteLength 는 읽어와야 할 바이트의 개수를 의미합니다. 바이트형 변수일 경우 1, 워드 형일 경우 2, long 형이나 single 형일 경우 4 를 써줍니다.

```
DIM A AS INTEGER
DIM B AS INTEGER
A = 100
EEWRITE 0,A,2        ‘ 0 번지에 변수 A 의 내용을 저장합니다.
B = EEREAD(0,2)      ‘ 0 번지에서 읽어와서 변수 B 에 저장합니다.
```

Eewrite

EEWRITE Address, Data, ByteLength

```
Address : 번지 (0~4095)
Data : 저장할 값이 들어있는 변수 또는 상수 (정수형 값만 저장가능)
ByteLength : 써야 할 바이트 수를 가리키는 상수 또는 변수
```

EEPROM 의 지정한 Address 에 데이터를 기록합니다. ByteLength 는 써야 할 바이트의 개수를 의미합니다. 바이트형 변수일 경우 1, 워드 형일 경우 2, long 형이나 single 형일 경우 4 를 써줍니다.

```
DIM A AS INTEGER
DIM B AS INTEGER
A = 100
EEWRITE 0,A,2        ' 0 번지에 변수 A 의 내용을 저장합니다.
B = EEREAD(0,2)      ' 0 번지에서 읽어와서 변수 B 에 저장합니다.
```

EEPROM 에서 데이터를 써넣는 경우, 약 3~5 밀리 초 정도의 시간이 소요됩니다. EEPROM 에서 읽어오는 경우에는 거의 시간이 걸리지 않습니다. 또한 EEPROM 의 같은 번지에 데이터를 썼다 지웠다 할 수 있는 수가 10 만번 정도로 제한되어 있습니다.

다음은 EEPROM 과 밧데리 백업의 장단점을 분석한 표입니다. 참고하셔서 기록할 데이터의 성격에 맞는 메모리를 선택하여 사용하시기 바랍니다.

종류	밧데리 백업 SRAM	EEPROM
유지기간	3 개월~1 년 (밧데리 용량에 따라 가변)	40 년
라이팅 횟수	제한 없음	10 만번
라이팅시간	없음	3~5 밀리 초
데이터 용도	순간정전 등에 대비할 수 있도록 유지되어야 하는 값. 자주 갱신되는 값. 예) 제품 생산량 카운트	갱신되는 횟수가 작은 값. 오랜 시간 유지되어야 하는 값. 예) 제품 시리얼 번호

* 주의사항 : EEPROM 은 수명이 있는 메모리입니다. 즉, 재 라이팅할 수 있는 대략 100 만번정도로 제한되어 있습니다. 따라서 프로그램을 작성하실 때, 너무 빈번히 라이팅을 하지 않도록 주의하시고, 라이팅을 시도하기전에 해당번지를 읽어봐서 같은 값이며 재차 라이팅을 할 필요가 없으므로, 스킵하도록 하시기바랍니다.

포인터와 RAM 직접억세스

C 언어에서 제공하는 포인터 개념을 BASIC 에서도 사용할 수 있도록 하였습니다. 어떤 변수의 저장되어 있는 RAM 주소를 알아내는 함수와 그 주소를 이용해서 값을 써넣거나 읽어 올 수 있는 명령입니다.

Memadr( )

Variable = MEMADR (TargetVariable)

```
Variable : 결과가 저장될 변수
TargetVariable : 주소를 알아낼 목적 변수
```

특정변수의 주소를 알 수 있습니다 .큐블록 BASIC 의 모든 변수는 RAM 에 저장됩니다. 변수의 할당은 컴파일러에서 알아서 할당하지만, MEMADR 함수를 이용한다면 해당변수가 위치한 주소를 알 수 있습니다.

```
Dim A as Single
Dim Adr as Integer
Adr = Memadr(A) '실수형 변수 A 의 주소를 반환합니다.
```

Peek( )

Variable = PEEK (Address, Length)

```
Variable : 결과가 저장될 변수
Address : RAM 의 주소
length : 읽어올 데이터의 길이
```

램의 지정한 주소에서 지정한 길이만큼의 데이터를 직접 읽어와 변수에 할당합니다.

Poke

POKE Address, Value, Length

```
Address : RAM 의 주소
Value : 저장할 데이터
length : 써야 할 데이터의 길이
```

램의 지정한 주소에서 임의의 값을 지정한길이(Length)만큼 직접 기입하는 명령입니다.

다음은 Peek, Poke, Memadr 명령의 활용예입니다. EEWRITE 명령은 정수형 데이터를 취급하는 명령이므로, 실수형 변수 값을 저장할 수 없습니다. 만약 EEWRITE, EEREAD 에서 실수형 데이터를 저장한 뒤 읽어 들인다면 다음과 같은 결과를 볼 수 있을 것입니다.

```
Const Device = CB280
Dim f1 As Single, f2 As Single
f1 = 3.14

Eewrite 0,f1,4
f2 = Eeread(0,4)
Debug Float f2,cr
```

Debug 창에는 3.14 가 나와야 하는데, 3.00000 이 표시됩니다. 그 이유는 EEWRITE 명령에서 F1 변수가 정수형 데이터로 자동 변환되었기 때문입니다. 실수형 데이터를 있는 그대로 EEPROM 에 저장하기 위해서는 메모리에 직접 접근하여, 데이터를 읽어오는 수밖에 없습니다. 다음 프로그램은 Memadr 과 Peek, Poke 명령으로 실수형 데이터의 EEPROM 저장을 구현한 것입니다.

```
Const Device = CB280
Dim f1 As Single, f2 As Single
f1 = 3.14

Eewrite 10,Peek(Memadr(f1),4),4
Poke Memadr(F2),Eeread(10,4),4

Debug Float F2,CR
```

Debug 창에 결과는 3.14 로 제대로 표시됩니다.

Memadr(f1)으로 f1 변수의 어드레스를 알아낸 뒤. Peek 명령으로 해당 번지에 들어있는 값을 직접 접근하여 4 바이트 읽어옵니다. 여기까지 Peek(Memadr(f1),4) 로 표현할 수 있습니다. 그 값을 EEPROM 에 저장하는 것입니다.
읽어올 때에는 반대로 Poke 명령으로 Memadr(f2)의 어드레스에 직접 기입합니다.

주의사항

Poke 명령을 사용할 때에는 충분한 주의를 기울여야 합니다. 포인터 구조를 정확히 이해하지 못한 상황에서 사용한다면, 전체 프로그램의 동작에 치명적인 영향을 줄 수 있으므로 주의해야 합니다. Peek, Poke 는 큐블록의 데이터 메모리인 SRAM 에만 접근할 수 있으며, I/O 레지스터나 프로그램 메모리 (플레쉬 영역)에는 접근할 수 없습니다.

카운터

카운터 포트는 일반 I/O 와 같이 쓸 수 있도록 되어 있으므로, 카운터 입력을 사용하려면 I/O 기능은 사용하지 못하게 됩니다. CUBLOC 에 내장된 카운터는 내부 프로그램 수행여부와 상관없이 동작되는 별도의 하드웨어로 구성되어 있습니다. 프로그램이 실행 중 일 때에도 항상 카운터 입력을 받아들이고 있으므로, 외부로부터 신호입력을 놓치는 일이 발생하지 않습니다.

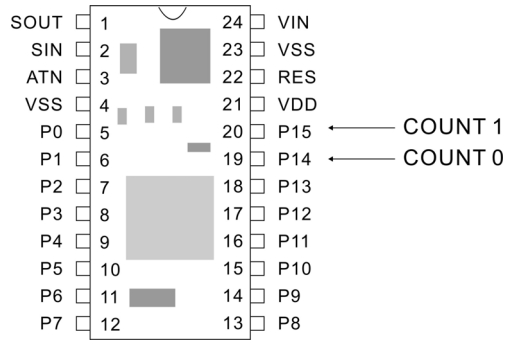
CUBLOC 은 2 개의 카운터 입력 핀이 있으며, 이중 채널 1 번은 항상 카운터 입력을 받을 수 있습니다. 해당 포트를 입력 상태로만 만들어준다면 카운터 입력을 위한 준비는 끝난 것입니다. 카운터 채널 0 번을 사용하려면 PWM0,1,2 채널의 사용을 포기해야만 합니다. 카운터 채널 0 과 PWM0,1,2 번 채널은 내부적으로 같은 자원을 활용하고 있기 때문에 둘 중 하나를 선택해서 사용해야만 합니다. CUBLOC 은 모두 6 개의 PWM 채널 출력을 지원하므로, PWM 출력 중 3,4,5 번 채널 3 개는 카운터 사용여부와 상관없이 사용 가능합니다. 카운터채널 0 을 사용하기 위해서는 SET COUNT0 명령을 통해 카운터 채널 0 사용여부를 선언해 주어야 합니다. 최초 디폴트상태에서는 OFF 로 되어 있어서 COUNT 채널 0 을 사용할 수 없는 상태로 되어 있습니다.

Count( )

Variable = COUNT(channel)

```
Variable : 결과를 저장할 변수
Channel : 카운터 채널
```

카운트 입력 핀으로부터 받은 카운터 개수를 반환합니다. CUBLOC 에는 2 개의 카운터 채널이 있습니다. 카운터는 별도의 하드웨어로 구성되어 있기 때문에, 베이직 프로그램 수행 여부와 관계없이 항상 핀으로부터 들어오는 펄스의 개수를 카운트하고 있다가 COUNT 함수를 통해서 읽어 낼 수 있습니다. CB220, CB320, CB280, CB380 의 경우 카운터 채널 1 으로 쓸 수 있는 포트 15 의 경우 일반 I/O 포트로도 사용할 수 있기 때문에 카운터로 쓰기 위해서는 반드시 입력 포트로 만들어 주어야 합니다. CB290 의 경우 카운터 채널 1 로 쓸 수 있는 23 번 포트를 입력으로 만들어 주어야 합니다.



카운터 채널 0 을 사용하기 위해서는 하나의 명령을 더 써주어야 합니다. SET COUNT0 명령을 사용해서 카운터 채널 0 을 사용할 것임을 선언해 주어야 합니다. 카운터 채널 0 을 사용하면 PWM0,1,2 번 채널을 사용할 수 없습니다. 카운터 채널 0 과 PWM0,1,2 채널은 같은 자원을 사용하기 때문에 둘 중 하나만을 선택해 주어야 합니다. 다시 PWM0,1,2 채널을 사용하려면 SET COUNT0 OFF 명령을 실행하면 됩니다.

```
DIM R AS INTEGER
INPUT 15              ' 15 번 포트를 입력을 만듭니다.(카운터 채널 1)
R = COUNT(1)          ' 카운트된 값을 읽어옵니다.

SET COUNT0 ON         ' 카운터 채널 0 을 활성화 시킵니다.(PWM0,1,2 는 사용불가)
INPUT 14              ' 14 번 포트를 입력을 만듭니다.(카운터 채널 0)
R = COUNT(0)          ' 카운트된 값을 읽어옵니다.
```

Countreset

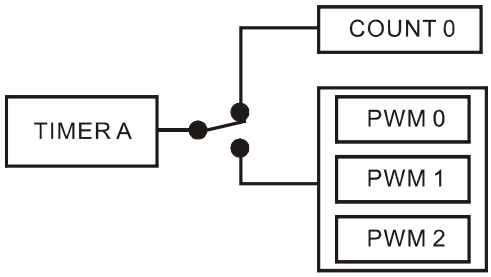
COUNTRESET channel

Channel : 카운터 채널

원하는 채널의 카운트 값을 0 으로 리셋 시킵니다.

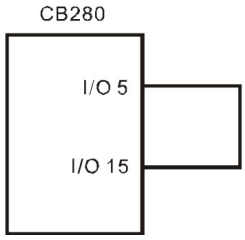
```
COUNTRESET 0 '채널 0 을 클리어 합니다.
COUNTRESET 1 '채널 1 을 클리어 합니다.
```

다음은 PWM 채널과 COUNTER 와의 관계를 블록도로 표현한 것입니다. PWM0,1,2 는 하나의 타이머로 구동되며, 이곳에 카운터 채널 0 이 연결되어 있어, 카운터와 PWM 중 한쪽만 사용할 수 있습니다.



DEMO PROGRAM

카운터입력 기능을 사용해서, 주파수를 측정하는 프로그램입니다. 큐블록 CB280 의 포트 15(COUNTER 채널 1)과 포트 5 (PWM 채널 0)을 서로 연결시킵니다.

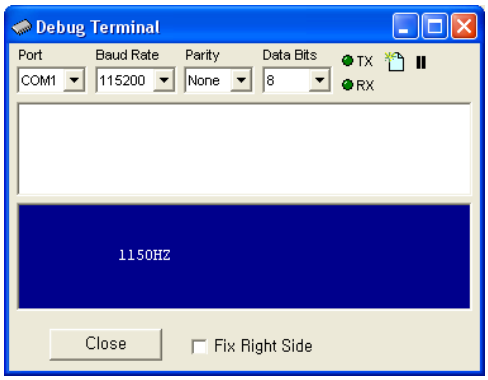


이렇게 하면, 포트 5 번에서 출력되는 PWM 파형이 COUNTER 1 번 채널로 입력됩니다.

FREQOUT (또는 PWM 명령)으로 파형을 발생시키고, COUNT 명령으로 읽어 들이는 것입니다.

1 초에 몇 개의 펄스가 입력되었는지 체크하면 주파수를 알 수 있습니다.

```
Const Device = CB280
Dim A As Integer
Input 15
Low 5
Freqout 0,2000
Low 0
On Timer(100) Gosub GetFreq  '1 초마다 한번씩 GetFreq 루틴으로 점프
Do
Loop      '인터럽트를 위해 무한 대기

GetFreq:
A = Count(1)    ' 현재 카운터의 값을 읽어옵니다.
Debug Goxy,10,2
Debug dec5 A,"Hz"
Countreset(1)   ' 다음을 위해서 카운터를 클리어 합니다.
Reverse 0       ' 0 번 포트를 스코프로 측정하면,
              ' 1 초마다 루프를 도는지 알 수 있음.
Return
```



디버그 창에 1150HZ 라고 표시됩니다. Freqout 0,2000 으로 명령하면 1150Hz 가 출력됩니다. (Freqout 명령설명 참조)

이것을 Count(1)명령으로 1 초마다 한번씩 읽어 들여 표시합니다.

1 초마다 한번씩 실행하기 위해 On Timer 인터럽트를 사용한 것입니다.

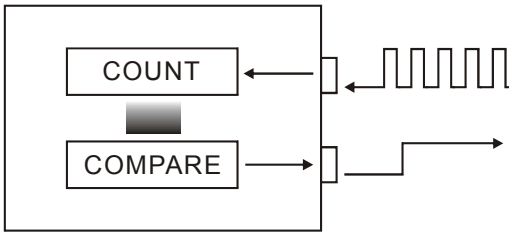
컴페어

고속카운터가 목표치에 도달하였을 때, 특정포트를 High 나 Low 로 만들어주는 명령어입니다.

Compare

COMPARE channel, target#, port, targetstate

```
Channel : 카운터 채널
Target# : 목표치 펄스갯수
Port : 출력포트 (입력전용포트 사용금지)
Targetstate : (목표상태), 바꾸고자 하는 출력포트의 상태
```



이 명령은 고속 카운터와 연동되어 동작하는 명령입니다. 고속카운터로 입력되는 펄스의 개수를 항상 감시하고 있다가 목표치를 초과하면 정해준 포트를 원하는 상태로 바꿉니다. 이때 포트는 반드시 출력이 가능한 포트(입력 포트제외)로 설정해 주십시오. 목표상태를 1 로 하면, 목표치에 도달했을 때 해당포트가 High 가 됩니다. 반대로 0 으로 하면 해당포트의 상태가 Low 가 됩니다.

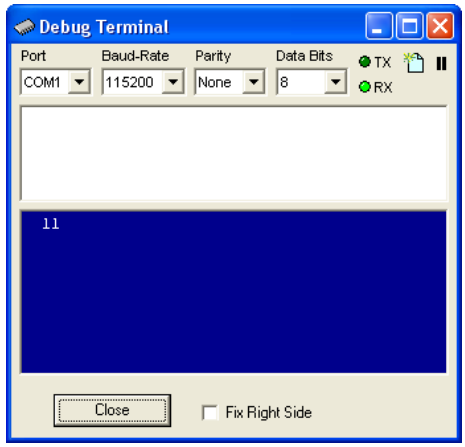
이 명령어를 실행하면 목표상태의 반대로 해당포트가 바뀌어 집니다. 유저가 일부러 바꿀 필요가 없습니다. 고속카운터 채널 0 은 목표치를 최대 65535 까지 설정할 수 있습니다. 채널 1 은 목표치를 최대 255 까지 설정할 수 있습니다.

채널	비교범위
COUNT 채널 0	0~255
COUNT 채널 1	0 ~65535

고속카운터는 32 비트까지 저장할 수 있지만, COMPARE 명령에서 비교할 수 있는 범위는 다소 제한적입니다. 그 이유는 COMPARE 는 CUBLOC 메인칩의 하드웨어기능을 활용하여, 유저 프로그램실행 속도에 영향을 주지 않도록 설계되었기 때문입니다.

또한 채널 0 은 Set Count0 On 명령어를 사용하여 활성화시켜준 뒤에 Compare 명령으로 목표치를 설정해 주어야 합니다.

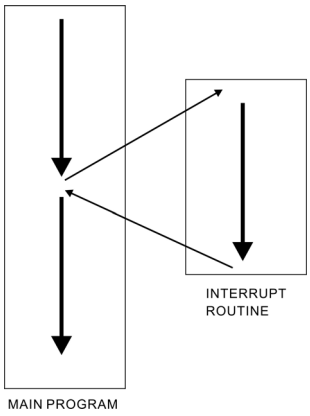
```
Dim i As Integer
Set Count0 On
Compare 0,10,61,1
Do
      i = Count(0)
      Debug Goxy,0,0,dec4 i,Cr
      Delay 100
Loop
```



위의 경우 채널 0 을 사용했고, 목표치를 10 으로 설정해두었습니다 .카운터 0 의 값이 10 을 넘었을 때, 포트 61 번이 High 가 됩니다.

인터럽트

인터럽트는 메인 프로그램 실행 도중 반드시 처리해야 할 상황에 대한 즉각적인 대처를 위한 장치입니다. 인터럽트는 ON…GOSUB 명령에 의한 인터럽트 선언에 의해 활성화되며, 해당 인터럽트가 발생되면 메인 프로그램의 실행을 잠시 멈추고, ON…GOSUB 명령에서 지정한 라벨로 점프하게 됩니다. 인터럽트 루틴에서는 필요한 조치를 하고 RETURN 하는 것으로 인터럽트에 대한 처리를 모두 마치게 됩니다.



외부 키 입력 입력이나 RS232 수신과 같이, 언제 발생할지 모르는 임의의 사건에 대하여, 메인 프로그램에서 무조건 대기할 수는 없기 때문에, 인터럽트가 필요한 것입니다. 메인 프로그램은 다른 일을 실행하고 있더라도, 외부 펄스엣지가 검출되거나, RS232 데이터가 들어오면, 메인 프로그램의 실행을 멈추고 지정한 인터럽트 루틴을 수행하게 됩니다. 이때 주의할 점은 인터럽트 루틴의 가장 끝에는 반드시 RETURN 명령어를 써주어야 한다는 것입니다. 인터럽트 루틴 수행을 끝마치고 메인 프로그램을 복귀하기 위해서 입니다.

CUBLOC 인터럽트 메커니즘은 하나의 인터럽트 종류에 대하여, 하나의 인터럽트 루틴을 할당할 수 있고, 인터럽트 루틴을 실행하는 동안 동일 종류의 인터럽트 요구는 무시됩니다. 만일 RS232 RECV 인터럽트가 발생되어 해당 인터럽트 루틴을 수행하고 있는 중간에 RS232 데이터가 수신되면, 인터럽트 루틴이 끝날 때까지 수행이 유보됩니다. 하지만 RS232 수신되는 데이터는 계속해서 버퍼에 쌓이고 있습니다. RECV 인터럽트 루틴 수행도중 INT 엣지 인터럽트가 들어오면 RECV 인터럽트루틴 수행을 잠시 멈추고 INT 엣지 입력 인터럽트 처리루틴을 수행합니다. CUBLOC 에서는 인터럽트의 종류가 틀리면, 중복 인터럽트가 허용되고, 종류가 같으면 인터럽트 요구가 무시됩니다. 각각의 인터럽트는 SET ONRECV 등의 명령을 통해 임시로 ON 또는 OFF 할 수 있으며, SET GLOBALINT 명령을 통해 전체 인터럽트를 ON 또는 OFF 할 수 있습니다.

다음은 큐블록에서 지원하는 BASIC 인터럽트 종류를 요약한 표입니다.

인터럽트 명	설명
On Timer	일정 시간간격으로 인터럽트를 발생
On Int	외부 핀 입력의 상태변화를 감지하여 인터럽트 발생
On Recv	RS232 수신이 발생되면 인터럽트 발생
On LadderInt	Ladder Logic 에서 인터럽트 요구가 있을 때 인터럽트 발생
On Pad	Pad 통신에서의 수신데이터가 있으면 인터럽트 발생

On Timer

ON TIMER(interval) GOSUB label

Interval : 인터럽트 시간간격 1=10mS, 2=20mS.....65535=655350mS
1~65535 까지 사용가능

일정한 시간 간격으로 처리해야 될 일이 있을 경우 사용하는 인터럽트 입니다. 괄호 안에 있는 수치가 시간 간격을 의미합니다. 1 일 경우 10 밀리 초 간격으로 인터럽트 루틴을 호출합니다. 100 일 경우 1 초 간격으로 인터럽트 루틴을 호출합니다. 최대 65535 까지 사용할 수 있으며 이 경우 655.35 초 간격이 됩니다. 간격에 0 을 입력했을 경우에는 문법에러가 됩니다.

이 인터럽트는 사용시 특히 주의해야 할 사항이 있습니다. 인터럽트 루틴에서 지정한 시간간격보다 빠른 시간 안에 인터럽트 루틴 수행이 끝마치도록 해야 합니다. 예를 들어 10mS 간격으로 인터럽트 루틴을 수행하도록 해놓았는데, 인터럽트 루틴에서 20mS 정도의 실행시간이 걸린다고 하면, 계속해서 인터럽트 루틴만 실행하게 되므로, 본 프로그램은 영원히 실행되지 않게 됩니다.

```
ON TIMER(100) GOSUB TIMERTN
DIM I AS INTEGER
I = 0
DO
LOOP

TIMERTN:
I = I + 1      ' 변수 I 는 1 초마다 1 씩 증가하는 카운터가 됩니다.
RETURN
```

주의사항

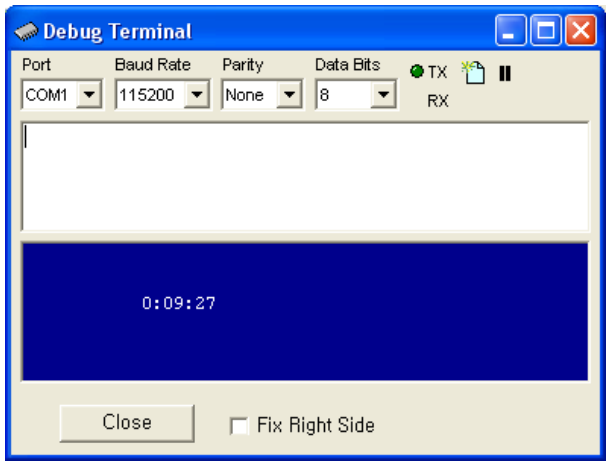
LADDER 도 ON TIMER 처럼 BASIC 수행 중 일정 시간간격으로 LADDER 를 실행하는 방식이기 때문에, ON TIMER 와 동시에 사용되었을 경우, LADDER 실행에 영향을 미칠 수 있습니다. 따라서 LADDER 와 ON TIMER 를 동시에 사용하는 것은 가급적 피해주시고, 어쩔 수 없이 동시에 사용해야 한다면 ON TIMER 의 시간간격을 100mS 이상으로 설정해 주시기 바랍니다.

DEMO PROGRAM

On Timer 인터럽트를 이용해서 만든 타이머 프로그램입니다.

```
    Const Device = CB280
    Dim ss As Integer
    Dim mm As Integer
    Dim hh As Integer
    Ramclear
    On Timer(100) Gosub ClockTick
    Do
    Loop

ClockTick:                  ' 1 초마다 이곳을 수행합니다.
    Debug Goxy,10,2
    Debug Dp(hh,2,0),":"
    Debug Dp(mm,2,1),":"
    Debug Dp(ss,2,1)
    Incr ss                 ' 1 초 증가
    If ss > 60 Then
          ss=0
          Incr mm           ' 1 분 증가
          If mm > 60 Then
                mm=0
                Incr hh     ' 1 시간 증가
          End If
    End If
    Return
```

실행시키면 디버그 터미널에 다음과 같은 내용이 나타납니다. 시간을 표시하는 것처럼 타이머의 동작을 표시하고 있습니다.



On Int

ON INTx GOSUB label

x : 0~3 사이의 값, 외부 인터럽트 채널 번호

외부로부터 인터럽트 입력을 받을 경우 사용합니다. CUBLOC 에는 4 개의 외부 인터럽트 핀이 있습니다. 인터럽트 핀을 입력으로 정의하고, 여기에 인터럽트 신호를 입력합니다. 인터럽트 신호는 하강엣지, 상승 엣지, 변화감지의 세가지 모드로 선택할 수 있습니다. 인터럽트 입력모드를 선택하는 명령은 SET ONINTx 명령입니다.

상승엣지 하강엣지

```
  DIM A AS INTEGER
  ON INT0 GOSUB GETINT0
  SET INT0 0            '하강엣지에서 검출
  DO
  LOOP

GETINT0:
  A=A+1                 '검출 횟수를 기록
  RETURN
```

인터럽트 선언문이 두개일때는 어떻게 되나?

```
  ON INT0 GOSUB GETINT0
  ON INT0 GOSUB GETINT1
```

같은 인터럽트에 대해서 두개의 선언문이 있을 경우에는 가장 최근에 실행된 선언문이 유효하고, 종전의 것은 취소됩니다. 위의 경우 INT0 이 인터럽트가 발생되면 GETINT1 으로 점프합니다.

INT 인터럽트를 비롯한 모든 큐블록의 인터럽트 (ON RECV, ON TIMER 등등)도 마찬가지입니다. 하나의 인터럽트당 하나의 벡터 (실행루틴)을 지원합니다.

인터럽트 금지/재동작

인터럽트를 금지시키거나, 재 동작시킬 필요가 있을 때 사용하는 명령입니다. 전체 인터럽트를 금지 시키거나 재 동작시킬 수 있는 Set Onglobal 명령과, 각각의 인터럽트를 제어하는 명령이 있습니다.

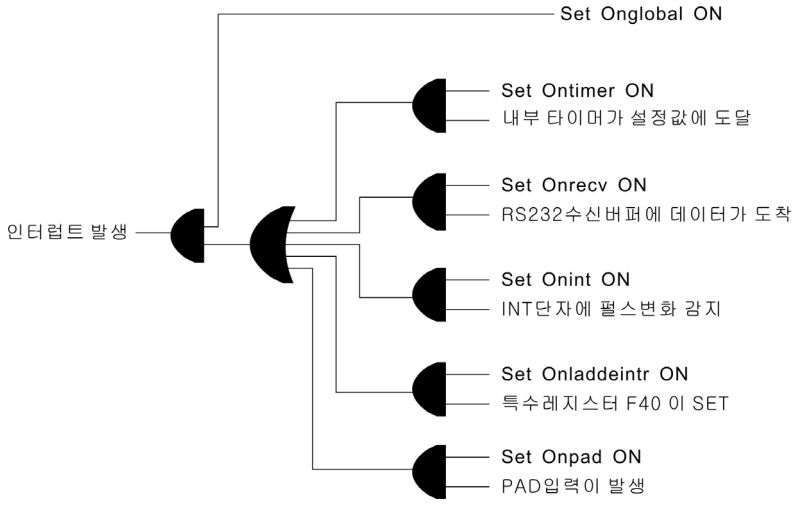
Set Onglobal

SET ONGLOBAL on/off

전체 인터럽트를 on 또는 off 하는 명령입니다. 디폴트 상태에서 ON 입니다.

```
SET ONGLOBAL OFF
```

사용하는 인터럽트가 없다면 소스프로그램 맨 앞에 SET ONGLOBAL OFF 명령을 써주는 것이 좋습니다. CUBLOC OS 에서 인터럽트 수신 체크하는 부분을 실행하지 않으므로 전체적인 프로그램 실행시간이 다소 빨라집니다. 다음은 큐블록의 인터럽트 발생 메커니즘을 한눈에 알기 쉽게 그림으로 표현한 것입니다.



Set Int

SET INTx mode

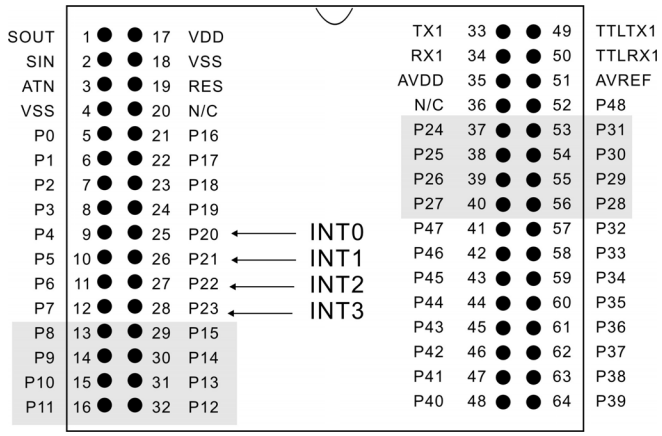
x : 0~3 사이의 값, 외부 인터럽트 채널 번호

mode : 0=하강엣지, 1=상승엣지, 2=둘다검출

외부로부터 들어오는 인터럽트 입력신호의 종류를 결정합니다. 0 일 경우 하강엣지에서 인터럽트를 발생시키고, 1 일 경우 상승엣지에서 인터럽트를 발생시킵니다. 2 일 경우 두 가지 경우 모두에서 인터럽트를 발생시킵니다.

```
SET INT0 0          '하강엣지에서 검출
```

인터럽트 수신 핀의 위치는 다음과 같습니다.



Set Ontimer

SET ONTIMER on/off

타이머 인터럽트를 on 또는 off 하는 명령입니다. ON TIMER 명령 수행 시 ON 상태가 됩니다.

Set Onrecv

SET ONRECV0 on/off

SET ONRECV1 on/off

해당 채널의 RS232 수신 인터럽트를 on 또는 off 하는 명령입니다. ON RECV 명령 수행 시 ON 상태가 됩니다.

```
SET ONRECV1 ON
SET ONRECV1 OFF
```

Set Onint

SET ONINTx on/off

INT 수신 인터럽트를 on 또는 off 하는 명령입니다. ON INT 명령 수행 시 ON 상태가 됩니다.

```
SET ONINT0 ON
SET ONINT1 ON
SET ONINT1 OFF
SET ONINT2 OFF
SET ONINT3 ON
```

Set Onpad

SET ONPAD on/off

ONPAD 인터럽트를 on 또는 off 하는 명령입니다. ON ONPAD 명령 수행 시 ON 상태가 됩니다.

```
SET ONPAD ON
```

Set OnLadderint

SET ONLADDERINT on/off

LADDERINT 인터럽트를 on 또는 off 하는 명령입니다. ON LADDERINT 명령 수행 시 ON 상태가 됩니다.

인터럽트를 좀더 깊숙이 살펴보면..

큐블록 베이직의 인터럽트는 큐블록의 OS 가 관리하는 일종의 이벤트 처리입니다. 따라서 "MCIRO CONTROLLER"에서 하드웨어적으로 처리되는 인터럽트와는 차이가 있습니다.

1. 일단 한 종류의 인터럽트가 발생되면 인터럽트루틴을 수행하는 동안, 같은 종류의 인터럽트는 처리할 수 없습니다. 하지만 다른 종류의 인터럽트는 받고, 처리할 수 있습니다.

2. 종류가 다른 인터럽트가 비슷한 시점에 걸리면, 최근에 걸린 것이 먼저 처리됩니다. ON TIMER 인터럽트가 걸려서, 해당 인터럽트 루틴을 수행하려고 하는데, 바로 ON INT 인터럽트가 발생되면, ON INT 인터럽트 루틴을 먼저수행하고, ON INT 인터럽트 루틴이 종료되면, ON TIMER 루틴을 마저 실행합니다.

3. 전체인터럽트를 ON / OFF 하는 Set OnGlobal 명령은 프로그램에서 특별히 건드리지 않는 이상, 인터럽트 발생시 자동적으로 On / Off 되지 않습니다. "Micro Controller"의 경우 인터럽트가 발생되면 GIE (Global Interrupt Enable) 플레그가 자동적으로 금지상태가 되고, 인터럽트 수행을 끝마치면 가능(Enable)상태가 되지만, 큐블록에서는 이런 기능이 들어 있지 않습니다. 따라서 하나의 인터럽트 루틴수행 중 다른 인터럽트도 받을 수 있습니다.

4. 인터럽트루틴의 가장 끝에는 반드시 Return 명령이 있어야 합니다. 인터럽트 루틴 안에 있는 Return 명령은 인터럽트가 발생되기 이전의 실행지점으로 되돌리는 일과 더불어, 해당 인터럽트의 금지상태도 해제시켜줍니다.

5. 인터럽트의 사용횟수에는 제한이 없습니다.

6. Delay, Pulsout 명령수행 중 인터럽트 발생이 가능합니다. 따라서 Delay 시간, Pulsout 의 펄스주기 등이 인터럽트의 영향으로 가변 될 수 있습니다. 만약 Delay 시간과 Pulsout 주기를 정확히 지켜주고 싶다면, 해당 명령 앞에는 Set Onglobal Off 명령을 써주고, 뒤에는 On 명령을 써주십시오. 그렇게 하면, 해당 명령을 수행하는 동안 인터럽트 요구를 받지 않게 됩니다.

```
Set Onglobal Off
Delay 100              ' 인터럽트의 방해를 받고 싶지 않은 Delay 명령
Set Onglobal On
```

7. 인터럽트를 하나도 사용하지 않는 프로그램이라면, 프로그램 맨 처음에 Set Onglobal Off 명령을 써주게 되면, 프로그램 전체의 실행시간이 조금 빨라집니다. 왜냐하면, 매번 명령을 실행할 때마다 인터럽트 발생여부를 체크하지 않기 때문입니다. 최초 디폴트 상태에서는 Set Onglobal On 상태입니다.

8 RS232 수신 인터럽트인 On Recv 의 경우, 인터럽트 루틴 중 수신하는 데이터는 버퍼에 계속해서 쌓이게 됩니다. 인터럽트 루틴이 끝날 때까지 들어온 데이터는 인터럽트에 영향을 주지 않습니다. 즉, 인터럽트루틴 종료 후 수신버퍼에 아직도 데이터가 남아있다면, 다시 On Recv 인터럽트가 발생됩니다. 따라서 인터럽트 루틴 안에서 수신된 모든 데이터를 읽어서 처리하고, 그래도 남은 데이터가 있고 (처리를 원치 않는다면) Bclr 명령으로 버퍼를 청소해 두어야, 재차 인터럽트가 발생되지 않습니다.

9. 똑 같은 인터럽트 선언을 두번하였다면, 최근에 실행된 것이 유효합니다.

LADDER 관련명령

BASIC 에서 LADDER 의 실행과 관련해서 일부 기능을 콘트롤 하거나 I/O 핀의 사용을 선언할 수 있는 명령이 있습니다. CUBLOC 의 운영체계는 처음에 전원이 들어온 뒤 BASIC 명령부터 해석하여 실행합니다. BASIC 에서 LADDER 를 실행하는 명령을 수행하기 전까지는 LADDER 명령을 실행하지 않습니다.

LADDER 의 실행을 결정하는 명령과, LADDER 에서 사용하는 "별명"선언명령, 그리고 I/O 포트의 상태를 결정하는 명령이 있습니다.

Set ladder on/off

SET LADDER On[/Off]

l0mS 간격으로 LADDER 를 실행개시 하는 명령입니다. 최초에는 OFF 상태입니다.

Ladderscan

LADDERSCAN

LADDER 를 강제로 한 스캔 실행하는 명령입니다. Ladderscan 명령을 Do..Loop 와 같은 무한루프 안에 두면, 가변 스캔타임으로 보다 빠르게 Ladder 를 실행할 수 있습니다. 10mS 보다 빠른 간격으로 레더를 실행하고 싶다면 아래와 같은 방법으로 프로그램을 작성하십시오.

```
DO
      LADDERSCAN
LOOP
```

주의사항 : DO…LOOOP 안에 LADDERSCAN 명령만 있을 경우 더 이상 다른 BASIC 명령은 수행을 멈추게 됩니다. 만약 BASIC 명령이 필요한 경우가 있다면 DO…LOOP 안에 LADDERSCAN 명령과 같이 두는 방법으로 BASIC 과 동시에 실행을 할 수 있습니다. 당연히 DO…LOOP 안에 BASIC 명령어가 많이 있다면 그만큼 LADDER 의 스캔속도는 떨어지게 됩니다.

Alias

ALIAS Relayname = AliasName

Relayname : P0, M0, T0 등의 릴레이 이름
AliasName : 릴레이 이름 대신 사용할 별명이름

P0, M0, C0 등과 같은 릴레이 이름대신 사용할 "별명"을 선언하는 명령입니다. 선언된 별명을 사용하면 보다 쉽게 LADDER 를 작성하고 이해할 수 있습니다.

```
Alias M0 = Rstate
Alias M0 = Kstate
Alias P0 = StartSw
```

Aliason
Aliasoff

여러 개의 Alias 를 선언하는 경우에 사용하는 명령입니다. 좀더 간편하게 입력할 수 있습니다.

```
AliasOn
  M0 = Rstate
  M1 = Rstate2
  M2 = Rstate3
AliasOff
```

Usepin

USEPIN I/O, In/Out, AliasName

I/O : I/O 포트 번호
In/Out : 입출력상태, In 또는 Out
AliasName : 릴레이 이름 대신 사용할 별명이름

LADDER 에서 사용할 I/O 핀을 정의하는 명령입니다. I/O 포트는 BASIC 과 LADDER 에서 모두 사용할 수 있지만, LADDER 에서 사용하고자 할 때에는 반드시 USEPIN 명령을 사용해서 LADDER 에서 사용할 것임을 명시해 두어야 합니다.

LADDER 에서는 I/O 포트를 주기적으로 "리플레쉬"하기 때문에, BASIC 에서 사용하는 I/O 포트와 반드시 구분해 두어야 할 필요가 있습니다.

USEPIN 명령에서는 동시에 입출력 상태와 "별명"도 같이 정의할 수 있습니다.

```
USEPIN 0,IN,START
USEPIN 2,IN,BKEY
USEPIN 3,OUT,MOTOR
```

BASIC 에서 I/O 핀을 사용하기 위해서 특별히 사용해야 될 명령어는 없습니다. 파워 온 상태에서 기본적으로 모든 I/O 핀은 BASIC 에서 사용하도록 되어 있기 때문입니다.

주의사항 : I/O 핀의 모든 관리책임은 유저에게 있습니다. CUBLOC STUDIO 에서 컴파일 할 때, 잘못된 I/O 할당에 대하여 특별한 에러멧세지를 발생시킬 수 없기 때문에, 유저는 어떤 I/O 핀을 BASIC 에서 사용하고, 어떤 I/O 핀을 LADDER 에서 사용할 것인지 사전에 잘 계획하여, USEPIN 명령으로 나눈 뒤, 해당영역에서 자신의 I/O 만을 사용하도록 각별히 주의하여 프로그램을 작성하시기 바랍니다.

Freepin

FREEPIN I/O, In/Out, AliasName

I/O : I/O 포트 번호

USEPIN 을 다시 해제해주는 명령입니다. 앞에서 USEPIN 으로 선언해서 LADDER 쪽으로 할당된 I/O 포트를 다시 BASIC 쪽으로 할당하는 명령입니다.

On Ladderint Gosub

ON LADDERINT GOSUB label

LADDER 에서 특수릴레이 F40 에 1 을 써넣으면, ON Ladderint Gosub 에서 선언한 라벨로 점프합니다. LADDER 수행도중 BASIC 의 명령의 도움이 필요하거나, BASIC 의 특정루틴을 실행해야 할 필요가 있을 때 사용합니다.

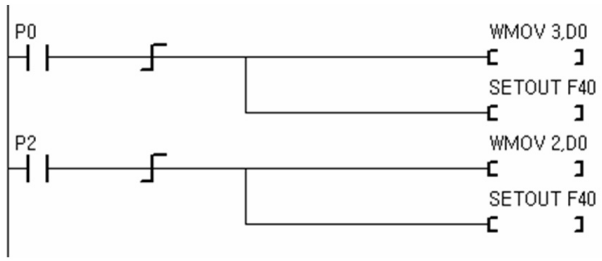
LADDER 프로그램 작성시, 반드시 SETOUT 명령과 미분명령 DF 를 사용해서 F40 에 1 을 써넣어야 합니다. 이후 BASIC 에서 인터럽트 루틴을 모두 끝마쳤을 때, F40 을 클리어 합니다.

인터럽트 루틴 수행 중에는 F40 에 1 을 써넣어도 아무 효과가 일어나질 않습니다. BASIC 에서 F40 을 클리어 하면 인터럽트 수행이 모두 끝났다는 것을 의미하며, 다음 인터럽트를 받을 준비가 되었다는 것을 뜻합니다.

```
    Usepin 0,In
    Set Ladder On
    Set Display 0,0,16,77,50
    On Ladderint Gosub msg1_rtn
    Dim i As Integer
    Low 1
    Do
         i=i+1
         Byteout 1,i
         Delay 200
    Loop
msg1_rtn:
    Locate 0,0
    Print "ON Ladderint",Dec i
    Reverse 1
    Return
```

LADDER 에서 P0 에 입력이 들어올 때, LCD 에 문자를 표시하는 샘플 프로그램입니다. 평상시 BASIC 에서는 DO...LOOP 에서 BYTEOUT 처리를 하고 있습니다. 그러다가 LADDER 에서 F40 에 1 이 기입되면, ON LADDERINT 에서 선언한 MSG1_RTN 인터럽트 루틴을 수행합니다. 이 루틴에서 PRINT 명령으로 LCD 에 문자를 표시하는 것입니다.

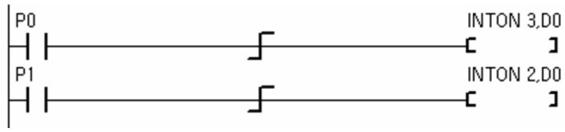
LADDER 에서 BASIC 으로 인터럽트를 발생시키는 방법은 F40 을 이용하는 단 하나의 방법뿐이지만, 데이터 영역인 D 영역의 특정번지에 원하는 처리의 종류를 기입한 후, 인터럽트를 발생시키고, BASIC 에서 해당 D 영역을 참조하여, 어떤 처리를 원하는지 읽어내는 방법을 사용하면, 하나의 인터럽트로 여러 가지 처리를 할 수 있습니다.



P0 에 입력이 들어오면 D0 에 3 을 넣고, F40 을 1 로 만듭니다. P2 에 입력이 들어오면 D0 에 2 를 넣고, F40 을 1 로 만듭니다. 즉, D0 에 인터럽트 루틴에서 처리해야 할 일의 종류 등을 기입해 두는 것입니다.

```
msg1_rtn:
    If  D(0)=3 Then
          Locate 0,0
          Print "ON Ladderint",Dec i
    End If
    If  D(0)=2 Then
          Locate 0,0
          Print "TEST PROGRAM",Dec i
    End If
    Return
```

위의 WMOV 3,D0 과 SETOUT F40 두 줄로 작성한 명령을 INTON 명령으로 바꾸면 한 줄로 간단하게 작성할 수 있습니다. INTON 명령은 WMOV 와 같은 동작을 하는 명령입니다. 다만 SEROUT F40 명령을 동시에 수행합니다.



디버깅

Debug

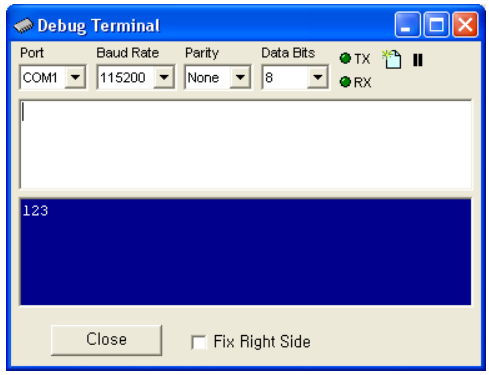
DEBUG data

`data : PC 로 전송할 데이터`

“디버그 터미널”로 데이터를 전송하는 명령입니다. 큐블록에서는 디버그 명령을 사용해서, 동작 중 보고 싶은 변수 값이나, 동작상태 등을 체크합니다. 일종의 “디버그 툴”이라고 할 수 있습니다.

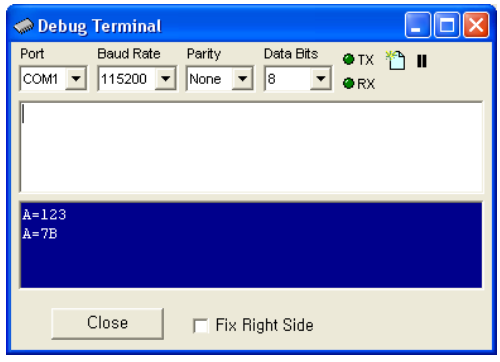
Debug 명령이 동작하기 위해서는, 큐블록과 PC 가 연결된 상태이어야 합니다. 이 상태에서 Debug 명령이 수행되면, 큐블록에서 PC 쪽으로 데이터를 전송하게 되고, PC 에서는 “디버그 터미널”에 그 값이 표시되게 됩니다.

```
Const Device = CB280
Dim A As Integer
A = 123
Debug Dec A
```



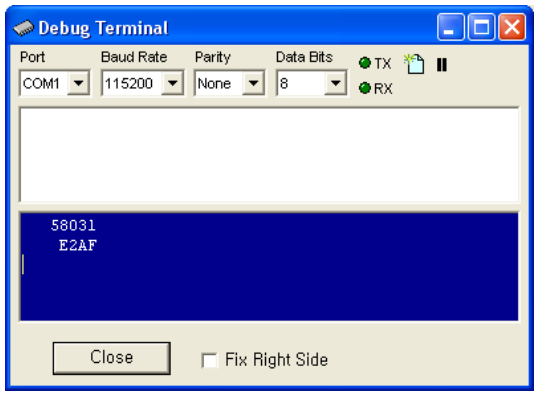
DEC 와 HEX 를 사용하여 표시할 변수의 진법을 변환합니다. HEX 나 DEC 없이 변수만 적어준다면 변수의 내용이 ASCII 코드로 출력되기 때문에, 알 수 없는 캐릭터가 표시될 것입니다. 변수의 내용을 보고 싶다면 반드시 DEC 또는 HEX 를 사용해야 만 합니다. 아래의 예처럼 DEC 나 HEX 뒤에 ?를 기호를 적어준다면 변수명과 함께 표시됩니다.

```
Const Device = CB280
Dim A As Integer
A = 123
Debug Dec? A,Cr
Debug Hex? A,Cr
```



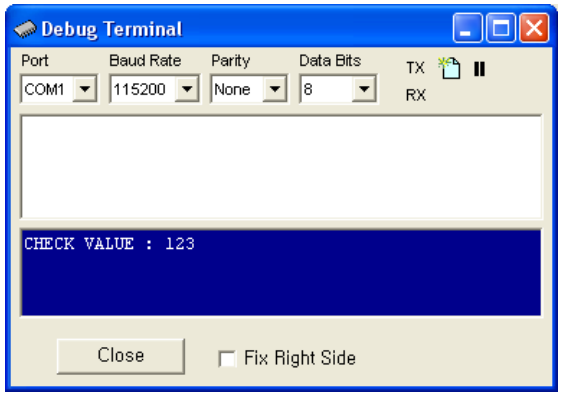
숫자를 사용하여 자릿수를 제한하는 방법도 있습니다.

```
Const Device = CB280
Dim A As Integer
A = 123567
Debug Dec8 A,Cr
Debug Hex8 A,Cr
```



문자열을 같이 출력할 수 도 있고, 줄 바꿈 문자를 같이 출력할 수도 있습니다.

```
Const Device = CB280
Dim A As Integer
A = 123
Debug "Check Value : ",Dec A
```



이와 같이 DEBUG 에서는 문자열 처리함수와 형식변환자를 사용하여 원하는 문자열을 디버그 창에 표시할 수 있는 명령입니다. CUBLOC BASIC 실행도중 DEBUG 명령을 만나면 무조건 DEBUG 창에 그 내용이 표시되는 것입니다.

따라서, 프로그램 중간에 DEBUG 명령을 끼워 넣어서 실행시킨 뒤, 화면에 DEBUG 수행 결과가 표시되었다면, 적어도 그 부분까지는 실행이 되었다는 것을 증명하는 것입니다. 만약 프로그램 버그 등으로 인해 무한루프에 빠지게 된다면, 실행이 미치지 않는 부분이 생길 것입니다. 그리고 변수의 값을 보면서 프로그래머가 원하는 데로 프로그램이 동작하고 있는지의 여부도 확인할 수 있습니다.

디버그 창에서 위에 있는 흰색 창 부분에 문자를 입력하면 CUBLOC 다운로드 포트로 입력한 캐릭터의 ASCII 코드가 전송됩니다. 이 기능은 Debug 명령과는 관계가 없지만, 디버그 터미널에서 데이터를 입력하고 싶을 때 사용할 수 있는 기능입니다.

주의사항

LADDER 에서 MONITORING 을 사용할 때에는 BASIC 의 DEBUG 를 사용할 수 없습니다. 반대로 DEBUG 를 사용 중에는 MONITORING 을 사용할 수 없습니다.

다음은 DEBUG 명령과 함께 사용할 수 있는 제어코드입니다. CUBLOC STUDIO 에 있는 DEBUG 터미널에서의 표시위치를 조정하거나, 화면을 클리어 하는 등의 기능을 가지고 있습니다.

제어코드	코드	설명	사용 예
CLR	0	DEBUG 스크린 클리어	Debug CLR
HOME	1	DEBUG 스크린의 최상위 왼쪽 끝으로 커서이동	Debug HOME
GOXY	2	커서를 원하는 좌표로 이동	Debug GOXY, 4, 3
CSLE	3	커서를 왼쪽으로 한칸이동	
CSRI	4	커서를 오른쪽으로 한칸이동	
CSUP	5	커서를 위로 한칸이동	
CSDN	6	커서를 아래로 한칸이동	
BELL	7	벨소리	
BKSP	8	BACK SPACE	
LF	10	LINE FEED	Debug “ABC”,LF
CLRRI	11	커서의 오른쪽을 모두 지움 (라인 끝까지)	
CLRDN	12	커서의 아래쪽을 모두 지움	
CR	13, 10	캐리지 리턴	Debug, “ABC”,CR

이 제어코드는 DEBUG 명령과 함께 사용해야 합니다.

```
DEBUG GOXY,5,5,DEC I
DEBUG CLR,”TEST PROGRAM”
```

Set Debug

SET DEBUG

DEBUG 명령을 활성화 또는 비활성화 하는 명령입니다. 디폴트 상태에서는 활성화 되어 있습니다. 보통 CUBLOC 에서는 DEBUG 명령을 프로그램 중간에 끼워 넣는 방식으로 디버깅을 합니다. 프로그램이 완성되어, 디버깅이 필요 없는 경우 DEBUG 명령을 끼워 넣은 곳을 전부 찾아 다니면서 지워야 하는 불편함이 있습니다. 이 경우 프로그램 맨 앞줄에 SET DEBUG OFF 라는 명령만 넣어준다면 프로그램 내에서 사용된 모든 DEBUG 명령이 컴파일 되지 않습니다. 프로그램 메모리 공간도 차지 하지 않으므로 실제로 DEBUG 명령을 제거한 것과 같은 효과를 가지고 있습니다.

```
SET DEBUG OFF  ‘DEBUG 명령을 번역하지 않습니다.
```

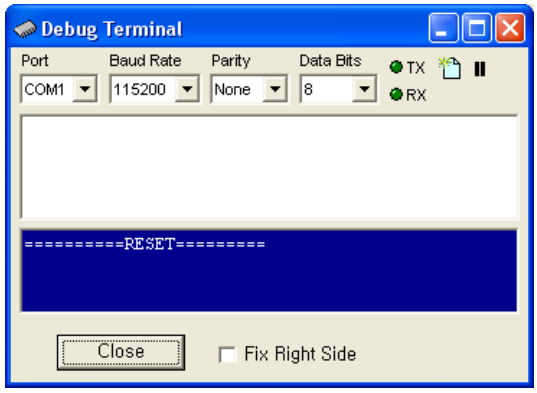
DEBUG 명령 활용법

Debug 명령이 가지고 있는 기본적인 기능, 즉 "실행 중 변수 값을 확인"하는 기능 이외에도 활용하기에 따라서, 다양한 용도로 활용할 수 있습니다. Debug 명령은 유저 여러분께서 프로그램을 작성하시면서 버그를 잡는데 있어서 많은 도움이 될 것입니다.

1. 칩이 리셋 되는지를 확인하는 방법.

프로그램 제일 첫 부분에 Debug 명령으로 특별한 문자를 표시하도록 해 둔다면, 큐블록이 실행 중 리셋이 걸리는지를 확인할 수 있습니다. 프로그램에서 Return 명령 등을 잘못된 곳에 사용한다면, 리셋이 발생되는데, 이 방법을 사용하면 리셋이 걸렸는지 확실하게 확인해 볼 수 있습니다.

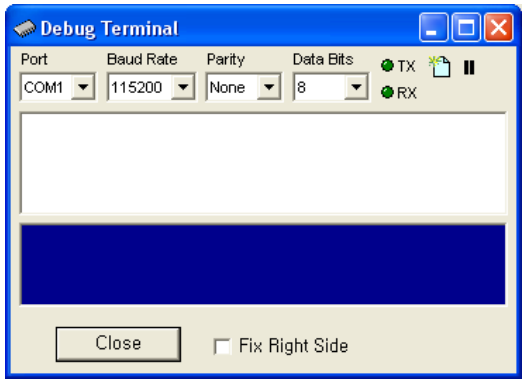
```
Const Device = CB280
Debug "==========Reset========="
Do
      High 0
      Delay 200
      Low 0
      Delay 200
Loop
```



2. 프로그램의 원하는 부분이 실행되었는지 확인하는 방법

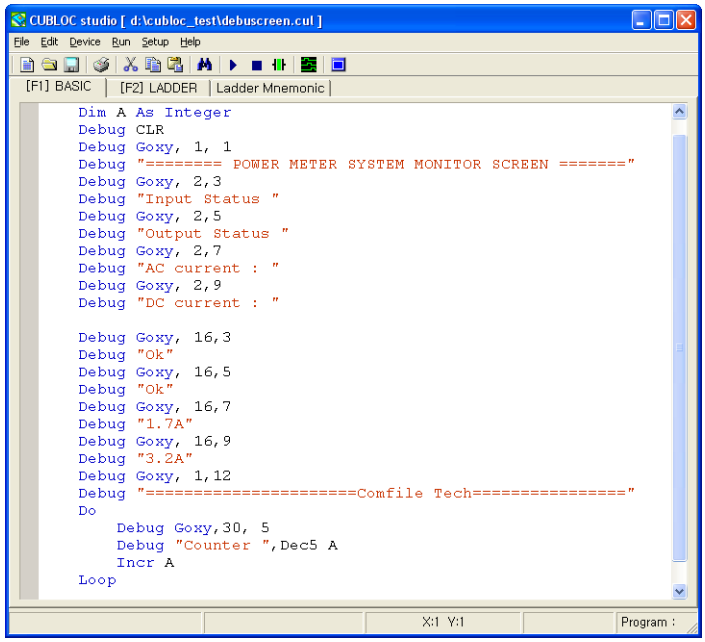
Debug 명령을 특정한 부분에 끼워 넣어서, 그 부분이 실행되고 있는지 확인하는 방법입니다. 만약 실행되지 않는다면 Debug 터미널에 해당문자가 나타나지 않게 됩니다.

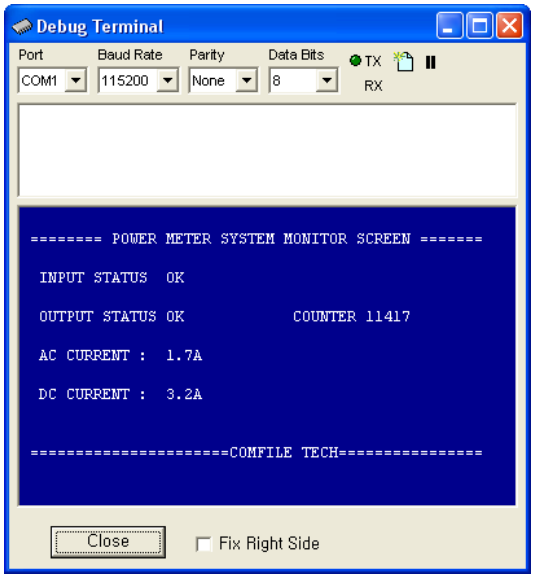
```
Const Device = CB280
Do
      High 0
      Delay 200
      Low 0
      Delay 200
Loop
Debug "Comfile Technology"
```



3. 디버그 터미널을 커다란 LCD로 생각하고 각종 디스플레이 정보를 표시하는 방법

어떤 프로그램을 개발하다 보면, 시스템 운영상태를 LCD 디스플레이 등에 표시하고 싶은 경우가 있습니다. 점검 중 어떤 상태를 표시하고, 실제 운영 시에는 표시하지 않아도 되는 그런 화면을 "시스템 모니터"화면이라고 부르기도 합니다만, 이것을 구현하기 위해서 LCD 를 추가로 연결하지 않고, DEBUG 터미널을 이용하면 됩니다.





Debug CLR 이라고 명령하면, Debug 화면이 모두 클리어 됩니다. Debug Goxy, 3, 2 와 같은 명령은 Debug 화면에서 표시위치를 정하는 명령입니다.

Debug 명령을 모두 지우고 싶다면 소스 프로그램 맨 처음에 Set Debug Off 명령을 써주면 됩니다.

즉, 개발할 때에는 Debug 터미널을 또 하나의 정보 창으로 이용하다가, 개발이 끝나면 Set Debug Off 를 하면 됩니다.

Rnd( )

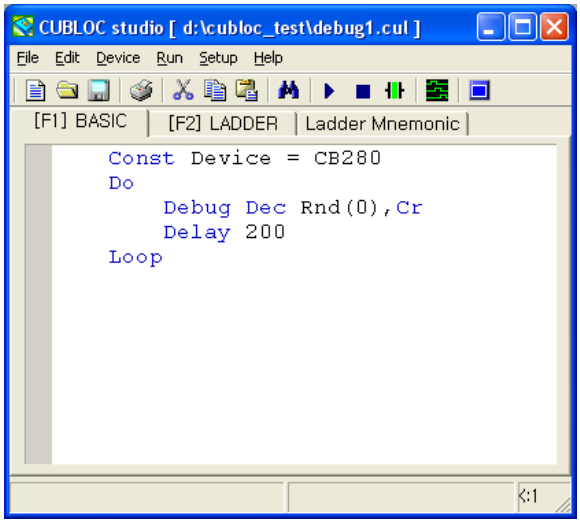
Variable = RND(0)

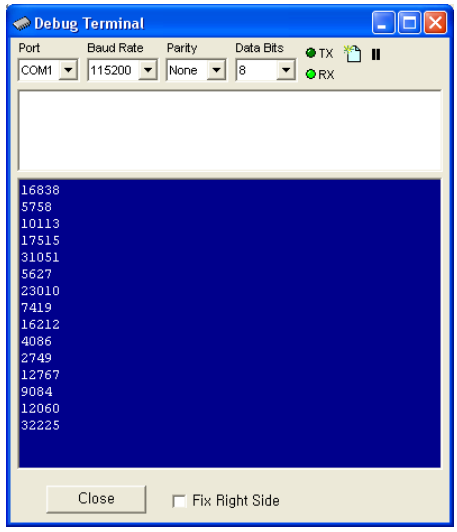
난수를 발생시키는 명령입니다. 0~65535 사이의 난수를 발생시켜 변수 variable 에 저장합니다. 괄호 안의 숫자는 아무 의미도 없는 더미(Dummy) 숫자입니다.

```
DIM A AS INTEGER
A = RND(0)
```

NOTE

실제로 CUBLOC 에서 발생시키는 난수는 "의사난수(Pseudo Random)"입니다. 그 전에 발생되었던 난수의 값으로 다음 난수를 계산해내는 방식입니다. 전원을 껐다 켜면 항상 같은 패턴으로 난수가 나오게 됩니다. 다음에 어떤 수가 나올지 예측할 수 없기 때문에 난수라고 말할 수 있지만, 프로그램초기화 이후에 동일한 패턴으로 수가 발생되므로, 진정한 의미의 난수는 아닙니다.





Reset

Reset

강제로 소프트 리셋 시키는 명령입니다. 프로그램의 가장 처음부터 다시 실행합니다. (Cubloc Studio V1.4.F 부터 사용가능)

```
Reset
```

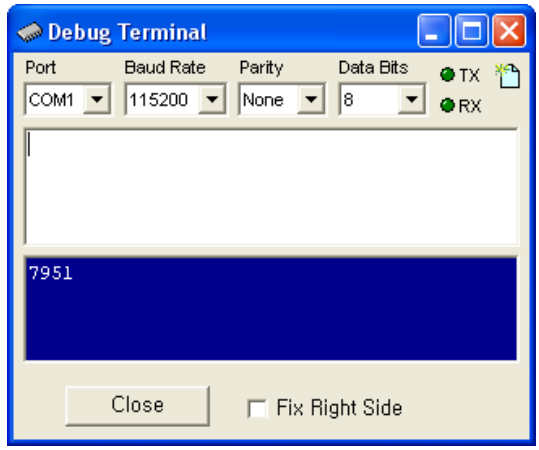
Ramclear

RAMCLEAR

CUBLOC BASIC 의 RAM 을 클리어 합니다. BASIC 의 데이터 메모리는 파워 온 시 자동적으로 클리어 되지 않으므로, 의도하지 않은 엉뚱한 쓰레기 값이 초기치로 설정될 수 있습니다. 이를 방지하기 위해서 일부러 0 을 대입해주거나, Ramclear 명령으로 전체 메모리를 클리어 해 줍니다.

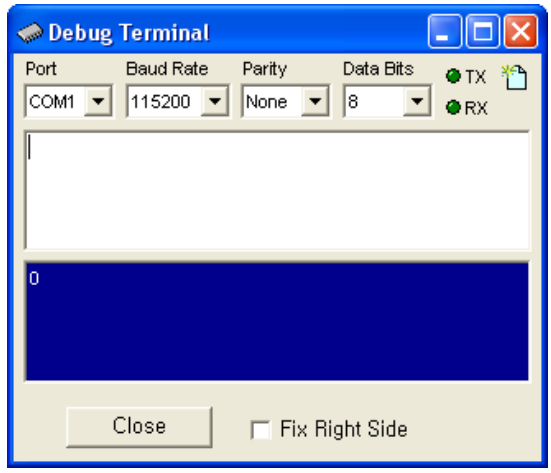
*밧데리 백업을 지원하는 큐블록 코어모듈이 있습니다. 이 모듈에서 Ramclear 를 하지 않는다면, 전원 Off 전 상황을 그대로 유지하게 됩니다. EEPROM 을 Write 횟수에 제한이 있으므로 자주 변경되는 값은 백업된 데이터 메모리 쪽을 사용해야 합니다.

```
Const Device = CB280
Dim i As Integer
Debug Dec i
```



이 경우, 별도의 초기화를 하지 않았기 때문에 디버그 터미널에는 엉뚱한 값이 표시됩니다. 이 값은 SRAM 상에 남아있던 값이기 때문에, 어떤 값이 저장되어 있을지 전혀 알 수 없습니다. 이런 값을 쓰레기값(Garbage)이라고 합니다.

```
Const Device = CB280
Dim i As Integer
Ramclear
Debug Dec i
```



Ramclear 명령으로, 클리어 한 경우입니다. 디버그 터미널에 0 이 표시된 것을 볼 수 있습니다.

리얼타임 관련 명령

초,분,시와 같은 실시간(Real Time)을 다루는 명령입니다. 큐블록중에서는 RTC 칩을 내장한 모델 (CB290, CT17XX)와 내장하지 않은 모델이 있습니다.
RTC 칩이 없는 모델에서는 큐블록 내부의 시스템 클록 RTC 로부터 시간정보 (초,분,시)를 읽어오거나, 재기입할 수 있습니다. 이를 위해서 Time 함수와 Timeset 명령어를 사용합니다.

RTC 칩이란..

RTC 칩이란 Real Time Clock 칩이란 뜻으로, 우리들이 일상생활에서 사용하고 있는 시계와 같다고 볼 수 있습니다. RTC 칩에 있는 시간정보는 자동적으로 갱신되며, 년/ 월/ 일/ 시/ 분/ 초의 내용을 모두 포함하고 있습니다. 또한 밧데리 (또는 슈퍼콘덴서)를 연결했을 경우에는 큐블록 코어모듈의 전원공급이 중단된 상태에서도 계속해서 시간정보는 갱신됩니다. 따라서 전원 ON, OFF 상황과 관계없이 시간에 정보를 얻을 수 있습니다.

Time( )

Variable = TIME (address)

```
Variable : 결과를 저장할 변수 (0~6 까지 사용가능)
address : 시간정보가 기억되어 있는 주소
```

0 에서 6 번지까지는 RTC 칩으로부터 정보를 읽어오는 영역입니다. 따라서 RTC 칩이 내장된 코어모듈에서만 사용할 수 있습니다. 어드레스에 따라 다른 정보가 들어있으며, 정보의 내용은 BCD 코드형태로 기록되어 있습니다.

번지	내용	범위	비트구성					
0	초	0~59		10 자릿수			10 이하 자릿수	
1	분	0~59		10 자릿수			10 이하 자릿수	
2	시	0~23			10 자릿수		10 이하 자릿수	
3	일	01~31			10 자릿수		10 이하 자릿수	
4	요일	0~6						10 이하 자릿수
5	월	1~12				10	10 이하 자릿수	
6	년도	00~99	10 자릿수				10 이하 자릿수	

요일의 경우 다음 표를 참조하십시오.

일요일	0
월요일	1
화요일	2
수요일	3
목요일	4
금요일	5
토요일	6

시스템 클록 RTC

RTC 칩이 없는 모델에서는 시스템클록 RTC 로부터 실시간정보를 얻게됩니다. TIME, TIMESET 명령어로 억세스 할 수 있으며, 아래 표와 같이 10 번지 이후를 사용합니다.

번지	내용	범위
10	초	0~59
11	분	0~59
12	시	0~65535
13	연속된 초	0~65535

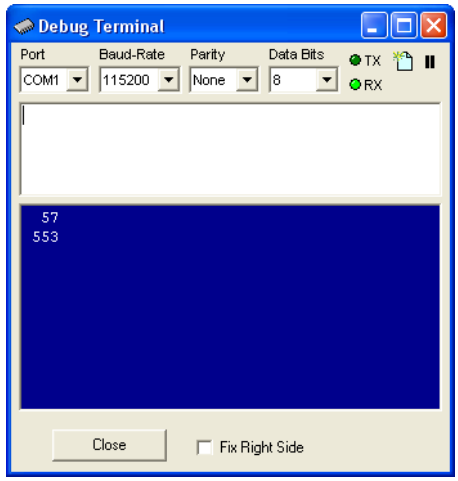
10 (초)번지는 1 초마다 1 씩 증가되어, 60 이 되면 11 (분)번지가 1 증가됩니다. 이후 60 분이 되면 12(시)번지가 1 증가됩니다. 12(시)번지는 65535 까지 계속 증가되다가, 65535 이후에는 다시 0 이 됩니다. 이 번지의 값은 파워온시 0 이 되며, TIMESET 명령을 사용해서 다른 값을 써넣을수도 있습니다. 이 번지의 내용은 RTC 칩과는 달리, 전원 OFF 상황에서 계속 갱신되지 않습니다.

또한 시스템클록 RTC (번지 10~13) 의 내용은 앞에서 설명한 번지의 내용처럼 BCD 코드로 기록되어 있지 않고, 바이너리값 그대로 기록되어 집니다. 따라서 유저는 BCD to Binary 변환을 할 필요가 없습니다.

시스템클록 RTC (번지 10~13) 의 내용은 시스템 클록으로부터 갱신되어지는 정보이며, 실제시간과 약간의 오차가 발생할 수 있으며, 앞에서 설명한 RTC 칩의 내용과는 일치하지 않습니다.

다음은 샘플 프로그램입니다.

```
Const Device = CB405
Dim i As Integer
Cls
Timeset 10,58
Timeset 13,254
Do
    i = Time(10)
    Debug Goxy,0,0,dec4 i,Cr
    Debug Goxy,0,1,dec4 Time(13)
    Delay 100
Loop
```



13 번지는 10 번지와 같이 1 초마다 1 씩 증가됩니다. 10 번지는 60 초가 되면다시 0 으로 리셋되지만, 13 번지는 65535 가 될때까지 계속 증가됩니다. 65535 에서 1 초가 더 지나면, 다시 0 이 됩니다. 연속적인 경과시간을 초단위로 알고 싶을 때 사용합니다.

Timeset

TIMESET address, value

address : 시간정보가 기억되어 있는 주소 (0~6 까지 사용가능)

value : 시간정보 값

RTC 칩에 새로운 정보를 기입하는 명령입니다. 시간을 다시 설정할 때 필요한 명령입니다. 초기 출하상태에서는 알 수 없는 값이 들어있습니다.

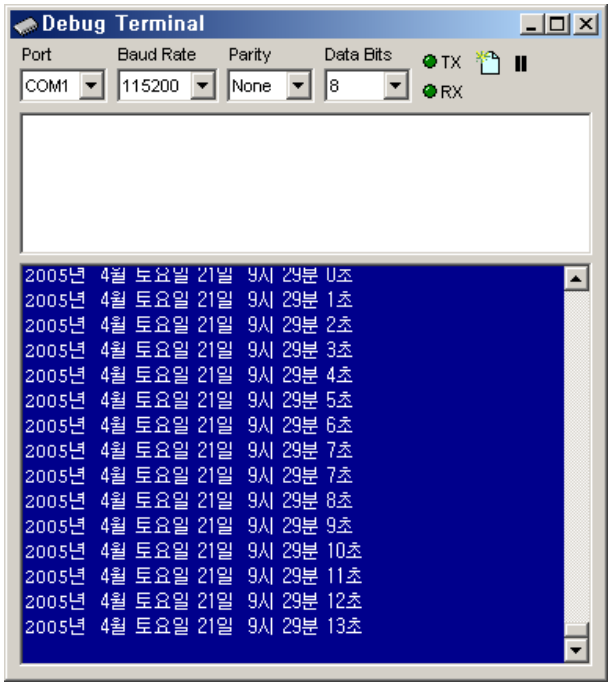
번지	내용	범위	비트구성					
0	초	0~59		10 자릿수			10 이하 자릿수	
1	분	0~59		10 자릿수			10 이하 자릿수	
2	시	0~23			10 자릿수		10 이하 자릿수	
3	일	01~31			10 자릿수		10 이하 자릿수	
4	요일	0~6						10 이하 자릿수
5	월	1~12				10	10 이하 자릿수	
6	년도	00~99	10 자릿수				10 이하 자릿수	

다음은 Timeset 과 Time ( ) 명령을 사용해서 새로운 시간정보를 셋팅하고, 그 내용을 표시하는 샘플 프로그램입니다.

```
Const Device =CT1720
Dim I As Byte
Timeset 0,0      '초
Timeset 1,&H24   '분
Timeset 2,&H9    '시
Timeset 3,&H21   '일
Timeset 4,&H6    '요일
Timeset 5,&H4    '월
Timeset 6,&H5    '연도

Do
      I = Time(6)
      Debug "200",Hex I,"년 "
      I = Time(5)
      Debug Hex2 I,"월 "
      I = Time(4)
      Select Case I
      Case 0
            Debug "일요일 "
      Case 1
            Debug "월요일 "
      Case 2
            Debug "화요일 "
      Case 3
            Debug "수요일 "
      Case 4
```

```
            Debug "목요일 "
        Case 5
            Debug "금요일 "
        Case 6
            Debug "토요일 "
        End Select
        I = Time(3)
        Debug Hex2 I,"일 "
        I = Time(2)
        Debug Hex2 I,"시 "
        I = Time(1)
        Debug Hex2 I,"분 "
        I = Time(0)
        Debug Hex I,"초",cr
        Delay 1000
    Loop
```

디버그 창에 다음과 같은 결과가 표시됩니다.



주의사항!

RTC 칩에서 읽은 데이터는 주변온도 및 전압조건에 따라 하루에 약 +/- 10 초 또는 그 이상의 오차가 발생할 수 있습니다. 따라서 주기적으로 현재 시간으로 맞추어주는 것이 좋습니다. CB290 의 경우 별도의 백업밧데리 (또는 슈퍼콘덴서)가 없다면, 전원 OFF 시 현재 시간을 더 이상 유지하지 않고, 엉뚱한 값으로 초기화될 수 있습니다.

Freqout

FREQOUT Channel, FreqValue

```
Channel : PWM 채널 (I/O 핀 번호가 아님)를 가리키는 변수 또는 상수
FreqValue : 주파수 파라메터, 1~65535 까지 입력가능
```

이 명령은 특정 주파수의 펄스를 지속적으로 출력하도록 하는 명령으로, 큐블록의 PWM 채널을 사용하는 명령입니다. 해당 채널에 스피커를 연결한 뒤 이 명령을 사용해서 음을 발생시킬 수도 있고, 스테핑 모터 / 서보모터 콘트롤을 위한 파형을 발생시킬 수도 있습니다. 이 명령으로 발생시킨 펄스는 큐블록의 하드웨어에 의해서 백그라운드에서 실행되므로, 사용자는 Freqout 명령 수행 후, 곧바로 다른 명령어를 수행할 수 있습니다.

PWM 명령에서는 포트번호가 아닌 채널번호를 사용하므로, 모델 별 채널에 따른 포트위치를 확인하시기 바랍니다. CB220, CB280, CB320, CB380 의 경우 포트 5,6,7 에 PWM 채널 0,1,2 가 할당되어 있습니다.

FreqValue 값에 따라 발생 주파수를 변경할 수 있습니다. 다음 표는 FreqValue 값에 따른 주파수 변경 표입니다. 1 이면 가장 높은 주파수, 65535 일 경우 가장 낮은 주파수가 발생됩니다. 0 사용시 파형이 발생되지 않습니다.

FreqValue	주파수
1	1152 KHz
2	768 kHz
3	576 KHz
4	460.8KHz
5	384 KHz
10	209.3 KHz
20	109.7 KHz
30	74.4 KHz
100	22.83 KHz

FreqValue	주파수
200	11.52 KHz
1000	2.3 KHz
2000	1.15 KHz
3000	768 Hz
4000	576 Hz
10000	230 Hz
20000	115.2 Hz
30000	76.8 Hz
65535	35.16 Hz

FreqValue 값이 100 이상일 경우 다음 공식에 의해 주파수를 계산할 수 있습니다.

```
주파수 = 2304000 / FreqValue
```

Freqout 명령을 사용하기 전에, 해당 포트를 Output 상태로 만들어 주어야 합니다. 주파수 발생을 중단하고 싶다면, PWMOFF 명령을 사용하시기 바랍니다.

다음은 샘플프로그램입니다.

```
Const Device = cb280
Dim i As Integer
Low 5                   ' 해당 채널을 Output 상태로 만든다.
i = 1
Freqout 0,10            ' 209.3Khz 의 파형을 발생
Do                      ' 무한루프
Loop
```

Freqout 명령은 PWM 자원을 사용하는 것이므로, 몇 가지 제약사항이 있습니다. PWM 0,1,2 채널은 하나의 타이머로 동작하기 때문에, 만약 채널 0 에서 Freqout 을 사용했다면, 채널 0 은 물론이고, 채널 1, 2 에서도 PWM 명령을 사용할 수 없습니다.

마찬가지로 PWM 채널 3,4,5 도 하나의 타이머로 되어 있어, 채널 3 에서 Freqout 을 사용했다면 채널 3,4,5 모두에서 PWM 명령을 사용할 수 없습니다.

PWM 채널 0 과 채널 3 에서 각각 다른 주파수를 발생시키는 것은 가능합니다.

정리하면, 큐블록 에서 Freqout 은 두 개의 다른 주파수를 발생시킬 수 있으며, Freqout 명령을 사용할 때 PWM 명령사용은 제한 받게 됩니다.

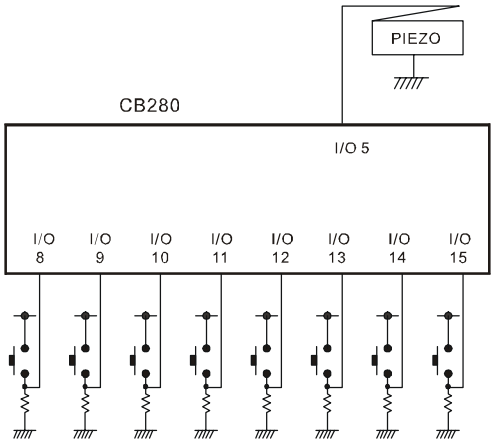
다음은 음계에 따른 주파수를 Freqout 에서 사용할 수 있도록 변환한 표입니다. Freqout 명령으로 음을 연주하고자 할 때 아래 표를 참고하시기 바랍니다.

음계	음계(한글)	2 옥타브	3 옥타브	4 옥타브	5 옥타브
A	라	20945	10473	5236	2618
Bb	시b	19770	9885	4942	2471
B	시	18660	9330	4665	2333
C	도	17613	8806	4403	2202
Db	레b	16624	8312	4156	2078
D	레	15691	7846	3923	1961
Eb	미b	14811	7405	3703	1851
E	미	13979	6990	3495	1747
F	파	13195	6597	3299	1649
Gb	솔b	12454	6227	3114	1557
G	솔	11755	5878	2939	1469
Ab	라b	11095	5548	2774	1387

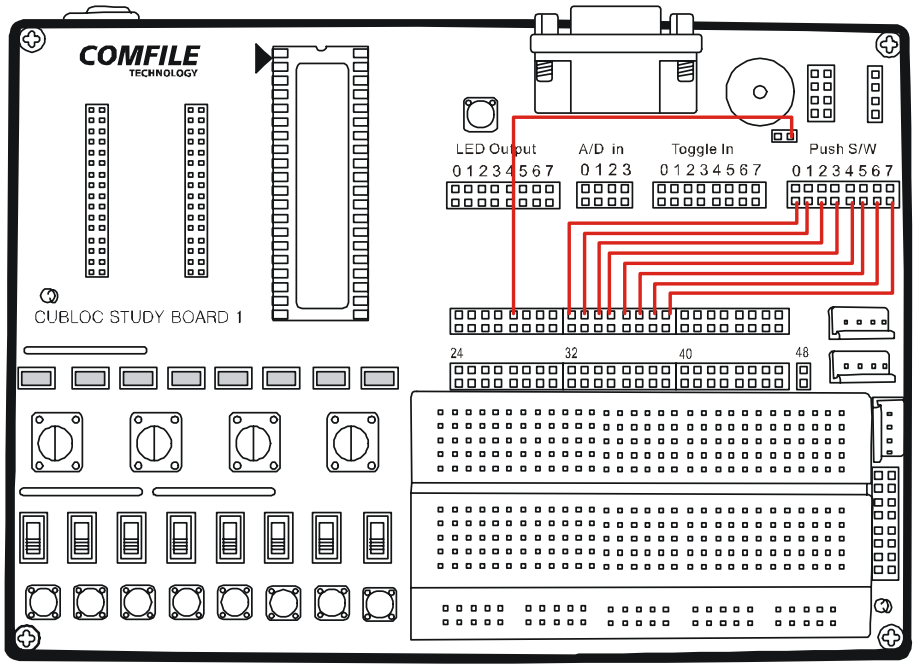
```
Freqout 0,5236          ' 4 옥타브의 A(라)음을 발생 (440Hz)
Freqout 0,1469          ' 5 옥타브의 G(솔)음을 발생
```

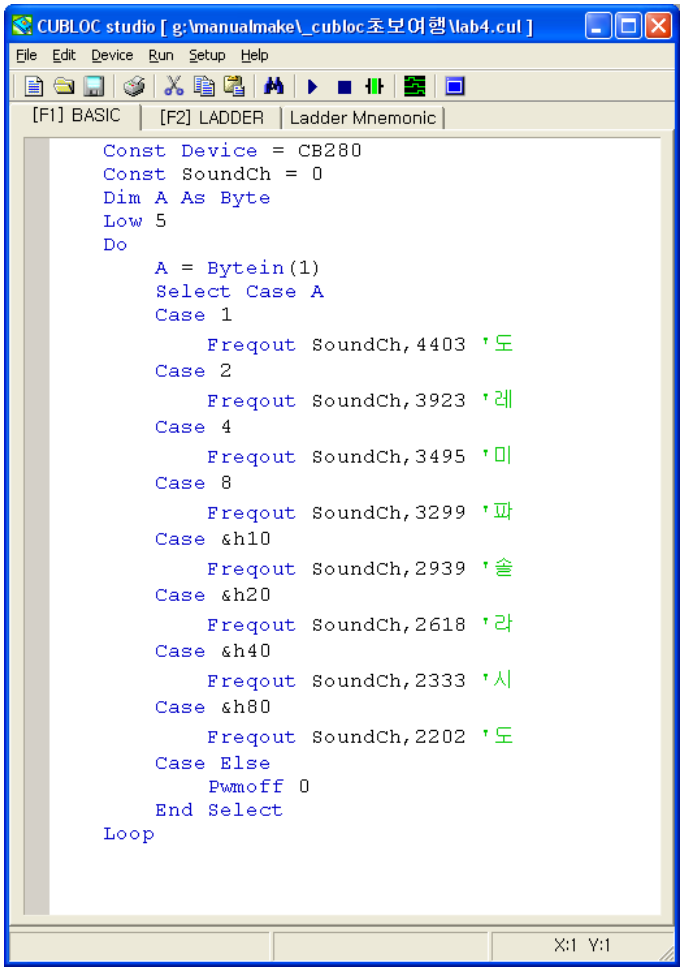
DEMO PROGRAM

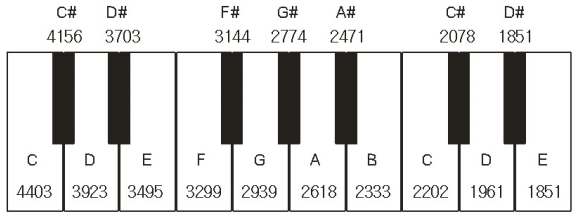
Freqout 명령을 이용해서 만든 피아노입니다. 5 번 포트를 PIEZO 에 연결하고, 8 번부터 15 번포트는 스위치에 연결합니다. 스위치를 누르면 “도, 레, 미, 파..”음을 연주합니다.



CUBLOC STUDY BOARD-1 을 가지고 계시다면, 아래그림을 보고 배선하시기 바랍니다.




CUBLOC studio [g:\manualmake_cubloc초보여행\lab4.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
Const Device = CB280
Const SoundCh = 0
Dim A As Byte
Low 5
Do
A = Bytein(1)
Select Case A
Case 1
Freqout SoundCh,4403 '도
Case 2
Freqout SoundCh,3923 '레
Case 4
Freqout SoundCh,3495 '미
Case 8
Freqout SoundCh,3299 '파
Case &h10
Freqout SoundCh,2939 '솔
Case &h20
Freqout SoundCh,2618 '라
Case &h40
Freqout SoundCh,2333 '시
Case &h80
Freqout SoundCh,2202 '도
Case Else
Pwmoff 0
End Select
Loop
X:1 Y:1


C# D# F# G# A# C# D#
4156 3703 3144 2774 2471 2078 1851
C D E F G A B C D E
4403 3923 3495 3299 2939 2618 2333 2202 1961 1851

펄스 출력 명령

지정된 포트로 원하는 개수만큼의 펄스를 출력할 수 있는 명령입니다. FREQOUT 과 PWM 명령은 펄스의 주파수는 제어할 수 있지만, 출력펄스의 수를 제한할 수 없는 명령입니다. 여기에서 소개하는 명령어는 주파수 뿐만 아니라 펄스의 개수도 조정할 수 있고, PWM 전용 포트가 아닌, 일반 포트로도 펄스를 출력할 수 있는 기능을 가지고 있습니다.

Steppulse

STEPPULSE Channel, Port, Freq, Qty

```
Channel : StepPulse 를 위한 채널 (0 또는 1)
Port : 출력가능한 포트 번호 (입력전용 포트는 사용할 수 없음)
Freq : 출력주파수 (최대 15000 까지 사용가능)
Qty : 펄스 출력 개수 (2147483647 개까지)
```

코어모듈에 따라 사용할 수 있는 채널의 차이가 있습니다.

코어모듈명	채널수	채널	사용할수 없는 PWM 채널
CB220, 280, 290, CT172X/C CB320, 380	1 개	0	채널 0 은 3, 4, 5
CB405	2 개	0 또는 1	채널 0 은 3, 4, 5 채널 1 은 6, 7, 8

이 명령은 큐블록 코어내부에 있는 PWM 용 카운터를 자원으로 사용합니다. 따라서 채널 0 을 사용할 때 PWM3, 4, 5 를 사용할 수 없습니다. CB405 의 경우 채널 1 을 사용할 때, PWM6, 7, 8 을 사용할 수 없습니다.

CB2XX, CB3XX 시리즈의 경우 채널 0 만 사용가능하며, CB405 의 경우 2 개의 채널을 동시에 사용가능합니다.

펄스는 출력가능한 포트중 아무포트에서나 사용가능합니다. 일단 STEPPULSE 명령이 실행되면, 해당 포트는 출력상태가 됩니다. 펄스 출력이 모두 끝난후에도 출력상태를 계속 유지합니다.

출력 주파수는 HZ 를 의미합니다. 1 을 입력하면 초당 1 개의 펄스가 나가는 것이고, 1000 을 입력하면 초당 1000 개의 펄스가 나가는 것으로 즉, 1KHz 의 펄스가 출력됩니다. 큐블록에서는 최대 15KHz 까지 가능하므로, 15000 까지 사용가능합니다. 이 이상의 숫자를 쓰면, 자동적으로 15000 으로 조정됩니다.

펄스 출력갯수는 펄스의 개수를 의미합니다. 1 로 적으면 단 한 개의 펄스가 출력됩니다. 1000 개로 적으면 1000 개의 펄스가 정해진 주파수 (속도)로 출력됩니다. 최대 2147483647 개까지 출력할 수 있습니다.

펄스의 출력이 모두 끝날때까지 대기하지 않고, STEPPULSE 명령은 바로 수행을 종료합니다. 즉, 백그라운드에서 자동적으로 펄스를 출력합니다. 펄스가 출력되는동안 다른 명령을 수행할 수 있다는 뜻입니다.

Stepstop

STEPSTOP Channel

Channel : StepPulse 를 위한 채널 (0 또는 1)

출력중인 펄스를 중단시키는 명령입니다. 채널번호만 써주면 StepPulse 에서 지정한 포트의 펄스출력이 중단됩니다.

Stepstat( )

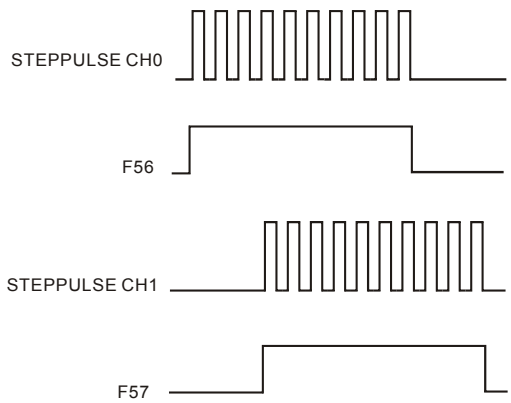
Variable = STEPSTAT (Channel)

Variable : 결과를 저장할 변수 (0~6 까지 사용가능)

Channel : StepPulse 를 위한 채널 (0 또는 1)

출력중인 펄스의 개수의 2 배수를 반환하는 함수 입니다. 500 개의 펄스출력이 남아있다면 1000 을 반환합니다. 펄스 출력이 모두 종료되었다면 0 을 리턴합니다.

펄스 출력상태를 레더로직의 특수릴레이로도 확인할 수 있습니다. 채널 0 이 출력중일때는 _F(56)이 1 이 됩니다. 채널 1 이 출력중일때는 F(57)이 1 이 됩니다.StepStop 에 의해서 멈추거나, 정해진 펄스가 모두 출력되어서 멈춘 뒤에는 자동적으로 0 이 됩니다.

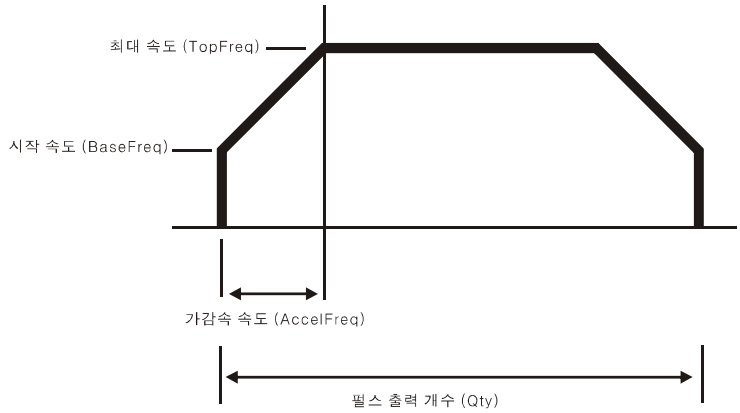


Stepaccel

STEPACCEL Channel, Port, BaseFreq, TopFreq, AccelFreq, Qty

```
Channel : StepPulse 를 위한 채널 (0 만 사용가능)
Port : 출력가능한 포트 번호 (입력전용 포트는 사용할 수 없음)
BaseFreq : 시작주파수 (최대주파수보다는 작은값으로)
TopFreq : 최대주파수 (최대 3300 까지 사용가능)
AccelFreq : 가감속 속도
Qty : 펄스 출력 개수 (2147483647 개까지)
```

```
예) Stepaccel 0, 8, 10, 5000, 400, 1000      ' <-- 8 번 포트로 펄스를 1000 개를
                                              ' 가감속파형으로 출력합니다.
```



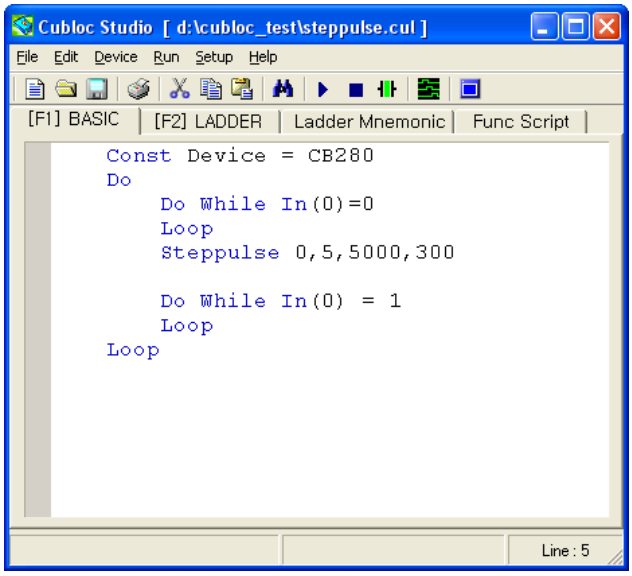
Stepaccel 명령어는 가감속 패턴형태로 파형을 발생시키는 명령입니다. 주로 스탭모터 구동용 펄스를 만들때 사용합니다. 이 명령은 모든 CUBLOC 에서 채널 0 만 사용가능합니다. 이 명령사용시 PWM3,4,5 을 사용할 수 없습니다. 채널은 1 개만 사용가능하지만 출력포트를 바꿀수 있기 때문에, 여러개의 모터를 연결할 수 있습니다.

동시에 모터를 구동시킬 수는 없지만, 여러개의 모터를 순차적으로 구동시키는 것은 가능합니다. 입력전용포트를 제외한 모든 출력포트를 이용할 수 있습니다.

파라메터중 AccelFreq 가 가감속 속도를 의미합니다. 이 숫자가 클수록 가감속시간이 줄어듭니다. BaseFreq 보다는 2 배이상 큰수를 사용하십시오. 다음은 Stepaccel 을 사용한 예제 소스입니다. 8 번 포트로 스탭출력을 합니다.

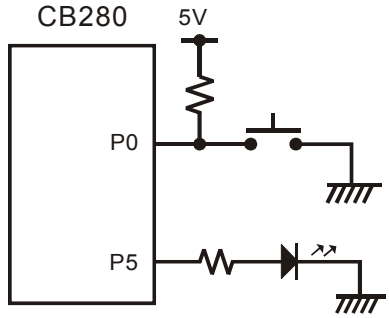
```
Const Device = CT1720
Do
    Wait 200
    Stepaccel 0,8,10,4000,4400,30000
    Do While Stepstat(0) > 0
    Loop
    Wait 1500
Loop
```

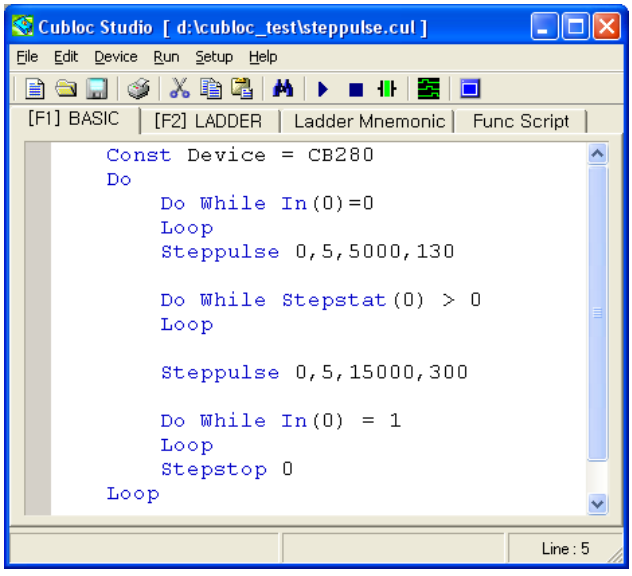
DEMO PROGRAM



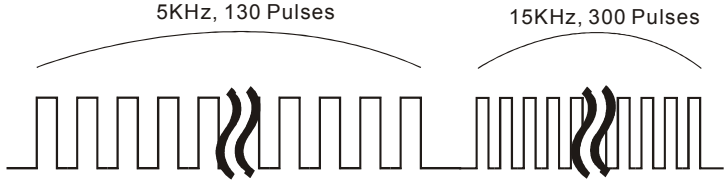
포트 0 에 연결된 스위치를 누르면, 포트 5 번에 5KHz 의 속도로 300 개의 펄스를 출력합니다.

회로는 다음과 같습니다. 포트 5 번에는 편의상 LED 를 연결해 놓았습니다.

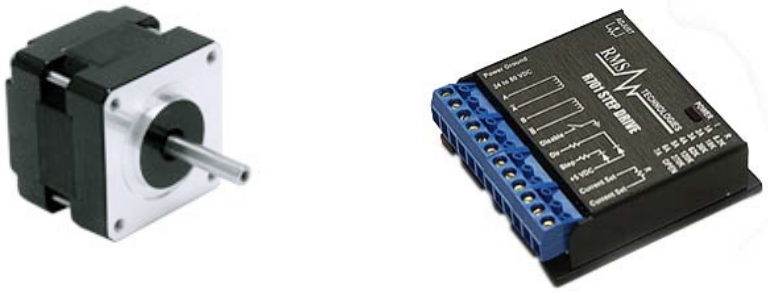


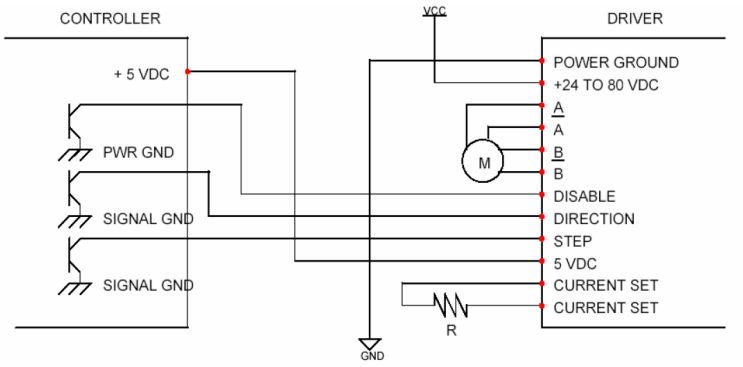


포트 0 에 연결된 스위치를 누르면, 포트 5 번에 우선 5KHz 의 속도로 130 개의 펄스가 출력되고, 이어서 15KHz 의 속도로 300 개의 펄스가 출력됩니다.



아래 그림과 같은 STEP 모터와 "모터 드라이버"를 연결하여 모터를 구동시킬 수 있습니다.





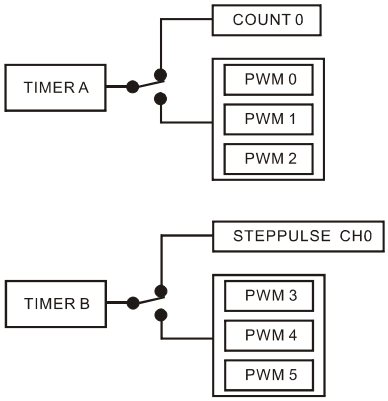
큐블록과 드라이버는 3 개의 I/O 로 연결됩니다. 이중 DISABLE 은 모터회전을 금지시키는 단순한 기능입니다. DIRECTION 은 모터의 회전방향을 결정합니다. (예를 들면 HIGH 이면 시계방향, LOW 이면 시계반대방향)

STEP 에 펄스를 공급해주면 펄스의 개수만큼 모터가 회전합니다. 1 펄스에 1.8 도가 움직이는 모터라면 10 개의 펄스에 18 도가 움직입니다.

사용상의 제한이 있는 기능

CT172X/C 메인 칩의 제한된 주변장치로 인해, COUNT 와 PWM, STEPPULSE 명령사용시 약간의 제약사항이 있습니다. 각각의 명령을 해설하는 부분에서도 이미 설명해놓은 부분입니다만, 이곳에서 전체적으로 정리해서 설명드리도록 하겠습니다.

CT172X/C 메인칩은 2 개의 내부 타이머를 가지고 있습니다. 이중 하나 (TIMER A)는 고속카운터 채널 0 에서 사용하거나, PWM0,1,2 에서 사용할 수 있습니다. 둘중 하나를 선택해야 합니다. 파워온시에는 PWM0,1,2 로 선택되어 있습니다. SET COUNT0 ON 명령을 쓰면, 고속카운터 채널 0 이 활성화되고, PWM0,1,2 는 비활성화됩니다.



또 다른 타이머 (TIMER B)는 PWM 3,4,5 또는 STEPPULSE 채널 0 에서 사용할 수 있습니다. 역시 둘중 하나만 사용가능합니다. 파워온시에는 PWM 3,4,5 로 선택되어 있습니다. 별도의 설정명령어 없이 STEPPULSE 명령어를 쓰는 순간 PWM 3, 4, 5 는 비활성화 됩니다. 따라서 STEPPULSE 명령 사용전에 PWM 3,4,5 는 반드시 OFF 상태이어야 합니다.

STEPPULSE 명령의 실행이 종료된 뒤에 PWM 3,4,5 를 사용할 수 있습니다. 동시에 사용할 수 없을뿐, 순차적으로 사용하는 것은 가능하다는 뜻입니다.

Wait

Wait value

밀리 초단위로 시간지연시키는 명령어입니다. 내부의 시스템 클록을 사용하므로 정확한 시간을 딜레이 시켜줍니다. 단, 10 밀리 초간격으로 인식하고 동작합니다. 즉 Wait 15 라고 하면 15 밀리초를 딜레이하지 않고, 10 밀리초를 딜레이합니다.

내부의 시스템 클록이 10 밀리초마다 한번씩 갱신되기 때문입니다. 이 명령은 기존 Delay 명령을 대신해서 사용할 수 있습니다.

Delay 명령은 SUB 형 라이브러리로 되어 있고, For..next 루프를 사용해서 딜레이를 하기 때문에, 정확한 시간을 예측할 수 없습니다. DELAY 100 을 하면 100 밀리 초와 비슷한 범위에서 딜레이 합니다.

DELAY 명령을 수행하때마다 딜레이시간이 달라지지는 않습니다만, 정확한 시간을 예측할수 없으므로 불편한 경우가 있었습니다. 앞으로는 wait 명령으로 시간지연을 하시기 바랍니다 .이 명령은 CUBLOC STUDIO V2.0.H 이후부터 사용가능합니다.

최대 딜레이 가능시간은 (LONG 형 변수까지 사용가능) 10 부터 2147483640 밀리초입니다.

```
Wait 10         ' 정확히 10 밀리 초를 딜레이 합니다.
Wait 200        ' 정확히 200 밀리 초를 딜레이 합니다.
Wait 2000       ' 정확히 2 초를 딜레이 합니다.
```

제 8 장
CUBLOC BASIC
통신관련기능

RS232 통신

CUBLOC 에는 최대 4 채널의 RS232C 가 내장되어 있으며 이중 채널 0 은 다운로드/모니터링으로 사용하고, 나머지 1, 2, 3 을 범용통신으로 사용할 수 있습니다. 다음 표는 코어모듈에 따른 지원채널수 입니다.

제품명	사용 가능한 RS232 채널	채널수
CT172X/C	0, 1	2 개

CUBLOC 은 송,수신 버퍼를 사용하므로, BASIC 프로그램이 다른 일을 하는 도중에도 데이터를 수신하거나 송신할 수 있습니다. 수신 버퍼에 데이터가 도착하면 인터럽트를 발생시켜, 특정 루틴을 수행하도록 할 수 있습니다.

Opencom

OPENCOM channel, baudrate, protocol, recvsize, sendsize

```
channel : 사용채널 (0 부터 3)
Baudrate : 보레이트 (반드시 상수로만 사용)
protocol : 프로토콜의 종류
recvsize : 수신용 버퍼의 크기 (최대 1024 까지)
sendsize : 송신용 버퍼의 크기 (최대 1024 까지)
```

RS232 를 사용하기 위해서 반드시 소스 프로그램 초기에서 선언해야 되는 명령입니다. Baudrate 에는 원하는 보레이트를 적어주면 됩니다. CUBLOC 에서 지원 가능한 보레이트는 다음과 같습니다. (아래 적힌 보레이트 이외의 보레이트는 사용할 수 없습니다. 예를들어 10080, 8000, 98400 과 같은 숫자를 보레이트로 사용할 수 없습니다.)

2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400

보레이트는 반드시 상수형태로 사용하여야합니다. 변수 및 배열을 사용할 수 없습니다.

```
DIM BA as Integer
BA = 19200
OPENCOM 1,BA,3,10,10       '보레이트 위치에 변수를 사용할 수 없습니다.
```

protocol 에는 데이터의 형식을 결정하는 코드를 적어줍니다. 한 바이트를 구성하는데 3 개의 필드가 있으며, 3 개의 필드에 대한 설정치는 다음과 같습니다.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			패리티		스톱비트 수	비트	수
			0	0 = NONE	0=1 스톱비트	0	0 = 5 비트
			0	1 = Reserve*	1=2 스톱비트	0	1 = 6 비트
			1	0 = 짝수		1	0 = 7 비트
			1	1 = 홀수		1	1 = 8 비트

*Reserve : 차후에 사용하기 위해서 남겨놓은 부분이므로 사용해서는 안됩니다.

아래 예를 참조하시어 올바른 protocol 을 입력하시기 바랍니다.

비트 수	패리티	스톱비트	입력 값
8	NONE	1	3
8	EVEN	1	19 (16 진수 13)
8	ODD	1	27 (16 진수 1B)
7	NONE	1	2
7	EVEN	1	18 (16 진수 12)
7	ODD	1	26 (16 진수 1A)

```
OPENCOM 1, 19200, 3, 30, 20 '8 비트,NONE 패리티,1 스톱비트로 설정
```

OPENCOM 명령에서 송신용 버퍼와 수신용 버퍼의 크기를 결정할 수 있습니다. 송,수신용 버퍼는 데이터 메모리의 영역을 차지하게 됩니다. 최대 1024 까지 설정할 수 있지만, 너무 많은 영역을 할당하게 되면, 그 만큼 변수영역으로 사용할 수 있는 공간이 줄어들게 됩니다. 수신버퍼는 30~100 이하, 송신버퍼는 30~50 이하로 하면 무난히 사용할 수 있습니다.

Set Rs232

Set Rs232 channel, baudrate, protocol

channel : 사용채널 (0 부터 3)
Baudrate : 보레이트 (상수로만 사용가능)
protocol : 프로토콜의 종류

Opencom 명령은 하나의 소스에서 단 한번밖에 사용할 수 없기 때문에, 한번 설정된 보레이트나 프로토콜을 다른 것으로 바꿀 수 없습니다. Set RS232 명령을 사용하면 Rs232 포트가 Open 된 상태에서 보레이트, 프로토콜을 다른 것으로 바꿀 수 있습니다. 단, 버퍼 설정상태는 바꿀 수 없습니다. Baudrate, Protocol 의 사용법은 Opencom 명령에서의 사용법과 같습니다.

```
Const Device = CB280
Dim A As Integer
Opencom 1,115200,3,10,20
A = 0
Do while A<100              ' 100 까지는 115200 으로 전송하고
      Putstr 1,Dec A,Cr
      Delay 200
      Incr A
Loop

Set Rs232 1,19200,3         ' 보레이트 다시 정의 (버퍼는 그대로 사용)
A = 0
Do while A<100              ' 여기서부터는 19200 으로 전송
      Putstr 1,Dec A,Cr
      Delay 200
      Incr A
Loop
```

Set Rs485

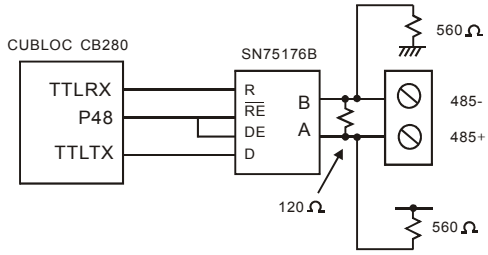
Set Rs485 channel, pin

```
channel : 사용채널 (1 부터 3)
pin : Transmit Enable pin 으로 사용할 i/o 포트 번호
```

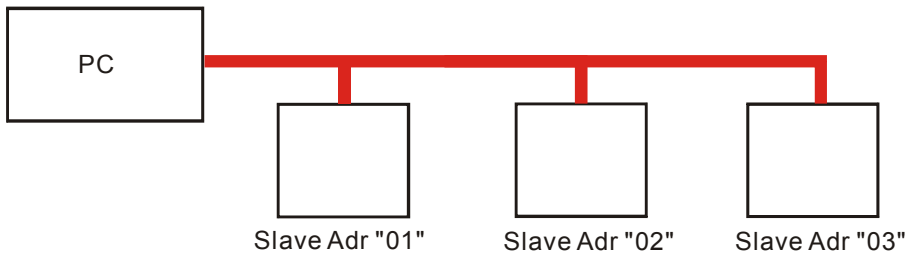
RS485 는 RS232 에 비교해서 송신거리가 길고, 1:N 통신이 가능한 방식입니다. 하나의 마스터에 여러 개의 CUBLOC 을 연결해서, 통신하고자할 때에는 RS485 변환칩으로 회로를 구성해주야 합니다.

RS485 는 단 2 가닥의 통신선을 사용하기 때문에 어느 한순간, 송신 또는 수신만 할 수 있습니다. 즉 "반이중"통신방식입니다. 또한, RS485 통신선에는 GND 선이 없기 때문에, 외부로부터의 노이즈가 메인기판의 GND 선으로 직접 유입되는 것을 막을 수 있습니다.

TTL232 신호를 RS485 신호로 변환하기 위해서 다음과 같은 회로를 사용합니다.



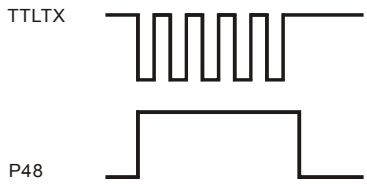
RS485 회로에는 "송신허가"제어선이 필요합니다. RS485 신호선은 여러 개의 디바이스가 공유하고 있습니다. 이중 하나의 디바이스만 송신상태이고 나머지는 수신상태입니다. 아래 그림을 보십시오. 만약 PC 에서 SlaveAdr "01" 디바이스에게 Read 명령을 내렸을 경우, Slave Adr "01" 디바이스에서는 읽어온 값을 PC 로 보내야 합니다. 이를 위해서는 CUBLOC 측에서 송신허가를 해서 "RS485 신호선"을 점유한뒤, 데이터를 송신합니다.



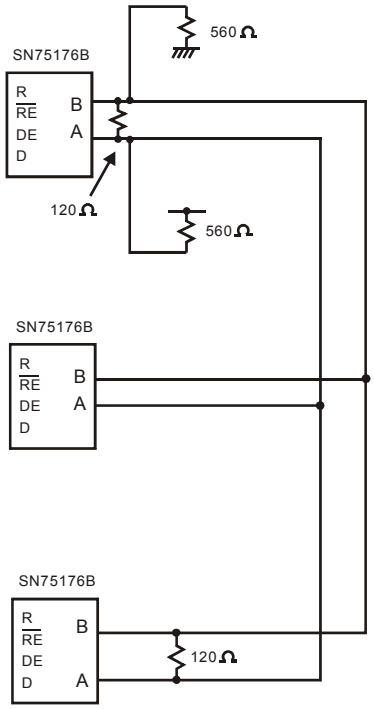
*CT1721C 에는 채널 1 을 RS232 와 RS485 로 선택하여 사용할 수 있도록 되어 있습니다. 그리고 송신허가핀으로 P15 를 사용하고 있습니다. 따라서 RS485 기능을 사용할때에는 P15 를 다른용도로 사용할 수 없습니다. (반드시 OPEN 상태로 두세요) 그리고 점퍼핀을 RS485 쪽으로 SHORT 시켜두시기 바랍니다.

SET RS485 명령을 이용해서 송신허가핀을 결정합니다. 이 명령실행 이후부터 데이터 송신때마다 송신허가 신호(ACTIVE HIGH)가 해당 핀으로 출력됩니다. 큐블록의 OS 가 알아서 자동적으로 HIGH 로 만들어주므로, 유저는 신경쓸 필요가 없습니다. TTLTX 핀이 STOP 비트를 포함한 모든 출력을 끝낼때까지 HIGH 를 유지합니다.
이 기능이 없다면 유저가 일일이 소스프로그램상에서 송신명령어 바로 직전에 TE 핀을 HIGH 로 만들고, 송신이 끝나면 LOW 로 만드는 명령어를 써주어야 합니다만, SET RS485 명령만 내리면, 이러한 번거로움없이 RS485 통신을 사용하실 수 있습니다.

```
SET RS485 1,48
```



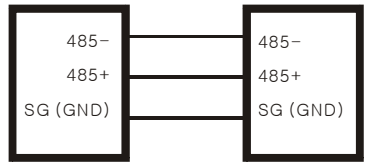
송신 허가 핀은 평상시 I/O 포트로 사용할 수 없습니다.



1: N 으로 RS485 를 사용하는 경우에는 왼쪽 회로를 참고하여 종단저항 (120 오옴)을 연결하시기 바랍니다.

종단저항은 반드시 맨끝에 두군데에서만 사용하시기 바랍니다. 모든 RS485 포트에다가 종단저항을 연결할 경우 임피던스가 너무 낮아져 통신에 장애가 발생할 수 있습니다.

* 그라운드가 서로 틀린 두 장치간에 RS485 연결을 하셨다면 GND 도 서로 연결하여 주시기 바랍니다.



Put

PUT channel, data, bytelength

channel : 사용채널 (0 부터 3)
Data : 송신데이터
Bytelength : 송신할 바이트 수

RS232 포트에 데이터를 송신하는 명령입니다. 송신 데이터에는 문자열 변수,상수를 제외한 나머지 변수 및 상수를 사용할 수 있습니다. 송신할 바이트 수만큼 데이터를 송신합니다. 문자열 데이터를 송신하려면 PUTSTR 명령을 사용합니다.

```
Opencom 1,19200,0,50,10
Dim A As Byte
A = &HA0
Put 1,A,1         '&HA0 이 전송됩니다.
```

실제로는 송신데이터를 송신용 버퍼에 저장합니다. 맨 앞에 있는 데이터부터 해당 채널로 송신됩니다. PUT 명령은 순식간에 실행이 끝나고, 바로 다음 행을 실행하게 되지만, RS232 송신은 CUBLOC BASIC 인터프리터에 의해서 송신버퍼가 모두 비워질 때까지 계속해서 수행됩니다.

만약 PUT 명령을 실행했을 때, 송신용 버퍼가 꽉 차있거나, 새로운 데이터가 들어갈만한 공간이 확보되지 않았을 경우, PUT 명령은 버퍼가 비워질 때까지 기다리지 않습니다. 즉, 송신할 데이터를 송신버퍼에 넣지 못하고 수행을 종료하게 되는 것입니다. 이런 경우를 미리 막기 위해서 송신버퍼의 크기를 사전에 확인한 후 PUT 명령을 사용하는 습관을 들이는 것이 좋습니다.

```
If Bfree(1,1) > 2 Then '송신 버퍼가 2 바이트 이상 비워져 있다면
    Put 1,A,2
End If
```

BFREE()는 버퍼에 얼마만큼의 여유가 있는지 확인하는 함수입니다.

TIPS

PUT 명령과 PUTSTR 명령 수행 시, 만약 송신버퍼에 원하는 데이터를 모두 저장했는지의 여부를 확인하고 싶다면, SYS(0)를 확인해보시기 바랍니다. SYS(0)에는 실제로 저장한 데이터의 수가 들어있습니다.

```
Opencom 1,19200,0,50,10
Putstr 1,"COMFILE"
Debug Dec Sys(0)          ' 7 이 표시되면 모두 송신버퍼에 저장한 것입니다.
```

Putstr

PUTSTR channel, data…

channel : 사용채널 (0 부터 3)
Data : 문자열로 된 데이터

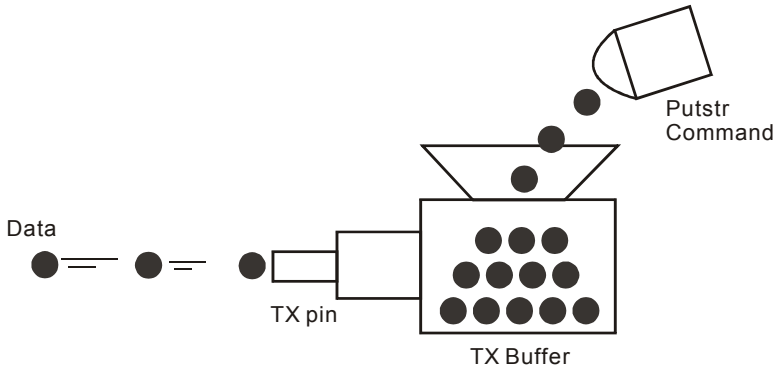
RS232 포트에 문자열 데이터를 송신하는 명령입니다.

```
Opencom 1,19200,0,50,10
Putstr 1,"COMFILE TECHNOLOGY", Dec I, Cr
```

PUT 명령과 마찬가지로 송신 버퍼에 저장하고 실행을 종료합니다. 이후 송신은 CUBLOC BASIC 인터프리터에서 알아서 수행합니다. 송신버퍼가 충분히 남아있는지 확인하고 명령을 사용해야 합니다. 만약 송신 버퍼가 충분히 남지 않았다면, 남은 양만큼의 데이터만 저장되고, 나머지는 저장하지 못하게 됩니다. 이런 일을 막기 위해서는 사전에 송신버퍼의 여유공간을 확인하거나, 충분한 송신버퍼를 확보하는 것이 좋습니다.

RS232 데이터가 송신되는 방식을 이해할 수 있도록 다음 예를 들어보겠습니다. 다음 그림은 야구연습장에서 볼 수 있는 "자동 야구공 발사장치"를 표현해 본 것입니다. 야구공이 들어있는 통을 발사장치에 부어버리는 동작이 Putstr 명령어 또는 다른 송신 명령(PUTA, PUT 등)입니다.

그렇게 되면, 발사장치안에는 여러 개의 야구공이 들어있습니다. 송신버퍼가 채워진 것으로 보시면 됩니다. 이후, 일정한 시간간격으로 하나씩 발사됩니다. 큐블록 펌웨어에서 송신버퍼에 들어있는 데이터를 하나씩 TX 핀으로 내보내는것과 같습니다.



발사장치 안에 있는 야구공이 모두 떨어지면 발사가 중지됩니다. 마찬가지로 송신버퍼에 있는 데이터가 모두 송신되면, 더 이상 송신되지 않습니다.

Puta

PUTA channel, ArrayName, bytelength

channel : 사용채널 (0 부터 3)
ArrayName : 송신데이터가 들어있는 배열 명 (반드시 바이트형)
Bytelength : 송신할 바이트 수

바이트형 배열의 내용을 일괄 전송할 수 있는 전송명령입니다. ArrayName 에 배열 명을 써주고, ByteLength 에 보낼 바이트 수를 써줍니다. 이때 보낼 바이트순은 선언된 배열요소 수보다 작아야 합니다. 이 명령이 실행되면, 배열의 첫 번째 요소부터 지정된 바이트 수만큼 해당 채널로 송신됩니다.

```
Dim A(10) As Byte
Opencom 1,19200,0,50,10
Puta 1,A,10                ' 배열 A의 요소 중 10 바이트만 송신합니다.
```

*주의사항 : 선언된 배열 요소 수 보다 많은 수의 데이터를 전송한다면, 뒷부분은 예상치 못한 값이 송신됩니다. 즉, 위의 예제에서 Puta 1,A,12 라고 작성한다면, 뒤의 2 바이트는 엉뚱한 값이 전송되는 것입니다. 주의하시기 바랍니다.

Puta2

PUTA2 channel, ArrayName, bytelength, Untilchar

channel : 사용채널 (0 부터 3)
ArrayName : 송신데이터가 들어있는 배열 명 (반드시 바이트형)
Bytelength : 송신할 바이트 수
UntilChar : 수신종료 캐릭터코드

PUTA 와 동일한 기능을 수행하는 명령어입니다. 송신데이터중 Untilchar 에 해당하는 코드가 발견되면 송신을 중단합니다. Untilchar 까지는 송신이 됩니다. 남은 뒷 부분의 데이터는 송신되지 않고 무시됩니다.

Bytelength 에서 지정한 수까지 송신을 했는데도 Untilchar 를 발견하지 못했다면, 송신을 중단합니다. 이 명령어는 맨뒤에 종료코드를 가진 프레임단위의 통신을 구현하는데 유용하게 활용할 수 있습니다. Untilchar 는 반드시 숫자나 바이트형 변수로 지정해주시기 바랍니다.

Get( )

Variable = GET(channel, length)

```
Variable : 결과가 저장될 변수 (문자열,실수형 변수는 사용할 수 없음)
channel : 사용채널 (0 부터 3)
length : 수신할 데이터 수
```

RS232 포트로부터 데이터를 수신하는 명령입니다. 포트로부터 직접 읽는 것이 아니라, 수신버퍼에 저장된 값을 읽어옵니다. 수신버퍼에 데이터가 없다면 데이터가 도착할 때까지 기다리지 않고 명령수행을 종료합니다. 즉 아무것도 읽어오지 못하고 의미 없는 Gabage(쓰레기)값만 반환됩니다. 이것을 막기 위해서는 BLEN( ) 함수를 사용해서, 수신버퍼에 데이터가 있는지 확인한 후 GET 명령을 수행해야 합니다.

보통의 경우 수신 인터럽트가 발생된뒤 인터럽트 루틴에 가서 GET 명령을 사용하게 됩니다. 괄호 안에는 채널번호와 수신해야 될 데이터의 바이트 수를 적어줍니다. 바이트형 변수에 값을 할당할 때에는 1 로 적어줍니다. INTEGER 형에 저장할 때에는 2 로 적어줍니다. LONG 이나 SINGLE 형에 저장할 때에는 4 로 적어줍니다. 4 이상의 값은 사용할 수 없습니다. 이것 역시도 버퍼상에 원하는 바이트수가 모두 도착했는지 미리 확인한 후에 읽어와야 합니다. 그렇지 않은 경우 엉뚱한 값을 받게 됩니다.

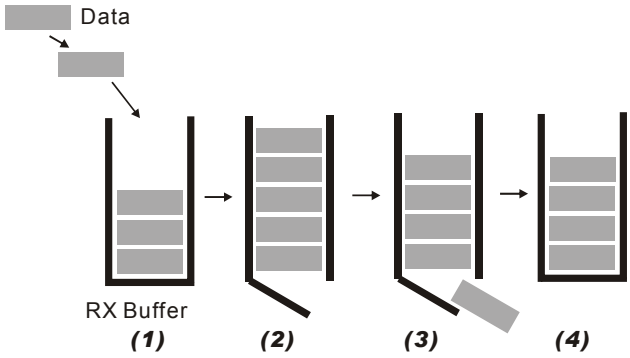
TIPS

GET 이나 GETSTR 명령을 실행했는데 원하는 길이만큼 데이터를 읽어왔는지의 여부를 확인하고 싶다면, GET 또는 GETSTR 명령실행직후 SYS(1)를 확인해 보시기 바랍니다. SYS(1)에는 실제로 읽어온 데이터의 바이트수가 기록되어 있습니다. 만약 5 바이트를 읽으려고 했는데, SYS(1)에 4 가 들어있다면 1 바이트를 읽지 못한 채 수행이 종료된 것입니다.

```
Opencom 1,19200,0,50,10
A = Get(1,4)
Debug Dec Sys(1)  ' 4 가 표시되면 4 바이트를 모두 읽어온 것입니다.
```

큐블록에서의 RS232 수신 동작

다음은 큐블록의 RS232 데이터 수신 데이터의 흐름구조를 알기 쉽게 표현한 그림입니다. 수신된 데이터는 버퍼에 저장됩니다. 이 과정은 큐블록의 OS 에서 자동적으로 수행하므로 유저가 신경쓸 필요는 없습니다. 이후 GET 명령이 수행되면 가장 처음 들어온 데이터가 버퍼로부터 나와 변수에 저장됩니다.



이 그림에서 알수 있듯이 처음 5 개의 데이터가 수신되었고, (3)번 그림에서 1 개를 읽어낸뒤 (4)번그림에서 처럼 4 개의 데이터가 남아있게 됩니다.

수신버퍼에 단 하나의 데이터라도 수신된다면 ON RECVx 인터럽트가 발생됩니다. 만약 큐블록에 이 기능이 없다면,여러분이 짠 프로그램에서 수신 시점을 알아내기 위해서, 계속 수신버퍼를 체크해야만 할 것입니다.

수신 인터럽트가 있기 때문에, 여러분은 RS232 수신 시점을 신경쓰지 않고, 다른 일을 수행하고 있다가, 데이터가 수신되면 곧바로 인터럽트 루틴에서 버퍼에 있는 데이터를 읽어오면 되는 것입니다.

SET UNTIL 명령을 사용하면, 한바이트가 수신될 때마다 인터럽트를 발생하는 것을 막고, 여러 개의 데이터가 수신되거나, 특정 종료코드를 포함한 데이터가 수신되었을 때 인터럽트를 발생시킬 수 있습니다. 그렇게 함으로써, 인터럽트가 너무 빈번히 실행되는 것을 막을 수 있습니다.

큐블록에서는 프로그램의 원활한 흐름을 위해, 수신명령에서 기다리지 않습니다. 즉, GET 명령 실행시, 수신버퍼에 데이터가 없다고 해서, 데이터가 들어올 때까지 기다리지 않는 다는 뜻입니다. 만약, 수신버퍼에 데이터가 올 때까지 기다리게 된다면, GET 명령어에서 프로그램이 잠시동안 정지되거나, 영원히 정지될 수도 있기 때문입니다. 이것은 여러분이 만든 시스템이 멀평션 (아무것도 안함)에 빠질수 도 있다는 것을 의미하기 때문에, 시스템성능에 치명적일 수 있습니다. (데이터의 일부가 안들어오더라도 시스템 전체가 멈춰버리는 일이 발생되어서는 안되기 때문입니다.) 따라서, 수신버퍼의 데이터 유무여부를 사전에 확인한한후 GET 명령 (또는 GETSTR, GETA 등)을 사용하시기 바랍니다.

Getstr( )

Variable = GETSTR(channel, length)

```
Variable : 결과가 저장될 문자열 변수
channel : 사용채널 (0 부터 3)
length : 수신할 데이터 수
```

수신 버퍼에 있는 데이터에서 지정된 바이트수만큼의 데이터를 읽어와 문자열 변수에 저장하는 명령입니다.
이 명령어는 지정된 데이터수 (length)의 숫자만큼 수신버퍼에서 무조건 데이터를 읽어옵니다. 단, 수신버퍼에 들어있는 데이터수가 length 보다 적은경우에는 읽어올 수 있는 데이터까지 읽어온상태에서 명령수행을 종료합니다.

문자열데이터의 종료문자와 크기에 대하여..

데이터를 다 읽어온 다음에는 문자열 데이터의 끝을 의미하는 NULL 캐릭터를 가장끝에 자동적으로 저장합니다. 만약 읽어온 데이터중 NULL 문자 (ASCII 코드 0)가 포함되어 있다면, 실제로 그 곳까지가 문자열 데이터로 인식됩니다. 하지만 Getstr 명령어에서는 읽어오는 데이터중에 NULL 문자가 포함되어 있어도 상관하지 않고, 정해진 숫자 (length)만큼의 데이트를 무조건 읽어온후, 맨뒤에 Null 문자를 저장합니다.

따라서 length 를 정할 때, 저장할 곳의 "문자열변수"와 같은 사이즈를 선택하거나, 적은 사이즈로 선택되어야 합니다. CUBLOC BASIC 의 특성상 RUN TIME ERROR 를 발생시킬수 없기 때문에, 사이즈의 오류가 발생한다면, 예상하지 않은 결과를 얻게 됩니다. 예를 들어 length 를 문자열변수의 크기보다 크게 지정했을 경우,다른 변수의 영역까지 침범할 수 있습니다. Getstr 명령을 사용할 때, 반드시 length 의 크기를 문자열 변수크기보다 같거나 작게 지정해 주어야합니다.

```
Dim a2 as string * 10
A2 = getstr(1,20)    ' 문자열변수의 크기는 10 인데 수신바이트는 20 으로 했다면,
                     ' 컴파일시 에러발생은 되지
                     ' 않지만, 실행중 다른변수의 내용이 파괴될수 있습니다.
```

Getstr2( )

Variable = GETSTR2(channel, length, Untilchar)

```
Variable : 결과가 저장될 문자열 변수
channel : 사용채널 (0 부터 3)
length : 수신할 데이터 수
UntilChar : 수신종료 캐릭터코드
```

GETSTR 함수와 동일한 동작을 하는 명령어입니다. 단, 읽어오는 데이터중 UntilChar 에 해당되는 코드가 발견되면, 명령어 수행을 종료합니다. Untilchar 는 문자열 변수에 저장됩니다. Length 에서 지정한 숫자까지 버퍼에서 읽어왔는데도 Untilchar 를 발견할 수 없었다면, 더 이상 읽어오지 않고 명령수행을 종료합니다.

Geta

GETA channel, ArrayName, bytelength

channel : 사용채널 (0 부터 3)
ArrayName : 수신데이터를 저장할 배열 명
Bytelength : 수신할 바이트 수

바이트형 배열에 수신된 내용을 일괄 저장할 수 있는 전송명령입니다. ArrayName 에 배열 명을 써주고, ByteLength 에 수신할 바이트 수를 써줍니다. 이때 수신할 바이트 수는 선언된 배열요소 수와 같거나 작아야 합니다. 지정한 배열 첫 번째 요소부터 바이트 수만큼 저장됩니다. 이 명령이 실행되기 전 수신버퍼에 해당바이트 수만큼 데이터가 들어와 있는지 확인할 필요가 있습니다. 만약 수신버퍼에 들어와 있는 데이터수가 모자를 경우, 나머지 데이터가 수신될 때까지 기다리지 않고 명령수행을 종료합니다. SYS(1)에는 실제로 수신된 바이트수가 들어 있습니다.

```
Dim A(10) As Byte
Opencom 1,19200,0,50,10
Geta 1,A,10                ' 채널 1 에서 10 바이트를 수신해서 배열 A 에 저장합니다.
```

Geta2

GETA2 channel, ArrayName, bytelength, UntilChar

channel : 사용채널 (0 부터 3)
ArrayName : 수신데이터를 저장할 배열 명
Bytelength : 수신할 바이트 수
UntilChar : 수신종료 캐릭터코드

GETA 와 동일한 동작을 수행하는 명령입니다. GETA 는 일정한 수의 데이터를 읽어오는 것만 가능합니다만, GETA2 는 수신버퍼에 있는 데이터중 특정코드가 있는 곳까지만 읽어올 수 있는 명령입니다. 예를들어 수신된 데이터중 코드 10 이 있는 곳까지만 읽어올 수 있습니다. 수신바이트수까지 다 읽었는데도 UntilChar 를 발견할 수 없었다면, 읽어오는 동작을 중단합니다. Untilchar 도 배열에 저장됩니다. 이 명령어를 실행한 직후, SYS(1)을 읽어보면 실제로 수신된 바이트수가 들어있습니다.

```
Dim A(10) As Byte
Opencom 1,19200,0,50,10
Geta2 1,A,20,10            ' 채널 1 에서 코드 10 을 발견할때까지 읽어서 배열  A 에
                           ' 저장합니다. 20 바이트를 다 읽을때까지 코드 10 이 없으면
                           ' 명령수행을 종료합니다.
```

다음은 PUTA2, GETSTR2 명령을 테스트해볼 수 있는 프로그램입니다. 디바이스는 CB405 를 사용했고, RS232 채널 3 에 RX,TX 핀을 서로 묶어두었습니다. (송신데이터를 그대로 수신할 수 있도록..) 그리고 포트 4 와 포트 10 에 PUSH 스위치를 연결하여, 누르면 HIGH 가 되도록 회로를 구성한뒤 실험하였습니다.

```
    Const Device = CB405
    Dim st1 As String
    Dim st2 As String
    Dim aa1 As String
    Dim aa2 As String
    aa1 = "programmer"+Chr(10)
    aa2 = "timer"+Chr(10)
    st1 = "comfile tech "
    st2 = "cubloc & CT172X/C controller"
    Opencom 3,115200,3,100,100
    On recv3 Gosub aaa
    Set Until 3,100,10
    Do
          Debug st1,Cr
          Debug addst(st1,st2),Cr
          Delay 200
          If In(4) = 1 Then
                Puta2 3,aa1_a,80,10    ' ←-- 포트 4 스위치를 누르면 이곳이 실행
                Do While In(4) = 1
                Loop
          Endif
          If In(10) = 1 Then
                Puta2 3,aa2 a,80,10    ' ←- 포트 10 스위치를 누르면 이곳이 실행
                Do While In(10) = 1
                Loop
          Endif
    Loop

aaa:
    st2 = Getstr2(3,120,10)           '←-- 바로 이 부분이 GETSTR2 하는 부분
    Bclr 3,0
    Return

    End

Function addst(sta1 As String , sta2 As String ) As String
    addst = sta1 + sta2
End Function
```

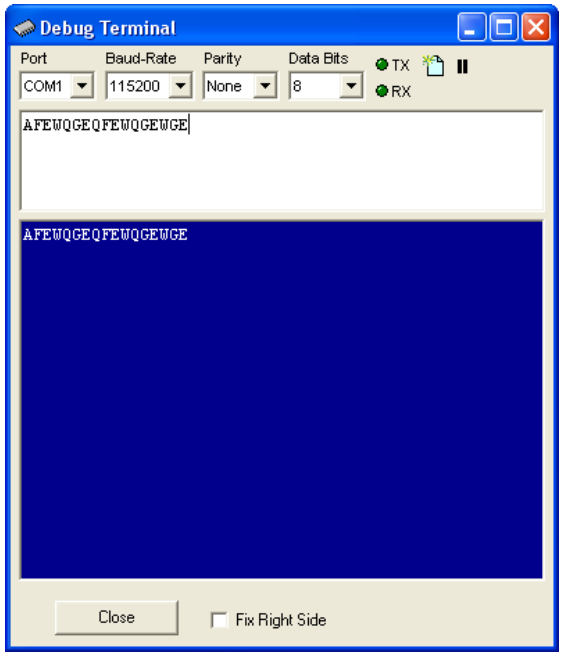
포트 4 또는 포트 10 에 연결된 스위치를 누르면, DEBUG 화면에 표시되는 데이터가 변경됩니다. PUTA2 에 의해서 송신된 데이터를 GETSTR2 로 읽어들인뒤 DEBUG 터미널에 표시하는 프로그램입니다.

TIPS : 채널 0 의 사용

RS232 통신 관련 명령의 채널번호에 0 번을 사용하면, 큐블록에 있는 Download 용 RS232 포트를 억세스 할 수 있습니다. 이 채널은 PC 와 연결되어 있으므로, PC 와 데이터를 주고 받을 일이 있다면, OPENCOM 에서 0 번 채널을 OPEN 한 후, GET, PUT 명령 등으로 데이터를 주고 받을 수 있습니다.
단, 채널 0 을 사용하려면 DEBUG 명령이나, LADDER 모니터링 기능을 사용할 수 없으므로, 주의하시기 바랍니다.

```
    Const Device = CB280
    Dim I As Byte
    Opencom 0,115200,3,50,50
    On Recv0 Gosub RCV_0
    Do
    Loop

RCV 0:
    I = Get(0,1)  ' 입력 들어오는 데이터를
    Put 0,I,1     ' 그대로 송신합니다.
    Return
```

이 프로그램을 실행시킨 후 디버그 창을 열어서 (실행 메뉴에서 열 수 있습니다.) 송신 칸에 커서를 위치시킨 후, 타이핑을 해보세요! DEBUG 창에 타이핑한 캐릭터가 그대로 표시되는 것을 볼 수 있습니다.



CheckBf( )

Variable = CheckBf(channel)

```
Variable : 결과가 저장될 변수 (문자열, 실수형 변수는 사용할 수 없음)
channel : 사용채널 (0 부터 3)
```

수신버퍼에 영향을 주지 않고, 수신버퍼에 있는 내용을 확인할 수 있는 명령입니다. GET 과 GETSTR 명령은 읽어낸 바이트 수만큼 수신버퍼에서 데이터를 삭제합니다. 하지만 CHECKBF 명령은 데이터를 읽어내긴 하지만 수신버퍼에서 데이터를 삭제하진 않습니다. 수신버퍼에서 값을 읽기 전에 확인하는 용도로 사용할 수 있는 명령입니다.
이 명령은 GET 명령과 동일한 명령이지만, 단 한 바이트만을 읽어볼 수 있습니다. 결과를 저장할 변수는 문자열과 실수형변수를 제외한 나머지 정수형 변수를 사용해야 합니다.

Sys( )

Variable = SYS(address)

```
Variable : 결과가 저장될 변수
address : 번지
```

시스템의 내부 상태를 읽어올 수 있는 명령입니다. RS232 통신과 관련하여 다음과 같은 어드레스에 들어 있는 값을 읽어올 수 있습니다.

Address 0 : PUT, PUTSTR 명령실행 후 송신버퍼에 실제로 저장된 바이트 수
Address 1 : GET, GETSTR 명령실행 후 수신버퍼에 실제로 저장된 바이트 수
Address 5 : 10mS 마다 1 씩 증가하는 타이머
Address 6 : 데이터 메모리 (램)의 상한 어드레스

SYS(5)의 경우 10mS 마다 1 씩 증가하는 타이머입니다. 읽을 수만 있고, 그 값을 바꿀 수는 없습니다. 최대 65535 까지 증가한 뒤, 다시 0 부터 반복됩니다. SYS(5)를 이용하면, 일정한 시간 간격을 지연시키거나, 실행시간 등을 측정하는 용도로 응용할 수 있습니다. 일종의 기준시간이라고 볼 수 있습니다.

SYS(6)을 읽으면 큐블록의 데이터메모리의 상한어드레스를 알 수 있습니다. 최초 파워 온 시 상한어드레스는 0 이 됩니다. SUB 나 Function 부 프로그램을 콜 하면 상한어드레스가 증가됩니다 .부 프로그램 안에서 다른 부 프로그램을 콜 하면 또 상한어드레스가 증가됩니다. 이 상한 어드레스가, 큐블록의 데이터메모리 용량을 초과하면 Overflow 가 되므로, 최대 메모리 안에서 “부 프로그램을 중첩사용”할 수 있도록 해야 합니다. Sys(6)을 읽어서, 최대 허용 메모리보다 약 100 바이트 정도 여유가 있도록 하십시오. CB280 의 경우 최대 메모리가 1948 바이트 이므로, SYS(6)이 1848 이상이면, 더 이상 Sub,Function 을 중첩실행 하지 않아야 합니다.

```
A = Sys(6)   '변수 A 에 현재상한선을 저장합니다.
```

Blen( )

Variable = BLEN(channel, bufferkind)

Variable : 결과가 저장될 변수
channel : RS232 채널
bufferkind : 버퍼종류, 0=수신버퍼, 1=송신버퍼

RS232 포트 채널의 수신용 또는 송신용 버퍼에 쌓인 데이터의 개수를 반환합니다. 버퍼에 아무것도 들어있지 않다면 0 을 반환합니다. 수신버퍼의 경우, 외부로부터 RS232 데이터를 수신하면 내부데이터 수신버퍼에 저장됩니다. BLEN 명령을 사용하면 몇 바이트의 데이터가 수신되었는지 확인할 수 있습니다. GET 명령을 사용하기 전에 GET 명령에서 수신할 데이터만큼 수신되어 있는지 확인할 필요가 있습니다. 원하는 양만큼 수신되었을 때, GET 이나 GETSTR 명령을 사용해서 데이터를 꺼냅니다.

수신버퍼가 꽉 찰 때까지 GET 명령을 사용해서 데이터를 읽어내지 않는다면, 그 이후에 들어오는 데이터는 더 이상 저장할 공간이 없으므로, 잃어버리게 됩니다. 이런 일을 막기 위해서는 수신 인터럽트를 사용해서 RS232 데이터가 수신되는 데로 읽어내거나, 충분한 수의 수신버퍼를 확보해야 합니다. 수신 버퍼가 꽉 차서 더 이상 데이터가 저장할 곳이 없는 경우, 이후에 들어오는 데이터는 잃어버리게 됩니다.

```
Dim A As Byte
Opencom 1,19200,0,100,50
On Recv1 DATARECV RTN      'RS232 데이터가 수신되면 DATARECV RTN 으로 점프
Do
Loop      ' 무한루프

DATARECV_RTN:
    If Blen(1,0) > 0 Then  ' 버퍼에 저장된 데이터가 1 바이트라도 있으면..
         A = Get(1)  ' 1 바이트만 읽습니다.
    End If
Return               ' 인터럽트 루틴을 종료
```

Bclr

BCLR channel, bufferkind

channel : RS232 채널
bufferkind : 버퍼종류, 0=수신버퍼, 1=송신버퍼, 2=모두 클리어

RS232 버퍼를 모두 클리어 합니다. 채널과 버퍼종류를 적어주어야 합니다. 버퍼 종류를 0 으로 하면 수신버퍼를 클리어하고, 1 로 하면 송신버퍼를 클리어하고, 2 로 하면 송수신버퍼를 모두 클리어 합니다.

```
BCLR 1,0          ‘ 1 번 채널의 수신버퍼를 모두 클리어 합니다.
```

Bfree( )

Variable = BFREE(channel, bufferkind)

Variable : 결과가 저장될 변수
channel : RS232 채널 (0 부터 3)
bufferkind : 버퍼종류, 0=수신버퍼, 1=송신버퍼

수신버퍼 또는 송신버퍼의 여유분 개수를 반환합니다. 송신의 경우, PUT 또는 PUTSTR 명령으로 데이터를 송신용 버퍼에 기입하려면, 송신버퍼에 충분한 공간이 있는지 확인할 필요가 있습니다.

```
Dim A As Byte
Opencom 1,19200,0, 100, 50
If Bfree(1,1)>10 Then
    Put "TECHNOLOGY"
End If
```

버퍼 사이즈를 50 으로 설정했다면 49 개까지 사용할 수 있습니다. 설정한 사이즈보다 하나 작은 사이즈까지 사용가능합니다.

WaitTx

WAITTX channel

channel : RS232 채널 (0 부터 3)

송신버퍼가 모두 비워질 때까지 대기하는 명령입니다. 아래 예처럼 DO..LOOP 와 BFREE()를 사용해서 송신버퍼가 모두 비워질 때까지 기다리는 방법과 동일합니다.

```
Opencom 1,19200,0, 100, 50
Putstr 1,"ILOVEYOU",Cr
Do While Bfree(1,1)<49  ' 앞에 명령에서 전송중인 데이터가 모두 전송될 때까지 대기
Loop
Putstr 1,"ABCDEFG",Cr      ' 다음 전송을 시작합니다.
```

WAITTX 명령을 사용하면 좀더 간편하게 송신버퍼가 비워지는 것을 기다렸다가, 새로운 데이터를 송신할 수 있습니다.

```
Opencom 1,19200,0, 100, 50
Putstr 1,"ILOVEYOU",Cr
Waittx 1              ' 앞에 명령에서 전송중인 데이터가 모두 전송될 때까지 대기
Putstr 1,"ABCDEFG",Cr      ' 다음 전송을 시작합니다.
```

이 명령에서 대기 중일 때, 인터럽트 발생이 가능합니다. 즉, CUBLOC 의 인터럽트 시스템에 영향을 주지 않는 명령어이므로, 자유롭게 사용할 수 있습니다.

On Recvx

ON RECVx GOSUB label

X 위치에 1,2,3 중하나가 위치합니다. 채널 x (1 부터 3) 의 RS232 버퍼에 데이터가 수신되면 label 로 점프합니다. Label 에는 인터럽트 루틴이 위치해 있어야 하며, 맨 나중에는 RETURN 명령을 적어주어야 합니다.

```
DIM A(5) AS BYTE
OPENCOM 1,19200,0, 100, 50
ON RECV1 DATARECV_RTN     'RS232 채널 1 에 데이터가 수신되면 DATARECV_RTN 으로
DO
LOOP      ' 무한루프

DATARECV RTN:
    IF BLEN(1,0) > 4 THEN
          A(0) = GET(1,1)  ' 1 바이트만 읽습니다.
          A(1) = GET(1,1)  ' 1 바이트만 읽습니다.
          A(2) = GET(1,1)  ' 1 바이트만 읽습니다.
          A(3) = GET(1,1)  ' 1 바이트만 읽습니다.
          A(4) = GET(1,1)  ' 1 바이트만 읽습니다.
    END IF
RETURN               ' 인터럽트 루틴을 종료
```

RECV 인터럽트 루틴을 수행하는 동안, 중복해서 같은 종류의 인터럽트가 발생되지는 않습니다만, 수신 인터럽트 루틴의 수행을 끝마치고, 메인 프로그램으로 복귀했을 때, 수신버퍼에 데이터가 남아있다면 계속해서 인터럽트가 발생하게 됩니다. 즉, RECV 인터럽트의 발생여부는 RS232 수신여부를 가지고 판단하는 것이 아니라, 수신버퍼에 데이터가 남아있는지 여부를 판단하는 것입니다.

같은 채널에서의 On Recv 는 프로그램에서 단 한번만 사용가능하며, 중복사용하였을 경우, 가장 최근에 실행된 것이 유효하고, 이전것은 무시됩니다.

Set Until

SET UNTIL channel, packetlength, untilchar

```
channel : 사용채널 (0 부터 3)
packetlength : 패킷수
untilchar : 종료캐릭터
```

일종의 조건부 수신 인터럽트 발생 선언문입니다. ON RECV 만 선언한 상태에서는 수신버퍼에 단 1 바이트의 데이터만 도착해도 수신 인터럽트가 발생됩니다. 계속해서 RS232 데이터가 도착한다면, 너무 잦은 인터럽트 발생으로 인해 본 프로그램 원활한 흐름에 영향을 줄 수 있습니다.
SET UNTIL 선언을 사용해서, 원하는 종료코드가 도착했을 경우에만 수신 인터럽트를 발생하도록 하면, 너무 잦은 인터럽트 발생을 막을 수도 있고, 원하는 하나의 프레임을 갖춘 데이터가 도착했을 경우에만 인터럽트를 발생시킬 수도 있습니다.

즉, SET UNTIL 에 사용한 종료캐릭터가 도착하기까지 계속해서 데이터를 수신하고 있다가, 종료캐릭터가 도착하면 수신 인터럽트가 발생하는 것입니다.

만약, 수신버퍼를 초과하는 양의 데이터가 도착했는데도 지정한 종료캐릭터가 나타나지 않을 경우를 대비해서 최대수신 가능 바이트 수를 정해주어야 합니다. 이것이 PACKET 수 입니다. 패킷수를 10 바이트로 선언하면, 10 바이트까지 데이터를 수신하다가 종료코드를 발견하면 인터럽트를 발생시키고, 10 바이트 안에 종료코드를 발견하지 못하면 10 바이트만 받은 상태에서 인터럽트를 발생시킵니다.

이 명령은 ON RECV 선언문과 함께 사용해야 합니다. ON RECV 선언문 바로 밑에 작성해주시기 바랍니다.

```
Dim A(5) As Byte
Opencom 1,19200,0, 100, 50
On Recv1 DATARECV RTN
Set Until 1,99,"S"
```

위의 예를 보면, 패킷 사이즈는 99 바이트입니다. 즉, 99 바이트 이내에 종료코드인 “S”를 발견하지 못하면 인터럽트가 발생됩니다.

```
SET UNTIL 1,5
```

위의 예와 같이, 종료코드를 생략한다면, PACKET 사이즈만 유효한 상태가 됩니다. 이 때에는 종료코드를 검사하지 않고, 5 바이트 수신 시마다 인터럽트가 발생하게 됩니다. PACKET 사이즈를 생략한 형태는 사용할 수 없습니다.

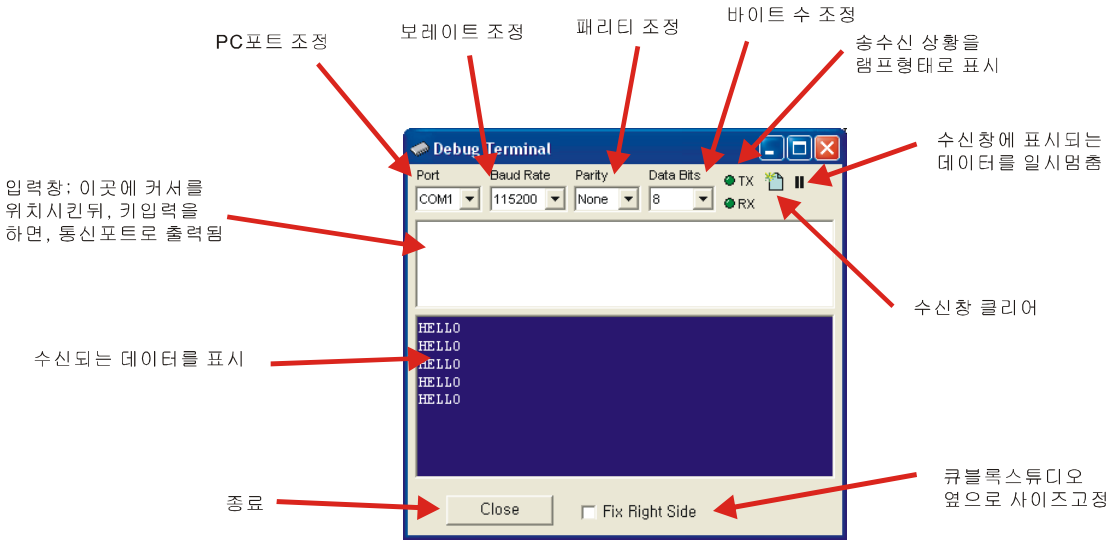
종료코드를 문자열이 아닌 코드로만 지정할 경우에는 아래의 예처럼 숫자만 적어줍니다.

```
SET UNTIL 1,100,4
```

종료코드는 단 한 바이트만 지정할 수 있습니다.

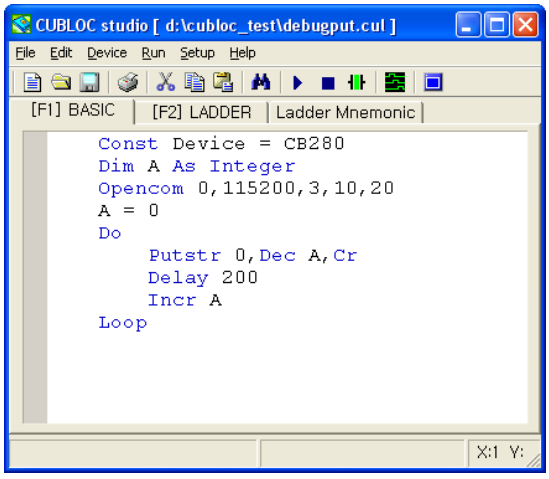
디버그 터미널의 기능

사실,“디버그 터미널”은 하이퍼 터미널과 같은 “PC 통신 프로그램”의 축소판이라고 보시면 됩니다. 간단하게 보레이트와 통신포트를 바꿀 수 있고, PC 의 RS232 포트로부터 들어오는 내용을 표시하고, 입력 창에 입력하는 내용이 PC 의 RS232 로 출력됩니다. 따라서 “디버그 터미널”을 RS232 통신 테스트용으로 충분히 사용하실 수 있습니다.



Debug 명령의 비밀

Debug 명령은 큐블록의 통신채널 0 번 (다운로드 되는 포트가 통신채널 0 번임)으로 출력하는 Putstr 명령과 똑 같은 명령입니다.



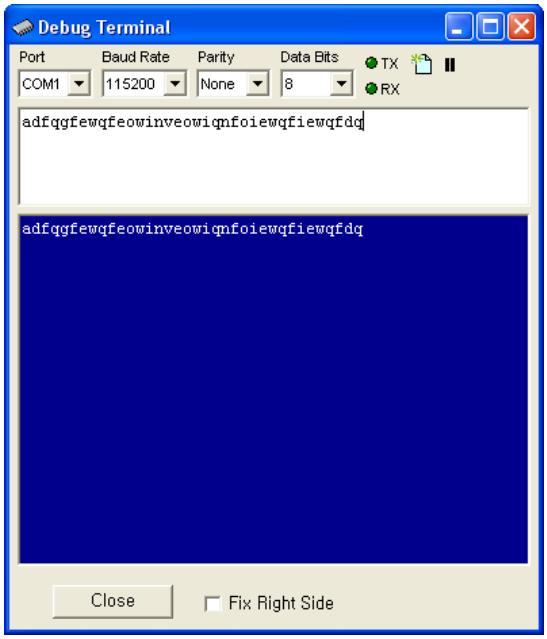
이렇게 소스를 작성하면, Debug 명령을 사용한 것과 똑 같은 효과를 낼 수 있습니다. 단지 불편한 것이 있다면, 툴 바에서 “디버그 터미널”을 클릭해 주어야 한다는 것입니다.

Debug 명령은 Opencom 선언도 필요 없이, 디버그 터미널을 따로 클릭해 줄 필요 없이 좀더 편리하게 만들어진 Putstr 명령이라고 볼 수 있습니다.

DEMO PROGRAM

디버그 터미널의 입력 창으로 데이터를 입력해서 출력 창에 표시하는 프로그램입니다.

```
    Const Device = cb280
    Dim A As Byte
    Opencom 1,115200,3,50,10
    Do
        A = Get(1,1)
        If Sys(1) = 1 Then
            Put 1,A,1
        End If
    Loop
```



이 소스를 입력하고 다운로드 할 때에는 CUBLOC STUDY BOARD-1 의 다운로드 포트(위쪽)으로 케이블을 연결하고, 테스트할 때에는 채널 1 포트(아래쪽)으로 연결하는 것을 잊지 마세요.

디버그 터미널을 띄우고, 입력 창에 커서를 위치시킨 뒤, 키보드로 아무 키나 타이핑 해보세요. 똑 같은 문자가 바로 아래쪽 출력 창에 표시됩니다.

프로그램에서 입력되는 캐릭터를 바로 출력하도록 해두었기 때문입니다.

```
    Const Device = cb280
    Dim A As Byte
    Opencom 1,115200,3,50,10
    On Recv1 Gosub GetCHAR
    Do
    Loop

GetCHAR:
    A = Get(1,1)
    Put 1,A,1
    Return
```

같은 기능을 하는 프로그램을 On Recv 인터럽트를 이용해서 작성한 것입니다.

SPI 통신

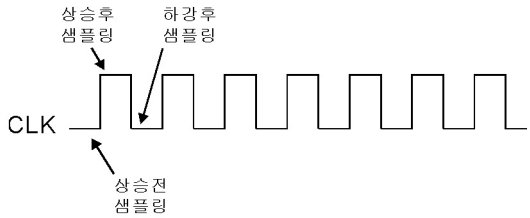
SHIFTIN, SHIFTOUT 명령은 SPI, Microwire 와 같은 시리얼 통신을 구현하기 위한 명령입니다. EEPROM 이나 ADC, DAC 와 같은 반도체 소자를 SPI 시리얼 통신으로 콘트롤 하는 경우에 사용하는 명령입니다.

Shiftin( )

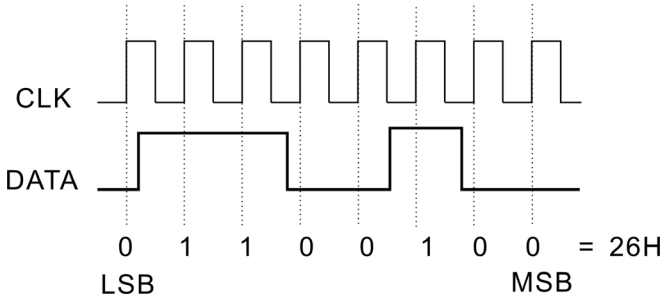
Variable = SHIFTIN(clock, data, mode, bitlength)

```
Variable : 결과가 저장될 변수
Clock : 클록 발생 포트
Data : 데이터 발생 포트
Mode : 0 = LSB 우선 (아래쪽 비트부터 수신), clock 상승 후 샘플링
       1 = MSB 우선 (위쪽 비트부터 수신), clock 상승 후 샘플링
       2 = LSB 우선 (아래쪽 비트부터 수신), clock 하강 후 샘플링
       3 = MSB 우선 (위쪽 비트부터 수신), clock 하강 후 샘플링
       4 = LSB 우선 (아래쪽 비트부터 수신), clock 상승 전 샘플링
       5 = MSB 우선 (위쪽 비트부터 수신), clock 상승 전 샘플링
bitlength : 비트 수 (1~16 사이 값)
```

쉬프트 입력을 받는 명령입니다. CLOCK, DATA 핀을 사용하는 비동기 수신 명령입니다.




```
DIM A AS BYTE
A = SHIFTIN(3,4,0,8) '3 번포트 클록, 4 번포트 데이터, 0 번 모드, 8 비트 수신
```



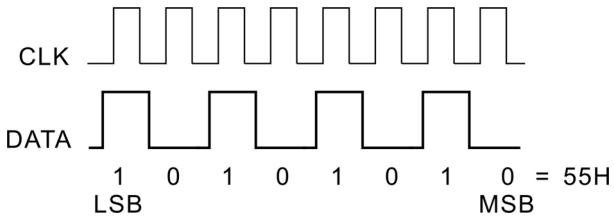
Shiftout

SHIFTOUT clock, data, mode, variable, bitlength

```
Clock : 클록발생 포트
Data : 데이터 발생 포트
Mode : 0 = LSB 우선 (아래쪽 비트부터 송신)
       1 = MSB 우선 (위쪽 비트부터 송신)
       2 = MSB 우선 (위쪽 비트부터 송신), ACK 발생 (I2C 용)
variable : 송신할 데이터가 저장된 변수
bitlength : 비트 수 (1~16 사이 값)
```

쉬프트 출력을 하는 명령입니다. 모드는 3 가지가 있습니다. 모드 2 번은 IIC 프로토콜에서 사용하는 방식입니다. IIC 에서는 8 비트 전송마다 ACK(일종의 전송확인절차)를 발생시켜 줍니다.

```
SHIFTOUT 3,4,0,&H55,8    '3 번포트 클록, 4 번포트 데이터, 0 번 모드,8 비트 송신, &H55 를 송신
```



다음은 어플리케이션 노트 10 “MCP3202 12 비트 A/D 변환 칩”을 사용하는 예제입니다. 자세한 설명은 어플리케이션 노트를 참고하시기 바랍니다.

```
Const Device = CB280
Const iodi = 7
Const iodo = 6
Const ioclk = 5
Const iocs = 4
Dim I As Byte
Dim ad As Integer
Do
      Low iocs
      i = &b1011 'Channel 0
      'i = &h1111 'Channel 1
      Shiftout ioclk,iodi,0,i,4
      ad = Shiftin(ioclk,iodo,3,12)
      High iocs
      Debug Dec ad,cr
      Delay 100
Loop
```

Spi

Indata = SPI(Outdata, Bits)

```
Indata : input data, 입력된 데이터
Outdata : output data, 출력할 데이터
bits : Number of bits (1 to 32), 비트수 최대 32 까지
```

송신과 동시에 수신을 수행할수 있는 SPI 통신 코멘드입니다. SHIFTOUT, SHIFTIN 명령은 송신과 수신동작이 분리되어 있지만, SPI 함수는 송신과 수신을 동시에 수행하도록 되어 있습니다. 이 명령은 마스터 SPI 통신으로 만 사용가능합니다. 특정핀을 사용하지고 않고, i/o 포트를 자유롭게 사용할 수 있습니다.

참고적으로 PAD 통신에 이용되는 하드웨어 SPI 는 슬레이브 통신만 가능합니다.

SET SPI 명령은 SPI( ) 함수를 사용하기 전에 선언하는 명령입니다. 클럭, 데이터출력, 데이터입력 포트, 입출력 모드등을 결정합니다.

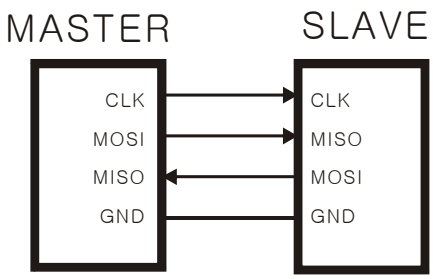
Set Spi

SET SPI clk, mosi, miso, mode

```
clk : 클록에 사용할 포트
mosi : 데이터 출력 포트
miso : 데이터 입력포트
mode : 입출력 모드로 비트 4 개를 가지고 모드를 결정합니다.
     bit 3: 0=MSB 부터 시작, 1=LSB 부터 시작
     bit 2: 클럭의 극성 ; 0=LOW 상태에서 대기, 1=HIGH 상태에서 대기
     bit 1: OUTPUT 샘플링 지점 ; 0=앞의 엣지에서 샘플링, 1=뒤에 엣지에서 샘플링
     bit 0: INPUT 샘플링 지점 ; 0=앞의 엣지에서 샘플링, 1=뒤에 엣지에서 샘플링
    예) Set Spi 9,8,7,0
```

사용예)

```
Const Device = CB280
Dim Dtin as Byte
Set Spi 9,8,7,0
Dtin = Spi(Dtout,32)
```



I2C 통신

큐블록에서는 I2C 통신을 보다 쉽게 구현하기 위한 명령 셋을 제공하고 있습니다. I2C 통신은 매우 폭넓게 이용되고 있는 통신 프로토콜이며, 주로 ADC, EEPROM, DAC, External I/O 와 같은 칩과 정보를 교환하기 위해서 사용합니다.

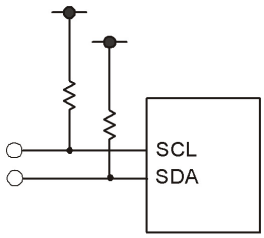
I2C 는 SDA 와 SCL, 두 선을 사용하며, 통신의 주체에 따라 마스터 디바이스와 슬레이브 디바이스로 구분되어 있습니다. 큐블록에서는 마스터 기능만 사용할 수 있습니다.

Set i2c

SET I2C DataPin, ClockPin

```
DataPin : SDA, 데이터 송수신 핀
ClockPin : SCL, 클록 송수신 핀
```

I2C 통신에서 사용할 데이터 핀과 클록 핀을 정의하는 명령입니다. 이 명령이 실행되면 선언된 두 핀은 모두 OUTPUT, HIGH 상태가 됩니다. 두 핀 모두 입/출력 겸용 핀으로 선택하여 주시고, 반드시 4.7K 옴의 저항으로 풀업시켜 주십시오.



일부 큐블록 코어모델에는 입력전용 핀, 출력전용 핀이 있습니다. 이 핀들은 I2C 통신 선으로 사용할 수 없습니다.

OPENCOM 과 달리 SET I2C 명령은 다시 사용할 수 있습니다. 하나의 큐블록 / 큐터치 제품에 여러 개의 I2C 포트를 연결하였을 경우, SET I2C 명령을 사용해서 I2C 포트를 바꿔가면서 사용하는 것이 가능합니다.

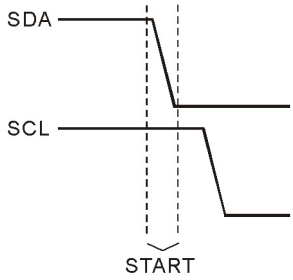
```
SET I2C 8,9          '8 번을 SDA 로 9 번을 SCL 로 정의
I2CSTART
IF I2CWRITE(&H10100000) = 1 THEN ERR_PROC
I2CSTOP
SET I2C 10,11        '10 번을 SDA 로 11 번을 SCL 로 정의
I2CSTART
IF I2CWRITE(&H10100000) = 1 THEN ERR PROC
I2CSTOP
```

이런식으로 여러 개의 I/O 를 I2C 포트로 동시에 사용할 수 있습니다.

I2Cstart

I2CSTART

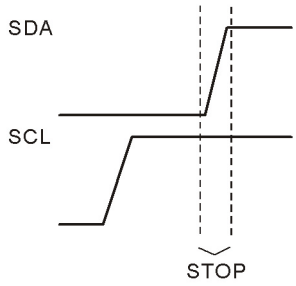
I2C 클록과 데이터 핀을 Start 상태로 만듭니다. 이 명령 수행 후에 SCL, SDA 핀은 모두 LOW 상태가 됩니다.



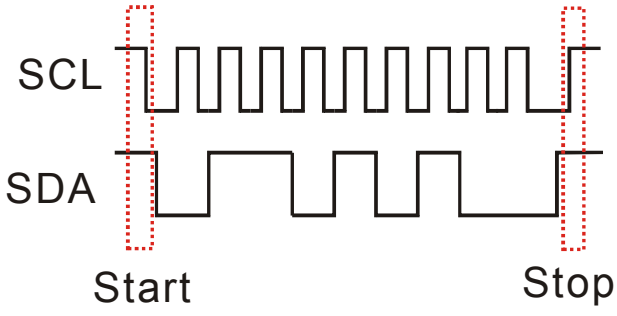
I2Cstop

I2CSTOP

I2C 클록과 데이터 핀을 Stop 상태로 만듭니다. 이 명령 수행 후에 SCL, SDA 핀은 모두 HIGH 상태가 됩니다.



다음은 한바이트의 I2C 데이터에서 Start, Stop 상황을 보여줍니다. SCL 과 SDA 선은 평상시 Pull Up 저항에 의해 High 상태로 있다가, Start 조건이 발생되면, 모두 Low 가 됩니다. 이후 데이터를 송/수신이 모두 끝나면 Stop 조건이 발생되어 SCL, SDA 모두 High 상태가 되는 것을 볼 수 있습니다.

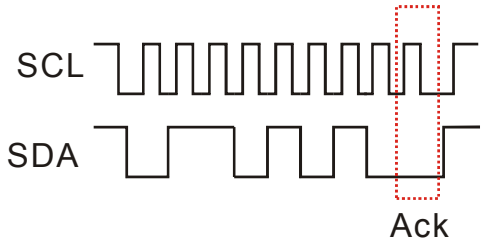


I2Cread( )

Variable = I2CREAD(dummy)

```
Variable : 결과를 저장할 변수명
dummy: 의미 없는 값.
```

SET I2C 에서 설정한 클록과 데이터 핀으로 부터 한 바이트의 데이트를 읽어옵니다. 괄호 안에는 아무것도 적어 주지 않습니다. Acknowledge 를 발생시킵니다. 좀더 구체적으로 설명하자면 한 바이트를 읽어온뒤 SCL 에 펄스 하나를 발생시키는 동안 SDA 를 LOW 로 잡아둡니다. 이렇게 하면 데이터를 읽어오는 상대측에 Acknowledge 를 보내는 것입니다.




```
A = I2CREAD(0)
```

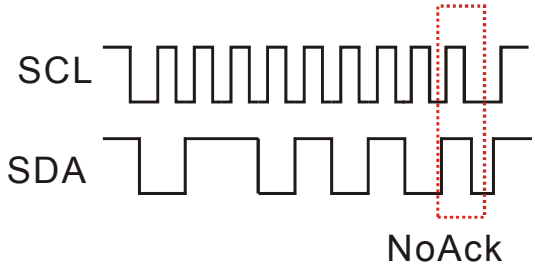
괄호 안에는 의미 없는 숫자 하나를 써줍니다. 보통은 0 을 씁니다.

I2Creadna( )

Variable = I2CREADNA(dummy)

```
Variable : 결과를 저장할 변수명
dummy: 의미 없는 값.
```

I2CREAD 와 같은 명령어입니다. 다만 Acknowledge 를 발생시켜주지 않습니다. 좀더 구체적으로 설명하자면 한 바이트를 읽어온뒤 SCL 에 펄스 하나를 발생시키는 동안 SDA 를 HIGH 로 잡아둡니다. 이렇게 하면 데이터를 읽어오는 상대측에 Acknowledge 를 보내지 않는 것입니다.



I2Cwrite( )

Variable = I2CWRITE data

```
Variable : Acknowledge 의 응답여부 (1=응답 없음, 0=응답 있음)
data : 전송할 변수/상수
```

SET I2C 에서 설정한 클록과 데이터 핀으로 한 바이트의 전송합니다. Acknowledge 펄스를 발생시켜 주며 응답이 있을 경우에는 0 을 리턴하고 없을 경우에는 1 을 리턴합니다. Acknowledge 응답이 없다면 I2C 연결선이 끊어져 있거나, 칩이 없는 경우, 또는 전원 공급이 안되었을 경우 등 다양한 경우가 있습니다. 이런 경우, I2C 통신을 중단하고 에러처리 동작을 하시기 바랍니다.

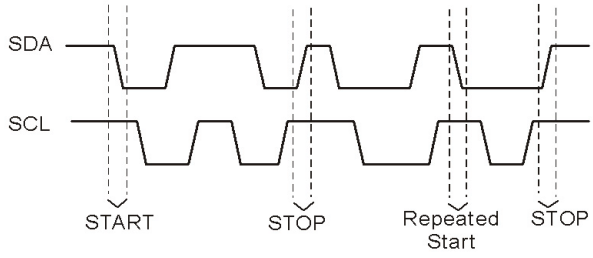
```
IF I2CWRITE(DATA)=1 THEN GOTO ERR_PROC
```

Acknowledge 에 대한 응답여부를 확인하고 싶지 않은 경우에는 어떤 변수에 결과를 저장하는 형식으로 사용하시면 됩니다.

```
A = I2CWRITE(DATA) ' Acknowledge 처리를 하고 싶지 않은 경우.
```

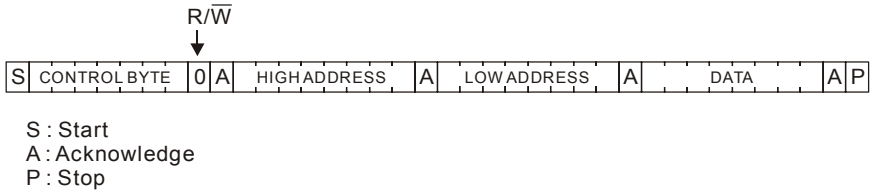
I2C 의 START, STOP 조건

SCL(클록)핀과 SDA(데이터)핀이 모두 HIGH 일 때에는 대기상태입니다. 대기상태에서 START 조건 (SCL 이 HIGH 인 상태에서 SDA 가 HIGH→LOW 됨)이 생기면 I2C 데이터 전송이 시작됩니다.
SCL 과 SDA 가 모두 LOW 일 때에는 BUSY 상태로 데이터 전송도중이라는 의미가 됩니다. 이때 STOP 조건 (SCL 이 HIGH 상태에서 SDA 가 LOW→HIGH 됨)이 생기면 I2C 는 대기상태가 됩니다. 그리고 I2C 에는 Repeated Start 라는 조건도 있습니다. STOP 조건 없이 START 조건만 다시 반복되는 상황입니다.



I2C 방식의 EEPROM 사용 예

I2C 인터페이스를 지원하는 EEPROM 24LC32 를 예를 들어 설명하도록 하겠습니다. EEPROM 의 데이터 북을 보면 다음과 같은 그림을 볼 수 있습니다. EEPROM 에 한 바이트를 라이팅 하기 위해 어떤 순서로 I2C 전송을 해야 하는지 요약한 그림입니다.



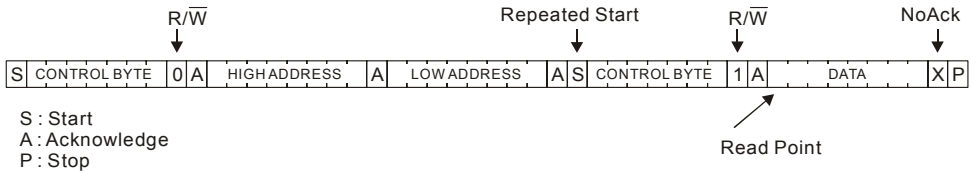
맨 앞의 S 는 Start 조건을 뜻합니다. Control Byte 는 EEPROM 고유의 구분기호와도 같은 것이며 24LC32 의 경우 상위 4 비트에 1010 을 적어줍니다. 하위 3 비트 칩 선택 어드레스 (Chip Select Address)를 적어둡니다. 칩 선택 어드레스는 칩의 1,2,3 번 단자에 의해서 바꿀 수 있도록 되어 있습니다.

R/W 는 Read 일 경우 1, Write 일 경우 0 을 적어줍니다. 지금은 한 바이트를 라이트하는 중이므로 0 을 적어줍니다. 그리고 EEPROM 에서 I2C 를 제대로 수신했는지 확인하기 위한 Acknowlege 를 체크합니다. 그리고 HIGH 주소와 LOW 주소를 보낸 뒤, 데이터를 보내고, STOP 조건을 만들어 주는 것으로 1 바이트에 대한 기입이 완료됩니다. EEPROM 의 경우 추가적으로 라이팅 딜레이 타임이 있는데, 대략 5mS 정도를 기다려준 뒤 다음 I2C 신호를 전송해야 합니다.

이를 CUBLOC 의 소스로 표현해 보면 다음과 같습니다.

```
SET I2C 8,9          ‘8 번을 SDA 로 9 번을 SCL 로 정의
I2CSTART
IF I2CWRITE(&H10100000) = 1 THEN ERR_PROC  ‘칩 선택 어드레스가 0 일 경우
IF I2CWRITE(ADR.BYTE1) = 1 THEN ERR_PROC   ‘ADDRESS WRITE
IF I2CWRITE(ADR.LOWBYTE) = 1 THEN ERR_PROC
IF I2CWRITE(DATA) = 0 THEN ERR_PROC        ‘1 바이트 WRITE
I2CSTOP
DELAY 5              ‘ WRITE 가 끝날 때까지 대기
```

이번에는 1 바이트를 읽어오는 과정에 대하여 살펴보겠습니다. 1 바이트 라이트하는 상황보다는 다소 복잡해 보이지만, 앞에서 어드레스를 기입하는 부분과 뒤에 한 바이트를 읽어오는 부분을 분리시켜 본다면 이해가 쉽게 되실 것입니다.



Read Point 라고 쓴 지점에서 비로소 EEPROM 으로부터 데이터를 읽어오게 됩니다. 앞 부분은 데이터를 읽어올 어드레스를 셋팅하는 과정입니다. 이를 큐블록 프로그램으로 정리하면 다음과 같습니다.

```
Set I2c 8,9          '8 번을 SDA 로 9 번을 SCL 로 정의
I2cstart
If I2cwrite(&H10100000) = 1 Then ERR_PROC  '칩 선택 어드레스가 0 일 경우
If I2cwrite(ADR.BYTE1) = 1 Then ERR_PROC   'ADDRESS WRITE
If I2cwrite(ADR.LOWBYTE) = 1 Then ERR PROC
I2cstart                                   'Repeated Start
If I2cwrite(&H10100001) = 1 Then ERR_PROC  '이번엔 읽기 명령으로..
DATA = I2cread(0)                          '결과는 DATA 에 저장됨.
I2cstop
```

이번엔 여러 개의 데이터를 반복적으로 읽어오는 방법에 대하여 설명하겠습니다. 앞에서 1 바이트만 읽어오는 부분의 맨 끝에 STOP 조건을 발생시키지 않고, 계속 읽어온다면 EEPROM 내부에서 어드레스가 자동적으로 1 씩 증가되므로, 인접한 어드레스에 대하여 계속 읽어올 수 있습니다. 여러 개 데이터를 연속적으로 읽을 경우, 이렇게 한다면 앞부분의 어드레스 셋팅에 들어가는 시간을 절약할 수 있습니다.

```
Set I2c 8,9          '8 번을 SDA 로 9 번을 SCL 로 정의
I2cstart
If I2cwrite(&H10100000) = 1 Then ERR_PROC  '칩 선택 어드레스가 0 일 경우
If I2cwrite(ADR.BYTE1) = 1 Then ERR PROC   'ADDRESS WRITE
If I2cwrite(ADR.LOWBYTE) = 1 Then ERR_PROC
I2cstart                                   'Repeated Start
If I2cwrite(&H10100001) = 1 Then ERR_PROC  '이번엔 읽기 명령으로..
For I = 0 To 10
    ADATA(I) = I2cread(0)       '10 바이트를 연속읽기, ADATA 는 배열
Next
I2cstop
```

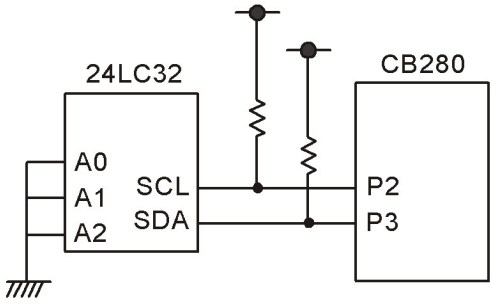
참고사항

I2CREAD 에서는 Acknowledge 에 대한 확인을 하지 않습니다. Acknowlege 는 칩의 유무, 선의 단락, 전원의 이상이 있을 경우를 확인하기 위해서 사용하는데 이미 앞에서 실행한 I2CWRITE 에서 확인되었기 때문입니다. I2CWRITE 에서 이상이 없었다면, 불과 수 마이크로 초 뒤 상황에서 이상이 발생될 가능성은 극히 드물기 때문에, I2CREAD 에서 Acknowledge 체크를 하지 않습니다.

DEMO PROGRAM

다음은 CB280 와 EEPROM 24LC32 를 연결하여 EEPROM 의 특정번지에 값을 써넣었다가 읽어서 DEBUG 창에 표시해주는 샘플 프로그램입니다.

```
    Const Device = cb280
    Dim adr As Integer
    Dim data As Byte
    Dim a As Byte
    data = &ha1
    adr = &h3
    Set I2c 3,2
    Do
          ' 1 바이트 쓰기
          I2cstart
          If I2cwrite(&b10100000)= 1 Then Goto err proc
          a=I2cwrite(adr.byte1)
          a=I2cwrite(adr.lowbyte)
          a=I2cwrite(data)
          I2cstop
          Delay 1000
          ' 1 바이트 읽기
          I2cstart
          a=I2cwrite(&b10100000)
          a=I2cwrite(adr.byte1)
          a=I2cwrite(adr.lowbyte)
          I2cstart
          a=I2cwrite(&b10100001)
          a=I2cread(0)
          I2cstop
          ' 결과 표시
          Debug Hex a,cr
          Delay 500
    Loop

err proc:
    Debug "Error !"
    Do
    Loop
```



I2C 에 대한 추가설명

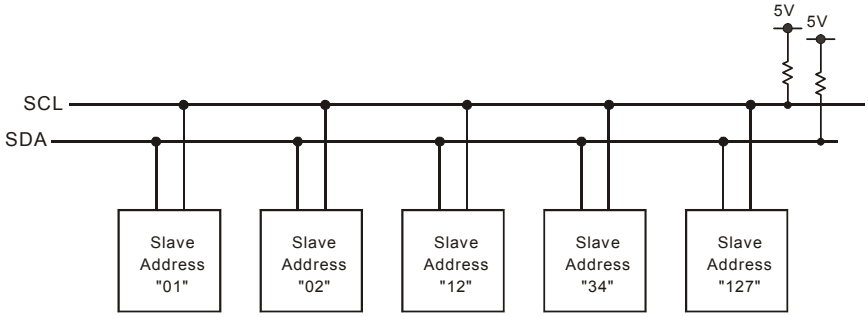
I2C 는 큐블록에서 주로 사용하는 통신 프로토콜입니다. 이전 제품인 PICBASIC 에서는 RS232C 를 주로 사용하였고, 모든 I/O 포트에서 RS232 를 사용할 수 있도록 하였으나, SERIN, SEROUT 명령의 갖고 있는 여러 가지 구조적인 문제점 때문에, 큐블록에서는 RS232 대신 I2C 를 주요 통신방식으로 채택하였습니다.

SERIN, SEROUT 명령이 소프트적으로 구현한 RS232 라면, I2CWRITE, I2CREAD 도 소프트적으로 구현한 I2C 통신 명령입니다. 다른 점은 I2CREAD, I2CWRITE 명령 수행이 LADDER LOGIC 과 BASIC 인터럽트 발생을 방해하지 않도록 되어 있다는 것과, 데이터가 수신될 때까지 무한정 기다리는 "멀펑션" 발생요소가 없다는 것입니다.

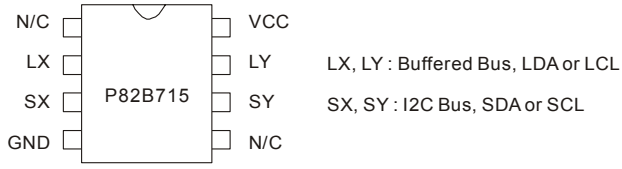
I2C 는 마스터, 슬레이브 통신방식으로 되어 있고, 큐블록은 항상 마스터 측이기 때문에, 데이터를 보낼 시점과 받는 시점을 주도적으로 콘트롤 할 수 있는 구조로 되어 있기 때문이다.

EEPROM, ADC 와 같은 I2C 방식의 슬레이브 모드 디바이스는 큐블록이 명령을 줄 때까지 대기하고 있다가, 큐블록의 요구가 있을 때 즉시, 응답해 주는 구조로 되어 있어, I2C 통신상에 어떠한 딜레이요인도 발생되지 않습니다.

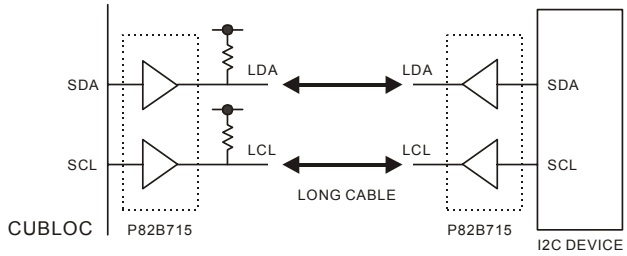
또한, I2CWRITE, I2CREAD 명령을 사용한다면 큐블록의 I/O 포트 중 입출력 가능한 양방향성 I/O 포트라면 제한없이 사용할 수 있고, 하나의 버스에 여러 개의 디바이스를 연결해 사용하는 것도 가능합니다.



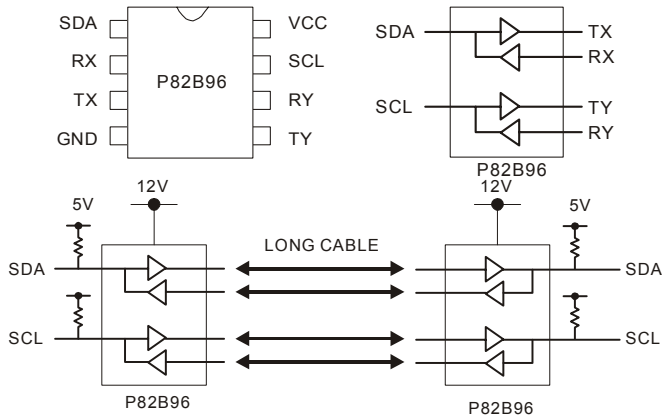
I2C 는 송신거리가 긴 어플리케이션에 적합하지 않다고 알려져 있지만 필립스에서 나온 장거리 전송 칩 P82B715 을 사용한다면 최대 1.6KM 까지 송수신이 가능하며, I2C 용 버퍼 칩인 P82B96 을 사용한다면 전원 및 그라운드가 분리된 두 개의 장치에서 I2C 사용이 가능합니다.



최대 1.6 킬로미터까지 송수신이 가능한 P82B715 를 사용한 연결도 입니다.



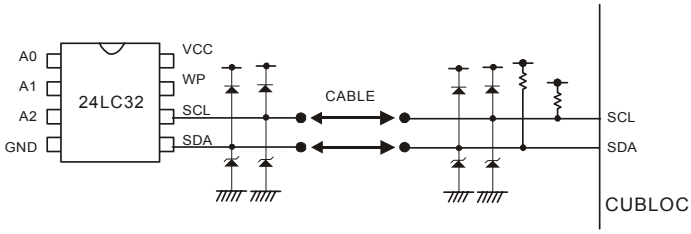
P82B96 을 사용하면 전원과 그라운드가 별개로 된, 두 개 장치간의 I2C 연결이 가능합니다.



칩에 대한 구체적인 내용은 필립스 사의 홈페이지 http://www.standardics.philips.com/ 를 참조하여 보시기 바랍니다.

케이블을 사용하긴 하는데, 장거리 인터페이스 전용 칩을 쓰기에는 가까운 거리, 예를 들어 1 미터 이내이면서 케이스 안에 서로 연결하는 경우에는 그냥 연결하지 마시고 반드시 아래와 같은 최소한의 보호회로를 사용해서 연결하시기 바랍니다.

칩의 I/O 핀이 그대로 케이블에 노출되어 있는 경우 케이블에 노이즈가 실리면 칩까지 못쓰게 되는 경우가 발생할 가능성이 있습니다. 아래와 같은 보호 다이오드 만으로도 일정부분 노이즈 감쇄효과를 얻을 수 있습니다. 하지만 완벽하지는 않으므로, 노이즈가 심한 환경에서는 반드시 I2C 인터페이스 칩으로 연결하는 것이 바람직합니다.



PAD 통신

PC 를 보면 키보드와 마우스를 연결할 수 있는 입력장치 전용 포트가 있습니다. CUBLOC 에도 외부 입력장치를 연결할 수 있는 전용 포트가 마련되어 있습니다. 이 포트에 키패드나 터치패드 등을 연결하여, 키 입력이 있을 때, CUBLOC 에서 인터럽트를 발생시킬 수 있습니다. 이 포트는 SLAVE 모드의 하드웨어 SPI 를 사용하고 있습니다.

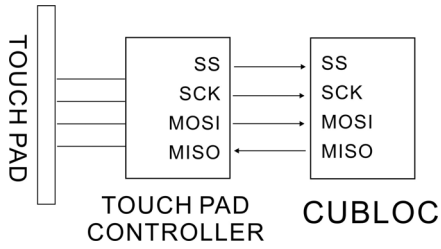
* CT172X/C 에서는 PAD 통신 포트를 "터치패드 입력"용으로 할당해 두었습니다. 따라서 여러분은 PAD 통신을 다른 용도로 사용할 수 없으며, 오로지 터치패드입력을 받기 위한 용도로만 사용하실 수 있습니다.

Set pad

SET PAD mode, packet,buffersize

```
mode : 비트우선, 샘플링 위치 지정
packet : 인터럽트가 발생해야 할 최소한의 바이트 단위
buffersize : 수신용 버퍼의 크기
```

PAD 통신을 사용하기 위해서 반드시 소스 프로그램 초기에서 선언해야 되는 명령입니다. PAD 통신은 모두 4 가닥의 신호 선을 사용합니다. 데이터 동기의 기준이 되는 SCK (Clock)와 SS (Slave Select)신호, 그리고 MOSI (Master Out Slave In), MISO (Master In Slave Out)신호가 필요합니다.



packet 은 인터럽트를 발생시킬 최소단위 바이트 숫자를 의미합니다. 터치패드의 경우 x, y 좌표 4 바이트가 도착해야 인터럽트를 발생시킬 수 있으므로, 4 가 됩니다.

buffersize 는 수신버퍼의 총 크기입니다. 최소한 packet + 1 의 크기를 가지고 있어야 합니다. buffersize 를 여유있게 정해야, 인터럽트 루틴 수행 중에 들어오는 키 입력도 계속 보관할 수 있습니다. 보통 packet 의 5,10 배로 정합니다.

다음은 CT172X/C 에서 PAD 입력을 사용하고 있는 기본적인 샘플 프로그램입니다.

```
'
'   DEMO FOR CT172X/C
'
Const Device = CT1720
Dim TX1 As Integer, TY1 As Integer
Set Pad 0,4,5                ' ← (1) Touch PAD 입력 활성화
On Pad Gosub abc             ' ← (2) 인터럽트 선언
```

```
    Do
    Loop
abc:
    TX1 = Getpad(2)              '← (3) 인터럽트 서비스 루틴
    TY1 = Getpad(2)
    Circlefill TX1,TY1,10        '← (4) 터치 지점에 원표시
    Return
```

(1) SET PAD 0, 4, 5 : 명령에 의해 PAD 명령 입력이 활성화 됩니다. (명령형식: SET PAD mode, packet size, buffer size). CT172X/C 에는 터치패드입력을 감지하여 SPI 신호를 발생시켜주는 별도의 "터치콘트롤러"가 내장되어 있습니다. 이 "터치 콘트롤러"에서 발생시키는 신호는 mode =0 에 해당합니다. (MSB 우선, clk 상승에지에서 샘플링) 입력패킷은 4 바이트 (X, Y 가 각각 2 바이트씩)로 되어 있습니다. 버퍼사이즈는 4 보다 하나 큰 5 바이트로 설정하였습니다.

(2) ON Pad Gosub ABC : 이 명령은 PAD 인터럽트 선언문입니다. PAD 입력이 발생하면 ABC 라는 라벨로 점프합니다.

(3) 인터럽트 서비스 루틴입니다. PAD 입력이 발생되면 이곳을 실행하게 됩니다. Getpad 명령으로 버퍼에 수신된 데이터를 읽어옵니다. 첫 2 바이트는 x 축 좌표, 뒤에 2 바이트는 y 축 좌표입니다.

(4) 해당위치에 원을 표시합니다.

On PAD

ON PAD GOSUB label

SET PAD 에서 지정한 packet 사이즈가 버퍼에 도착하면 인터럽트가 발생하고, ON PAD 에서 지정한 label 로 점프합니다. Label 에 있는 일련의 프로그램을 수행하고 난 뒤 복귀합니다. label 에 있는 루틴이 “pad 입력 처리를 위한 인터럽트 루틴”입니다. 맨 끝에는 반드시 RETURN 명령을 적어주어야 합니다.

```
ON PAD Gosub DATARECV_RTN    ‘수신버퍼에 데이터가 도착되면 점프
```

Getpad( )

Variable = GETPAD(length)

```
length : 수신할 바이트 수
```

PAD 통신의 수신버퍼로부터 데이터를 수신하는 명령입니다. 읽어올 바이트 수를 지정할 수 있습니다. INTEGER 형일 경우 2, LONG 형일 경우 4 를 적어줍니다.

* 이외에도 CUBLOC BASIC 에는 CUNET 통신이라는 DISPLAY 를 위한 통신채널이 있는데, CT172X/C 에서는 그래픽 LCD 에 연결해 두었습니다. 따라서 유저는 CUNET 을 별도로 사용할 수 없습니다. 그리고 CUNET 에 관련된 정의문도 따로 정의하실 필요가 없습니다. CONST DEVICE = CT1720 이라는 명령어만 사용하시면, CUBLOC BASIC 에서 알아서 설정하기 때문입니다.

MODBUS 통신

큐블록에서는 MODBUS 를 지원하고 있습니다. MODBUS 는 RS232 채널 1 을 사용하여 연결하며, **ASCII, RTU 모드를 모두 지원합니다.** 큐블록에서 MODBUS 슬레이브 모드를 동작 시키기 위해서 SET MODBUS 명령을 사용합니다.

Set modbus

SET MODBUS mode, slaveaddress, returninterval

```
mode : 0=ASCII, 1=RTU
slaveaddress : 슬레이브 어드레스 (1~254 사이의 값)
returninterval : 응답 지연 간격 (1~255 사이의 값, 생략시 1)
```

MODBUS 슬레이브 동작을 개시하기 위한 명령입니다. 반드시 OPENCOM 이후에 작성해 주어야 하며, 채널 1 에서만 동작합니다. OPENCOM 에서 지정한 보레이트와 비트 수, 패리티 등의 조건으로 외부기기와 통신합니다.

```
Opencom 1,115200,3,80,80     '수신버퍼는 50 이상으로 설정하십시오.
Set Modbus 0,1,100           'ASCII 모드, SLAVE ADDRESS=1 로 설정, 응답지연 100
```

응답지연이란, 큐블록에서 MODBUS 프레임을 수신한후, 일정시간을 대기하는 것을 말합니다. 너무 빨리 응답을 하게되면, 마스터측에서 데이터를 미처 수신하지 못하는 상황이 발생하기도 합니다. 적절한 응답지연을 주는 것으로 이러한 문제를 해결할 수 있습니다. 생략시 1 로 설정 (약 200 마이크로초)되며, 100 이면 약 4.5mS 이고, 255 이면 11mS 후에 응답을 하게 됩니다. 이 값은 1 부터 255 까지 사용가능합니다.

이 명령 실행 이후에 외부로부터 들어오는 MODBUS 수신요구에 대하여 CUBLOC 은 응답하게 됩니다. CUBLOC 은 MODBUS 에서 사용하는 펑션코드중 1,2,3,4,5,6,15,16 만 지원합니다.

펑션코드 (10 진)	동작
1	Read Coil Status
2	Read Input Status
3	Read Holding Registers
4	Read Input Registers
5	Force Single Coil
6	Preset Single Register
15	Force Multiple Coils
16	Preset Multiple Registers

이 펑션코드는 레지스터나 코일(Coil)에 값을 쓰거나 읽는 명령들입니다. 모드버스에서는 4 가지 기억장소를 가지고 코멘드를 수행하게 됩니다. 정리하면 다음과 같은 영역으로 구분되어 집니다.

기억장소명	대상 비트수	읽기/ 쓰기 가능여부	관련 펑션코드
Coil	1 bit	R/W	1, 5, 15
Input Status	1 bit	Read only	2
Holding Register	16 bit	R/W	3, 6, 16
Input Register	16 bit	Read only	4

*기억장소명 뒷부분에 Register 가 있으면 16 비트영역이고, 없으면 1 비트 영역입니다.

MODBUS 에선 각 기억장소별로 어드레스가 존재합니다. 4 개의 기억장소가 있으므로, 4 개의 어드레스군이 존재할 수 있습니다. 큐블록에서는 비트를 취급하는 기억장소 (Coil 과 Input Status)가 하나의 어드레스를 사용하고, 워드(16 비트)를 취급하는 기억장소 (Holding Register, Input Register)를 하나의 어드레스로 묶어서 사용합니다.

큐블록의 레더 릴레이 영역은 Read Only 영역이 따로 존재하지 않고, 모든 영역에서 Read/Write 가 가능하기 때문에 2 개의 어드레스 군만 사용하는 것으로 조금 간략화 시킨 것입니다.

그리고 큐블록의 레더 릴레이 영역은 P, M, F, C 등의 알파벳으로 구분하는데, MODBUS 에서는 숫자로 된 어드레스를 사용하므로, 아래 표를 참고하셔서 릴레이영역과 MODBUS 어드레스의 관계를 참고하시기 바랍니다.

비트영역 (Coil, Input Status)		워드영역 (Holding/Input Registers)	
관련 펑션코드 : 1,2,4,15		관련펑션코드 : 3,4,6,16	
어드레스 (16 진)	데이터 영역	어드레스 (16 진)	데이터 영역
0000H	P 영역	0000H	
1000H	M 영역	1000H	
2000H		2000H	
3000H		3000H	
4000H	F 영역	4000H	
5000H		5000H	T 영역
6000H		6000H	C 영역
7000H		7000H	D 영역
8000H		8000H	WP 영역
9000H		9000H	WM 영역
0A000H		0A000H	WF 영역

비트를 다루는 펑션코드에서는 비트영역 어드레스를 억세스하고, 워드를 다루는 펑션코드에서는 워드영역을 억세스 합니다.

빈칸으로 되어 있는 부분은 사용하지 않는 어드레스입니다.

Device 어드레스에 대하여..

위의 표에서 설명한 어드레스는 MODBUS 어드레스를 뜻합니다. Host 장비 (PC 나 HMI 소프트웨어등)에서 MODBUS 를 사용할 때 일반적으로 사용하는 Device 어드레스라는 것이 있습니다. Device 어드레스는 다음과 같은 규칙을 가지고 있습니다.

Device 어드레스	Modbus 어드레스	설명
1…10000	Device 어드레스 – 1	1 을 뺀값이 Modbus 어드레스가 됩니다.
40001 … 50000	Device 어드레스 – 40001	40001 을 뺀값이 Modbus 어드레스입니다.

즉, Device 어드레스가 40000 번지 이후이면 워드영역을 취급하는 것이므로, 유저는 펑션코드에 신경쓰지 않고, 워드영역에 접근할 수 있습니다.
큐블록에서 Device 어드레스를 사용해서 Modbus 를 사용하려면 아래 어드레스표를 참고하셔야 합니다. Device 어드레스는 10 진수로 되어 있음을 주의하셔야합니다. 앞에서 설명한 Modbus 어드레스는 16 진수로 표기되어 있습니다.

비트영역 (Coil, Input Status)	
관련 펑션코드 : 1,2,4,15	
Device 어드레스 (10 진)	데이터 영역
1 to 128	P 영역
385 to 512	F 영역
4097 to 8192	M 영역

워드영역 (Holding/Input Registers)	
관련평션코드 : 3,4,6,16	
Device 어드레스 (10 진)	데이터 영역
40001 to 41000	D 영역
41001 to 42000	T 영역
42001 to 43000	C 영역
43001 to 44000	WM 영역

유효구간에 대하여…

MODBUS 통신 사용시, 모델별로 유효한 구간에 대하여 올바른 값을 반환하는것에 유의하시기 바랍니다. 예를 들어 CB280 의 경우 D0~D99 만 존재합니다. 따라서 Device 어드레스 40001 부터 40099 까지만 사용할 수 있습니다 .
향후 확장가능성을 대비해 40001 부터 41000 이라는 어드레스 공간이 확보되어 있지만, 실제로 사용할수 있는 구간은 40001 부터 40099 까지이며, 나머지구간은 사용할 수 없습니다.
CB280 의 M 영역의 경우, M0~M511 까지만 유효하므로, 4097 부터 4609 까지만 사용할 수 있습니다 .유효구간을 벗어난 영역을 통신하였을 경우, 엉뚱한 값을 반환하거나, 기록되지 않을 수 있으며, 예상치못한 결과를 초래할 수 있으므로, 주의하시기 바랍니다. 유효구간은 모델마다 차이가 있으므로, 릴레이 범위표를 보고 확인하시기 바랍니다.

MODBUS 에 대하여...

MODBUS 란 MODICON 이라는 회사에서 자사의 PLC 를 위하여 개발된 PLC 접속 프로토콜로써, PLC 와 외부기기와의 인터페이스를 위하여 고안된 통신 방식입니다.

주로 터치스크린과 같은 HMI 기기와, PC 상에서 운용되는 SCADA 소프트웨어등과 통신하는 목적으로 사용합니다. 현재 시판중인 대다수의 "터치스크린 판넬"과 PC 용 HMI, SCADA 소프트웨어에서는 MODBUS 를 지원하고 있을 정도로 널리 사용되고 있는 프로토콜입니다.

MODBUS 는 마스터-슬레이브의 개념을 가지고 운용됩니다. 마스터는 데이터를 제공하는 능동적 디바이스이고, 슬레이브는 데이터를 받아들이고, 응답하는 수동적 디바이스를 의미합니다. 즉, 슬레이브에서는 오로지 마스터가 보낸 데이터에 대한 응답만 가능할 뿐, 자체적으로 데이터를 송신할 수는 없습니다.

슬레이브는 자기 자신의 고유 어드레스 (Slave Address)를 가지고 있으며, 마스터에서는 이 Slave Address 를 사용해서, 여러 개의 슬레이브 중 특정 슬레이브를 하나만을 지정해서 통신할 수 있습니다.

1:1 연결에는 RS232, RS422 을 사용하고, 1:N 연결에는 RS485 를 사용합니다. 1:1 연결에서도 Slave Address 는 사용합니다.
마스터가 보내는 하나의 멧세지 집합을 "프레임"이라고 부르며, 하나의 프레임에는 슬레이브의 어드레스, 펑션코드, 데이터, 에러체크 코드 등을 포함하고 있습니다. 슬레이브에서는 마스터가 보낸 "프레임"을 분석하여, 응답할 때에도 역시 하나의 "프레임"을 형성하여 데이터를 보내줍니다.

즉, MODBUS 는 송수신 데이터의 "프레임"구성에 대한 서로간의 약속이라고 볼 수 있습니다.

MODBUS 는 ASCII 코드만을 사용하는 ASCII 방식과 바이너리 데이터를 사용하는 RTU 방식이 있습니다. ASCII 방식보다 RTU 방식이 더 짧게 데이터를 구성할 수 있습니다. 또한 에러체크방법으로 ASCII 방식 LRC 를 사용하고 RTU 에서는 CRC 를 사용합니다.

다음은 ASCII 방식과 RTU 방식의 사용 예입니다.

필드명	*예 (Hex)*	*ASCII 방식*	*RTU 방식*
헤더		: (colon)	없음
슬레이브 어드레스	0X03	0 3	0X03
펑션코드	0X01	0 1	0X01
시작어드레스 HI	0X00	0 0	0X00
시작어드레스 LO	0X13	1 3	0X13
길이 HI	0X00	0 0	0X00
길이 LO	0X25	2 5	0X25
에러체크		LRC (2 바이트)	CRC(2 바이트)
종료코드		CR LF	없음
총바이트수		17 바이트	8 바이트

ASCII 방식은 콜론(:)으로 시작해서 CR,LF 로 종료하게 됩니다.

START	SLAVE ADR	FUNCTION	DATA	LRC	END
: (COLON)	2 바이트	2 바이트	n 바이트	2 바이트	CR,LF

RTU 방식은 특별한 헤더와 종료코드 없이 약 4 바이트 정도의 블랭크구간으로 시작과 끝을 판단합니다.

START	SLAVE ADR	FUNCTION	DATA	CRC	END
T1-T2-T3-T4	1 바이트	1 바이트	n 바이트	2 바이트	T1-T2-T3-T4

펑션코드 01 : Read Coil Status
펑션코드 02 : Read Input Status

PLC 의 비트 (릴레이) 상태를 읽어올 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 P 릴레이 20~56 번을 읽어오는 예제입니다.

Query:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X01	1	0 1	2
시작어드레스 HI	0X00	1	0 0	2
시작어드레스 LO	0X13	1	1 3	2
길이 HI	0X00	1	0 0	2
길이 LO	0X25	1	2 5	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

LRC 는 맨 앞뒤에 있는 콜론과 CR,LF 제외한 나머지 숫자를 모두 합한 뒤, 8 비트 이상 값을 제외시키고, 나머지 값을 2 의 보수로 만든 값입니다. 위의 경우 3h + 1h + 13h + 25h = 3Ch 가 되고 3Ch 의 2 의 보수인 0C4h 가 LRC 값이 됩니다. 다음은 실제로 마스터에서 송신되는 내용을 ASCII 코드와 HEX 값으로 표현한 것입니다.

ASCII	:	0	3	0	1	0	0	1	3	0	0	2	5	C	4	CR	LF
hex	3 A	30	33	30	31	30	30	31	33	30	30	32	35	43	34	13	10

이에 대한 응답은 아래와 같습니다. Response:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X01	1	0 1	2
바이트 카운트	0X05	1	0 5	2
데이터 1	0X53	1	5 3	2
데이터 2	0X6B	1	6 B	2
데이터 3	0X01	1	0 1	2
데이터 4	0XF4	1	F 4	2
데이터 5	0X1B	1	1 B	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

응답의 DATA 구성에 대하여 자세히 보면, 비트 20~27 까지 한 바이트를 구성합니다. 20 번 비트가 LSB 에 위치하고, 27 번 비트가 MSB 에 위치하도록 구성하여 응답합니다. 이렇게 해서 5 바이트를 만들어 송신하고, 끝에 남는 비트는 알 수 없는 (Dummy)값으로 송신됩니다.

펑션코드 03 : Read Holding Registers
펑션코드 04 : Read Input Registers

PLC 의 1 워드 데이터 상태를 읽어올 수 있는 펑션코드입니다. 주로 카운터, 타이머, 데이터 (C, T, D)영역의 데이터를 읽어올 때 사용합니다. 다음은 슬레이브 어드레스 3 번의 D 릴레이 0~2 번을 읽어오는 예제입니다.

Query:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X03	1	0 3	2
시작어드레스 HI	0X70	1	7 0	2
시작어드레스 LO	0X00	1	0 0	2
길이 HI	0X00	1	0 0	2
길이 LO	0X03	1	0 3	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

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

Response:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X03	1	0 3	2
바이트 카운트	0X06	1	0 6	2
데이터 1 HI	0X03	1	0 3	2
데이터 1 LO	0XE8	1	E 8	2
데이터 2 HI	0X01	1	0 1	2
데이터 2 LO	0XF4	1	F 4	2
데이터 3 HI	0X05	1	0 5	2
데이터 3 LO	0X33	1	3 3	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

펑션코드 05 : Force Single Coil

PLC 의 특정 릴레이 상태를 강제로 변화시킬 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 P1 릴레이를 ON 시키는 예제입니다. 데이터 필드에 ON 할 때에는 FF 00 을 OFF 할 때에는 00 00 을 보냅니다.

Query:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X05	1	0 5	2
시작어드레스 HI	0X01	1	0 1	2
시작어드레스 LO	0X00	1	0 0	2
데이터 HI	0XFF	1	F F	2
데이터 LO	0X00	1	0 0	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

이에 대한 응답은 아래와 같습니다. 데이터 필드에 FF 00 또는 00 00 이외의 값이 들어 있다면 아무런 변화도 일어나지 않습니다. 데이터 필드에는 반드시 FF 00 또는 00 00 만 사용해야 합니다.

Response:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X05	1	0 5	2
시작어드레스 HI	0X01	1	0 1	2
시작어드레스 LO	0X00	1	0 0	2
데이터 HI	0XFF	1	F F	2
데이터 LO	0X00	1	0 0	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

펑션코드 06 : Preset Single Registers

PLC 의 특정 데이터 영역 1 워드의 값을 강제로 변화시킬 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 D1 영역의 값을 변화 시키는 예제입니다.

Query:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X06	1	0 6	2
시작어드레스 HI	0X70	1	0 1	2
시작어드레스 LO	0X01	1	7 0	2
데이터 HI	0X12	1	1 2	2
데이터 LO	0X34	1	3 4	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

이에 대한 응답은 아래와 같습니다.

Response:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X06	1	0 6	2
시작어드레스 HI	0X70	1	0 1	2
시작어드레스 LO	0X01	1	7 0	2
데이터 HI	0X12	1	1 2	2
데이터 LO	0X34	1	3 4	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

펑션코드 15 : Force Multiple Coils

여러 개의 PLC 의 릴레이를 강제로 변화시킬 수 있는 펑션코드입니다. 다음은 슬레이브 어드레스 3 번의 P 릴레이 20~30 번을 변화시키는 예제입니다.

Query:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X0F	1	0 F	2
시작어드레스 HI	0X00	1	0 0	2
시작어드레스 LO	0X14	1	1 4	2
길이 HI	0X00	1	0 0	2
길이 LO	0X0B	1	0 B	2
바이트 카운트	0X02	1	0 2	2
데이터 1	0XD1	1	D 1	2
데이터 2	0X05	1	0 5	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

DATA 구성에 대하여 자세히 보면, 비트 20~27 까지 한 바이트를 구성합니다. 20 번 비트가 LSB 에 위치하고, 27 번 비트가 MSB 에 위치하도록 구성하여 응답합니다. 이렇게 해서 2 바이트를 만들어 송신하고, 끝에 남는 비트는 0 으로 처리합니다.

Bit	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0	1
Relay	P27	P26	P25	P24	P23	P22	P21	P20						P30	P29	P28

이에 대한 응답은 다음과 같습니다.

Response:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X0F	1	0 F	2
시작어드레스 HI	0X00	1	0 0	2
시작어드레스 LO	0X14	1	1 4	2
길이 HI	0X00	1	0 0	2
길이 LO	0X0B	1	0 B	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

펑션코드 16 : Preset Multiple Regs

여러 개의 PLC 워드 단위 데이터 영역을 강제로 변화시킬 수 있는 펑션코드입니다. 주로 타이머, 카운터, 데이터(T, C, D)영역에 새로운 값을 넣을 때 사용합니다. 다음은 슬레이브 어드레스 3 번의 D 영역 0~2 번을 변화시키는 예제입니다.

Query:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X10	1	1 0	2
시작어드레스 HI	0X70	1	7 0	2
시작어드레스 LO	0X00	1	0 0	2
길이 HI	0X00	1	0 0	2
길이 LO	0X03	1	0 3	2
바이트 카운트	0X06	1	0 6	2
데이터 1 HI	0XD1	1	D 1	2
데이터 1 LO	0X03	1	0 3	2
데이터 2 HI	0X0A	1	0 A	2
데이터 2 LO	0X12	1	1 2	2
데이터 3 HI	0X04	1	0 4	2
데이터 3 LO	0X05	1	0 5	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

이에 대한 응답은 다음과 같습니다.

Response:

필드명	*RTU 방식*	*RTU 방식 바이트 수*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더			: (colon)	1
슬레이브 어드레스	0X03	1	0 3	2
펑션코드	0X10	1	1 0	2
시작어드레스 HI	0X70	1	7 0	2
시작어드레스 LO	0X00	1	0 0	2
길이 HI	0X00	1	0 0	2
길이 LO	0X03	1	0 3	2
에러체크	CRC	2	LRC	2
종료코드			CR LF	2

에러처리

마스터에서 보내온 데이터에 오류가 있는 경우, 또는 슬레이브의 어떤 상황이 발생하여 제대로 응답할 수가 없는 상황에서 슬레이브는 에러코드를 송신합니다. 이 에러처리는 ASCII 방식에서만 발생하고, RTU 방식에서는 에러가 있을경우 응답하지 않습니다.

필드명	*예 (Hex)*	*ASCII 방식*	*ASCII 방식 바이트 수*
헤더		: (colon)	1
슬레이브 어드레스	0X03	0 3	2
펑션코드	0X81	8 1	2
에러 코드	0X09	0 9	2
에러체크		LRC	2
종료코드		CR LF	2

에러코드는 다음과 같은 종류가 있습니다.

코드	에러 명	설명
01	ILLEGAL FUNCTION	지원하지 않는 펑션코드를 수신하였을 때 발생됩니다.
02	ILLEGAL DATA ADDRESS	맞지 않는 어드레스를 수신하였을 때 발생됩니다.
03	ILLEGAL DATA VALUE	잘못된 데이터를 수신하였을 때 발생됩니다.
09	LRC UNMATCH	잘못된 체크섬(LRC)를 수신하였을 때 발생됩니다.

MODBUS 관련명령

MODBUS RTU 마스터 동작을 좀더 쉽게 구현하기 위해 만들어진 명령입니다. GETCRC 명령을 사용하면, 별도로 CRC 계산을 해줄 필요가 없어 편리합니다.

Getcrc

GETCRC variable,, ArrayName, bytelength

```
variable : 결과를 저장할 변수명 (Integer 형 변수)
ArrayName : 데이터가 들어있는 배열명 (반드시 바이트형 배열)
Bytelength : 계산할 바이트 수
```

결과가 저장될 Variable 위치에는 반드시 Integer 형 변수를 사용해야 합니다. 왜냐하면 16 비트결과를 반환하기 때문입니다. 배열은 반드시 바이트형 배열을 사용해야하고, 사전에 데이터가 들어가 있어야 합니다. 배열의 내용을 가지고 CRC16 계산을 한뒤 결과를 Variable 에 저장합니다. 배열명에는 괄호없이 배열명만 적어줍니다. 예를 들어 Dim A(10) as Byte 로 선언되었을 경우, A 만 적어줍니다. 몇바이트를 가지고 계산할 것인지를 알기위해 Bytelength 도 포함해주어야 합니다.

```
Const Device = CB280
 Opencom 1,115200,3,80,20
 Set Modbus 1,9
 Dim A(20) As Byte
 Dim B As Integer
 Ramclear
 Usepin 0,Out
 Usepin 9,Out

 Set Ladder On

 A(0) = 9
 A(1) = 2
 A(2) = 3
 A(3) = 0
 A(4) = 10
 A(5) = 23

 Getcrc B,A,6        '배열명을 적을때에는 괄호를 사용하지 않고 적어줍니다.
 Debug Hex B,Cr
```

* 바이트형 배열을 사용하지 않은 경우 별도로 에러가 발생되지는 않습니다. 하지만, 배열내부의 데이터를 바이트 단위로 불러옵니다. 예를들어 Getcrc B,_D,5 라고 명령하였을 경우, _D(0) 부터 저장된 5 바이트를 CRC 계산합니다. _D 배열은 레더의 D 영역을 저장하는 Word (16 비트)형 배열이므로, 주의해야 합니다. _D(0)부터 _D(4)까지 계산하는 것이 아니라, _D(0)에 있는 16 비트값과 _D(1)에 있는 16 비트값, 그리고 _D(2)에 있는 하위 8 비트를 가지고 CRC 계산을 수행합니다.

MODBUS 마스터 모드 구현 (ASCII 모드)

CUBLOC 을 MODBUS 마스터로 사용할 수 있는 방법에 대하여 설명합니다. MODBUS 마스터모드를 처리하기 위한 특별한 명령어는 없습니다. 마스터 모드는 RS232 데이터를 발생시키고, 수신할 수 있는 기능만 있으면, 특별한 명령어 없이 처리할 수 있으므로, CUBLOC 의 GET, PUT 과 같은 RS232 송수신 명령으로 MODBUS 마스터 기능을 모두 구현할 수 있습니다. 다음은 MODBUS 마스터를 BASIC 으로 구현한 샘플 프로그램 입니다.

두 개의 CB280 의 RS232 CH1 을 서로 연결 (TX-RX, RX-TX 가 되도록)하시고, 한쪽에는 Master 소스를 다른 한쪽에는 Slave 용 소스를 다운로드 한 뒤, 실행시켜 보세요! Slave 쪽 CB280 의 포트 0~2 에 LED 를 연결하세요! 그러면 Master 쪽에서 Slave 의 LED 를 MODBUS 를 사용해서 On/Off 제어하는 것을 확인할 수 있습니다.

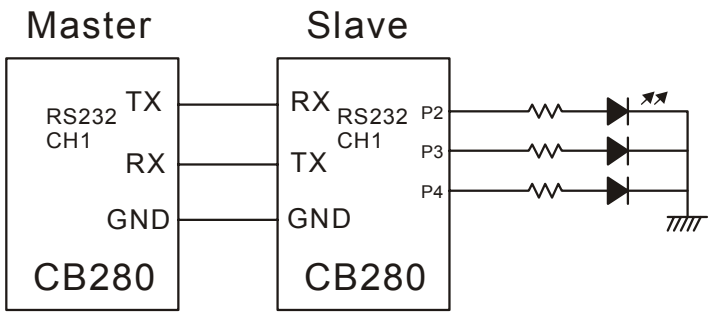
마스터 쪽에는 아래 소스를 입력하세요

```
'Master Source

Const Device = cb280
    Dim RDATA As String * 80
    Dim a As Byte, ct As Byte
    Dim b As String * 17
    Dim Port As Integer

    Opencom 1,115200,3,80,80
    On Recv1 Gosub GETMODBUS     ' Data Receive Interrupt routine
    Set Until 1,60,10            ' When Ending Code (10)
                                 ' on Channel 1 is discovered,
                                 ' create an interrupt
    Do
          For Port=2 To 4
                BitWrite Port, 1            'Turn P0,P1,P2 ON!
                Delay 100
          Next
          For Port=2 To 4
                BitWrite Port, 0            'Turn P0,P1,P2 OFF!
                Delay 100
          Next

    Loop

GETMODBUS:
    If Blen(1,0) > 0 Then  ' If buffer empty then
          A=Blen(1,0)            ' Store the buffer length in A!
          Debug "GOT RESPONSE: "
          B=Getstr(1,A)          ' Store received data in B
          Debug B
    End If
    Return

End
```

```
Sub BitWrite(K As Integer, D As Integer)
      Dim LRC As Integer
      Putstr 1,":0305"
      Putstr 1,Hp(k,4,1)
      If D=0 Then
            Putstr 1,"0000"
            LRC = -(3+5+K.Byte1+K.Byte0)'Calculate LRC
      Else
            Putstr 1,"00FF"
            LRC = -(3+5+K.Byte1+K.Byte0+0xFF) 'Calculate LRC
      End If
      Putstr 1,Hex2(LRC),13,10    'Send

End Sub
```

슬레이브 쪽에는 아래 소스를 입력하세요!

```
‘ Slave Source
    Const Device = cb280
    Opencom 1,115200,3,80,80
    set modbus 0,3
    Usepin 2, Out
    Usepin 3, Out
    Usepin 4, Out
    Set Ladder On
```



MODBUS 마스터 모드 구현 (RTU 모드)

다음은 MODBUS RTU 모드로 마스터 통신을 구현한 예제 프로그램입니다. 이 라이브러리는 송신만 처리하고 있습니다. 수신 동작은 따로 처리해주어야 합니다.

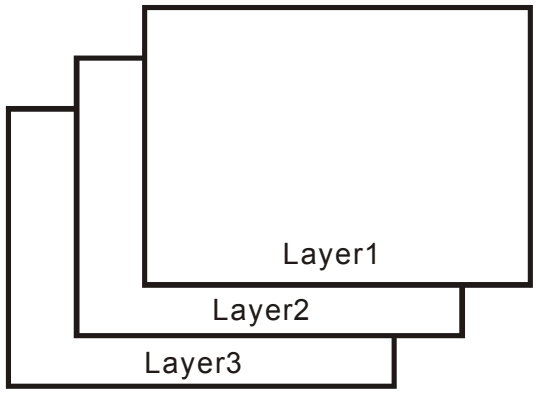
```
Const Device = CB280
Set Debug Off
Dim i As Integer
Dim _Txb(80) As Byte
Dim MBdata(32) As Integer
Dim _Crcres As Integer
Opencom 1,115200,3,150,150

' 이 부분이 메인 루틴입니다. 여기에 여러분의 소스를 작성해 넣으시면 됩니다.
Do
      Incr i
'     MBreadReg 1,2,0,1
'     MBReadCoil 1,2,0,1
      MBWriteMulReg 1,2,0,8
      'MBWriteSingleReg 1,2,0,1
      Wait 1000
Loop

End

' 여기서부터 모드버스 RTU 용 라이브러리입니다. 수신처리는 하지 않고, 송신만 처리했습니다.

Sub MBReadCoil( MBch As Byte,  MBslave As Byte, MBadr As Integer, MBlen As Integer)
       Txb(0) =  MBslave
       Txb(1) = 1
       Txb(2) =  MBadr.Byte1
       Txb(3) =  MBadr.Byte0
       Txb(4) =  MBlen.Byte1
       Txb(5) =  MBlen.Byte0
      Getcrc  Crcres, Txb,6
       Txb(6) =  Crcres.Byte1
       Txb(7) =  Crcres.Byte0
      Puta  MBch, Txb,8
End Sub

Sub MBreadReg( MBch As Byte,  MBslave As Byte, MBadr As Integer, MBlen As Integer)
       Txb(0) =  MBslave
       Txb(1) = 3
       Txb(2) =  MBadr.Byte1
       Txb(3) =  MBadr.Byte0
       Txb(4) =  MBlen.Byte1
       Txb(5) =  MBlen.Byte0
      Getcrc  Crcres, Txb,6
       Txb(6) =  Crcres.Byte1
       Txb(7) =  Crcres.Byte0
      Puta  MBch, Txb,8
End Sub

Sub MBWriteSingleReg( MBch As Byte,  MBslave As Byte, MBadr As Integer, MBdata As Integer)
       Txb(0) =  MBslave
       Txb(1) = 6
       Txb(2) =  MBadr.Byte1
       Txb(3) =  MBadr.Byte0
       Txb(4) =  MBdata.Byte1
       Txb(5) =  MBdata.Byte0
      Getcrc _Crcres,_Txb,6
```

```
        Txb(6) =  Crcres.Byte1
        Txb(7) =  Crcres.Byte0
       Puta  MBch, Txb,8
 End Sub

 Sub MBWriteSingleCoil( MBch As Byte,  MBslave As Byte, MBadr As Integer, MBdata As Integer)
        Txb(0) =  MBslave
        Txb(1) = 5
        Txb(2) =  MBadr.Byte1
        Txb(3) =  MBadr.Byte0
       If  MBdata <> 0 Then  MBdata = &hff00
        Txb(4) =  MBdata.Byte1
       _Txb(5) = _MBdata.Byte0
       Getcrc _Crcres,_Txb,6
       _Txb(6) = _Crcres.Byte1
       _Txb(7) = _Crcres.Byte0
       Puta _MBch,_Txb,8
 End Sub

 '     Write Multiple Register
 '     여러개의 Word 데이터를 Register 에 기록하기
 '  사용법
 '  먼저 MBdata 배열(Integer 형)에 데이터를 기록하고
 '  MBwriteMulReg 를 호출합니다. 이때 길이는 MBdata 배열의 길이를 초과해선 안됩니다.
 '  MBdata 배열은 여러분이 필요한만큼 따로 정의해서 사용하십시오.(최대 32)
 '  예) Dim MBdata(10) as integer '10 개의 Integer 형 데이터를 저장할 경우
 '  선언위치는 프로그램의 맨앞에 선언하십시오.
 '
 Sub MBWriteMulReg( MBch As Byte,  MBslave As Byte, MBadr As Integer, MBlen As Integer)
       Dim _mbi As Integer,_mbj As Integer
     ' 최대길이가 32 를 초과할경우 32 로 조절해준다. 왜냐하면
    ' _Txb 버퍼용량의 한계때문
    ' CB405 처럼 ram 용량이 풍부한경우에는 더 늘려쓰는것이 가능하지만
    ' TPC91a, CB220, CB280 과 같은 저용량의 모듈에서도 사용가능하도록 한것이다.
       If  MBlen > 32 Then  MBlen = 32

        Txb(0) =  MBslave
        Txb(1) = 16
        Txb(2) =  MBadr.Byte1
        Txb(3) =  MBadr.Byte0
        Txb(4) =  MBlen.Byte1
        Txb(5) =  MBlen.Byte0
        Txb(6) =  MBlen << 1
        mbj = 7
       For  mbi = 0 To  MBlen
             Txb( mbj) = MBdata( mbi).Byte1
            Incr  mbj
             Txb( mbj) = MBdata( mbi).Byte0
            Incr  mbj
       Next
       Getcrc  Crcres, Txb, mbj
        Txb( mbj) =  Crcres.Byte1
       Incr  mbj
        Txb( mbj) =  Crcres.Byte0
       Incr  mbj
       Puta  MBch, Txb, mbj
End Sub
```

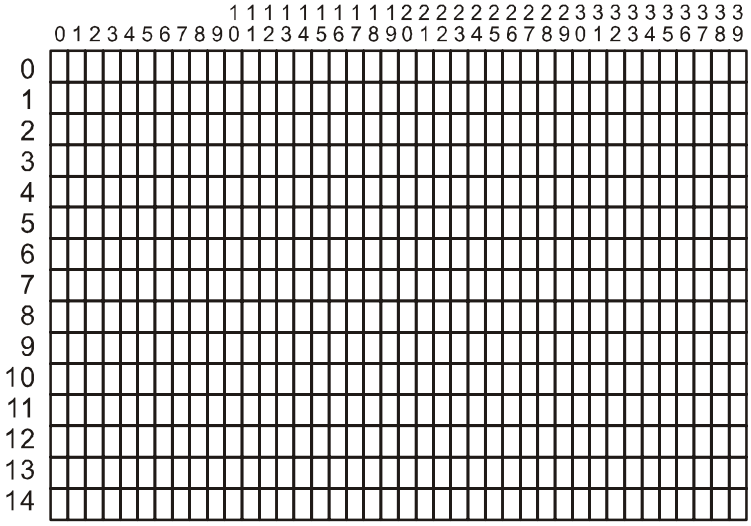
제 9 장
디스플레이 라이브러리

CT172X/C 에서 사용할 수 있는 디스플레이 관련 명령어의 집합입니다. 여기서 사용하는 다른 CUTOUCH 제품인 CT2400 등에서 사용할 수 없습니다. 오로지 CT172X/C 에서만 사용할 수 있는 명령어입니다.

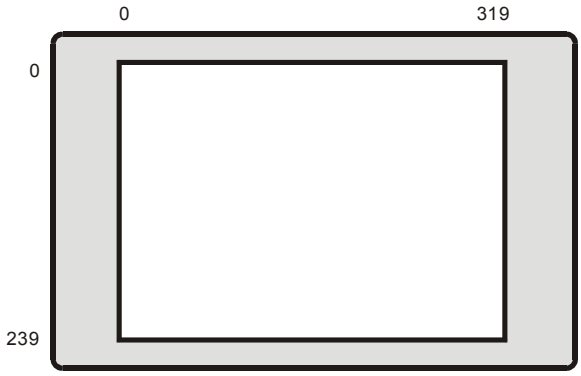
CT172X/C 그래픽 라이브러리



텍스트레이어의 좌표 계는 아래 그림과 같이 X 축으로 0~39, Y 축으로 0~14 까지 입니다. 캐릭터 하나의 크기는 8 x 16 도트이며 영문자 하나를 표시할 수 있습니다. 한글을 표시하기 위해서는 16 x 16 도트가 필요합니다. 따라서 하나의 텍스트 화면 한 줄에는 영문 40 자, 한글 20 자를 표시할 수 있습니다.



그래픽 화면의 좌표 계는 도트단위로 되어 있습니다. CT172X/C 는 320 x 240 해상도이므로 아래 그림과 같이 좌표를 지정합니다.



만약 유효범위를 넘는 좌표를 지정할 경우, 엉뚱한 위치에 표시되거나, 잘못된 그림을 그리는 경우가 있으므로 주의하시기 바랍니다. 그래픽 레이어에 문자를 표시할 때에는, 도트 단위로 원하는 위치에 표시할 수 있고, 행간이나 자간의 간격을 조정할 수 있습니다. 반면 텍스트 레이어에는 정해진 위치에만 문자를 표시할 수 있고, 행간이나 자간을 조정할 수 없습니다.

그래픽 레이어는 마음대로 그림을 그릴 수 있고, 원하는 위치에 마음대로 글자를 표현할 수 있고, 사용할 수 있는 글꼴에도 제한이 없지만, 세밀한 부분까지 신경 써주어야 하는 단점이 있습니다. 텍스트 레이어는 그림을 표시할 수 없고, 글꼴사용에도 제한이 있지만, 캐릭터 표시만 필요한 경우에 간편하게 사용할 수 있습니다.

텍스트 레이어와 그래픽 레이어는 서로 겹쳐서 표시되므로, 배경화면은 그래픽 레이어에 그리고, 텍스트는 텍스트 레이어에 표시하는 것도 가능합니다.

Cls

CLS

LCD 를 초기화하고, 모든 레이어를 지웁니다. 화면클리어에 필요한 시간을 DELAY 명령을 써서 기다린 후 다음 명령을 실행하십시오.

```
CLS
DELAY 200
```

Clear

CLEAR layer

특정 레이어만 지우거나 모든 레이어를 지웁니다.

```
CLEAR 1 ‘ 텍스트 레이어인 레이어 1 을 지웁니다.
CLEAR 2 ‘ 그래픽 레이어인 레이어 2 를 지웁니다.
CLEAR 0 ‘ 모든 레이어를 지웁니다. CLS 와 동일명령
```

Cson

CSRON

LCD 커서를 표시합니다. (디폴트 상태에서는 커서가 표시되지 않습니다.)

Csroff

CSROFF

LCD 커서를 지웁니다.

Locate

LOCATE x,y

```
X : LCD 의 X 좌표를 나타내는 변수/상수
Y : LCD 의 Y 좌표를 나타내는 변수/상수
```

텍스트 레이어상의 커서 위치를 X,Y 좌표를 새로 설정합니다. CLS 명령 실행 후에는 0,0 상태입니다.

```
LOCATE 1,1 ‘1,1 위치로 커서이동
PRINT “COMFILE”
```

Print

PRINT String/Variable

String : 문자열 변수,상수 또는 형식변환자 등
Variable : 변수(상수)를 사용하면, 해당 변수(상수)의 값이 그대로 출력됩니다.

텍스트레이어에 문자를 표시합니다. 그래픽 레이어에 문자를 표시할 때에는 GPRINT 명령을 사용합니다.

```
LOCATE 1,1 '1,1 위치로 커서이동
PRINT "COMFILE",DEC I
```

Layer

LAYER layer1mode, layer2 mode, layer3 mode

Layer1mode : 레이어 1 의 상태를 결정 (0=off, 1=on, 2=flash)
Layer2mode : 레이어 2 의 상태를 결정 (0=off, 1=on, 2=flash)
Layer3mode : 레이어 3 의 상태를 결정 (0=off, 1=on, 2=flash)

해당 레이어를 ON 또는 OFF 하는 명령어입니다. 0 으로 하면 OFF, 1 로 하면 ON 되며, 2 로 하면 16Hz 로 ON/OFF 를 플레쉬 하여 반투명으로 켜진 것처럼 표시됩니다. 디폴트 값으로 레이어 1,2 는 ON 되어 있고, 레이어 3 은 OFF 되어 있습니다.

일반적으로 이 명령은 그리는 과정을 보여주고 싶지 않을 때 사용합니다. 레이어를 OFF 하고 LINE, CIRCLE 명령 등을 사용하여 그림을 완성한 뒤, 다시 레이어를 ON 하면 완성된 그림이 순식간에 표시됩니다.

GLayer

GLAYER layernumber

Layernumber : 그래픽 레이어로 사용할 레이어 번호 (1,2,3)

3 개의 레이어가 있고 이중 하나의 레이어를 그래픽레이어로 선택할 수 있습니다 .그래픽 레이어로 선택된 레이어는 LINE, CIRCLE, BOX 등의 그래픽 명령어의 대상이 됩니다. 보통 레이어 1 은 텍스트 레이어로 사용되고, 레이어 2 를 그래픽레이어로 사용합니다.

레이어 1 을 그래픽 레이어로 사용할 수 있습니다. 이 경우 텍스트모드에 있는 글자를 그래픽명령으로 지우거나 덧칠할 수 있게 됩니다. 텍스트 레이어의 일부분을 지우거나 할 때 사용할 수 있습니다. 레이어 3 을 그래픽 레이어로 지정하고, Layer 명령으로 레이어 3 을 ON 하면 레이어 3 을 활용할 수 있습니다.

Overlay

OVERLAY opermode

```
opermode : 연산모드 설정 (0=or, 1=and, 2=xor )
```

레이어 1 과 레이어 2 의 논리연산 모드를 지정합니다. 레이어 1 은 텍스트 레이어이고, 레이어 2 는 그래픽 레이어로 사용됩니다. 따라서 이 명령은 텍스트레이어와 그래픽레이어간의 겹침 상태를 어떻게 할 것인가 결정해주는 명령입니다. 디폴트상태는 xor 로 되어 있습니다. 이 경우 겹친 부분이 반전되어 표시됩니다. 겹친부분을 반전시키지 않으려면 or 상태로 설정하시면 됩니다.

Contrast

CONTRAST value

```
value : 정수형 변수 또는 상수
```

LCD 의 밝기를 조절하는 명령어입니다.

Light

LIGHT value

```
value : 백라이트 0=OFF, 1=ON
```

백라이트를 ON, OFF 합니다. 디폴트는 ON 입니다.

Wmode

WMODE value

```
value : 0=FAST, 1=SLOW
```

LCD 에 데이터를 기록하는 방식을 선택하는 명령입니다. LCD 에 그림이나 문자를 그리기 위해서는 LCD 에 있는 메모리에 데이터를 WRITE 해야 합니다. 이때 LCD 주사방식을 무시하고 무조건 WRITE 하는 모드가 FAST 모드입니다. FAST 모드는 빠르게 표시를 할 수 있지만, LCD 주사방식을 무시하기 때문에 화면에 SNOW 현상이 나타날 수 있습니다. SNOW 현상은 LCD 에 있는 그림에 노이즈가 표시되는 현상인데, 화면에 있는 그림을 고치거나 갱신할 때 화면전체에 지저분하게 보이는 현상을 말합니다. 이런 현상을 없애기 위해서는 LCD 주사도중 화면에 영향을 주지 않는 구간을 골라서, 데이터를 WRITE 하는 방법인 SLOW 모드를 사용하면 됩니다. 단, SLOW 모드에서는 화면에 SNOW 현상이 없는 대신, 표시속도가 느려지게 됩니다. 화면표시가 많은 프로그램에서는 전체프로그램의 속도저하가 생기게 됩니다.

CT172X/C 에서는 WAITDRAW 명령을 사용해서 그림을 그리는 시간을 기다려줄 수 있습니다. WMODE 1 을 사용해서 표시속도가 느려진다면 반드시 WAITDRAW 명령을 사용해서, 일부 그림이 SKIP 되는 현상을 막아주어야 합니다.

일반적으로 초기화면을 표시할 때, LAYER 명령을 사용해서 해당레이어를 OFF 상태로 만든 뒤 그림을 모두 그린 후 다시 ON 상태로 하면, SNOW 현상을 줄인 상태에서 화면을 표시할 수 있습니다. 화면에 숫자가 갱신된다거나, 일부 그림을 자주 바꿔야 하는 경우에만 SLOW 모드로 설정하고 처리한다면, 화면 떨림 현상을 다소 줄인 상태에서 화면표시를 갱신할 수 있습니다.

Font

FONT hfont, efont

```
hfont : 0~8 사이의 숫자, 한글글꼴의 종류를 선택
efont : 0 또는 1, 영문글꼴을 선택, 0=고정 폭, 1=가변 폭
```

CT172X/C 에는 한글 9 종 영문 2 종의 글꼴데이터가 있는 폰트 롬이 내장되어 있습니다. 이중 원하는 한가지 글꼴을 선택하는 명령입니다.

위 화면은 실제 GHLCD 상에서 폰트를 표시한 후, 사진으로 촬영한 것으로, 폰트 별로 어떤 모양을 하고 있는지를 알 수 있습니다. CT172X/C 에서 지원하는 폰트는 벡터폰트가 아닌, 비트맵 폰트입니다. 따라서 각각의 폰트마다 크기와 모양이 정해져 있으며, 사용할 수 있는 글꼴의 모양도 정해져 있습니다.

한글글꼴번호	글꼴
0	10 x 16 돋움
1	10 x 16 샘물
2	16 x 16 바탕
3	16 x 16 돋움
4	16 x 16 굴림
5	16 x 16 으뜸
6	24 x 24 돋움
7	24 x 24 바탕
8	48 x 48 돋움

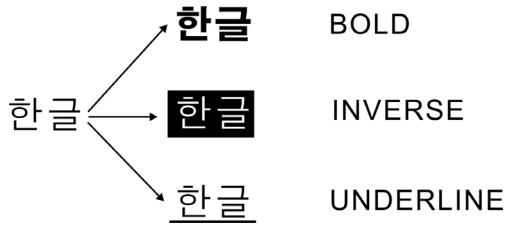
영문글꼴은 크게 나누어 고정 폭 글꼴과 가변 폭 글꼴이 있습니다. 고정 폭은 말 그대로, 일정한 폭을 갖는 글꼴을 말합니다. 모든 알파벳이 똑 같은 크기를 가지고 있기 때문에, 좌표계산은 편리해도 보기에는 미려하지 못합니다. 가변 폭 글꼴은 영문자 I 처럼 가느다란 글자와 W 처럼 두꺼운 글자의 글자 폭이 다른 글꼴입니다. 고정 폭보다는 보기가 훨씬 좋지만, 글자마다 간격이 다르므로 좌표계산에 어려움이 있습니다. 그리고, 텍스트 레이어에서 사용할 수 있는 글꼴에 제한이 있습니다. 한글 글꼴의 경우 2,3,4,5 번 폰트만 사용할 수 있고 영문자는 고정 폭 글꼴만 사용할 수 있습니다.

Style

STYLE bold, inverse, underline

```
bold : 0=볼드해제, 1=한글만 굵게, 2=영문만 굵게, 3=모두 굵게
inverse : 0=반전해제, 1=반전
underline : 0=밑줄해제, 1=밑줄
```

글꼴에 볼드 (굵게 표시), 밑줄, 반전효과를 추가할 수 있는 명령입니다. 디폴트 상태에서는 모두 해제상태입니다.

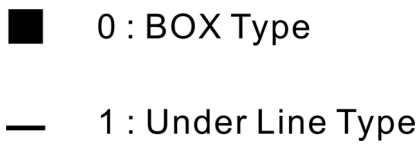


Cmode

CMODE value

```
value : 0=BOX type, 1=Underline type
```

텍스트레이어에 표시할 커서의 모양을 선택하는 명령입니다. 디폴트값은 박스형입니다.

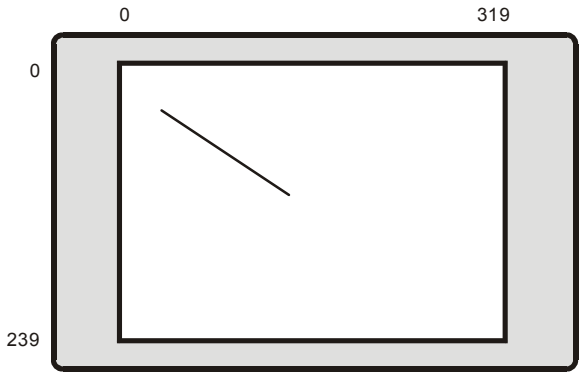


Line

LINE x1, y1, x2, y2

그래픽 레이어에 X1,Y1 지점으로부터 X2,Y2 지점까지 직선을 표시합니다.

```
LINE  10,20,100,120  '선 그리기
```

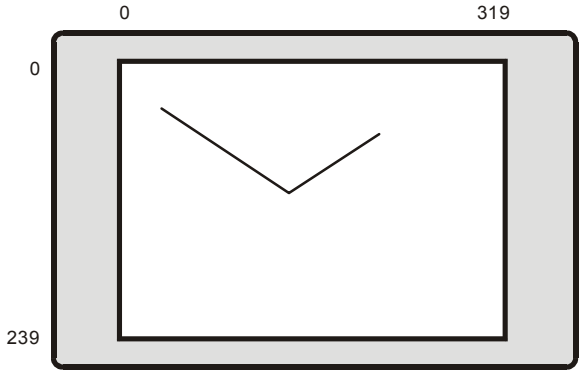


Lineto

LINETO x, y

계속해서 선을 그릴 경우, 먼저 선을 그린지점에 이어진 선을 그립니다.

```
LINETO  200,50  '선 이어 그리기
```

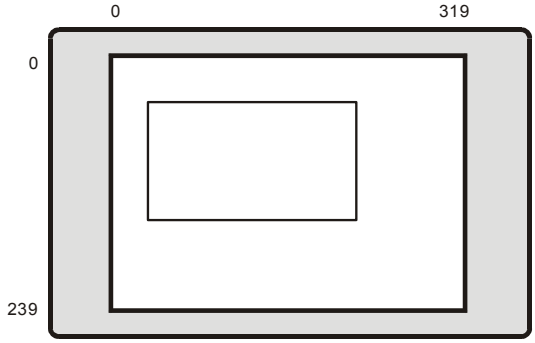


Box

BOX x1, y1, x2, y2

그래픽 레이어에 X1,Y1 지점으로부터 X2,Y2 지점까지를 대각선으로 하는 사각형을 그립니다. 그림을 그릴 수 있는 최대 좌표를 초과하지 않는 값으로 좌표를 지정해야 합니다. CT172X/C 에서는 X 축 0~319, Y 축 0~239 까지입니다.

```
BOX  10,20,200,100   '박스 그리기
```



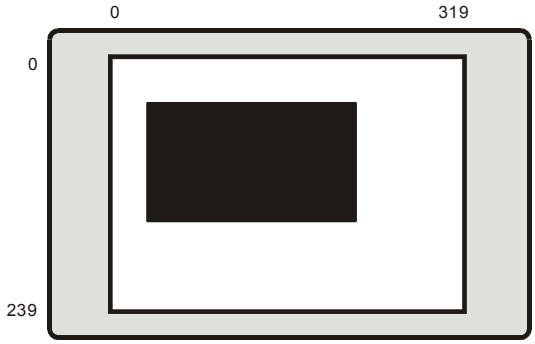
Boxfill

BOXFILL x1, y1, x2, y2,logic

```
logic : 0=OR, 1=AND, 2=XOR
```

그래픽 레이어에 X1,Y1 지점으로부터 X2,Y2 지점까지를 대각선으로 하는 채워진 사각형을 그립니다. Logic 을 사용하여 이미 그려져 있는 배경화면과의 연산종류를 지정할 수 있습니다.
0 으로 하면 or 연산이므로 배경화면과 겹쳐져서 표시됩니다.
1 로 하면 and 연산이므로 겹쳐진 부분만 표시됩니다.
2 로하면 xor 연산이므로, 겹쳐진 부분이 반전되어 표시됩니다.

```
BOXFILL  10,20,200,100,0   '박스 그리기
```

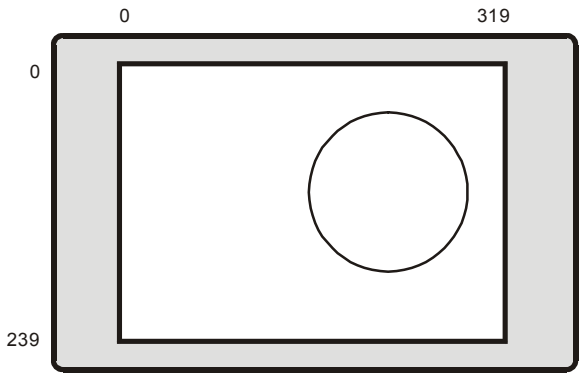


Circle

CIRCLE x, y, r

그래픽 레이어에 X, Y 지점을 중심으로 하고 r 을 반지름으로 하는 원을 그립니다. 특히 이 명령의 경우, 반드시 최대 좌표를 초과하지 않는 값으로 좌표를 지정해야 합니다. 좌표지정이 잘못되었을 경우 엉뚱한 위치에 원이 표시되거나, 표시되지 않을 수 있습니다.

```
CIRCLE  200,100,50  '원 그리기
```

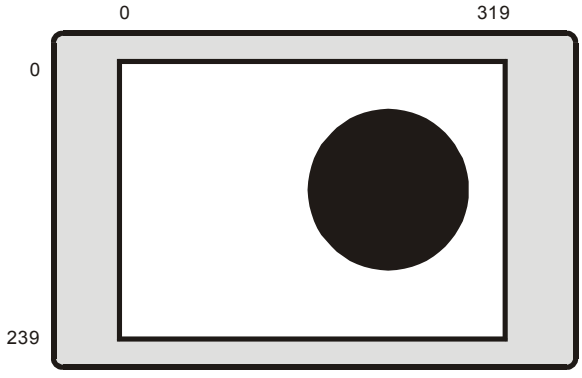


Circlefill

CIRCLEFILL x, y, r

그래픽 레이어에 X, Y 지점을 중심으로 하고 r 을 반지름으로 하는 채워진 원을 그립니다. 특히 이 명령의 경우, 반드시 최대 좌표를 초과하지 않는 값으로 좌표를 지정해야 합니다. 좌표지정이 잘못되었을 경우 엉뚱한 위치에 원이 표시되거나, 표시되지 않을 수 있습니다.

```
CIRCLEFILL  200,100,50  '채워진 원 그리기
```

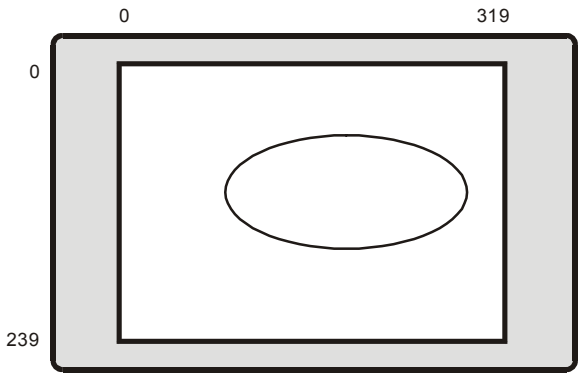


Ellipse

ELLIPSE x, y, r1, r2

그래픽 레이어에 X, Y 지점을 중심으로 하고 r1 을 가로반지름, r2 를 세로반지름으로 하는 타원을 그립니다. 특히 이 명령의 경우, 반드시 최대 좌표를 초과하지 않는 값으로 좌표를 지정해야 합니다. 좌표지정이 잘못되었을 경우 엉뚱한 위치에 원이 표시되거나, 표시되지 않을 수 있습니다.

```
ELLIPSE  200,100,100,50  ‘타원 그리기
```

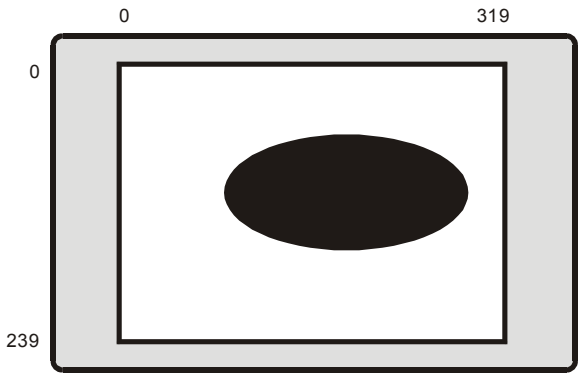


Elfill

ELFILL x, y, r1, r2

그래픽 레이어에 X, Y 지점을 중심으로 하고 r1 을 가로반지름, r2 를 세로반지름으로 하는 채워진 타원을 그립니다. 특히 이 명령의 경우, 반드시 최대 좌표를 초과하지 않는 값으로 좌표를 지정해야 합니다. 좌표지정이 잘못되었을 경우 엉뚱한 위치에 원이 표시되거나, 표시되지 않을 수 있습니다.

```
ELFILL  200,100,100,50  ‘채워진 타원 그리기
```



Glocate

GLOCATE x, y

그래픽 레이어에 새로운 위치를 정하는 명령입니다. 이 명령으로 화면상에서 어떤 변화는 나타나지 않습니다. 그래픽 레이어상의 텍스트를 표시하고자 할 때, GLOCATE 명령을 먼저 사용하여 위치를 정한 다음에 텍스트를 표시하면 해당위치에 표시됩니다.

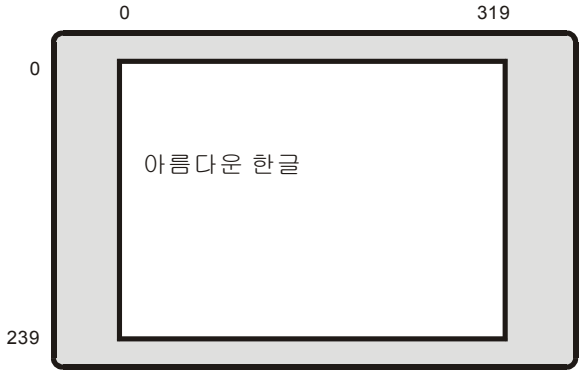
```
GLOCATE 128,32  '이 위치에 텍스트를 표시
```

Gprint

GPRINT string

그래픽 레이어에 문자를 표시하는 명령입니다. 그래픽 레이어에는 도트단위로 위치를 자유롭게 지정할 수 있으며, 자간과 행간도 조정할 수 있고, 사용할 수 있는 글꼴의 제한도 없습니다. 다만 화면바깥으로 초과되는 문자에 대하여, 자동적으로 줄바꿈되지 않습니다. 텍스트레이어의 경우 텍스트를 계속 표시하게 되면 줄 바꿈과 스크롤이 자동적으로 발생되지만, 그래픽 레이어에서는 텍스트를 그래픽의 일부분으로 처리하기 때문입니다. 줄 바꿈을 하고자 할 때에는 줄 바꿈 문자인 CR 을 뒤에 써줍니다.

```
GPRINT "아름다운 한글",CR  '문자를 표시하고 한 줄 아래로
```

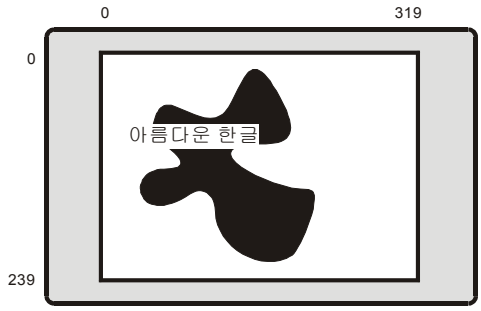


Dprint

DPRINT string

이 명령은 GPRINT 와 동일한 기능을 하지만, 그래픽 레이어상의 배경화면과 겹쳐 표시 하지 않고, 직접 문자를 표시하는 명령입니다. 따라서 기존 바탕화면과의 연산과정이 생략되므로, 좀더 빠르게 문자를 표시할 수 있습니다. 주로 카운터 값과 같이 자주 갱신해야 하는 숫자,문자 등을 표시할 때 사용합니다.
DPRINT 명령에는 GPRINT 명령과 비교해서, 다소 사용상에 제한이 있습니다. DPRINT 명령은 기존 좌표계를 그대로 사용하지만 X 좌표는 8 로 나누어 떨어지지 않는 위치에 표시하면, 인근 위치로 이동되어 표시됩니다. 즉 X 축의 좌표를 8 의 배수 간격으로만 사용할 수 있습니다. (0,8,16,등과 같이)
한글글꼴은 2,3,4,6,7 만 사용가능하고, 영문폰트는 고정 폭만 사용 가능합니다. 그리고 행간,자간을 조정하는 OFFSET 명령도 DPRINT 명령에서는 적용되지 않습니다.

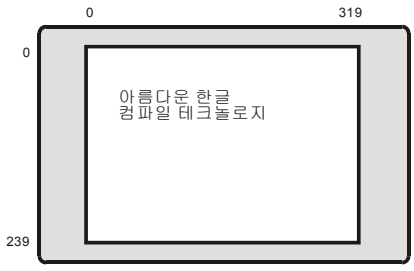
```
GPRINT "아름다운 한글",CR  '문자를 표시하고 한 줄 아래로
```

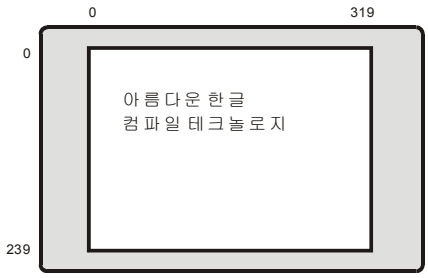


Offset

OFFSET x, y

그래픽 레이어상에 표시되는 문자에 대한 자간,행간 간격을 설정하는 명령입니다. 디폴트 값은 모두 0 이입니다. X 의 값을 0 이상의 값을 정하면 그만큼 자간의 간격이 벌어지고, Y 의 값을 0 이상으로 하면 그만큼 글자간의 간격이 벌어지게 됩니다. 자간을 조정하기 전에는 글자들이 붙어서 표시됩니다.





OFFSET 3,3 '자간, 행간을 3 으로 조정합니다.

Pset

PSET x, y

그래픽 레이어에 X1,Y1 지점에 점을 찍는 명령입니다.

```
PSET 200,100 ‘점 찍기
```

Color

COLOR value

그래픽 명령에서 사용할 색상을 지정합니다. CT172X/C 의 경우 흑백 LCD 이므로, 0 또는 1 로 지정합니다. 1 일 경우 색을 칠하고, 0 일 경우 지워집니다. 이 명령은 모든 그래픽 명령, LINE, CIRCLE, BOX 등에 영향을 주게 됩니다. 디폴트값은 1 입니다.

```
COLOR 1 ‘칼라설정, 이후 사용되는 그래픽명령어에 영향을 줌
```

Linestyle

LINESTYLE value

점선을 사용하고자 할 때 사용하는 명령입니다. Value=0 으로 하면 직선이 됩니다. 이 값을 1 이상의 값으로 하면 점선이 표시됩니다. 숫자가 클수록 점선의 간격도 커집니다. 이 명령으로 LINE 뿐만 아니라, BOX, CIRCLE, ELLIPSE, ARC 등의 그래픽 명령어도 영향을 받습니다. 디폴트 상태에서는 0(직선)입니다.

```
LINESTYLE 1 ‘이후 사용되는 그래픽 명령어에서 점선을 사용
```

Dotsize

DOTSIZE value, style

그래픽명령에서 사용될 기본단위인 점의 크기를 결정하는 명령입니다.이 명령을 사용해서 점의 크기를 변경하면, 이후 사용되는 그래픽명령에서 모두 적용됩니다. 점의 크기(value)는 최소 0 에서부터 255 까지 사용될 수 있으며, 점의 크기가 클수록 그리는 속도도 느려집니다. 점의 스타일(style)을 0 으로 하면 사각형 형태가 되고, 1 로 하면 원 형태가 됩니다. 디폴트상태에서는 value=0, style=0 입니다.

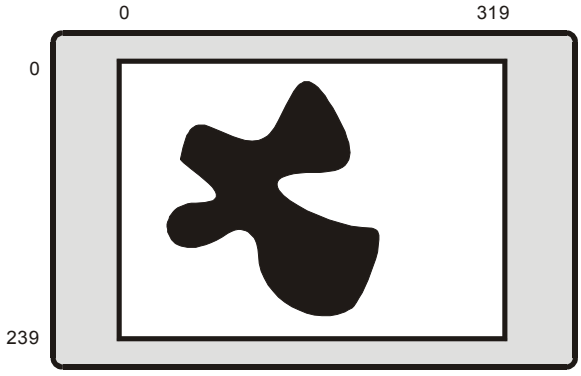
```
DOTSIZE 1,1 ‘이후 사용되는 그래픽 명령어에서 점의 크기를 조정(크기 1, 원 형태)
```

Paint

PAINT x, y

닫혀진 폐곡선 내부를 칠하는 명령입니다. 닫혀진 곡선이 아니면 화면 전체가 칠해지게 되므로 주의해야 합니다.

```
PAINT 100,100  '폐곡선 내부를 채웁니다.
```

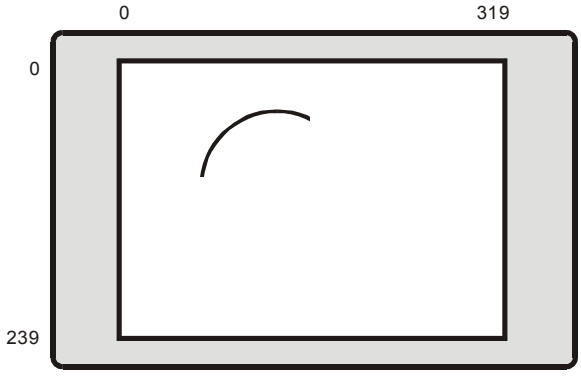


Arc

ARC x, y, r, start, end

완전한 원이 아닌, 원의 일부, 즉 호를 그리는 명령입니다. 그래픽 레이어에 x, y 를 중심점으로 하고,r 을 반지름으로 하는 원에서, start, end 를 각으로 하는 사이의 원을 그립니다. 각도는 0~ 320 이내의 값이어야 하고, 본 명령의 경우 반드시 최대 좌표 계를 초과하지 않는 좌표를 지정해야 제대로 된 호를 그릴 수 있습니다. 이 명령으로 완전한 원을 그릴 수 없으며, 0~320 도 사이의 호만 표현할 수 있습니다.

```
ARC 200,60, 100, 10, 20  '호를 그립니다.
```



Defchr

DEFCHR code, data

Code : 캐릭터 코드 (&hdb30 ~ &hdbff)

Data : 32 바이트의 비트맵 데이터

숫자, 영문자, 한글 이외에 유저가 임의대로 캐릭터를 정의해서 사용할 수 있는 명령입니다. 완성형 한글 코드 중 사용하지 않는 일부 영역에 16 by 16 의 비트맵 데이터 (32 바이트)를 저장해 두었다가, 한글 코드 대신 화면에 표시할 수 있습니다.

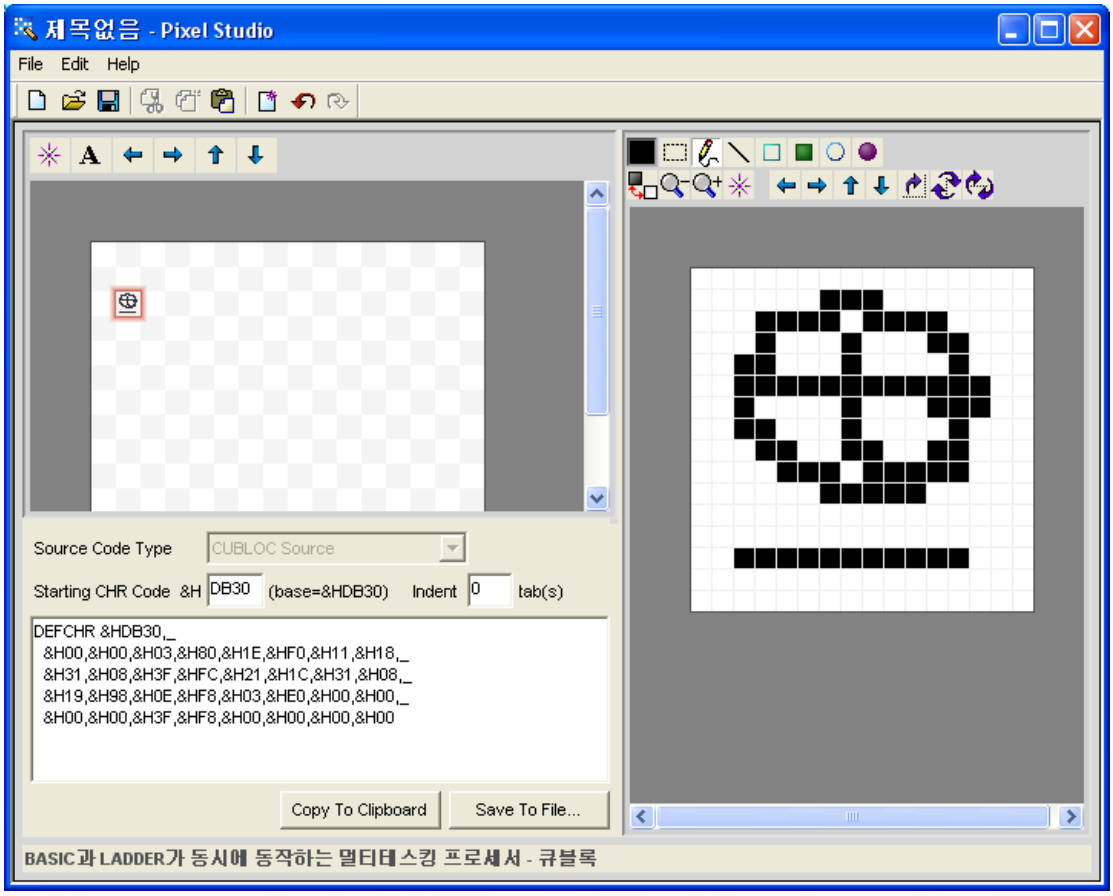
텍스트 레이어, 그래픽 레이어에 모두 표시할 수 있으며, 그래픽 레이어에 표시할 때에는 GPRINT,명령을 사용하고 텍스트레이어에 표시할 때에는 PRINT 명령을 사용합니다. 주로 회사로고나 특별한 캐릭터 등을 따로 정의하여 사용하는 용도로 사용합니다. 총 207 개의 정의 영역이 있으며, 플레쉬 영역에 기록되는 것이 아니므로, 프로그램 맨 처음에 DEFCHR 명령을 가지고 정의해 주어야 합니다.

```
DEFCHR &HDB30,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,_
      &HAA,&HAA,&HAA,&H55,&HAA,&HAA,&HAA,&HAA,
      &HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,
      &HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA

print CHR(&HDB30)
```

표시할 때에는 CHR 함수를 사용해서 표시하면 됩니다.

사실, 이런 방법으로 그림이나 심볼을 그리기란 쉬운 방법이 아닙니다. 저희가 제공하는 Pixel Studio 를 사용하면 보다 편리한 방법으로 그림이나 심볼을 그린 뒤 DEFCHR 명령 포맷으로 바꿀 수 있습니다.

캐릭터 정의를 도와주는 유틸리티 : Pixel Studio



픽셀 스튜디오는 www.comfile.co.kr 에서 다운로드 받을 수 있는 공개 소프트웨어입니다. 그림을 그린 후 DEFCHR 명령 포맷으로 변환시켜주는 유틸리티 소프트웨어 입니다. 이 소프트웨어를 사용하면, 편리하게 심볼 및 그림을 작성할 수 있습니다. 사용방법은 다운로드 후 별도의 사용설명서 파일을 참고하시기 바랍니다.

Bmp

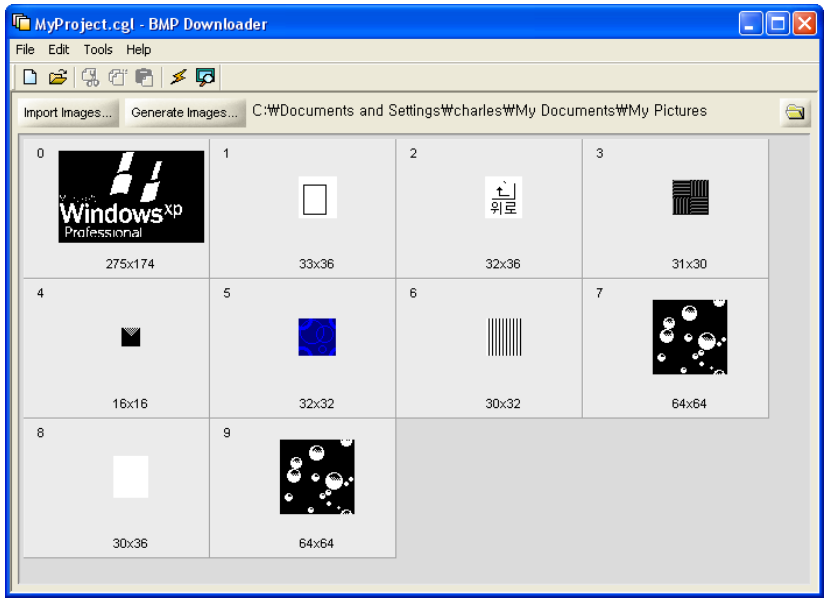
BMP x, y, filenumber, layer

```
X, y : BMP 파일을 표시할 위치
Filenumber : 저장된 BMP 파일의 번호
Layer : 표시할 레이어
```

CT172X/C 에는 BMP 파일을 다운로드 할 수 있는 별도의 FLASH 영역이 있습니다. 저장된 BMP 파일은 나중에 BMP 명령으로 화면에 불러낼 수 있습니다. BMP 파일의 크기는 다양하기 때문에, 표시하고자 하는 위치를 정해 주어야 합니다. 표시하고자 하는 레이어도 따로 정해 줄 수 있습니다.

* CT172X/C 에는 BMP 파일을 다운로드 할 수 있는 104,448 바이트의 공간이 있습니다. 이 용량은 320 x 240 풀 화면을 10 개정도 저장할 수 있는 크기이며, 이 보다 작은 사이즈의 BMP 파일의 경우 다 많은 숫자를 저장할 수 있습니다. 저장할 데이터는 반드시 "흑백 비트맵"형식으로 저장해야 합니다. 플레쉬 영역이므로 프로그램을 다운로드 할 때마다 매번 다운로드 할 필요 없이, 한번만 다운로드 해 놓으면 됩니다.

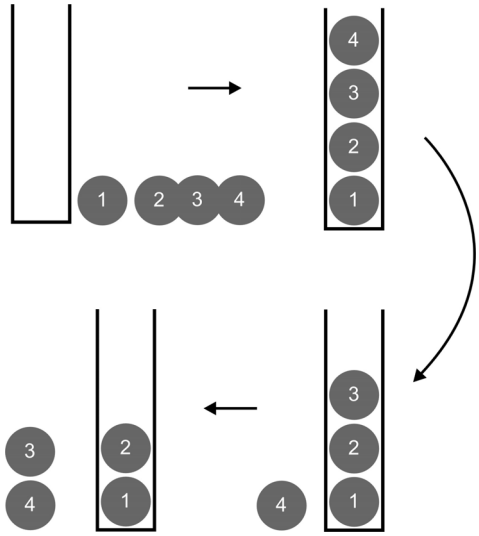
BMP파일을 CT172X/C에 다운로드하기 위해서는 "Bmp Downloader"라는 프로그램이 필요합니다. 이 프로그램은 www.comfile.co.kr에서 다운로드 받을 수 있으며, 사용방법은 다운로드 시 함께 제공되는 사용설명서를 참조하시기 바랍니다.



그래픽 데이터 PUSH, POP 관련 명령

CT172X/C 에는 그래픽 화면의 데이터를 보관할 수 있는 별도의 스택이 있습니다. 이곳에 현재 그래픽화면의 일부분을 보관할 수 있으며, 나중에 보관해놓은 화면을 다시 꺼내 올 수 있습니다. 이러한 기능이 필요한 이유는 화면의 일부분을 다른 곳을 이동하거나 복사해야 될 경우가 생기기 때문입니다. 팝업메뉴 등을 구현할 때, 화면의 일부분을 복사해 두었다가, 다시 복구해야 하는 일이 발생하게 되는 이러한 일련의 작업등을 편리하게 구현할 수 있도록 도와주는 명령군입니다.
그래픽 화면을 도트단위로 세밀하게 잘라서 보관 및 복귀할 수 있는 GPUSH, GPOP 명령이 있고, 바이트 단위로 잘라서 보관 및 복귀를 할 수 있는 HPUSH, HPOP 명령이 있습니다. 도트단위 명령어는 세밀한 데이터를 취급할 수 있는 반면, 복잡한 연산과정을 거치므로 속도가 느린 단점이 있고, 반면 바이트 단위 명령어는 세밀한 데이터를 처리할 순 없지만, 빠른 속도로 보관 및 복구를 할 수 있습니다.
그림데이터를 보관하는 스택은 LIFO (Last in First out)의 구조를 가지고 있어서, 가장 나중에 넣은 데이터를 제일 먼저 POP 할 수 있습니다. 아래 그림을 보시면 쉽게 이해할 수 있습니다. 테니스 공을 넣을 수 있는 통이 있습니다. 한쪽으로만 넣을 수 있고 한쪽은 막혀있는 구조로, **가장 먼저 넣은 공을 가장 나중에 꺼냅니다.**

CT172X/C 에는 32KBYTE 의 그래픽 STACK 이 별도로 내장되어 있으며, 변수를 저장하는 32 K 와는 별도의 영역으로 존재하고 있습니다. 한 화면이 9600 바이트이니까, 전체화면을 저장할 경우 대략 3.4 화면 정도를 저장할 수 있습니다.

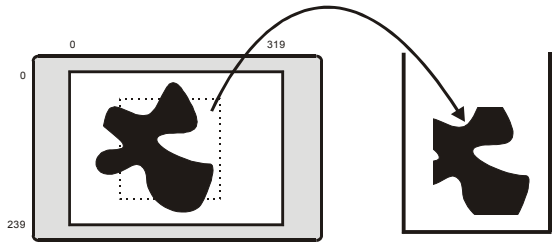


Gpush

GPUSH x1, y1, x2, y2, layer

x1,y1- x2, y2 를 대각선으로 하는 박스의 내부 그림을 스택에 PUSH 합니다. 이 명령으로 인해 화면에는 아무 변화가 생기지는 않습니다. 화면의 데이터를 스택에 보관하는 작업을 하게 됩니다.

```
GPUSH 10,20,200,100,2
```



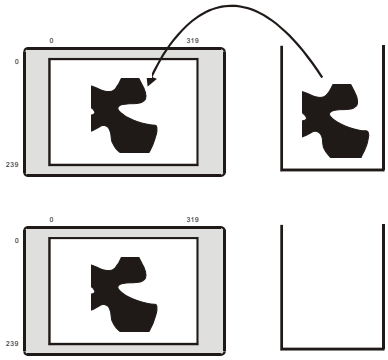
Gpop

GPOP x, y, layer, logic

```
logic =0 : 배경화면과 OR 연산으로 표시
logic =1 : 배경화면과 AND 연산으로 표시
logic =2 : 배경화면과 XOR 연산으로 표시
logic =3: 배경화면을 지운 후 표시
```

스택에 저장된 데이터를 원하는 위치에 복귀시키는 명령입니다. X, y 는 복귀시킬 좌표입니다. 저장해놓은 좌표에 그대로 복귀시켜도 되고, 다른 위치에 복귀시켜도 무방합니다. 이 경우에는 마치 COPY 한 것과 동일한 효과를 낼 수 있습니다. CT172X/C 에는 별도의 GCOPY 명령이 없으므로 그래픽화면의 일부분을 다른 곳에 COPY 하고 싶을 때에는 GPUSH 한 다음 GPOP 하면 됩니다.
GPOP 명령에서 layer 를 따로 지정할 수 도 있습니다. Layer 를 지정하여 다른 레이어에 표시할 수도 있습니다. Logic 을 사용하여 복귀할 때 기존에 있던 바탕화면과의 연산 방법을 지정할 수 있습니다. 기존화면과 겹쳐서 표시하고 싶다면 0 으로 하면 됩니다.

```
GPOP 120,20,2,0
```



Gpaste

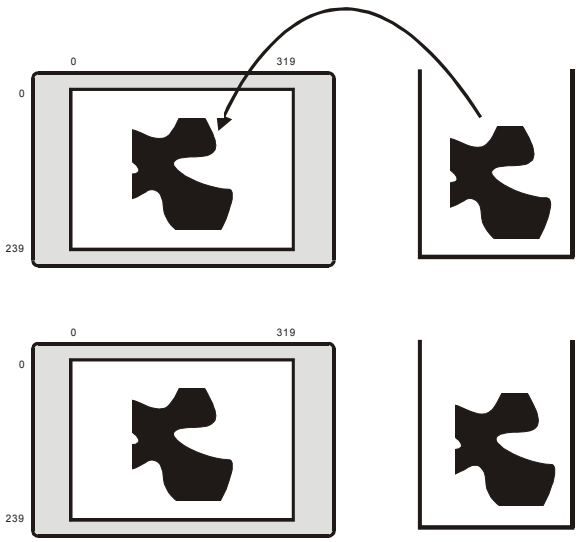
GPASTE x, y, layer, logic

```
logic =0 : 배경화면과 OR 연산으로 표시
logic =1 : 배경화면과 AND 연산으로 표시
logic =2 : 배경화면과 XOR 연산으로 표시
logic =3 : 배경화면을 지운 후 표시
```

스택에 저장된 데이터를 원하는 위치에 복귀시키는 명령입니다. GPOP 과 동일한 명령입니다만 다른 점은 PASTE 는 스택포인터를 조정하지 않기 때문에, 같은 데이터를 여러 번 PASTE 할 수 있습니다. 즉, 한번 PUSH 된 데이터를 한번 POP 해 버리면 더 이상 POP 할 데이터가 없습니다만, PASTE 의 경우 여러 번 PASTE 할 수 있기 때문에, 같은 데이터를 여러 곳에 표시하고 싶을 때 사용하면 유용한 명령입니다. PASTE 를 여러 번 반복해서 원하는 만큼 그래픽 데이터를 표시했다면, 최종적으로 GPOP 을 해야 한다는 것을 잊지 말아야 합니다. 잘못하면 스택 OVERFLOW 가 발생할 수 있습니다.

주의사항

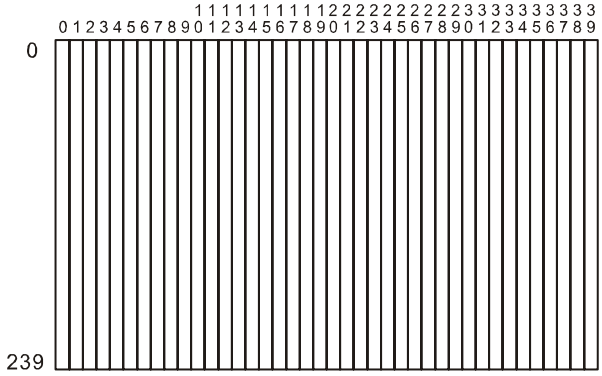
GPUSH 로 저장해 놓은 데이터는 반드시 GPOP, GPASTE 를 통해 복구해야 합니다. HPOP 이나 HPASTE 로 복구하는 경우에는 엉뚱한 데이터가 표시되거나, 표시되지 않을 수 있습니다.



Hpush

HPUSH x1, y1, x2, y2, layer

HPUSH, HPOP, HPASTE 명령의 경우 X 좌표는 반드시 0~39 사이의 값으로 지정해야 합니다. 이 명령어들은 바이트단위로 데이터를 취급하기 때문에, X 축 좌표를 8 로 나누어 0 부터 39 로 지정합니다.



Y 축 좌표는 평상시와 마찬가지로 0~239 로 지정하면 됩니다.
이 명령어는 GPUSH 와 마찬가지로 화면에 있는 데이터를 스택에 저장하기 때문에 화면에는 아무런 변화가 일어나지 않습니다.

```
HPUSH 6,20,12,100,2
```

Hpop

HPOP x, y, layer

스택에 저장된 데이터를 원하는 위치에 복귀시키는 명령입니다. X, y 는 복귀시킬 좌표이고, X 축 좌표는 0~39 사이의 값을 사용해야 합니다. HPOP 명령에서 logic 연산을 지원하지 않습니다. 무조건 배경화면을 지우고 표시합니다.

```
HPOP 10,20,2,0
```

Hpaste

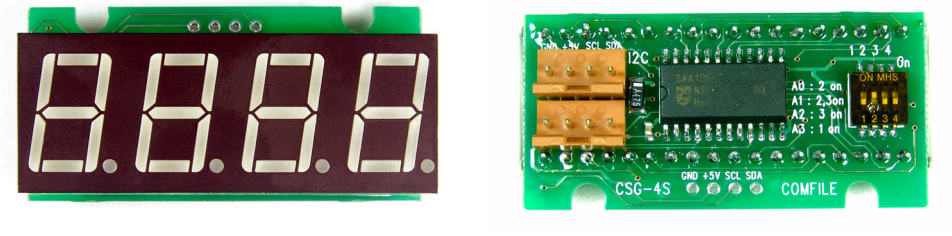
HPaste x, y, layer,

HPOP 과 동일한 명령입니다만 다른 점은 HPASTE 는 스택포인터를 조정하지 않기 때문에, 같은 데이터를 여러 번 PASTE 할 수 있습니다. HPUSH 로 저장해 놓은 데이터는 반드시 HPOP, HPASTE 를 통해 복구해야 합니다. GPOP 이나 GPASTE 로 복구하는 경우에는 엉뚱한 데이터가 표시되거나, 표시되지 않을 수 있습니다.

세븐세그먼트 디스플레이 CSG 시리즈

간단한 숫자 등을 디스플레이 하기 위해 "세븐 세그먼트"를 많이 사용합니다. 세븐 세그먼트는 8 개의 LED 를 숫자모양이 되도록 배치해 놓은 것으로, 실생활에서 많이 볼 수 있는 가장 대중적인 디스플레이 소자입니다.

세븐세그먼트를 구동하기 위해서는 다소 복잡한 "다이나믹 디스플레이"라는 방법을 사용해야 하므로, 의외로 사용하기 까다로운 디스플레이 소자입니다. 좀더 쉽게 세븐세그먼트에 숫자를 표시하기 위해서 CSG 모듈이라는 제품을 시판하고 있습니다.



그림에서 볼 수 있듯이, 제품의 전면에는 4 DIGIT 의 세븐세그먼트가 있고, 뒷면에는 I2C 연결을 위한 접속 케이블이 있습니다. 큐블록과 I2C 접속을 연결하신 뒤 몇 가지 명령어를 사용하는 것만으로 쉽게 숫자를 표시할 수 있습니다. 다음은 CSG 모듈을 구동하기 위해 사용하는 라이브러리 명령어들입니다.

사용명령	설명	사용 예
CSGDEC SlaveAdr, Data	10 진으로 숫자를 표시	CSGDEC 0, I
CSGHEX SlaveAdr, Data	16 진으로 숫자를 표시	CSGHEX 0,I
CSGNPUT SlaveAdr, Digit, Data	4 Digit 중 원하는 자리에 숫자를 표시	CSGNPUT 0,0,8
CSGXPUT SlaveAdr, Digit, Data	4 Digit 중 원하는 자리에 2 진 데이터로 표시	CSGNPUT 0,0,9

CSGDEC 라는 명령어만 사용해도 간단하게 10 진수 형식으로 세븐세그먼트에 숫자를 표시할 수 있습니다.

```
Const Device = cb280
Set I2c 9,8
b=8
Do
      Csgdec 0,b
      Delay 100
      b = b + 1
      If b=0 Then b=200
Loop
```

CSG 관련명령을 사용하기 위해서는 반드시 **프로그램 맨 앞부분에 SET I2C 9,8 명령을 선언**해 주어야 합니다. 9 번 핀에 SDA, 8 번 핀에 SCL 단자가 연결되어 있다는 선언문입니다.

만약 9,8 번 I/O 핀이 아닌 다른 I/O 를 사용하려면, SET I2C 명령에서 I/O 포트 숫자만 바꾸어주면 됩니다.

실제로 CSGDEC 는 다음과 같은 부 프로그램으로 구성된 라이브러리입니다.

```
Sub CsgDec(  cssla As Byte,   csdata As Integer)
    Dim   csstr As String * 4
    __csstr = Dec4 __csdata
    csgnput   cssla,0,  csstr a(0)
    csgnput   cssla,1,  csstr a(1)
    csgnput __cssla,2,__csstr_a(2)
    csgnput   cssla,3,  csstr a(3)
End Sub
```

이 부 프로그램에서 CSGNPUT 을 호출하는 것을 알 수 있습니다. CSGNPUT 이 CSG 모듈을 구동하기 위한 가장 기본적인 라이브러리인 셈입니다.

CSGNPUT 은 4 DIGIT 의 자리 중에서 원하는 위치에 원하는 숫자를 표시할 수 있는 명령입니다. 이 명령을 직접 사용한다면 좀더 자유롭게 CSG 모듈을 구동할 수 있습니다. 만약, 점을 찍고 싶다면, 아래의 예에서처럼 Data 위치에 &H80 을 더해주면 됩니다. 즉, 상위 비트가 1 이면 해당 위치에 점과 함께 표시하는 것입니다.

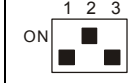
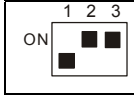
```
csgnput __cssla,2,__csstr_a(2) + &H80
```

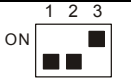
CSGNPUT 은 숫자 0~9 그리고 16 진 표시를 위하여 A~F 까지 표시합니다. 0~F 이외의 값은 표시하지 않습니다.

CSGXPUT 명령어는 앞의 명령과 비슷하지만, 8 개의 LED 를 개별적으로 제어할 수 있는 명령어입니다. 숫자 이외의 형태를 표시하고자 할 때에 이 명령어를 사용하시기 바랍니다.

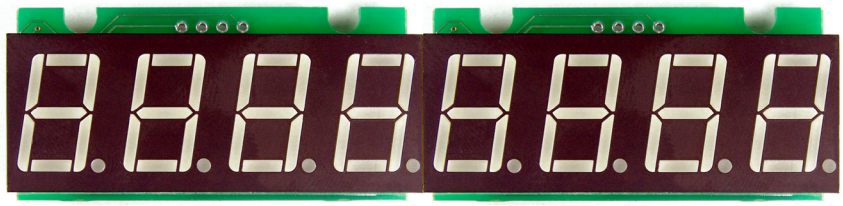
Slave Address

CSG 모듈 뒷면에는 Slave Address 를 조정할 수 있는 딥스위치가 있습니다 .0 부터 3 까지 4 개의 어드레스를 셋팅할 수 있으므로, 하나의 I2C 라인에 최대 4 개의 CSG 모듈을 연결할 수 있습니다.

DIP 스위치 상태	슬레이브 어드레스
	0
	1

DIP 스위치 상태	슬레이브 어드레스
	2
	3

4 Digit 이상의 숫자를 표시하려면, 2 개의 CSG 모듈을 아래 사진처럼 나란히 연결한 뒤 슬레이브 어드레스를 서로 다르게 해서 구동합니다.



Csgnput

CSGNPUT slaveadr, digit, data

```
slaveadr : CSG 모듈의 Slave Address
digit : 위치 (0~3)
data : 표시 데이터 (&h30~&h39, &h41~&h4f) 이외 값은 표시안함
```

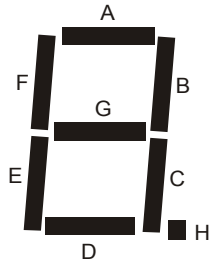
CSG 모듈의 원하는 위치에 원하는 숫자를 표시하는 명령입니다. DATA 의 상위 1 비트는 Dot 를 ON 할 때 사용하는 비트입니다. data 는 ASCII 코드로 써주어야 하며, 숫자 0~9, 영문자 A~F 까지만 표시할 수 있습니다.
이 명령은 부프로그램 라이브러리이며, 소스 앞부분에 SET I2C 명령을 선언해주어야 동작합니다.

Csgxput

CSGXPUT slaveadr, digit, data

```
slaveadr : CSG 모듈의 Slave Address
digit : 위치 (0~3)
data : 표시 데이터
```

CSG 모듈의 원하는 위치에 원하는 LED 를 ON 하는 명령입니다. 8 개의 LED 를 각각 ON /OFF 할 수 있으므로, Csgnput 명령으로 표현할 수 없는 숫자 이외의 모양을 표현하고 싶을 때 사용합니다.



비트	7	6	5	4	3	2	1	0
LED	H	G	F	E	D	C	B	A

영문자 L 을 표현하고 싶다면 F, E, D 를 ON 해야 합니다. 그러면 비트구성이 0011 1000 이 되므로, 16 진수로는 &H38 이 됩니다. CSGXPUT 0, 0, &H38 과 같은 식으로 명령을 작성하는 것입니다. 이 명령은 부 프로그램 라이브러리이며, 소스 앞에 SET I2C 명령을 선언해주어야 동작합니다.

Csgdec

CSGDEC slaveadr, data

slaveadr : CSG 모듈의 Slave Address
data : 표시 데이터

10 진수 형식으로 데이터를 표시합니다.
이 명령은 부 프로그램 라이브러리이며, 소스 앞부분에 SET I2C 명령을 선언해주어야 동작합니다

Csghex

CSGHEX slaveadr, data

slaveadr : CSG 모듈의 Slave Address
data : 표시 데이터

16 진수 형식으로 데이터를 표시합니다. 소스 앞부분에 SET I2C 명령을 선언해주어야 동작합니다

제 10 장
CUBLOC
부 프로그램
라이브러리

부 프로그램 라이브러리

앞에서 설명한 라이브러리는 CUBLOC 인터프리터에서 단일 명령어의 형식으로 지원하는 라이브러리입니다. 부 프로그램 라이브러리는 "단일명령어"형식이 아닌 CUBLOC 의 부 프로그램의 형태로 지원되는 라이브러리입니다. 프로그램 작성시 자주 사용하게 되는 기능들을 부 프로그램으로 구현해 놓은 것으로, 유저는 명령어를 사용하는 것처럼 편리하게 사용할 수 있습니다. 시스템 부 프로그램 라이브러리로 분류된 명령어는 대부분 실행하는데 시간이 걸리는 명령들입니다. 예를 들어 DELAY 명령을 기본 명령어로 하게 되면, 인터럽트 발생에 영향을 주는 등, 프로그램의 원활한 흐름을 방해하게 되므로, 부 프로그램 형태로 지원하는 것입니다. 부 프로그램은 결국에는 CUBLOC 소스로 번역되므로, 인터럽트 발생에 영향을 주지 않습니다.

Delay

DELAY value

밀리 초단위로 시간지연을 하기 위한 SUB 형 라이브러리입니다. DELAY 100 을 하면 약 100 밀리 초를 딜레이 합니다. DELAY 명령에 의한 시간 지연은 정확한 시간을 측정하여 딜레이 하는 것이 아니라, FOR...NEXT 루프에 의한 루프반복수행으로 시간을 지연시키는 것이므로, 본 명령어로 정밀한 시간관리를 요구하는 목적으로 사용하는 것은 적합하지 않습니다. DELAY 에서의 시간은 대략적인 시간을 의미합니다.

```
DELAY 10          ' 약 10 밀리 초를 딜레이 합니다.
DELAY 200         ' 약 200 밀리 초를 딜레이 합니다.
```

좀더 정확한 시간단위로 딜레이를 하려면 Wait 명령을 사용하시기 바랍니다. Wait 명령은 시스템 클록을 사용한 딜레이 명령이므로, 정확한 시간을 측정하여 딜레이를 수행합니다. 즉 Wait 2000 이라고 하면 정확히 2 초를 기다립니다. 단 이 명령은 10mS 이하 시간은 무시하므로 주의하시기 바랍니다. 예를들어 Wait 15 라고 하면 10 밀리초만 딜레이합니다.

Pause

PAUSE value

DELAY 와 완전히 동일한 역할을 수행하는 라이브러리입니다.

Udelay

UDELAY value

아주 작은 단위의 시간을 지연시키고자 할 때 사용하는 명령으로, 기본 70~80 마이크로 초의 시간에 value 당 8~9 마이크로 초를 더 지연시킵니다. 예를 들어 Udelay 10 이라고 했을 경우, 80 + 80 정도인 160 마이크로 초를 지연시킵니다. (Udelay 0 일 경우는 80 마이크로 초). 이 명령어는 실행 시 LADDER 가 동시에 실행된다면, LADDER 의 실행시간에 따라 지연시간이 영향을 받게 됩니다. 또한 이 명령 수행 시 BASIC 인터럽트의 수행이 가능하므로, BASIC 인터럽트 발생시 지연시간 역시 영향을 받게 됩니다.
Delay,시간에 영향이 없도록 하고 싶다면, 레더실행과 전체 인터럽트를 중지시켜야 합니다.

Keyin

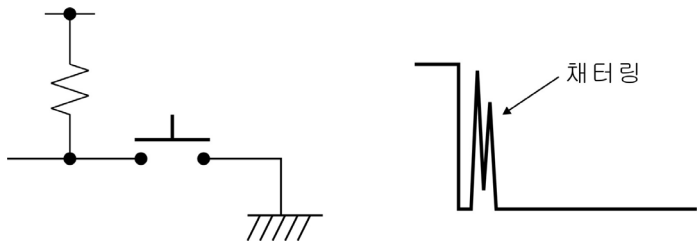
Variable = KEYIN(pin, chattime)

```
Variable : 결과가 저장될 변수 (BYTE 형을 리턴)
Pin : 입력 사용 가능한 I/O 핀을 가리키는 변수 또는 상수
Chattime : 채터링 제거 시간
```

채터링이 제거된 키 입력을 받아들이는 FUNCTION 형 라이브러리입니다. KEYIN 은 아래그림처럼 LOW ACTIVE 입력 시에만 사용할 수 있습니다. HIGH ACTIVE 입력 시에는 KEYINH 를 사용해야 합니다. 결과는 입력 시 0, 입력이 없을 시에 1 이 리턴됩니다.

채터링 제거시간에 10 을 쓰면 약 10 밀리초동안 채터링 입력여부를 체크하게 됩니다. 100 을 쓰면 약 100 밀리초 동안 채터링여부를 체크합니다. 채터링은 대략 10 밀리 초 전후에서 발생되므로 10 전후의 값을 써주는 것이 무난합니다.

```
A = KEYIN(1,10) '1 번 포트에서 10mS 간격으로 채터링이 제거된 입력을 받아온다.
```



Keyinh

Variable = KEYINH(pin, chattime)

```
Variable : 결과가 저장될 변수 (BYTE 형을 리턴)
Pin : 입력 사용 가능한 I/O 핀을 가리키는 변수 또는 상수
Chattime : 채터링 제거 시간
```

KEYINH 은 HIGH ACTIVE 입력 시에만 사용할 수 있습니다. LOW ACTIVE 입력 시에는 KEYIN 를 사용해야 합니다. 결과는 입력 시 1, 입력이 없을 시에 0 이 리턴됩니다. 채터링 제거시간은 KEYIN 의 경우와 동일합니다.

```
A = KEYINH(1,100) '1 번 포트에서 100 간격으로 채터링이 제거된 입력을 받아온다.
```

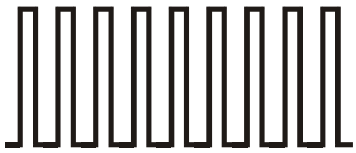
Beep

BEEP Pin, Length

```
Pin : 출력 가능한 I/O 핀을 가리키는 변수 또는 상수
Length : 펄스 출력 시간
```

Beep 사운드를 발생시키는 SUB 형 라이브러리입니다. I/O 핀 중 하나에 PIEZO 나 Speaker 를 연결한 뒤 이 라이브러리를 사용하면 짧은 BEEP 사운드가 나옵니다. 키터치음이나 경고음 등을 발생시키는 용도로 사용하시기 바랍니다. 이 라이브러리가 실행되면 해당 포트는 자동적으로 출력상태가 됩니다.

```
BEEP 2, 100   '2 번포트에 100 기간 동안 BEEP 사운드 출력
```



Pulsout

PULSOUT Pin, Period

```
Pin : 출력 가능한 I/O 핀을 가리키는 변수 또는 상수
Period : 펄스 간격
```

단 한번의 펄스를 출력시키는 SUB 형 라이브러리입니다. High 펄스가 나오게 하려면 해당 I/O 핀을 미리 LOW 상태로 만들어 두어야 합니다. 반대로 Low 펄스가 나오게 하려면 미리 HIGH 상태로 만들어 두어야 합니다. Period 를 10 으로 하면 2.5mS 정도의 펄스가 출력되고, 100 으로 하면 25mS 정도의 펄스가 출력됩니다.

```
LOW 2
PULSOUT 2, 100   '10mS HIGH 펄스
```

```
HIGH 2
PULSOUT 2, 100   '10mS LOW 펄스
```

지속적이면서 일정한 주파수를 유지하는 펄스를 발생시키려면 Freqout 명령을 사용하십시오.

Tadin

Variable = TADIN(channel)

```
Variable : 결과가 저장될 변수 (INTEGER 형을 리턴)
channel : AD 입력 채널 (포트 번호가 아님)
```

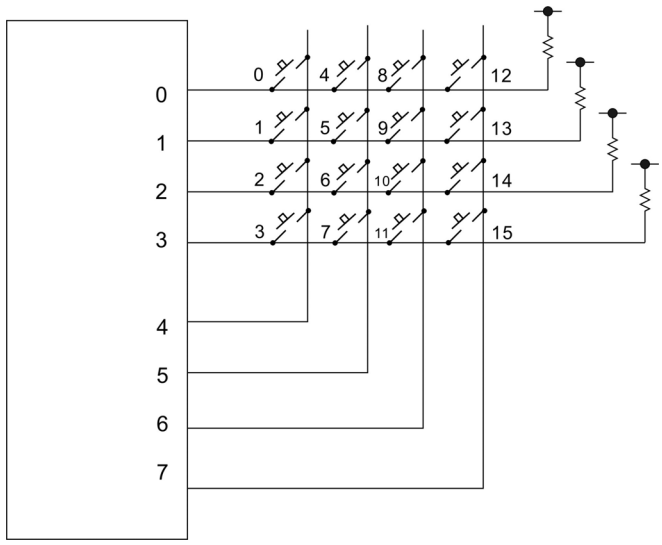
해당채널을 10 번 연속으로 ADIN 변환을 한 값을 평균 내어 반환하는 라이브러리입니다. 일정한 노이즈가 섞여서 들어오는 입력신호의 경우 TADIN 로 A/D 변환을 수행하면 보다 정밀한 결과를 얻을 수 있습니다.

Keypad

Variable = KEYPAD(portblock)

```
Variable : 결과가 저장될 변수 (BYTE 형을 리턴)
Portblock : 키입력받을 포트 블록
```

키 매트릭스 입력을 받을 수 있는 라이브러리입니다. 하나의 포트 블록에 연결된 최대 4 x 4 키 매트릭스의 입력 값을 읽어오게 됩니다. 포트 블록의 하위 4 비트는 입력 포트로, 상위 4 비트는 출력 포트로 사용합니다. 다음 회로는 포트블록 0 에 연결된 키 매트릭스의 회로입니다.




```
A = KEYPAD(0) '0 번 포트블록에 연결된 키 매트릭스의 상태를 읽어옵니다.
```

아무 키도 눌려지지 않았다면 255 를 리턴하고, 눌려진 키가 있다면 그 위치의 키 스캔코드 값을 리턴합니다.

Ekeypad

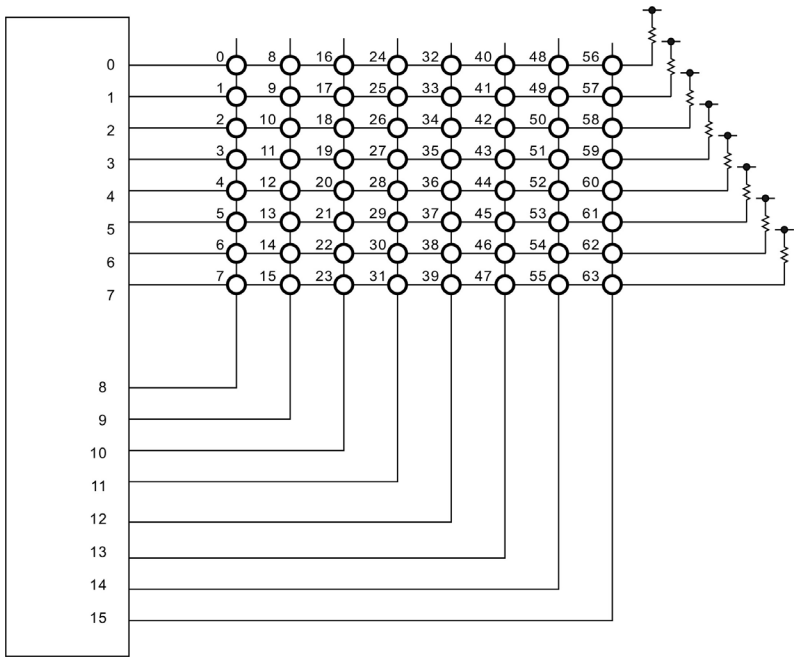
Variable = EKEYPAD(portblockIn, portblockOut)

```
Variable : 결과가 저장될 변수 (BYTE 형을 리턴)
PortblockIn : 키 입력 받을 포트 블록 (입력 측)
PortblockOut : 키 입력 받을 포트 블록 (출력 측)
```

KEYPAD 명령을 좀더 확장하여 최대 64 키까지 읽을 수 있는 명령입니다. 두 개의 포트 블록에 연결된 8 x 8 키 매트릭스의 키 입력 상태까지 읽어올 수 있습니다. 입력 측 포트블록과 출력 측 포트블록을 따로 지정해 주어야 합니다.입력 측 포트블록은 입력으로 사용 가능한 포트블록을 사용해야 합니다. 마찬가지고 출력 측 포트블록은 출력으로 사용하는 한 포트블록을 사용해야 합니다.

입력 측 포트블록 중 사용하지 핀은 반드시 저항을 사용해서 5V 와 연결해두어야 합니다. 이 핀을 다른 용도로 사용하는 것은 불가능합니다.

출력 측 포트블록 중 사용하지 않는 핀은 OPEN 상태로 두십시오. 이 핀을 다른 용도로 사용할 수 없습니다. 다음 회로는 포트블록 0 을 입력 측으로 블록 1 을 출력 측으로 사용한 경우의 회로입니다.



아무 키도 눌려지지 않았다면 255 를 리턴하고, 눌려진 키가 있다면 그 위치의 키 스캔코드 값을 리턴합니다.

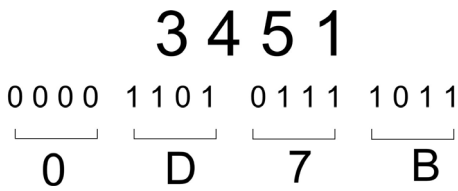
Bin2bcd

Variable = BIN2BCD(binvalue)

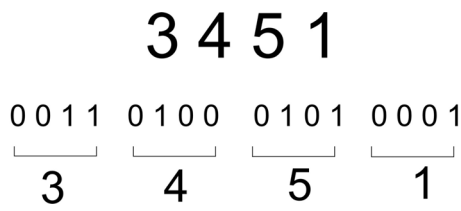
Variable : 결과가 저장될 변수 (LONG 형을 리턴)
binvalue : 변환할 바이너리 값

바이너리 값을 BCD 코드로 변환해주는 라이브러리입니다. BCD 코드는 니블단위로 10 진수를 표현하는 방법입니다.

예를 들어 3451 을 바이너리 (2 진수)로 표현하면 아래와 같이 됩니다. 십진수 3451 은 0000 1101 0111 1011 로 저장됩니다. 16 진으로 읽어보면 0d7b 가 됩니다. 2 진 값을 보고 10 진 값을 바로 알 수는 없습니다.



2 진 값을 보고 10 진수를 쉽게 변환하기 위해 만든 코드체계가 바로 BCD 코드입니다. 3451 을 BCD 로 표시하면 다음과 같습니다.



2 진 값을 읽으면 곧바로 십진 값을 알 수 있기 때문에, BCD 코드는 자주 사용됩니다. BIN2BCD 명령을 이용하면 바이너리 값을 BCD 코드로 변환해 줍니다.

```
I=123456
j = bin2bcd(i)
```

Bcd2bin

Variable = BCD2BIN(bcdvalue)

Variable : 결과가 저장될 변수 (LONG 형을 리턴)
bcdvalue : 변환할 BCD 값

위 명령과는 반대로 BCD 코드를 바이너리 값으로 변환해주는 라이브러리입니다.

CFBSLIB.BAS

CUBLOC STUDIO 가 설치된 폴더안에는 CFBSLIB.BAS 라는 파일이 들어 있습니다. 이 파일이 바로 부프로그램 라이브러리가 저장되어 있는 파일입니다. 다음은 CFBSLIB.BAS 의 일부분 입니다. 이 소스를 잘 보시면 어떤식으로 부프로그램 라이브러리가 동작하는지 아실 수 있습니다.

```
'
'
'   CFBASIC System Libary V1.0
'
'         COMFILE Technology
End

Sub Delay(  dl As Long)
   __dl1 Var Long
   __dl2 Var Integer
   For   dl1=0 To   dl
   For   dl2=0 To 1
   Nop
   Nop
   Nop
   Next
   Next
End Sub

#ifdef __usepause
Sub Pause(  dl As Long)
     dl1 Var Long
   __dl2 Var Integer
   For __dl1=0 To __dl
   For   dl2=0 To 1
   Nop
   Nop
   Nop
   Next
   Next
End Sub
#Endif

#ifdef __useeewrite
Sub Eewrite(  ad As Integer,   dt As Long,   ln As Byte)
    Dim   eei As Byte
    For __eei = 1 To __ln
          Do
          Loop Until Sys(4)=0
          eew __ad,__dt
          Incr __ad
            dt =   dt >> 8
    Next
End Sub
#Endif
```

```
#ifdef   usekeyin
Function Keyin(  pt As Byte,   dl As Integer) As Byte
   Dim __dl1 As Integer
   For   dl1 = 0 To   dl
      If In(  pt) = 1 Then
         Keyin = 1
         Exit Function
      End If
   Next
   Keyin = 0
End Function
#Endif

#ifdef __usekeyinh
Function Keyinh(  pt As Byte,   dl As Integer) As Byte
   Dim   dl1 As Integer
   For __dl1 = 0 To __dl
      If In(  pt) = 0 Then
         Keyinh = 0
         Exit Function
      End If
   Next
   Keyinh = 1
End Function
#Endif

#ifdef __usebeep
Sub Beep(  pt As Byte,   ln As Integer)
   Dim   k As Integer,  m As Integer
   __k = 0
   Do
      High   pt
      Incr   k
      Low __pt
   Loop Until   k =   ln
End Sub
#Endif

#ifdef   usetadin
Function Tadin(  num As Byte) As Integer
    Dim __ii As Integer, __ta As Long
      ta = 0
    For   ii = 0 To 9
          __ta = __ta + Adin(__num)
    Next
    Tadin =   TA / 10
End Function
#Endif

#ifdef   useCsgnput
Sub Csgnput(__cssla As Byte, __csdigit As Byte, __csdata As Byte)
```

```
     Const Byte __cs_convtb = (&h3f,&h6,&h5b,&h4f,&h66,&h6d,&h7d,&h27,_
&h7f,&h6f,&h5f,&h7c,&h39,&h5e,&h79,&h71)
     Dim   cs temp As Byte,   cs dot As Byte
     If __cssla = 255 Then Exit Sub
     I2cstart
       cssla =   cssla * 2 + &h70
     __cs_temp = I2cwrite(__cssla)
     __cs_temp = I2cwrite(&h0)
       cs temp = I2cwrite(&b00110111) 'control byte
     I2cstop
     I2cstart
       cs temp = I2cwrite(  cssla)
       cs temp = I2cwrite(  csdigit+1)
     __cs_dot = __csdata And 128
     __csdata = __csdata And &h7f
       csdata =   csdata - &h30
     If   csdata > 10 Then   csdata =   csdata - 7
     If __csdata < 16 Then
            cs temp =   cs convtb(  csdata)
     Else
           __cs_temp = 0
     End If
       cs temp =   cs temp +   cs dot
     __cs_temp = I2cwrite(__cs_temp)
     I2cstop
  End Sub
  #Endif

  #ifdef   usecsgxput
  Sub Csgxput(  cssla As Byte,   csdigit As Byte,   csdata As Byte)
     Dim __cs_temp As Byte
     I2cstart
       cssla =   cssla * 2 + &h70
       cs temp = I2cwrite(  cssla)
     __cs_temp = I2cwrite(&h0)
       cs temp = I2cwrite(&b00110111) 'control byte
     I2cstop
     I2cstart
     __cs_temp = I2cwrite(__cssla)
       cs temp = I2cwrite(  csdigit+1)
       cs temp = I2cwrite(  csdata)
     I2cstop
  End Sub
  #Endif

  #ifdef __usecsgdec
  Sub Csgdec(  cssla As Byte,   csdata As Integer)
     Dim   csstr As String * 4
     __csstr = Dec4 __csdata
     Csgnput   cssla,0,  csstr a(0)
     Csgnput   cssla,1,  csstr a(1)
     Csgnput __cssla,2,__csstr_a(2)
```

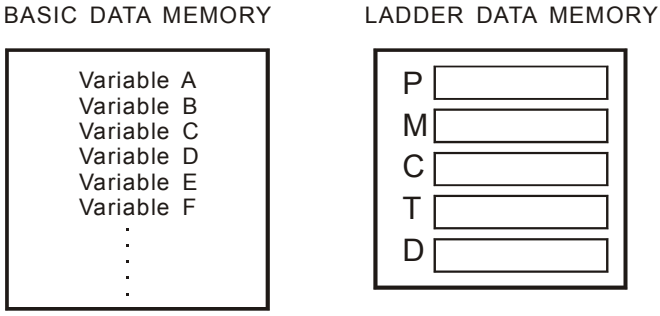
```
    Csgnput __cssla,3,__csstr_a(3)
End Sub
#Endif

#ifdef   usecsghex
Sub Csghex(  cssla As Byte,   csdata As Integer)
    Dim __csstr As String * 4
    __csstr = hex4 __csdata
    Csgnput   cssla,0,  csstr a(0)
    Csgnput   cssla,1,  csstr a(1)
    Csgnput __cssla,2,__csstr_a(2)
    Csgnput   cssla,3,  csstr a(3)
End Sub
#Endif

#ifdef   usekeypad
Function   KEYPAD SUB(  pb As Byte) As Byte
    Dim __pbr As Byte
      pbr =   pb * 8
      KEYPAD SUB = 0
    If In(__pbr) = 0 Then Exit Function
      KEYPAD SUB = 1
    Incr   pbr
    If In(__pbr) = 0 Then Exit Function
    __KEYPAD_SUB = 2
    Incr   pbr
    If In(  pbr) = 0 Then Exit Function
    __KEYPAD_SUB = 3
    Incr   pbr
    If In(  pbr) = 0 Then Exit Function
    __KEYPAD_SUB = &HFF
End Function

Function Keypad(  pb As Byte) As Byte
    Dim __KP As Byte
    Byteout   pb, &B11101111
      KP =   KEYPAD SUB(  pb)
    If __KP < &HFF Then
         Keypad = __KP
         Exit Function
    End If
    Byteout __pb, &B11011111
      KP =   KEYPAD SUB(  pb)
    If   KP < &HFF Then
         Keypad = __KP + 4
         Exit Function
    End If
    Byteout   pb, &B10111111
    __KP = __KEYPAD_SUB(__pb)
    If   KP < &HFF Then
         Keypad =   KP + 8
         Exit Function
```

```
    End If
    Byteout   pb, &B01111111
      KP =   KEYPAD SUB(  pb)
    If __KP < &HFF Then
         Keypad =   KP + 12
         Exit Function
    End If
    Keypad = &HFF
End Function
#Endif
```

제 11 장 BASIC 과 LADDER 의 LINK

데이터 공유

CUBLOC 에서 BASIC 과 LADDER 는 각각 고유의 데이터 영역이 존재합니다.



BASIC 에서 LADDER 의 데이터 영역을 억세스 할 수 있습니다. 이를 위해서 BASIC 에서 다음과 같은 시스템 배열 변수를 지원하고 있습니다. 이 변수를 억세스 하는 것만으로, 쉽게 LADDER 의 데이터 영역에 값을 기입하거나 읽어올 수 있습니다.

레더 억세스용 시스템 배열	억세스 단위	LADDER 의 해당영역
_P	비트 단위 _P(0) ~ P(127)	P 릴레이 영역
_M	비트 단위 _P(0) ~ P(511)	M 릴레이 영역
_WP	워드단위 _WP(0) ~ _WP(7)	P 영역을 워드단위로 억세스
_WM	워드단위 _WM(0) ~ _WM(31)	M 영역을 워드단위로 억세스
_T	워드단위 _T(0) ~ _T(99)	T 영역 (타이머)
_C	워드단위 _C(0) ~ _C(49)	C 영역 (카운터)
_D	워드단위 _D(0) ~ _D(99)	D 영역 (데이터)

P 와 M 릴레이는 비트 단위로 억세스 되고, 나머지 C,T,D 영역은 워드 단위로 억세스 됩니다. P 와 M 영역을 워드 단위로 억세스 하려면 _WP, _WD 를 사용합니다. 예를 들어 _WP(0)은 P0 부터 P15 까지 를 1 워드로 합친 것을 의미합니다. 다음은 BASIC 에서 LADDER 데이터 영역을 억세스 하는 샘플 프로그램입니다.

```
 D(0) = 1234
_D(1) = 3456
 D(2) = 100
FOR I = 0 TO 99
    _M(I) = 0
NEXT
IF _P(3) = 1 THEN _M(127) = 1
```

거꾸로 LADDER 에서 BASIC 변수를 참조하거나, 값을 바꾸는 것은 불가능하며, F 릴레이 영역은 억세스 할 수 없습니다.

ALIAS 로 지정한 영역을 BASIC 에서 사용

ALIAS 명령은 LADDER 에서 사용할 별명을 지정하는 명령입니다. “별명”으로 지정된 이름을 BASIC 프로그램에서 사용한다면, 해당영역에 값을 기입하거나 읽어올 수 있습니다.

```
USEPIN 0,IN,START
USEPIN 1,OUT,RELAY
ALIAS M0 = MOTORSTATE
ALIAS M1 = RELAY1STATE
ALIAS T1 = SUBTIMER

RELAY = 0            ' 1 번 포트를 LOW 상태로 만듭니다.
MOTORSTATE = 1       ' M0 를 1 로 만듭니다. _M(0) = 1 과 같은 동작을 합니다.

A = RELAY1STATE      ' 변수 A 에 M1 상태를 저장합니다.
B = SUBTIMER         ' 변수 B 에 T1 의 내용을 저장합니다.
```

LADDER 만 사용

BASIC 을 사용하지 않아도 된다면, CUBLOC 에서는 LADDER 만으로 프로그램을 작성할 수 있습니다. 하지만 최소한의 BASIC 프로그램은 작성해 주어야 합니다. I/O 포트의 사용선언과 LADDER 기동을 위한 명령입니다.

```
CONST DEVICE = CB280       '디바이스 모델 선택

USEPIN 0,IN,START          '사용 포트 선언
USEPIN 1,OUT,RELAY

ALIAS M0 = MOTORSTATE      '별명 지정
ALIAS M1 = RELAY1STATE

SET LADDER ON              '레더 시작
```

BASIC 에서 디바이스모델 선언, ALIAS 로 별명선언, I/O 포트의 설정, LADDER 의 기동, 등을 처리합니다.

BASIC 만 사용

CUBLOC 의 BASIC 기능만을 이용하고 싶다면, 레더를 기동하는 어떠한 명령도 사용하지 않은 상태에서 사용하시면 됩니다. SET LADDER ON 이나 LADDERSCAN 과 같은 명령어가 없다면 LADDER 가 있어도 실행되지 않기 때문에 BASIC 프로그램만 동작하게 됩니다.

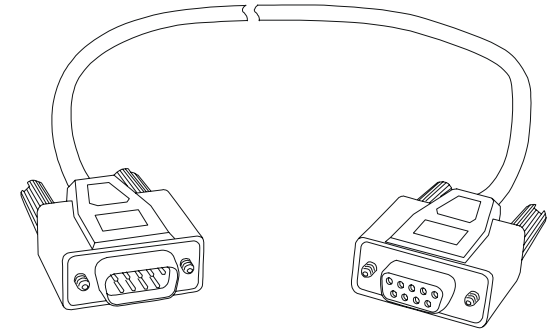
```
SET LADDER ON  '이 명령어만 없다면 LADDER 는 실행되지 않습니다.
LADDERSCAN     '이 명령어도 사용하지 않아야 합니다.
```

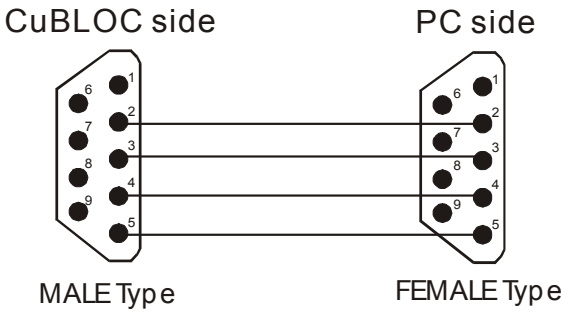
MEMO

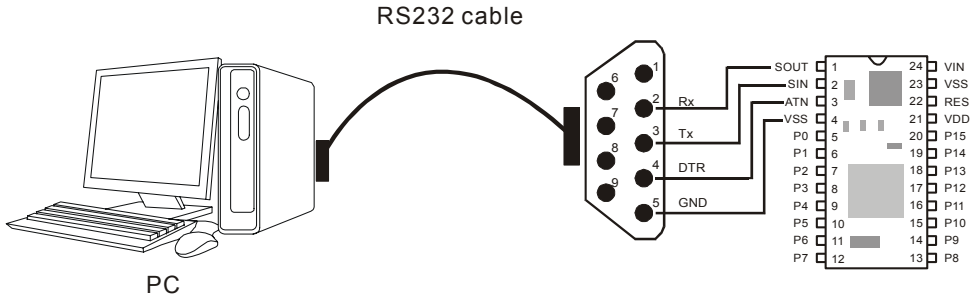
제 12 장
외부 I/O 연결

다운로드 케이블 연결

CUBLOC 을 사용하기 위해서 우선적으로 PC 와의 RS232 다운로드 케이블을 연결해야 합니다. 다운로드 케이블은 1:1 연결을 사용하며 총 9 가닥 중 4 가닥의 신호 선만을 사용합니다.





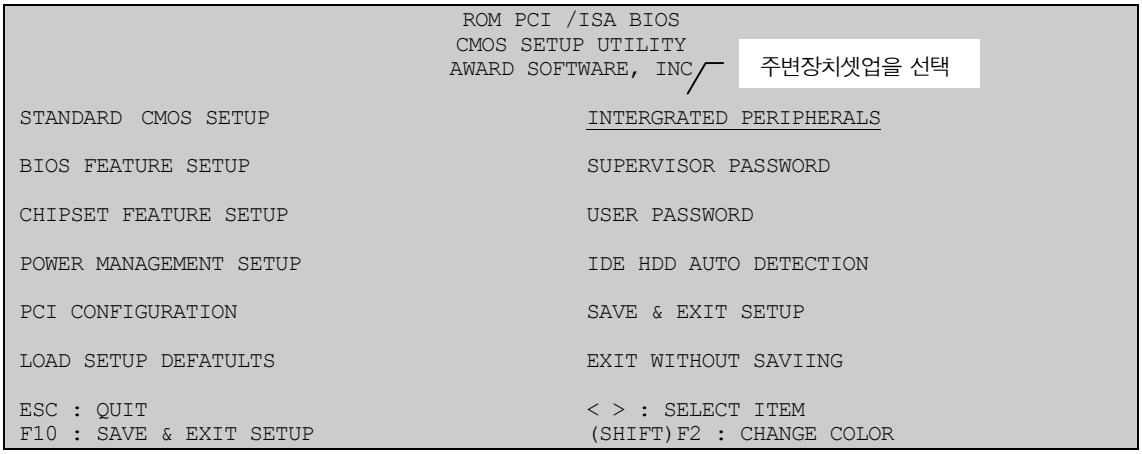


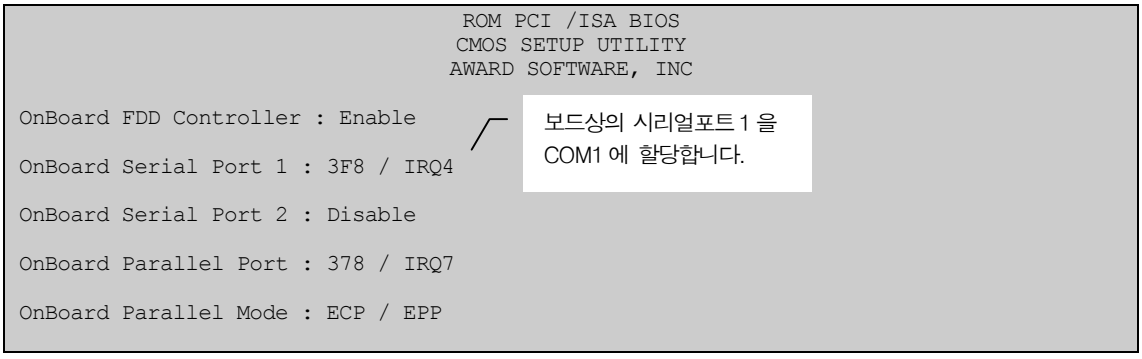
PC 통신포트의 점검

간혹 PC 상의 통신포트 설정이 잘못되어 있어, CUBLOC 다운로드가 안 되는 경우가 있습니다. PC 에는 보통 1~2 개의 RS232 통신포트가 내장되어 있지만, 이 통신포트를 사용 가능한 상태로 만들어 주는 것은 유저의 몫입니다. PC 의 RS232 통신포트를 제대로 동작시키기 위해서는 BIOS 셋업과 윈도우 드라이버설치까지 제대로 구성되어 있어야 하기 때문입니다.

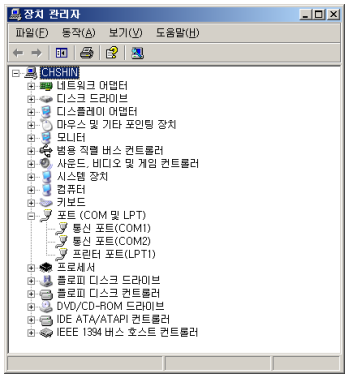
우선 BIOS 상에서 RS232 통신포트 설정방법에 대하여 설명하겠습니다. 컴퓨터 부팅 시에 DEL 키를 누르면 (PC 에 따라서 다른 키를 사용하는 경우도 있음) BIOS SETUP 화면이 표시됩니다. BIOS 의 종류에 따라서 다른 화면이 표시됩니다.

다음 화면그림을 참조하여, 마더보드상에 있는 RS232 통신포트가 Disable 상태로 되어 있지 않도록 설정하시기 바랍니다.



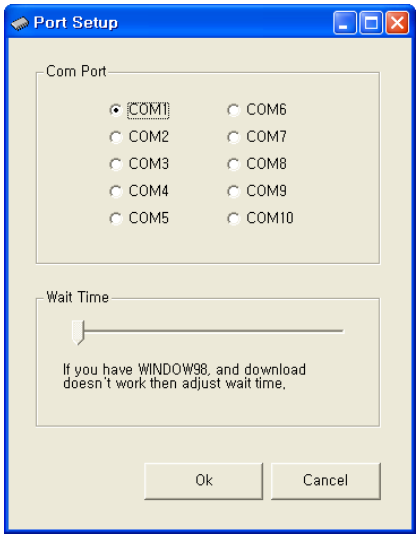


다음은 윈도우가 COM 포트 드라이버를 제대로 인식하고 있는지 체크해보아야 합니다. <내 컴퓨터 – 시스템 정보 표시 – 하드웨어 – 장치관리자>까지 선택하여 들어가 보면, 아래 화면이 표시됩니다. (윈도우 XP 의 경우)



이 다이얼 로그 박스에서 “통신포트 (COM1)”이 표시되어 있는지 확인하시기 바랍니다. 만약 통신포트가 나와있지 않다면 <동작>메뉴에서 <하드웨어변경사항 검색>을 눌러, COM1 포트를 추가하시기 바랍니다.

끝으로 CUBLOC STUDIO 의 “SETUP”메뉴에서 “PC 인터페이스 설정”에서 COM1 포트로 셋팅해주면 CUBLOC 통신포트 설정이 완료된 것입니다.

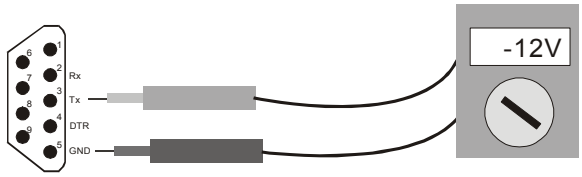


아래 부분의 Wait Time 은 Window 98 유저를 위한 셋팅입니다. XP, 2000 유저는 조정할 필요가 없습니다.

Window 98 은 컴퓨터의 속도에 따라서, 다운로드 시 에러가 발생되는 경우가 있습니다. Wait Time 속도를 조정해 가면서, 한번씩 다운로드를 시도해보십시오.

그러면, 동작되는 레벨이 있을 것입니다. 그곳으로 한 번만 셋팅되어 있으면, 이후로는 조정할 필요가 없습니다.

모든 설정이 다 끝났다면, RS232 다운로드 케이블의 3 번 단자에서 –12V 정도가 나오고 있는지를 확인해 보시기 바랍니다.간혹 케이블이 잘못되었거나, PC 내부 마더보드의 컨넥터가 연결되어 있지 않는 경우도 발생하기 때문입니다.



RS232 포트 셋팅은 최초 한번만 제대로 설치한다면, 이후부터는 신경 쓸 필요가 없게 됩니다.

RS232 포트가 없는 PC 또는 노트북에서 사용방법

노트북이라면 PCMCIA-TO-RS232 카드를 권장합니다. USB-TO-RS232 보다 안정적이고 빠른 다운로드 속도를 지원합니다. (강원전자 NETMATE C-191)

최근에는 PC 에도 RS232 포트가 없는 경우가 있습니다. 하지만 마더보드에는 최소한 COM 포트 1 개정도는 구비하고 있는 것이 보통입니다. 다만 외부로 연결되어 있지 않은 경우입니다. 해당 마더보드의 매뉴얼을 잘 참조하시어, 외부로 RS232 포트를 뽑아내어 사용하시는 것이 가장 좋습니다. 아래와 같은 PCI 슬롯방식 RS232 확장카드를 사용하는 방법도 있습니다.

*부득이한 경우가 아니라면 USB-TO-RS232 사용을 권장하지 않습니다. USB-TO-RS232 케이블은 내부에 있은 마이컴에서 버퍼링을 한뒤 재출력하는 방식이라 속도가 느리고, 동작이 불안정합니다.

*주의사항 : RS232 케이블의 길이는 최대 2 미터까지만 사용할 수 있습니다. 그 이상의 거리에서는 다운로드 및 디버깅 / 모니터링의 정확한 동작을 보증할 수 없습니다.

다운로드 시 발생되는 에러멧세지

1. RS232 통신에러입니다. 케이블상태나 전원상태, PC 의 통신 설정상태를 점검해보십시오.
케이블을 연결하지 않았거나, CUBLOC 코어모듈에 전원이 공급되지 않았을 경우, 또는 PC 상의 통신포트 설정에 이상이 있는 경우 발생하는 에러 멧세지입니다. **펌웨어가 제대로 다운로드 안되었을때도 이 멧세지가 나옵니다.** SETUP 메뉴에 있는 펌웨어 다운로드를 눌러서, 펌웨어를 다운로드한뒤 사용하시기 바랍니다.

2. Ladder 에서 END 를 찾을 수 없습니다.
소스파일 자체를 오픈하지 않았거나, LADDER 부에서 END 명령을 작성하지 않은 경우입니다. 이 경우는 통신문제와는 관계없는 경우입니다.

3. Ladder 컴파일 에러
LADDER 부에서 문법상의 오류 및 회로 결선상의 오류가 있는 경우입니다. 이 경우는 통신문제와는 관계없는 경우입니다.

4. 존재하지 않는 RS232 포트입니다.
PC 상의 COM 포트 셋업에 문제가 있는 경우이거나, 존재하지 않는 COM 포트를 선택한 경우입니다. 앞에서 설명한 BIOS 셋업상태와 윈도우 시스템에서의 COM 포트 설정상태를 점검해 보아야 합니다. 앞에서 설명한대로 설정했는데도 불구하고 똑 같은 에러가 발생한다면 PC 전문가의 도움을 받아야 합니다. 이 에러는 CUBLOC STUDIO 에서 COM 포트를 찾지 못하는 문제이므로, CUBLOC 하드웨어 상태와는 무관한 문제입니다.

5. CB220 모듈이 아닙니다. CONST DEVICE 문에서 선언한 모듈과 다른 모듈이 연결되어 있습니다.
CONST DEVICE 선언문을 사용해서 어떤 큐블록 코어모듈을 사용 중인지를 CUBLOC STUDIO 가 알 수 있도록 해주어야 합니다. <사용 예 : CONST DEVICE = CB280 >

6. 체크섬 불일치 발생: 플레쉬 메모리의 일부가 손상되었습니다.
큐블록 코어모듈 내부에 있는 플레쉬 메모리의 일부가 손상되거나, 다운로드 중 전원의 불안이나 기타요인 등으로 인해, 제대로 플레쉬에 기입되지 못한 경우입니다. 이 경우 다시 다운로드 하면 제대로 다운로드 될 수도 있습니다. 반복해서 같은 에러멧세지가 발생한다면 메뉴상의 “펌웨어 다운로드”를 실행한 뒤 다시 해보시고, 그래도 안 된다면, 큐블록 코어모듈을 다른 것으로 교체하여 사용하시기 바랍니다. 큐블록 내부에 있는 플레쉬 메모리는 최대 10 만번까지 다운로드 가능하며, 그 이상 사용하였을 경우, 특정 메모리 블록이 손상될 수 있습니다.

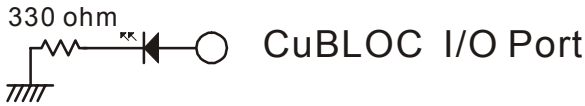
7. 억세스 코드가 잘못되었습니다.
큐블록 내부에 있는 “슈퍼바이저”칩이 손상된 경우입니다. 이 경우에도 펌웨어 다운로드를 다시 해보시고, 그래도 안 된다면 큐블록 코어모듈을 다른 것으로 교체해야 합니다.

입출력 회로 구성법

다음은 LED, 스위치, 볼륨 등 기초적인 소자들을 "큐블록"과 연결하는 회로입니다. CT172X/C 의 I/O 에도 이와 같은 방법으로 기초소자를 연결할 수 있습니다.

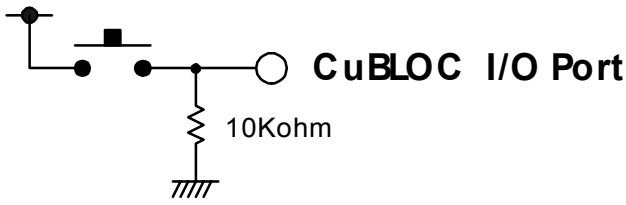
LED 연결회로

큐블록의 I/O 포트에 다음회로와 같이 LED 를 연결한 뒤, 해당 포트로 HIGH 를 출력하면 LED 가 켜집니다.



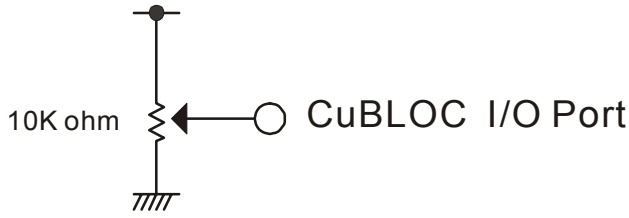
스위치 연결회로

큐블록의 I/O 포트에 다음회로와 같이 스위치를 연결한 뒤, 해당 포트로 INPUT 상태로 만들고, 스위치를 누르면 HIGH 가 입력되고, 스위치를 누르지 않으면 LOW 가 입력됩니다.



볼륨 연결회로

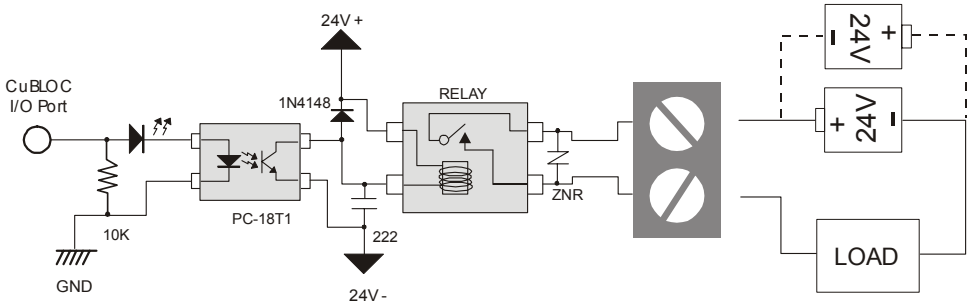
큐블록의 A/D 입력 가능한 I/O 포트에 다음회로와 같이 볼륨을 연결한 뒤, ADIN 명령으로 볼륨의 상태를 읽어올 수 있습니다.



큐블록 코어모듈은 기본적으로 5V 로 구동됩니다. 5V 를 초과하는 전원전압을 I/O 로 사용하고자 할 때에는 별도의 변환회로를 추가해주어야 합니다.

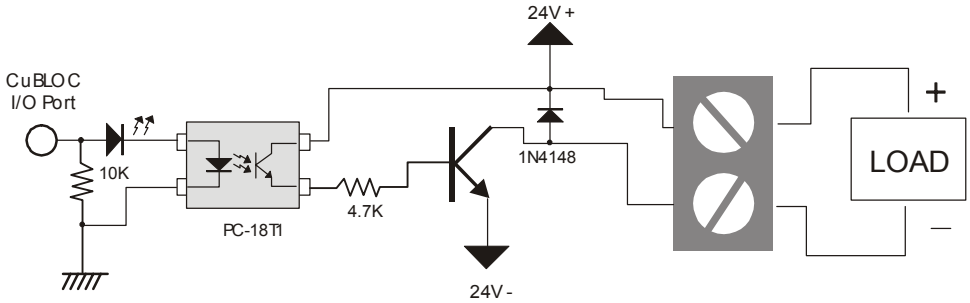
RELAY 출력 회로

큐블록 I/O 포트에 릴레이를 연결하는 기본회로입니다. 포토커플러를 사용하여 24V 측과 전원을 완전히 분리하였으므로, 24V 측의 노이즈 등이 5V 측으로 전달되지 않도록 구성된 회로입니다. (LOAD 란 부하를 의미합니다.) RELAY 출력회로는 단지 접점회로에 불과하므로, 외부에 별도의 전원을 구성해주어야 합니다.



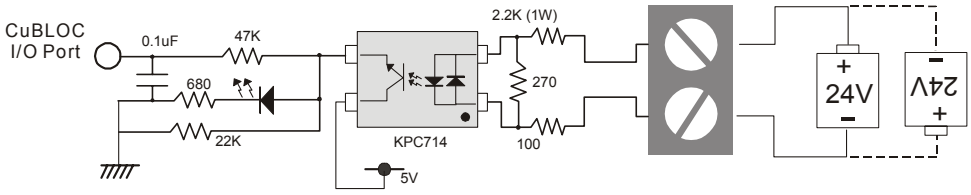
NPN TR 출력 회로

NPN TR 을 포토커플러를 사용해서 5V 측 전원과 분리시켜 연결한 회로입니다.



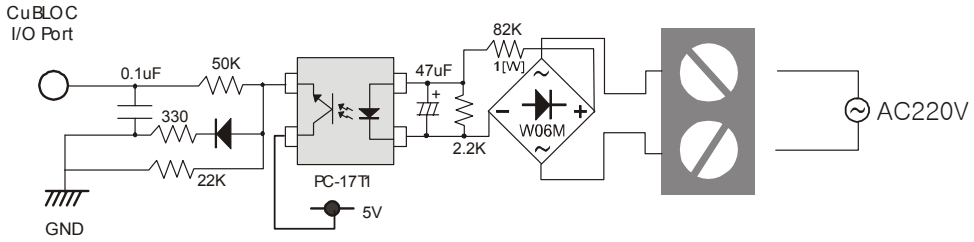
DC24V 입력회로

양극성 포트커플러를 사용해서 DC24V 의 입력신호가 들어오면 CUBLOC 에 HIGH 가 입력되는 회로입니다.



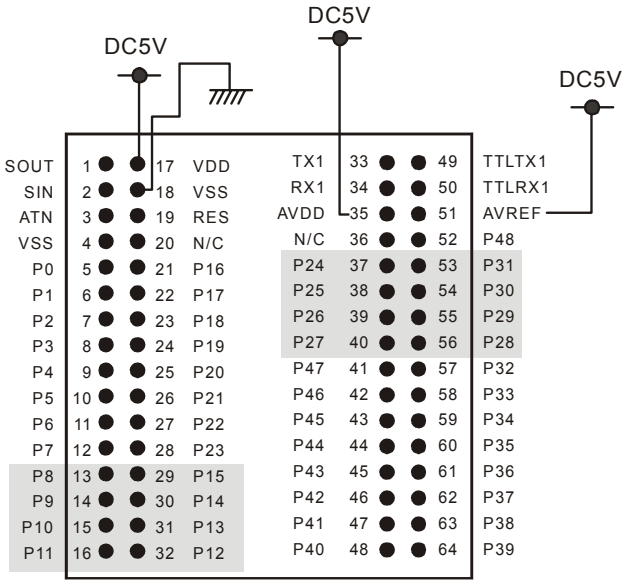
AC220V 입력 회로

AC220V 를 입력하면, CUBLOC 에 HIGH 가 입력되는 회로입니다.



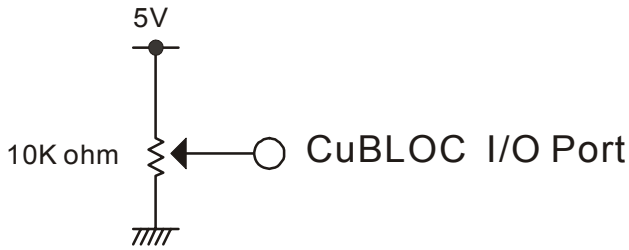
A/D 입력 회로

A/D 입력을 위해서 CB280 모델은 AVDD 와 AVREF 에 5V 를 연결해 주어야 합니다. AVDD 는 ADC 회로에 전원을 인가해주고, AVREF 는 AD 변환의 기준이 되는 전압을 알려주는 기능을 합니다. AVREF 가 5V 이면 0~5V 사이의 입력 값을 A/D 변환하고, AVREF 가 3V 이면 0~3V 사이의 입력 값을 A/D 변환합니다.

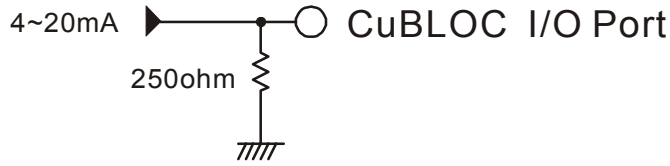


CB220, CB320 모델에서는 AVDD 와 AVREF 단자가 내부적으로 5V 에 연결되어 있고, 핀 아웃에 별도로 나와 있지 않습니다.

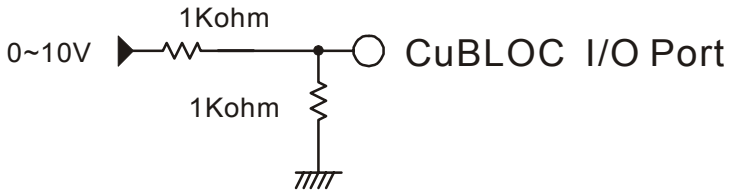
다음은 가장 간단한 형태의 A/D 변환회로로, 볼륨을 사용한 A/D 변환 회로입니다. 볼륨을 돌리면 0~1023 사이의 값으로 A/D 변환을 수행합니다. 0V 일 때, 0 을, 5V 일 때 1023 을 반환합니다.



다음은 4~20mA 입력을 받는 A/D 변환회로입니다. 250 오옴의 저항을 구할 수 없을 때에는 220ohm 과 30ohm 을 직렬로 연결하여 사용합니다. (정밀저항 사용요망)

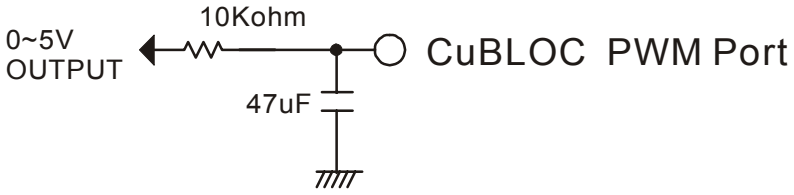


0~10V 입력을 받는 경우에는 다음과 같이 저항 2 개를 써서 분압하는 방식으로 A/D 변환을 수행합니다.



PWM 으로 D/A 변환 하는 회로

CUBLOC 에는 PWM 포트가 6 개 있습니다. 이 PWM 포트에 간단한 회로를 부착하면 D/A 변환기로 사용할 수 있습니다.



부 록

부록 1: ASCII 코드표

코드	문자	코드	문자	코드	문자	코드	문자
00H	NUL	20H	SPACE	40H	@	60H	`
01H	SOH	21H	!	41H	A	61H	a
02H	STX	22H	“	42H	B	62H	b
03H	ETX	23H	#	43H	C	63H	c
04H	EOT	24H	$	44H	D	64H	d
05H	ENQ	25H	%	45H	E	65H	e
06H	ACK	26H	&	46H	F	66H	f
07H	BEL	27H	‘	47H	G	67H	g
08H	BS	28H	(	48H	H	68H	h
09H	HT	29H	)	49H	I	69H	I
0AH	LF	2AH	*	4AH	J	6AH	j
0BH	VT	2BH	+	4BH	K	6BH	k
0CH	FF	2CH	,	4CH	L	6CH	l
0DH	CR	2DH	–	4DH	M	6DH	m
0EH	SO	2EH	.	4EH	N	6EH	n
0FH	SI	2FH	/	4FH	O	6FH	o
10H	DLE	30H	0	50H	P	70H	p
11H	DC1	31H	1	51H	Q	71H	q
12H	DC2	32H	2	52H	R	72H	r
13H	DC3	33H	3	53H	S	73H	s
14H	DC4	34H	4	54H	T	74H	t
15H	NAK	35H	5	55H	U	75H	u
16H	SYN	36H	6	56H	V	76H	v
17H	ETB	37H	7	57H	W	77H	w
18H	CAN	38H	8	58H	X	78H	x
19H	EM	39H	9	59H	Y	79H	y
1AH	SUB	3AH	:	5AH	Z	7AH	z
1BH	ESC	3BH	;	5BH	[	7BH	{
1CH	FS	3CH	<	5CH	₩	7CH	|
1DH	GS	3DH	=	5DH	]	7DH	}
1EH	RS	3EH	>	5EH	^	7EH	~
1FH	US	3FH	?	5FH	_	7FH	DEL

릴레이 표현

CT172X/C 에서의 릴레이

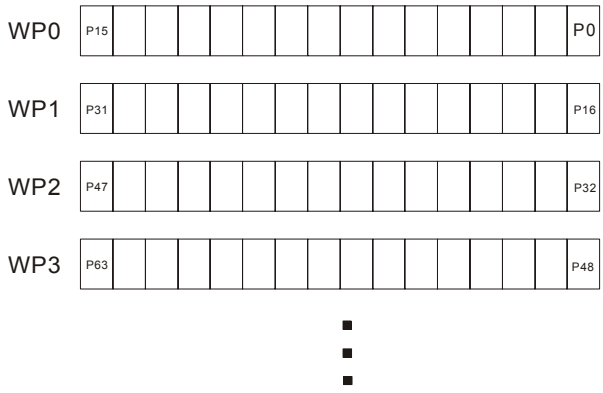
CT172X/C 에서 사용할 수 있는 릴레이를 정리한 표입니다.

릴레이 명칭	범위	단위	기능
입출력 릴레이 P	P0~P127	1 비트	외부 기기와의 인터페이스
내부릴레이 M	M0~M2047	1 비트	내부 상태의 보존
특수기능 릴레이 F	F0~F127	1 비트	시스템 상태
타이머 T	T0~T255	16 비트 (1 워드)	타이머용
카운터 C	C0~C255	16 비트 (1 워드)	카운터용
스텝제어 S	S0~S15	256 스탭(1 바이트)	스텝제어용
데이터 영역 D	D0~D511	16 비트 (1 워드)	데이터보관

P,M,F 는 비트단위로 되어 있고, T,C,D 는 워드단위로 되어 있습니다. 비트 영역으로 되어 있는 P, M, F 를 워드단위로 억세스 하려면 WP, WM, WF 을 사용해야 합니다.

릴레이 명칭	범위	단위	기능
WP	WP0~7	16 비트 (1 워드)	P 영역의 워드단위 억세스
WM	WM0~WM63	16 비트 (1 워드)	M 영역의 워드단위 억세스
WF	WF0~WF7	16 비트 (1 워드)	F 영역의 워드단위 억세스

WP0 는 P0~P15 까지의 내용을 담고 있으며, P0 이 가장 아래쪽(LSB)에 P15 가 가장 위쪽(MSB)에 위치합니다. WMOV 등과 같은 응용명령어 군에서 사용하면 편리합니다.



특수릴레이

특수릴레이를 통하여 CUBLOC 의 상태나 타이밍 정보, 시스템정보 등을 알 수 있습니다. (빈칸은 사용하지 않는 특수릴레이입니다.)

특수릴레이	설명
F0	상시 OFF
F1	상시 ON
F2	**최초 LADDER 실행 시 1 SCAN 만 On**
F3	
F4	
F5	
F6	
F7	
F8	10mS 마다 1 SCAN 만 On
F9	100mS 마다 1SCAN 만 On
F10	
F11	
F12	
F13	
F14	
F15	
F16	1 스캔타임간격으로 ON/OFF 를 반복
F17	2 스캔타임간격으로 ON/OFF 를 반복
F18	4 스캔타임간격으로 ON/OFF 를 반복
F19	8 스캔타임간격으로 ON/OFF 를 반복
F20	16 스캔타임간격으로 ON/OFF 를 반복
F21	32 스캔타임간격으로 ON/OFF 를 반복
F22	64 스캔타임간격으로 ON/OFF 를 반복
F23	128 스캔타임간격으로 ON/OFF 를 반복
F24	10mS 마다 ON/OFF 를 반복
F25	20mS 마다 ON/OFF 를 반복
F26	40mS 마다 ON/OFF 를 반복
F27	80mS 마다 ON/OFF 를 반복
F28	160mS 마다 ON/OFF 를 반복
F29	320mS 마다 ON/OFF 를 반복
F30	640mS 마다 ON/OFF 를 반복
F31	1.28 초마다 ON/OFF 를 반복

F32	5.12 초마다 ON/OFF 를 반복
F33	10.24 초마다 ON/OFF 를 반복
F34	20.48 초마다 ON/OFF 를 반복
F35	40.96 초마다 ON/OFF 를 반복
F36	81.92 초마다 ON/OFF 를 반복
F37	163.84 초마다 ON/OFF 를 반복
F38	327.68 초마다 ON/OFF 를 반복
F39	655.36 초마다 ON/OFF 를 반복
F40	**BASIC 으로 LADDERINT 발생**
F41	
F42	
F43	
F44	
F45	
F46	
F47	

* F40 에 1 을 써넣으면 BASIC 으로 인터럽트를 발생시킬 수 있습니다.

* F2 는 최초실행 시 (SET LADDER ON 실행 시점)에서 한 스캔만 ON 됩니다. 초기화 관련 처리를 할 때 사용합니다.

*공백으로 되어 있는 특수릴레이는 사용하지 않는 특수릴레이입니다.

#

?

A

B

C

D

E

F

G

H

I

K

L

M

V

W