베이직 언어와 레더로직을 사용하는 산업용 임베디드 콘트롤러

CUBLOC

BASIC 언어중심

사용설명서

사용설명서 Version 2.2

COMFILE
TECHNOLOGY

컴파일 테크놀로지 주식회사

www.comfile.co.kr

머릿말

본 제품은 컴파일 테크놀로지의 자체 기술력으로 OS 부터 컴파일러까지 개발한 제품입니다. 2005 년 출시이후 지금까지도 많은 분들의 사랑을 받고 있는 토종 산업용 임베디드 콘트롤러 제품입니다. 초보자들도 쉽게 배울 수 있는 BASIC 언어와 자동화 현장에서 널리 사용되고 있는 레더로직을 동시에 사용할 수 있는 컨트롤러로써, 전 세계에서 비슷한 제품조차 찾아볼 수 없는 독보적인 형태의 제품입니다.

MCU 보다는 쉽게, 그리고 PLC 로 할수 없는 복잡한 제어도 가능한 큐블록을 활용해서 여러분의 프로젝트도 쉽고 빠르게 완성하시기 바랍니다.

컴파일 테크놀로지 주식회사

등록상표

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

명칭

CUBLOC 은 모든 CUBLOC 제품군을 통칭하는 이름입니다.

알림

본 설명서의 내용은 사전 통보 없이 변경될 수 있습니다. 본 제품의 기능은 성능 개선을 위하여 사전 통보 없이 변경될 수 있습니다. 본 제품을 이용하여 만든 제품의 동작여부에 대해, 그리고 2 차적인 피해에 대하여 폐사에서는 어떠한 법적 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 버전이지만, 인쇄된 후 시간이 경과된 뒤에 새로운 내용이 추가되거나, 기존내용이 바뀔 가능성이 있습니다. 최신 버전의 설명서는 항상 인터넷 홈페이지 (www.comfile.co.kr)에서 확인하시기 바랍니다. **본 제품 (CUBLOC 및 CUTOUCH, MSB 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.**

Copyright 컴파일 테크놀로지㈜ Since 1996

CUBLOC 은 인터프리터 방식입니다.

CUBLOC 은 인터프리터 방식이므로 실행속도가 MCU 만큼 빠르지 않습니다. 간혹 MCU 를 경험해보신 분들이 인터프리터 방식의 차이점을 이해하지 못하고, 큐블록을 사용하시다가 문제가 생기는 경우가 있습니다. 큐블록은 다음과 같은점을 주의해서 코딩해야 합니다.

1. 인터럽트 루틴을 간단하게 코딩하세요. MCU 만큼 실행속도가 빠르지 않으므로, 인터럽트 루틴에서 너무 많은 시간을 보내고 있다면, 그로인해 메인 프로그램이 실행되지 않을 수 있습니다.

2. RS232 수신 분석하는 일은 하지 말아주십시오. RS232 데이터가 들어올때마다 그 내용을 파악하는데, 시간을 대부분 소비해서 다른 부분이 원활하게 돌아가지 않게됩니다. CUBLOC 에 기본적으로 내장된 모드버스를 활용하시기 바랍니다.

참고로, PC 용 언어도 크게 두종류로 나눌 수 있습니다.

- 네이티브 언어 : C++, 직접 기계어로 컴파일되는 방식
- Managed 언어 : C#, JAVA, 인터프리터가 중간에서 실행을 관리해주는 방식

C#과 마찬가지로 CUBLOC 도 인터프리터 방식입니다

- 컴파일 방식 : MCU, 모아콘, Arduino
- 인터프리터 방식 : 큐블록

큐블록은 레더로직과 BASIC 언어를 효율적으로 관리하기 위해 인터프리터 방식으로 설계되었습니다.

빠른 인터럽트 처리, RS232C 수신분석을 필요로 하는 어플리케이션이라면 MCU 처럼 컴파일 방식을 사용하는 MOACON 또는 FA-DUINO 를 선택해 주세요.

차 례

제 1 장 CUBLOC 의 기초 **9**

BASIC 과 레더로직을 동시에 지원 10
개발환경 10
CUBLOC STUDIO 11
다양한 CUBLOC 제품군 12

제 2 장 CUBLOC STUDIO **13**

CUBLOC STUDIO 사용법 기초 14
DEMO : 소스입력과 저장/실행 16
메뉴 설명 19
다운로드 준비 21
펌웨어 다운로드 23
디바이스 선택 24
에디터의 환경설정 25

제 3 장 CUBLOC BASIC **26**

1. CUBLOC BASIC 의 기본구성 27
3. 지역변수와 전역변수 31
4. 변수형 32
5. 문자열 34
6. 배열 36
7. 상수 38
8. 연산자 40
9. 비트, 바이트 지정자 47
10. 형식변환자 49
11. 문자열 처리함수 52
12. 전처리기 59
13. 조건 컴파일 61
14. 사용 디바이스 선언 63

제 4 장 CUBLOC BASIC 흐름제어 명령문 **64**

IF..THEN..ELSEIF…ELSE..ENDIF 65
SELECT..CASE 67
DO..LOOP 68
FOR..NEXT 69
GOTO 71
GOSUB..RETURN 71

제 5 장 CUBLOC BASIC 라이브러리 **72**

라이브러리의 종류 73
디지털 입출력 74
INPUT 74
OUTPUT 75
IN( ) 76
OUT 77
HIGH 78
LOW 79
BYTEOUT 80
BYTEIN( ) 81
OUTSTAT( ) 81
REVERSE 82
아날로그 입출력 83
ADIN( ) 83
PWM 87
PWMOFF 88
EEPROM 억세스 89
EEREAD( ) 89
EEWRITE 90
카운터 91
COUNT( ) 92
COUNTRESET 92
컴페어 94
COMPARE 94
인터럽트 96
ON TIMER 97
ON INT 99
인터럽트 금지/재동작 100
SET ONGLOBAL 100
SET INT 101
SET ONTIMER 101
SET ONRECV 101
SET ONINT 102
SET ONPAD 102
SET ONLADDERINT 102
LADDER 관련명령 104
SET LADDER ON/OFF 104
ALIAS 105
ALIASON ALIASOFF 105
USEPIN 106
FREEPIN 106
ON LADDERINT 107

디버깅 109
DEBUG 109
SET DEBUG 111
DEBUG 명령 활용법 112
RND( ) 114
RESET 114
RAMCLEAR 115
TIME( ) 116
TIMESET 117
RTCREAD( ) 119
RTCWRITE 121
FREQOUT 123
펄스 출력 명령 126
STEPPULSE 126
STEPSTOP 127
STEPSTAT( ) 127
STEPACCEL 128
WAIT 131
제 6 장 CUBLOC BASIC 통신관련기능 132
RS232 통신 133
OPENCOM 137
SET RS232 138
SET RS485 139
PUT 141
PUTSTR 142
PUTA 143
PUTA2 143
GET( ) 144
GETSTR( ) 146
GETSTR2( ) 147
GETA 148
GETA2 148
CHECKBF( ) 150
SYS( ) 151
BLEN( ) 152
BCLR 153
BFREE( ) 154
WAITTX 155
ON RECVX 156
SET UNTIL 157
MODBUS 통신 161
SET MODBUS 161
펑션코드 01 : READ COIL STATUS 166

펑션코드 02 : READ INPUT STATUS 166
펑션코드 03 : READ HOLDING REGISTERS 167
펑션코드 04 : READ INPUT REGISTERS 167
펑션코드 05 : FORCE SINGLE COIL 168
펑션코드 06 : PRESET SINGLE REGISTERS 169
펑션코드 15 : FORCE MULTIPLE COILS 170
펑션코드 16 : PRESET MULTIPLE REGS 171
에러처리 172
MODBUS 관련명령 173
GETCRC 173
MODBUS 마스터 모드 구현 (RTU 모드) 174

제 7 장 CUBLOC 디스플레이 라이브러리 176

영문 LCD 제품군 – CLCD 177
한글 LCD 제품군 – HLCD 177
SET DISPLAY 178
CLS 180
CSRON 180
CSROFF 180
CLCDOUT 181
LOCATE 181
PRINT 182
CLCD 모듈 183
CSG 시리즈 187
CSGNPUT 189
CSGXPUT 189
CSGDEC 190
CSGHEX 190

제 8 장 CUBLOC 부 프로그램 라이브러리 191

UDELAY 192
DELAY 192
KEYIN 193
KEYINH 193
BEEP 194
TADIN 194
BIN2BCD 195
BCD2BIN 195

제 9 장 BASIC 과 LADDER 의 LINK 196

데이터 공유 197
D 영역에 저장된 실수데이터를 BASIC 에서 사용하는 방법 198

D 영역에 저장된 32 비트 정수데이터를 BASIC 에서 사용하는 방법 *199*
LADDER 만 사용 200
BASIC 만 사용 200

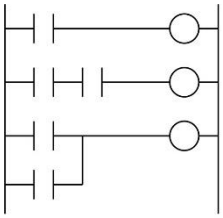
부 록 201

부록: ASCII 코드표 202
다운로드에러시 문제 해결방법 203
A/S 를 보내주시기전, 한번더 점검해주세요. 213

제 1 장
CUBLOC 의 기초

BASIC 과 레더로직을 동시에 지원

CUBLOC 은 BASIC 언어와 LADDER LOGIC 을 동시에 사용할 수 있습니다. LADDER LOGIC 은 입력 신호에 대한 즉각적인 반응을 할 수 있다는 장점이 있고, BASIC 언어는 복잡한 수식과 유저 인터페이스 관련 코딩을 할 수 있다는 장점이 있습니다. 이 두 방식의 장점을 하나의 콘트롤러에서 사용할 수 있도록 만든 제품이 CUBLOC 입니다.



LADDER LOGIC

```
DIM A AS INTEGER
IF IN(0) = 0 THEN
  OUT 2,A
END IF
GOTO LAB1
```

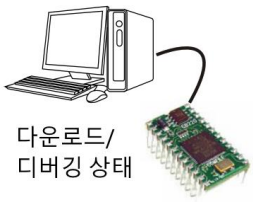
BASIC

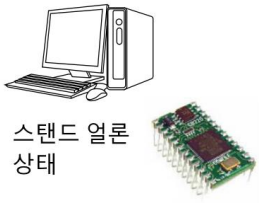
또한, CUBLOC 은 BASIC 과 LADDER LOGIC 이 동시에 실행되는 멀티테스킹 구조로 되어 있어, BASIC 수행중 백그라운드에서 LADDER LOGIC 이 동시에 실행됩니다. BASIC 에서 레더의 데이터 영역을 참조할 수 있어서, 실행 중 발생하는 정보를 교환할 수 있습니다.

개발환경

CUBLOC 을 사용하기 위해서는 Windows 7 / 10 / 11 운영체계와 RS232 포트를 갖춘 PC 가 필요하고, PC 와 RS232C 케이블을 사용해서 연결합니다. PC 에 RS232C 포트가 없는 경우에는 별도의 USB-TO-RS232C 케이블을 추가로 연결하신뒤 사용하시기 바랍니다.

PC 와 연결된 상태에서 다운로드 및 디버깅이 가능합니다. PC 와 연결을 해제하면 혼자서 동작가능한 스탠드얼론 상태가 됩니다.

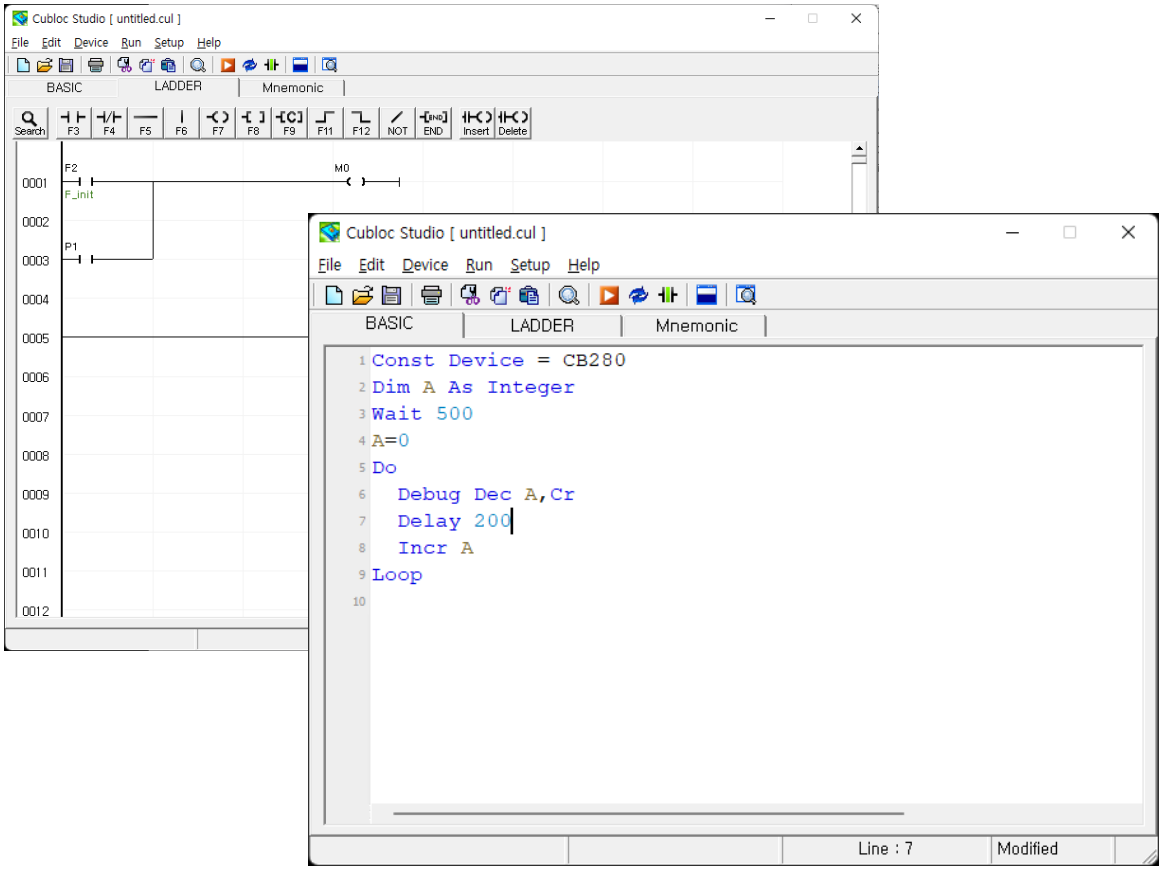




CUBLOC 에는 플레쉬 메모리가 내장되어 있어, 한번 기억된 프로그램은 전원이 없어도 계속 유지됩니다. 또한 나중에 언제든지 새로운 프로그램을 다시 다운로드 할 수 있습니다.

CUBLOC STUDIO

CUBLOC 을 사용하는 데 필요한 통합개발환경 소프트웨어인 CUBLOC-STUDIO 은 인터넷 홈페이지 www.comfile.co.kr 을 통하여 제공됩니다.



CUBLOC STUDIO 실행 최소 PC 최소 사양

- 프로세서 : i3 이상급
- 메모리 : 제약없음
- HDD : 1GB 이상
- OS : 윈도우 7 이상
- 기타 사항 : RS232C 또는 USB 포트중 하나 필요

다양한 CUBLOC 제품군

코어모듈

코어 모듈은 PCB 에 장착할 수 있는 반도체형으로 되어 있습니다. 유저 여러분이 임의대로 I/O 에 추가 회로를 부가하여, 원하는 보드를 설계할 수 있으므로, 다양한 제품을 만드실 수 있습니다. 단 PCB 를 직접 제작할 수 있는 분들만 사용가능합니다. 만약 PCB 제작이 어렵다면 MSB 제품군을 사용하세요.

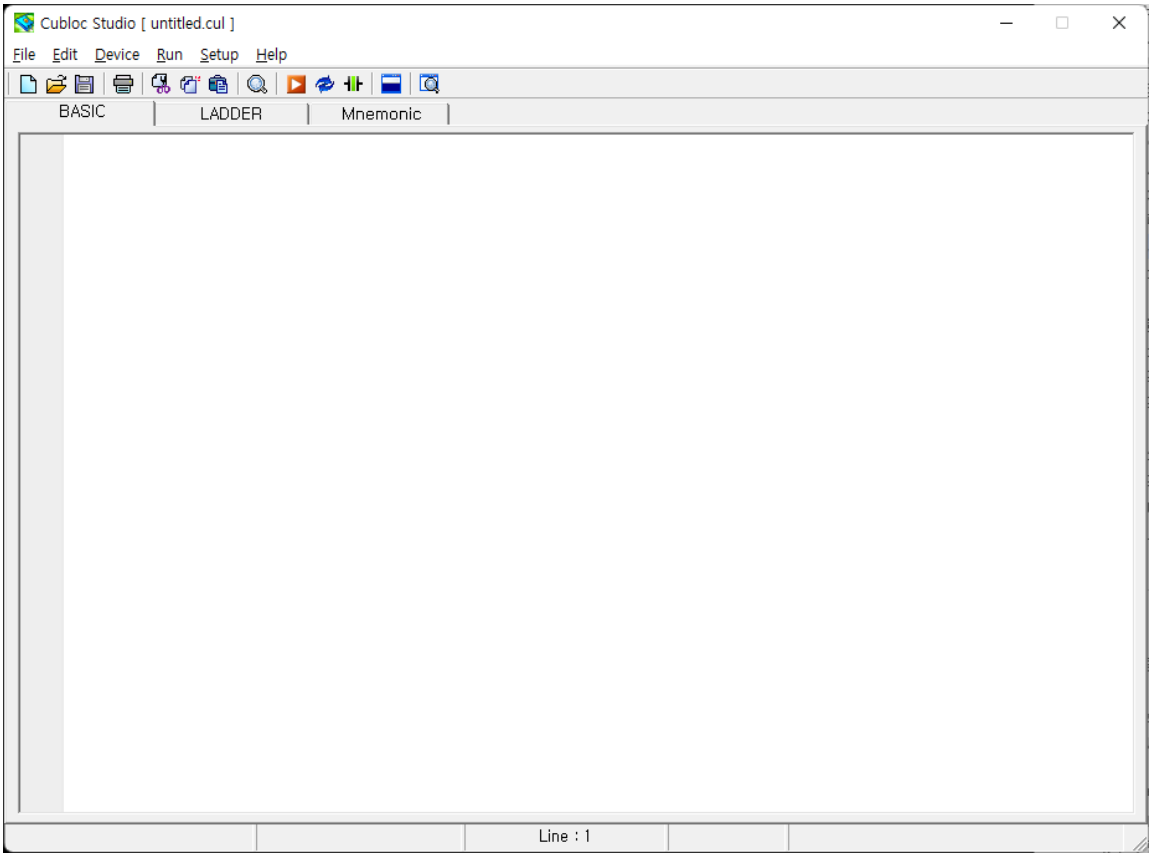
MSB 시리즈

베이스보드와 전원장치, CUBLOC 코어 모듈이 일체화되어 있는 제품입니다. 여러분은 단지 전원과 RS232C 케이블만 연결하면 곧바로 사용가능한 상태가 됩니다. 마치 PLC 처럼 현장에서 바로 용응할 수 있는 제품입니다

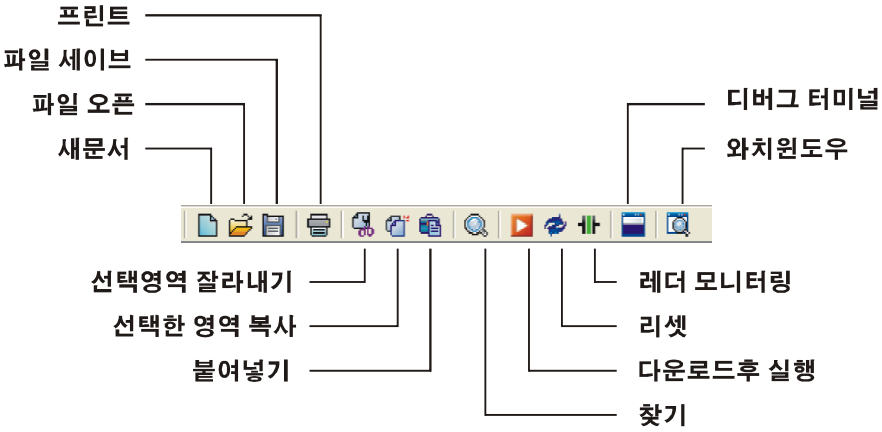
제 2 장
CUBLOC STUDIO

CUBLOC STUDIO 사용법 기초

인터넷 홈페이지 www.comfile.co.kr 자료실에서 다운로드 받아서 설치할 수 있습니다. CUBLOC STUDIO 을 설치한 후 실행하면 다음과 같은 빈 화면이 표시됩니다.



상단의 툴 바를 이용하여 기본적인 조작을 할 수 있습니다.



소스파일은 두개가 생성됩니다.

소스파일은 확장자 .CUL 과 .CUB, 두 개의 파일로 저장됩니다. 따라서 파일을 따로 백업하거나, 이동할 때 반드시 두 개 파일을 같이 옮겨야 합니다.

오브젝트 다운로드

컴파일/다운로드 후 파일명.OBJ 파일이 자동적으로 생성됩니다. 이 파일을 File 메뉴에 있는 "오브젝트 다운로드"에서 선택하면, CUBLOC 으로 다운로드됩니다. 만약 현장에 소스 전달이 곤란하시다면, 오브젝트파일만 전달해주셔 됩니다. 파일메뉴에 있는 오브젝트 다운로드를 사용하시면 오브젝트파일이 다운로드 됩니다.

강력한 프로텍션 기능 제공 (소스 업로드 불가함)

"CUBLOC"에 저장했다가 나중에 다시 불러오는 **"소스 업로드"기능은 지원하지 않으므로, 주의하시기 바랍니다.** 소스를 잃어버렸을 경우, CUBLOC 에서 소스를 다시 불러올 수 있는 방법은 없으므로 소스 프로그램의 보관 및 백업에 항상 신경 써 주시기 바랍니다.

원터치 다운로드

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.

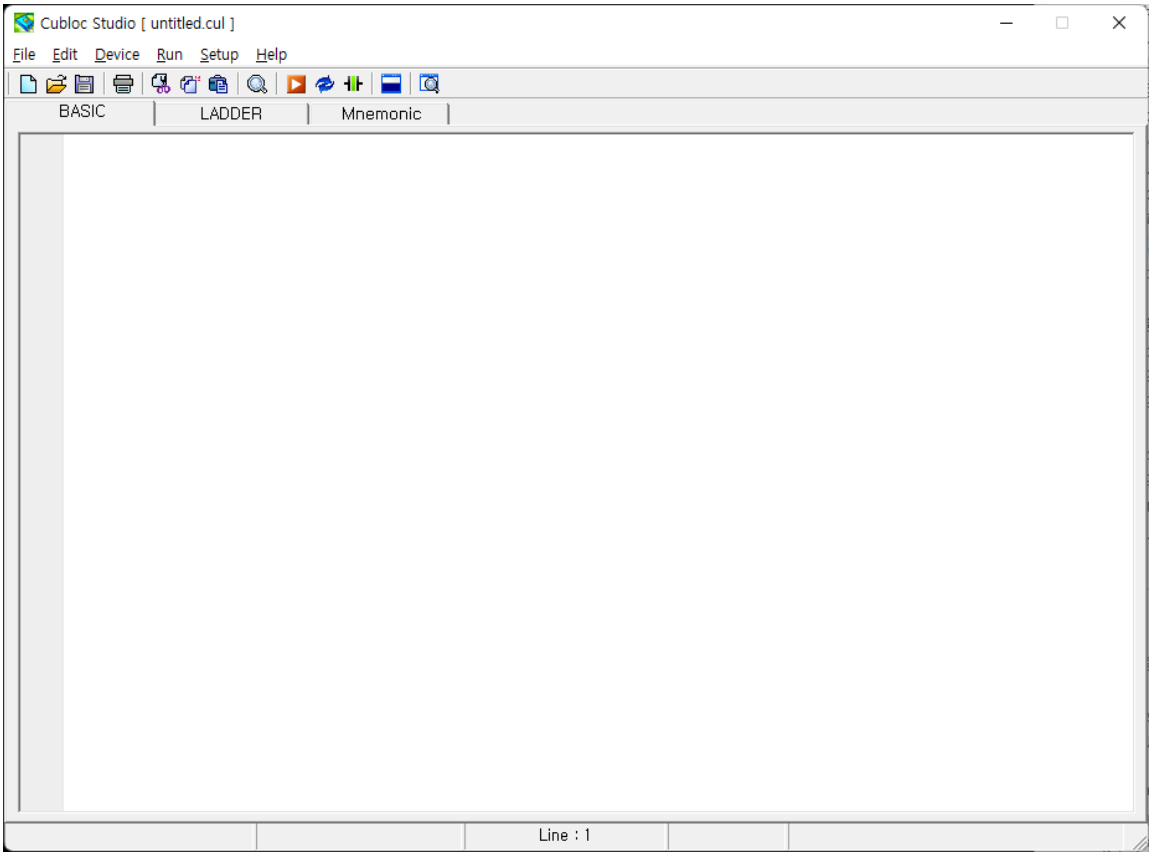
Editor 단축키

소스작성을 담당하고 있는 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 저장해 주세요.
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

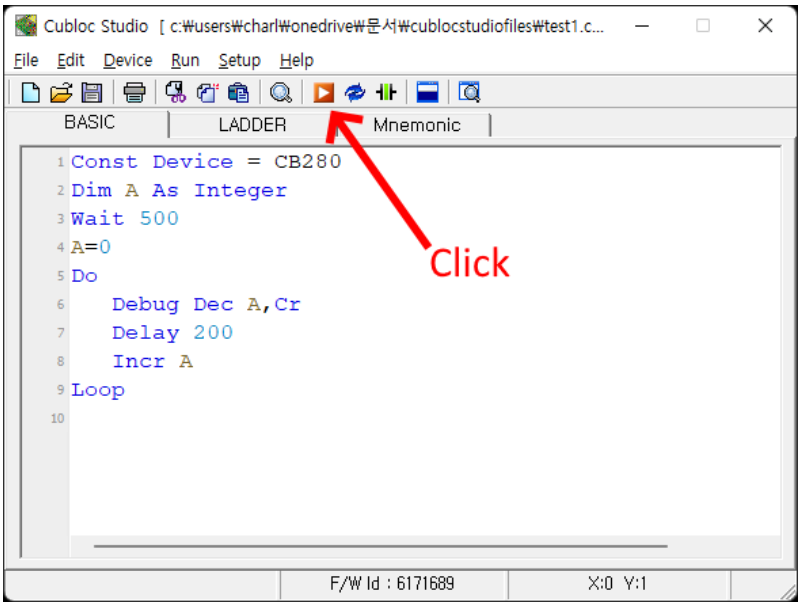
DEMO : 소스입력과 저장/실행

CUBLOC 스튜디오는 에디터와 컴파일러, 레더편집기, 디버거등을 내장하고 있는 소프트웨어입니다. CUBLOC STUDIO 을 실행시키면 다음과 같은 화면이 뜹니다. 만약 아래 화면처럼 깨끗하게 비어있는 상태가 아니라면, File 메뉴의 New 를 선택하십시오.



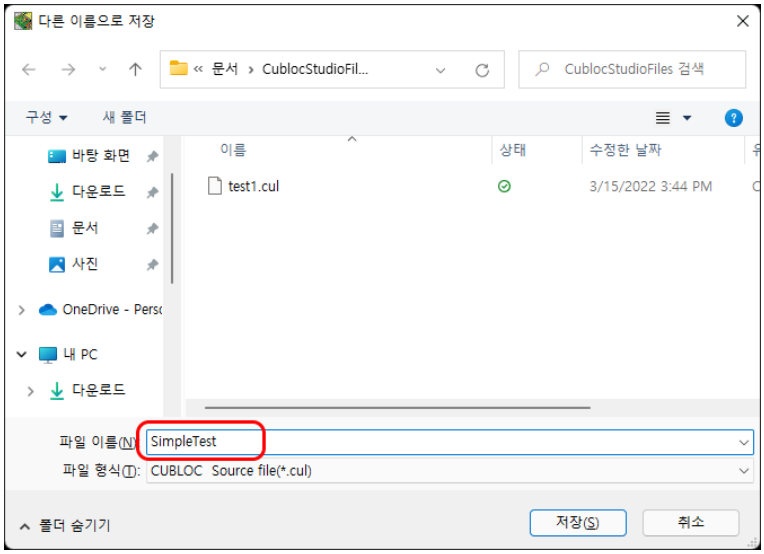
여기에 다음과 같은 소스 프로그램을 입력해 보겠습니다. (아래 소스를 Ctrl+C 해서 CUBLOC Studio 에 Paste 해도 됩니다.)

```
Const Device = CB280
Dim A as Integer
Wait 500
A=0
Do
  Debug Dec A,Cr
  Delay 200
  Incr A
Loop
```



Const, Dim 과 같은 BASIC 예약어는 파란색으로 표시됩니다. 제대로 입력하셨다면 Run 메뉴에서 Run 을 누르거나, 화면 상단에 있는 툴 박스에서 실행 아이콘을 클릭하거나, CTRL+R 키를 누르세요. 세가지 모두 "실행"하라는 명령입니다. 이중 편한 것을 선택하시면 됩니다.

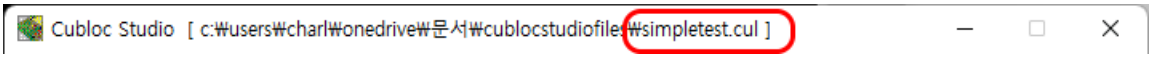
한번도 저장한 적이 없다면 다음과 같은 Save As 다이얼로그 박스가 표시됩니다.



File Name 에 저장하고 싶은 파일명을 타이핑하면 됩니다. 여기에서는 simpletest 라고 입력합니다.

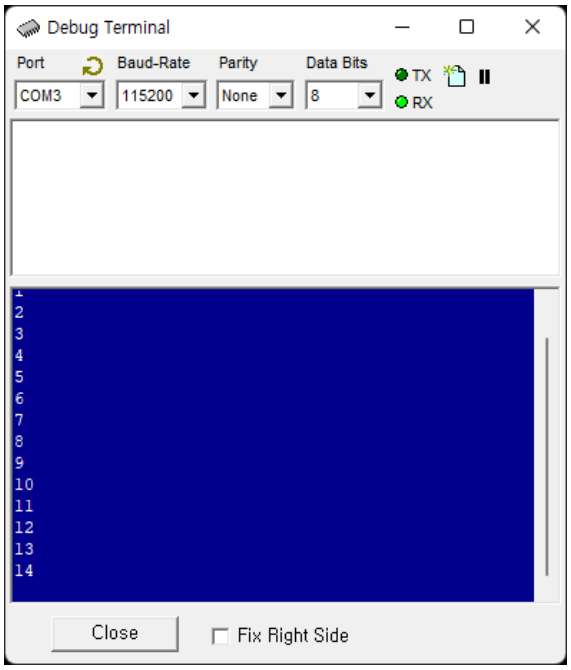
Simpletest 까지만 입력하고 Enter 키를 누르세요. 그러면 자동적으로 확장자가 붙어서 저장됩니다. 앞에서도 설명했듯이, CUBLOC STUDIO 의 파일은 .CUB 와 .CUL 두 개의 파일로 저장됩니다.

CUBLOC 스튜디오 상단에 simpletest.cul 이라는 이름으로 저장된 것을 확인할 수 있습니다. (최초에는 untitled.cul 로 되어 있다가 세이브 된 후에 simpletest.cul 로 바뀐 것입니다.)



한번 세이브 된 후로는 RUN 할 때마다 세이브하므로, 일부러 세이브를 할 필요는 없습니다. 하지만, 소스입력도중 CTRL+S 를 누르면 "저장"이 됩니다.

화면에 디버그 창이 나오고 증가되는 숫자가 표시됩니다.



CLOSE 보턴을 누르면, 디버그 창을 종료합니다.

*** 다운로드 에러가 발생했다구요? 그렇다면 본 사용설명서 부록에 있는 "다운로드시 문제해결방법"을 보시고 원인을 찾아 보시기 바랍니다.**

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.
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 으로 다운로드 합니다.
Utility for CUTOUCH	CUTOUCH 를 기능을 제공합니다. 이 부분에 대한설명은 별도로 되어 있는 CUTOUCH 사용설명서를 참조하세요.
최근 편집파일	최근에 편집한 파일 4 개를 보여줍니다. 이중 하나를 선택하면, 오픈됩니다.
끝내기	CUBLOC STUDIO 를 종료합니다.

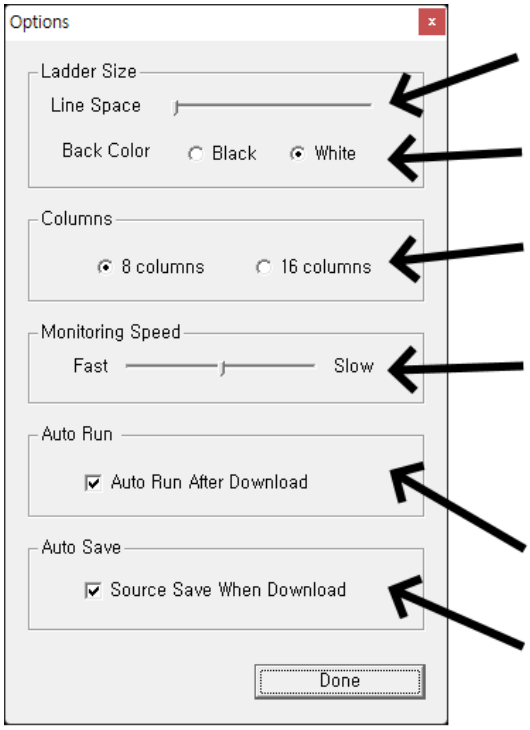
실행 (Run) 메뉴

메뉴	*설명*
다운로드후 실행	BASIC 과 LADDER 를 컴파일하고, 에러가 없으면 CUBLOC 으로 다운로드 후 실행합니다. *실행 시 "오브젝트"파일이 자동적으로 생성됩니다.
레더로직 Run	모니터링중에 사용할 수 있습니다. 레더동작을 개시합니다.
레더로직 Stop	모니터링중에 사용할 수 있습니다. 레더동작을 중지합니다.
리셋	CUBLOC 을 리셋 시킵니다.
LADDER 모니터링 시작	LADDER 모니터링을 시작합니다.
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.
펌웨어 다운로드후 실행	펌웨어(OS)를 다운로드한후 유저프로그램도 다운로드하고 실행시킵니다. 여러 개의 CUBLOC 에 프로그램을 주입할 때 사용하면 편리합니다.
릴레이 사용현황 보기	(컴파일 후) 레더에서 사용한 릴레이를 보여줍니다.
와치윈도우	레더의 릴레이 사용현황을 한눈에 볼수 있는 창을 띄워줍니다. 레더가 동작중이고, CUBLOC 과 연결된 상태에서만 사용가능합니다.
문법검사	CUBLOC 이 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 포트중 하나를 선택합니다.
에디터 환경 설정	BASIC 텍스트 에디터의 환경을 설정하는 메뉴입니다.
레더로직 환경설정	CUBLOC Studio 의 레더로직 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 펌웨어 (OS)를 다운로드 합니다. 이 메뉴를 사용하지 않아도, RUN 시에 STUDIO 에서 펌웨어 업그레이드 여부를 판단하여, 질문 창을 띄워줍니다.

레더로직 환경 설정 창에 대한 자세한 설명입니다.



레더의 사이즈를 조절하는 메뉴

레더편집 영역의 바탕색을 조절

레더 컬럼수 (기본값 8 컬럼)

레더 모니터링 속도를 조절하는 메뉴
Fast 에 가까울수록 갱신속도가 빨라져서, CUBLOC 의 I/O 상황이 바로 화면상에 표시됩니다. 하지만 통신량이 많아져서 PC 가 느려질수 있습니다.

다운로드후 자동으로 실행할지 결정

다운로드시 소스를 세이브할 지를 결정
(외부 TEXT 에디터를 쓸때에는 OFF 하세요.)

AUTO RUN 이란

CUBLOC 이 기계와 연결이 되어 있는 상황에서는 다운로드 후 자동으로 실행하도록 하면 예상치 못한 상황으로 인해,기계 동작에 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.

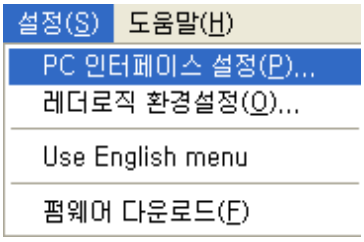
다운로드 준비

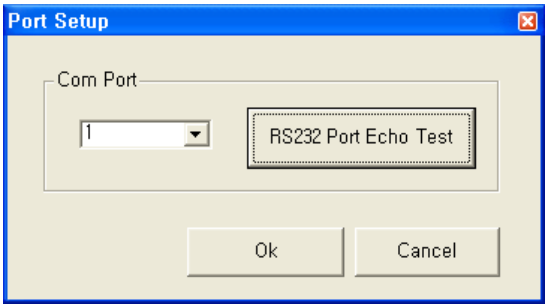
CUBLOC STUDIO 에서 작성한 소스프로그램은 PC 의 RS232C 포트를 통해 CUBLOC 에 다운로드 할 수 있습니다. 이를 위해 다운로드 케이블이 필요합니다. 한번 다운로드된 소스 프로그램은 전원이 꺼져도 계속 유지됩니다. 우선, 다운로드 케이블을 사용해서 PC 의 COM 포트와 CUBLOC 을 연결해 주십시오.

RS232C 케이블	3 핀 다운로드 케이블
 CUSB 및 , BASE 보드, CT1721C 용	 MSB 시리즈용

다운로드 케이블은 CUBLOC 관련 제품에 포함되어 있지 않습니다. 최초에 별도로 구매하셔야 합니다.

CUBLOC STUDIO 의 설정 메뉴의 <PC 인터페이스 설정>을 눌러서 COM 포트부터 맞춰주어야 합니다.





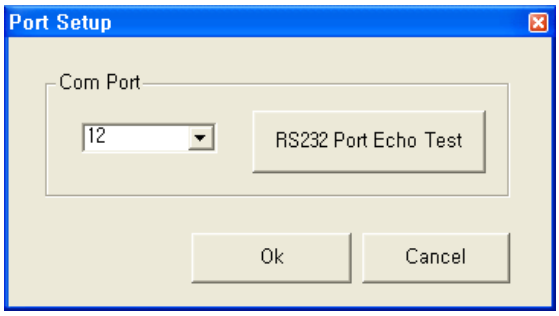
PC 에는 보통 1 개의 COM 포트가 있으며, 대부분 1 로 되어 있어 있습니다.

만약, COM 포트가 없는 PC 또는 노트북에서는 중간에 USB-TO-RS232C 케이블을 사용해야 합니다.

이때 주의할 점은 USB-TO-RS232C 케이블 드라이버부터 먼저 설치해야 한다는 것입니다.

윈도우 XP 의 경우는 별도로 드라이버를 설치해야 하지만, WINDOWS 7 이후버전은 USB 케이블 장착시 자동 설치됩니다.

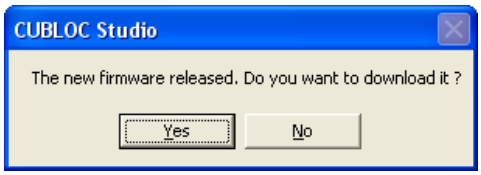
드라이버 설치후, CUBLOC STUDIO 의 설정 메뉴의 <PC 인터페이스 설정>을 보면, 1 번이 아닌 다른 번호가 보일 것입니다. (만약 안보인다면 드라이버 설치가 잘못된 것입니다.) 이 번호를 선택하십시오.



이 과정에 문제가 있다면, 본 매뉴얼 가장 뒷부분 부록에 있는 다운로드 에러 발생시 해결방법을 참조하셔서, PC 의 COM 포트 문제를 해결하신뒤 사용하시기 바랍니다.

펌웨어 다운로드

펌웨어란 CUBLOC 내부에 저장되는 "오퍼레이팅 시스템"을 말합니다. 즉, CUBLOC 을 운영하는 시스템 소프트웨어입니다. CUBLOC 을 최초 구입했을 때, 그리고 인터넷에서 새로운 버전의 CUBLOC 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 진행됩니다.

펌웨어 다운로드를 유저가 직접 하실 수도 있습니다. "Setup"메뉴의 "Firmware download"를 선택하시면 됩니다. CUBLOC 이 갑자기 다운로드가 안될 경우 펌웨어 다운로드를 하시면, CUBLOC 의 메모리를 전부 클리어 한 상태에서 펌웨어를 재기입하기 때문에, 이전 프로그램의 잘못으로 인한 "멀펑션"상태에서 벗어날 수 있습니다. (단 기존에 있던 유저 소스는 지워집니다. 주의하세요)

참고하세요!

DO..LOOP 안에 DEBUG 명령어 하나만 두었을 경우에 너무 빠른 RS232 통신이 발생되므로, 반드시 DELAY 명령과 함께 사용하시기 바랍니다. 너무 빠른 DEBUG 반복수행으로 인해 PC 측에서 이를 감당할 수 없는 상황이 생깁니다.

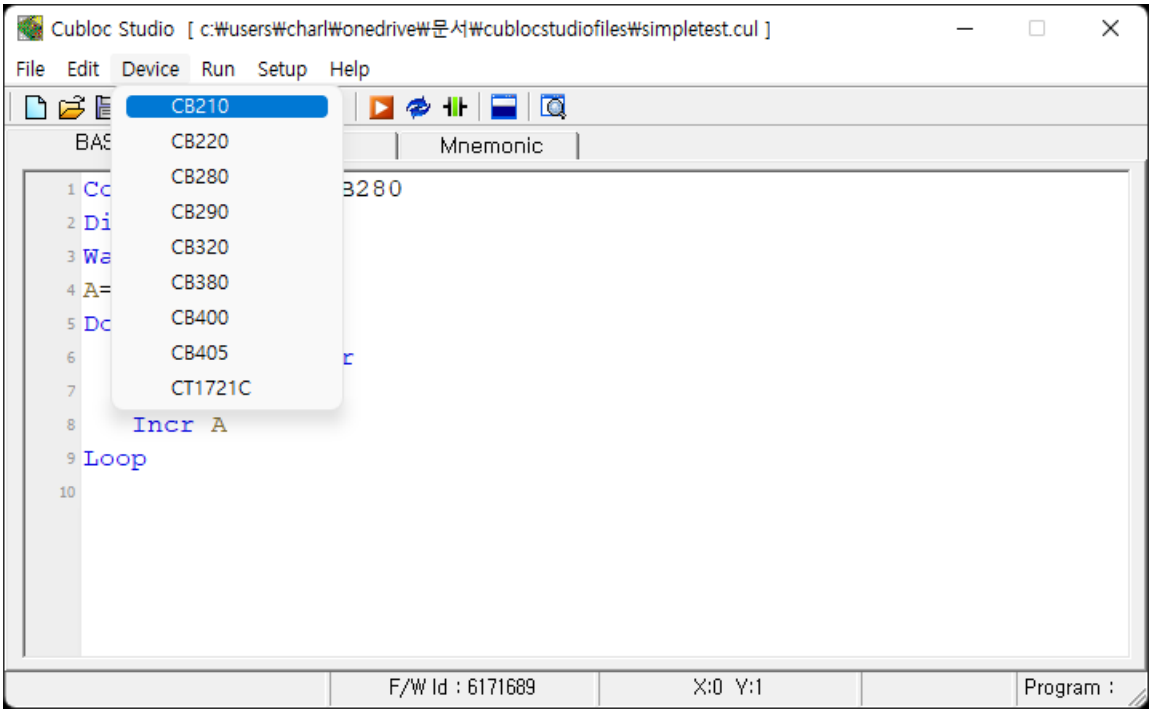
```
Do
   Debug Dec I,CR
   Delay 200   ' 0.2 초 대기
Loop
```

또한, CUBLOC 실행과 동시에 Debug 명령이 수행될 경우, PC 측 CUBLOC STUDIO 에 Debug 창이 미처 열리기도 전 상황이여서 맨처음 보낸 Debug 수행결과가 표시되지 않을 수 있으니, 맨 처음에 Debug 창이 표시할 수 있는 준비시간을 딜레이 해주어야 합니다.

```
Wait 500  ' 0.5 초 있다가 실행 개시
Do
   Debug Dec I,CR
   Delay 200
Loop
```

디바이스 선택

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



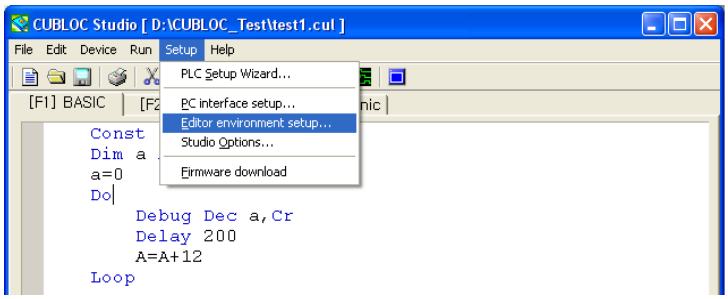
이미 Const Device 문이 소스상에 존재하고 있을 경우에는 그 문장을 찾아서, 뒷부분의 디바이스명만 고쳐줍니다.

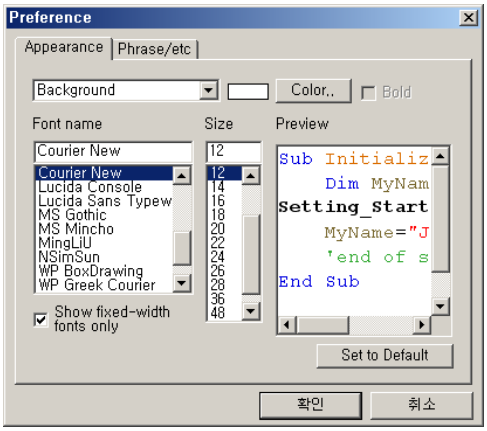
일부 제품은 Const Device 대신 #include 문을 사용합니다. 이 경우에는 직접 타이핑 해주어야 합니다.

```
#include "CB405RT"   ' CB405RT 는 Const Device = CB405RT 대신 include 문을 사용합니다.
```

에디터의 환경설정

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

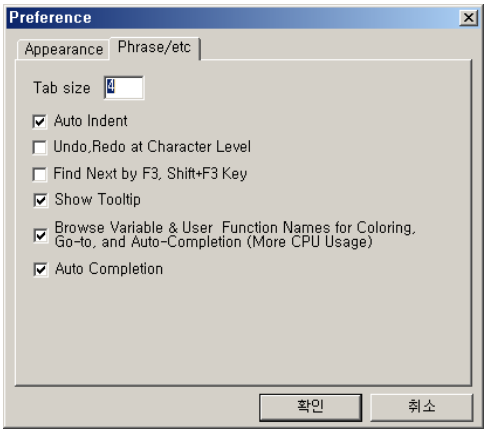




Appearance 탭

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 : 탭 기본사이즈를 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.

제 3 장
CUBLOC BASIC

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
```

다음은 CUBLOC 베이직에서 일반적으로 적용되는 규칙입니다.

1.1 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)
```

1.2 대소문자 구분

CUBLOC 에서는 대소문자를 구분하지 않습니다.

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

MOTOR 와 motor 는 같은 변수로 취급됩니다.

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)으로 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 ADDVALUE(B As Byte) As Byte
   ADDVALUE = B + 1     '부 프로그램명인 ADDVALUE 에 리턴 값을 저장합니다.
End Function
```

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

```
Function ADDVALUE(B As Byte)   ' 생략시 Long 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     ‘K를 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     ‘K를 지역변수로 선언
   For K=0 To DL
   Next
End Sub
```

인수가 여러 개 필요한 경우에는 콤머(,)를 사용해서 계속 사용할 수 있습니다.

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

2.4 부 프로그램 인수와 리턴 값의 사용상 제한

1. String 형은 인수 또는 리턴값으로 사용할 수 없습니다.
2. 배열 전체를 인수로 전달할 수 없습니다. 배열의 한 요소만 전달하는 것은 가능합니다.
3. 부 프로그램에 사용되는 인수는 모두 “값에 의한 참조” (Call by value)입니다. 즉 부 프로그램 내부에서 전달받은 인수의 값을 변경하여도, 본래 인수의 값에는 영향을 미치지 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.

```
Dim A As Integer
  :
  :
Gosub ABC
  :
ABC:
  :
End
```

```
Sub  DEF(B as Byte)
  :
  :
End Sub
```

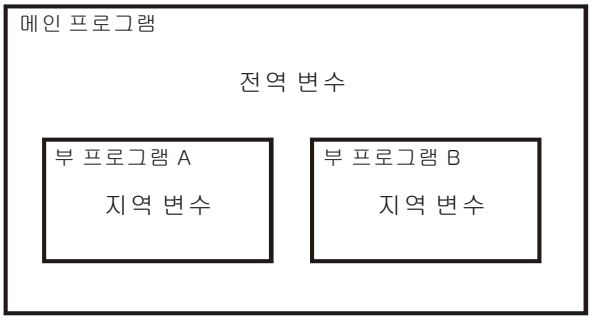
```
Function  GHI(C as Byte)
  :
  :
End Function
```

*** BASIC 에서는 부 프로그램과 메인 프로그램을 구분해 주기 위해 END 명령을 사용합니다.**

3. 지역변수와 전역변수

부 프로그램 안에서 변수를 선언한 경우에는 “지역변수”가 됩니다. 지역변수는 부 프로그램이 호출되면 생성되었다가, 호출이 끝나면 소멸되는 변수 입니다. 지역변수는 다른 부 프로그램에서 참조할 수 없습니다.

부 프로그램에서 필요한 변수는 “지역변수”를 선언한뒤 사용해야, 함수의 독립성이 보장됩니다. 반면 메인 프로그램에서 선언된 변수는 “전역변수”입니다. 전역변수는 메인 프로그램뿐만 아니라 프로그램 전체의 모든 부 프로그램에서 사용할 수 있습니다.




```
Dim  A  As  Integer   ‘ A 를 전역변수로 선언
LOOP1:
   A = A + 1
   Debug  Dp(A),CR   ‘ A 의 값을 10 진으로 화면 DEBUG 창에 표시
   DELAYTIME         ‘ 부 프로그램 DELAYTIME 을 호출
   Goto LOOP1
   End               ‘ 메인 프로그램의 끝을 지정, 여기까지가 메인 이라는 뜻

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

위 프로그램의 경우 A 가 전역변수, K 가 지역변수가 됩니다. A 는 전역변수이므로 전체프로그램에서 사용 가능하지만 K 는 DELAYTIME 부 프로그램 안에서만 사용 가능합니다.

지역변수에는 모든 변수형을 사용할 수 있지만, 배열은 사용할 수 없습니다. 배열은 반드시 전역변수로만 선언하여 주십시오.

*** 지역변수명과 전역변수명을 같은 이름으로 선언할 수 없습니다.**

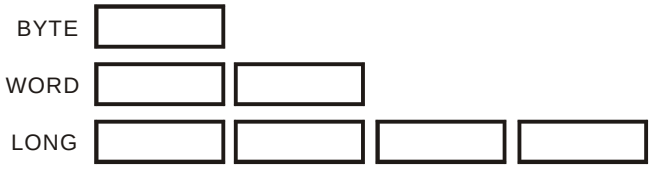
4. 변수형

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 -3.402823×10^{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)

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 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

CUBLOC 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.

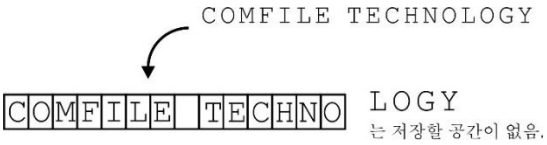
5. 문자열

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.
```

5.1 문자열 비교

문자열은 = 또는 <>연산자를 사용해서 비교할 수 있습니다. 이때 반드시 문자열변수 와 문자열변수 비교 또는 문자열 변수와 문자열 상수 비교만 사용하실 수 있습니다.

```
Dim Str1 As String * 12
Str1 = "Test"
If Str1 = "Test" Then
Debug "Same"
Endif
```

5.2 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 바이트 크기로 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"

6. 배열

CUBLOC BASIC 에서는 배열을 최대 2 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
```

배열의 인수는 0 부터 시작됩니다. 따라서 20 요소로 정의하면 0 부터 19 까지 사용할 수 있습니다.

문자열 배열은 지원하지 않습니다.

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

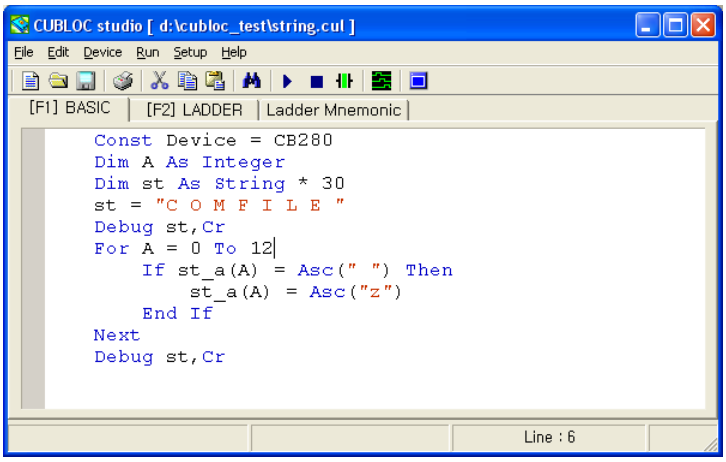
6.1 문자열 변수의 배열접근법

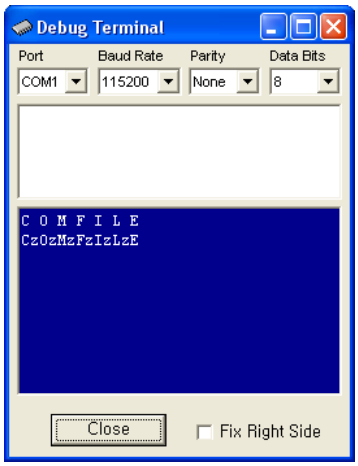
문자열 변수를 마치 1 차원 배열을 다루듯이 취급할 수 있다면 편리한 경우가 많이 있습니다. 예를 들어 문자열 변수 안에 있는 내용 중 단 한글자만 바꾼다던 지, 문자열 변수 안에 있는 내용 중 공백문자가 몇 개나 있는지 확인하고 싶을 때, 문자열 관련 함수로 프로그램을 짠다면 너무 번거롭게 됩니다.

CUBLOC 에서는 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.





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. 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)
```

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

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

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.

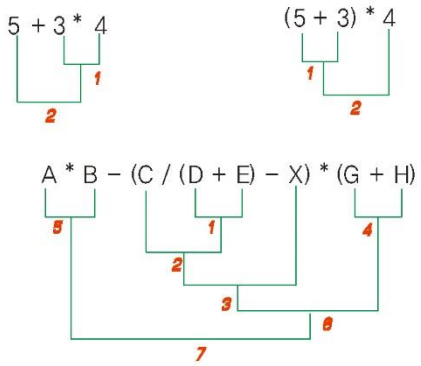
$$\frac{1}{2} \blacktriangleright 1/2 \qquad \frac{5}{3+4} \blacktriangleright 5/(3+4)$$

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

8.2 연산 우선순위

하나의 식에서 연산자마다 우선순위를 가지고 있으며, 같은 우선순위에서는 왼쪽에서 오른쪽으로 연산을 진행합니다. CUBLOC 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 (<<, >>)



8.3 진법표현

CUBLOC BASIC 에서 진법에 따라 상수를 표현하는 방법은 다음과 같습니다..

```
10 진수 :    10, 20, 32, 1234

16 진수 :    &HA, &H1234, &HABCD

2 진수 :     &B10001010, &B10101,
```

8.4 단항연산자

더하기, 나누기 등과 같이 연산을 하기 위해 두 개의 수가 필요한 연산자는 이항연산자라고 부릅니다. 단항연산자는 하나의 수를 가지고 연산을 수행하는 연산자입니다.

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 위치의 비트를 1 로 하는 32 비트 정수 값을 리턴합니다.

NCD 에서는 괄호가 필요합니다.

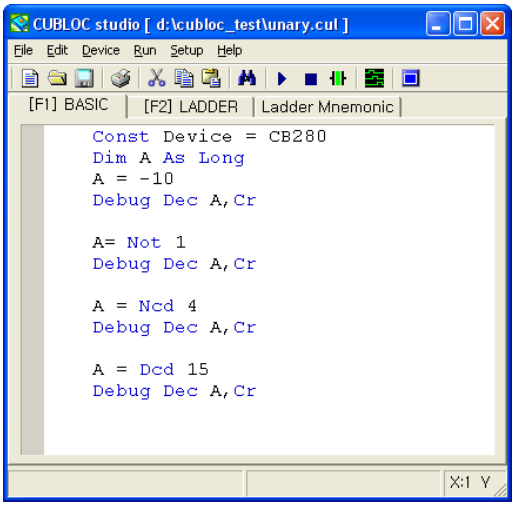
```
I = NCD(4)      '결과는 0000000000010000  4 번째 비트가 1 이 된 것입니다.
```

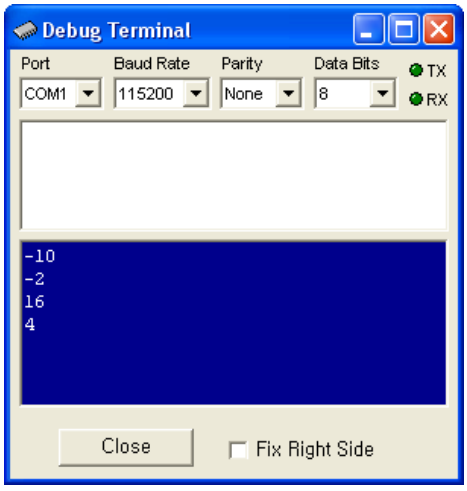
DCD

앞의 NCD 와 반대개념입니다. 어떤 숫자의 비트가 있는 곳의 위치를 알려줍니다. 단, 1 로 되어있는 가장 첫번째비트를 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.





8.5 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 EX 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 정수부만 취해서 실수형으로 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)
```

8.6 실수연산시 주의사항

실수형변수를 포함한 연산식에서는 정수형 상수도 다음과 같이 실수형으로 작성해야합니다.

```
Dim F1 As Single
Dim F2 As Single

F2 = 3645.0     ' 3654 는 정수지만 .0 을 붙여 실수형태로 작성해서 대입합니다.
F1 = F2 * 3.0   ' 정수형 상수 3 을 3.0 으로 바꾸어 표기.
```

8.7 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.

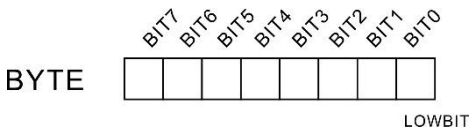
사용 예)

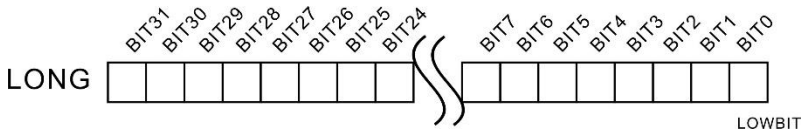
```
DIM  A  AS INTEGER
DIM  B  AS BYTE
B = A.BYTE1        ‘ 변수 A 의 BYTE1 를 변수 B 에 저장
A.LOWBYTE = &H12 ‘ 변수 A 의 LOWBYTE 에 &H12 를 저장
```

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

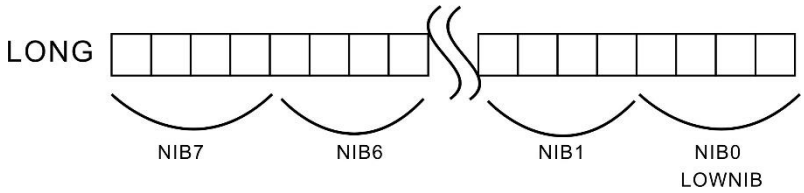




변수의 특정니블을 지정하기 위해서 사용하는 문자는 다음과 같습니다. 니블이란 4 비트를 의미합니다.

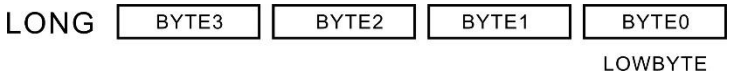
LOWNIB 변수의 NIBBLE 0
NIB0~7 변수의 NIBBLE 0~7

사용 예)A.NIB3 = 7 ‘변수 A 의 니블 3 을 7 로 만듦



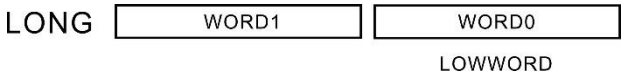
변수의 특정바이트를 지정하기 위해서 사용하는 문자는 다음과 같습니다.

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



- **비트 바이트 지정자는 _D(0).BIT0 과 같이 레더영역 참조변수 뒤에 사용할 수 없습니다.**
- **비트 바이트 지정자는 A(0).BIT0 과 같이 배열 뒤에 사용할 수 없습니다.**

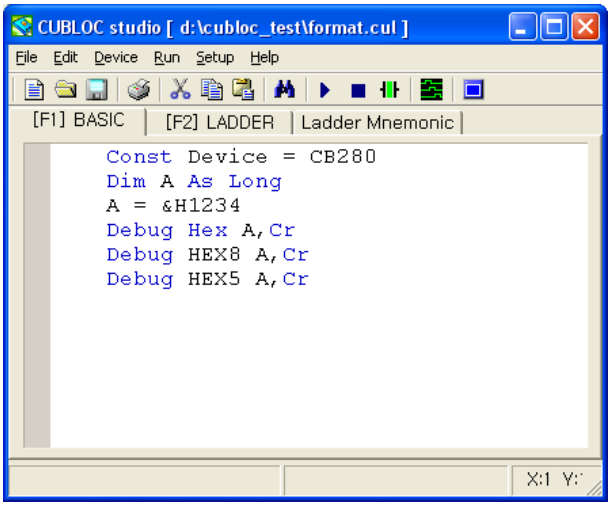
10. 형식변환자

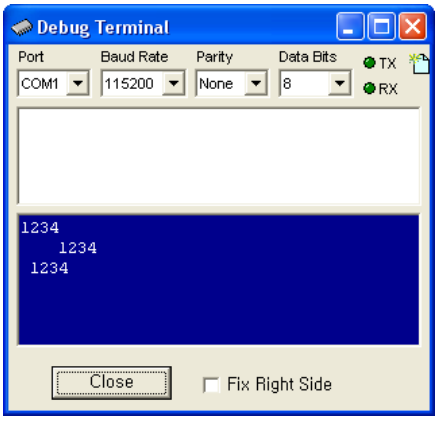
형식변환자는 변수의 내용을 표시할 때 십진수로 표시할 것인지, 16 진수로 표시할 것인지, 자릿수는 얼마로 할 것인지 등을 결정하는 문장입니다.

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

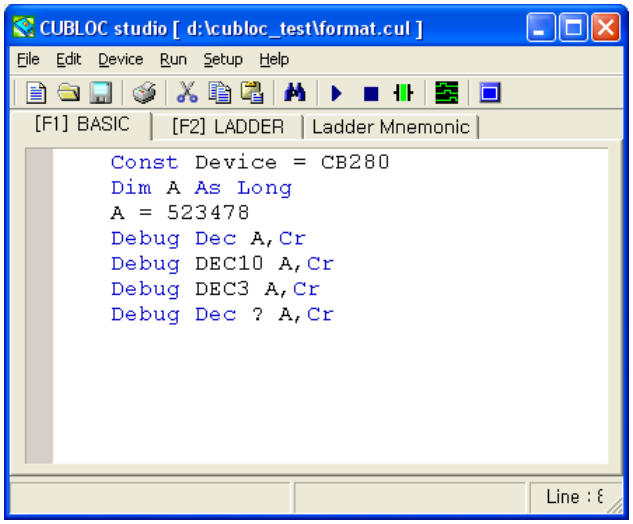
변수의 내용을 부호 있는 10 진수로 표시합니다. 바로 뒤에 숫자가 오는 경우에는 “표시하고자 하는 자릿수”를 의미합니다. DEC 에서 자릿수는 1~11 까지 사용 가능합니다.

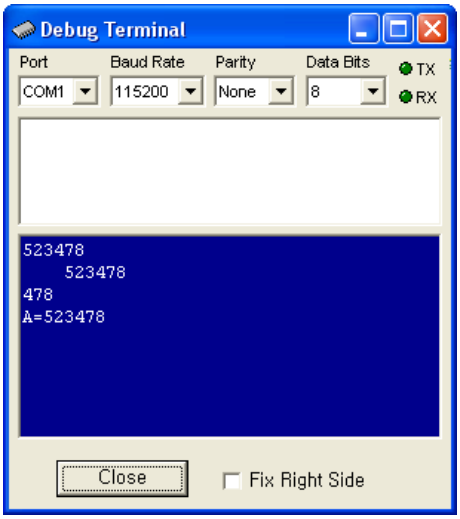
```
DEBUG DEC A     'A 가 1234 일 경우 1234 가 그대로 표시됩니다.
DEBUG DEC10 A   'A 가 1234 일 경우 bbbbbb1234 로 표시, 앞부분의 b 자리에
                '공백이 표시됨
DEBUG DEC3 A    'A 가 1234 일 경우 234 로 표시, 앞부분이 잘립니다.
```

?

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)
```

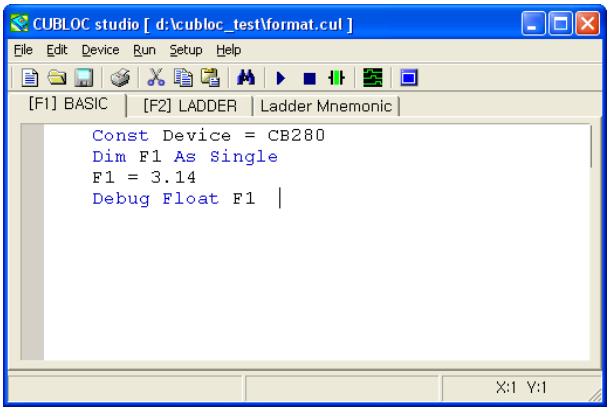


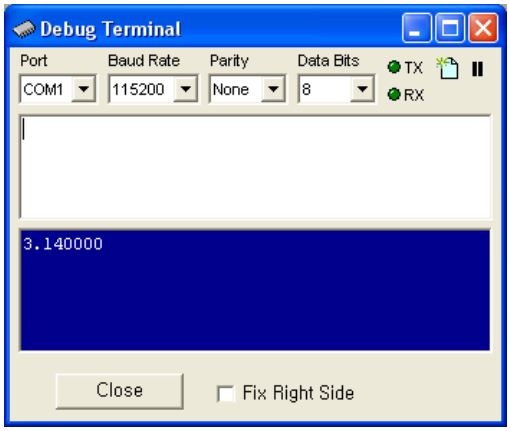


FLOAT

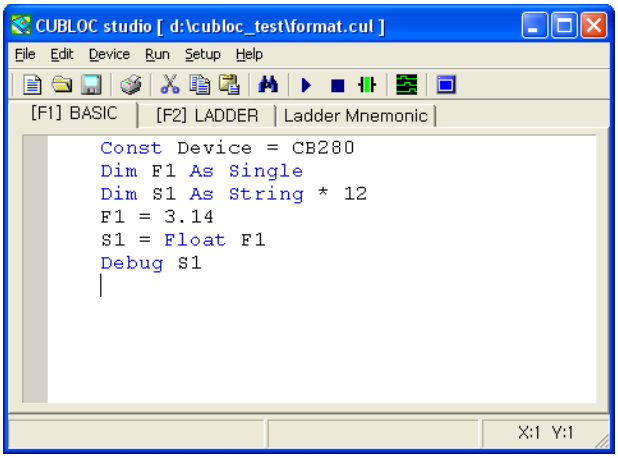
실수를 표시하고자 할 때 사용합니다. FLOAT 에서는 표시자릿수는 별도로 지정하지 않습니다.

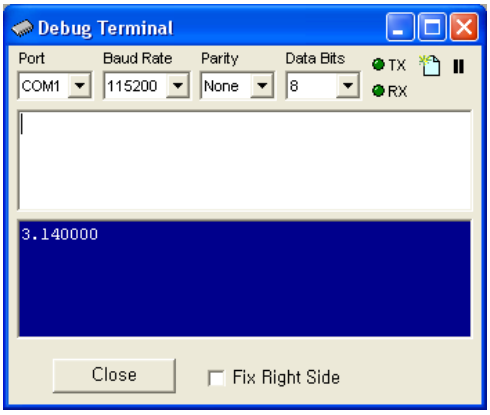
```
DIM F1 AS SINGLE
F1 = 3.14
DEBUG FLOAT F1              '3.140000 로 표시됩니다.
```





문자열 변수에 일단 저장했다가 표시하는 것도 가능합니다.



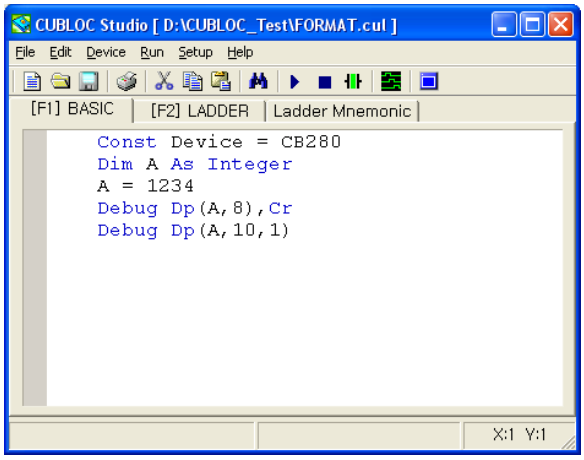


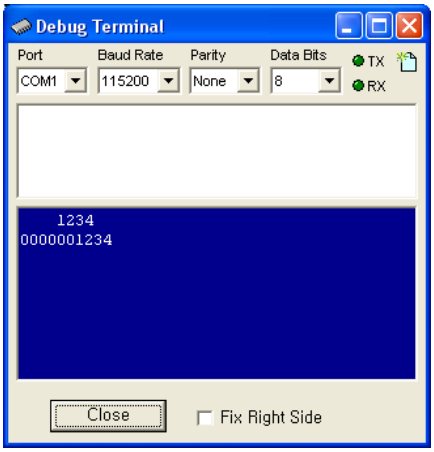
11. 문자열 처리함수

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.
```

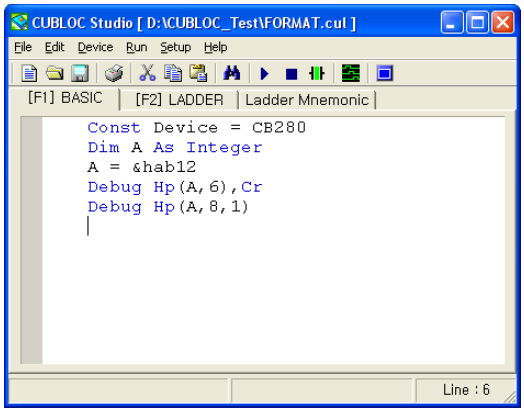


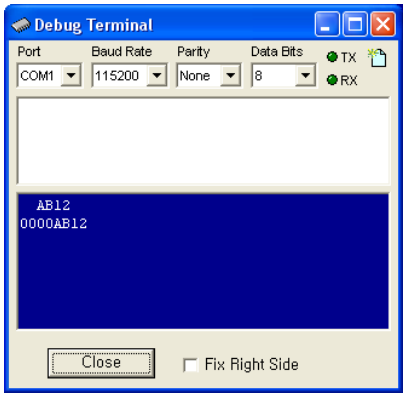


HP(Value, Length, Zero)

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.
```

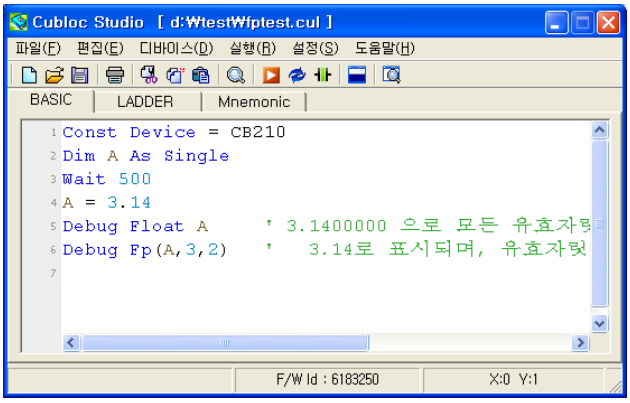


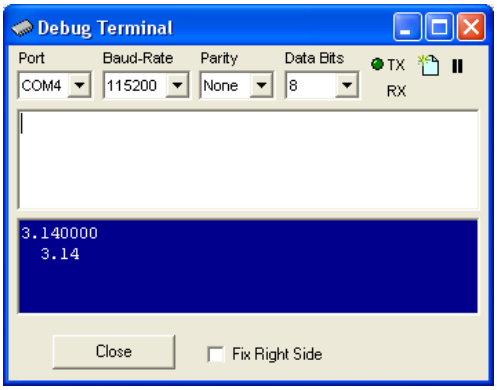


FP(Value, 정수부길이, 소수부길이)

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.
```

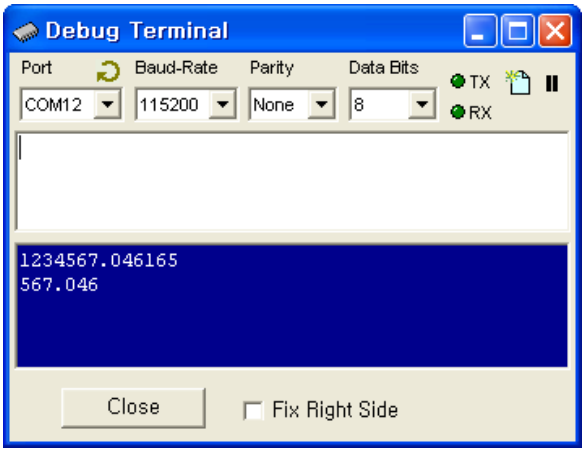




마이너스 표시도 정수부의 한자리를 차지하므로, -3.44 와 같은식으로 표시하려면 정수부를 2 자리수로 해야됩니다.

실제값보다 작은 자릿수로 지정했을 경우, 다음과 같이 앞부분이 잘려서 나올수가 있으므로 주의바랍니다.

```
a1 = 1234567.12
Debug Float a1,Cr
Debug Fp(a1,3,3)
```



*CUBLOC 은 32 비트 배정도 실수를 사용하므로 정밀도가 다소 떨어집니다. 3.4 가 3.39999 로 표시될 수 있음을 감안하고 사용하시기 바랍니다.

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)

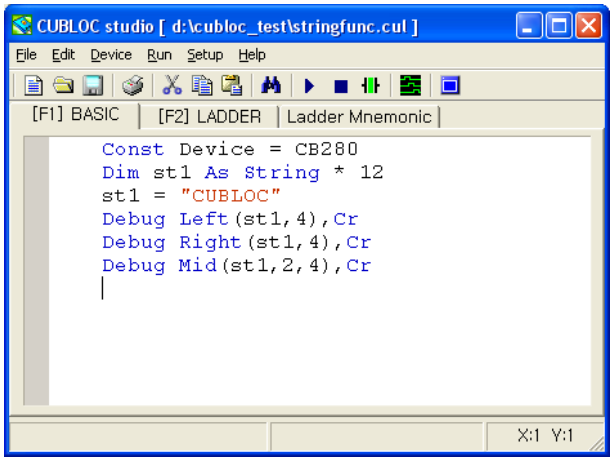
오른쪽에서 자릿수만큼 잘라낸 문자열을 반환합니다. 자릿수는 반드시 상수를 사용해야 합니다.

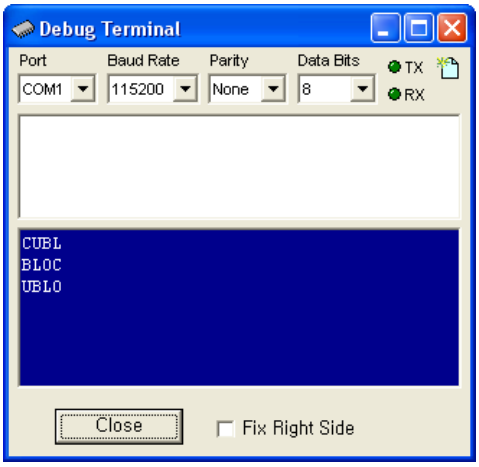
```
DIM ST1 AS STRING * 12
ST1 = "CUBLOC"
DEBUG RIGHT(ST1,4)  'BLOC 가 표시됩니다.
```

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)

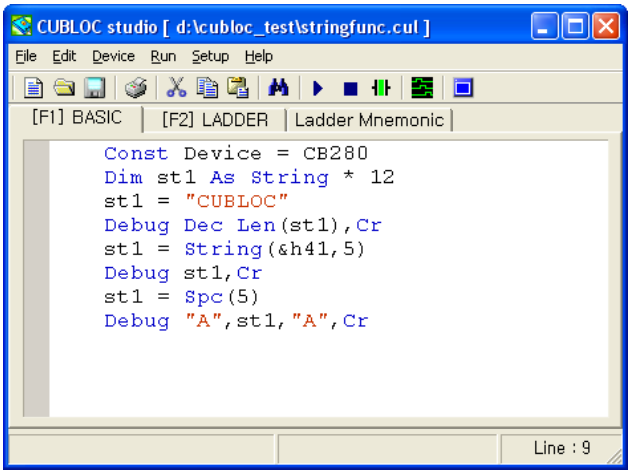
아스키코드에 해당하는 문자를 자릿수만큼의 크기를 가진 문자열로 만들어 반환합니다.

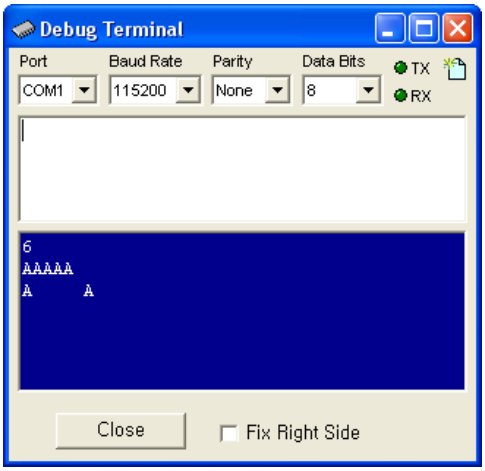
```
DIM ST1 AS STRING * 12
ST1 = STRING(&H41,5)
DEBUG ST1      'AAAAA 이 표시됩니다.&H41 은 A 의 아스키코드입니다.
```

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)

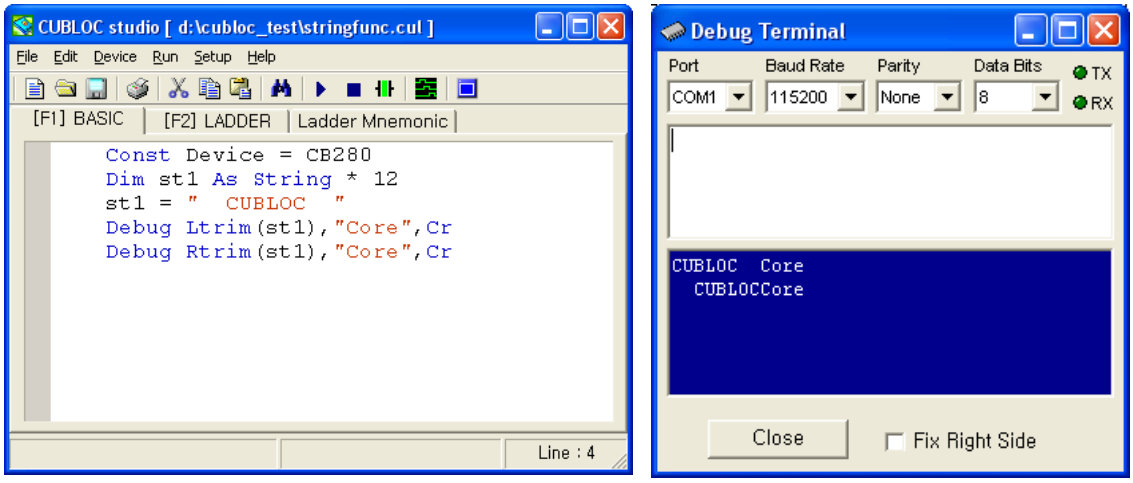
문자열의 왼쪽 공백을 제거하여 반환합니다.

```
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)

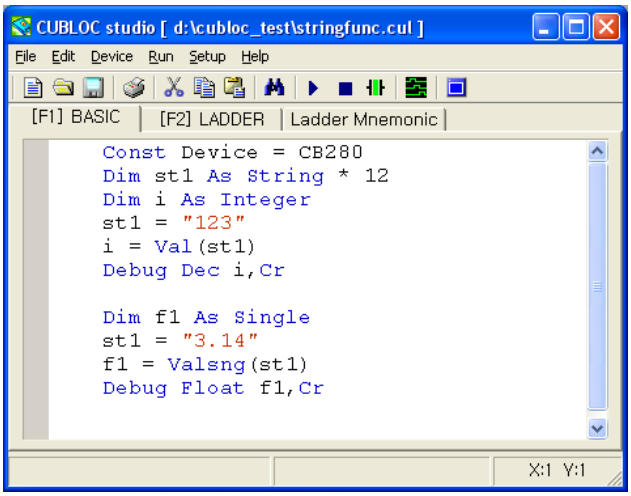
10 진수로 되어 있는 문자열의 내용을 정수형 숫자로 바꾸어 반환합니다.

```
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)

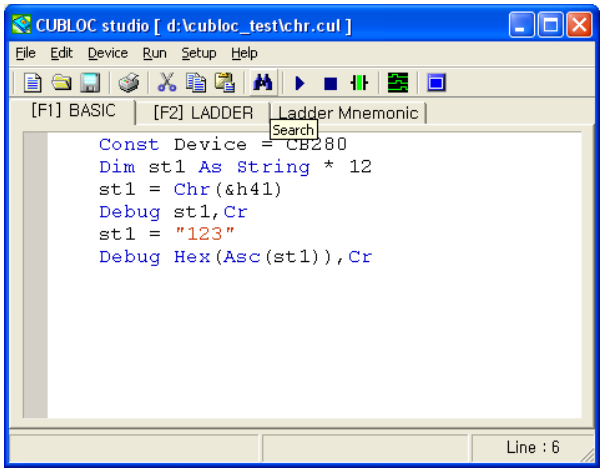
아스키코드에 해당하는 문자를 반환합니다.

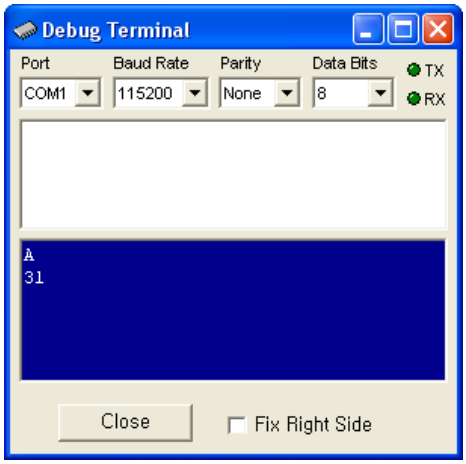
```
DIM ST1 AS STRING * 12
ST1 = CHR(&H41)
DEBUG ST1        ‘ A 가 표시됩니다. 문자 A 의 아스키코드는 &H41 입니다.
```

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"

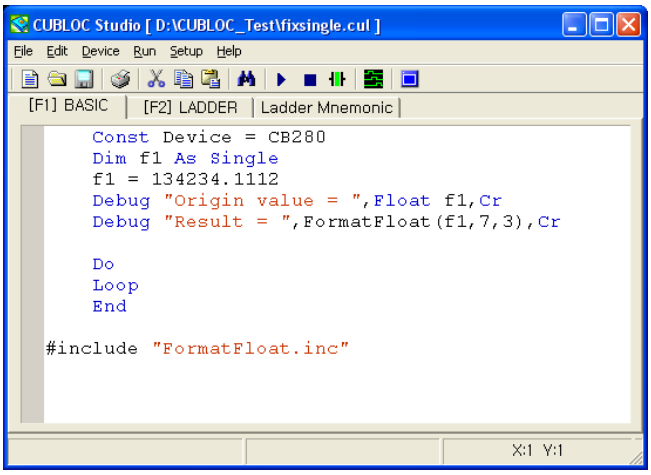
파일을 삽입합니다. 삽입된 파일은 반드시 베이직 소스파일이어야 합니다. 소스가 들어있는 폴더에 같이 들어있는 경우에는 아래와 같이 파일명만 적어줍니다.

```
#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.

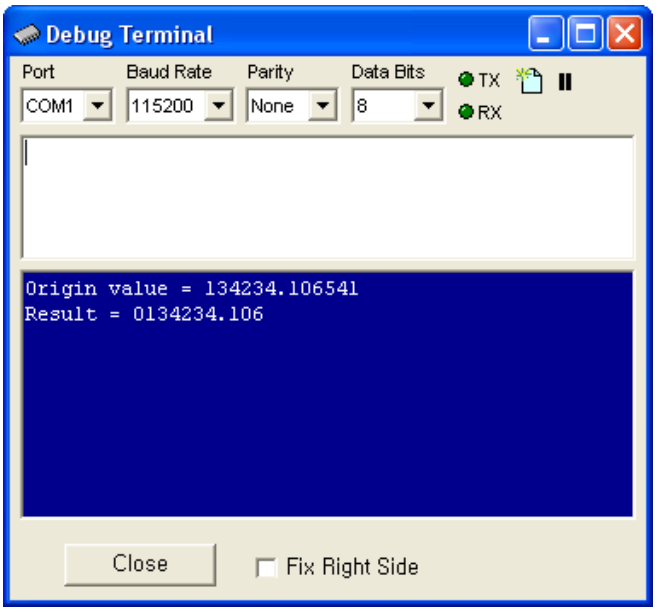
이미 완성된 부 프로그램 (더 이상 고칠 필요가 없다고 판단되는 ...)은 따로 파일로 만들어 놓은 뒤 #include 문을 사용하면, 소스프로그램을 좀더 깔끔하게 정리할 수 있게 됩니다.

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.
```

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
```

#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 선언한 적이 있는 이름을 적어준다면 조건결과는 참이 되며 #ifdef…#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 #ifndef…#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
```

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 0,8,10,3300,4400,30000
#If 0
   Do While Stepstat(0) > 0          ' 이 블록을 실행하지 않습니다.
   Loop                              ' 이 블록을 실행하지 않습니다.
#Endif

   Wait 1500
Loop
```

14. 사용 디바이스 선언

여러 종류의 CUBLOC 제품중 어떤 제품을 사용하는지 컴파일러에게 알려주어야 합니다. 제품에 따라 CONST DEVICE 명령 또는 #Include 명령을 사용합니다.

이 명령은 반드시 소스프로그램의 가장 첫 번째 줄에 위치해야 합니다.

코어모듈 사용시

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

#include "CB405RT"          ' 코어모듈중 유일하게 CB405RT 만  #include 명령을 사용합니다.
```

일체형 CUSB 시리즈 사용시

CUSB 시리즈에는 CB280 코어모듈이 장착되어 있습니다. 따라서 다음과 같은 명령어를 사용합니다.

```
CONST DEVICE = CB280
```

MSB 시리즈 사용시

MSB 시리즈는 CONST DEVICE 명령대신 #include 명령을 사용합니다.

```
#include "MSB6XX"           'MSB6XX 시리즈를 사용합니다.
#include "MSB7XX"           'MSB7XX 시리즈를 사용합니다.
```

CUTOUCH 제품 사용시

CT1721C 는 다음과 같이 선언합니다.

```
CONST DEVICE = CT1720
```

제 4 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
```

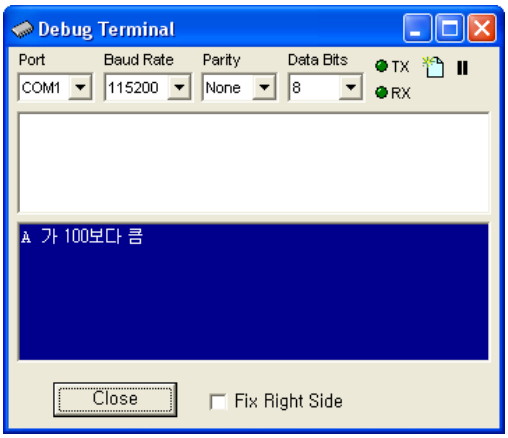
- 주의사항 : ELSEIF 에서 ELSE 와 IF 는 붙여서 작성하세요.
- EndIF 를 사용할 때, End 와 If 사이에 공백이 있어도 되고, 공백 없이 붙여서 사용해도 됩니다.
- 같은 변수 형끼리 비교해야 합니다. 실수형변수와 정수형 변수를 서로 비교할 수 없습니다.

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
' A 와 B 는 비교불가
IF A = B THEN A = 0
```

문자열 변수는 문자열변수끼리만 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
    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.

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
```

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
```

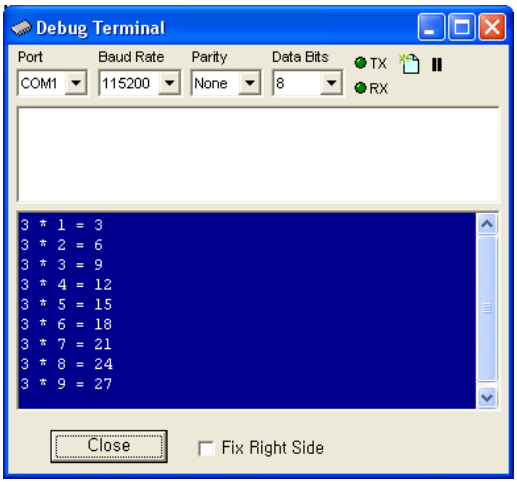
STEP -1 을 사용해서 스텝 다운하는 경우에도 음수 값을 저장할 수 있는 LONG 형 변수를 사용해야 합니다.

```
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
```

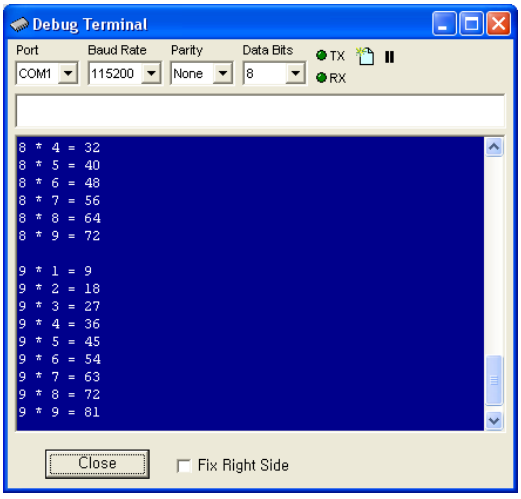
DEMO PROGRAM

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.
```

제 5 장
CUBLOC BASIC
라이브러리

라이브러리의 종류

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

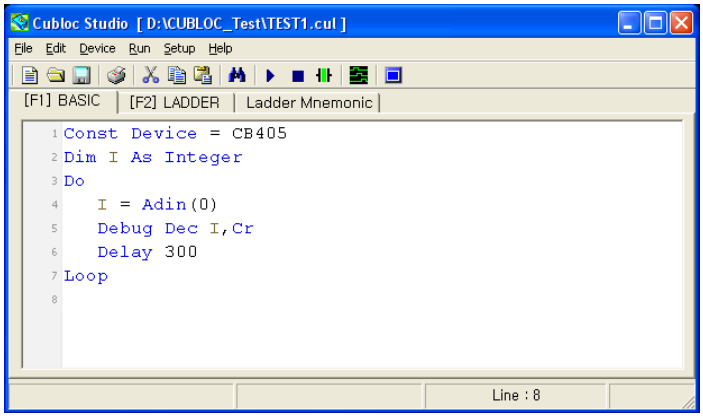
라이브러리에는 “명령문”형식과 “함수”형식이 있습니다.

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 Port

```
Port : 사용 가능한 I/O 를 가리키는 변수 또는 상수
```

코어모듈

코어모듈은 기본적으로 양방향성 I/O 포트를 지원합니다. 따라서 입출력 방향을 유저가 결정해주어야 합니다. Input 명령은 코어모듈의 특정 포트를 HIGH-Z (하이임피던스) 입력상태로 설정합니다. CUBLOC 코어 모듈의 모든 I/O 는 파워 온 시 HIGH-Z 입력상태가 됩니다. 하이 임피던스란 저항이 매우 높다는 뜻으로, HIGH 도 LOW 도 아닌 상태를 말합니다. 이 명령어는 출력전용 포트에는 사용할 수 없습니다.

```
INPUT 8     '8 번 포트를 입력상태로 만듭니다.
```

CB290 의 출력전용 포트는 Set Outonly On 명령어를 실행하기 전까지는 HIGH-Z 상태였다가, Set Outonly On 명령실행후 Output 상태가 됩니다. 이때 미리 상태를 결정해두지 않는다면 엉뚱한값이 출력될수 있으므로, 사전에 초기화한뒤 Set Outonly On 을 실행하시기 바랍니다.

일체형 CUSB 제품

일체형 CUSB 제품은 CB280 코어모듈이 장착되어있습니다. CB280 코어모듈 자체의 입출력 설정은 자유롭지만 CUSB 제품에 따라서 입력으로 할당된 포트와 출력으로 할당된 포트가 있습니다. CUSB 제품은 그에 맞추어서 입출력 상태를 소스프로그램 맨 첫부분에서 정의해주어야 합니다. 해당 모델의 설명서를 참조하시기 바랍니다.

일체형 MSB 제품

일체형 MSB 제품은 0 부터 31 까지는 입력 회로, 32 부터 63 까지는 출력 회로로 연결되어 있습니다. 입력포트는 특별히 Input 명령을 사용하지 않아도 됩니다. 출력포트는 최초 파워온시 OFF 상태가 됩니다.

Output

OUTPUT Port

```
Port : 사용 가능한 I/O 를 가리키는 변수 또는 상수 (모델 별로 차이가 있음)
```

코어모듈

코어모듈의 특정 포트를 출력상태로 설정합니다. CUBLOC 코어 모듈의 모든 I/O 는 파워 온 시 입력상태가 됩니다. INPUT 또는 OUTPUT 명령을 사용해서 I/O 의 입출력상태를 조정할 수 있습니다. 이 명령어는 입력전용 I/O 에는 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.

일체형 CUSB 제품

일체형 CUSB 제품은 CB280 코어모듈이 장착되어있습니다. CB280 코어모듈 자체의 입출력 설정은 자유롭지만 CUSB 제품에 따라서 입력으로 할당된 포트와 출력으로 할당된 포트가 있습니다. CUSB 제품은 그에 맞추어서 입출력 상태를 소스프로그램 맨 첫부분에서 정의해주어야 합니다. 해당 모델의 설명서를 참조하시기 바랍니다.

일체형 MSB 제품

일체형 MSB 제품은 0 부터 31 까지는 입력 회로, 32 부터 63 까지는 출력회로로 연결되어 있습니다. 파워온시 모든 출력포트는 OFF 상태입니다. 이후 Output 명령으로 출력포트를 출력상태로 바꾸면, 출력이 시작됩니다. 이때 쓰레기 값 (파워 온시 임의로 결정되어 버리는 엉뚱한 값)으로 출력이 나가는 경우가 발생 할 수 있기 때문에, Output 명령대신 High 나 Low 명령으로 확실한 상태를 결정해주시는 것이 좋습니다.

In( )

Variable = IN(Port)

```
Variable : 결과가 저장될 변수
Port : 사용 가능한 I/O 를 가리키는 변수 또는 상수
```

코어모듈

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

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

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

코어모듈에서는 0 부터 5V 까지 입력이 가능합니다.

일체형 CUSB 제품

일체형 CUSB 제품은 CB280 코어모듈이 장착되어있습니다. 입력포트에는 옵토커플러를 사용한 회로가 추가되어 있습니다. 해당 포트로 12V 또는 24V 가 입력하면 In( )함수에서 1 을 읽어옵니다.

일체형 MSB 제품

일체형 MSB 제품은 24V 입력회로가 포트 0 부터 31 까지 내장되어 있습니다. 해당 포트로 24V 를 입력하면 In()함수에서 1 을 읽어옵니다.

Out

OUT Port, Value

```
Port : 사용 가능한 I/O 를 가리키는 변수 또는 상수
Value : I/O 에 출력할 값을 가지고 있는 변수 또는 상수
```

코어모듈

코어모듈의 특정 I/O 를 출력으로 변경하고, 포트의 상태를 HIGH 또는 LOW 로 변경하는 명령입니다. 따라서 별도로 Output 명령을 사용해서 출력포트로 설정해 줄 필요가 없습니다. 이 명령어는 입력전용 I/O 에는 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.

일체형 CUSB 제품

일체형 CUSB 제품의 출력포트는 RELAY 로 되어 있습니다. Out 명령어로 High(1)를 출력하면 RELAY 의 접점은 붙습니다. (접점 ON 상태가 됩니다.) Low(0)를 출력하면 RELAY 의 접점은 떨어집니다. (접점 OFF 상태가 됩니다.)

일체형 MSB 제품

일체형 MSB 제품의 출력포트는 NPN TR 또는 릴레이로 되어 있습니다. Out 명령어로 High(1)를 출력하면 NPN TR 또는 릴레이가 ON 상태가 됩니다. Low(0)를 출력하면 NPN TR 또는 릴레이가 OFF 상태가 됩니다.

High

HIGH Port

```
Port : 사용 가능한 I/O 를 가리키는 정수형 변수 또는 상수
```

특정 I/O 출력을 HIGH 상태로 만듭니다. 이 명령을 사용하여 I/O 를 HIGH 로 만들면, 해당 포트는 OUTPUT 상태가 됩니다.

이 명령어는 입력전용 포트에는 사용할 수 없습니다.

```
HIGH 8    '8 번 포트 출력을 HIGH 로 만듭니다.
```

다른 CUBLOC 제품군의 출력설명은 앞페이지의 Out 설명을 참조하세요.

Low

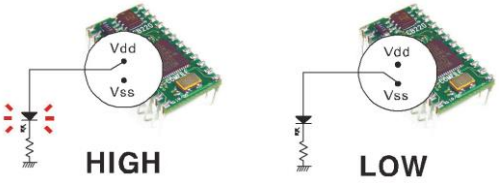
LOW Port

```
Port : 사용 가능한 I/O 를 가리키는 변수 또는 상수
```

특정 포트 출력을 LOW 상태로 만듭니다. 이 명령을 사용하여 특정 포트를 LOW 로 만들면, 해당포트는 OUTPUT 상태가 됩니다.

이 명령어는 입력 전용 포트에는 사용할 수 없습니다.

```
LOW 8          '8 번 포트 출력을 LOW 로 만듭니다.
```



다음 샘플 프로그램을 실행시키면, P8 에 연결된 LED 가 일정한 시간간격으로 점멸합니다.

```
Const Device = CB280
Do
      High 8
      Delay 100
      Low 8
      Delay 100
Loop
```

Byteout

BYTEOUT PortBlock, Value

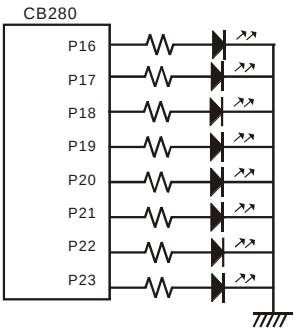
```
PortBlock : 사용 가능한 포트블록을 가리키는 변수 또는 상수
Value : 포트블록에 출력할 변수 또는 상수 (0~255 사이의 값)
```

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.

포트블록	해당포트
0	0 ~ 7
1	8 ~ 15
2	16 ~ 23
3	24 ~ 31
4	32 ~ 39
5	40 ~ 47
6	48 ~ 55
7	56 ~ 63
8	64 ~ 71
9	72 ~ 79

모델 별로 사용 가능한 포트블록에는 차이가 있습니다. BYTEOUT 명령을 사용하면 해당 포트 블록은 모두 OUTPUT 상태로 설정되고, 원하는 값이 출력됩니다. 다음 샘플프로그램을 실행시키면, CB280 의 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 : 포트블록을 가리키는 변수 또는 상수
```

특정 포트블록에서 포트 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.

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

Outstat( )

Variable = OUTSTAT(Port)

```
Variable : 결과가 저장될 변수
Port :  I/O 포트를 가리키는 변수 또는 상수
```

특정 포트에 출력중인 값을 읽어옵니다. 이 함수는 입력포트에는 사용할 수 없습니다.

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

Reverse

REVERSE Port

```
Port : I/O 포트를 가리키는 변수 또는 상수
```

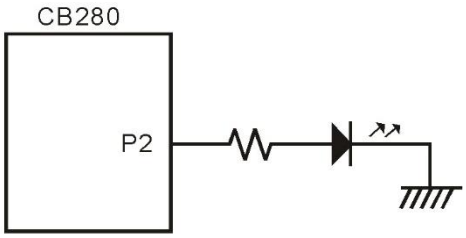
특정 포트의 출력을 반전시킵니다. HIGH 를 출력 중이면 LOW 로 만들고, LOW 를 출력 중이면 HIGH 로 만듭니다. 이 명령어는 입력 전용포트에는 사용할 수 없습니다.

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

DEMO PROGRAM

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

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



아날로그 입출력

CUBLOC 코어모듈 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)

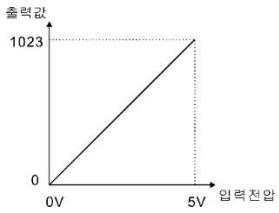
```
Variable : 결과가 저장될 변수
Channel : AD 입력 채널 (I/O 포트 번호가 아님)
```

입력중인 아날로그 값을 읽어서 지정한 변수에 저장합니다. CUBLOC 모델에 따라 사용할 수 있는 아날로그 입력 포트가 다소 차이가 있습니다.

코어모듈에서의 A/D 입력

코어모듈에는 별도의 AVREF 단자가 있습니다. AD 입력 채널에 0~AVREF 사이의 전압을 인가하면 0~1023 사이의 값으로 변환합니다. AVREF 단자에는 2V~5V 까지 사용가능하며, 일반적으로 5V 를 연결해 줍니다.

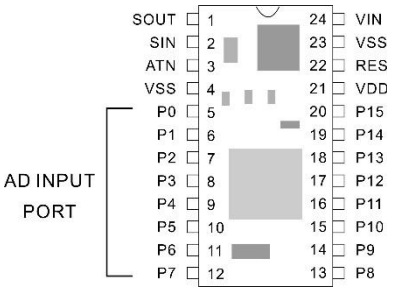
AVREF 에 3V 를 연결하면 0~3V 사이를 0~1023 으로 변환하게 됩니다. AVREF 에 5V 이상의 전압을 인가할 수는 없으므로 주의하시기 바랍니다. 일부 CUBLOC 코어 모델의 경우 별도의 AVREF 단자가 나와있지 않은 경우가 있는데, 이 모델에서는 내부적으로 AVREF 단자가 5V 에 연결되어 있습니다.

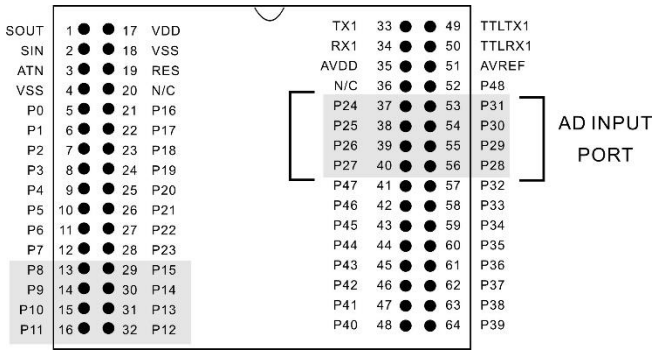



```
DIM A AS INTEGER
INPUT 24        ' 입력상태로 만든다.
A=ADIN(0)       '0 번 채널에서 A/D 변환을 해서 변수 A 에 저장
```

ADIN (채널) 에서 채널은 포트번호가 아닙니다. **반드시 입력받고자 하는 채널 번호 (0 부터 시작)을 입력해 주시기 바랍니다. 해당 포트는 입력 상태이여야 합니다.** 파워온 뒤 모든 포트는 입력상태이기 때문에 특별히 바꿔야할 일은 없지만, 만약 중간에 출력상태로 바뀌었다면, 다시 입력상태로 바꾼뒤 ADIN 함수를 사용하시기 바랍니다.

다음은 코어모듈의 모델 별로 A/D 입력 가능 포트를 표시한 것입니다.





모델 별 A/D 채널과 포트와의 관계는 다음 표를 참조하시기 바랍니다. 이표에 나열한 AD 는 모두 10 비트 AD 입니다. 결과값은 0 부터 1023 사이의 값이 됩니다.

채널-모델명	CB210	CB220 CB320	CB280 CB380	CB290	CT172X	CB405 CB400	CB405RT
A/D channel 0	I/O 16	I/O 0	I/O 24	I/O 8	I/O 0	I/O 16	I/O 16
A/D channel 1	I/O 17	I/O 1	I/O 25	I/O 9	I/O 1	I/O 17	I/O 17
A/D channel 2	I/O 18	I/O 2	I/O 26	I/O 10	I/O 2	I/O 18	I/O 18
A/D channel 3	I/O 19	I/O 3	I/O 27	I/O 11	I/O 3	I/O 19	I/O 19
A/D channel 4	I/O 20	I/O 4	I/O 28	I/O 12	I/O 4	I/O 20	I/O 20
A/D channel 5	I/O 21	I/O 5	I/O 29	I/O 13	I/O 5	I/O 21	I/O 21
A/D channel 6		I/O 6	I/O 30	I/O 14	I/O 6	I/O 22	I/O 22
A/D channel 7		I/O 7	I/O 31	I/O 15	I/O 7	I/O 23	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.

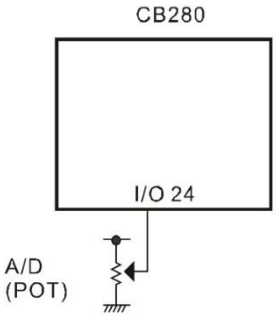
CB405RT 는 16 비트 AD 변환기를 내장하고 있으며, 사용명령어도 다릅니다. <코어모듈 사용설명서>를 참조하시기 바랍니다.

http://www.comfile.co.kr/download/cubloc/cubloc_core_module.pdf 참조

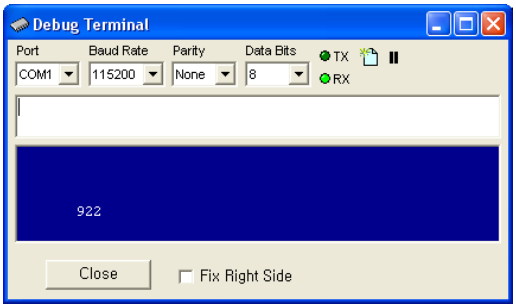
DEMO PROGRAM

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.



MSB 시리즈에서의 A/D 입력

일부 MSB 시리즈에는 A/D 입력 포트를 갖추고 있습니다. (MSB612RA-DC 의 경우)

A/D 입력 채널	종류	결과값
0~3	0~20mA	0~1023
4~7	0~10VDC	0~1023

코어모듈에 있는 A/D 포트를 제대로 활용하기 위해서는 추가적인 입력보호회로 및 노이즈필터회로가 필요합니다. 하지만 MSB 시리즈는 기본적인 LPF (로우패스필터)회로와 보호회로가 내장되어 있습니다.

```
#Include "MSB6XX"
Dim A As Integer
Do
      A=Adin(0)
      Debug Goxy,5,3
      Debug dec5 A
      Delay 200
Loop
```

위 예제 프로그램은 채널 0 으로부터 값을 읽어 Debug 창에 표시하는 프로그램입니다.

Pwm

PWM Channel, Duty, Width

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

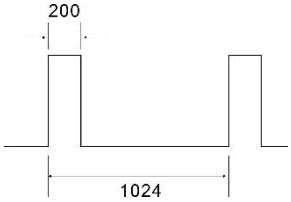
원하는 채널에 PWM 파형을 출력합니다. PWM 명령에서는 포트번호가 아닌 채널번호를 사용하므로, 모델 별 채널에 따른 포트위치를 확인하시기 바랍니다. CB280/ CB380 의 경우 포트 5,6,7 에 PWM 채널 0,1,2 가 할당되어 있습니다. PWM 명령을 사용하기 전에 해당 포트를 반드시 OUTPUT 상태로 만들어 주어야 합니다.

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.

주파수는 주기값에 의해서 결정됩니다. 원하는 주파수가 200KHz 미만일때 다음 공식에 의해 주파수를 계산할 수 있습니다.

```
주기값 = (2304000 / 주파수) - 1
```

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
```

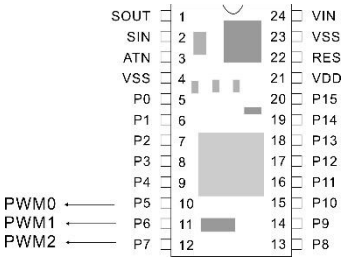
PWM 0, 1, 2 번 채널의 주기(Width)는 같은 값을 사용해야 합니다. 내부적으로 하나의 타이머로 구동되기 때문입니다. PWM 3, 4, 5 도 주기(Width)는 하나의 값으로 통일해야 합니다. PWM 0, 1, 2 와 PWM 3, 4, 5 는 서로 달라도 상관없습니다.

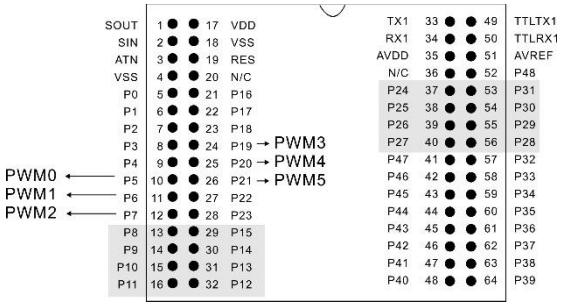
Pwmoff

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.

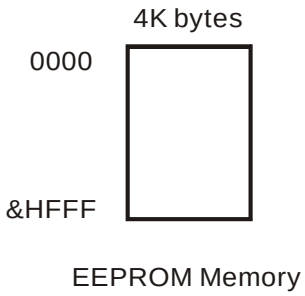
채널	CB210	CB220 CB320	CB280 CB380	CB290	CT1721C	CB405 CB405RT CB400
PWM0		I/O 5	I/O 5	I/O 5	I/O 8	I/O 5
PWM1		I/O 6	I/O 6	I/O 6	I/O 9	I/O 6
PWM2		I/O 7	I/O 7	I/O 7	I/O 10	I/O 7
PWM3	I/O 3	I/O 19	I/O 19	I/O 89	I/O 11	I/O 27
PWM4	I/O 4	I/O 20	I/O 20	I/O 90	I/O 12	I/O 28
PWM5	I/O 5	I/O 21	I/O 21	I/O 91	I/O 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

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

```
Pwmoff 0
Input 5
```

EEPROM 억세스

모든 CUBLOC 제품군에는 4K 의 EEPROM 이 내장되어 있습니다.



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 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.
Delay 5
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 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.
Delay 5
B = EEREAD(0,2)        ' 0 번지에서 읽어와서 변수 B 에 저장합니다.
```

EEPROM 에서 데이터를 써넣는 경우, 약 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 정도입니다. 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 10 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.

```
If A <> Eeread(0,1) Then
 Eewrite 0,A,1      '써야할 값(A)가 해당번지와 다를 경우에만 값을 기입합니다.
 Delay 5
Endif
```

카운터

CUBLOC 시스템에 내장된 고속카운터 기능은 내부 BASIC 또는 레더 프로그램 수행여부와 상관없이 동작되는 별도의 하드웨어로 구성되어 있습니다. BASIC 또는 레더 프로그램이 실행 중 일 때에도 항상 카운터 입력을 받아들이고 있으므로, 외부로부터 신호입력을 놓치는 일이 발생하지 않습니다.

코어모듈 사용시

코어모듈의 카운터 포트는 일반 I/O 와 같이 쓸 수 있도록 되어 있으므로, 카운터 입력을 사용하려면 I/O 기능은 사용하지 못하게 됩니다. CUBLOC 코어모듈은 2 개의 카운터 입력 포트가 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 코어모듈은 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.

CUSB 사용시

CUSB 시리즈에도 CB280 코어모듈이 장착되어 있으므로, 코어모듈의 그대로 적용됩니다. 고속카운터 입력포트가 없는 CUSB-14R 에서는 고속카운터 기능을 사용할 수 없습니다. 고속카운터 입력을 원하신다면, 고속카운터 입력포트가 있는 상위모델을 선택하시기 바랍니다.

CUTOUCH 사용시

CT1721C 에는 2 개의 고속카운터 입력기능이 있습니다. Set Count0 On 을 앞에서 선언하시면 2 채널 모두 사용가능합니다.

MSB 시리즈 사용시

MSB7XX 시리즈는 2 개의 고속카운터 입력이 있으며 항상 사용 가능한 상태로 준비되어 있습니다. 유저는 단지 Count 함수로 현재 카운트 중인 값을 읽어오기만 하면 됩니다. Set Count0 명령도 사전에 정의할 필요가 없습니다.

Count( )

Variable = COUNT(channel)

```
Variable : 결과를 저장할 변수
Channel : 카운터 채널 (0 또는 1)
```

카운트 입력 포트로부터 받은 카운터 개수를 반환합니다. CUBLOC 에는 2 개의 32 비트 고속 카운터가 있습니다. 카운터는 별도의 하드웨어로 구성되어 있기 때문에, 베이직 프로그램 수행 여부와 관계없이 항상 카운트입력 포트로부터 들어오는 펄스의 개수를 카운트하고 있다가 COUNT 함수를 통해서 읽어 낼 수 있습니다. 코어모듈의 경우 카운터 입력 포트는 사전에 입력모드로 해두어야 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

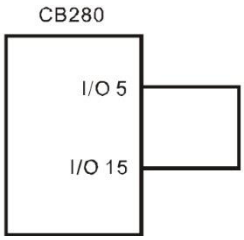
```
Channel : 카운터 채널
```

원하는 채널의 카운트 값을 0 으로 리셋 시킵니다.

```
COUNTRESET 0  '채널 0 을 카운트 값을 0 으로 만듭니다.
COUNTRESET 1  '채널 1 을 카운트 값을 0 으로 만듭니다.
```

DEMO PROGRAM

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

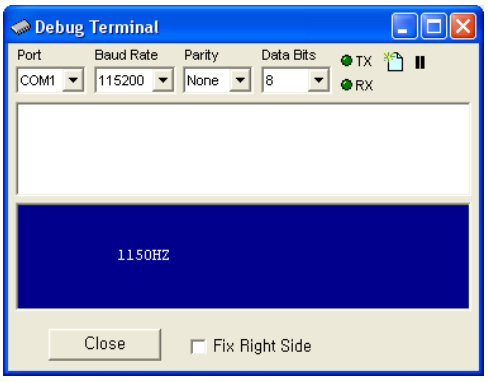


이렇게 하면, 포트 5 번에서 출력되는 PWM 파형이 COUNTER 1 번 채널로 입력됩니다.

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.

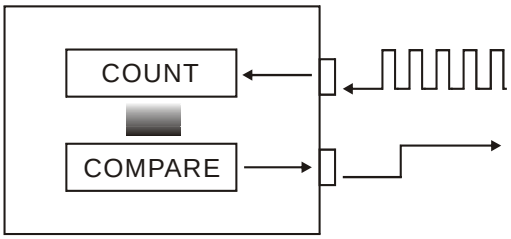
컴페어

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 High가 됩니다. 반대로 0 으로 하면 해당포트의 상태가 Low 가 됩니다.

이 명령어를 실행하면 목표상태의 반대로 해당포트가 바뀝니다. 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.

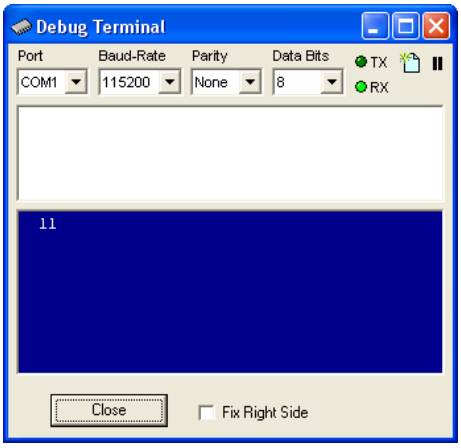
채널	비교범위
COUNT 채널 0	0~65535
COUNT 채널 1	0 ~255

고속카운터는 16 비트까지 저장할 수 있지만, COMPARE 명령에서 비교할 수 있는 범위는 다소 제한적입니다. 그 이유는 COMPARE 는 CUBLOC 메인칩의 하드웨어 기능을 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.

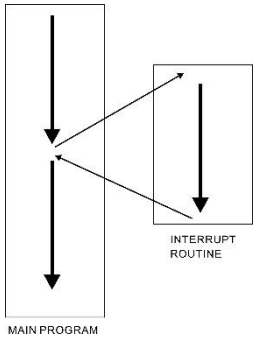
```
Dim i As Integer
Set Count0 On
Compare 0,10,61,1
Do
      i = Count(0)
      Debug Goxy,0,0,dec4 i,Cr
      Delay 100
Loop
```

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 예외적으로 처리해야 할 상황을 위해서 필요합니다. 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.



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 CUBLOC 에서 지원하는 BASIC 인터럽트 종류를 요약한 표입니다.

인터럽트 명	설명
On Timer	일정 시간간격으로 인터럽트를 발생
On Int	외부 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.

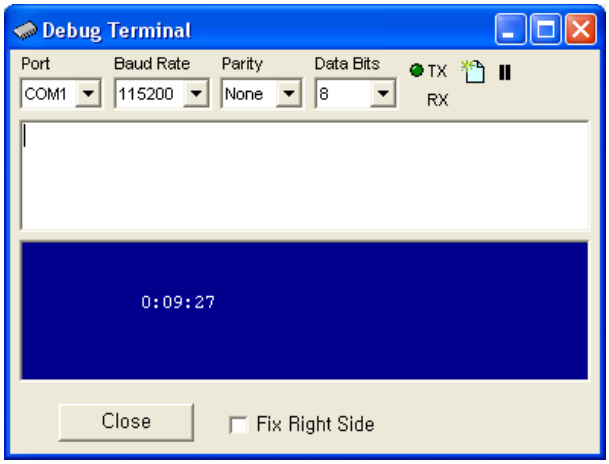
```
ON TIMER(100) GOSUB TIMERTN
DIM I AS INTEGER
I = 0
DO
LOOP

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

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. 일부 CUBLOC 제품에는 4 개의 외부 인터럽트 포트가 있습니다. 인터럽트 포트를 입력으로 정의하고, 여기에 인터럽트 신호를 입력합니다. 인터럽트 신호는 하강엣지, 상승 엣지, 둘다입력의 세가지 모드로 선택할 수 있습니다. 인터럽트 입력모드를 선택하는 명령은 SET INTx 명령입니다.

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

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

본 명령은 외부인터럽트 입력 단자가 없는 CUSB, MSB 모델에서 사용할 수 없습니다.

인터럽트 금지/재동작

인터럽트를 금지시키거나, 재 동작시킬 필요가 있을 때 사용하는 명령입니다. 전체 인터럽트를 금지시키거나 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 써주세요. CUBLOC OS 에서 인터럽트 수신 체크하는 부분을 실행하지 않으므로 전체적인 프로그램 실행시간이 다소 빨라집니다.

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
```

본 명령은 외부인터럽트 입력 단자가 없는 CUSB, MSB 모델에서 사용할 수 없습니다.

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.

CUBLOC 에서 인터럽트 사용시 유의사항..

MCU 에서는 인터럽트를 자주 사용하여 프로그램을 작성합니다. 하지만 CUBLOC 에서는 어쩔수 없는 경우가 아니라면, 인터럽트를 사용하지 않는 것이 더 좋습니다.

CUBLOC 은 하드웨어적으로 인터럽트를 처리하는 방식이 아니고, 베이직 인터프리터가 인터럽트의 발생유무를 판단하여 해당루틴으로 점프하는 방식을 채택하고 있으므로, 반응속도가 MCU 만큼 즉각적이지 않습니다.

또한 인터럽트 루틴에 들어가도, 역시 베이직 인터프리터 방식으로 명령어들을 처리하다보니, 짧은 처리시간을 요하는 인터럽트 루틴의 경우에도 처리가 어렵습니다.

따라서, **꼭 필요한 경우가 아니면, 인터럽트를 사용하지 마십시오.**

1. 인터럽트를 사용하는 경우에는 반드시, 처리루틴안에서 프로세스를 길게 붙잡아 두지마시고, 빠르게 플레그정도만 설정하고 빠져야 합니다.

인터럽트 루틴안에서 RS232 송신을 한다던가, DELAY 명령을 쓰는 것은 지극히 잘못된 방법입니다.

주된처리는 MAIN 처리부에서 하시고, 인터럽트는 특정플레그만 셋팅시켜서, 메인프로그램에서 그 플레그를 확인하는 방법이 제일 좋습니다.

2. 전체 프로그램에서 인터럽트는 1,2 개정도 종류만 사용하시기 바랍니다. 여러 개의 인터럽트가 중복적으로 발생할 경우, 예기치 못한 상황으로 빠질 수 있습니다.

결론적으로 말씀드리면, 인터럽트 때문에 메인프로그램이 정상적으로 동작하지 않는 상황이 발생하지 않도록 프로그램을 작성해야합니다.

인터럽트 우선순위

동시에 인터럽트가 발생하면 우선순위가 높은 인터럽트가 먼저 수행됩니다.

순위 1 : on Ladder 인터럽트
순위 2 : on Int 인터럽트
순위 3 : on Recv 인터럽트
순위 4 : on Pad 인터럽트
순위 5 : on Timer 인터럽트

LADDER 관련명령

CUBLOC 의 운영체계는 처음에 전원이 들어온 뒤 BASIC 명령부터 해석하여 실행합니다. BASIC 에서 LADDER 를 실행하는 명령을 수행하기 전까지는 LADDER 를 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]

10mS 간격으로 LADDER 를 실행 개시 하는 명령입니다. 최초에는 OFF 상태입니다.

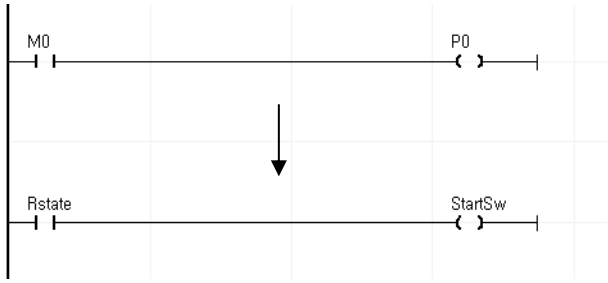
Alias

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 M1 = Kstate
Alias P0 = StartSw
```



BASIC 에서는 Alias 를 사용할 수 없습니다.

```
Alias M0 = Rstate
Rstate = 0     ' 베이직에서는 Alias 를 사용할 수 없습니다.
_M(0) = 0      ' 이런식으로 언더바를 사용한 릴레이표현을 사용해야 합니다.
```

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 포트를 정의하는 명령입니다. 파워온시 모든 I/O 포트는 BASIC 쪽에서 사용하도록 되어 있습니다. LADDER 에서 사용하고자 할 때에는 반드시 USEPIN 명령을 사용해서 LADDER 에서 사용할 것임을 명시해 두어야 합니다.

```
USEPIN 0,IN    ' 0 번포트는 레더쪽에서 사용합니다.
```

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

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

Freepin

FREEPIN I/O

```
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

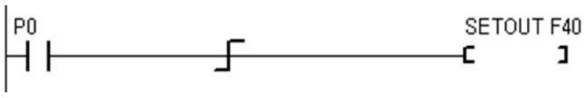
ON LADDERINT GOSUB label

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 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 해야 합니다.

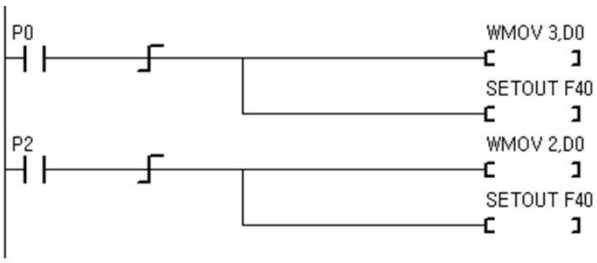
인터럽트 루틴 수행 중에는 레더쪽에서 F40 에 1 을 써넣어도 아무 효과가 일어나질 않습니다. BASIC 에서 F40 을 클리어 하면 인터럽트 수행이 모두 끝났다는 것을 의미하며, 다음 인터럽트를 받을 준비가 되었다는 것을 뜻합니다.

```
  Const Device = CB210
  Dim i As Integer
  Usepin 0,In
  i=0
  Low 30
  Set Ladder On
  On Ladderint Gosub msg1_rtn
  Do
     Reverse 30
     Delay 200
  Loop
 msg1_rtn:          ' 레더에서 인터럽트가 발생하면 Debug 창에 표시
  Debug "ON Ladderint ",Dec i,Cr
  Incr i
  _F(40) = 0        ‘ 레더 인터럽트 처리가 끝났으므로 F40 을 클리어 합니다.
  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.

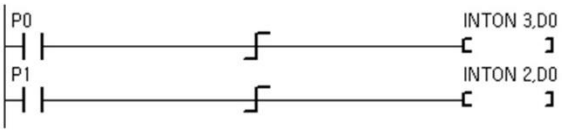
LADDER 에서 BASIC 으로 인터럽트를 발생시키는 방법은 F40 을 이용하는 단 하나의 방법뿐이지만, 데이터 영역인 D 영역의 특정번지에 원하는 처리의 종류를 기입한 후, 인터럽트를 발생시키고, BASIC 에서 해당 D 영역을 참조하여, 어떤 처리를 원하는지 읽어내는 방법을 사용하면, 하나의 인터럽트로 여러 가지 처리를 할 수 있습니다.



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

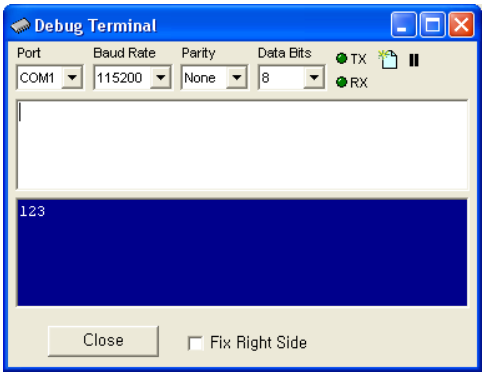
DEBUG data

```
data : PC 로 전송할 데이터
```

“디버그 터미널”로 데이터를 전송하는 명령입니다. CUBLOC 에서는 디버그 명령을 사용해서, 동작 중 보고 싶은 변수 값을 PC 상에 표시하거나, 동작상태 등을 체크합니다. 일종의 “디버그 툴”이라고 할 수 있습니다.

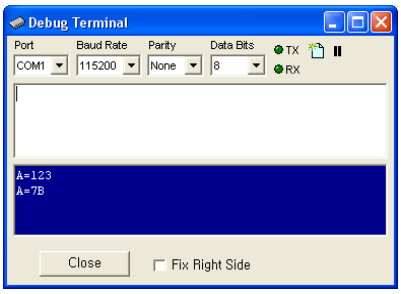
Debug 명령이 동작하기 위해서는, CUBLOC 과 PC 가 연결된 상태이어야 합니다. 이 상태에서 Debug 명령이 수행되면, CUBLOC 에서 PC 쪽으로 데이터를 전송하게 되고, PC 에서는 “디버그 터미널”에 그 값이 표시되게 됩니다.

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



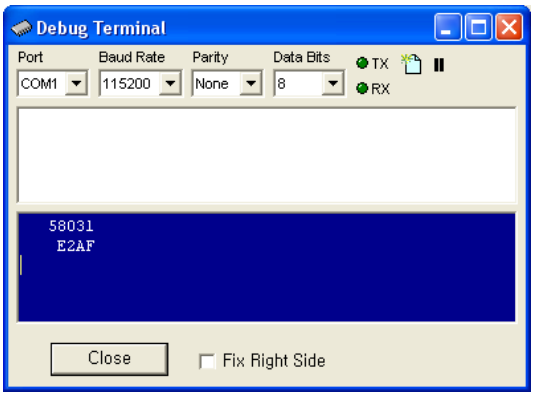
DEC 와 HEX 를 사용하여 표시할 변수의 진법을 변환합니다. 변수의 내용을 보고 싶다면 반드시 DEC 또는 HEX 를 사용해야 만 합니다. 아래의 예처럼 DEC 나 HEX 뒤에 ?를 기호를 적어준다면 변수명과 함께 표시됩니다.

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



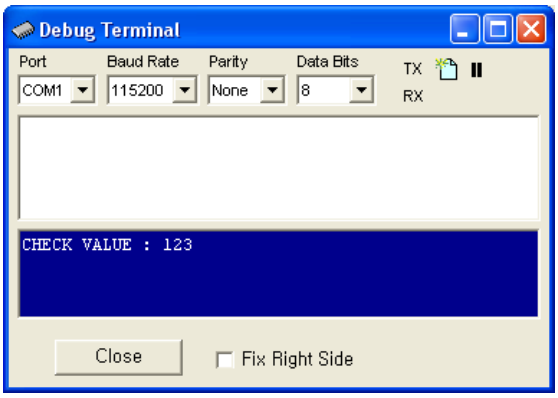
숫자를 사용하여 자릿수를 제한하는 방법도 있습니다.

```
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

```
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.
```

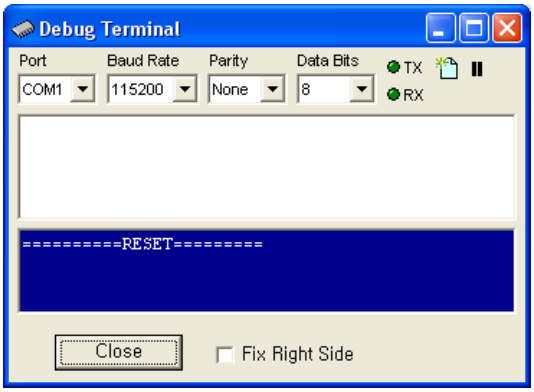
DEBUG 명령 활용법

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, CUBLOC 이 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.

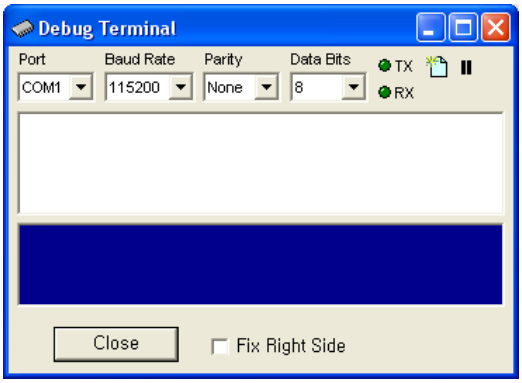
```
Const Device = CB280
Debug      "==========Reset=========="

Do
   High 0
   Delay 200
   Low 0
   Delay 200
Loop
```



2. 프로그램의 원하는 부분이 실행되었는지 확인하는 방법

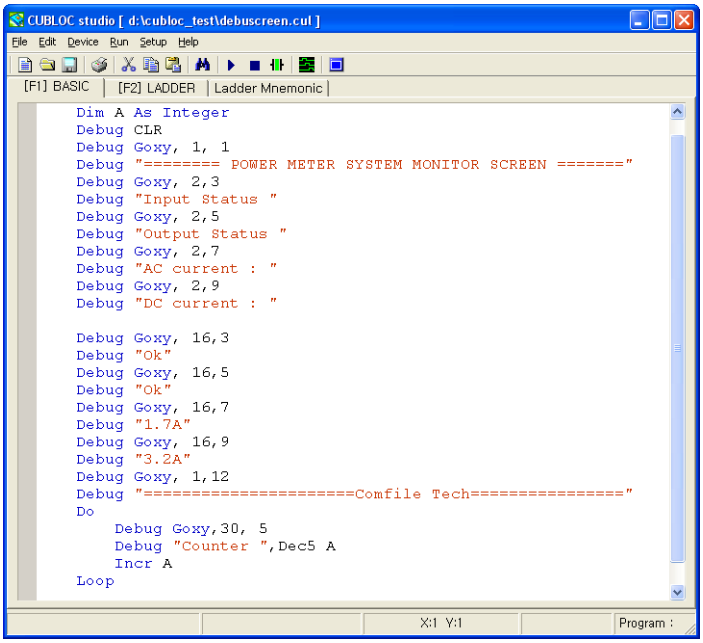
Debug 명령을 특정한 부분에 끼워 넣어서, 그 부분이 실행되고 있는지 확인하는 방법입니다. 만약 실행되지 않는다면 Debug 터미널에 해당문자가 나타나지 않게 됩니다.

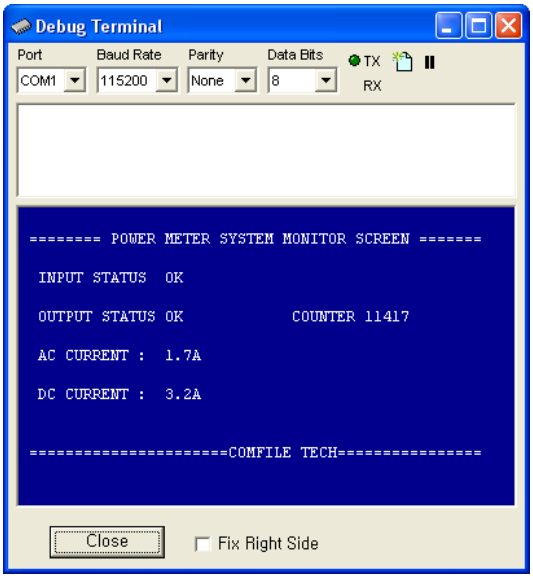
```
Const Device = CB280
Do
   High 0
   Delay 200
   Low 0
   Delay 200
Loop
Debug "Comfile Technology"
```



3. 디버그 터미널을 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 창 으로 이용하다가, 개발이 끝나면 Set Debug Off 를 하면 됩니다.

Rnd( )

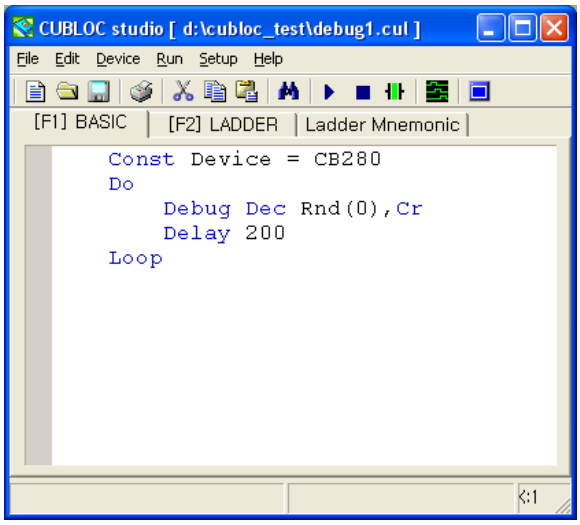
Variable = RND(0)

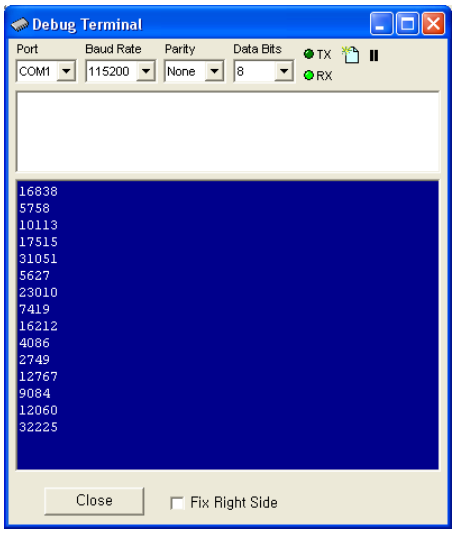
난수를 발생시키는 명령입니다. 0~32767 사이의 난수를 발생시켜 변수 variable 에 저장합니다. 괄호 안의 숫자는 아무 의미도 없는 더미(Dummy) 숫자입니다.

```
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.

```
Reset
```

Ramclear

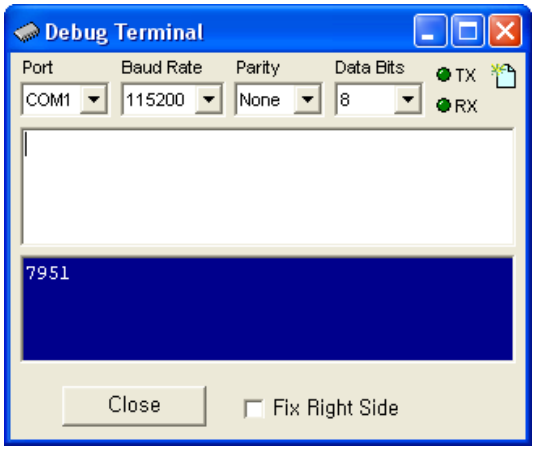
RAMCLEAR

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 CUBLOC 제품이 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.

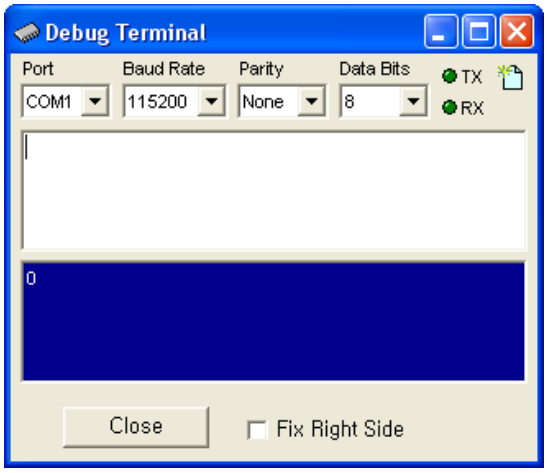
```
Const Device = CB280
Dim i As Integer
Debug Dec i
```

이 경우, 별도의 초기화를 하지 않았기 때문에 디버그터미널에는 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.



Time( )

Variable = TIME (address)

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

***CB290, CT1721C 모델에서만 0~6 번지 사용이 가능합니다.**

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 CUBLOC 의 전원공급이 중단된 상태에서도 계속해서 시간정보는 갱신됩니다. 따라서 전원 ON, OFF 상황과 관계없이 시간에 정보를 얻을 수 있습니다.

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.

<table>
<tr><th>번지</th><th>내용</th><th>범위</th><th colspan="8">비트구성</th></tr>
<tr><td>0</td><td>초</td><td>0~59</td><td></td><td colspan="3">10 자릿수</td><td colspan="4">10 이하 자릿수</td></tr>
<tr><td>1</td><td>분</td><td>0~59</td><td></td><td colspan="3">10 자릿수</td><td colspan="4">10 이하 자릿수</td></tr>
<tr><td>2</td><td>시</td><td>0~23</td><td></td><td></td><td colspan="2">10 자릿수</td><td colspan="4">10 이하 자릿수</td></tr>
<tr><td>3</td><td>일</td><td>01~31</td><td></td><td></td><td colspan="2">10 자릿수</td><td colspan="4">10 이하 자릿수</td></tr>
<tr><td>4</td><td>요일</td><td>0~6</td><td></td><td></td><td></td><td></td><td></td><td colspan="3">10 이하 자릿수</td></tr>
<tr><td>5</td><td>월</td><td>1~12</td><td></td><td></td><td></td><td>10</td><td colspan="4">10 이하 자릿수</td></tr>
<tr><td>6</td><td>년도</td><td>00~99</td><td colspan="4">10 자릿수</td><td colspan="4">10 이하 자릿수</td></tr>
</table>

요일의 경우 다음 표를 참조하십시오.

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

Timeset

TIMESET address, value

```
address : 시간정보가 기억되어 있는 주소 (0~6 까지 사용가능)
value : 시간정보 값
```

***CB290, CT1721C 모델에서만 0~6 번지 사용이 가능합니다.**

RTC 칩에 새로운 정보를 기입하는 명령입니다. 시간을 다시 설정할 때 필요한 명령입니다. 초기 출하상태에서는 알 수 없는 값이 들어있습니다. CB290, CT1721C 모델에서만 본 명령어를 사용하여 RTC 칩 억세스가 가능합니다.

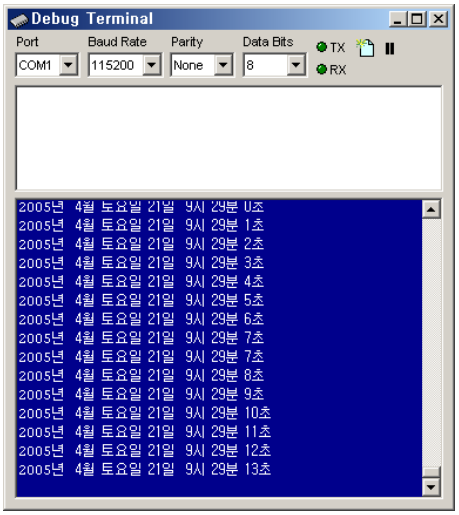
번지	내용	범위	비트구성					
0	초	0~59		10 자릿수			10 이하 자릿수	
1	분	0~59		10 자릿수			10 이하 자릿수	
2	시	0~23			10 자릿수		10 이하 자릿수	
3	일	01~31			10 자릿수		10 이하 자릿수	
4	요일	0~6						10 이하 자릿수
5	월	1~12				10	10 이하 자릿수	
6	년도	00~99	10 자릿수				10 이하 자릿수	

다음은 Timeset 과 Time ( ) 명령을 사용해서 새로운 시간 정보를 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 +/- 수십초 또는 그 이상의 오차가 발생할 수 있습니다. 따라서 주기적으로 현재 시간으로 맞추어주는 것이 좋습니다.

현재시간과 100% 일치를 원하신다면 GPS 수신모듈을 추가해서 표준시를 읽어오는등 다른 방법을 강구해야 합니다. RTC 는 크리스털에 의존하고, 크리스털은 100% 정확성을 보장하지 않으므로, 외부 동기 없이 정확한 시간을 맞추는 것은 현실적으로 불가능합니다.

RTCread( )

Variable = RTCread(address)

```
Variable : 결과를 저장할 변수 (바이트형)
Address : RTC 의 번지
```

***CB405RT 와 MSB764 에서만 사용가능합니다.**

RTC 칩에서 데이터를 읽어오는 함수입니다.

주소	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	기능 및 범위
0	0	10 단위 초			10 이하 초				초 (0~59)
1	0	10 단위 분			10 이하 분				분 (0~59)
2	0	12/24	AM/PM	10 시간	10 이하 시간				시간 (0~23)
3	0	0	0	0	0	요일			1~7 (1=일요일, 2=월, 7=토)
4	0	0	10 날짜		10 이하 날짜				날짜 (1~31)
5	세기	0	0	10 월	10 이하 월				월 (1~12)
6	10 연도				10 이하 연도				연도 0 ~ 99

0 번지에는 초 데이터가 저장됩니다. 상위 4 비트에는 10 보다 큰 단위의 초가 저장되고, 하위 4 비트에는 10 보다 작은 단위의 초가 저장됩니다.

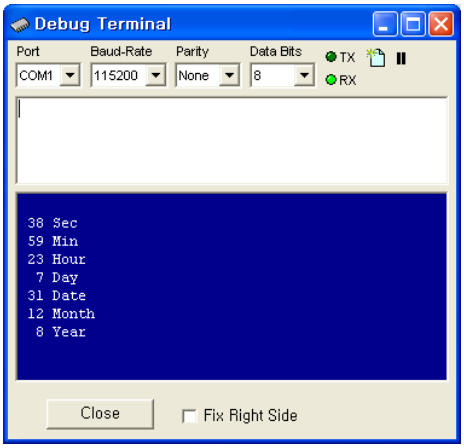
따라서 읽어온 값을 그대로 사용하면, 엉뚱한 값이 되고 맙니다. 11 초의 경우에는 00010001 로 저장되어 있기 때문입니다. 이것을 십진수로 표시한다면, 17 초로 표시됩니다.

따라서 RTCread 로 읽은 값을 표시할 경우에는 16 진 형으로 표시해야 제대로 된 값이 표시됩니다.

```
A = RTCread(0)
Debug Hex2 A,Cr
```

```
#include "CB405RT"        ' ← Const Device 문장대신 #include 를 써줍니다.
Dim i As Integer

Wait 100
RTCwrite 0,&h20      ' Sec
RTCwrite 1,&h59      ' Min
RTCwrite 2,&h23      ' Hour 24h
RTCwrite 3,&h7       ' day 1-7, 1=Sun, 2=Mon, 3=Tue, 4=Wed, 5=Thu, 6=FRI, 7=SAT
RTCwrite 4,&h31      ' Date
RTCwrite 5,&h12      ' Month
RTCwrite 6,&h08      ' Year

Do
   i = RTCread(0)
   Debug Goxy,1,1,Hex2 i, " Sec"
   i = RTCread(1)
   Debug Goxy,1,2,Hex2 i, " Min"
   i = RTCread(2) And &h3f
   Debug Goxy,1,3,Hex2 i, " Hour"
   i = RTCread(3)
   Debug Goxy,1,4,Hex2 i, " Day"
   i = RTCread(4)
   Debug Goxy,1,5,Hex2 i, " Date"
   i = RTCread(5)
   Debug Goxy,1,6,Hex2 i, " Month"
   i = RTCread(6)
   Debug Goxy,1,7,Hex2 i, " Year"
   Wait 500
Loop
```

이 예제 프로그램을 실행화면 다음과 같은 Debug 창이 표시됩니다.



RTCwrite

RTCwrite address, data

```
address : RTC 칩의 어드레스
data : 저장할 정수형 변수/상수
```

***CB405RT 와 MSB764 에서만 사용가능합니다.**

RTC 칩의 지정한 어드레스 (address)에 새로운값 (data)를 써넣습니다.

TIPS

다른 RTC 칩이 장시간 동작했을 때, 시간이 틀어지는 주요원인은 바로 온도 때문입니다. 상온에서는 제대로 동작하던 RTC 칩들도 저온이나 고온이 되면, 크리스털의 발진이 느려지거나 빨라지기 때문에 시간이 틀리는 결과가 나오게 됩니다.

DS3231 는 이러한 단점을 보완한 RTC 칩입니다. 내부에 서미스터(온도센서)를 가지고 있으며, 32KHz 크리스털도 내장하고 있습니다. 그리고 온도센서에서 읽어온값으로 발진주파수를 적당하게 조절해줍니다. 결과적으로 온도차에 따른 오차를 줄여주어, 보다 정확한 시간을 유지시켜줄 수 있는 것입니다.

하지만 100%정확성을 보장하지는 않으므로, 최소한의 관리는 필요합니다. 이점 주의하시기 바랍니다.

시스템 클록을 사용하는 실시간정보

RTC 칩이 없는 모델에서는 TIME, TIMESET 명령어로, 아래 표와 같이 10 번지 이후를 사용합니다.
시스템 클록은 RTC 칩보다 정확하지 않으며 오차가 있을 수 있습니다. 그리고 밧데리백업이 불가능하여, 전원이 없을 때 시간이 자동으로 증가되지 않습니다.

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 시스템 클록 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.

Freqout

FREQOUT Channel, FreqValue

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

이 명령은 특정 주파수의 펄스를 지속적으로 출력하도록 하는 명령으로, CUBLOC 의 PWM 채널을 사용하는 명령입니다. 해당 채널에 스피커를 연결한 뒤 이 명령을 사용해서 음을 발생시킬 수도 있습니다. 이 명령으로 발생시킨 펄스는 CUBLOC 의 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

원하는 주파수가 200KHz 미만일때 다음 공식에 의해 주파수를 계산할 수 있습니다.

```
주기값(FreqValue) = (2304000 / 주파수) - 1
```

Freqout 명령을 사용하기 전에, 해당 포트를 Output 상태로 만들어 주어야 합니다. 펄스 출력을 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.

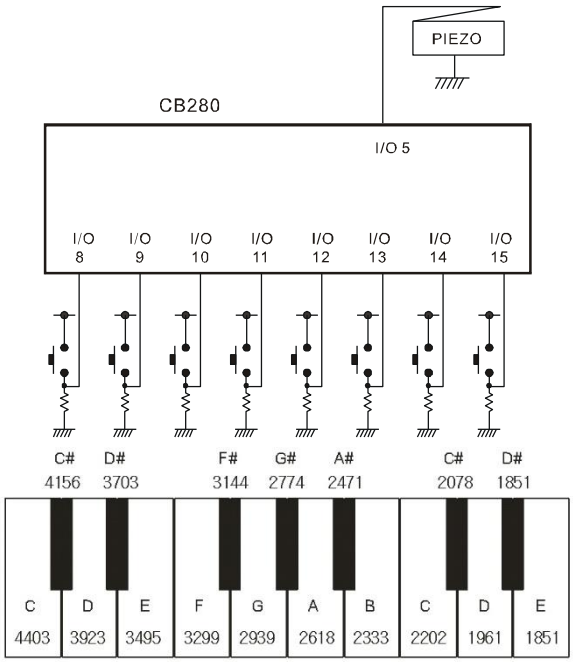
PWM 채널 0 과 채널 3 에서 각각 다른 주파수를 발생시키는 것은 가능합니다.PWM 채널 0,1,2 를 사용하면 고속카운터 채널 0 을 사용할 수 없습니다. PWM 채널 3,4,5 를 사용하면 고속카운터 채널 1 을 사용할 수 없습니다.

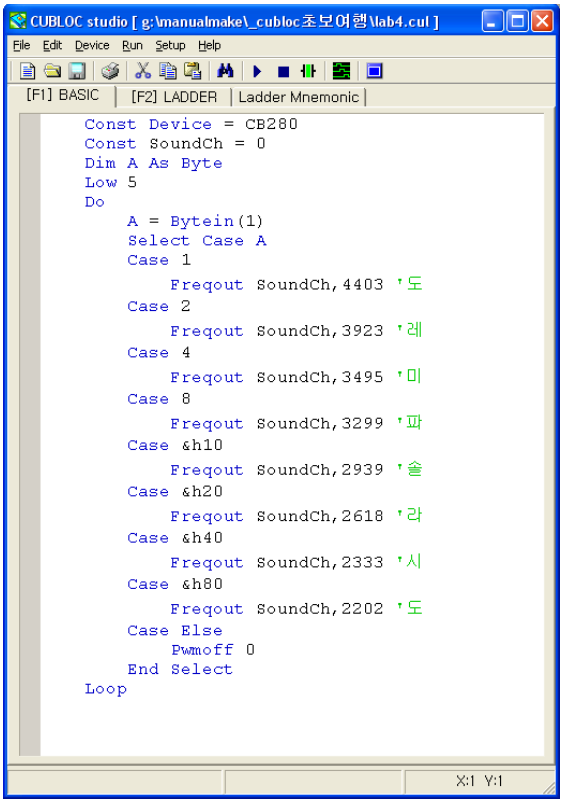
정리하면, CUBLOC 에서 Freqout 은 두 개의 다른 주파수를 발생시킬 수 있으며, Freqout 명령을 사용할 때 PWM 명령사용은 제한 받게 됩니다. 다음은 음계에 따른 주파수를 Freqout 에서 사용할 수 있도록 변환한 표입니다. Freqout 명령으로 음을 연주하고자 할 때 아래 표를 참고하시기 바랍니다.

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

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.






```
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
```

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 (20 ~ 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 CB320, 380	1 개	0	채널 0 은 3, 4, 5
CB400, CB405	2 개	0 또는 1	채널 0 은 3, 4, 5 채널 1 은 6, 7, 8

이 명령은 CUBLOC 내부에 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. 20 을 입력하면 초당 20 개의 펄스가 나가는 것이고, 1000 을 입력하면 초당 1000 개의 펄스가 나가는 것으로 즉, 1KHz 의 펄스가 출력됩니다. 20 부터 15000 까지 입력 가능합니다.

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.

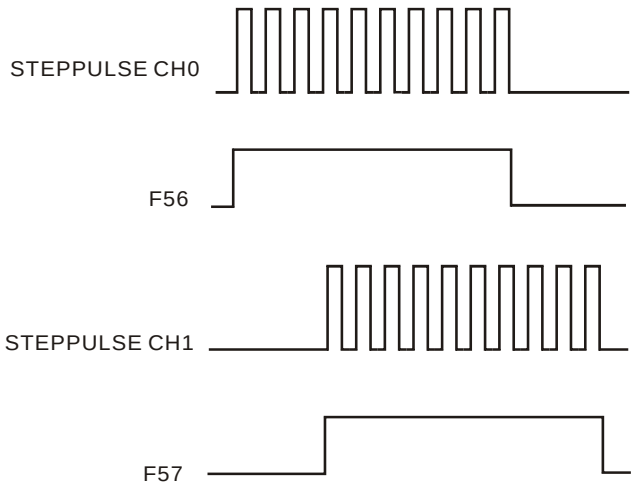
Stepstat( )

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 멈춘뒤에는 자동적으로 0 이 됩니다.

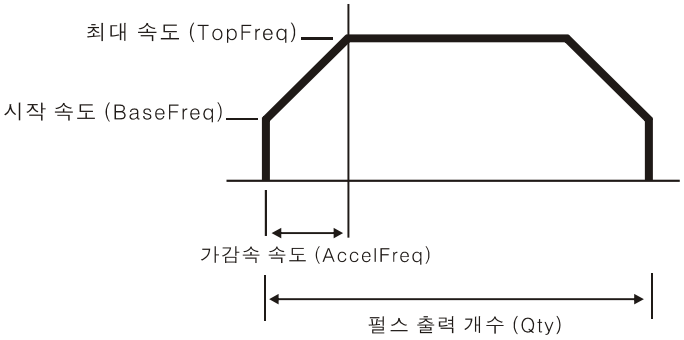


Stepaccel

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 (20 ~ 3300, 최대주파수보다는 작은값으로)
TopFreq : 최대주파수 (20 ~ 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.



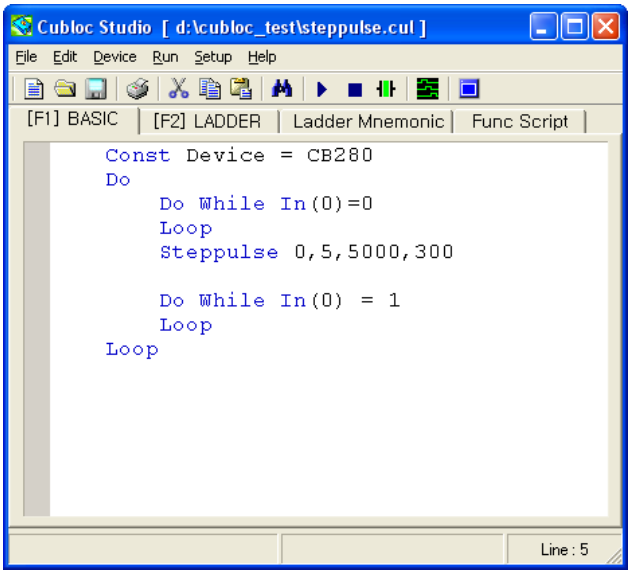
Stepaccel 명령어는 가감속 패턴형태로 파형을 발생시키는 명령입니다. 주로 스탭모터 구동용 펄스를 만들때 사용합니다. 이 명령은 모든 CUBLOC 에서 채널 0 만 사용가능합니다. 이 명령사용시 PWM3,4,5 을 사용할 수 없습니다. 채널은 1 개만 사용가능하지만 출력포트를 바꿀수 있기 때문에, 여러개의 모터를 연결할 수 있습니다.

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.

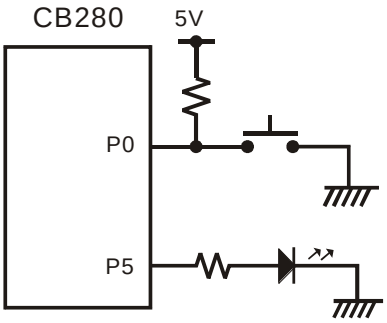
```
Const Device = CT1720
Do
   Wait 200
   Stepaccel 0,8,10,3300,4400,30000
   Do While Stepstat(0) > 0
   Loop
   Wait 1500
Loop
```

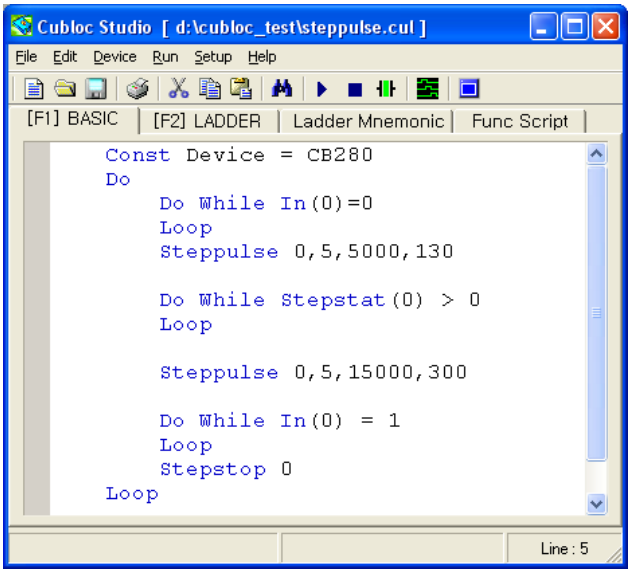
DEMO PROGRAM



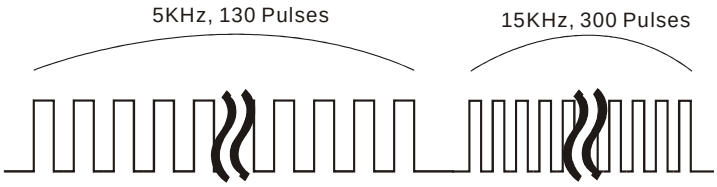
포트 0에 연결된 스위치를 누르면, 포트 5 번에 5KHz 의 속도로 300 개의 펄스를 출력합니다.

회로는 다음과 같습니다. 포트 5 번에는 편의상 LED 를 연결해 놓았습니다.

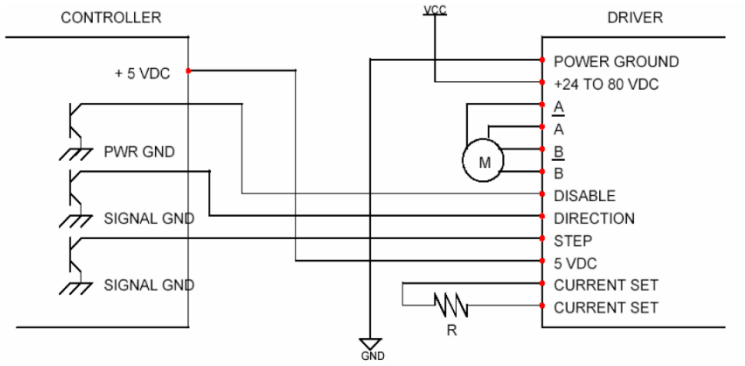




포트 0 에 연결된 스위치를 누르면, 포트 5 번에 우선 5KHz 의 속도로 130 개의 펄스가 출력되고, 이어서 15KHz 의 속도로 300 개의 펄스가 출력됩니다.



아래 그림과 같은 STEP 모터와 “모터 드라이버”를 연결하여 모터를 구동시킬 수 있습니다.



CUBLOC 과 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.

Wait

Wait value

밀리 초단위로 시간지연시키는 명령어입니다. 단, 10 밀리 초간격으로 인식하고 동작합니다. 즉 Wait 15 라고 하면 15 밀리초를 딜레이하지 않고, 10 밀리초를 딜레이합니다.

최대 딜레이 가능시간은 (LONG 형 변수까지 사용가능) 10 부터 2147483640 밀리초입니다.

```
Wait 10        ' 10 밀리 초를 딜레이 합니다.
Wait 200       ' 200 밀리 초를 딜레이 합니다.
Wait 2000      ' 2 초를 딜레이 합니다.
```

*Wait 는 정확한 시간을 보장하지 않습니다.

DELAY 명령은 실행시 서브루틴 처리상황에 따라 속도차이가 있을수 있으므로, 가능한한 Wait 명령을 사용하시기 바랍니다.

```
Delay 200      ' 200 밀리초보다 더 걸릴수 있습니다.
Wait 200       ' 200 밀리초에 근접한 시간으로 딜레이 합니다.
```

제 6 장
CUBLOC BASIC
통신관련기능

RS232 통신

CUBLOC 에는 최대 3 채널의 RS232C 가 내장되어 있습니다.

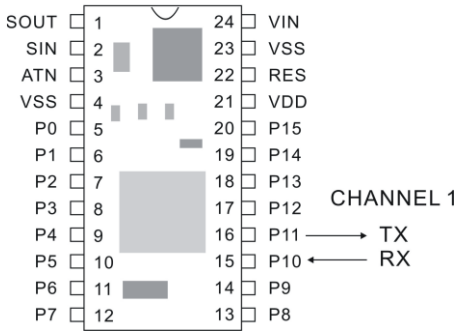
	모델명	사용 가능한 RS232 채널	채널수
코어모듈	CB210, CB220, CB280, CB320, CB380, CB290	1	1 개
	CB400, CB405, CB405RT	1, 2, 3	3 개
CUSB 시리즈	CUSB 전 제품	1	1 개
MSB 시리즈	MSB 전 제품	1, (2)	최대 2 개
CUTOUCH	CT1721C	1	1 개

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.

CB220, CB320 코어모듈에서의 RS232C 사용

CB220, CB320 모델의 경우 포트 10,11 을 RS232 C 채널 1 으로 사용할 수 있도록 되어 있습니다. 따라서 RS232 채널 1 을 사용할 때에는 포트 10,11 번을 사용할 수 없습니다. 이 포트에서는 5V 레벨의 RS232 신호를 입/출력합니다. (5V 의 RS232 신호를 편의상 TTL232 라고 부릅니다.)

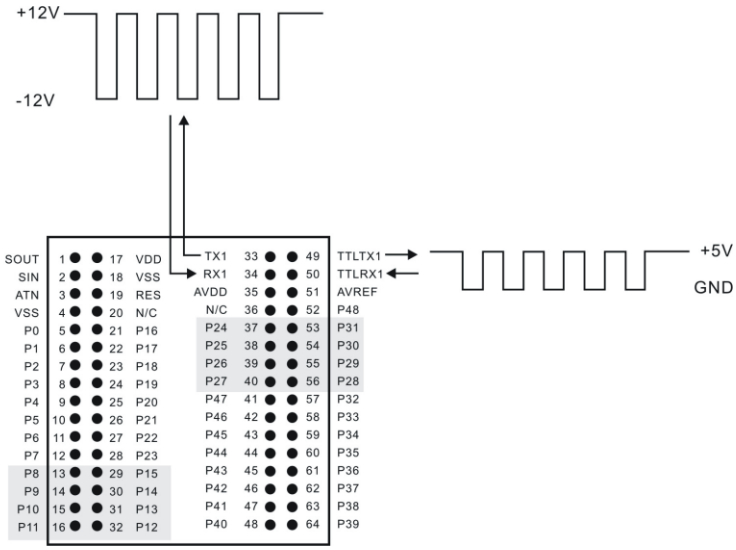
OPENCOM 명령 실행 시, 해당포트의 입출력 상태가 결정되므로 유저가 I/O 포트의 입출력 상태를 따로 정의해 주어야 할 필요가 없습니다.



CB280, CB380, CB290 코어모듈에서의 RS232 사용

CB280, CB290 모델의 경우 별도의 RS232C 채널 1 이 있습니다. 따라서 RS232 채널 1 을 사용하기 위해 사용을 포기해야 하는 I/O 는 없습니다. 그리고 채널 1 에 2 종류의 TX, RX 가 있습니다.
TTLRX, TTLTX 는 5V 레벨의 신호가 나오기 때문에, RS422 이나 RS485 와 같은 다른 통신방식을 사용하기 위한 회로를 추가할 수 있습니다.

RX,TX 는 RS232 의 +/- 12V 신호를 취급하기 때문에, RS232 통신을 쓰는 경우라면 별도의 회로 없이 RX, TX 에 연결해주는 것만으로도 사용할 수 있습니다. 이때에는 반드시 TTLRX, TTLTX 를 아무것도 연결하지 않은 상태로 놔두어야 합니다.



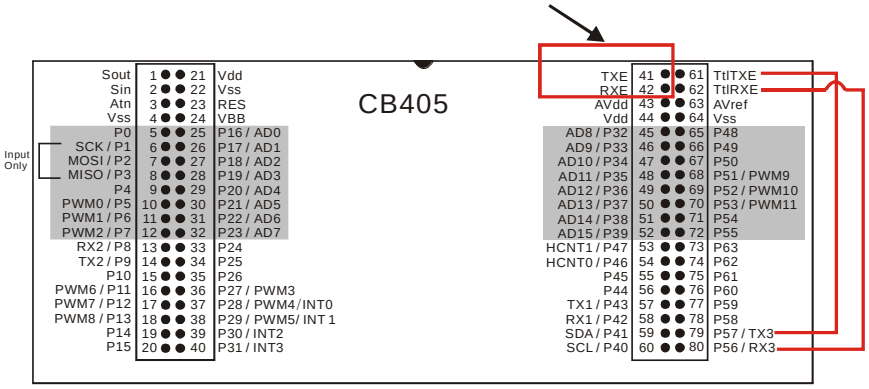
CB400/ 405 / 405RT 코어모듈에서의 RS232 사용

CB400 /405 /405RT 의 경우에는 일반 I/O 포트에 RS232 신호선이 같이 연결되어 있습니다. 이 포트는 모두 5V 레벨의 RS232 신호를 입/출력합니다.

채널	I/O 포트번호	RS232 기능
1	P42	RX
	P43	TX
2	P8	RX
	P9	TX
3	P56	RX
	P57	TX

그리고 CB400/405 펙케지 안에, 5V 레벨의 TTL232 에서 +/-12V 레벨의 RS232 로 신호를 변경해주는 MAX232 회로가 들어있습니다 .이것을 이용하려면, 다음 그림처럼 RX,TX 선을 TTXRXE, TTLTXE 로 연결해 주어야 합니다. 다음 그림은 채널 3 의 RX, TX 선을 연결한 경우입니다.

TXE, RXE 에 +/-12V 의 RS232 신호를 바로 연결할 수 있습니다.



CUSB 에서의 RS232 사용

CUSB 시리즈는 +/-12V 레벨의 RS232 콘넥터가 1 채널 내장되어 있으므로, 유저가 추가로 회로를 구성할 필요가 없습니다.

MSB 에서의 RS232 사용

MSB 시리즈는 +/-12V 레벨의 RS232 콘넥터가 1 채널 또는 2 채널 내장되어 있으므로, 유저가 추가로 회로를 구성할 필요가 없습니다.

CUTOUCH 에서의 RS232 사용

CT1721C 는 +/-12V 레벨의 RS232 콘넥터가 1 채널 내장되어 있으며, RS485 로도 전환하여 사용하실 수 있습니다.

이에대한 자세한 설명은 CUTOUCH 사용 설명서를 참조하시기 바랍니다.

Opencom

OPENCOM channel, baudrate, protocol, recvsize, sendsize

```
channel : 사용채널 (1 부터 3)
Baudrate : 보레이트 (변수 또는 상수) 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200 중 하나
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. 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

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 (1 부터 3)
Baudrate : 보레이트 (변수 또는 상수)
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, port

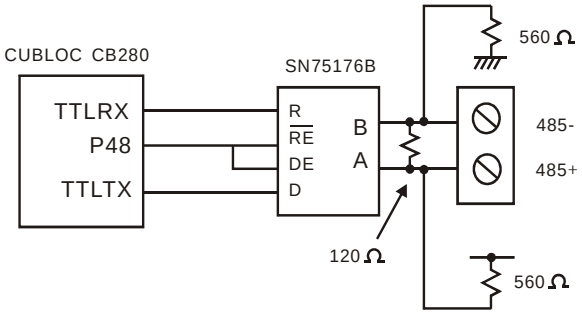
```
channel : 사용채널 (1 부터 3)
port : Transmit Enable 으로 사용할 i/o 포트 번호
```

*** MSB 시리즈에서 사용할 수 없습니다.**

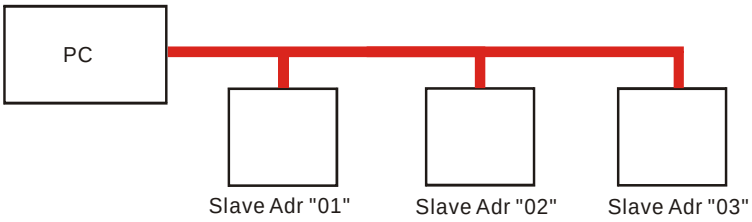
본 명령어는 코어모듈을 사용해서 RS485 회로를 구성할 때, 사용할 수 있는 명령어입니다.

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.

TTL232 신호를 RS485 신호로 변환하기 위해서 다음과 같은 회로를 사용합니다.

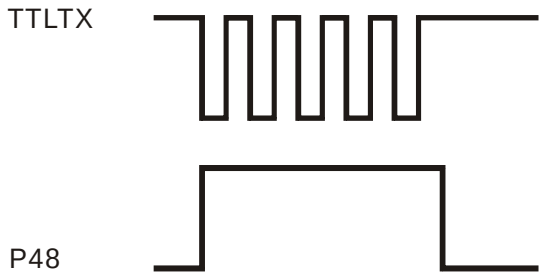


RS485 회로에는 "송신허가"제어선이 필요합니다. RS485 신호선은 여러 개의 디바이스가 공유하고 있습니다. 이중 하나의 디바이스만 송신상태이고 나머지는 수신상태입니다. 아래 그림을 보십시오. 만약 PC 에서 SlaveAdr "01" 디바이스에게 Read 명령을 내렸을 경우, Slave Adr "01" 디바이스에서는 읽어온 값을 PC 로 보내야 합니다. 이를 위해서는 CUBLOC 측에서 송신허가를 해서 "RS485 신호선"을 점유한 뒤, 데이터를 송신합니다.

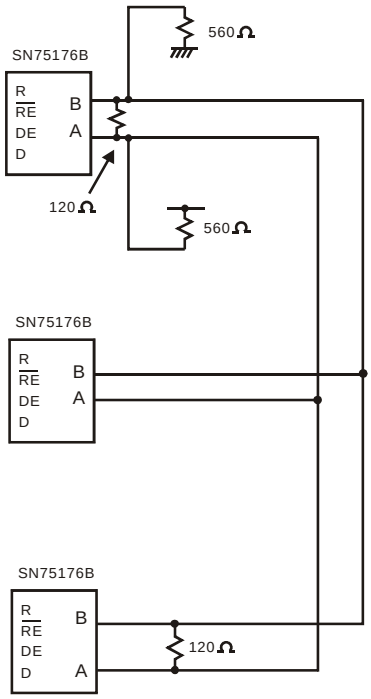


SET RS485 명령을 이용해서 송신허가 포트를 결정합니다. 이 명령실행 이후부터 데이터 송신때마다 송신허가 신호 (ACTIVE HIGH)가 해당 포트로 출력됩니다. CUBLOC 의 OS 가 알아서 자동적으로 HIGH 로 만들어주므로, 유저는 신경쓸 필요가 없습니다. TTLTX 는 STOP 비트를 포함한 모든 출력을 끝낼때까지 HIGH 를 유지합니다.
이 기능이 없다면 유저가 일일이 소스프로그램상에서 송신명령어 바로 직전에 T 를 HIGH 로 만들고, 송신이 끝나면 LOW 로 만드는 명령어를 써주어야 합니다만, SET RS485 명령만 내리면, 이러한 번거로움없이 RS485 통신을 사용하실 수 있습니다.

```
SET RS485 1,48
```



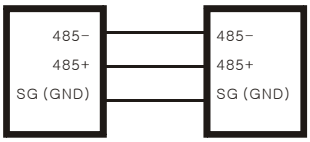
송신 허가 포트는 평상시 I/O 포트로 사용할 수 없습니다.



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 (1 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 여유공간을 사전에 확인한 후 PUT 명령을 사용하는 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 (1 부터 3)
Data : 문자열 (한번에 최대 255 바이트까지 송신가능)
```

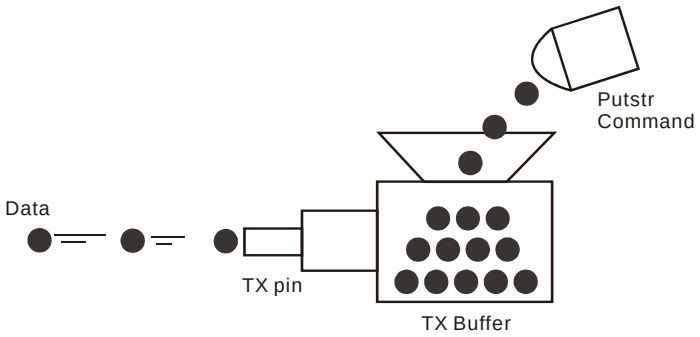
RS232 포트에 문자열 데이터를 송신하는 명령입니다.

```
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. CUBLOC OS 에서 송신버퍼에 들어있는 데이터를 하나씩 TX 로 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 (1 부터 3)
ArrayName : 송신데이터가 들어있는 배열 명 (반드시 바이트형) 상수배열은 사용못함
Bytelength : 송신할 바이트 수 (한번에 최대 65535 바이트)
```

바이트형 배열의 내용을 일괄 전송할 수 있는 전송명령입니다. ArrayName 에 배열 명을 써주고, ByteLength 에 보낼 바이트 수를 써줍니다. 이때 보낼 바이트 수는 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 A 의 요소 중  앞에서부터 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 (1 부터 3)
ArrayName : 송신데이터가 들어있는 배열 명 (반드시 바이트형) 상수배열은 사용못함
Bytelength : 송신할 바이트 수  (한번에 최대 65535 바이트)
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 (1 부터 3)
length : 수신할 데이터 수 (최대 4 까지 사용가능)
```

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 1로 적어줍니다. INTEGER 형에 저장할 때에는 2 로 적어줍니다. LONG 이나 SINGLE 형에 저장할 때에는 4로 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.

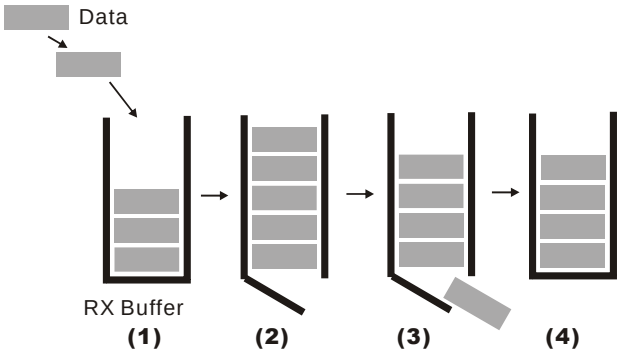
TIPS

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.
```

CUBLOC 에서의 RS232 수신 동작

다음은 CUBLOC 의 RS232 데이터 수신 데이터의 흐름구조를 알기 쉽게 표현한 그림입니다. 수신된 데이터는 버퍼에 저장됩니다. 이 과정은 CUBLOC 의 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 CUBLOC 에 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, 특정 종료코드를 포함한 데이터가 수신되었을 때 인터럽트를 발생시킬 수 있습니다. 그렇게 함으로써, 인터럽트가 너무 빈번히 실행되는 것을 막을 수 있습니다.

CUBLOC 에서는 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 올때까지 기다리게 된다면, GET 명령어에서 프로그램이 잠시동안 정지되거나, 영원히 정지될 수도 있기 때문입니다. 이것은 여러분이 만든 시스템이 멀펑션 (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 (1 부터 3)
length : 수신할 데이터 수
```

수신 버퍼에 있는 데이터에서 지정된 바이트수만큼의 데이터를 읽어와 문자열 변수에 저장하는 명령입니다.
수신버퍼에 들어있는 데이터수가 length 보다 적은경우에는 읽어올 수 있는 데이터까지 읽어온상태에서 명령수행을 종료합니다. 명령수행뒤 SYS(1)에는 실제로 수신된 바이트수가 들어 있습니다.

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 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 (1 부터 3)
length : 수신할 데이터 수
UntilChar : 수신종료 캐릭터 Ascii 코드
```

GETSTR 함수와 동일한 동작을 하는 명령어입니다. 단, 읽어오는 데이터중 UntilChar 에 해당되는 Ascii 코드가 발견되면, UntilChar 는 문자열 변수에 저장되지 않습니다. 수신버퍼에 데이터가 남아있어도, 그대로 두고 명령수행을 종료합니다. Length 에서 지정한 숫자까지 버퍼에서 읽어왔는데도 Untilchar 를 발견할 수 없다면, 명령수행을 종료합니다. 명령수행뒤 SYS(1)에는 실제로 수신된 바이트수가 들어 있습니다. Untilchar 의 경우 수신된것으로 판단하여 Sys(1)에는 포함됩니다.

Geta

GETA channel, ArrayName, bytelength

```
channel : 사용채널 (1 부터 3)
ArrayName : 수신데이터를 저장할 배열 명
Bytelength : 수신할 바이트 수
```

바이트형 배열에 수신된 내용을 일괄 저장할 수 있는 수신명령입니다. 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. 이 명령수행뒤 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 (1 부터 3)
ArrayName : 수신데이터를 저장할 배열 명
Bytelength : 수신할 바이트 수
UntilChar : 수신종료 코드
```

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. 예를 들어 수신된 데이터중 코드 10 이 있는 곳까지만 읽어올 수 있습니다. 수신 바이트 수까지 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 를 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 & cutouch 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.

CheckBf( )

Variable = CheckBf(channel)

```
Variable : 결과가 저장될 변수 (문자열, 실수형 변수는 사용할 수 없음)
channel : 사용채널 (1 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
*기타 다른 어드레스 번호는 사용하지 않는 번호입니다.

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 CUBLOC 의 데이터 메모리의 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, CUBLOC 의 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.

수신 버퍼가 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 수신 버퍼를 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 Gosub 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)      ' 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 송신 버퍼를 클리어하고, 2 로 하면 송수신 버퍼를 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 (1 부터 3)
bufferkind : 버퍼종류, 0=수신버퍼, 1=송신버퍼
```

수신버퍼 또는 송신버퍼의 여유분 Byte 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
   Putstr 1, "TECHNOLOGY"
End If
```

버퍼 사이즈를 50 으로 설정했다면 49 개까지 사용할 수 있습니다. 설정한 사이즈보다 하나 작은 사이즈까지 사용 가능합니다.

WaitTx

WAITTX channel

```
channel : RS232 채널 (1 부터 3)
```

송신버퍼가 모두 비워질 때까지 대기하는 명령입니다.

```
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.
```

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 GOSUB 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, charCheck

```
channel : 사용채널 (1 부터 3)
packetlength : 패킷수
untilchar : 종료캐릭터
charCheck : 종료캐릭터 체크여부 (0=체크함, 1=체크안함)
```

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 GOSUB DATARECV_RTN
Set Until 1,99,"S",0
```

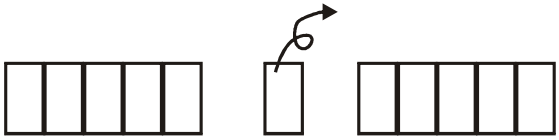
위의 예를 보면, 패킷 사이즈는 99 바이트입니다. 즉, 99 바이트 이내에 종료코드인 “S”를 발견하지 못하면 인터럽트가 발생됩니다.

바이너리 패킷통신

종료캐릭터 없이 패킷수만 가지고 통신하는 경우도 발생합니다. 바이너리값이 수신되는 경우 0 부터 255 사이의 아무값이나 올수 있기 때문에, 이런 경우에는 종료캐릭터 검사기능이 오히려 불편을 초래합니다.

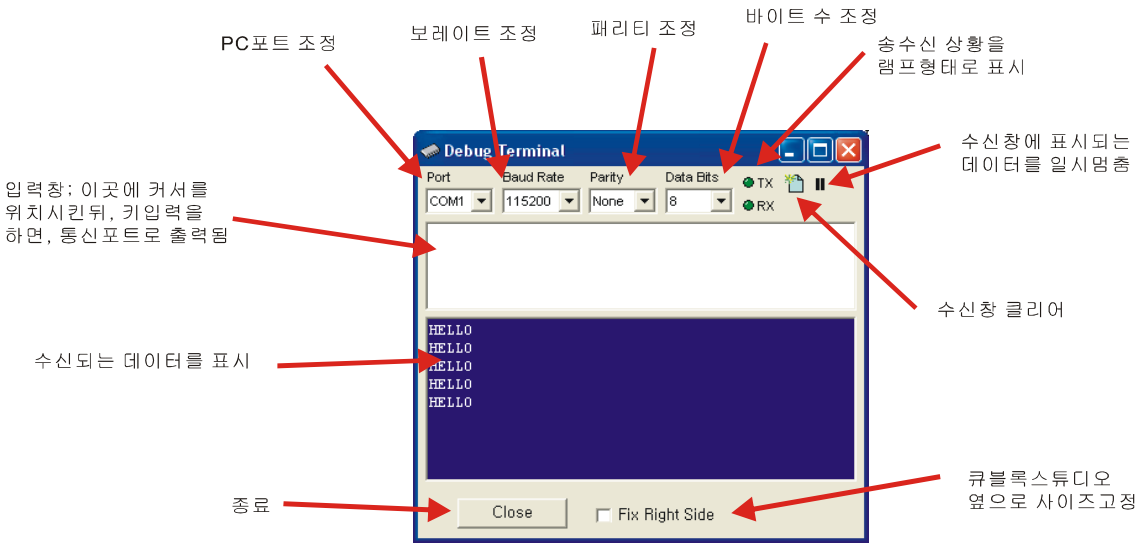
네번째 파라메터인 charCheck 를 1 로 하면, 패킷수만 가지고 인터럽이 발생됩니다. 즉, 종료캐릭터 포함여부를 체크하지 않습니다. (이 기능은 CUBLOC STUDIO V3.1.1 이후부터 지원됩니다.)

주의할 사항은, 연속적으로 오는 데이터만 유효하다는 점입니다. 일반적으로 관련있는 패킷은 연속적으로 전송되기 때문입니다. 만약 5 바이트를 수신하는데 있어서, 1 바이트만 먼저 도착하고 나머지 4 바이트가 나중에 도착하는 경우는 처음 1 바이트를 자동삭제하도록 되어 있습니다. (노이즈나 gabage 로 판단)



디버그 터미널의 기능

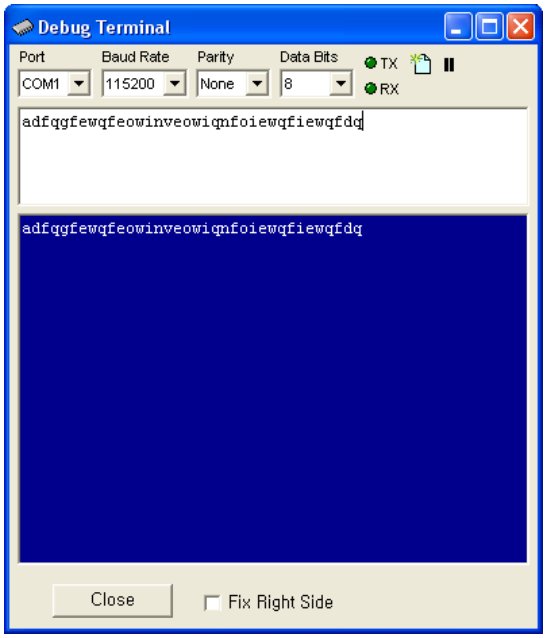
사실,“디버그 터미널”은 하이퍼 터미널과 같은 “PC 통신 프로그램”의 축소판이라고 보시면 됩니다. 간단하게 보레이트와 통신포트를 바꿀 수 있고, PC 의 RS232 포트로부터 들어오는 내용을 표시하고, 입력창에 입력하는 내용이 PC 의 RS232 로 출력됩니다. 따라서 “디버그 터미널”을 RS232 통신 테스트용으로 충분히 사용하실 수 있습니다.



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.

MODBUS 통신

CUBLOC 에서는 MODBUS 를 지원하고 있습니다. MODBUS 는 RS232 채널 1 을 사용하여 연결하며, 모든 모델이 **RTU 모드를 지원합니다. 일부 모델에서는 ASCII 모드를 지원하지 않습니다.** MODBUS ASCII 모드를 지원하지 않는 모델은 CB400, CB405, CB405RT, MSB 시리즈 입니다.

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, CUBLOC 에서 MODBUS 프레임을 수신한후, 일정시간을 대기하는 것을 말합니다. 너무 빨리 응답을하게되면, 마스터측에서 데이터를 미쳐 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(10진)	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 어드레스 군이 존재할 수 있습니다. CUBLOC 에서는 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.

CUBLOC 의 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 CUBLOC 의 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	
5000H		5000H	T 영역
6000H		6000H	C 영역
7000H		7000H	D 영역
8000H		8000H	WP 영역
9000H		9000H	WM 영역

비트를 다루는 펑션코드에서는 비트영역 어드레스를 억세스하고, 워드를 다루는 펑션코드에서는 워드영역을 억세스 합니다. 빈칸으로 되어 있는 부분은 사용하지 않는 어드레스입니다.

HMI 나 SCADA 사용시 어드레스

다음은 HMI / SCADA 소프트웨어등에서 사용하고 있는 어드레스 체계입니다.

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

이 경우, HMI / SCADA 쪽에 어드레스 입력시, 다음표를 참고하시여 입력하시기 바랍니다.

워드영역 (Holding/Input Registers)	
관련평션코드 : 3,4,6,16	
접근하고자 하는 CUBLOC 측 영역	HMI / SCADA 에 입력해야 될 어드레스
D 영역 (D0 ~ D511)	40001 ~ 40512
T 영역 (T0 ~ T255)	41001 ~ 41256
C 영역 (C 0~ C255)	42001 ~ 42256
WM 영역 (WM0 ~ WM255)	43001 ~ 43256

비트영역 (Coil, Input Status)	
관련 펑션코드 : 1,2,4,15	
접근하고자 하는 CUBLOC 측 영역	HMI / SCADA 에 입력해야 될 어드레스
P 영역 (P0 ~ P127)	1 to 128 또는 10001 to 10128
M 영역 (M0 ~ M2047)	4097 to 6144 또는 14097 to 16144

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 바이트 수*	*ASCII 방식*	*ASCII 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	3A	30	33	30	31	30	30	31	33	30	30	32	35	43	34	0D	0A

이에 대한 응답은 아래와 같습니다.

Response:

필드명	*RTU 방식*	*RTU 바이트 수*	*ASCII 방식*	*ASCII 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	0X00	1	0 0	2
시작어드레스 LO	0X01	1	0 1	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	0X00	1	0 0	2
시작어드레스 LO	0X01	1	0 1	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	7 0	2
시작어드레스 LO	0X01	1	0 1	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	7 0	2
시작어드레스 LO	0X01	1	0 1	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

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. 예를들어 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 바이트단위로 불러옵니다. 예를들어 Getcrc B,_D,5 라고 명령하였을 경우, _D(0) 부터 저장된 5 바이트를 CRC 계산합니다.

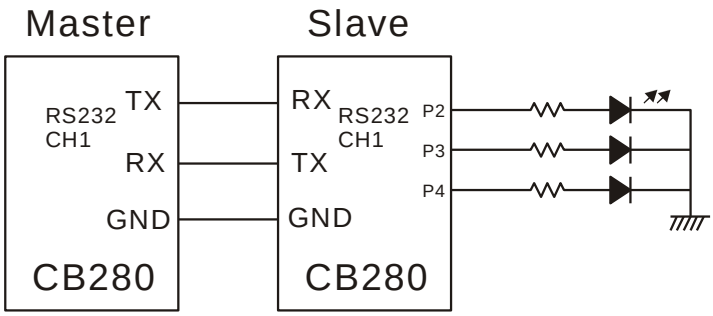
_D 배열은 레더의 D 영역을 저장하는 Word (16 비트)형 배열이므로, 주의해야 합니다. _D(0)부터 _D(4)까지 계산하는 것이 아니라, _D(0)에 있는 16 비트값과 _D(1)에 있는 16 비트값, 그리고 _D(2)에 있는 하위 8 비트를 가지고 CRC 계산을 수행합니다.

MODBUS 마스터 모드 구현 (RTU 모드)

다음은 MODBUS RTU 모드로 마스터 통신을 구현한 예제 프로그램입니다. 이 라이브러리는 송신만 처리하고 있습니다.

```
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 7 장
CUBLOC
디스플레이
라이브러리

CUBLOC 에는 CLCD, GHLCD 등 컴파일에서 시판중인 LCD 제품군과 연결하여 디스플레이를 할 수 있습니다. GHLCD 를 사용하여 선,원,박스 등을 그리거나, 텍스트를 표시할 수 있으며, 이를 위한 전용명령어가 준비되어 있습니다. LCD 관련 명령을 사용하기 위해서는 사용하고 계신 모델에 장착된 LCD 에 대하여 기본적인 사항을 알아야 할 필요가 있습니다.

영문 lcd 제품군 - CLCD

CLCD 는 단순한 영문자/숫자 등을 표시할 수 있는 캐릭터 LCD 입니다. LCD 뒷면에 시리얼 데이터 를 받아서 LCD 를 제어해주는 콘트롤보드가 있는 것을 볼 수 있습니다.

CLCD 는 I2C 또는 RS232C 방식으로 데이터를 받을 수 있는 LCD 모듈입니다.,

한글 lcd 제품군 - HLCD

HLCD 는 한글및 영문자/숫자 등을 표시할 수 있는 캐릭터 LCD 입니다.

Set Display

SET DISPLAY kind, method, baud, buffersize

```
Kind       : 0=ALCD / HLCD, 1=GHLCD GHB3224, 2=CLCD
Method     : 송신방법 0=CuNET, 1=RS232 CH1, 2=RS232 CH2, 3=RS232 CH3
baud       : 보레이트 (또는 CuNet 의 SLAVE 어드레스)
BufferSize : 송신버퍼크기
```

디스플레이를 연결할 포트에 대한 선언명령으로, 프로그램 맨 초기에서 한번만 사용할 수 있는 명령입니다. 모든 디스플레이관련 명령어는 SET DISPLAY 에서 선언된 포트와 통신방법으로 데이터를 송신합니다.

kind 는 디스플레이 할 LCD 의 종류를 선택하는 파라메터입니다. CLCD 를 사용할 때에는 methode 를 0 으로 설정하시기 바랍니다.

Method = 1 (RS232 채널 1 사용) ; ALCD/ HLCD 사용시

CUBLOC 에 내장되어 있는 RS232 채널 1 을 사용하는 모드입니다. ALCD 와 HLCD 에서 사용할 수 있습니다. 사용가능한 포트는 모델 별로 다릅니다. CB220 / CB320 모델의 경우 포트 1 에 연결합니다.

CB280/ CB380 모델의 경우 별도의 RS232C 채널 1 을 위한 포트가 있습니다. 49 번 핀에 연결할 수 있습니다. 49 번 핀은 5V 레벨의 RS232C 신호를 출력합니다.

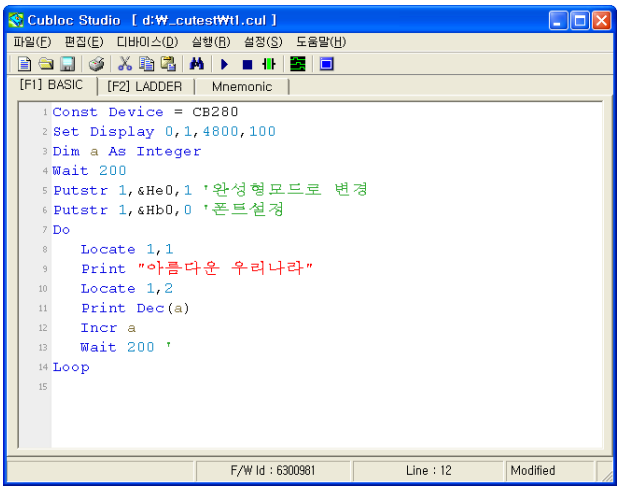
보레이트는 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200 중 하나를 적어줍니다.

송신 버퍼크기는 50~128 사이의 숫자가 적당합니다. 송신 버퍼의 크기가 너무 작으면, 데이터가 모두 표시되지 않는 경우가 발생합니다. 너무 크면, 그만큼의 데이터 메모리를 차지하게 되므로 적당한 크기로 선택하십시오.

```
SET DISPLAY 0,1,19200,50 '19200 보레이트에 송신버퍼는 50 으로..
```

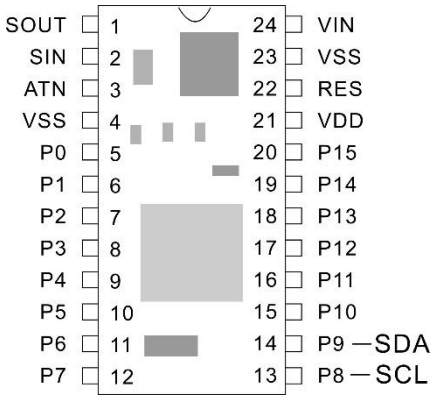
SET DISPLAY 명령은 소스의 첫 부분에서 단 한번만 사용할 수 있는 명령입니다.

다음은 HLCD114 제품에서 한글을 표시하는 예제 프로그램입니다. HLCD114 는 4800 보레이트로 맞추어주십시오. 이 프로그램을 실행시키면 HLCD 화면상에 “아름다운 우리나라”라는 글자와 증가중인 숫자가 표시됩니다.



Method = 0 (CuNET 사용) ; CLCD 사용시

CuNET 은 CUBLOC 에 내장되어 있는 I2C 포트를 이용한 주변장치 연결용 통신 채널입니다. CB220 / CB320 의 경우 I/O 포트 8 (클록)과 포트 9(데이터)를 사용해서 연결합니다. 디바이스 종류에 따라 CuNET 의 포트위치가 다르므로, 핀 레이아웃을 참조하시기 바랍니다.



CuNET 을 지원하는 디스플레이 제품군에서 사용할 수 있으며, CuNET 통신은 보레이트를 사용하지 않습니다. SET DISPLAY 문에서 보레이트 지정대신, SLAVE 어드레스를 지정합니다.

```
SET DISPLAY 2,0,1,50 ‘CLCD, 어드레스는 1 로, 송신버퍼는 50 으로..
```

CUNET 와 I2C

CUNET 은 I2C 와 동일한 방식입니다. LCD 를 위한 전용 I2C 포트를 CUNET 이라고 부르고 있고 LCD 를 위한 전용명령어 (LOCATE, CLCDOUT, PRINT 등)를 지원합니다. CUNET 은 MCU 의 하드웨어 I2C 기능을 사용하고 있습니다.

CUBLOC 의 I2C 관련 명령어 I2CWRITE, I2CREAD 등은 소프트웨어적으로 I2C 통신을 구현한 것입니다.

Cls

CLS
LCD 를 초기화하고, 화면을 클리어합니다. 클리어 명령 실행 후에는 100 밀리 초 정도의 딜레이가 필요합니다.

```
Cls
Wait 100
```

Csron

CSRON
LCD 커서를 표시합니다. (디폴트 상태에서는 커서가 표시됩니다.)

Csroff

CSROFF
LCD 커서를 지웁니다.

Clcdout

CLCDOUT x,y, String/Variable

```
X : LCD 의 X 좌표를 나타내는 변수/상수
Y : LCD 의 Y 좌표를 나타내는 변수/상수
String : 문자열 변수,상수 또는 형식변환자 등
Variable :변수(상수)를 사용하면, 해당 변수(상수)의 값이 그대로 출력됩니다.
```

해당위치에 문자를 표시합니다.

```
CLCDOUT 3,3,"COMFILE",DEC I
```

Locate

LOCATE x,y

```
X : LCD 의 X 좌표를 나타내는 변수/상수
Y : LCD 의 Y 좌표를 나타내는 변수/상수
```

LCD 의 X,Y 좌표를 새로 설정합니다. CLS 명령 실행 후에는 커서위치는 0,0 입니다.

```
LOCATE 1,1        '1,1 위치로 커서이동
PRINT "COMFILE"
```

Print

Print 0x1b, value,value…

Print 명령어는 CLCD 백라이트 On/OFF 용으로 사용할 수 있습니다.

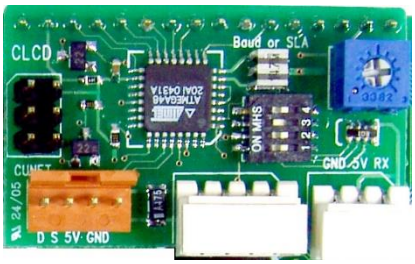
Print 0x1b,0x62 : 백라이트 Off
Print 0x1b,0x42 : 백라이트 On

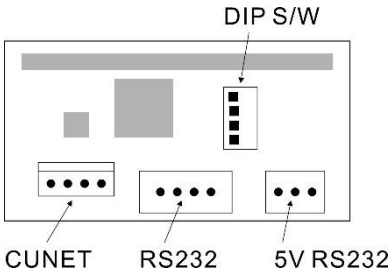
사용예)

```
Dim a As Integer
Set Display 2,0,0,50

a=1234

Do
   Print 0x1b,0x62  '백라이트 off
   Delay 500
   Print 0x1b,0x42 '백라이트 on
   Delay 500
   Clcdout 1,1,Dec a         '숫자 출력
   Incr a
Loop
```

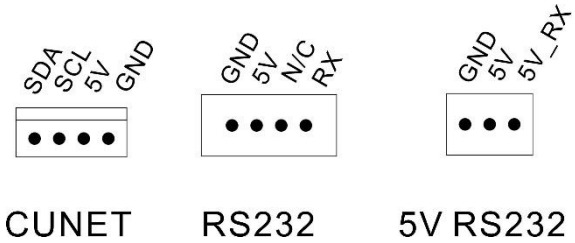
CLCD 모듈

CLCD 모듈의 뒷면에는 콘트롤보드가 부착되어 있습니다. 이 콘트롤 보드가 CUNET 신호를 받아서 LCD 모듈을 구동합니다. .



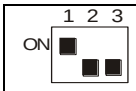
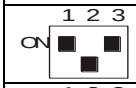
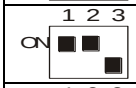
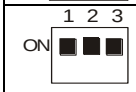


CLCD 는 CUNET 통신 이외에도 RS232 신호를 받을 수 있도록 되어 있습니다. RS232 에는 2 종류의 콘넥터가 있는데, 3 핀 콘넥터 5V 레벨의 RS232 신호를 연결하는 콘넥터입니다. PC 와 직접 연결하고자 할 때에는 +/- 12V 레벨의 RS232 신호를 연결하는 4 핀 콘넥터를 사용합니다.



딥스위치로 보레이트 및 I2C 슬레이브 어드레스를 설정할 수 있습니다. DIP 스위치 중 4 번 스위치는 사용하지 않는 스위치입니다.

DIP 스위치 상태	RS232 보레이트	I2C 의 SLAVE ADDRESS
1 2 3 ON	2400	0
1 2 3 ON	4800	1
1 2 3 ON	9600	2
1 2 3 ON	19200	3

	28800	4
	38400	5
	57600	6
	115200	7

CUNET 과 RS232 는 동시에 동작하도록 되어 있으나, 실제로 사용할 때에는 둘 중 하나만 연결해서 사용하시기 바랍니다. 두 가지 모두 연결되어 있다면, 어느 한쪽이 동작할 때 다른 한쪽은 동작하지 않도록 해야 합니다. 다음은 CLCD 의 명령코드 표입니다.

명령	전송예 (16 진)	바이트 수	수행시간	설명
ESC ' C'	1B 43	2	15mS	화면을 클리어 합니다. 이 명령 수행 후 반드시 15mS 를 딜레이 해주어 야합니다.
ESC 'S'	1B 53	2		커서표시를 ON 합니다.(디폴트)
ESC 's'	1B 73	2		커서표시를 OFF 합니다.
ESC 'B'	1B 42	2		백라이트를 ON 합니다. (디폴트)
ESC 'b'	1B 62	2		백라이트를 OFF 합니다.
ESC 'H'	1B 48	2		LOCATE 0,0 위치로 갑니다.
ESC 'L'	1B 4C xx yy	4	100 uS	커서위치를 변경합니다.
ESC 'D'	1B 44 code 8byte	10		캐릭터코드 8 부터 15 까지 8 개 영역은 유저가 정의할 수 있는 영역입니다. 이 영역에 임의의 비트맵데이터를 전송하는 명령입니다.
1	01	1		1 번째 행의 왼쪽 끝으로 이동
2	02	1		2 번째 행의 왼쪽 끝으로 이동
3	03	1		3 번째 행의 왼쪽 끝으로 이동
4	04	1		4 번째 행의 왼쪽 끝으로 이동

수신된 데이터가 명령코드가 아니면, LCD 화면상에 그대로 표시합니다.

RS232 연결 시, 4 핀 콘넥터를 사용할 때에는 보레이트 38400 이상을 사용할 수 없습니다. 5V RS232 (3 핀 콘넥터)는 115200 보레이트까지 모두 사용할 수 있습니다.

다음은 CB280 모듈에서 CLCD 모듈을 CUNET 으로 접속하여 표시하는 샘플 프로그램입니다. 본 프로그램을 실행시키면 LCD 상에 숫자가 증가되는 상황을 볼 수 있습니다.

```
Const Device = Cb280
Set Display 2,0,1,50  '딥스위치를 조정해서 SLAVE ADDRESS 를 1 로 맞춥니다.
Dim i As Integer
Delay 100             ' 최초 딜레이 (CLCD 모듈이 START UP 하는 시간을 기다려줌)
Cls
Delay 200             ' CLCD 가 클리어 하는데 걸리는 시간을 기다림
Csroff
Locate 5,2
Print "Start!!!"
Delay 500
Cls
Delay 100
Do
   Incr i
   Locate 0,0
   Print "COMFILE"
   Locate 1,3
   Print "CUBLOC ",Dec i
   delay 100
Loop
```

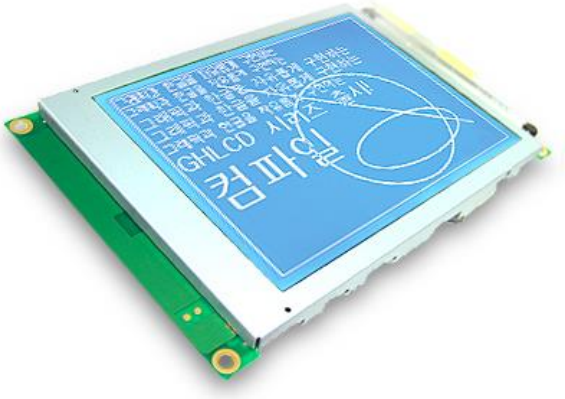
*** CLCD 상의 SLAVE ADDRESS 와 SET DISPLAY 문의 SLAVE ADDRESS 가 반드시 일치해야 합니다.**

다음은 CLCD 의 캐릭터정의 코멘드 사용예입니다.

```
Const Device = CB320
Set Display 2,0,0,80
Wait 100
Cls
Wait 100
Print 0x1b,"D",8,0k,0,0,0,15,15,0,0,0
Print 0x1b,"D",9,0,0x10,0x18,0x1c,0x1c,0x18,0x10,0
Locate 0,0
Print "abc"
Print 8,9,10,11,12,13,14,15
Do
Loop
```

그래픽한글 lcd 제품 GHB3224 라이브러리

GHLCD 는 그래픽과 한글/영숫자 표시가 가능한 제품입니다. CLCD 와 달리 다양한 명령어 체계를 가지고 있으며, 선/원/박스/타원 등의 그래픽 기본 명령과 BMP 파일 저장 및 표시, 화면영역 중 일부분 보관 및 재 표시 기능 등을 다양하게 갖추고 있습니다. CUBLOC 에서는 이러한 GHLCD 를 쉽게 제어할 수 있도록 LINE, BOX, CIRCLE 등의 그래픽 전용코멘드를 지원합니다.



GHLCD 모델 중 GHB3224 에서 사용된 LCD 는 흑백 STN 방식의 LCD 이며 320 x 240 의 해상도를 가지고 있습니다.

GHLCD 관련 라이브러리는 CUTOUCH 사용설명서에서 다루고 있습니다.

http://www.comfile.co.kr/download/cubloc/cutouch_manual.pdf

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. CUBLOC 과 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 포트에 SDA, 8 번 포트에 SCL 단자가 연결되어 있다는 선언문입니다.

만약 9,8 번 I/O 가 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.

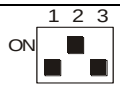
```
csgnput __cssla,2,__csstr_a(2) + &H80
```

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
```

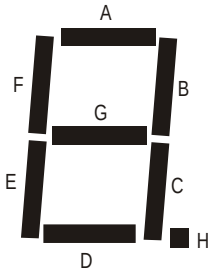
CSG 모듈의 원하는 위치에 원하는 숫자를 표시하는 명령입니다. DATA 의 상위 1 비트는 Dot 를 ON 할 때 사용하는 비트입니다. data 는 ASCII 코드로 써주어야 하며, 숫자 0~9, 영문자 A~F 까지만 표시할 수 있습니다.
이 명령은 부프로그램 라이브러리이며, 소스 앞부분에 SET I2C 명령을 선언해주어야 동작합니다.

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 8 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.

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.

Delay

DELAY value

시간을 지연시켜주는 명령입니다. Value 는 밀리초 단위입니다만, 정확하지는 않습니다. Wait 명령은 비교적 정확한 시간을 딜레이 해줍니다. Delay 명령대신 Wait 명령을 사용하시는 것이 좋습니다.

Keyin

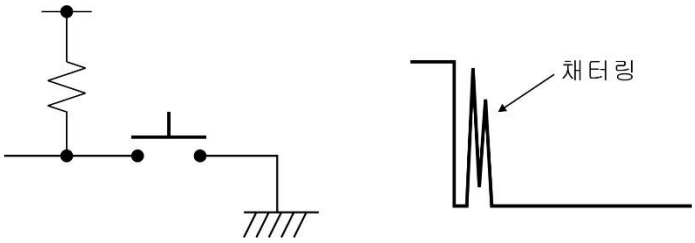
Variable = KEYIN(port, chattime)

```
Variable : 결과가 저장될 변수 (BYTE 형을 리턴)
Port : 입력 사용 가능한 I/O 를 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 밀리초동안 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(port, chattime)

```
Variable : 결과가 저장될 변수 (BYTE 형을 리턴)
Port : 입력 사용 가능한 I/O 를 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.
```

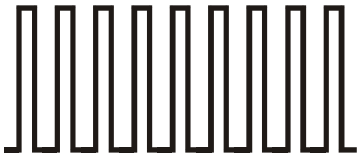
Beep

BEEP Port, Length

```
Port : 출력 가능한 I/O 를 가리키는 변수 또는 상수
Length : 펄스 출력 시간
```

Beep 사운드를 발생시키는 SUB 형 라이브러리입니다. I/O 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
```



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.

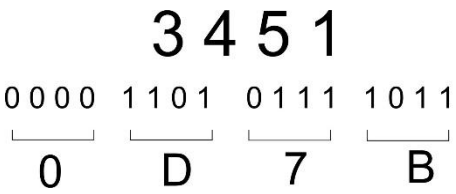
Bin2bcd

Variable = BIN2BCD(binvalue)

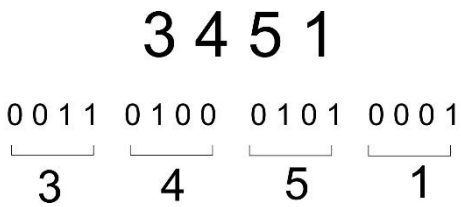
```
Variable : 결과가 저장될 변수 (LONG 형을 리턴)
binvalue : 변환할 바이너리 값
```

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)

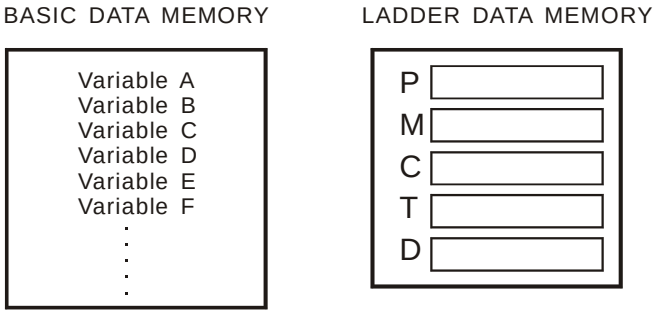
```
Variable : 결과가 저장될 변수 (LONG 형을 리턴)
bcdvalue : 변환할 BCD 값
```

위 명령과는 반대로 BCD 코드를 바이너리 값으로 변환해주는 라이브러리입니다.

제 9 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 _M(0) ~ M(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 워드단위로 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.

D 영역에 저장된 실수데이터를 BASIC 에서 사용하는 방법

이 기능은 CUBLOS STUDIO V4.2 이후 버전에서만 사용할 수 있습니다.

D 영역에 저장된 데이터가 실수형(SINGLE 형)일 경우, 이 내용을 BASIC 에서 불러오면 이상한 숫자가 표시됩니다. 실수로 포맷화된 데이터이기 때문입니다.

_D 대신에 _A 를 사용하면 D 영역에 있는 데이터를 실수형으로 처리해줍니다.

```
#include "MSB6XX"
Opencom 1,115200,3,200,200
Set Modbus 1,1,100

Do
 Delay 1000
 Debug Float _A(1),Cr   '_A(1)은 Single 형 변수로 취급됩니다.
Loop
```

Single 형은 32 비트이기 때문에 D 영역 2 개가 필요합니다.

_A(0)	D0, D1
_A(1)	D2, D3
_A(2)	D4, D5
:	:
_A(40)	D80, D81

따라서 _A(1)은 D2,3 영역에 들어있는 값을 의미합니다.

* 디바이스 종류에 따라서 쓸 수 있는 최대 번지가 다릅니다. D 영역을 100 개까지 쓸 수 있는 디바이스라면 _A 는 0 부터 49 까지 사용할 수 있습니다.

D 영역에 저장된 32 비트 정수데이터를 BASIC 에서 사용하는 방법

이 기능은 CUBLOS STUDIO V4.2 이후 버전에서만 사용할 수 있습니다.

D 영역에 저장된 데이터가 32 비트 정수형(LONG 형)일 경우, _D 대신에 _B 를 사용하면 D 영역에 있는 데이터를 LONG 형으로 처리해줍니다.

```
#include "MSB6XX"
Opencom 1,115200,3,200,200
Set Modbus 1,1,100
Dim L1 As Long
Do
 Delay 1000
 L1 = _B(1)    '_B(1)은 LONG 형 변수로 취급됩니다.
Loop
```

Long 형은 32 비트이기 때문에 D 영역 2 개가 필요합니다.

_B(0)	D0, D1
_B(1)	D2, D3
_B(2)	D4, D5
:	:
_B(40)	D80, D81

따라서 _B(1)은 D2,3 영역에 들어있는 값을 의미합니다.

* 디바이스 종류에 따라서 쓸 수 있는 최대 번지가 다릅니다. D 영역을 100 개까지 쓸 수 있는 디바이스라면 _B 는 0 부터 49 까지 사용할 수 있습니다.

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.
```

부 록

부록: 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

다운로드에러시 문제 해결방법

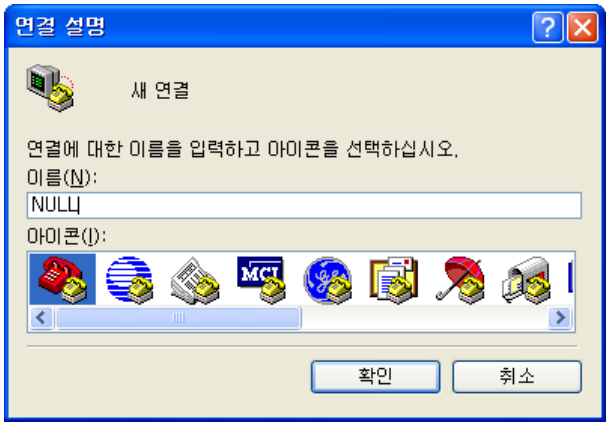
TinyPLC 제품 또는 CUBLOC 제품 사용자들이 가장 많이 전화로 물어보시는 것이 바로 “다운로드가 안된다” , “RS232 통신에러가 난다.”등 입니다.

이런경우 원인은 대부분 접속불량, 세팅잘못, 컴퓨터 이상등에서 발견되었습니다.
우선 다음 사항을 먼저 체크해보십시오.

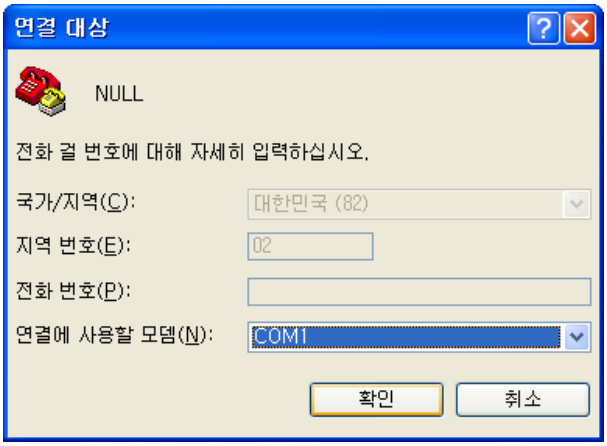
- "설정"메뉴에 "펌웨어 다운로드"먼저 해보시기 바랍니다.
- 다른 응용프로그램을 모두 닫고 해보시기 바랍니다.
- 다운로드포트를 상태를 체크 바랍니다. (접속불량, 엉뚱한 포트에 연결한 경우등)
- 컴퓨터를 재부팅한뒤 해보시기 바랍니다. (엉뚱한 곳에서 COM 포트를 잡고있는 경우)
- 다른컴퓨터에서 해보시기 바랍니다. (해당 컴퓨터의 COM 포트가 고장난경우)
- 그래도 안된다면 아래 적힌 순서대로 차근차근 점검해보시기 바랍니다.

1. PC 의 RS232 포트에 문제가 없는지 체크합니다.

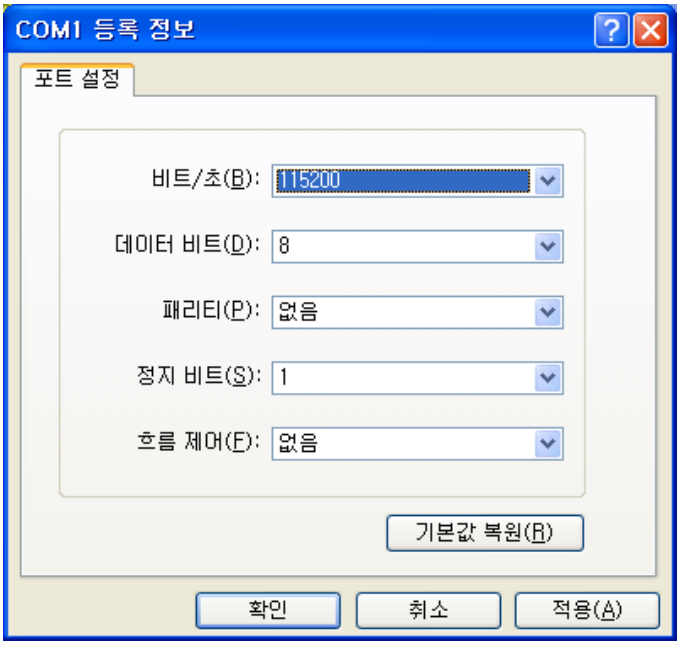
PC 상의 RS232 포트가 이상이 없는지를 먼저 체크합니다. 윈도우테의 경우 [시작] – [모든프로그램] –[보조프로그램]-[통신]에 가시면 “하이퍼 터미널”이라는 메뉴가 있습니다. 실행시키면 아래화면이 표시되는데, 이름은 아무거나 입력하세요. 윈도우 7 이상에서는 별도의 통신 터미널 프로그램을 사용해서 아래과정과 동일하게 테스트 해보실 수 있습니다.



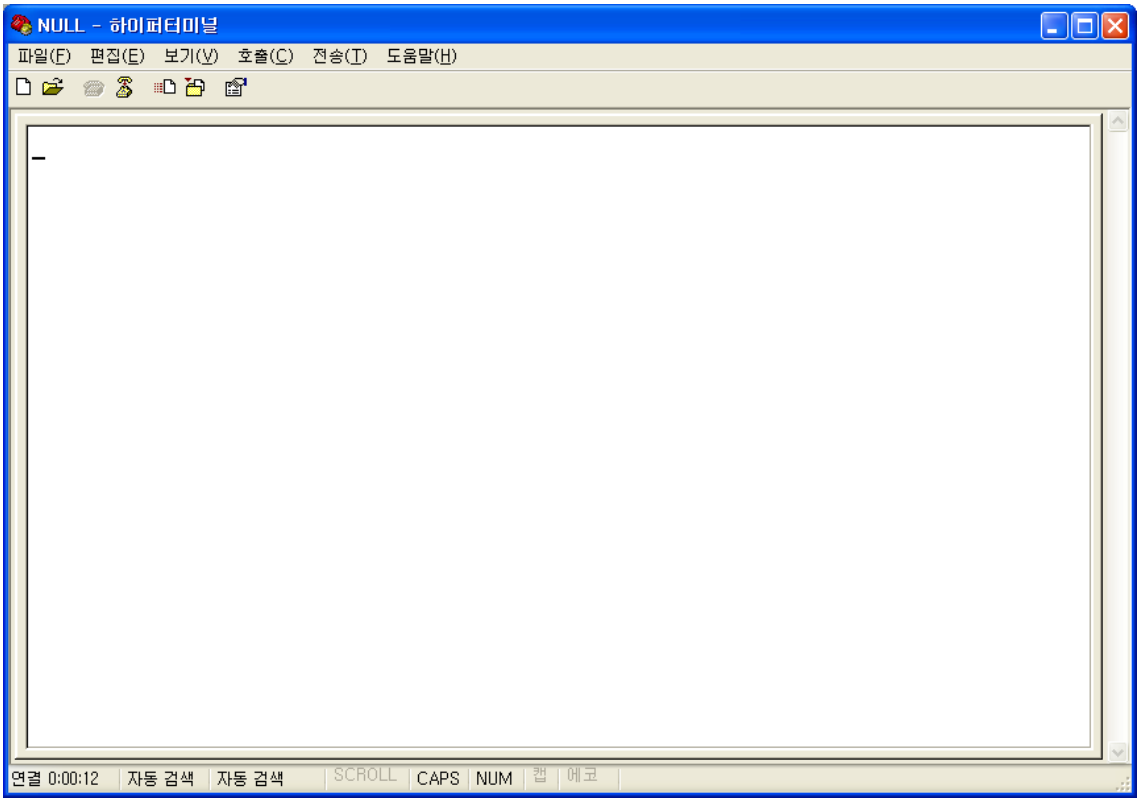
확인을 누르면 “연결에 사용할 모뎀” 을 지정하세요. (보통은 COM1 포트입니다.)



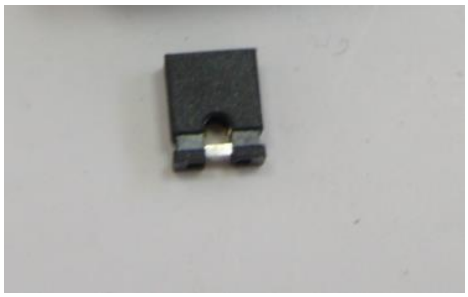
확인을 누르면 다음과 같은 설정화면이 표시됩니다. 비트/초는 115200 으로 흐름제어는 “없음”으로 하신뒤, 확인을 누르세요.



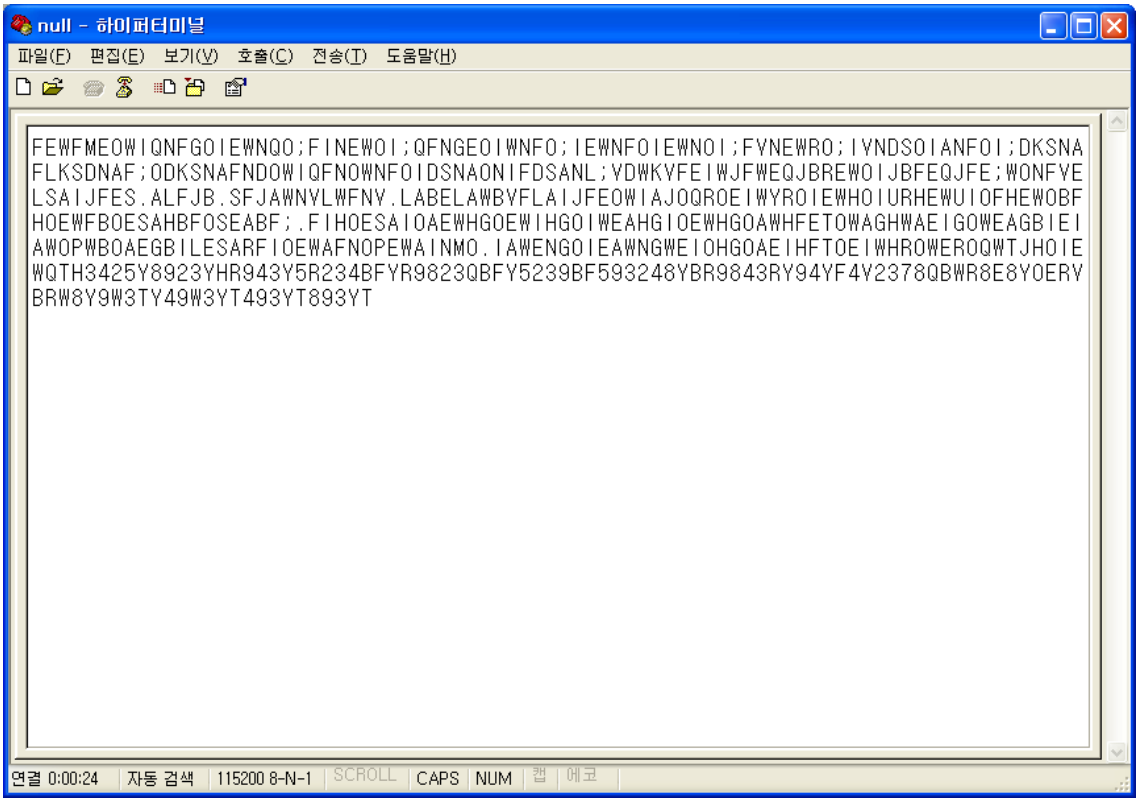
다음과 같은 화면이 됩니다. 이때 키보드의 아무키나 눌러보세요. 하이퍼 터미널화면에는 아무것도 표시되지 않습니다.



RS232 포트 케이블에 2 번 3 번단자를 쇼트 시켜보세요. 3 번 송신핀을 2 번 수신핀으로 쇼트시키면, PC 에서 송신된 데이터가 다시 PC 로 들어옵니다. 아래사진처럼 점퍼를 사용하시면 편리합니다.



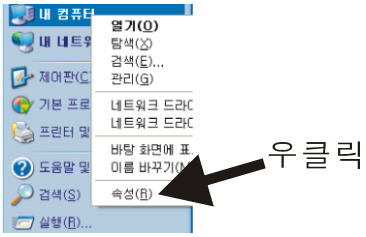
그리고 난뒤 다시 키보드의 아무키나 눌러보세요. 아래화면처럼 누른 키가 하이퍼터미널상에 표시되면 COM 포트는 정상입니다.



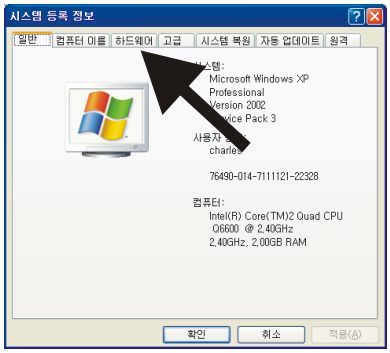
특히 RS232-TO –USB 케이블을 사용하시는 분들은 반드시 이 테스트를 해보시기 바랍니다. 간혹 드라이버가 잘못 설치되어 RS232 가 제대로 동작하지 않는 경우가 발생합니다.

2. PC 의 RS232 포트 통신설정을 체크합니다.

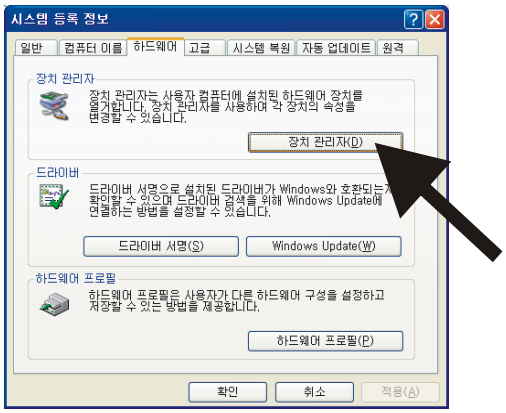
앞서의 하이퍼터미널 테스트 결과 아무 응답이 없다면, PC 의 COM 포트를 WINDOW 에서 인식하고 있는지를 살펴보십시오. 내 컴퓨터에 마우스 커서를 위치시킨뒤 우클릭하십시오.



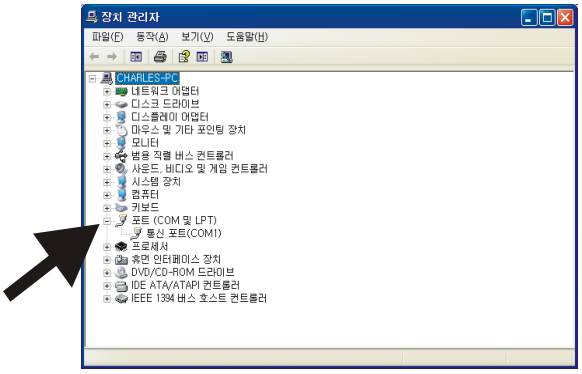
아래와 같은 박스가 표시되면 상단 [하드웨어]를 클릭하세요.



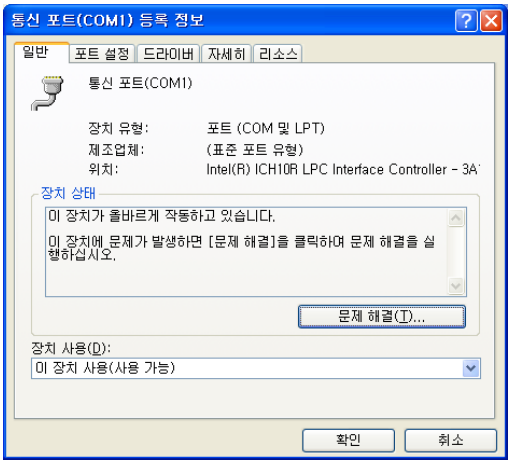
그다음 [장치관리자]를 클릭하세요.



그다음 [포트(COM 및 LPT)]라고 되어 있는 곳을 보면 통신포트가 나옵니다.



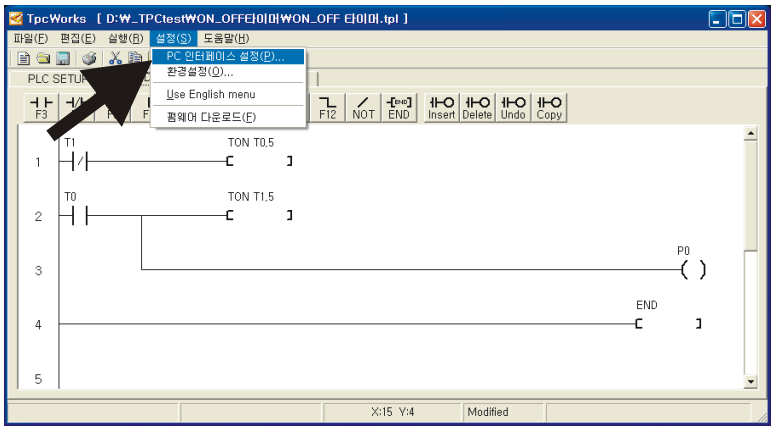
[통신포트(COM1)] 을 우클릭하고 [속성]을 선택하면 이 장치가 문제없이 동작중인지를 확인하실 수 있습니다.



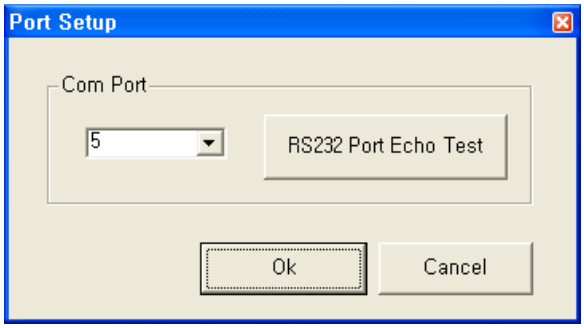
만약 사용 불가능이라고 나오면, 이 문제를 먼저 해결하십시오.

3. CUBLOC STUDIO COM 포트 셋업이 제대로 되어 있는지 확인합니다.

설정메뉴의 [PC 인터페이스 설정...]을 클릭하세요.

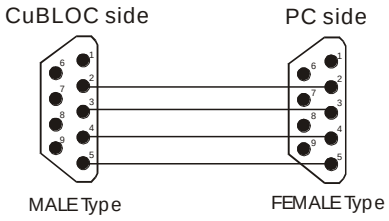


앞서 확인한 PC 의 COM 포트와 같은 포트로 설정해주십시오.



4. PC 와 CUBLOC 사이의 RS232 케이블 연결을 확인합니다.

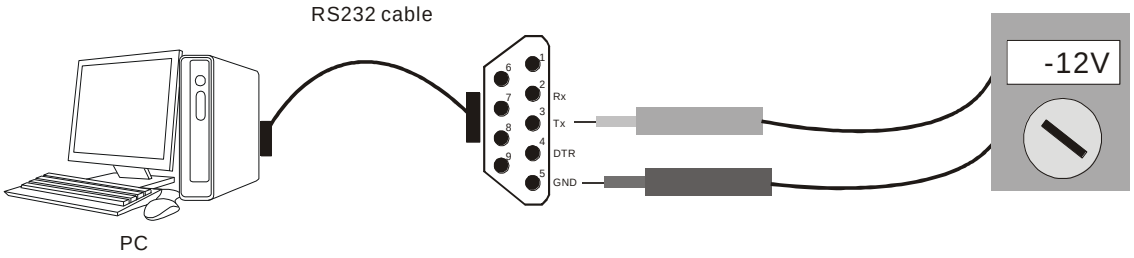
반드시 RS232 케이블은 저희 컴파일에서 구입한 케이블을 사용하십시오. 만약 따로 제작하셨다면 반드시 길이 2 미터 미만의 1:1 케이블 (1 번핀이 1 번핀, 2 번핀이 2 번핀….9 번핀이 9 번핀)으로 연결되어 있는 케이블을 사용하십시오.



간혹, 크로스케이블과 길이가 너무 긴케이블을 사용하여 문제가 발생하기도 합니다.

PC 에서 RS232 신호가 제대로 나오고 있는지도 확인하십시오. 간혹 WINDOWS 에서는 제대로 되어 있지만, 하드웨어적으로 접속이 안되어 있는 경우도 발생합니다. (마더보드 소켓과 접속이 안되어 있는경우, 접촉불량인경우)

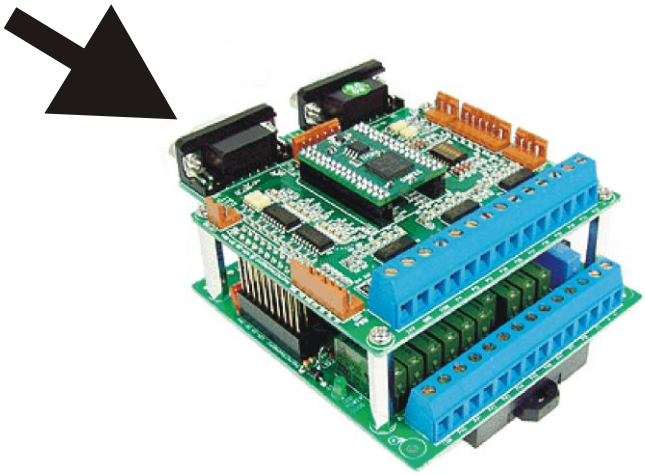
다음과 같은 방법으로 확인가능합니다. 테스터기로 PC 에 연결된 케이블의 3 번핀과 5 번핀을 측정했을 때 -10V 에서 -12V 정도가 나오고 있으면 정상입니다.



만약 아무전압도 측정이 안된다면, 케이블을 교체해보시거나, 컴퓨터 케이스를 열어 마더보드와의 연결유무를 확인해 보시기 바랍니다.

5. DOWNLOAD 포트에 연결하세요.

반드시 DOWNLOAD 포트 에 연결하세요. 간혹 RS232 CH1 에 연결하시는 분이 있습니다. RS232 CH1 은 MODBUS 통신을 위한 포트입니다.

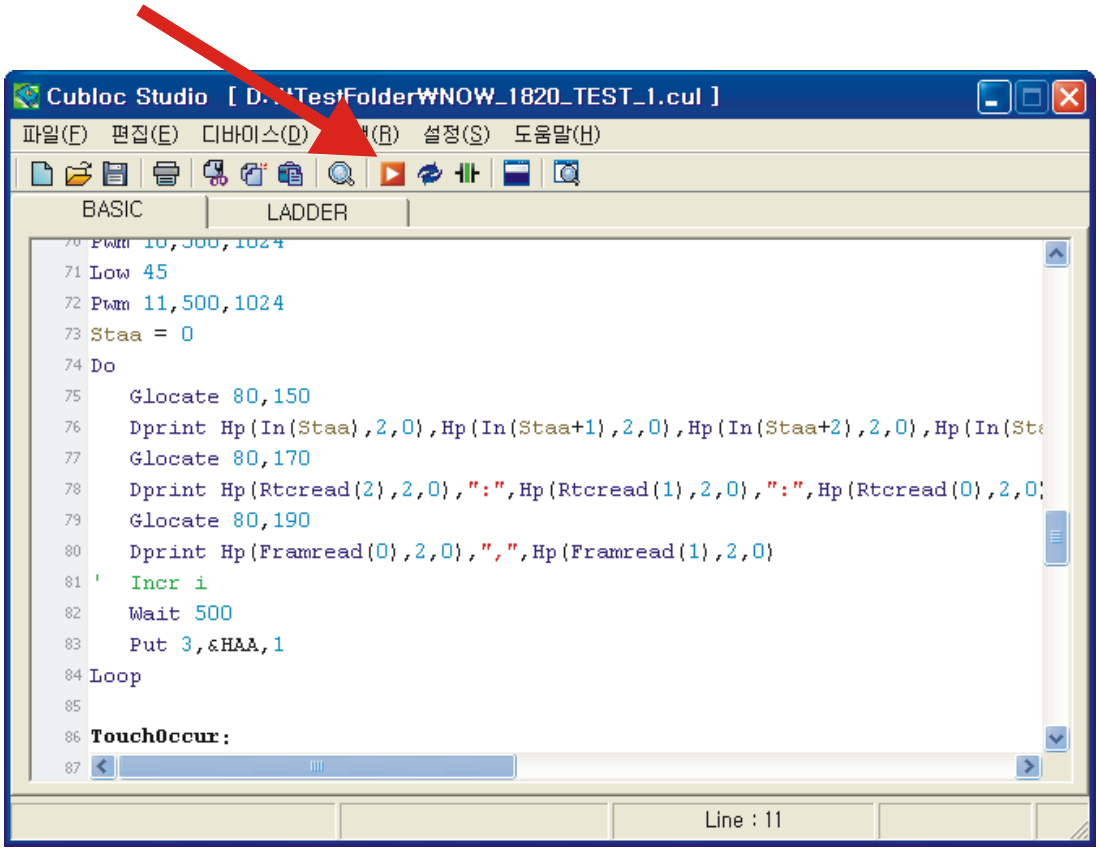


6. 전원을 인가하세요.

모든 연결이 끝난후에 보드에 전원을 인가하세요. 전원 LED 에 불이 들어오면 제대로 인가된 것입니다.

7. CUBLOC STUDIO 에서 RUN 아이콘을 클릭하세요.

이곳을 클릭하시면 다운로드가 진행됩니다.



8. 펌웨어 다운로드를 하신뒤 다시 해 보세요.

그래도 다운로드가 안된다면, [설정]메뉴에 있는 펌웨어 다운로드를 실행한뒤, 다시 RUN 아이콘을 눌러보십시오.

펌웨어 다운로드를 사용하면 CUBLOC 내부에 있는 OS 펌웨어를 모두 지우고 다시 재라이팅합니다. 간혹 정전기등의 영향으로 펌웨어의 일부가 손상된경우 이렇게 하면 다시 원상복귀됩니다.

9. Invalid RS232 또는 “존재하지 않는 RS232”라고 나오면 컴퓨터쪽만 점검하세요.

이 에러 멧세지는 컴퓨터의 문제입니다. CUBLOC, TinyPLC 제품과는 관계없이, 컴퓨터의 COM 포트가 없거나, 잘못세팅된 경우입니다.

10. A/S 를 의뢰해주세요.

위에 설명한대로 다 했는데도 동작을 안한다면, 모듈 불량일 가능성이 있습니다.

만약, 다른 모듈을 가지고 계신다면, 한번 교체해서 다운로드 해보세요. 다른 건 되는데 그거 하나만 동작을 안한다면, 모듈불량입니다. A/S 를 맡겨주시면 됩니다. (저희 본사로 택배 발송 요망)

*A/S 를 의뢰하실때에는 보드 (베이스보드)도 같이 보내주셔야 합니다. 간혹 접속불량등으로 인해 다운로드가 안되는 경우도 발생하기 때문입니다. 사용하고 계신 보드를 보내주셔야 정확한 원인을 찾을 수 있습니다.

A/S 를 보내주시기전, 한번더 점검해주세요.

저희 회사에 접수되는 모듈중 상당수가 이상이 없는 양품으로 판정되어 다시 돌려보내고 있습니다. 무려 50%가 넘습니다. 그중에서는 “펌웨어 다운로드”만 했더니 정상적으로 작동되는 경우가 많습니다.

안된다고 무조건 A/S를 보내지 마시고, 조금만 더 꼼꼼히 원인을 찾아주시기 바랍니다. 시간낭비, 비용낭비를 막을 수 있습니다.

감사합니다.