PLC / Embedded computer

CUBLOC™

User Manual

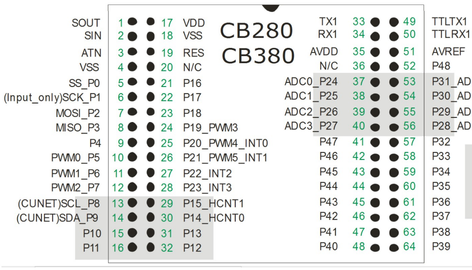
Last Updated: 2016-04-12

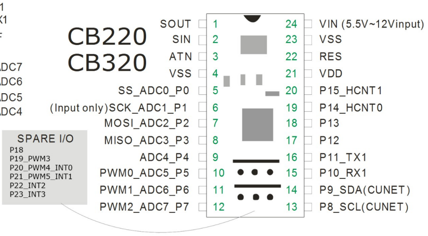
"Everything for Embedded Control"

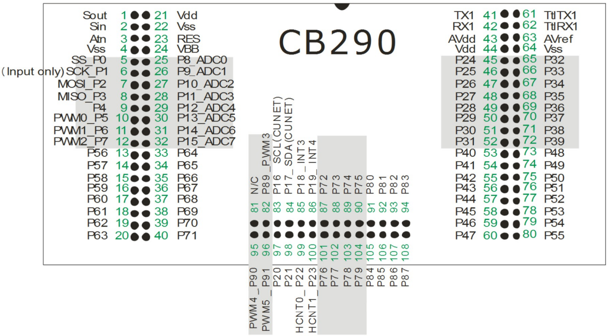
COMFILE
TECHNOLOGY

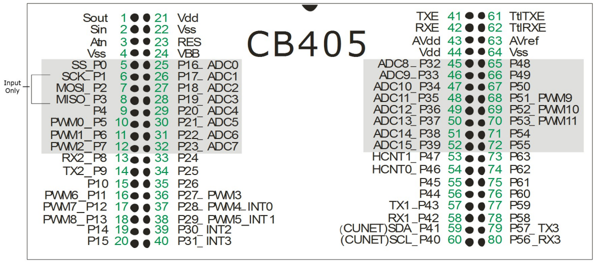
Comfile Technology Inc.
www.comfiletech.com

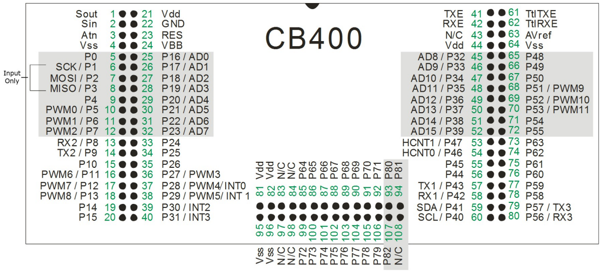
Blank Page


CB280
CB380
SOUT 1 17 VDD
SIN 2 18 VSS
ATN 3 19 RES
VSS 4 20 N/C
SS_P0 5 21 P16
(Input_only)SCK_P1 6 22 P17
MOSI_P2 7 23 P18
MISO_P3 8 24 P19_PWM3
P4 9 25 P20_PWM4_INT0
PWM0_P5 10 26 P21_PWM5_INT1
PWM1_P6 11 27 P22_INT2
PWM2_P7 12 28 P23_INT3
(CUNET)SCL_P8 13 29 P15_HCNT1
(CUNET)SDA_P9 14 30 P14_HCNT0
P10 15 31 P13
P11 16 32 P12
TX1 33 49 TTLTX1
RX1 34 50 TTLRX1
AVDD 35 51 AVREF
N/C 36 52 P48
ADC0_P24 37 53 P31_ADC7
ADC1_P25 38 54 P30_ADC6
ADC2_P26 39 55 P29_ADC5
ADC3_P27 40 56 P28_ADC4
P47 41 57 P32
P46 42 58 P33
P45 43 59 P34
P44 44 60 P35
P43 45 61 P36
P42 46 62 P37
P41 47 63 P38
P40 48 64 P39


CB220
CB320
SPARE I/O
P18
P19_PWM3
P20_PWM4_INT0
P21_PWM5_INT1
P22_INT2
P23_INT3
SOUT 1 24 VIN (5.5V~12Vinput)
SIN 2 23 VSS
ATN 3 22 RES
VSS 4 21 VDD
SS_ADC0_P0 5 20 P15_HCNT1
(Input only)SCK_ADC1_P1 6 19 P14_HCNT0
MOSI_ADC2_P2 7 18 P13
MISO_ADC3_P3 8 17 P12
ADC4_P4 9 16 P11_TX1
PWM0_ADC5_P5 10 15 P10_RX1
PWM1_ADC6_P6 11 14 P9_SDA(CUNET)
PWM2_ADC7_P7 12 13 P8_SCL(CUNET)


CB290
Sout 1 21 Vdd
Sin 2 22 Vss
Atn 3 23 RES
Vss 4 24 VBB
SS_P0 5 25 P8_ADC0
(Input only) SCK_P1 6 26 P9_ADC1
MOSI_P2 7 27 P10_ADC2
MISO_P3 8 28 P11_ADC3
P4 9 29 P12_ADC4
PWM0_P5 10 30 P13_ADC5
PWM1_P6 11 31 P14_ADC6
PWM2_P7 12 32 P15_ADC7
P56 13 33 P64
P57 14 34 P65
P58 15 35 P66
P59 16 36 P67
P60 17 37 P68
P61 18 38 P69
P62 19 39 P70
P63 20 40 P71
TX1 41 61 TtlTX1
RX1 42 62 TtlRX1
AVdd 43 63 AVref
Vdd 44 64 Vss
P24 45 65 P32
P25 46 66 P33
P26 47 67 P34
P27 48 68 P35
P28 49 69 P36
P29 50 70 P37
P30 51 71 P38
P31 52 72 P39
P40 53 73 P48
P41 54 74 P49
P42 55 75 P50
P43 56 76 P51
P44 57 77 P52
P45 58 78 P53
P46 59 79 P54
P47 60 80 P55
81 N/C
82 P89 _PWM3
83 P16 _SCL(CUNET)
84 P17 _SDA(CUNET)
85 P18 _INT3
86 P19 _INT4
87 P72
88 P73
89 P74
90 P75
91 P80
92 P81
93 P82
94 P83
95 PWM4 _P90
96 PWM5 _P91
97 P20
98 P21
99 HCNT0 _P22
100 HCNT1 _P23
101 P76
102 P77
103 P78
104 P79
105 P84
106 P85
107 P86
108 P87


CB405
Input Only
Sout 1 21 Vdd
Sin 2 22 Vss
Atn 3 23 RES
Vss 4 24 VBB
SS_P0 5 25 P16_ADC0
SCK_P1 6 26 P17_ADC1
MOSI_P2 7 27 P18_ADC2
MISO_P3 8 28 P19_ADC3
P4 9 29 P20_ADC4
PWM0_P5 10 30 P21_ADC5
PWM1_P6 11 31 P22_ADC6
PWM2_P7 12 32 P23_ADC7
RX2_P8 13 33 P24
TX2_P9 14 34 P25
P10 15 35 P26
PWM6_P11 16 36 P27_PWM3
PWM7_P12 17 37 P28_PWM4_INT0
PWM8_P13 18 38 P29_PWM5_INT1
P14 19 39 P30_INT2
P15 20 40 P31_INT3
TXE 41 61 TtlTXE
RXE 42 62 TtlRXE
AVdd 43 63 AVref
Vdd 44 64 Vss
ADC8_P32 45 65 P48
ADC9_P33 46 66 P49
ADC10_P34 47 67 P50
ADC11_P35 48 68 P51_PWM9
ADC12_P36 49 69 P52_PWM10
ADC13_P37 50 70 P53_PWM11
ADC14_P38 51 71 P54
ADC15_P39 52 72 P55
HCNT1_P47 53 73 P63
HCNT0_P46 54 74 P62
P45 55 75 P61
P44 56 76 P60
TX1_P43 57 77 P59
RX1_P42 58 78 P58
(CUNET)SDA_P41 59 79 P57_TX3
(CUNET)SCL_P40 60 80 P56_RX3


CB400
Input Only
Sout 1 21 Vdd
Sin 2 22 GND
Atn 3 23 RES
Vss 4 24 VBB
P0 5 25 P16 / AD0
SCK / P1 6 26 P17 / AD1
MOSI / P2 7 27 P18 / AD2
MISO / P3 8 28 P19 / AD3
P4 9 29 P20 / AD4
PWM0 / P5 10 30 P21 / AD5
PWM1 / P6 11 31 P22 / AD6
PWM2 / P7 12 32 P23 / AD7
RX2 / P8 13 33 P24
TX2 / P9 14 34 P25
P10 15 35 P26
PWM6 / P11 16 36 P27 / PWM3
PWM7 / P12 17 37 P28 / PWM4/INT0
PWM8 / P13 18 38 P29 / PWM5/ INT 1
P14 19 39 P30 / INT2
P15 20 40 P31 / INT3
TXE 41 61 TtlTXE
RXE 42 62 TtlRXE
N/C 43 63 AVref
Vdd 44 64 Vss
AD8 / P32 45 65 P48
AD9 / P33 46 66 P49
AD10 / P34 47 67 P50
AD11 / P35 48 68 P51 / PWM9
AD12 / P36 49 69 P52 / PWM10
AD13 / P37 50 70 P53 / PWM11
AD14 / P38 51 71 P54
AD15 / P39 52 72 P55
HCNT1 / P47 53 73 P63
HCNT0 / P46 54 74 P62
P45 55 75 P61
P44 56 76 P60
TX1 / P43 57 77 P59
RX1 / P42 58 78 P58
SDA / P41 59 79 P57 / TX3
SCL / P40 60 80 P56 / RX3
81 Vdd
82 Vdd
83 N/C
84 N/C
85 P64
86 P65
87 P66
88 P67
89 P68
90 P69
91 P70
92 P71
93 P80
94 P81
95 Vss
96 Vss
97 N/C
98 N/C
99 P72
100 P73
101 P74
102 P75
103 P76
104 P77
105 P78
106 P79
107 P82
108 N/C

Warranty

Comfile Technology provides a one year warranty on its products against defects in materials and workmanship. If you discover a defect, Comfile Technology will, at its option, repair the product, replace the product, or refund the purchase price. Simply return the product with a description of the problem and a copy of your invoice (if you do not have your invoice, please include your name and telephone number). This warranty does not apply if the product has been modified or damaged by accident, abuse, or misuse.

30-Day Money-Back Guarantee

If, within 30 days of having received your product, you find that it does not suit your needs, you may return it for a refund. Comfile Technology will refund the purchase price of the product, excluding shipping/handling costs. This does not apply if the product has been altered or damaged.

Copyright & Trademarks

Notice

This manual may be changed or updated without notice. Comfile Technology Inc. is not responsible for any actions taken outside the explanation of this manual. This product is protected by patents across the world. You may not change, copy, reproduce, or translate it without the consent of Comfile Technology Inc.

Disclaimer of Liability

Comfile Technology Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and costs or recovering, reprogramming, or reproducing any data stored in or use with Comfile Technology products.

Preface

Comfile Technology has been developing PLC and BASIC controllers since 1997. Leveraging previous experience, we are providing a unique product that is powerful, flexible, and has the best features of both BASIC controllers and PLCs (Programmable Logic Controllers).

Ladder Logic, which is a traditional way of programming PLCs for its outstanding reliability and straightforward design, cannot easily cope with graphic interfaces and other functions that require complex code. In these situations, the BASIC programming approach greatly simplifies the work required to implement many complex features.

Cubloc is able to execute BASIC and Ladder Logic simultaneously through on-chip multitasking. By sharing data in common memory, users are able to integrate both BASIC and Ladder Logic to efficiently take advantage of both programming approaches.

Cubloc was created for beginners and advanced users alike. Its simplified commands and programming tools are an easy way to get started with microcontrollers, yet the device is powerful enough to handle serious automation applications with minimal time spent in the programming phase.

With our Plug-N-Play displays, development boards, and relay boards, you will be able to put an application together in matter or hours, instead of months.

Comfile Technology, Inc.

Notice

The Start Kit or Industrial Kit comes with the latest version of Cubloc Studio at the time the CD was created.

- Please be aware that the software may be upgraded often.
- Please check www.comfiletech.com to download the latest version of Cubloc Studio.
- Please run Setup->Firmware Download after installing a new version of Cubloc Studio as the newest firmware is distributed with Cubloc Studio.
- Please check www.comfiletech.com for latest User's Manual.
- Please be sure to insert the Cubloc module correctly as inserting it improperly can damage the module.
- Please be aware that our 1 Year Warranty only covers defective items.

Table of Contents

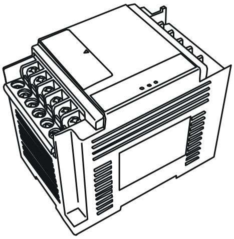
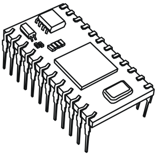
MEMO

Chapter 1: Getting Started

What is Cubloc?

Cubloc is different from the traditional PLCs that you may have encountered. Traditional PLCs are built into cases and have hardwired connections, but Cubloc is an "On-Chip" PLC/Industrial Controller, meaning you have more freedom and flexibility in the final product size and design.

Cubloc Modules are similar to traditional PLCs in that Ladder Logic can be used…but the small size allows developers to design custom PCBs for any application.

Traditional PLC	Cubloc
	

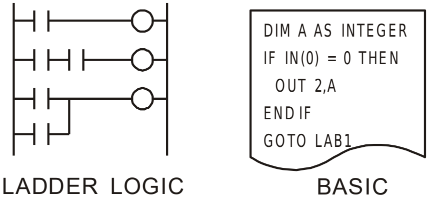
There are different models, each with a unique program memory size and number of I/O ports. Please make a selection based on your product's requirement.

Cubloc Specifications

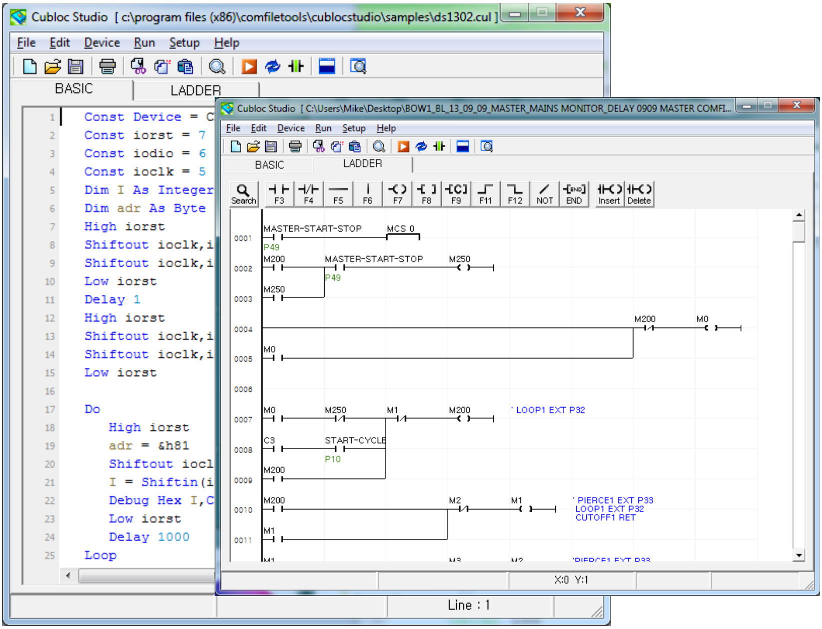
	CB210	CB220	CB280	CB290
Picture				
Program Memory	80KB	80KB	80KB	80KB
Data Memory	2KB(BASIC)+ 1KB(Ladder Logic)	2KB(BASIC)+ 1KB(Ladder Logic)	2KB(BASIC)+ 1KB(Ladder Logic)	24KB(BASIC)+ 4KB(Ladder Logic)
EEPROM	4KB EEPROM	4KB EEPROM	4KB EEPROM	4KB EEPROM
Program Speed	36,000 inst./sec	36,000 inst./sec	36,000 inst./sec	36,000 inst./sec
General Purpose I/O	20 I/O lines (5V TTL) (input/output configurable)	16 I/O lines (5V TTL) (input/output configurable) + Spare I/O 6 (5V TTL)	49 I/O lines (5V TTL) (input/output configurable)	91 I/O lines (5V TTL) (33 input only + 32 output only + 26 input/output configurable)
Serial Ports	1 serial port (Channel 1: TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps)	2 serial ports (Channel 0: RS232C 12V, Channel 1: TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps	2 serial ports (Channel 0: RS232C 12V, Channel 1: RS232C 12V & TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps	2 serial ports (Channel 0: RS232C 12V, Channel 1: RS232C 12V & TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps
Analog Inputs	6 Channel 10-bit ADCs	8 Channel 10-bit ADCs	8 Channel 10-bit ADCs	8 channel 10-bit ADCs
Analog Outputs	- 3 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz	- 3 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz	- 6 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz	- 6 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz
External Interrupts	None	4 channels (in spare I/O)	4 Channels	4 Channels
High Speed Counters	None	2 Channel 32-bit Counters (up to 2Mhz)	2 Channel 32-bit Counters (up to 2Mhz)	2 Channel 32-bit Counters (up to 2Mhz)
Power	DC 9V to 12V, 100mA or From USB	5 to 12V, 40mA (ports unloaded)	5V, 40mA (ports unloaded)	5V, 70mA (ports unloaded)
RTC	No	No	No	Yes
Data Memory Backup	None	None	None	Optional
Operating Temp.	-40 °C to 120 °C	-40 °C to 120 °C	-40 °C to 120 °C	-40 °C to 120 °C
Package	I/O Board	24-pin DIP 600mil	64-pin Module	108-pin Module
Size	2.9"L x 2.0"W x 0.4"H (75 x 53 x 12 mm)	1.2"L x 0.6"W x 0.4"H (30 x 15.3 x 11 mm)	1.4"L x 1"W x 0.4"H (35 x 25.4 x 11 mm)	2.4"L x 1.9"W x 0.5"H (59.4 x 47.8 x 13 mm)

	CB320	CB380	CB405	CB400
Picture				
Program Memory	200KB	200KB	200KB	200KB
Data Memory	6KB(BASIC)+ 1KB(Ladder Logic)	6KB(BASIC)+ 1KB(Ladder Logic)	51KB(BASIC)+4KB(Ladder Logic)+55KB(Heap)	6KB(BASIC)+ 1KB(Ladder Logic)
EEPROM	4KB EEPROM	4KB EEPROM	4KB EEPROM	4KB EEPROM
Program Speed	36,000 inst./sec	36,000 inst./sec	36,000 inst./sec	36,000 inst./sec
General Purpose I/O	16 I/O lines (5V TTL) (input/output configurable) + Spare I/O 6 (5V TTL)	49 I/O lines (5V TTL) (input/output configurable)	64 I/O lines (5V TTL) (input/output configurable)	83 I/O lines (5V TTL) (input/output configurable)
Serial Ports	2 serial ports (Channel 0: RS 232C 12V, Channel 1: TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps	2 serial ports (Channel 0: RS232C 12V, Channel 1: RS232C 12V & TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps	4 serial ports (Channel 0: RS232C 12V, Channel 1 to 3: RS232C TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps	4 serial ports (Channel 0: RS232C 12V, Channel 1 to 3: RS232C TTL 5V) - Configurable Baud rates: 2400bps to 230,400 bps
Analog Inputs	8 Channel 10-bit ADCs	8 Channel 10-bit ADCs	16 channel 10-bit ADCs	16 Channel 10-bit ADCs
Analog Outputs	- 6 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz	- 6 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz	- 12 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz	- 12 Channel 16-bit PWMs (DACs) - Frequency: 35hz to 1.5Mhz
External Interrupts	4 Channels (in Spare I/O)	4 Channels	4 Channels	4 Channels
High Speed Counters	2 Channel 32-bit Counters (up to 2Mhz)	2 Channel 32-bit Counters (up to 2Mhz)	2 Channel 32-bit Counters (up to 2Mhz)	2 Channel 32-bit Counters (up to 2Mhz)
Power	5 to 12V, 40mA (ports unloaded)	5V, 40mA (ports unloaded)	5V, 50mA (ports unloaded)	5V, 40mA (ports unloaded)
RTC	No	No	No	No
Data Memory Backup	None	None	Optional	None
Operating Temp.	-40 °C to 120 °C	-40 °C to 120 °C	-40 °C to 120 °C	-40 °C to 120 °C
Package	24-pin DIP 600mil	64-pin Module	80-pin Module	108-pin Module
Size	1.2"L x 0.6"W x 0.4"H (30 x 15.3 x 11 mm)	1.4"L x 1"W x 0.4"H (35 x 25.4 x 11 mm)	2.4"L x 1.9"W x 0.5"H (59.4 x 47.8 x 13 mm)	2.4"L x 1.9"W x 0.5"H (59.4 x 47.8 x 13 mm)

The main advantage of Cubloc is that it fills Ladder Logic's weaknesses with the BASIC language. Ladder Logic is good enough to replace sequence diagrams, but to collect data, print graphics, and process complex tasks is asking a little bit too much. That is why we added the BASIC language. You can now run both Ladder Logic and/or BASIC!



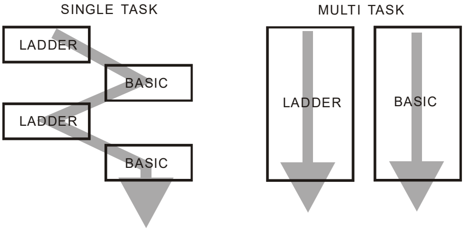
```
DIM A AS INTEGER
IF IN(0) = 0 THEN
  OUT 2,A
END IF
GOTO LAB1
```


Cubloc Studio [c:\program files (x86)\comfiletools\cublocstudio\samples\ds1302.cul]
File Edit Device Run Setup Help
BASIC
LADDER
Cubloc Studio [C:\Users\Mike\Desktop\BOW1_BL_13_09_09_MASTER_MAINS MONITOR_DELAY 0909 MASTER COMFI...
File Edit Device Run Setup Help
BASIC
LADDER
Search F3 F4 F5 F6 F7 F8 F9 F11 F12 NOT END Insert Delete
0001 MASTER-START-STOP P49 MCS 0
0002 M200 MASTER-START-STOP P49 M250
0003 M250
0004 M200 M0
0005 M0
0006
0007 M0 M250 M1 M200 ' LOOP1 EXT P32
0008 C3 START-CYCLE P10
0009 M200
0010 M200 M2 M1 ' PIERCE1 EXT P33 LOOP1 EXT P32 CUTOFF1 RET
0011 M1
M1 M3 M2 'PIERCE1 EXT P33
X:0 Y:1
Line : 1

```
Const Device = C
Const iorst = 7
Const iodio = 6
Const ioclk = 5
Dim I As Integer
Dim adr As Byte
High iorst
Shiftout ioclk,i
Shiftout ioclk,i
Low iorst
Delay 1
High iorst
Shiftout ioclk,i
Shiftout ioclk,i
Low iorst

Do
   High iorst
   adr = &h81
   Shiftout iocl
   I = Shiftin(i
   Debug Hex I,C
   Low iorst
   Delay 1000
Loop
```

Image of Cubloc Studio is shown above.

There are other PLCs on the current market that support both Ladder Logic and BASIC. However, these PLCs do not multi-task. Because BASIC is part of their Ladder Logic, it does not run independently like Cubloc or Cutouch. This can prove to be costly since BASIC is not real-time oriented and can delay the Ladder Logic scans, possible causing missed inputs or other undesired behavior. Cubloc, on the other hand, doesn't suffer from these weaknesses because it multitasks; guaranteeing accuracy and precise timing.



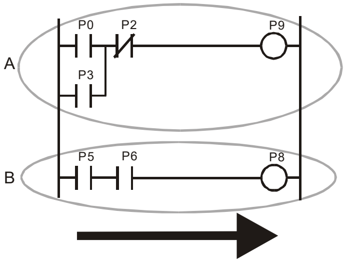
Cubloc is a brand new type of industrial controller. By being able to do things that traditional PLCs can’t, we have expanded the horizons of both PLCs and BASIC micro-computers.

Cubloc is fully backed by many Plug-N-Play peripherals such as our CuBASE industrial I/O Boards and Plug-N-Play Relay8 Boards. With these peripherals, controlling DC/AC devices is easy.

With 32-bit IEEE floating point math support and MODBUS ASCII/RTU support, the user will find that Cubloc and Cutouch are among the most versatile BASIC/PLC hybrid chips on the market today.

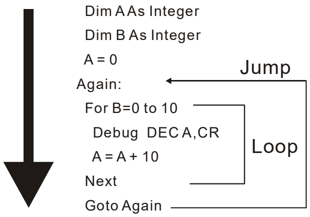
Ladder Logic and BASIC

Ladder Logic's greatest advantage is that all circuits are laid out in parallel; they are all processed as fast as the ladder scan time will allow. This allows for a more parallel execution path for unrelated functions.



As you can see above, both A and B circuits are in a waiting state, ready to turn the output On as soon as input is turned On. For example, if input P3 turned On, P9 would turn On.

In comparison, BASIC processes code in order, a type of "Sequential Processing."



```
 Dim A As Integer
 Dim B As Integer
 A = 0
Again:
 For B=0 to 10
  Debug  DEC A,CR
  A = A + 10
 Next
 Goto Again
```

These 2 types of programming languages have been used in different fields for a long time. Ladder Logic is used in automation controllers such as PLCs. On the other hand, BASIC and other programming languages such as C and Assembly have been used in PCs and MCUs.

Whether you are an experienced MCU or PLC user, you will be able to benefit by integrating both BASIC and Ladder Logic in your designs.

Another advantage of Ladder Logic is the ability to process input within a guaranteed slot of time. No matter how complex the circuit becomes, Ladder Logic is always ready to output when it receives input. This is the primary reason why it is used for machine control and other automation fields.

Ladder Logic is more logic oriented. It is not a complete programming language. To do complex processes, it has its limits. For example, to receive input from a keypad, display to 7 Segment or LCD, and process users' input is a difficult task for standard Ladder Logic.

But these things are rarely a problem for programming languages such as BASIC. BASIC is able to process floating point numbers, data communications, and other things beyond the scope of what Ladder Logic can do alone. Another advantage is that its syntax is very similar to the English language (IF, GOTO, etc…), allowing both beginners and experienced developers to learn in matter of hours instead of months. BASIC is a very common programming language, and many developers may be able to start programming a Cubloc with only a few glances at hardware-specific commands.

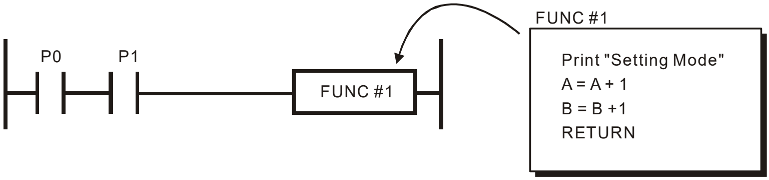
	Ladder Logic	Programming Languages (BASIC, C, ASM)
Device	PLC	PC or Micro-Computer
Application	Automation, Machine-Control	General Computing
Advantages	Sequencer, Bit Logic, Timers, Counters	Complex Math, Data Communication, Data Collection & Process, Analysis, Graphic Interface
Basic Mechanism	Parallel	Sequential

Ladder Logic’s parallelism and BASIC's sequential language both have advantages. Ladder Logic makes controlling unrelated parallel tasks easy, which can be difficult with BASIC. On the other hand, BASIC can easily process complex sequential tasks and has a wider range of commands and interface abilities.

That is why we created "Cubloc," where the user is free to use both Ladder Logic and/or BASIC based on the application being created. After understanding the advantages of both Ladder Logic and BASIC, the user will be able to create more efficient final products while saving development time and reducing cost.

Multi-tasking of Ladder Logic and BASIC

There are many ways to implement both BASIC and Ladder Logic in one processor. The current products on the market use BASIC as part of Ladder Logic. These products support BASIC and Ladder Logic but these products have a few disadvantages.



```
Print "Setting Mode"
A = A + 1
B = B +1
RETURN
```

The first disadvantage is that when BASIC is executing, the execution time of Ladder Logic also gets affected. For example, if BASIC creates in an infinite loop, Ladder Logic will stop. Ladder Logic's main advantage is that it can process input in a guaranteed scan time. If Ladder Logic cannot process within this guaranteed scan time because of BASIC, it might be better to not use BASIC at all.

The second disadvantage is that BASIC routines can only be started from Ladder Logic. BASIC is a powerful language and is able to process complex algorithms in a sequential manner. But if we can only use BASIC as part of Ladder Logic, we are not utilizing all of its capabilities.

The third disadvantage involves I/O. BASIC's execution of I/O can create unwanted collisions with Ladder Logic. The reason is that Ladder Logic I/O is updated once per scan, while BASIC I/O is accessed immediately.

To address these problems, we have created a BASIC and Ladder Logic processor that supports real-time multi-tasking. BASIC runs BASIC and Ladder Logic runs Ladder Logic, without causing collisions.

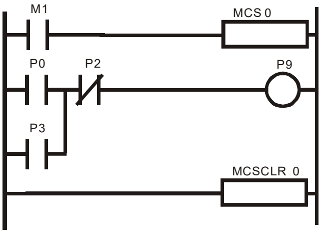
Even if you only use BASIC, you will be able to build innumerable applications. In comparison to many other BASIC processors on the market today, Cubloc's BASIC has a faster processing speed and the upper hand on its main features.

In the case of I/O, the user can specify the I/O used by BASIC and Ladder Logic, thereby eliminating I/O collision problems.

If you use Ladder, we recommend using some BASIC as a method of supervising the Ladder operations.

For example, there is a Master Control feature in Ladder Logic, allowing the user to set Control Zones. Control Zones are sections within the Ladder Logic containing portions of the control circuit. With the Master Control feature, the user can enable/disable Ladder Logic's Control Zones easily.

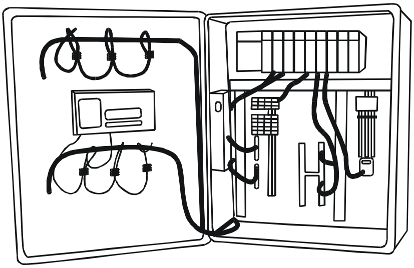
```
If A=1 THEN _M(1) = 1
If B=1 THEN _M(1) = 0
```



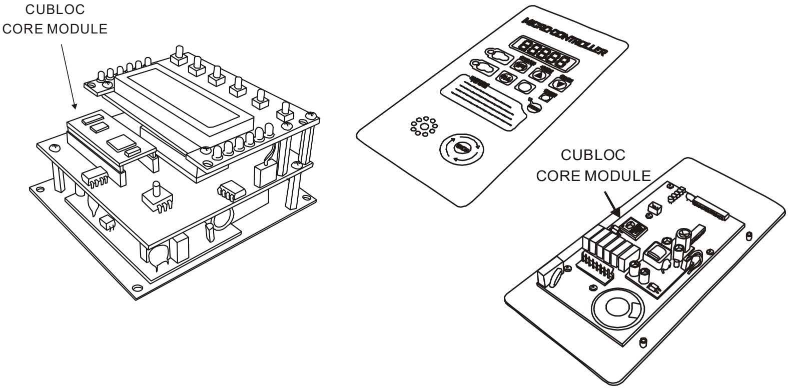
In BASIC, the user may read or write to Ladder Logic's data memory. In the above example, you can access Register M1 as _M(1) and write to it from BASIC.

Advantages of an On-Chip PLC/Embedded Computer

Cubloc's greatest advantages is that it is an "On-Chip" PLC. Normally, we think of a PLC as a block type case with input and output lines. These modules are usually mounted within yet another case, with external power supplies, additional output modules, and other wiring requirements.

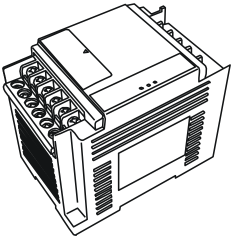
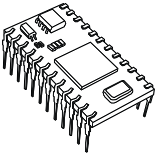


This is usually fine for one or two applications, but doesn’t lend itself easily to larger scale production. Cubloc modules can be easily integrated into a custom product, providing all the features of a PLC yet the professional appearance and lower manufacturing cost of a custom design.



Cubloc is an On-Chip PLC, allowing an easy fit on a PCB. You may use the PLC almost like an MCU. You can design a customized PCB for your desired product, reducing its cost and size, and most importantly, making your product one-of-a-kind.

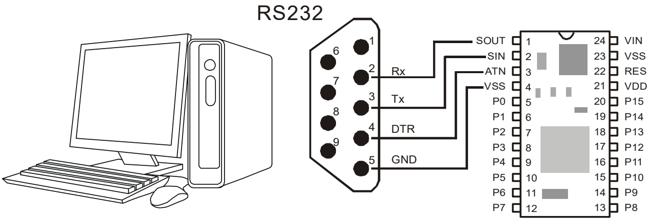
The following table shows a few differences between a traditional PLC and "On-Chip" PLC/Micro-computer, Cubloc.

	Traditional PLC	Cubloc
		
Production	Din Rail Attachment	PCB
Labor Costs	High	Low
Mass Production	Difficult	Easy
Final Product Cost	High	Low
Final Size	Large	Compact

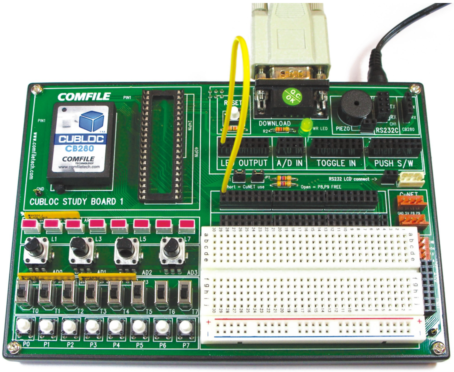
If you are currently distributing a system using a traditional PLC, please review our products and consider the reduction in cost if you were to use Cubloc instead. We believe that you will end up with a more superior product at a fraction of the cost.

Development Environment

Cubloc Studio can be installed on a Windows 7, Vista, XP, 2000, or 98 operating system equipped computer. If you would like to use it in a Linux/Unix/Macintosh environment, you will need to install a virtual machine (such as VMware) that allows your computer to run the Windows operating system. An RS-232 port is also required, or you may use a USB-to-RS232C converter.



Downloading and monitoring is possible when connected to the PC. When the Cubloc is disconnected from the PC, it goes into a stand-alone state. The main program is stored in Cubloc's flash memory, and will be retained between power cycles. Each Cubloc can be programmed and/or erased more than 10,000 times.



CB280 core module with Study Board

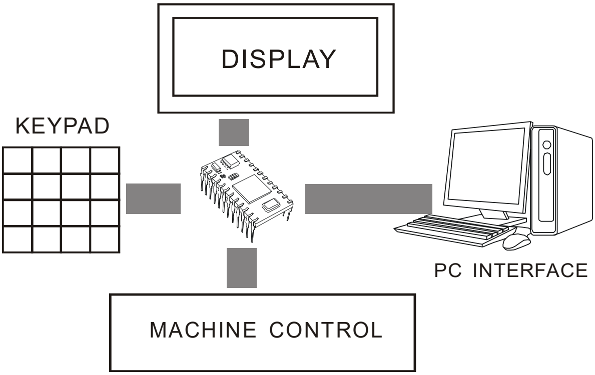
Hints for Traditional PLC Users

For users with much experience in traditional PLCs, they will find BASIC to be a completely new language. Cubloc is a PLC with BASIC language capabilities added. If uncomfortable with BASIC, however, the user may program using only the Ladder Logic.

Even a Ladder Logic user may be able to incorporate new features into the final product by making use of BASIC, as BASIC offer additional capabilities and flexibility for communicating with other devices besides PLCs.

To use Cubloc, the user does not have to know BASIC. If the user does not require LCD display or keypad usage, he or she does not need to use BASIC at all, and can resort to using only Ladder Logic.

As you can realize, more emphasis on human interfaces is becoming apparent in our industrial world. Cubloc is able to overcome the deficiencies and disadvantages of traditional PLCs by being able to use both BASIC and Ladder Logic language.



We provide many BASIC user interface libraries which you can simply copy and paste to achieve the user interface structure desired.

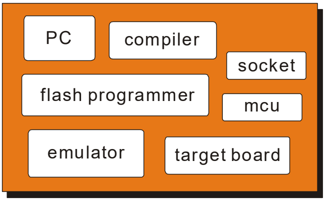
Hints for Microcontroller Users

Microcontrollers (MCU), such as the PIC, AVR, and the 8051, are self-contained programmable computers. For mass-production, MCUs can cut costs and reduce the overall product size. But one disadvantage is that it can be difficult to learn everything necessary to program an unfamiliar controller. The hardware, commands, and even programming tools vary widely between controller families. This can be a drawback for low quantity or frequently-modified projects.

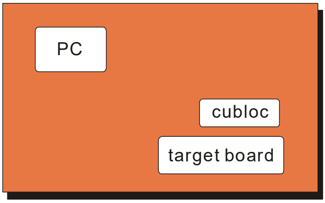
Even experienced engineers can feel that MCU programming is time-consuming. To make a final product, it takes many hours of programming and debugging with an MCU. Even after development, if bugs arise, it can be difficult to update the MCU.

In comparison, Comfile's Cubloc can cut the programmer's development time as much as 20 times, and provide an MCU-like chip that is upgradeable through an RS232 cable or even through the internet by using an XPORT. By providing a way to upgrade, the final product's value is increased.

If you have experience programming with MCUs, we guarantee you that development of your final product will be much easier with Cubloc. You will be able to spend more time designing the features of your final product, instead of spending hours relearning register locations and compiler syntax. Having Cubloc hardware on hand means that you can respond immediately to any equipment control needs.

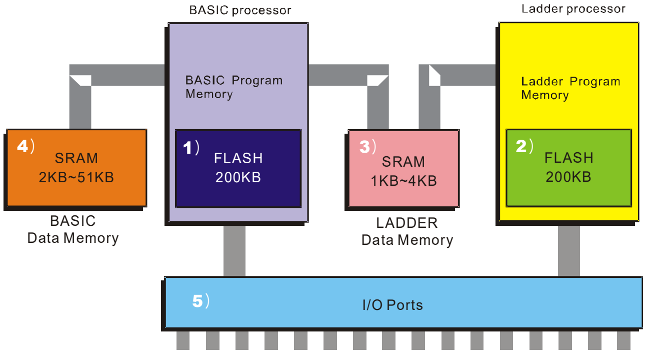


MCU engineer's desk



CUBLOC engineer's desk

Cubloc’s Internal Structure



The BASIC Interpreter controls a Flash storage area for the user’s BASIC programs. The Ladder Logic processor also has a Flash storage area for the user’s Ladder Logic program. I/O ports are shared between BASIC and Ladder Logic, allowing free access to both.

BASIC data memory can only be accessed by the BASIC Interpreter while Ladder Logic data memory can be accessed by both the BASIC Interpreter and the Ladder Logic Processor.

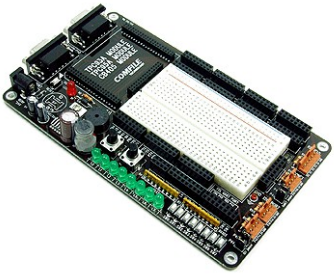
BASIC (1) and Ladder Logic (2) share the same Flash memory. The total available memory space is 80KB for some models and 200KB for others. BASIC and Ladder Logic can both use up to the entire memory area, if needed.

I/O ports (5) can be used both by BASIC and Ladder Logic. The user must specify I/O ports to use in both languages. All I/O ports can be used in Ladder Logic or BASIC.

Cubloc Peripherals

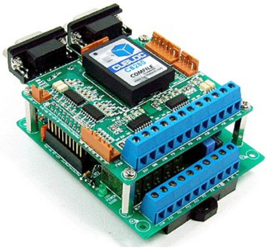
PROTO BOARD Series

Proto-boards for Cubloc can be used for testing and debugging your future products before starting PCB artwork or production. These proto-boards all include basic power and interface circuits.



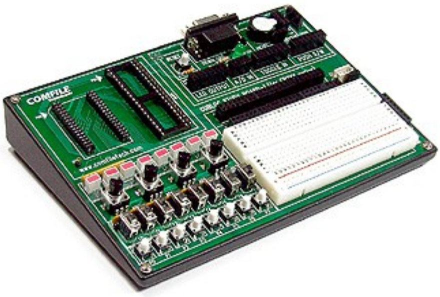
BASE BOARD / CUSB Series

The CUBASE and CUSB series are especially geared for the industrial field applications. Simply attach our **Plug-N-Play** relays to CUBASE output ports for implementing solenoids, limit switches, etc.,. With 24V input ports and DIN rail mounting brackets, the CUBASE and CUSB series integrate quickly into any automation project. For even greater integration, the CUSB series contains a switching power supply for direct operation from AC power (except CUSB-22D, requires 24V power). The CUSB modules have integrated relays and optoisolated inputs, all accessible through screw-clamp terminals.



STUDY BOARD

The Study Board is geared for Cubloc first-timers. Connections for simple experiments including switches, LEDs, RS232 communication, I2C, piezo, ADC, toggle switches, and LCDs are included. We recommend the Start Kits, which include a study board, a Cubloc module, all necessary cables, and a manual.



LCD DISPLAY Module (CLCD, GHLCD Series)

Various LCD displays are provided for use with Cubloc using the CUNET (I2C) protocol. With one line commands (PRINT, CLS, etc…), you can easily start printing to an LCD without complex commands.

CUNET is especially engineered for Cubloc displays, therefore, we recommend using CUNET supported LCDs for quick and easy development.
Our Graphic Display GHLCD allows you to download black and white bitmap images to the onboard memory and display them on demand.

Seven Segment Display Modules (CSG Series)

Seven segment display modules can be easily implemented using Cubloc’s I2C protocol and native commands.

Cutouch Series

Cutouch is an integration of our graphic LCD, touch panel, and Cubloc core module. With BASIC, you can control the LCD and touch panel. With Ladder Logic, I/O ports can be controlled in real-time.

MEMO

Chapter 2: Hardware

Hardware Features

Cubloc has the following features:

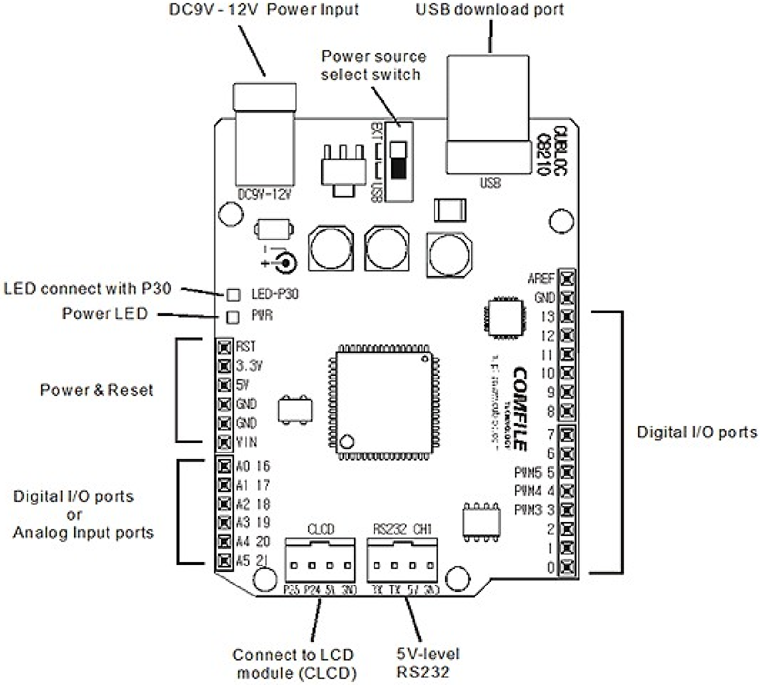
- (BASIC and/or Ladder Logic) 80KB or 200KB Flash Memory
- BASIC Execution Speed : 36,000 instructions per second
- LADDER Execution Speed : 10 millisecond scan time
- Data Memory for BASIC: 2KB to 51KB
- Data Memory for LADDER: 1KB to 4KB
- EEPROM Memory: 4KB
- 16 to 91 I/O pins (Ports)
- 8 to 16 10-bit ADC channels
- 3 to 12 PWM channels (DAC) from 8 to 16bit.
- UART (H/W RS232C ports) 2 to 4 channels
- RTC chip included (CB290)

Model Comparison Chart

Feature	**CB210**	**CB220**	**CB280**	**CB290**	**CB320**	**CB380**	**CB400**	**CB405**
Prog. Memory	80KB	80KB	80KB	80KB	200KB	200KB	200KB	200KB
Data Memory: **- BASIC** **- Ladder** **- Heap**	2KB 1KB	2KB 1KB	2KB 1KB	24KB 4KB	6KB 1KB	6KB 1KB	6KB 1KB	51KB 4KB 55KB
Battery Backup	N/A	N/A	N/A	Yes	N/A	N/A	N/A	Yes
EEPROM	4KB	4KB	4KB	4KB	4KB	4KB	4KB	4KB
I/O ports	20	16	49 + 2	91 + 2	16 + 6	49 + 2	83 + 2	64 + 2
Package	I/O Board	24 pin DIP	64 pin Module	108 pin Module	24 pin DIP	64 pin Module	108 pin Module	80 pin Module
ADCs	6	8	8	8	8	8	16	16
PWMs	3	3	6	6	3	6	12	12
RS-232 Ch.	1	2	2	2	2	2	4	4
External Interrupts	None	None	4	4	4	4	4	4
High-speed Counter Inputs	None	2	2	2	2	2	2	2
RTC	No	No	No	Yes	No	No	No	No

CB210

The CB210 has 20 digital I/O ports, 6 of which can be used for A/D input. It can be connected to a PC via USB for downloading and debugging and serial communication. It can be powered either by a 9V power supply, or through the USB port.

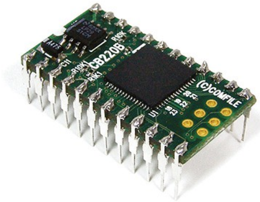


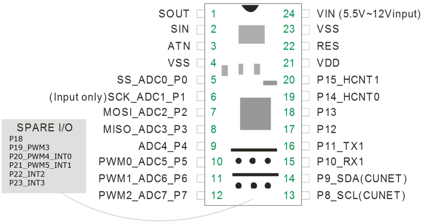
Please be aware of the following when using the CB210.

- Use Cubloc Studio V3.3.0 and above. Cubloc Studio can be downloaded from Comfile Technology's website at http://www.ComfileTech.com/. A USB Driver is included in the download. You must install the USB Driver to use the CB210.
- Only PWMs 3,4,and 5 are available. You cannot use PWM 0,1, and 2.
- You can use ports 16 to 21 for digital I/O.
- You can use ports 16 to 21 for A/D input, but you must first configure the ports as input ports. For A/D input, you should supply 3.3V-5V power to AVREF port.
- The CB210 cannot handle currents over 300mA.
- An LED is connected to port 30.

CB220 / CB320

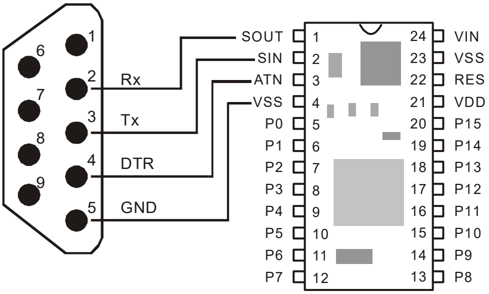
The CB220 and CB320 are a 24 pin wide DIP style packages. They have 16 I/O ports and an internal 5V power regulator. The CB220 and CB320 have 6 spare I/O ports.





Port	Pin	I/O	Port Block	Explanation
SOUT	1	OUT		DOWNLOAD SERIAL OUTPUT
SIN	2	IN		DOWNLOAD SERIAL INPUT
ATN	3	IN		DOWNLOAD SERIAL INPUT
VSS	4	POWER		GROUND
P0	5	I/O	Block 0	ADC0 / SPI SS
P1	6	Input		ADC1 / SPI SCK
P2	7	I/O		ADC2 / SPI MOSI
P3	8	I/O		ADC3 / SPI MISO
P4	9	I/O		ADC4
P5	10	I/O		PWM0 / ADC5
P6	11	I/O		PWM1 / ADC6
P7	12	I/O		PWM2 / ADC7
P8	13	I/O	Block 1	CuNET SCL
P9	14	I/O		CuNET SDA
P10	15	I/O		RS232C Channel 1 RX
P11	16	I/O		RS232C Channel 1 TX
P12	17	I/O		
P13	18	I/O		
P14	19	I/O		High-speed Counter channel 0
P15	20	I/O		High-speed Counter channel 1
P18		I/O	Block 2	
P19		I/O		PWM3
P20		I/O		PWM4 / INT0
P21		I/O		PWM5 / INT1
P22		I/O		INT2
P23		I/O		INT3
VDD	21	I/O		5V Output/Input
RES	22	IN		RESET Input (LOW signal resets!)
VSS	23	IN		GROUND
VIN	24	IN		5.5V to 12V Input Power

SIN, SOUT, ATN are RS-232 communication pins used to interface with a PC for downloading, debugging, monitoring, and serial communication. All Cubloc models have SOUT, SIN, ATN pins and are connected to a DE-9 connector (PC serial port) as shown below.



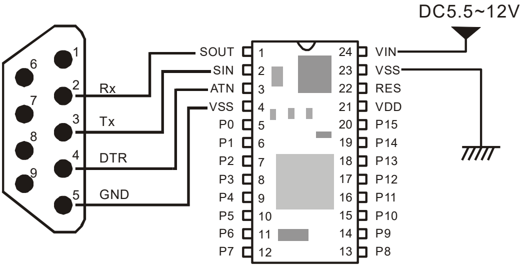
Other pins are mostly I/O ports. The user may select which ports (pins) to use as INPUT or OUTPUT. When set to INPUT, the pin enters a high-impedance state; when set to OUTPUT, the pin either outputs logic-low or logic-high. The maximum current (source/sink) available from the output ports is 25mA. The user is free to choose which I/O ports he/she will use for which purpose (such as ADC, PWM, etc…).

Supplying power to the CB220 / CB320

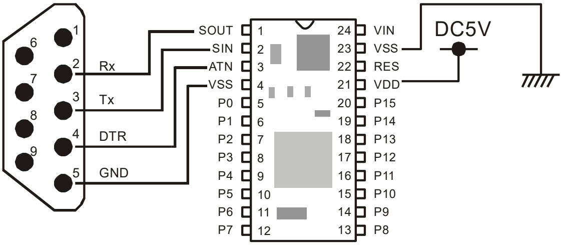
The CB220 and CB320 have an internal 5V power regulator that accepts a DC input between 5.5V and 12V.

It will produce a stable 5V at 100mA. When using the internal regulator, the supply voltage can be applied to pin 24, and 5V will appear on pin 21. If a 5V regulated power source is already available, the user may simply connect it to pin 21. If an application requires more than the 100mA of current, a separate power supply should be used.

Method 1

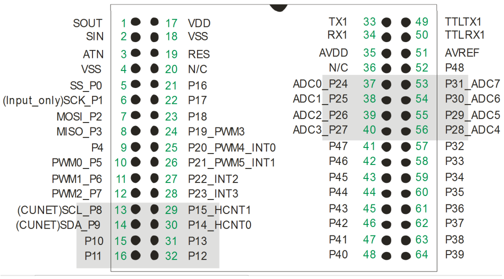


Method 2



CB280 / CB380

The CB280 and CB380 are packages of 64 pins of which 49 can be used for I/O. The CB280 and CB380 do not have an 5V regulator internal; you must supply a 5V regulated power source.

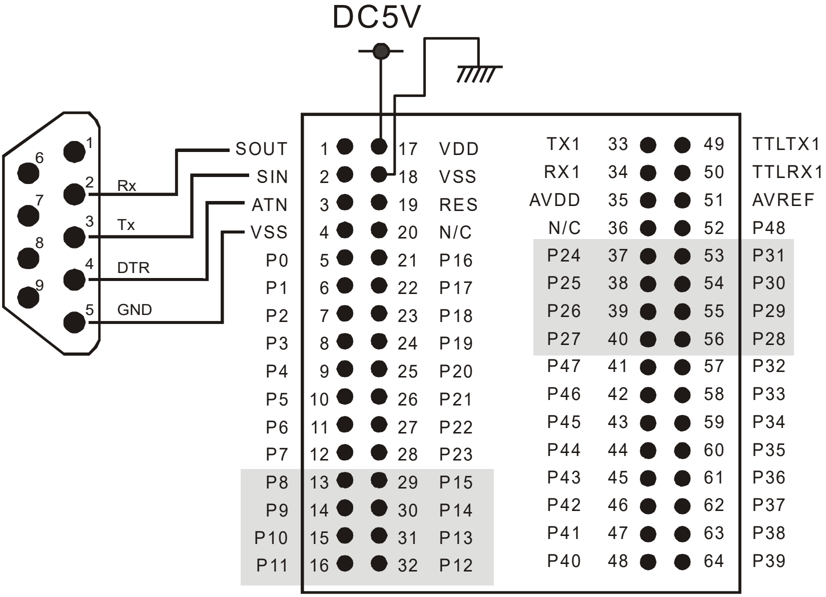


Port	Pin	I/O	Port Block	Explanation
SOUT	1	OUT		DOWNLOAD SERIAL OUTPUT
SIN	2	IN		DOWNLOAD SERIAL INPUT
ATN	3	IN		DOWNLOAD SERIAL INPUT
VSS	4	POWER		GROUND
P0	5	I/O	Block 0	SPI SS
P1	6	Input		SPI SCK
P2	7	I/O		SPI MOSI
P3	8	I/O		SP MISO
P4	9	I/O		
P5	10	I/O		PWM Channel 0
P6	11	I/O		PWM Channel 1
P7	12	I/O		PWM Channel 2
P8	13	I/O	Block 1	CuNET SCL
P9	14	I/O		CuNET SDA
P10	15	I/O		
P11	16	I/O		
P12	32	I/O		
P13	31	I/O		
P14	30	I/O		High-speed Counter Channel 0
P15	29	I/O		High-speed Counter Channel 1
P16	21	I/O	Block 2	
P17	22	I/O		
P18	23	I/O		
P19	24	I/O		PWM Channel 3
P20	25	I/O		PWM Channel 4 / INT Channel 0
P21	26	I/O		PWM Channel 5 / INT Channel 1
P22	27	I/O		INT Channel 2
P23	28	I/O		INT Channel 3

P24	37	I/O	Block 3	ADC0 : AD Channel 0
P25	38	I/O		ADC1 : AD Channel 1
P26	39	I/O		ADC2 : AD Channel 2
P27	40	I/O		ADC3 : AD Channel 3
P28	56	I/O		ADC4 : AD Channel 4
P29	55	I/O		ADC5 : AD Channel 5
P30	54	I/O		ADC6 : AD Channel 6
P31	53	I/O		ADC7 : AD Channel 7
P32	57	I/O	Block 4	
P33	58	I/O		
P34	59	I/O		
P35	60	I/O		
P36	61	I/O		
P37	62	I/O		
P38	63	I/O		
P39	64	I/O		
P40	48	I/O	Block 5	
P41	47	I/O		
P42	46	I/O		
P43	45	I/O		
P44	44	I/O		
P45	43	I/O		
P46	42	I/O		
P47	41	I/O		
P48	52	I/O		
VDD	17	IN		Power, 4.5V to 5.5V
VSS	18	IN		GROUND
RES	19	IN		RESET Input (LOW signal resets!), Normally HIGH or OPEN
TX1	33			RS232 Channel 1, +/- 12V Data Output
RX1	34			RS232 Channel 1, +/- 12V Data Input
AVDD	35			ADC Power
TTLTX1	49			RS232 Channel 1, 5V (TTL level) Data Output
TTLRX1	50			RS232 Channel 1, 5V (TTL level) Data Input
AVREF	51			ADC Reference Voltage

How to supply power to the CB280 and CB380

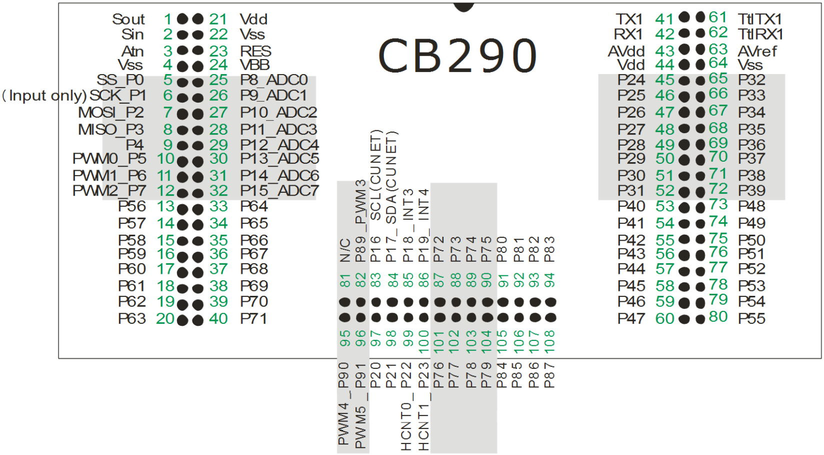
The CB280 and CB380 do not have an internal 5V regulator; you must provide your own 5V power as shown below.



* Pin 20 and 36 are not used, please DO NOT CONNECT anything to these pins.

CB290

The CB290 is a package of 108 pins of which 91 can be used as I/O ports. It has a battery-backup-capable 28KB of memory and an RTC. The CB290 does not have an internal 5V regulator. Of the 91 I/O ports, 32 ports are output only, 32 ports are input only, and rest can be set as output or input as desired by the user program.



Port	Pin	I/O	Port Block	Explanation
SOUT	1	OUT		DOWNLOAD SERIAL OUTPUT
SIN	2	IN		DOWNLOAD SERIAL INPUT
ATN	3	IN		DOWNLOAD SERIAL INPUT
VSS	4	POWER		GROUND
P0	5	I/O	Block 0	SPI SS
P1	6	Input		SPI SCK
P2	7	I/O		SPI MOSI
P3	8	I/O		SPI MISO
P4	9	I/O		
P5	10	I/O		PWM Channel 0
P6	11	I/O		PWM Channel 1
P7	12	I/O		PWM Channel 2
P8	25	I/O	Block 1	ADC0 : AD Channel 0
P9	26	I/O		ADC1 : AD Channel 1
P10	27	I/O		ADC2 : AD Channel 2
P11	28	I/O		ADC3 : AD Channel 3
P12	29	I/O		ADC4 : AD Channel 4
P13	30	I/O		ADC5 : AD Channel 5
P14	31	I/O		ADC6 : AD Channel 6
P15	32	I/O		ADC7 : AD Channel 7

P16	83	I/O	Block 2	CUNET SCL
P17	84	I/O		CUNET SDA
P18	85	I/O		INT Channel 2
P19	86	I/O		INT Channel 3
P20	97	I/O		
P21	98	I/O		
P22	99	I/O		High-speed Counter Channel 0
P23	100	I/O		High-speed Counter Channel 1
P24	45	Output	Block 3	
P25	46	Output		
P26	47	Output		
P27	48	Output		
P28	49	Output		
P29	50	Output		
P30	51	Output		
P31	52	Output		
P32	65	Output	Block 4	
P33	66	Output		
P34	67	Output		
P35	68	Output		
P36	69	Output		
P37	70	Output		
P38	71	Output		
P39	72	Output		
P40	53	Output	Block 5	
P41	54	Output		
P42	55	Output		
P43	56	Output		
P44	57	Output		
P45	58	Output		
P46	59	Output		
P47	60	Output		
P48	73	Output	Block 6	
P49	74	Output		
P50	75	Output		
P51	76	Output		
P52	77	Output		
P53	78	Output		
P54	79	Output		
P55	80	Output		
P56	13	Input	Block 7	
P57	14	Input		
P58	15	Input		
P59	16	Input		
P60	17	Input		
P61	18	Input		
P62	19	Input		
P63	20	Input		

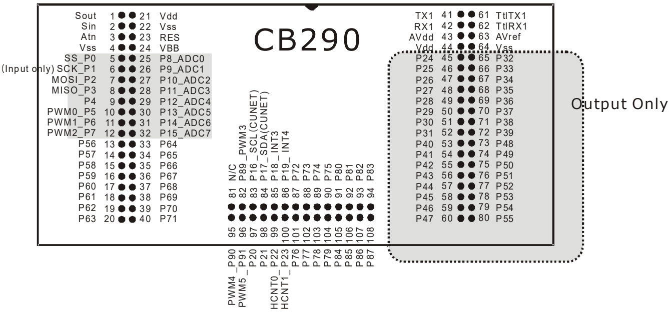
P64	33	Input	Block 8	
P65	34	Input		
P66	35	Input		
P67	36	Input		
P68	37	Input		
P69	38	Input		
P70	39	Input		
P71	40	Input		
P72	87	Input	Block 9	
P73	88	Input		
P74	89	Input		
P75	90	Input		
P76	101	Input		
P77	102	Input		
P78	103	Input		
P79	104	Input		
P80	91	Input	Block 10	
P81	92	Input		
P82	93	Input		
P83	94	Input		
P84	105	Input		
P85	106	Input		
P86	107	Input		
P87	108	Input		
P88	81	N/C	Block 11	N/C (Do not use this I/O
P89	82	I/O		PWM Channel 3
P90	95	I/O		PWM Channel 4 / INT Channel 0
P91	96	I/O		PWM Channel 5 / INT Channel 1
VDD	21,44	IN		Power, 4.5V to 5.5V
VSS	22,64	IN		GROUND
RES	23	IN		RESET Input (LOW signal resets!), Normally HIGH or OPEN
VBB	24	IN		Battery Backup
TX1	41			RS232 Channel 1, +/- 12V Data Output
RX1	42			RS232 Channel 1, +/- 12V Data Input
AVDD	43			ADC Power
TTLTX1	61			RS232 Channel 1, 5V (TTL level) Data Output
TTLRX1	62			RS232 Channel 1, 5V (TTL level) Data Input
AVREF	63			ADC Reference Voltage

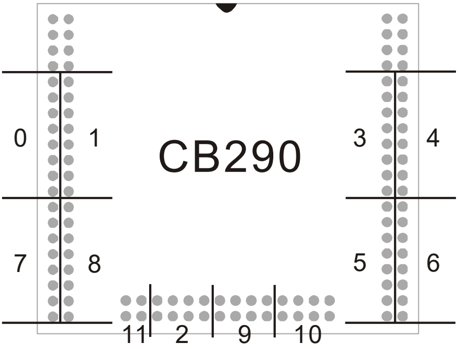
The CB290's output-only pins P24 to P55 are in a high impedance state(High-Z) at power ON. You must use `Set OutOnly On` to enable the pins if you wish to use them.

```
                Set  Outonly  On
```

This command only works with CB290 Rev. B. The revision number can be found on the bottom side of the CB290 module.

The Set Outonly command actually toggles virtual Port 88 to enable the output-only pins. If your program accidentally uses P88, you will see strange behavior on the output-only pins. Please do not access P88 in your Basic or Ladder Logic programs.

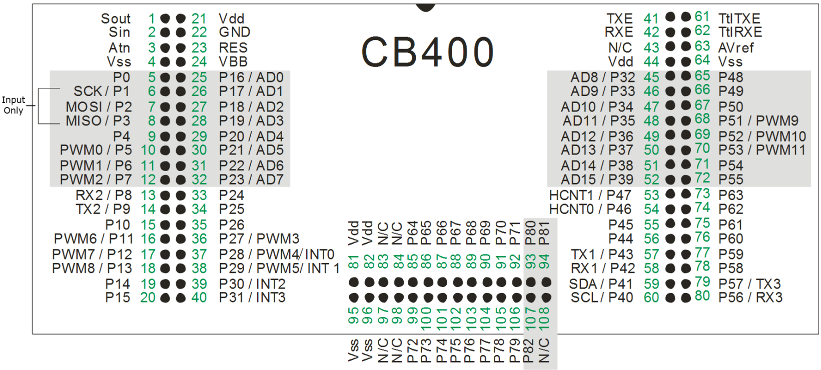




Port Blocks

CB400

The CB400 is a package of 108 pins of which 80 can be used as I/O ports. It has more I/O ports than the CB405, but less memory.



	CB400	CB405
I/O	83	64
RAM	7K (BASIC 6K, LADDER 1K)	110K (BASIC 51K, LADDER 4K) (HEAP 55K)
FLASH	200KB	
RS232	4 CH	
A/D	16 CH	
PWM	12 CH	

Name	Pin #	I/O	Explanation
SOUT	1	OUT	DOWNLOAD SERIAL OUTPUT
SIN	2	IN	DOWNLOAD SERIAL INPUT
ATN	3	IN	DOWNLOAD SERIAL INPUT
VSS	4, 22, 64	POWER IN	GROUND
VDD	21, 44	POWER IN	4.5V to 5.5V Power Supply
AVREF	63	IN	ADC Reference Voltage
VBB	24	POWER IN	Battery Backup (Can't be used)
RES	23	IN	RESET pin
TTLTXE	61	OUT	RS232 to TTL232 curcuit, TX contact
TTLRXE	62	IN	RS232 to TTL232 curcuit, RX contact
TXE	41	OUT	RS232 Output, +/- 12V
RXE	42	IN	RS232 Input, +/- 12V

The following I/O ports are described in blocks.

Block	Name	Pin #	I/O	Function	Explanation
0	P0	5	I/O	SPI SS	
	P1	6	Input	SPI SCK	Input Only
	P2	7	Input	SPI MOSI	Input Only
	P3	8	Input	SPI MISO	Input Only
	P4	9	I/O		
	P5	10	I/O	PWM CHANNEL 0	
	P6	11	I/O	PWM CHANNEL 1	
	P7	12	I/O	PWM CHANNEL 2	
1	P8	13	I/O	TTL232 RX2	TTLRX channel 2
	P9	14	I/O	TTL232 TX2	TTLTX channel 2
	P10	15	I/O		
	P11	16	I/O	PWM CHANNEL 6	
	P12	17	I/O	PWM CHANNEL 7	
	P13	18	I/O	PWM CHANNEL 8	
	P14	19	I/O		
	P15	20	I/O		
2	P16	25	I/O	AD CHANNEL 0	
	P17	26	I/O	AD CHANNEL 1	
	P18	27	I/O	AD CHANNEL 2	
	P19	28	I/O	AD CHANNEL 3	
	P20	29	I/O	AD CHANNEL 4	
	P21	30	I/O	AD CHANNEL 5	
	P22	31	I/O	AD CHANNEL 6	
	P23	32	I/O	AD CHANNEL 7	
3	P24	33	I/O	Co-processor SCL	*
	P25	34	I/O	Co-processor SDA	*
	P26	35	I/O	Co-processor INT	*
	P27	36	I/O	PWM3	
	P28	37	I/O	PWM4 / INT0	
	P29	38	I/O	PWM5 / INT1	
	P30	39	I/O	INT2	
	P31	40	I/O	INT3	

* Communication line for connecting to the coprocessor (Please try to save these pins for future coprocessor communication ports.)

4	P32	45	I/O	AD CHANNEL 8	
	P33	46	I/O	AD CHANNEL 9	
	P34	47	I/O	AD CHANNEL 10	
	P35	48	I/O	AD CHANNEL 11	
	P36	49	I/O	AD CHANNEL 12	
	P37	50	I/O	AD CHANNEL 13	
	P38	51	I/O	AD CHANNEL 14	
	P39	52	I/O	AD CHANNEL 15	

5	P40	60	I/O	SCL	CUNET Clock
	P41	59	I/O	SDA	CUNET Data
	P42	58	I/O	RX1	TTLRX Channel 1
	P43	57	I/O	TX1	TTLTX Channel 1
	P44	56	I/O		
	P45	55	I/O		
	P46	54	I/O	HCNT0	High-Speed Counter Ch 0
	P47	53	I/O	HCNT1	High-Speed Counter Ch 1

6	P48	65	I/O		
	P49	66	I/O		
	P50	67	I/O		
	P51	68	I/O	PWM Channel 9	
	P52	69	I/O	PWM Channel 10	
	P53	70	I/O	PWM Channel 11	
	P54	71	I/O		
	P55	72	I/O		

7	P56	80	I/O	RX3	TTLRX Channel3
	P57	79	I/O	TX3	TTLTX Channel 3
	P58	78	I/O		
	P59	77	I/O		
	P60	76	I/O		
	P61	75	I/O		
	P62	74	I/O		
	P63	73	I/O		

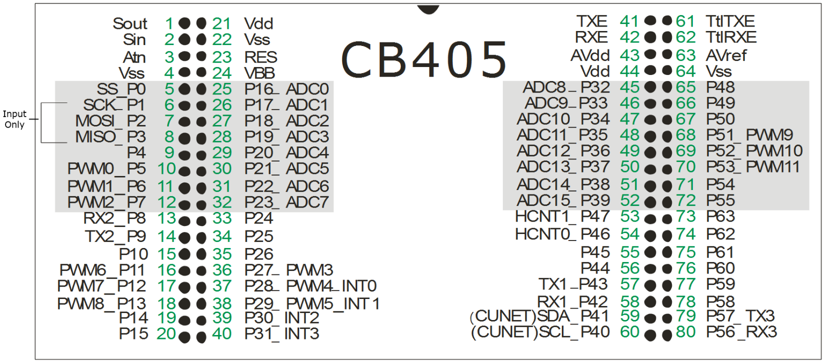
8	P64	85	I/O		
	P65	86	I/O		
	P66	87	I/O		
	P67	88	I/O		
	P68	89	I/O		
	P69	90	I/O		
	P70	91	I/O		
	P71	92	I/O		

9	P72	99	I/O		
	P73	100	I/O		
	P74	101	I/O		
	P75	102	I/O		
	P76	103	I/O		
	P77	104	I/O		
	P78	105	I/O		
	P79	106	I/O		

	P80	93	I/O		
	P81	94	I/O		
	P82	104	I/O		

CB405

The CB405 is a package of 80 pins of which 64 can be used as I/O ports. It has a battery-backup-capable 55KB of memory. The CB405 does not have an internal 5V regulator.



Name	Pin #	I/O	Explanation
SOUT	1	OUT	DOWNLOAD SERIAL OUTPUT
SIN	2	IN	DOWNLOAD SERIAL INPUT
ATN	3	IN	DOWNLOAD SERIAL INPUT
VSS	4, 22, 64	POWER IN	GROUND
VDD	21, 44	POWER IN	4.5V to 5.5V Power Supply
AVDD	43	POWER IN	ADC power
AVREF	63	IN	ADC Reference Voltage
VBB	24	POWER IN	Battery Backup
RES	23	IN	RESET pin
TTLTXE	61	OUT	RS232 to TTL232 curcuit, TX contact
TTLRXE	62	IN	RS232 to TTL232 curcuit, RX contact
TXE	41	OUT	RS232 Output, +/- 12V
RXE	42	IN	RS232 Input, +/- 12V

The **following** I/O ports are described in blocks.

Block	Name	Pin #	I/O	Function	Explanation
0	P0	5	I/O	SPI SS	
	P1	6	Input	SPI SCK	Input Only
	P2	7	Input	SPI MOSI	Input Only
	P3	8	Input	SPI MISO	Input Only
	P4	9	I/O		
	P5	10	I/O	PWM CHANNEL 0	
	P6	11	I/O	PWM CHANNEL 1	
	P7	12	I/O	PWM CHANNEL 2	
1	P8	13	I/O	TTL232 RX2	TTLRX channel 2
	P9	14	I/O	TTL232 TX2	TTLTX channel 2
	P10	15	I/O		
	P11	16	I/O	PWM CHANNEL 6	
	P12	17	I/O	PWM CHANNEL 7	
	P13	18	I/O	PWM CHANNEL 8	
	P14	19	I/O		
	P15	20	I/O		
2	P16	25	I/O	AD CHANNEL 0	
	P17	26	I/O	AD CHANNEL 1	
	P18	27	I/O	AD CHANNEL 2	
	P19	28	I/O	AD CHANNEL 3	
	P20	29	I/O	AD CHANNEL 4	
	P21	30	I/O	AD CHANNEL 5	
	P22	31	I/O	AD CHANNEL 6	
	P23	32	I/O	AD CHANNEL 7	
3	P24	33	I/O	Co-processor SCL	*
	P25	34	I/O	Co-processor SDA	*
	P26	35	I/O	Co-processor INT	*
	P27	36	I/O	PWM3	
	P28	37	I/O	PWM4 / INT0	
	P29	38	I/O	PWM5 / INT1	
	P30	39	I/O	INT2	
	P31	40	I/O	INT3	

* Communication line for connecting to the coprocessor (Please try to save these pins for future coprocessor communication ports.)

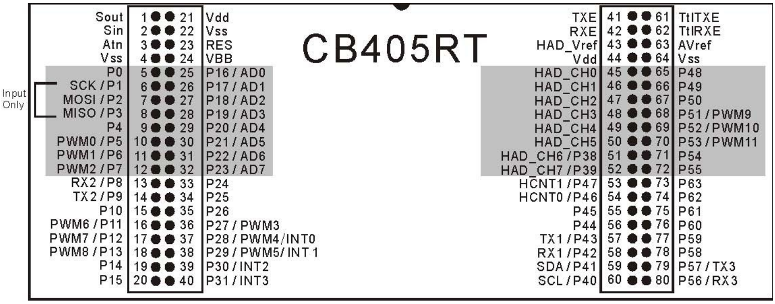
Block	Name	Pin #	I/O	Function	Explanation
4	P32	45	I/O	AD CHANNEL 8	
	P33	46	I/O	AD CHANNEL 9	
	P34	47	I/O	AD CHANNEL 10	
	P35	48	I/O	AD CHANNEL 11	
	P36	49	I/O	AD CHANNEL 12	
	P37	50	I/O	AD CHANNEL 13	
	P38	51	I/O	AD CHANNEL 14	
	P39	52	I/O	AD CHANNEL 15	
5	P40	60	I/O	SCL	CUNET clock pin
	P41	59	I/O	SDA	CUNET data pin
	P42	58	I/O	RX1	TTLRX channel 1
	P43	57	I/O	TX1	TTLTX channel 1
	P44	56	I/O		
	P45	55	I/O		
	P46	54	I/O	HCNT0	High-speed Counter 0
	P47	53	I/O	HCNT1	High-speed Counter 1
6	P48	65	I/O		
	P49	66	I/O		
	P50	67	I/O		
	P51	68	I/O	PWM CHANNEL 9	
	P52	69	I/O	PWM CHANNEL 10	
	P53	70	I/O	PWM CHANNEL 11	
	P54	71	I/O		
	P55	72	I/O		
7	P56	80	I/O	RX3	TTLRX channel 3
	P57	79	I/O	TX3	TTLTX channel 3
	P58	78	I/O		
	P59	77	I/O		
	P60	76	I/O		
	P61	75	I/O		
	P62	74	I/O		
	P63	73	I/O		

CB405RT

- CB405 core module
- equipped with a built-in accurate RTC (DS3231)
- equipped with a built-in 16-bit A/D converter (8 channels)

The CB405RT is a product, which adds an RTC and a 16-bit ADC to the original CB405. The external dimensions and pinout are the same as the CB405 with the exception that ports P32 to P37 are used for the 16-bit ADC and therefore cannot be used.

Refer chapter 13 for more information.



The following describes all pins excluding I/O ports.

Name	Pin No.	I/O	Description
SOUT	1	OUT	SERIAL OUTPUT for DOWNLOAD
SIN	2	IN	SERIAL OUTPUT for DOWNLOAD
ATN	3	IN	SERIAL OUTPUT for DOWNLOAD
VSS	4, 22, 64	POWER IN	GROUND
VDD	21, 44	POWER IN	Supplies 4.5V to 5.5V
HAD_Vref	43	IN	Supplies reference voltage to the16-bit ADC
AVREF	63	IN	Supplies reference voltage to the10-bit ADC
VBB	24	POWER IN	Battery connection pin for battery backup
RES	23	IN	RESET pin
TTLTXE	61	OUT	RS232 to TTL232 conversion circuit, TX connection terminal
TTLRXE	62	IN	RS232 to TTL232 conversion circuit, RX connection terminal
TXE	41	OUT	RS232 output terminal, connects with external RS232 port
RXE	42	IN	RS232 input terminal, connects with external RS232 port

The following describes the I/O ports, listed by their port blocks.

Block	Name	Pin No.	I/O	Special Function	Description
0	P0	5	I/O	SPI’s SS	
	P1	6	Input	SPI’s SCK	Input only pin
	P2	7	Input	SPI’s MOSI	Input only pin
	P3	8	Input	SPI’s MISO	Input only pin
	P4	9	I/O		
	P5	10	I/O	PWM CHANNEL 0	
	P6	11	I/O	PWM CHANNEL 1	
	P7	12	I/O	PWM CHANNEL 2	
1	P8	13	I/O	TTL232 RX2	
	P9	14	I/O	TTL232 TX2	
	P10	15	I/O		
	P11	16	I/O	PWM CHANNEL 6	
	P12	17	I/O	PWM CHANNEL 7	
	P13	18	I/O	PWM CHANNEL 8	
	P14	19	I/O		
	P15	20	I/O		
2	P16	25	I/O	AD CHANNEL 0	
	P17	26	I/O	AD CHANNEL 1	
	P18	27	I/O	AD CHANNEL 2	
	P19	28	I/O	AD CHANNEL 3	
	P20	29	I/O	AD CHANNEL 4	
	P21	30	I/O	AD CHANNEL 5	
	P22	31	I/O	AD CHANNEL 6	
	P23	32	I/O	AD CHANNEL 7	
3	P24	33	I/O		
	P25	34	I/O		
	P26	35	I/O		
	P27	36	I/O	PWM3	
	P28	37	I/O	PWM4 / INT0	
	P29	38	I/O	PWM5 / INT1	
	P30	39	I/O	INT2	
	P31	40	I/O	INT3	

This part has been changed from the original CB405.

Block	**Name**	**Pin No.**	**I/O**	**Special Function**	**Description**
Belongs to no block	HADCH0	45	INPUT	16bit AD CHANNEL 0	
	HADCH1	46	INPUT	16bit AD CHANNEL 1	
	HADCH2	47	INPUT	16bit AD CHANNEL 2	
	HADCH3	48	INPUT	16bit AD CHANNEL 3	
	HADCH4	49	INPUT	16bit AD CHANNEL 4	
	HADCH5	50	INPUT	16bit AD CHANNEL 5	
	HADCH6 / P38	51	I/O	16bit AD CHANNEL 6	P38 usable
	HADCH7 / P39	52	I/O	16bit AD CHANNEL 7	P39 usable

Note: Since ports P32 to P37 are internally connected to the RTC and the 16-bit ADC, if a program uses these ports, the RTC and ADC chip will not function properly. Also, if ports P38 and P39 are used for output, HADCH6 and HADCH7 cannot be used, respectively.

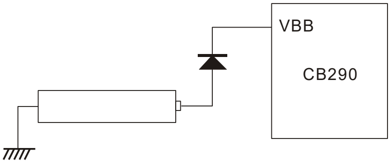
5	P40	60	I/O	SCL	CUNET clock
	P41	59	I/O	SDA	CUNET data
	P42	58	I/O	RX1	TTLRX channel 1
	P43	57	I/O	TX1	TTLTX channel 1
	P44	56	I/O		
	P45	55	I/O		
	P46	54	I/O	HCNT0	High-speed counter channel 0
	P47	53	I/O	HCNT1	High-speed counter channel 1

6	P48	65	I/O		
	P49	66	I/O		
	P50	67	I/O		
	P51	68	I/O	PWM CHANNEL 9	
	P52	69	I/O	PWM CHANNEL 10	
	P53	70	I/O	PWM CHANNEL 11	
	P54	71	I/O		
	P55	72	I/O		

7	P56	80	I/O	RX3	TTLRX channel 3
	P57	79	I/O	TX3	TTLTX channel 3
	P58	78	I/O		
	P59	77	I/O		
	P60	76	I/O		
	P61	75	I/O		
	P62	74	I/O		
	P63	73	I/O		

How to connect a battery to CB290 / CB405

When a supercapacitor is connected to VBB, the memory can be maintained anywhere from a couple days to a couple weeks once powered off. The CB290/CB405 consumes about 15-20mA of current when idling. For a longer backup period, a battery pack can be used. A protection diode, as shown below, is necessary when using a battery, as the device normally attempts to charge a capacitor through VBB. Due to the relatively high standby current for battery backup, it is recommended to keep the device powered if possible and only maintain battery backup for short periods in the case of an emergency.



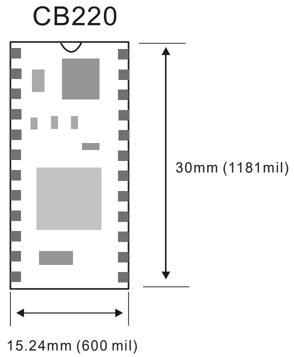
Power Features

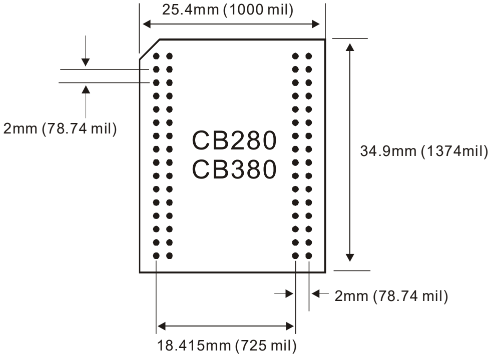
●Operating Voltage : 4.5V to 5.5V
●Operating Clock : 18.432MHz
●I/O Port Source Current : 20mA
●I/O Port Sink Current : 25mA
●Operating Temperature : -40 to 125 Degrees(Celcius)
●Maintenance Temperature: -60 to 140 Degrees(Celcius)
●Operating Humidity : 5 to 95% RH
(Keep the board's surface dry when in use)

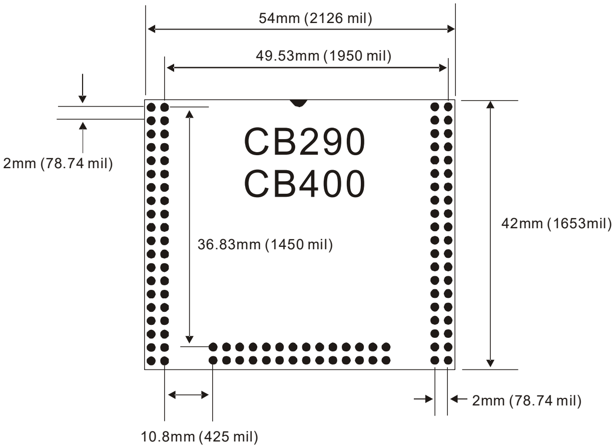
Additional Information

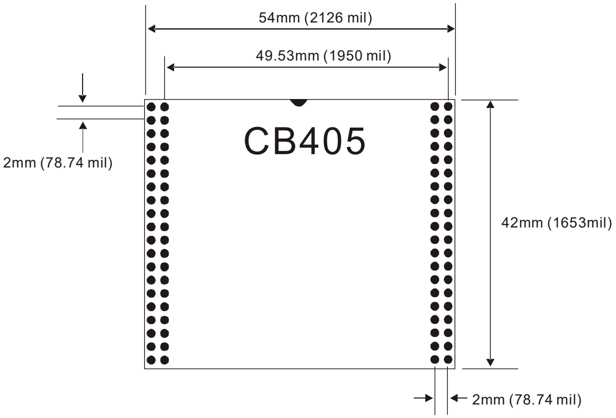
If a Cubloc module is supplied with power above the recommended voltage, the device can be destroyed. Also please be careful to avoid electrostatic discharge, as it too can destroy the device. Please be aware that P1 is an input-only pin. To reduce accidental power drain, please set unused pins to input. All I/O ports are set to input as default at power on. When not using SIN, SOUT, and ATN, please do not connect them to anything.

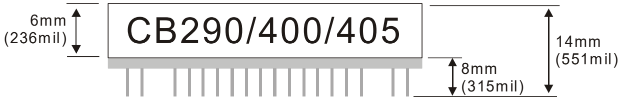
Dimensions


CB220
30mm (1181mil)
15.24mm (600 mil)

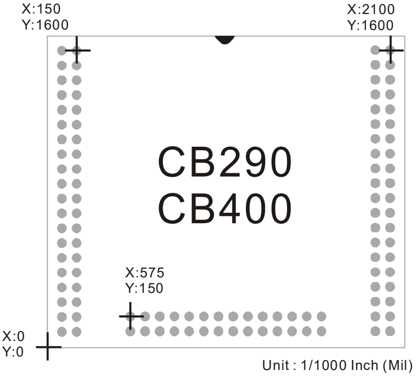

25.4mm (1000 mil)
2mm (78.74 mil)
CB280
CB380
34.9mm (1374mil)
2mm (78.74 mil)
18.415mm (725 mil)

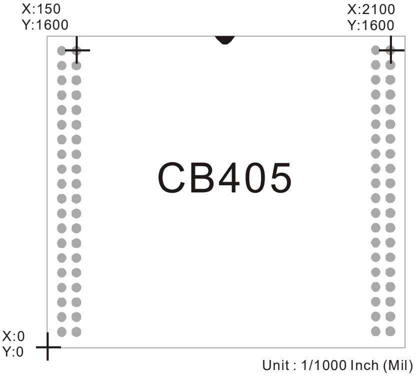

54mm (2126 mil)
49.53mm (1950 mil)
2mm (78.74 mil)
CB290
CB400
36.83mm (1450 mil)
42mm (1653mil)
2mm (78.74 mil)
10.8mm (425 mil)

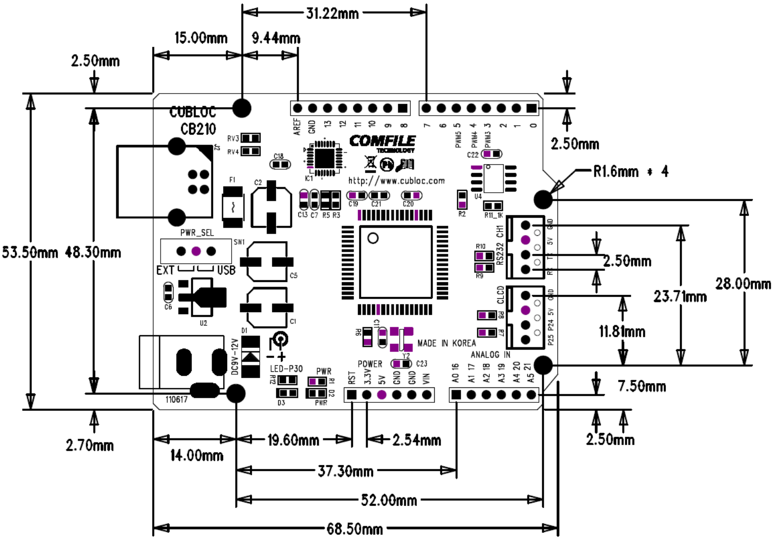

54mm (2126 mil)
49.53mm (1950 mil)
2mm (78.74 mil)
CB405
42mm (1653mil)
2mm (78.74 mil)


6mm
(236mil)
CB290/400/405
14mm
(551mil)
8mm
(315mil)

Please refer to the diagram below for PCB design. The numbers are offsets based on location 0, 0 (from the top left corner of the module's internal PCB; not the external plastic case).






31.22mm
15.00mm
9.44mm
2.50mm
CUBLOC
CB210
COMFILE
TECHNOLOGY
http://www.cubloc.com
AREF
GND
13
12
11
10
9
8
7
6
PWM5 5
PWM4 4
PWM3 3
2
1
0
2.50mm
R1.6mm * 4
PWR_SEL
EXT
USB
53.50mm
48.30mm
RS232 CH1
CLCD
GND
5V
TX
RX
P24
P25
2.50mm
23.71mm
28.00mm
11.81mm
MADE IN KOREA
ANALOG IN
POWER
RST
3.3V
5V
GND
GND
VIN
A0 16
A1 17
A2 18
A3 19
A4 20
A5 21
DC9V-12V
LED-P30
PWR
110617
7.50mm
2.50mm
2.70mm
14.00mm
19.60mm
2.54mm
37.30mm
52.00mm
68.50mm

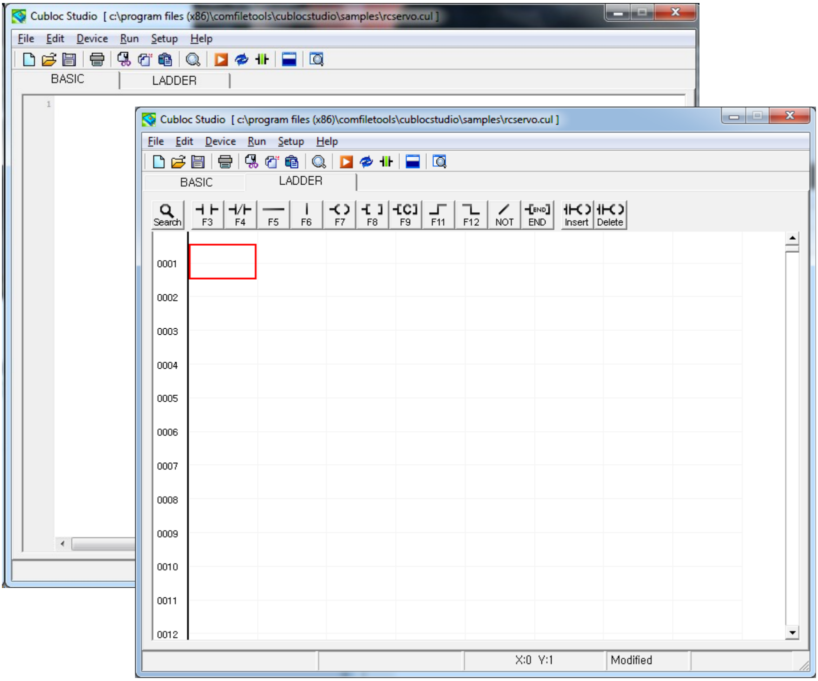
CB210

MEMO

Chapter 3: Cubloc Studio

About Cubloc Studio

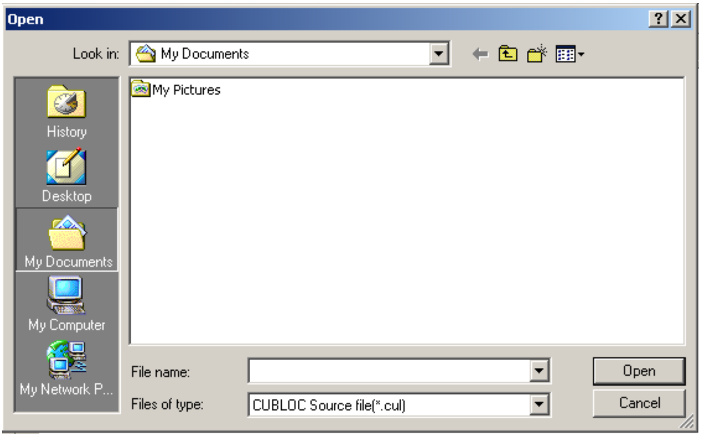
After installing Cubloc Studio and executing it, you will see the following screen.


Cubloc Studio [c:\program files (x86)\comfiletools\cublocstudio\samples\rcservo.cul]
File Edit Device Run Setup Help
BASIC
LADDER
1
Cubloc Studio [c:\program files (x86)\comfiletools\cublocstudio\samples\rcservo.cul]
File Edit Device Run Setup Help
BASIC
LADDER
Search
F3
F4
F5
F6
F7
F8
F9
F11
F12
NOT
END
Insert
Delete
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
X:0 Y:1
Modified

You will see that, at first, Cubloc Studio will default to the BASIC editor.

If you press F2, the screen will change to the Ladder Logic editor and if you press F1, it will switch back to the BASIC editor.

Source files are saved as two files under the file extensions .CUL and .CUB. If you need to backup or move source files, you must keep BOTH of these files together.



Look in: My Documents

File name:

Files of type: CUBLOC Source file(*.cul)

When opening a file, you will only see .CUL files. (.CUB files are not displayed, but they are in the same folder). When you open a .CUL file, Cubloc Studio automatically opens CUB file.

The source code can only be saved on the PC. Source code downloaded to the Cubloc module **can not** be uploaded back to the PC.

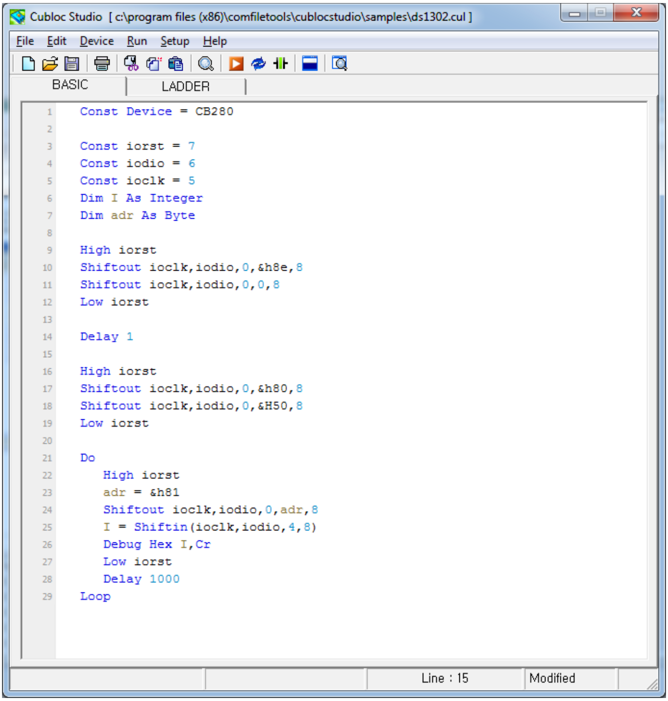
IMPORTANT

All Cubloc modules implement code obfuscation. By obfuscating the downloaded program data, the code is safe from any attempt to read part of the chip’s memory and copy the source code.

When you press the RUN button (or CTRL-R), Save, Compile, Download, and Execute are all automatically processed. Ladder Logic and BASIC both are compiled with one RUN button. If an error is found during compilation, the cursor will relocate to the error position.

Creating BASIC Code

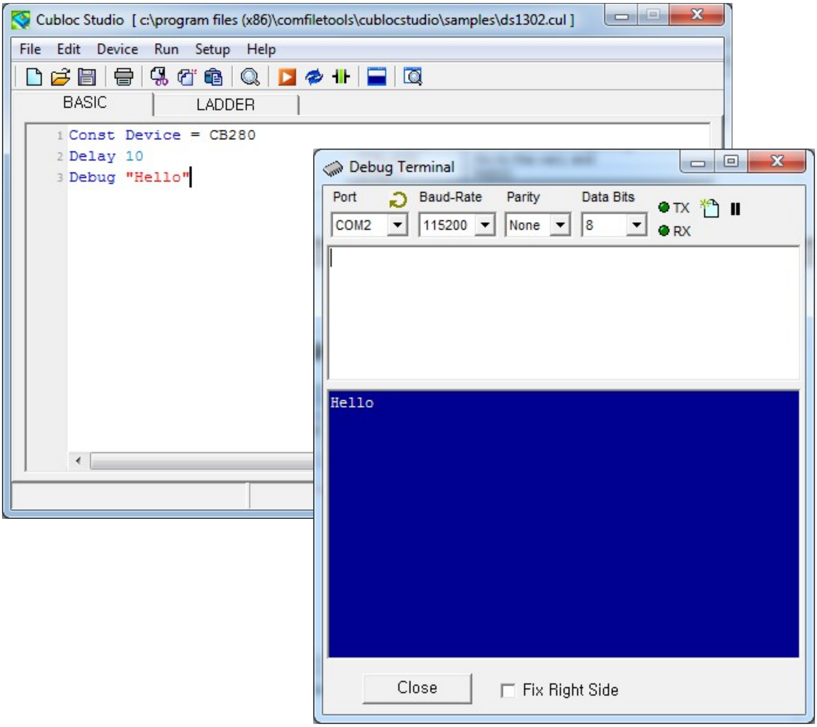
You can create BASIC code as shown below. Cubloc Studio's text editor is similar to most text editors, and performs syntax highlighting of certain commands.


Cubloc Studio [c:\program files (x86)\comfiletools\cublocstudio\samples\ds1302.cul]
File Edit Device Run Setup Help
BASIC
LADDER
Line : 15
Modified

```
Const Device = CB280

Const iorst = 7
Const iodio = 6
Const ioclk = 5
Dim I As Integer
Dim adr As Byte

High iorst
Shiftout ioclk,iodio,0,&h8e,8
Shiftout ioclk,iodio,0,0,8
Low iorst

Delay 1

High iorst
Shiftout ioclk,iodio,0,&h80,8
Shiftout ioclk,iodio,0,&H50,8
Low iorst

Do
   High iorst
   adr = &h81
   Shiftout ioclk,iodio,0,adr,8
   I = Shiftin(ioclk,iodio,4,8)
   Debug Hex I,Cr
   Low iorst
   Delay 1000
Loop
```

Shortcut	Explanation
CTRL-Z	UNDO
CTRL-O	OPEN
CTRL-S	SAVE
CTRL-C	COPY
CTRL-X	CUT
CTRL-V	PASTE
CTRL-F	FIND
CTRL-HOME	Go to the very beginning
CTRL-END	Go to the very end
CTRL-Y	REDO

Debugging



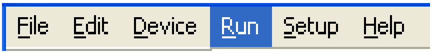
```
Const Device = CB280
Delay 10
Debug "Hello"
```

Port: COM2, Baud-Rate: 115200, Parity: None, Data Bits: 8

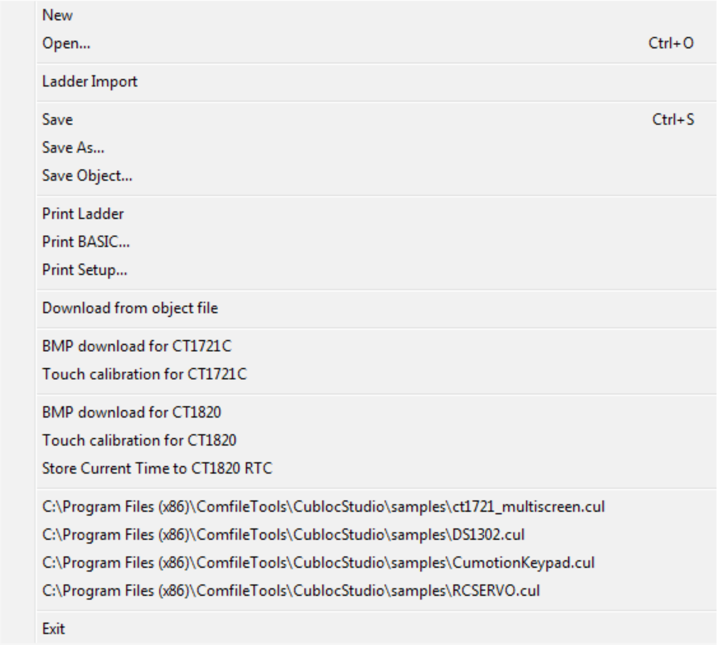
```
Hello
```

As shown in the screenshot above, the `Debug` statement can be used to monitor your BASIC program while it's running. Be aware that you cannot debug and monitor Ladder Logic simultaneously. You must remove all `Debug` statements or comment them out with an apostrophe before attempting to use Ladder Logic monitoring. Another option is to use the command `Set Debug Off` which will automatically ignore any `Debug` statements.

Menus



File Menu


New
Open... Ctrl+O
Ladder Import
Save Ctrl+S
Save As...
Save Object...
Print Ladder
Print BASIC...
Print Setup...
Download from object file
BMP download for CT1721C
Touch calibration for CT1721C
BMP download for CT1820
Touch calibration for CT1820
Store Current Time to CT1820 RTC
C:\Program Files (x86)\ComfileTools\CublocStudio\samples\ct1721_multiscreen.cul
C:\Program Files (x86)\ComfileTools\CublocStudio\samples\DS1302.cul
C:\Program Files (x86)\ComfileTools\CublocStudio\samples\CumotionKeypad.cul
C:\Program Files (x86)\ComfileTools\CublocStudio\samples\RCSERVO.cul
Exit

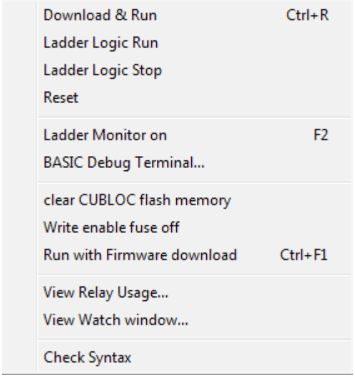
Menu	Explanation
New	Create new file.
Open	Open file.
Ladder Import	Import Ladder Logic part of a Cubloc program.
Save	Save current file.
Save As	Save current file under a different name.
Save Object	Save current program as an object file. Use this to protect your source code. An object file is a strictly binary format file so others cannot easily reverse engineer it. You can use "Download from Object File" to download an object file to Cubloc.
Print Ladder	Print Ladder Logic section only.
Print BASIC	Print BASIC section only.
Print Setup	Setup printer for printing Ladder Logic editor.
Download from Object file	Download an object file to the Cubloc module.
BMP download for	Download Bitmap to CT1721C

CT1721C	
Touch calibration for CT1721C	Calibrate CT1721C's Touch-screen
BMP download for CT1820	Download Bitmap to CT1820
Touch calibration for CT1721C	Calibrate CT1820's Touch-screen
Store Current Time to CT1820 RTC	Synchronize this PC's current time with the CT1820's RTC
Last 4 Files Edited	View last 4 files edited.
Exit	Exit Cubloc Studio

Device Menu

If a `Const Device` statement does not exist in your source code, you can use the "Device" menu can create a `Const Device` statement at the very beginning of your source code. If a `Const Device` statement already exists, using the "Device" menu will simply replace the `Const Device` statement.

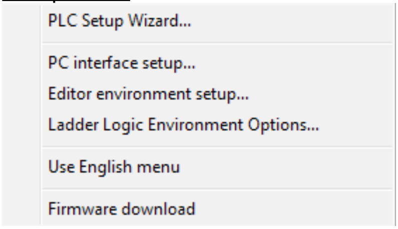
Run Menu



Menu	Explanation
Download & Run	Compile BASIC and Ladder Logic, download to Cubloc module if there are no errors, and restart the program automatically. To disable automatic restart, please go to **Setup->Studio Option** to change.
Ladder Logic Run	Continue execution of Ladder Logic while monitoring
Ladder Logic Stop	Stop execution of Ladder Logic while monitoring
Reset	Reset the Cubloc module.
Ladder Monitor on	Start Ladder Logic monitoring
BASIC Debug Terminal	Open BASIC Debug Terminal Window. This window opens automatically when there's a `Debug` statement in the source code.
Clear Cubloc's Flash	Clear Cubloc's Flash Memory.

Memory	
Write enable fuse off	This will turn off the download function for a Cubloc Core module to protect against noisy environments where the flash memory can be affected. Once you choose this menu, you will be unable to download new programs to your Cubloc module. You will be able to download again after a Firmware Download.
Run with Firmware download	View relays in use by Ladder Logic after compilation.
View Relay Usage	See, at a glance, the status of relays in a separate window.
View Watch window	See, at a glance, the status of relays in a separate window.
Check Syntax	Check the source code's syntax

Setup Menu



Menu	Explanation
PLC Setup Wizard	Wizard to automatically generate BASIC code.
PC Interface Setup	Setup the RS232 COM PORT for Download/Monitor. Select COM1 through COM4.
Editor Environment Setup	Configuration options for the BASIC editor.
Ladder Logic Environment Options	Configuration options for Ladder Logic.
Use English menu	Change the language of the menus to English
Firmware Download	Download Firmware to Cubloc. Please use this to download firmware to Cubloc manually.

Chapter 4: Cubloc BASIC Language

IMPORTANT

The device being used must be declared before using BASIC or Ladder Logic. Below is an example of declaring the Cubloc CB220 module.

```
Const Device = CB220    ' Use CB220.
```

This should be the first line of a program. When this command is not used, the CB220 model will be assumed.

```
Const Device = CT1720   ' Use CT1721.
Const Device = CB280    ' Use CB280.
```

Cubloc BASIC Features

Cubloc BASIC supports functions and subroutines.

The user is able to create subroutines and functions to organize their programs. Using subroutines and functions allows the user to reuse code, and organize programs for better readability.

```
Function  SUM( A As Integer,  B As Integer) As Integer
              Dim RES As Integer
              RES = A + B
              SUM = RES
End Function
```

Calculations can be done within conditional statements such as `If`, `While`, etc…

```
If  ((A + 1) = 100) Then GoTo ABC

If  ((A + 1) = 100) And (B / 100 = 20) OR C = 3 Then GoTo ABC
```

Hardware RS-232 Communication

Cubloc uses a hardware RS-232 UART instead of software RS-232 UART allowing real-time processing to continue during RS-232 operations.

A graphic LCD library is provided.

Cubloc provides a complete graphic LCD library for the Comfile GHLCD product. Boxes, lines, circles, and other graphics commands are easily implemented in a few lines of code.

Various Communication Protocols are supported.

CUNET : Display peripherals such as character LCDs
RS232 : up to 4 channels
MODBUS : built-in slave functions
I2C : I2C commands supported (`I2CRead I2CWrite`)
SPI : SPI commands supported (`ShiftIn`, `ShiftOut`)
PAD: Keypad, touchpad supported.

Simple BASIC program

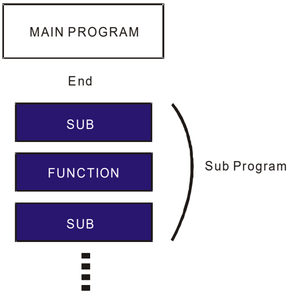
Below is an example of a simple BASIC program with a Do…Loop statement.

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

This program outputs the increasing binary value of A to Ports P0-P7. The next program uses a function to accomplish the same task:

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

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

By placing A=A+1 in a function, the user will be able to separate one big program into small manageable pieces. As you can see here, the main program ends at the End command, and functions are added afterwards.



Sub and Function

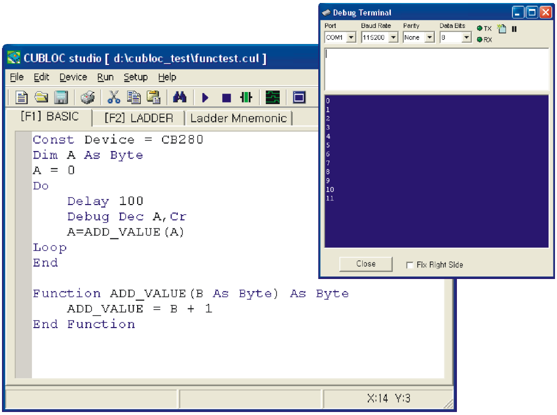
For subroutines, you can either use `Sub` or `Function`. `Function` can return a value, but `Sub` can't.

```
Sub SubName (Param1 As DataType [,ParamX As DataType][,…])
          Statements
          [Exit sub]          ' Exit during sub-routine
End Sub

Function FunctionName (Param1 As DataType [,…])[As ReturnDataType]
          Statements
         [Exit Function]      ' Exit during sub-routine
End Function
```

To return values using `Function`, simply store the final value as the name of the function as shown below:

```
Function ADD_VALUE(B As Byte) As Byte
         ADD_VALUE = B + 1    ' Return B+1.
End Function
```

DEMO PROGRAM



```
Const Device = CB280
Dim A As Byte
A = 0
Do
    Delay 100
    Debug Dec A,Cr
    A=ADD_VALUE(A)
Loop
End

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

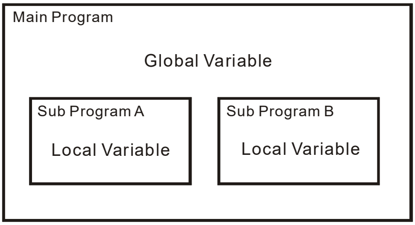
Port: COM1 Baud Rate: 115200 Parity: None Data Bits: 8

```
0
1
2
3
4
5
6
7
8
9
10
11
```

Global and Local Variables

When you declare variables inside a subroutine or a function, it is considered a "Local" variable. Local variables are created when the subroutine or function is called, and removed when the subroutine or function exits. This means that local variables will temporarily allocate data memory for the duration of the call. Local variables may only be referred to or used inside the subroutine or function in which they were declared.

On the other hand, global variables may be used anywhere in your program.



```
Dim  A  As  Integer          ' Declare A as Global Variable
LOOP1:
        A = A + 1
        Debug  Dp(A),CR      ' Display A on Debug screen
        DELAYTIME            ' Call Sub DELAYTIME
        GoTo LOOP1
        End                  ' End of Main Program

Sub  DELAYTIME()
        Dim  K  As  Integer   ' Declare K as Local Variable
        For K=0 To 10
        Next
End Sub
```

In the program above, `A` is declared as a global variable and `K` is declared as a local variable. `A` can be used anywhere in the program but `K` can only be used inside the subroutine `DELAYTIME`.

Arrays cannot be declared as local variables. **Arrays must be declared as global variables.**

Calling Subroutines

Once the subroutine is created, they can be used like any other statement.

For a subroutine, you do not need parenthesis around the parameters. Use commas to separate multiple parameters.

The example below shows how this is done:

```
DELAYTIME 100                   ' Call subroutine
End

Sub  DELAYTIME(DL As Integer)
       Dim  K  As  Integer     ' Declare K as Local Variable
       For K=0 To DL
       Next
End Sub
```

For a function, you need parenthesis around the parameters. Parenthesis are required even when there are no parameters.

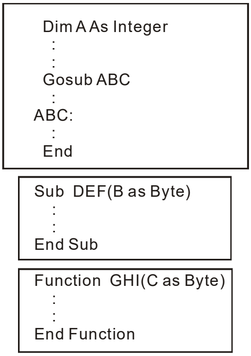
```
Dim K As Integer
   K = SUMAB(100,200) ' Call subroutine and store return value in K
   Debug Dec K,cr
End

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

Subroutine Position

Subroutines must be created after the main program. To do this, simply put `End` at the end of your main program as shown below. `End` is only required if you have subroutines

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

     End                  ' End of main program

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

Subroutines and functions are created after the `End` statement. `Gosub` subroutines must be within the main program like shown below:



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

Sub  DEF(B as Byte)
  :
  :
End Sub

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

* The `End` statement is used to differentiate between the BASIC main program and the program's subroutines. The `END` statement in Ladder Logic is used to indicate the final Ladder Logic rung.

Subroutine Parameters and Return Values

Functions may use any data type, except arrays, as parameters and return values:

```
   Dim A(10) As Integer

   Function ABC(A AS Single) as Single         ' Return Single value
   End Function

   Function ABC(A AS String * 12) as String *12 ' Return String value
   End Function

   Function ABC(A AS long)         ' Long value as a parameter
   End Function                    '  When  return  value  is  not  declared,
Long
                              ' will be used as return value.
```

Arrays cannot be used as parameters. The following is not allowed.

```
   ~~Function ARRAYUSING(A(10) AS Integer)~~ ' Arrays may not be used as
                                          ' parameters.
   ~~End Function~~
```

But you may use one element of an array as a parameter:

```
   Dim b(10) as Integer
   K = ARRAYUSING(b(10))     ' Use 10th element of array b as a parameter.

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

All subroutine parameters are passed by value, not by reference. If the parameter value is changed within a subroutine, it will not affect the variable passed to the subroutine.

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

    Sub ADDATEN(V As Integer)
           V = V + 10         ' A does not change when V is changed.
           ADDATEN = V
    End Sub
```

In contrast, some languages allow values to be passed by reference in which the actual memory address is passed to the subroutine. **Cubloc BASIC only supports passing by value.**

Too many characters in one line?

If you run out of room, you can use an underscore character (_) to go to the next line as shown here:

```
ST = "COMFILE TECHNOLOGY"
ST = "COMFILE _
            TECHNOLOGY"
```

Comments

Use an apostrophe/single-quote (') to add comments. Comments are discarded during at compile time, and will not consume any program memory.

```
ADD_VALUE = B + 1   ' Add 1 to B.(Comment)
```

Nested Subroutines

Nested subroutines are supported in Cubloc.

```
A=Floor(SQR(F))    ' Do Floor() on Sqr(F).
```

Colons

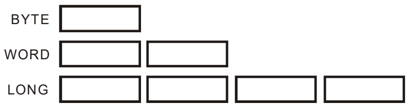
Colons cannot be used to append commands in Cubloc BASIC, as is possible in some other languages.

```
A=1: B=1 : C=1    ' Incorrect.

A=1               ' Correct.
B=1
C=1
```

Variables

There are 5 types of variables in Cubloc BASIC.

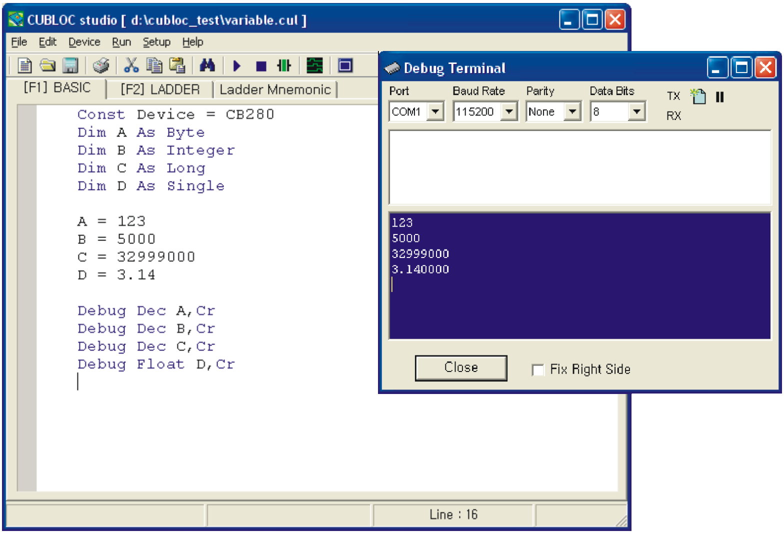
- Byte 8 bit positive number, 0 to 255
- Integer 16 bit positive number, 0 to 65535
- Long 32 bit positive/negative number,
  (-2147483648 to +2147483647)
- Single 32 bit floating point number,
  (-3.402823E+38 to 3.402823E+38)
- String character string, 0 TO 127 bytes



*For storing negative numbers, please use LONG or SINGLE.
Use the `Dim` statement for declaring variables as shown below:

```
Dim  A  As  Byte                    ' Declare A as Byte.
Dim  B  As  Integer, C As Byte      ' Commas may NOT be used.
Dim  ST1  As String * 12            ' Declare a String of 12 bytes.
Dim  ST2  As String                 ' String is 64 bytes default.
Dim  AR(10)  As Byte                ' Declare AR as a Byte Array.
Dim  AK(10,20)  As Integer          ' Declare AK as a 2-Dimensional Array
Dim  ST(10)  As String*10           ' Declare a String Array
```

DEMO PROGRAM



```
Const Device = CB280
Dim A As Byte
Dim B As Integer
Dim C As Long
Dim D As Single

A = 123
B = 5000
C = 32999000
D = 3.14

Debug Dec A,Cr
Debug Dec B,Cr
Debug Dec C,Cr
Debug Float D,Cr
```

```
123
5000
32999000
3.140000
```

Var Statement (Same as Dim)

Var can be used in the place of Dim to declare variables. Below are examples of how to use Var:

```
A     Var     Byte              ' Declare A as BYTE.
ST1   Var     String * 12       ' Declare ST1 as String of 12 bytes.
AR    Var     Byte(10)          ' Declare AR as Byte Array of 10.
AK    Var     Integer(10,20)    ' Declare AK as 2-D Integer Array
ST    Var     String *12 (10)   ' Declare String Array
```

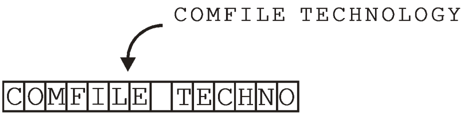
String

A String can be a maximum of 127 bytes in size. When the size is not declared, a default size of 64 bytes will be assumed.

```
Dim  ST   As  String * 14 ' For maximum usage of 14 bytes
Dim  ST2  As  String      ' Set as 64 byte String variable
```

When declaring a String as 14 bytes, an additional byte is allocated by the processor to store a NULL terminating character. When storing "COMFILE TECHNOLOGY" in a 14 byte String, the last 4 characters (bytes) will be truncated.

```
Dim  ST  As  String * 14
ST = "COMFILE TECHNOLOGY" '  "LOGY" is not stored
```



"LOGY" does not fit

In Cubloc BASIC, only the double-quote character(") can be used for String constants and Strings; a single-quote characters (') can not be used.

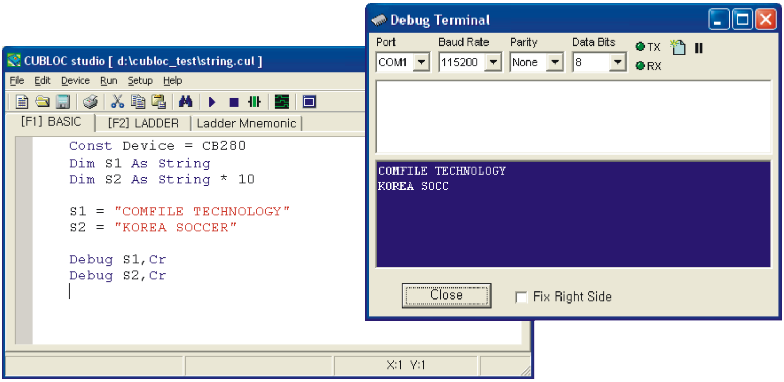
```
ST = "COMFILE " TECHNOLOGY" ' (") cannot be used inside the String.
ST = "COMFILE ' TECHNOLOGY" ' (') cannot be used inside the String.
ST = "COMFILE , TECHNOLOGY" ' (,) cannot be used inside the String.
```

Furthermore, Cubloc BASIC does not support character escape sequences. Use CHR(&H22) to include a double-quote character(") in a String. Commas and single-quotes also cannot be directly declared in a String. Use CHR(&H27) to included a single-quote character (') and CHR(&H2C) to included a comma (,).

Example for printing to an LCD:

```
Print Chr(&H22), "COMFILE TECHNOLOGY",Chr(&H22)   ' (")
Print Chr(&H27), "COMFILE " TECHNOLOGY",Chr(&H27) ' (') Apostrophe
```

DEMO PROGRAM



```
Const Device = CB280
Dim S1 As String
Dim S2 As String * 10

S1 = "COMFILE TECHNOLOGY"
S2 = "KOREA SOCCER"

Debug S1,Cr
Debug S2,Cr
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
COMFILE TECHNOLOGY
KOREA SOCC
```

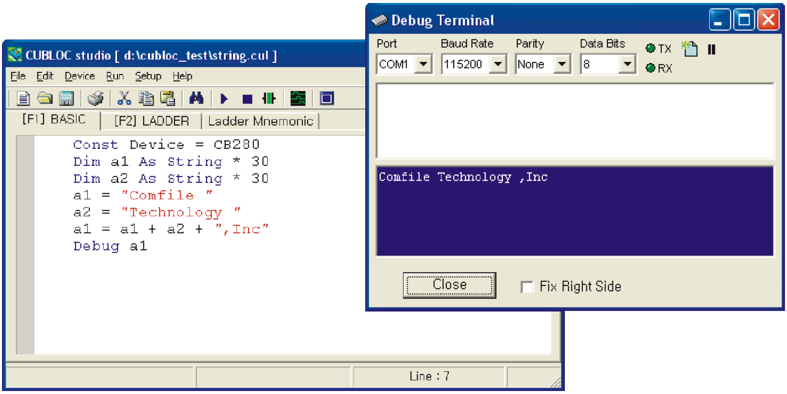
Concatenate Multiple Strings

To concatenate multiple strings together, use the + operator as shown below:

```
Dim a1 As String * 30
Dim a2 As String * 30
a1 = "Comfile "
a2 = "Technology "
a1 = a1 + a2 + ",Inc"
Debug a1,cr
```

The above program will show "Comfile Technology, Inc" on the debug screen.

DEMO PROGRAM



```
Const Device = CB280
Dim a1 As String * 30
Dim a2 As String * 30
a1 = "Comfile "
a2 = "Technology "
a1 = a1 + a2 + ",Inc"
Debug a1
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
Comfile Technology ,Inc
```

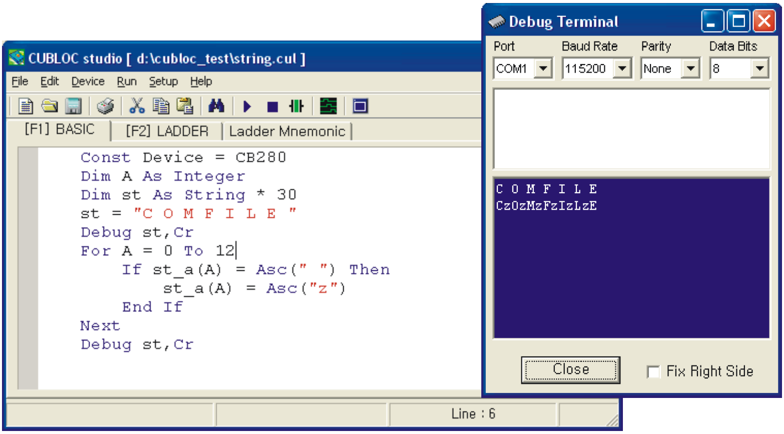
How to Access Individual Characters within a String

You can treat strings as a `Byte` array. Simply append "_A" after the name of your string variable as shown below:

```
Dim ST1 As String * 12       ' ST1_A Array is created at the same time.
ST1 = "123"
ST1_A(0) = ASC("A")          ' Store A in the first character of ST1.
```

When you declare `Dim St1 as String * 12`, `St1_A(12)` is also declared automatically by the Real-Time Operating System (RTOS). The string and the array use the same memory space. Whether you use the string or the array, you are still accessing same memory location.

The example below shows how to convert blank characters to z.



```
Const Device = CB280
Dim A As Integer
Dim st As String * 30
st = "C O M F I L E "
Debug st,Cr
For A = 0 To 12
    If st_a(A) = Asc(" ") Then
        st_a(A) = Asc("z")
    End If
Next
Debug st,Cr
```

Port: COM1 Baud Rate: 115200 Parity: None Data Bits: 8

```
C O M F I L E
CzOzMzFzIzLzE
```

This feature cannot be used with `String` arrays.

About Variable Memory Space

In the CB220 and CB280, 2KB (2048 bytes) of data memory is available. However, the memory is not dedicated entirely to variables. Some data memory is reserved for use by peripherals like the DISPLAY and the RS232 buffers. An additional 80 bytes are used for the `Debug` statement.

Subroutines, functions, and interrupt routines, when executing, also consume data memory, especially if they declare local variables. Of the available 2048 bytes, about 1800 bytes can be used for global variables. Space must be reserved for subroutines which may reduce the memory available for global variables. However, this can often save memory, as the local variables will be destroyed after the subroutine completes, releasing the memory to be used again.

When the user creates buffers with `SET DISPLAY` or `OPENCOM`, data memory will be consumed to allocate buffers.

Initializing Memory

In Cubloc BASIC, data memory is not cleared when powered on. The user should initialize variables or use the `RamClear` statement to initialize all data memory to 0. If data memory is not cleared, the values in memory could potentially be anything. If the contents of memory are to be predictable, you must either initialize each variable, or use the `Ramclear` statement.

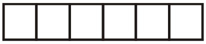
In the case of battery backed-up modules, the variables will remember their values after a power cycle (powering off and on).

Arrays

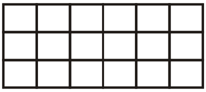
Cubloc BASIC supports arrays of up to 8 dimensions. Each dimension can contain up to 65,535 items.

```
Dim   A(20)    As   Byte          ' Declare an array of 20 Bytes
Dim   B(200)   As   Integer       ' Declare an array of 200 Integers
Dim   C(200)   As   Long          ' Declare an array of 200 Longs
Dim   D(20,10) As   Single        ' 2-dimensional Single array (20 x 10)
Dim   ST1(10)  As   String * 12   ' Declare String array
```

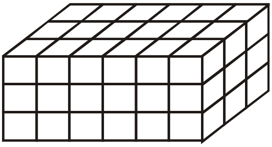
A(6)



A(3,6)



A(3,3,6)



Be sure to make note of how much memory is used when using multi-dimensional arrays.

```
' 13 * 10 = 130 Bytes of Data Memory
Dim   ST1(10) As String * 12

' 4 * 10 * 20 = 800 Bytes of Data Memory
Dim    D(20,10)  As   Single
```

Bit and Byte Modifiers

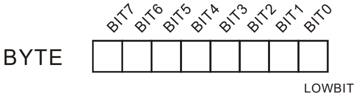
A variable's bits and bytes can be accessed individually by using the commands shown below:

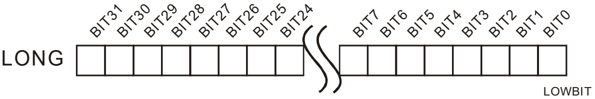
```
Dim  A  As Integer
A.LowByte = &H12    ' Store &H12 at A's lowest byte
```

Bit

LOWBIT	Variable's bit 0
BIT0 to 31	Variable's bit 0 through 31

```
A.BIT2 = 1  ' Make bit 2 of A equal to 1.
```



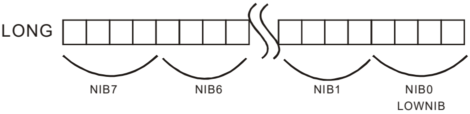


Nibble

A nibble is 4 bits. The user can access individual nibbles to make processing certain data formats easier.

LOWNIB	Variable's NIBBLE 0
NIB0 to 7	Variable's NIBBLE 0 to 7

```
A.NIB3 = 7 ' Store 7 in Nibble 3 of A
```

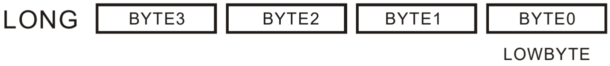


Byte

To access specific bytes of a variable, the following can be used.

LOWBYTE, BYTE0	BYTE 0 of Variable
BYTE1	BYTE 1 of Variable
BYTE2	BYTE 2 of Variable
BYTE3	BYTE 3 of Variable

```
A.Byte1 = &HAB  'Store &HAB in byte 1 of variable A
```

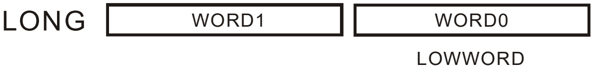


Word

To specify a certain Word of a variable, the following can be used:
(A Word is 16 bits)

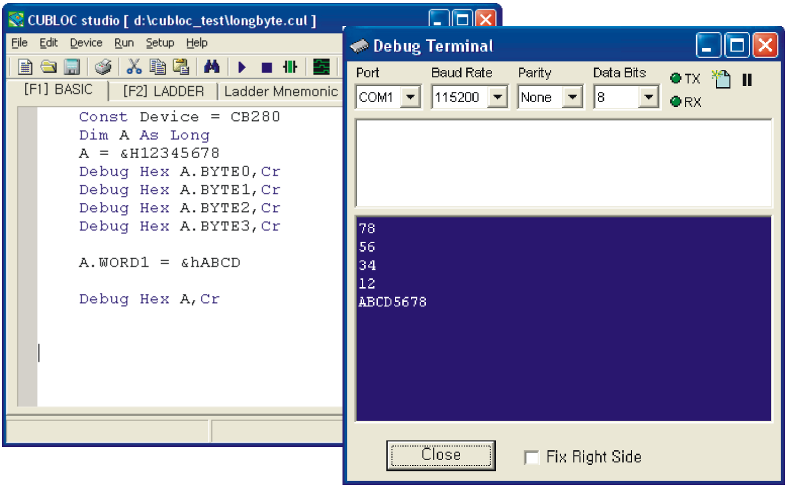
LOWWORD, WORD0	Word 0 of variable
WORD1	Word 1 of variable

```
A.Word1 = &HABCD  'Store &HABCD in word 1 of variable A
```



* Tip: Need to access 5 bits of a variable?
`NewVariable = Variable AND 0x1F.`
This will mask the last 5 bits of the variable.

DEMO PROGRAM



CUBLOC studio [d:\cubloc_test\longbyte.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
Debug Terminal
TX
RX
Close

```
Const Device = CB280
Dim A As Long
A = &H12345678
Debug Hex A.BYTE0,Cr
Debug Hex A.BYTE1,Cr
Debug Hex A.BYTE2,Cr
Debug Hex A.BYTE3,Cr

A.WORD1 = &hABCD

Debug Hex A,Cr
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
78
56
34
12
ABCD5678
```

□ Fix Right Side

Constants

Constants can be used to declare a constant value (a value that cannot be changed) within the program. This essentially allows a number to be assigned a name, often improving readability and debugging of the source code.

The CONST statement can be used to declare constants in Cubloc BASIC:

```
Const  PI  As  Single = 3.14159
Const  WRTTIME  As  Byte   = 10
Const  MSG1 As String = "ACCESS PORT"
```

When the constant is not given a type, the compiler will find an appropriate type for it as shown below:

```
Const  PI = 3.14159                ' Declare as SINGLE
Const  WRTTIME  = 10         ' Declare as Byte
Const  MYROOM   = 310        ' Declare as Integer since it’s over 255.
Const  MSG1 = "ACCESS PORT"        ' Declare as String
```

Con (Another CONST method)

The Con statement can be also used to declare constants as shown below:

```
PI            Con    3.14159          ' Declare as Single.
WRTTIME       Con    10               ' Declare as Byte
MYROOM        Con    310              ' Declare as Integer
MSG1          Con   "ACCESS PORT"      ' Declare as String
```

Constant Arrays...

In constant arrays, the user is able to store a list of values before the program begins. A program requiring a large number of constant values can be simplified as shown below:

```
Const Byte DATA1 = (31, 25, 102, 34, 1, 0, 0, 0, 0, 0, 65, 64, 34)
I = 0
A = DATA1(I)     ' Store 31 in A.
I = I + 1
A = DATA1(I)     ' Store 25 in A.
Const Byte DATA1 = ("CUBLOC SYSTEMS")
```

`String` data can be stored in `Byte` constant arrays. Each `Byte` contains the ASCII code of each character in the `String`. In the example above, If `DATA1(0)` is read, the ASCII code of 'C' is returned. Likewise if `DATA1(1)` is read, ASCII code of 'U' is returned.

`Integer` and floating point (`Single`) numbers can also be stored in arrays as shown below:

```
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)
```

For multiple-line constant arrays, end each line with a comma, or an underscore character as shown :

1)

```
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)
```

2)

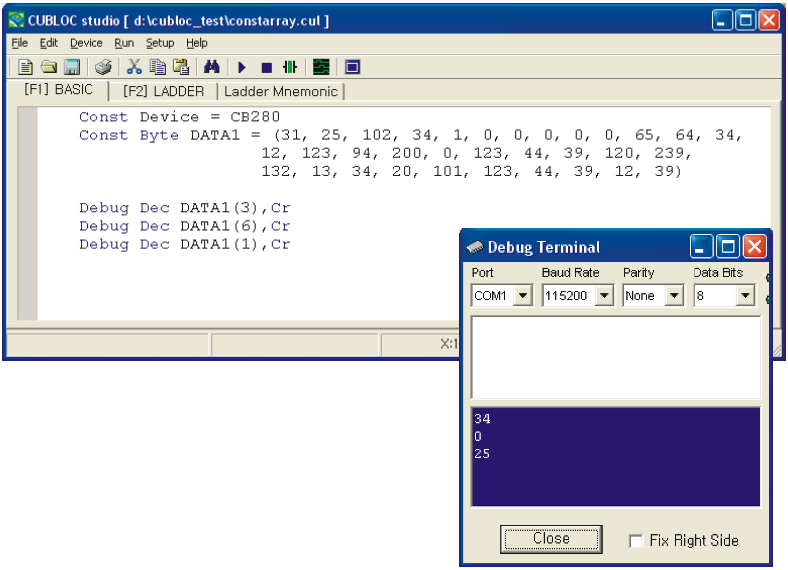
```
Const Byte DATA2 = (31, 25, 102, 34, 1, 0, 65, 64, 34_
                             , 101, 123, 44, 39, 12, 39)
```

String constant arrays are deprecated.

Please make note of the following differences between arrays and constant arrays.

	Array	Constant Array
Storage Location	Data Memory (SRAM)	Program Memory (FLASH)
When Allocated	During Program run	During Download
Can be Changed	Yes	No
Purpose	Store changing values	Store constant values
When Powered Off	Lost	Retained

DEMO PROGRAM


CUBLOC studio [d:\cubloc_test\constarray.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
X:1
Debug Terminal
Close
□ Fix Right Side

```
Const Device = CB280
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)

Debug Dec DATA1(3),Cr
Debug Dec DATA1(6),Cr
Debug Dec DATA1(1),Cr
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
34
0
25
```

Operators

When evaluating expressions in a programming language, it is important to understand the order of operations, also know as operator precedence. The following table describes Cubloc BASIC's operator precedence.

Operator	Explanation	Type	Precendence
^	To the power of	Math	First
*,/,MOD	Multiply, Divide, MOD	Math	
+,-	Add, Subtract	Math	
<<, >>	Left Shift, Right Shift	Logic	
<, >, <=, >=	Less than, Larger than, Less or Equal to , Larger or Equal to.	Comparison	
=, <>	Same, Different	Comparison	
AND, XOR, OR, NOT	Boolean AND, XOR, OR, & NOT	Logic	Last

Within each row in the table above, the operators are evaluated from left to right. Operators can be used in conditional statement as show below.

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

$$\frac{1}{2} \Longrightarrow 1/2$$

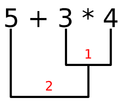
$$\frac{5}{3+4} \Longrightarrow 5 / (3+4)$$

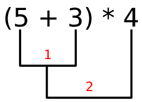
$$\frac{2+6}{3+4} \Longrightarrow (2+6) / (3+4)$$

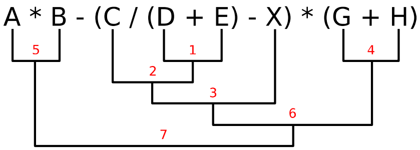
Use parenthesis to explicitly control the order of operations in an expression.

When multiple operators are used, the expression will be evaluated in the following order:

1) Operator(s) inside parenthesis
2) Negative Sign (–)
3) Exponent (^)
4) Multiplication, Division, Remainder (*, /, MOD)
5) Addition/Subtraction (+,-)
6) Bit-wise Left Shift, Bit-wise Right Shift (<<, >>)







Operators used in Cubloc BASIC may differ slightly from common mathematical operators. Please refer to the table below:

Operator	Math	Basic	Example
Add	+	+	3+4+5, 6+A
Subtract	-	-	10-3, 63-B
Multiply	X	*	2 * 4, A * 5
Division	÷	/	1234/3, 3843/A
To the power of	5^3	^	5^3, A^2
MOD	Remainder of	mod	102 mod 3

When numbers of different types are mixed in an expression, the final result is cast to the type of the assigned variable.

```
  Dim F1 As Single
  Dim A As Long
  F1 = 1.1234
  A = F1 * 3.14        ' A gets 3 even though result is 3.525456.
```

Please be sure to include a decimal point(.) when using floating point numbers even if your computer's language setting uses a different character for its decimal separator.

```
  F1 = 3.0/4.0         ' Write 3/4 as 3.0/4.0 for floating values
  F1 = 200.0 + Floor(A) * 12.0 + SQR(B)  '200 as 200.0, 12 as 12.0…
```

The `And`, `Xor`, and `Or` operators are used for both logical and bit-wise operations.

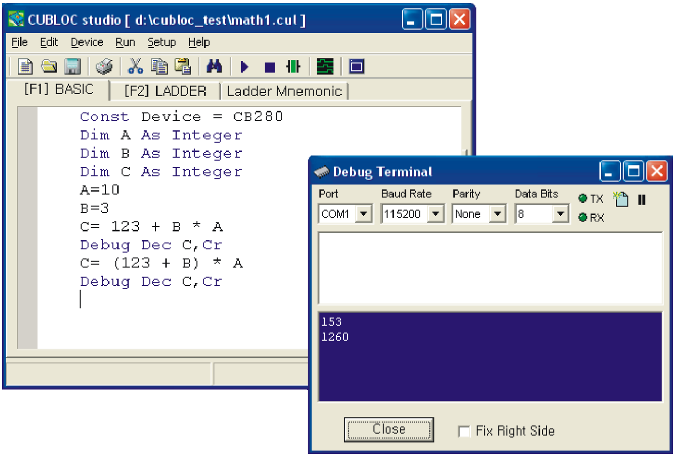
```
If A=1 And B=1 Then C=1 ' if A=1 and B=1 …(Logical Operation)
If A=1 Or B=1 Then C=1  ' if A=1 or B=1…(Logical Operation)

A = B And &HF 'Set the upper 4 bits to zero. (Bit-wise Operation)
A = B Xor &HF 'Invert the lower 4 bits. (Bit-wise Operation)
A = B Or &HF  'Set the lower 4 bits to 1. (Bit-wise Operation).
```

Strings can be compared with the = operator.

```
Dim ST1 AS String * 12
Dim ST2 AS String * 12
ST1 = "COMFILE"
ST2 = "CUBLOC"
If ST1=ST2 Then ST2 = "OK"    ' Check if ST1 is same as ST2.
```

When comparing Strings, Cubloc BASIC compares the ASCII value of each character in the String. Therefore, String comparisons are case-sensitive; "COMFILE" does not equal "comfile".

DEMO PROGRAM



Port: COM1, Baud Rate: 115200, Parity: None, Data Bits: 8

□ Fix Right Side

```
Const Device = CB280
Dim A As Integer
Dim B As Integer
Dim C As Integer
A=10
B=3
C= 123 + B * A
Debug Dec C,Cr
C= (123 + B) * A
Debug Dec C,Cr
```

```
153
1260
```

Expressing Numbers

There are three possible ways to represent numbers in Cubloc BASIC: binary, decimal and hexadecimal. The binary and hexadecimal representations are useful when interfacing to digital devices. The decimal representation is the standard human readable format.

Examples:

```
Binary :       &B10001010, &B10101,
               0b1001001, 0b1100

Decimal :      10, 20, 32, 1234

Hexadecimal :  &HA, &H1234, &HABCD
               0xABCD, 0x1234      ← Similar to C
               $1234, $ABCD        ← Similar to Assembly Language
```

The BASIC Preprocessor

The BASIC preprocessor is a macro processor that is used automatically by the compiler to transform your program before compilation. It is called a macro processor because it allows you to define macros, which are brief abbreviations for longer constructs.

Cubloc BASIC uses a preprocessor similar to the C language. Preprocessor directives like `#include` and `#define` can be used to include files and process code before compiling.

#include "filename"

Reuse code by including source files. For files in the same directory as the current source file, you can write the following:

```
#include "MYLIB.cub"
```

For files in other directories, you will need to include the full path name as shown here:

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

Using include files, you can store all of your common subroutines in a separate file. In this case, please make sure to `#include` the subroutine file at the very end of your program, after the `End` statement.

#define name constants

By using `#define`, you can assign names to values before compiling.

```
#define motorport 4
Low motorport
```

In the example above, `motorport` will be compiled as 4. You can also use `CONST` for similar tasks. However, `CONST` will use data memory; #define will only use program memory.

```
CONST motorport = 4
Low motorport
```

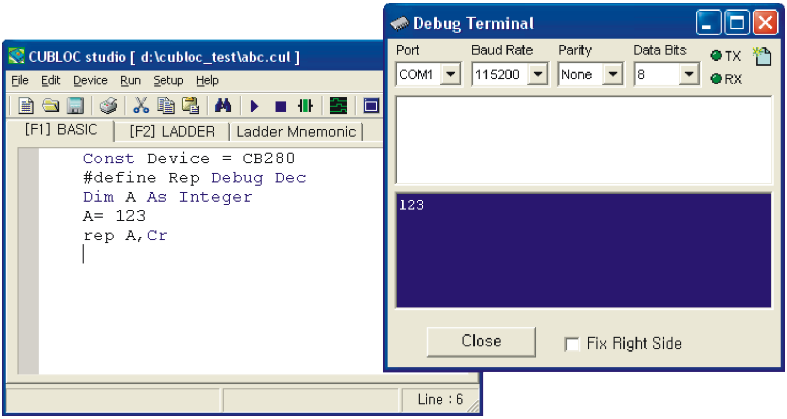
The following example uses #define for replacing a line of code:

```
#define FLAGREG1 2
#define f_led FLAGREG1.BIT0
#define calc (4+i)*256
f_led = 1                       ' Set FLAGREG1's bit zero to  1.
If f_led = 1 Then f_led = 0        ' Make it easier to read.
j = calc                        'Calculations can be simplified
```

NOTE

The "name" argument in a `#define` statement is not case-sensitive. For example, `#define ALPHA 0` and `#define alpha 0` do not define different constants. They both define the same constant.

DEMO PROGRAM



```
Const Device = CB280
#define Rep Debug Dec
Dim A As Integer
A= 123
rep A,Cr
```

```
123
```

Conditional Directives

A conditional directive is a directive that instructs the preprocessor to select whether or not to include a part of code before compilation. Preprocessor conditional directives can test arithmetic expressions, or whether a name is defined as a macro.

Here are some reasons to use a conditional.

- A program may need to use different code depending on the module it is to run on. In some cases the code for one module may be different on another module. With a conditional directive, a BASIC program may be programmed to compile on any of the Cubloc/Cutouch modules without making changes to the source code.
- If you want to be able to compile the same source file into two different programs. For example one version might be compiled with debugging statements, and one without.

#if constant
#endif

The preprocessor directive `#if` will compare a constant declared with `CONST` to another constant. If the `#if` statement is true, the statements inside the `#if…#endif` block will be compiled, otherwise the statements will be discarded.

```
Const Device = CB280

Delay 500

' Device only returns the decimal number
#If Device = 220
 Debug "CB220 module used!"
#endif
```

The above example illustrates how, depending on the type of Cubloc/Cutouch declared, you can decide to include a command in the final compilation of your program. Using conditional directives, you will be able to write applications for different Cubloc/Cutouch modules with just one source file.

Using the preprocessor directive #elseif or #else, you can create more complex #if…#endif expressions.

```
Const Device = CB220

Delay 500

' Device only returns the decimal number
#If Device = 220
 Debug "CB220 module used!"
#elseif device = 280
Debug "CB280 module used!"
#elseif device = 290
Debug "CB290 module used!"
#elseif device = 1720
Debug "CT1720 module used!"
#endif
```

#else may only be used ONCE in an #if expression. Also, you can only compare constants declared with the CONST statement in a #if directive.

#ifdef name
#endif

You can use #ifdef to check if a constant was previously defined with a #define directive or CONST statment. If the constant has been previously defined, the statements inside the #if…#endif block will be compiled, otherwise they will be discarded.

```
#define  LOWMODEL 0
#ifdef LOWMODEL
            Low 0
#endif
```

In the above example, since LOWMODEL is defined, the statement LOW 0 is compiled.

#else #elseifdef may be used for more complex expressions as shown below:

```
#ifdef LOWMODEL
            Low 0
#elseifdef HIGHMODEL
            High 0
#else
            Low 1
#endif
```

#ifndef name
#endif

#ifndef is the opposite of the #ifdef directive. If a constant has not been defined, the statements inside a #ifndef…#endif block will be compiled, otherwise the statements will be discarded.

```
#define  LOWMODEL 0
#ifndef LOWMODEL
            Low 0
#endif
```

#elseifndef and #else may be used for more complex expressions as shown below:

```
#ifndef LOWMODEL
            Low 0
#elseifndef HIGHMODEL
            High 0
#else
            Low 1
#endif
```

Finally, the directives may be mixed as shown below:

```
#if MODELNO = 0
            Low 0
#elseifdef HIGHMODEL
            High 0
#else
            Low 1
#endif
```

Nested #if directives are not supported; #if may not be used inside another #if.

To Use Only Ladder Logic

If you do not need to use BASIC, you can program in Ladder Logic alone. But you will need a few lines of BASIC to get started, as shown below:

```
Const Device = CB280   ' Select device

UsePin 0,In,START      ' Declare pins to use
UsePin 1,Out,RELAY

Alias M0 = MOTORSTATE  ' Set Aliases
Alias M1 = RELAY1STATE

Set Ladder On          ' Start Ladder.
```

Device model, aliases, and pin I/O mode must be set in BASIC. Ladder Logic must be started in BASIC with the `Set Ladder On` statement.

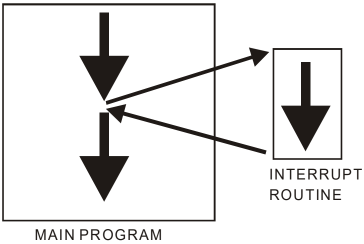
To Use Only BASIC

Ladder Logic is off as default.

```
Set Ladder On     ' If using only BASIC, don't use this statement
Ladderscan        ' Or this one
```

Interrupts

Interrupts are external events that stop a program's normal flow (interrupting the program) and immediately execute some other subroutine, called an interrupt service routine (ISR). The `On…GoSub` statement can be used to tell a program to listen for an interrupt and execute an ISR. When the interrupt occurs, the main program stops execution and jumps to the ISR designated by the `ON…GOSUB` statement. Once the ISR is finished, the `Return` statement is used to return execution back to the main program.



External key input can occur at any time and data can be received via RS-232 at any time. Since the main program cannot wait forever to receive these inputs, we need interrupts. If a key is pressed or serial data is received while the main program is running, an interrupt occurs and the main program jumps to the appropriate ISR.

While an ISR is running, subsequent **interrupts of the same type** are ignored. For example, if an RS-232 receive interrupt occurs during execution of previous RS-232 receive ISR, the new RS-232 receive interrupt will be ignored. On the other hand, if a timer interrupt occurs (See the `On Timer` statement) during execution of an RS-232 receive ISR, it will immediately interrupt the RS-232 receive ISR, execute the timer's ISR, and then return to the RS-232 receive ISR.

Interrupt Type	**Explanation**
`On Timer`	Initiates an interrupt on a periodic basis
`On Int`	Initiates an interrupt when an external input is received.
`On Recv`	Initiates an interrupt when data is received via RS-232
`On LadderInt`	Initiates an interrupt when Ladder Logic requests an interrupt
`On Pad`	Initiates an interrupt when Pad receives data

More about Interrupts…

The Cubloc and Cutouch have a Real-Time Operating System (RTOS) which controls interrupt events. This is slightly different from the microcontroller's hardware interrupts.

1. When interrupt A occurs, while interrupt A's ISR is executing, another interrupt A cannot occur. But interrupt B can occur. Here A and B are different types of interrupts. (e.g. `On Timer` and `On Recv`)

2. When interrupt B occurs while executing interrupt A's ISR, interrupt B's ISR will be executed immediately and execution will return to interrupt A's ISR to finish.

3. At the end of an ISR, be sure to include a `Return` statement. Otherwise, the program may malfunction.

4. If no interrupt is required for your program, you can increase the execution speed of the Cubloc by setting turning off all interrupts using the **`Set OnGlobal Off`**. By default, **`Set OnGlobal`** is set to `On`.

5. In the case of `On Recv`, data received during an `On Recv` ISR will simply be stored in the serial port's receive buffer. Therefore the data will not be lost. After the current `On Recv` ISR is finished, if there's new data in the receive buffer, another `On Recv` interrupt will be immediately generated. The **`BClr`** statement can be used if you do not want to process another `On Recv` interrupt.

6. If you declare an interrupt more than once, only the last one declared will be in effect.

IMPORTANT!!!!!

Please pay attention when creating the interrupt service routine (ISR). It must require less time to execute than the interval itself. If interval is set to 10ms, the ISR must execute within 10 ms. Otherwise, collisions can occur.

It is quite common for a microcontroller to make use of interrupts, but with the Cubloc, interrupts should be used sparingly.

The Cubloc does not handle hardware interrupts directly. When an interrupt occurs, the handler jumps to the BASIC interrupter. Because the BASIC interpreter is slower than directly processed machine code, extra latency can be added to the interrupt service routine and prevent the main routine from operating smoothly.

Therefore, we recommend using interrupts sparingly with the Cubloc. If interrupts are used, keep the interrupt service routine very light and fast. For example, writing to a serial port or using the Delay command is not recommended.

If you need a device that can handle interrupts with speed and ease, please consider the Moacon. The Moacon is an ARM based modular industrial controller from Comfile Technology designed for high speed, interrupt intensive applications.

Pointers using Peek, Poke, and MemAdr

The following is an example that uses the EEWrite statement and the EERead statement to read and write floating point data:

```
Const Device = CB280
Dim f1 As Single, f2 As Single
f1 = 3.14
EEWrite 0,f1,4
f2 = EERead(0,4)
Debug Float f2,cr
```

When you run this program, the debug window will show 3.00000 instead of 3.14. The reason is that the EEWrite statement automatically converts floating point values to whole numbers.

In order to store floating point values, we can use Peek and Poke to read the data directly:

```
Const Device = CB280
Dim F1 As Single, F2 As Single
F1 = 3.14
EEWrite 10,Peek(MemAdr(F1),4),4
Poke MemAdr(F2),EERead(10,4),4

Debug Float F2,CR
```

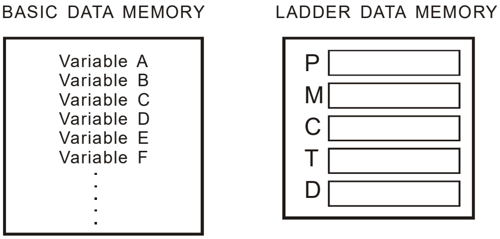
The debug window will now show 3.14.

We use MemAdr(F1) to find the memory address of F1 and then use the Peek statement to directly access the memory and write 4 bytes. We store that value in the EEPROM. Next, we use MemAdr(F2) and Poke to read 4 bytes directly.

Warning : Please use caution when using these command as incorrectly manipulating memory can affect the entire program. Peek and Poke may only access data memory.

Sharing Data

The Cubloc has separate BASIC and Ladder Logic data memory areas.



Ladder Logic data memory can be accessed from BASIC easily by using system variables. Using these system variables, data can be easily read from or written to Ladder Logic data memory.

System Variable (Array)	Access Units	Ladder Logic Register
_P	Bits _P(0) to _P(127)	P Register
_M	Bits _M(0) to _M(511)	M Register
_WP	Words _WP(0) to _WP(7)	P Register (Word Access)
_WM	Words _WM(0) to _WM(31)	M Register (Word Access)
_T	Words _T(0) to _T(99)	T Register (Timer)
_C	Words _C(0) to _C(49)	C Register (Counter)
_D	Words _D(0) to _D(99)	D Register (Data)

Registers P and M can be accessed in units of bits, and registers C, T, and D can be accessed in units of words. To access P and M registers in units of words, use `_WP` and `_WD`. For example, `_WP(0)` represents P0 through P15.

The following is an example program :

```
_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
```

Accessing BASIC variables from Ladder Logic is not possible, but you can use Ladder interrupts to request a BASIC routine to change a Ladder Logic variable.

Use Ladder Logic pins in BASIC using the Alias Command

The `Alias` command can be used to set aliases for Ladder Logic registers (**except the D register**). Aliases, although declared in BASIC, can only be used in Ladder Logic.

```
UsePin 0,In,START
UsePin 1,Out,RELAY

Alias M0 = MOTORSTATE
Alias M1 = RELAY1STATE
Alias T1 = SUBTIMER
```

Chapter 5: Cubloc BASIC Functions

Math Functions

Sin, Cos, Tan

Return sine, cos ine, and tangent values. Cubloc uses radians as units. Use a `Single` for most precise results.

```
A=Sin B                    ' Return the sine of B.
A=Cos B                    ' Return the cosine of B.
A=Tan B                    ' Return the tangent of B.
```

ASin, ACos, ATan

Return arc sine, arc cosine, and arc tangent values. Cubloc uses radians as units. Use a `Single` for the most precise results.

```
A=ASin B                   ' Return the arc sine of B.
A=ACos B                   ' Return the arc cosine of B.
A=ATan B                   ' Return the arc tangent of B.
```

Sinh, Cosh, Tanh

Return hyperbolic sine, hyperbolic cosine, and hyperbolic tangent values.

```
A=Sinh B                   ' Return the hyperbolic sine of B.
A=Cosh B                   ' Return the hyperbolic cosine of B.
A=Tanh B                   ' Return the hyperbolic tangent of B.
```

Sqr   Return Square Root of a value.

```
A=Sqr B                    ' Return the square root of B
```

Exp   Return e^x.

```
A=Exp X                    'Returns e$^x$.
```

Log, Log10   Return Log or Log10 of a value.

```
A=Log B   or   A=Log10 B
```

For the natural logarithm (Ln), simply do: `A=Log B`

Abs Return the absolute value of a `Long`.

```
Dim A As Long, B As Long
B = -1234
A=Abs B             ' Return |B|.
Debug Dec A         ' Print 1234
```

FAbs Return the absolute value of a `Single`.

```
Dim A As Single, B As Single
B=-1234.0
A=Fabs B            ' Return |B|.
Debug Float A       ' Print 1234.00
```

Floor Round down to the nearest whole number.

```
Dim A As Single, B As Single
B = 3.5
A=Floor B            ' Floor 3.5 gives 3.
Debug Float A        ' Print 3.0
```

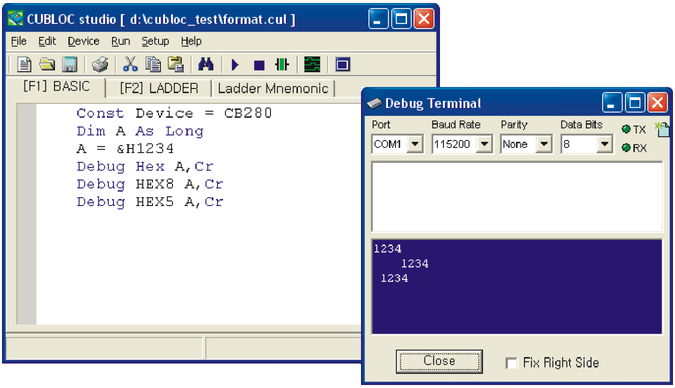
Type Conversion

Type conversion can be used to convert the variable to the desired representation.

Hex

Converts the variable to a string representation of a hexadecimal value (16 bit). `Hex8` means to convert to 8 decimal places. (1 to 8 can be used for decimal places)

```
Debug Hex A    ' If A is 123ABC, 123ABC is printed
Debug Hex8 A   ' If A is 123ABC, bb123ABC is printed,
               ' b is a blank space in this case.
Debug Hex5 A   ' If A is 123ABC, 23ABC is printed, first character
               ' is cut.
```



```
Const Device = CB280
Dim A As Long
A = &H1234
Debug Hex A,Cr
Debug HEX8 A,Cr
Debug HEX5 A,Cr
```

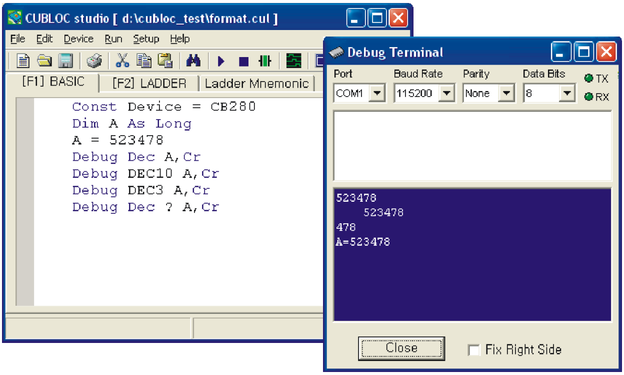
Port: COM1, Baud Rate: 115200, Parity: None, Data Bits: 8

```
1234
    1234
 1234
```

Dec

Converts an integer variable to a string representation of a decimal (10 bit). Dec8 means to convert to 8 decimal places. (1 to 11 can be used for decimal places)

```
Debug Dec A              ' If A is 1234, 1234 is printed.
Debug Dec10 A            ' If A is 1234, bbbbbb1234 is printed,
                         ' b is a blank space in this case.
Debug Dec3 A             ' If A is 1234, 234 is printed, first
                         ' character is cut
```



CUBLOC studio [d:\cubloc_test\format.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
Debug Terminal
TX
RX
Close
Fix Right Side

```
Const Device = CB280
Dim A As Long
A = 523478
Debug Dec A,Cr
Debug DEC10 A,Cr
Debug DEC3 A,Cr
Debug Dec ? A,Cr
```

Port: COM1, Baud Rate: 115200, Parity: None, Data Bits: 8

```
523478
    523478
478
A=523478
```

?

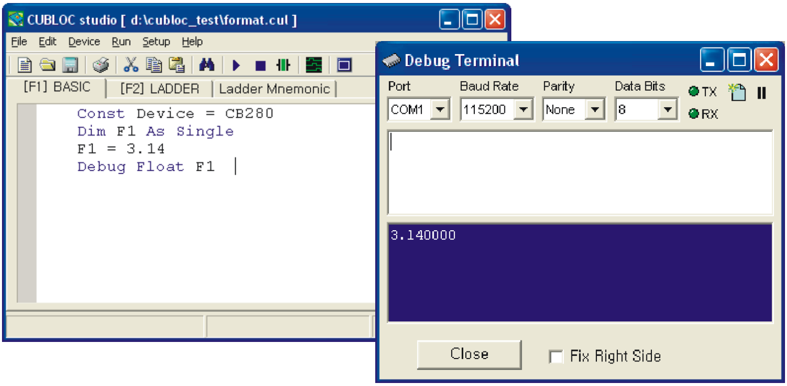
Include the name of the variable by using question mark (?). This question mark can only be used with HEX or DEC.

```
Debug Dec ? A             ' If A is 1234, "A=1234" will be printed.
Debug Hex ? A             ' If A is &HABCD, "A=ABCD" will be printed.
Debug Hex ? B             ' If B is a variable within a subroutine
                          ' (for example: subroutine CONV) with a value of
                          ' &HABCD "B_@_CONV=ABCD" will be printed
```

Float

Use `Float` to convert floating point values to `String`.

```
Const Device = cb280
Dim F1 As Single
F1 = 3.14
Debug Float F1,cr            ' Print "3.14000".

Dim ST As String * 15
ST = Float F1                ' First store in a String.
ST = Left(ST,3)              ' Convert to 3 decimal places
Debug ST                     ' Print "3.14".
```

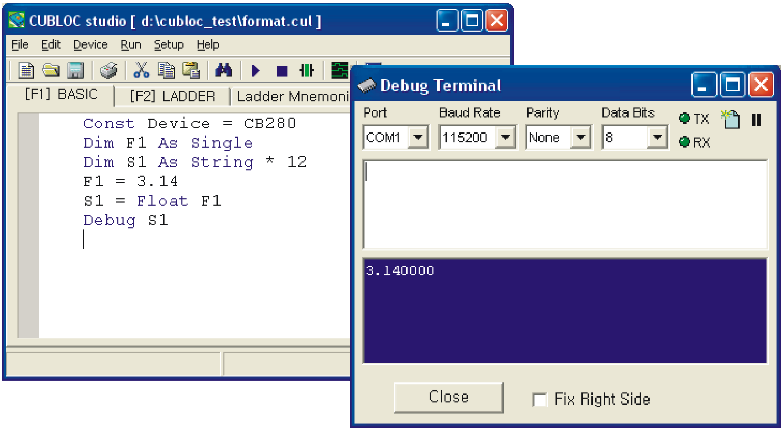


```
Const Device = CB280
Dim F1 As Single
F1 = 3.14
Debug Float F1
```

Port: COM1 Baud Rate: 115200 Parity: None Data Bits: 8

```
3.140000
```

You can also store a value in `String` before printing with `Debug` statements or displaying to the LCD as shown below.


CUBLOC studio [d:\cubloc_test\format.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemoni
Debug Terminal
TX
RX
Close
□ Fix Right Side

```
Const Device = CB280
Dim F1 As Single
Dim S1 As String * 12
F1 = 3.14
S1 = Float F1
Debug S1
```

Port: COM1 Baud Rate: 115200 Parity: None Data Bits: 8

```
3.140000
```

String Functions

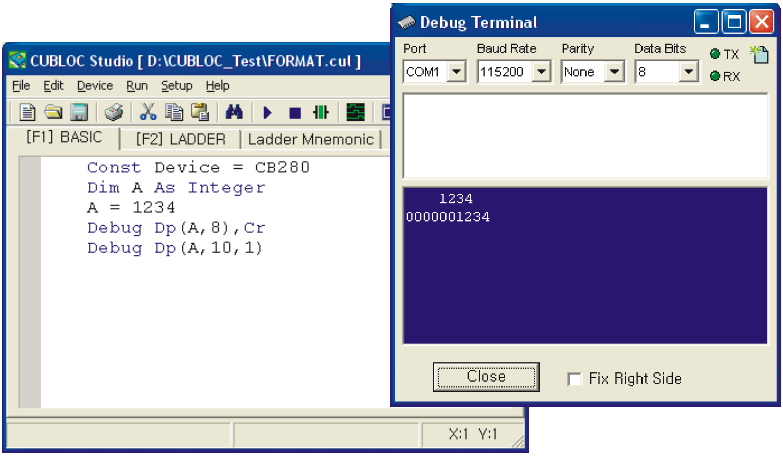
String functions are provided to assist the user in accessing and modifying data within a `String`.

DP(value, numberOfDigits, zeroPrint)

`DP` converts a variable into a decimal string representation.

If `zeroPrint` is set to 1, zeros are printed instead of blank spaces.

```
Dim A as Integer
Debug DP(A,10,0)     ' Convert A into decimal String representation.
                     ' Set display decimalPlaces to 10.
                     ' If A is 1234, "      1234" will be displayed.
                     ' (notice the 6 blank spaces)

Debug DP(A,10,1)     ' If A is 1234, 0000001234 will be displayed.
```


CUBLOC Studio [D:\CUBLOC_Test\FORMAT.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
Debug Terminal
TX
RX
Close
X:1 Y:1

```
Const Device = CB280
Dim A As Integer
A = 1234
Debug Dp(A,8),Cr
Debug Dp(A,10,1)
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

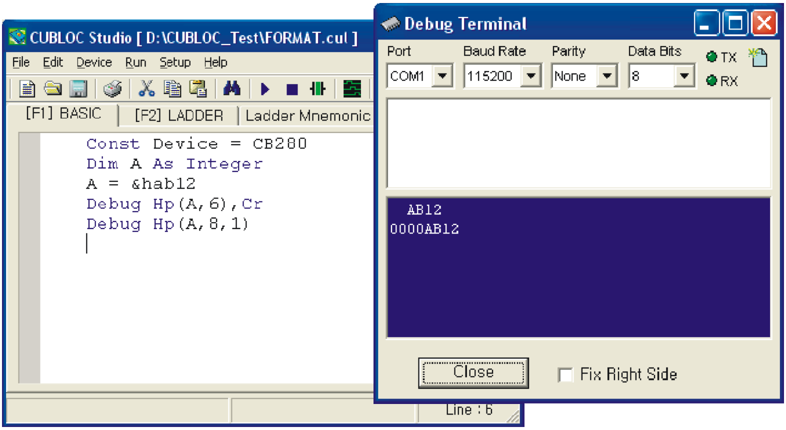
```
    1234
0000001234
```

□ Fix Right Side

HP(value, numberOfDigits, zeroPrint)

`HP` converts a variable into a hexadecimal string representation. If `ZeroPrint` is set to 1, zeroes are substituted for blank spaces.

```
Debug HP(A,4,0)       ' Convert A into HEX String representation
                      ' Set display decimal places to 4.
                      ' If A is ABC, bABC will be displayed.
                      ' (b stand for blank spaces.)

Debug HP(A,4,1)       ' If A is ABC, 0ABC will be displayed.
```



```
Const Device = CB280
Dim A As Integer
A = &hab12
Debug Hp(A,6),Cr
Debug Hp(A,8,1)
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

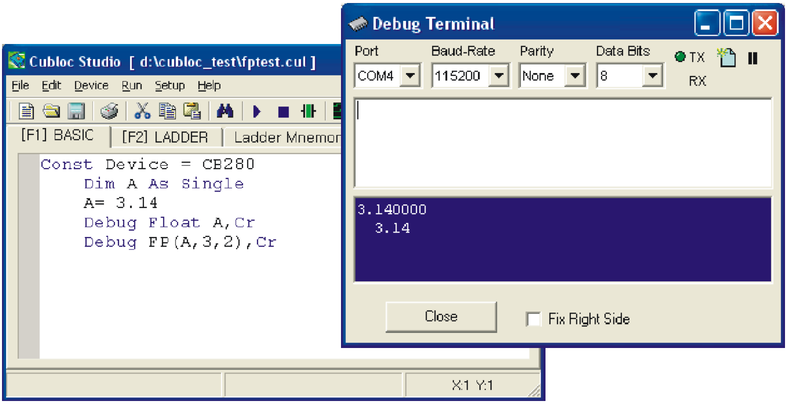
```
  AB12
0000AB12
```

FP (value, wholeNumberDigits, fractionalNumberDigits)

FP converts floating point variables into a formatted string with user defined whole and fractional number of digits.

```
Dim A as Single
A = 3.14
Debug Float A          ' 3.1400000 Prints all digits.
Debug FP(A,3,2)        '   3.14 Print user defined digits.
```

With the FP function, the user can control the number of digits to be used for string data when using Debug or displaying to an LCD.



```
Const Device = CB280
      Dim A As Single
      A= 3.14
      Debug Float A,Cr
      Debug FP(A,3,2),Cr
```

Port: COM4; Baud-Rate: 115200; Parity: None; Data Bits: 8

```
3.140000
  3.14
```

Cubloc floating-point values are stored in accordance to the IEEE724 format. The output of FP and Float may differ but the value stored in the variable will be the same.

Left(stringValue, numberOfCharacters)

Returns a specified number of characters from the left side of a `String`

```
Dim ST1 As String * 12
ST1 = "CUBLOC"
Debug Left(ST1,4)   ' "CUBL" is printed.
```

Right(stringValue, numberOfCharacters)

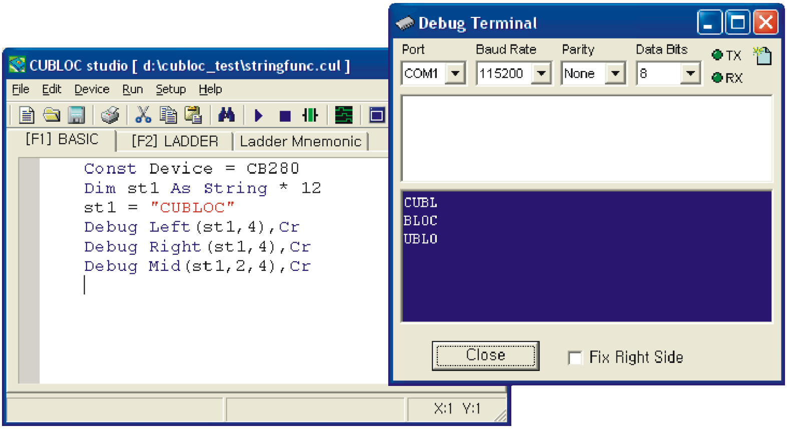
Returns a specified number of characters from the right side of a `String`.

```
Dim ST1 As String * 12
ST1 = "CUBLOC"
Debug Right(ST1,4)   ' "BLOC" is printed.
```

Mid(stringValue, location, numberOfCharacters)

Returns a specified number of characters from a string starting at a specified location.

```
Dim ST1 As String * 12
ST1 = "CUBLOC"
Debug Mid(ST1,2,4)   ' "UBLO" is printed.
```



```
Const Device = CB280
Dim st1 As String * 12
st1 = "CUBLOC"
Debug Left(st1,4),Cr
Debug Right(st1,4),Cr
Debug Mid(st1,2,4),Cr
```

Port: COM1 · Baud Rate: 115200 · Parity: None · Data Bits: 8 · □ Fix Right Side

```
CUBL
BLOC
UBLO
```

Len(stringValue)

Return the length of the given `String`.

```
Dim ST1 As String * 12
ST1 = "CUBLOC"
Debug Dec Len(ST1)  ' 6 is printed since there are 6 characters in ST1.
```

String(asciiCode, length)

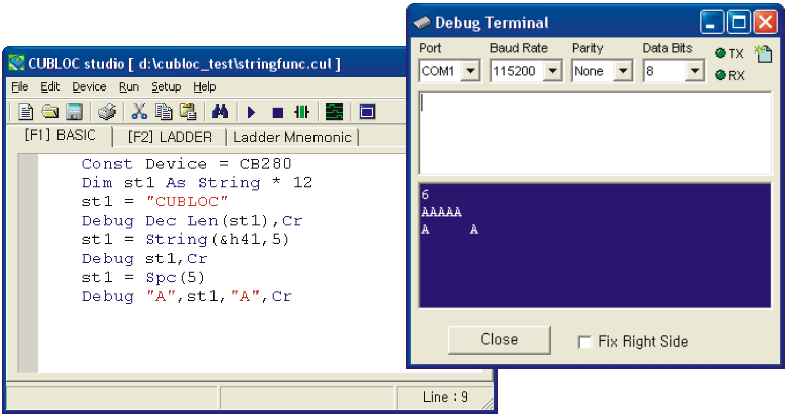
Creates a `String` of a specified length with the given ASCII code repeated for the length of the `String`.

```
Dim ST1 As String * 12
ST1 = String(&H41,5)
Debug ST1          ' AAAAA is printed.
                   ' &H41 is ASCII code for character A.
```

SPC(numberOfSpaces)

Create a specified amount of blank space

```
Dim ST1 As String * 12
ST1 = SPC(5)
Debug "A",ST1,"A"   ' A     A is printed.
                    ' Note the 5 blank spaced between each A.
```



```
Const Device = CB280
Dim st1 As String * 12
st1 = "CUBLOC"
Debug Dec Len(st1),Cr
st1 = String(&h41,5)
Debug st1,Cr
st1 = Spc(5)
Debug "A",st1,"A",Cr
```

Port: COM1 · Baud Rate: 115200 · Parity: None · Data Bits: 8

```
6
AAAAA
A     A
```

LTrim(stringValue)

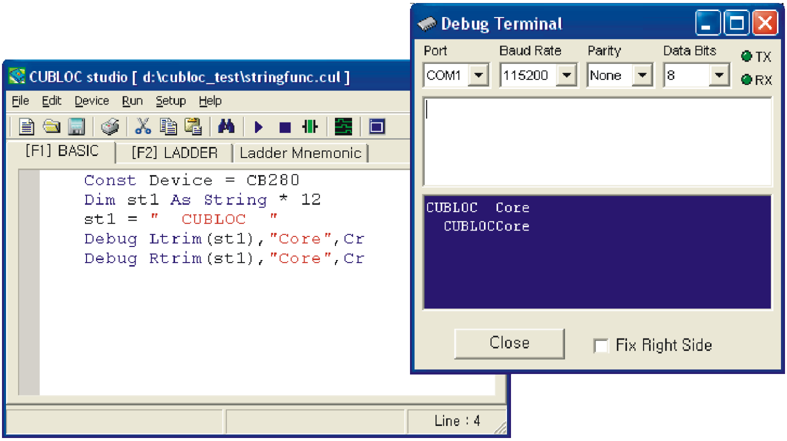
Removes all blank spaces from the left side of the given `String`.

```
Dim ST1 As String * 12
ST1 = "  COMFILE"
ST1 = LTrim(ST1)
Debug "AAA",ST1     ' AAACOMFILE is printed.
```

RTrim(stringValue)

Removes all blank spaces from the right side of the given `String`.

```
Dim ST1 As String * 12
ST1 = "COMFILE  "
ST1 = RTrim(ST1)
Debug ST1,"TECH"     ' COMFILETECH is printed.
                     ' Blank spaces on the right are removed.
```



```
Const Device = CB280
Dim st1 As String * 12
st1 = "  CUBLOC  "
Debug Ltrim(st1),"Core",Cr
Debug Rtrim(st1),"Core",Cr
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
CUBLOC  Core
  CUBLOCCore
```

Val(stringValue)

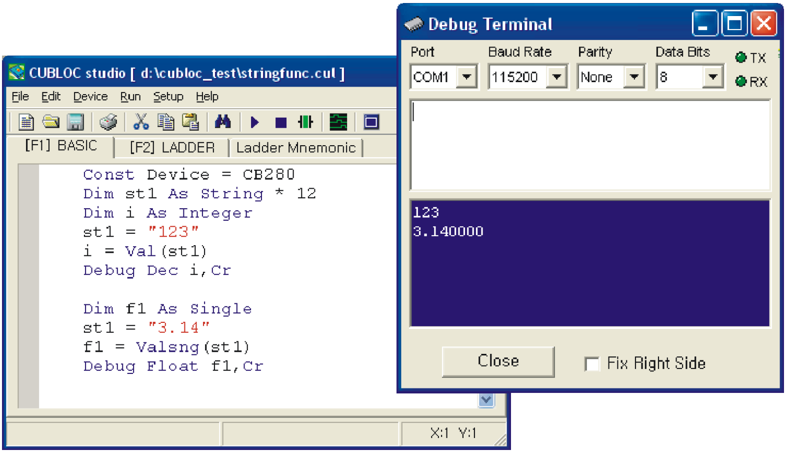
Parses a given `String` into its equivalent decimal value.

```
Dim ST1 As String * 12
Dim i As Integer
ST1 = "123"
i = Val(ST1)          ' 123 is stored in variable I as a number.
```

ValSng(stringValue)

Parses a given `String` into its equivalent floating point value.

```
Dim ST1 As String * 12
Dim F As Single
ST1 = "3.14"
F = ValSng(ST1)       ' 3.14 is stored in variable F as a floating
                      ' point number.
```



Port: COM1 Baud Rate: 115200 Parity: None Data Bits: 8

```
Const Device = CB280
Dim st1 As String * 12
Dim i As Integer
st1 = "123"
i = Val(st1)
Debug Dec i,Cr

Dim f1 As Single
st1 = "3.14"
f1 = Valsng(st1)
Debug Float f1,Cr
```

```
123
3.140000
```

ValHex(String variable)

Parses a given `String` in hexadecimal format into its numeric equivalent.

```
Dim ST1 AS String * 12
Dim i AS Long
ST1 = "ABCD123"
i = ValHex(ST1)            '&HABCD123 is stored in variable I
```

Chr(asciiCode)

Return the character represented by the given ASCII code.

```
Dim ST1 AS String * 12
ST1 = Chr(&H41)
Debug ST1               ' Print A. &H41 is the ASCII code for character A.
```

ASC(stringValue)

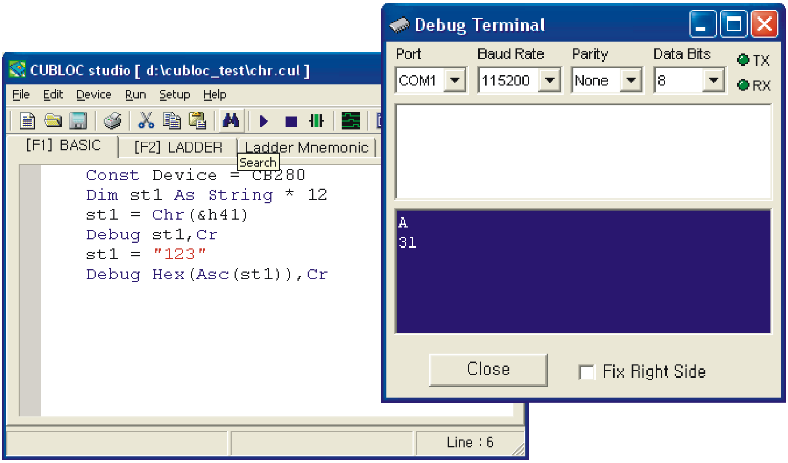
Return ASCII code of the first character in the given `String`.

```
Dim ST1 AS String * 12
Dim i AS Integer
ST1 = "123"
i = Asc(ST1)     ' &H31 is stored in variable I.  ASCII code of 1
                ' is &H31 or 0x31.
```

Caution:

A variable must be used when using string functions.

```
Debug Left("INTEGER",4)  ' A String constant cannot be used.
ST1 = "INTEGER"
Debug Left(ST1,4)        ' The String must be stored as a variable first.
```



```
Const Device = CB280
Dim st1 As String * 12
st1 = Chr(&h41)
Debug st1,Cr
st1 = "123"
Debug Hex(Asc(st1)),Cr
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
A
31
```

Chapter 6: Cubloc BASIC Statements & Library

AdIn( )

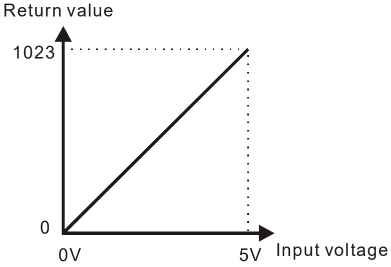
variable = AdIn (channel)

variable : Variable to store results (No String or Single)
channel : AD Channel Number (not I/O Pin Number)

Cubloc has several 10bit Analog to Digital Converters (ADCs) and 16bit Pulse Width Modulators (PWMs). The user can use an ADC to convert analog signals to digital, or use a PWM to convert digital signals to analog.

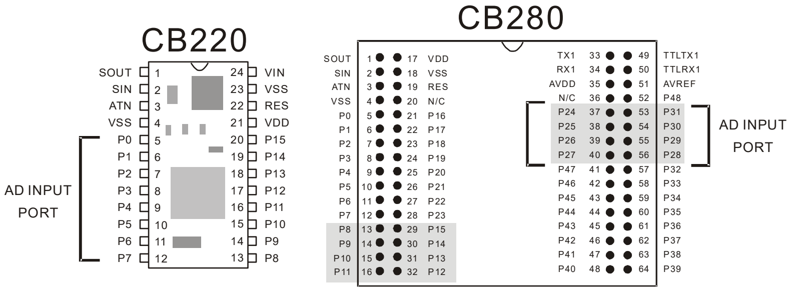
The `ADin` command reads the analog signal's amplitude and stores the result in a variable. Depending on the model, the number of ADC ports may vary. For the CB280, there are 8 AD ports (P24 to P31). An ADC port **must be set to input** before use.

When a voltage between 0 and AVREF is applied, that voltage is converted to a value from 0 to 1023. AVREF can accept voltages from 2V to 5V. The default reference is 5V. If an AVREF of 3V is used, voltages from 0 and 3V are converted to a value from 0 to 1023.
(*Note: CB220 AVREF is fixed to 5V)



```
Dim A As Integer
Input 24      ' Set port 24 to input.
A=AdIn(0)     ' Do a A/D conversion on channel 0 and
              ' store result in A
```

The CB220 and CB280 ADC ports are shown below:



Please refer to the following table for ADC channels.

	CB220	CB280	CB290	CT17xx	CB405
A/D channel 0	I/O 0	I/O 24	I/O 8	I/O 0	I/O 16
A/D channel 1	I/O 1	I/O 25	I/O 9	I/O 1	I/O 17
A/D channel 2	I/O 2	I/O 26	I/O 10	I/O 2	I/O 18
A/D channel 3	I/O 3	I/O 27	I/O 11	I/O 3	I/O 19
A/D channel 4	I/O 4	I/O 28	I/O 12	I/O 4	I/O 20
A/D channel 5	I/O 5	I/O 29	I/O 13	I/O 5	I/O 21
A/D channel 6	I/O 6	I/O 30	I/O 14	I/O 6	I/O 22
A/D channel 7	I/O 7	I/O 31	I/O 15	I/O 7	I/O 23
A/D channel 8					I/O 32
A/D channel 9					I/O 33
A/D channel 10					I/O 34
A/D channel 11					I/O 35
A/D channel 12					I/O 36
A/D channel 13					I/O 37
A/D channel 14					I/O 38
A/D channel 15					I/O 39

The `ADIn` statement does a single conversion per call. `TADIn` is a macro that returns the average of 10 conversions, giving the user more precise results. If you need more precision rather than speed, we recommend the using `TADIn` instead of `ADIn`. It is also possible to create your own averaging or filtering code for better precision.

Alias

Alias registerName = aliasName

registerName : Register name such as P0, M0, T0 (Do not use D registers)

aliasName : An Alias for the register chosen (up to 32 characters)

Aliases may be given to Ladder registers. Aliases can be used to make a program easier to read, understand, and debug. Please note that Aliases, although declared in BASIC, can only be used in Ladder Logic.

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

Bcd2Bin

variable = Bcd2Bin(bcdValue)

variable : Variable to store results (Returns LONG)
bcdValue : BCD value to convert to binary

Bcd2Bin converts a BCD (Binary Coded Decimal) number into a normal binary number, Cubloc BASIC's default number format. BCD is a way of expressing values as decimals.

For example, the number 3451 in binary is as shown below:

$$3451$$

$$\underbrace{0000}_{0}\quad\underbrace{1101}_{D}\quad\underbrace{0111}_{7}\quad\underbrace{1011}_{B}$$

The following is 3451 converted to BCD code. As you can see, each 4 bits represent one of the digits.

$$3451$$

$$\underbrace{0011}_{3}\quad\underbrace{0100}_{4}\quad\underbrace{0101}_{5}\quad\underbrace{0001}_{1}$$

This command is useful when the user needs to convert a variable for a device such as a 7 segment display, or a real-time clock.

```
Dim A As Integer
A=Bcd2Bin(&h1234)
Debug Dec A          ' Print 1234
```

See also Bin2Bcd.

BClr

BClr channel, buffertype

channel : RS232 Channel (0 to 3)
buffertype : 0=Receive, 1=Send, 2=Both

`Bclr` clears the specified RS-232 channel’s input buffer, output buffer, or both buffers. Use this statement if your code is about to receive data and there may be unneeded data already in the buffer.

```
BClr 1,0           ' Clear RS232 Channel 1’s rx buffer
BClr 1,1           ' Clear RS232 Channel 1’s tx buffer
BClr 1,2           ' Clear RS232 Channel 1’s rx & tx buffers
```

Beep

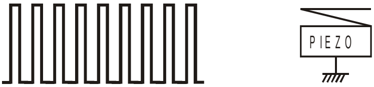
Beep port, length

port : Port number (0 to 255)

length : Pulse output period (1 to 65535)

Beep is used to create a beep sound. A piezo or a speaker can be connected to the specified port. A short beep will be generated. This is useful for creating button-press sound effects or alarm sounds. When this command is used, the specified port is automatically set to output.

```
Beep 2, 100    ' Output Beep on P2 for a period of 100
```



Bfree( )

variable = Bfree(channel, bufferType)

variable : Variable to store results (No String or Single)
channel : RS232 Channel number (0 to 3)
bufferType: 0=Receive Buffer, 1=Send Buffer

This function will return the number of free bytes in a receive or send buffer. When sending data, this command can be used to avoid overflowing the buffer. When receiving data, this command can help the program wait for a specified amount of data to be received before taking action.

```
Dim A As Byte
OpenCom 1,19200,0, 100, 50
If BFree(1,1)>10 Then
     Put "TECHNOLOGY"
End If
```

If the size of the buffer is set to 50, up to 49 bytes can be returned. When the buffer is empty, `BFree` will return 1 less than the buffer size.

Bin2Bcd

variable = Bin2Bcd(binValue)
variable : Variable to store results (Returns Long)
binValue : Binary value to be converted

`Bin2Bcd` converts a binary value to BCD (Binary Coded Decimal).

```
i = 123456
j = bin2Bcd(i)
Debug Hex j      ' Print 123456
```

See also `Bcd2Bin`.

BLen( )

variable = BLen(channel, bufferType)

variable : Variable to store results (No String or Single)
channel : RS-232 channel number (0 to 3)
bufferType: 0=receive buffer, 1=send buffer

BLen returns the current number of bytes in the specified RS-232 channel's buffer. If the buffer is empty, 0 will be returned. When receiving data, this function can be used to check how much data has been received before using Get or GetStr to read the data received.

If the receive buffer is full, it will not be able to receive any more data. To avoid this, use receive interrupts (See On Recv) or increase the buffer size.

```
Dim A As Byte
OpenCom 1,19200,0,100,50
On Recv DATARECV_RTN            ' When data is received through
                                ' RS232, jump to DATARECV_RTN

Do
Loop                            ' infinite loop

DATARECV_RTN:
      If BLen(1,0) > 0 Then     ' If there is at least 1 byte...
            A = Get(1)          ' Read 1 Byte
      End If
Return                          ' End Interrupt routine
```

ByteIn( )

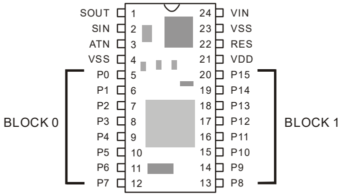
variable = ByteIn(portBlock)

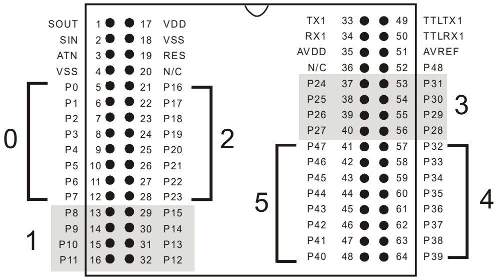
variable : Variable to store results (No String or Single)
portBlock : I/O Port Block Number (0 to 15)

`ByteIn` reads from an I/O port block (a group of 8 I/O ports). Ports 0 to 7 are block 0 and ports 8 to 15 are block 1. The port block numbers vary for the different models of Cubloc. When using this command, all I/O ports within the port block have their I/O mode set to input and their input is stored in a variable.

```
Dim A As Byte
A = ByteIn(0)  ' Read from Port Block 0 and store in variable A.
```

The CB220 and CB280 port block groupings are shown below. Please refer to the pin/port tables for the specific Cubloc module you are using.





ByteOut

ByteOut *portBlock, value*
portBlock : I/O Port Block Number (0 to 15).
value : Value to be output (0 to 255).

`ByteOut` outputs a value to a port block (a group of 8 I/O ports. Refer to `ByteIn`). When using this command, all I/O ports within the specified port block are set to output.

```
ByteOut 1,255       ' Output 255 to Port Block 1.
                    ' Ports 8 through 15 are set to HIGH.
```

NOTE: I/O Port 1 only supports input. Therefore, `ByteOut 0` will not set Port 1 to Output.

CheckBf( )

variable = CheckBf(channel)

variable : Variable to store results (No String or Single)
channel : RS232 Channel (0 to 3)

The command `CheckBf()` can be used to check the current data in the receive buffer without modification. Unlike `Get`, it will not erase the data after reading. Only 1 byte can be read at a time.

```
A = CheckBf(1)              ' Check current data in the receive buffer
```

Compare

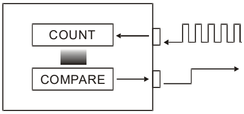
Compare channel, target#, port, targetState

channel : High-speed counter channel

target# : Target # of pulses (CH0: 0 to 65535, CH1: 0 to 255)

port : Output port (DO NOT USE input-only ports)

targetState : Target output port state



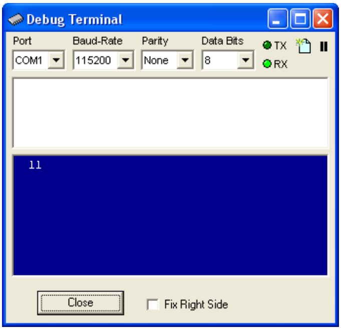
When the high-speed counter value reaches a set target point (`target#`), the processor will set an I/O Port to logic low or logic high.

If `targetState` is set to 1 and the target number of pulses (`target#`) have been received, the output port (`port`) will output logic high. Likewise, if the `targetState` is set to 0 and the target number of pulses (`target#`) have been received, the output port (`port`) will output logic low.

Channel	Compare Range
HCOUNT Channel 0	0 to 65535
HCOUNT Channel 1	0 to 255

The high-speed counter itself supports up to 32-bits, but the `Compare` is limited since this command was designed to not affect the overall multitasking of Cubloc's main processor. Note: For channel 0, please use the `Set Count0 On` command before using `Compare`.

```
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
```



Port: COM1, Baud-Rate: 115200, Parity: None, Data Bits: 8

```
11
```

The above uses high-speed counter channel 0 with a target of 10 pulses. When the counter value becomes 11, the output port (port 61) will output logic high.

Count( )

variable = Count(channel)

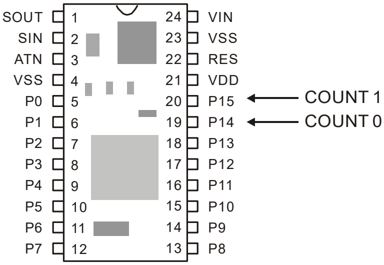
variable : Variable to store results. (No `String` *or* `Single`*)*

channel : Counter channel number (0 to 1)

Return the counted value from the specified counter channel. Please set the counter input ports to input before use of this command (refer to the pin/port table for the appropriate Cubloc module). Data types up to 32 bits in size can be counted (`Byte`, `Integer`, and `Long`). The maximum pulse frequency is 2MHz.

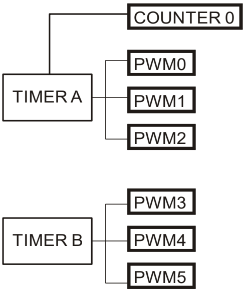
Cubloc's counter is hardware driven, meaning it runs independently from the main program. It is able to count reliably in real-time regardless of how busy the Cubloc processor is.

The Cubloc has 2 counter inputs. Counter channel 0 uses the same resources as PWM0 through PWM2, so you cannot use both at the same time. However, counter channel 1 can be used while PWM channel 0 is running. The `Set Count0 On` command must be executed before using counter channel 1. Counter channel 1 requires no additional settings.



```
Dim R As Integer
Input 15     ' Set port 15 as input. (Counter Channel 1)
R = Count(1) ' Read current counter value.

Set Count0 On       ' Activate counter channel 0
             ' (PWM0,1, and 2 becomes deactivated.)
Input 14     ' Set port 14 as input (Counter Channel 0)
R = Count(0) ' Read current Counter value.
```

As illustrated below, counter 0 uses the same resources as PWM0 through PWM2, so please be careful not to use both at the same time.



```
'
'      Measure frequency from pulse output PWM 0 channel
'
Const Device = CB280
Dim A as Integer
Input 15
Low 5
Freqout 0,2000
Low 0
On Timer(100) Gosub GetFreq
Do
Loop

GetFreq:
A = Count(1)
Debug goxy,10,2
Debug dec5 A
Countreset 1
Reverse 0
Return
```

CountReset

CountReset channel

channel : Counter channel (0 to 1)

Resets the specified counter to 0.

```
CountReset 0         ' Clear Channel 0
CountReset 1         ' Clear channel 1
```

Dcd

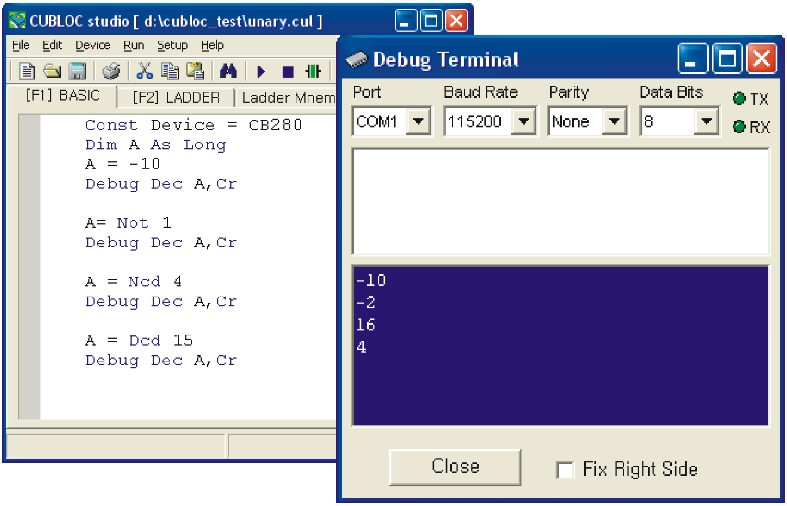
variable = Dcd source

variable : Variable to store results. (No `String` *or* `Single`*)*

source : source value

`Dcd` is the opposite of `Ncd`. It returns the bit position (counting from one) of the most significant bit that is a 1.

```
I = Dcd 15   ' Result is 4 since 15 = 0b00001111
```



Port: COM1, Baud Rate: 115200, Parity: None, Data Bits: 8

□ Fix Right Side

```
Const Device = CB280
Dim A As Long
A = -10
Debug Dec A,Cr

A= Not 1
Debug Dec A,Cr

A = Ncd 4
Debug Dec A,Cr

A = Dcd 15
Debug Dec A,Cr
```

```
-10
-2
16
4
```

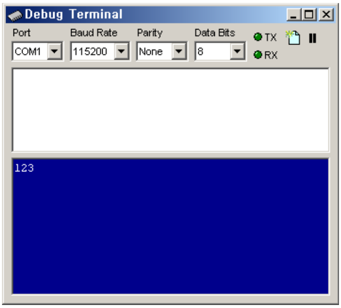
Debug

Debug data

data : data to send to PC

Cubloc supports RS-232 debugging with the `Debug` command. The user can insert `Debug` commands as desired within a program. The result of the `Debug` command is displayed on the Debug Terminal, which will automatically appear after the program is downloaded from Cubloc Studio.

```
Dim A AS Integer
A = 123
Debug  Dec  A
```



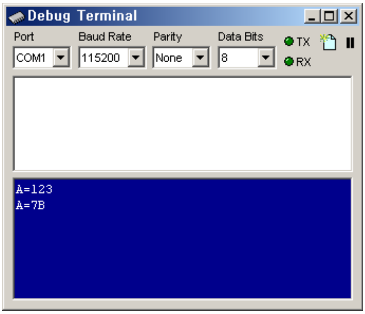
COM1 115200 None 8

```
123
```

Use `Dec` or `Hex` to convert numbers to strings for the `Debug` command. If you do not use `Dec` or `Hex`, numbers will be printed as raw ASCII, usually providing no useful output.

If you insert a question mark (?) before `Dec` or `Hex`, the variable's name will be printed before the value.

```
Debug  Dec?  A,Cr
Debug  Hex?  A,Cr
```

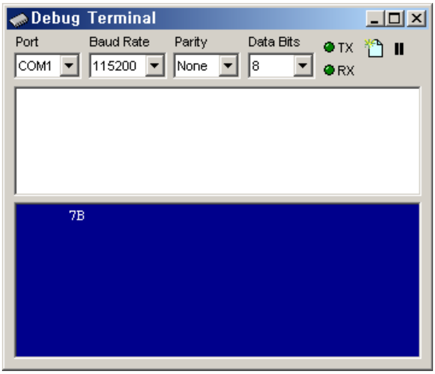


COM1 115200 None 8

```
A=123
A=7B
```

You can also specify the number of characters to print.

```
Debug  Hex8  A
```



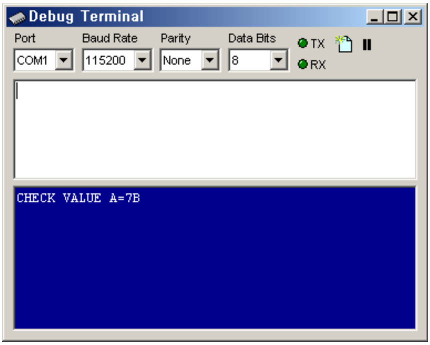
Port: COM1, Baud Rate: 115200, Parity: None, Data Bits: 8

```
      7B
```

The HEX command will accept 1 through 8. HEX8 will print as an 8 digit hexadecimal number. The DEC command will accept 1 through 10.

You are free to mix strings and numbers:

```
Debug  "CHECK VALUE " Hex?  A, Cr
```



Port: COM1, Baud Rate: 115200, Parity: None, Data Bits: 8

```
CHECK VALUE A=7B
```

The `Debug` command is useful for printing strings and numbers in a user-friendly format. During the execution of a Cubloc BASIC program, when the `Debug` command is encountered, the resulting values are immediately displayed on the debug terminal.

If you insert a `Debug` statement into a program and the debug terminal displays output, it verifies that the program has executed to that point. By using `Debug` commands, you will be able to detect the location of bugs in your program, and monitor variables changes in real time.

You can send data to the download port of Cubloc by entering text in the upper text box of the debug terminal. This is useful for interactive communication with the Cubloc.

Warning

The `Debug` command may not be used while monitoring in Ladder Logic. Likewise, Ladder Logic monitoring can not be used while debugging using `Debug` statements.

The following is a chart of commands that can be used with the `Debug` command. You can control the debug terminal's output like an output screen or LCD.

Command	Code	Explanation	Example Usage
CLR	0	Clear Debug screen	Debug CLR
HOME	1	Move cursor to the upper left corner of the Debug screen	Debug HOME
GOXY	2	Move cursor to X, Y	Debug GOXY, 4, 3
CSLE	3	Move cursor one to the left.	
CSRI	4	Move cursor one to the right	
CSUP	5	Move cursor one up	
CSDN	6	Move cursor one down	
BELL	7	Make beeping sound	
BKSP	8	BACK SPACE	
LF	10	LINE FEED	Debug "ABC",LF
CLRRI	11	Erase all characters on the right of cursor to the end of line.	
CLRDN	12	Erase all characters on the bottom of cursor	
CR	13, 10	Carriage Return (go to next line)	Debug "ABC",CR

You must use above commands within a `Debug` statement:

```
Debug Goxy,5,5,Dec I
Debug Clr,"TEST PROGRAM"
```

Decr

Decr variable

variable : Variable to decrement. (No String or Single)

Decrements variable by 1.

```
Decr A       ' Decrement A by 1.
```

Delay

Delay time

time : Time interval (variable or constant up to Long type value)

`Delay` delays program execution for the specified time in milliseconds. The `Delay` command is best used for small amounts of time. We recommend not using it for time measurements and other time-critical applications, as the actual delay time can vary depending on other running tasks.

```
Delay 10      ' Delay for about 10 ms.
Delay 200     ' Delay for about 200 ms.
```

Delay is implemented as the following subroutine.

```
Sub Delay(dl As long)
   dl1 var Long
   dl2 var Integer
   For dl1=0 To dl
     For dl2=0 To 1
              Nop
              Nop
              Nop
     Next
   Next
End Sub
```

Do...Loop

Do...Loop will infinitely loop the enclosed commands unless Do While or Do Until is used to conditionally terminate the loop. An Exit Do statement can also be used within the DO...LOOP to exit from the loop.

```
Do
    Commands
Loop
```

```
Dim K As Integer
Do
    K=Adin(0)            ' Read AD input from channel 0
    Debug Dec K,Cr
    Delay 1000
Loop
```

In the above example, the program will loop indefinitely inside Do and Loop. An Exit Do or GOTO statement must be used to get out of the infinite loop.

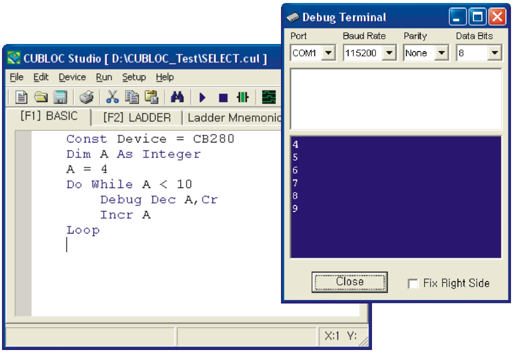
```
Do While [Condition]
        Commands
        [Exit Do]
Loop

Do
        Commands
        [Exit Do]
Loop While [Condition]
```

Do...While will loop indefinitely until the While condition is met.

```
Do Until [Condition]
        Commands
        [Exit Do]
Loop

Do
        Commands
        [Exit Do]
Loop Until [Condition]
```

Do...Until will loop indefinitely until Until condition is met.

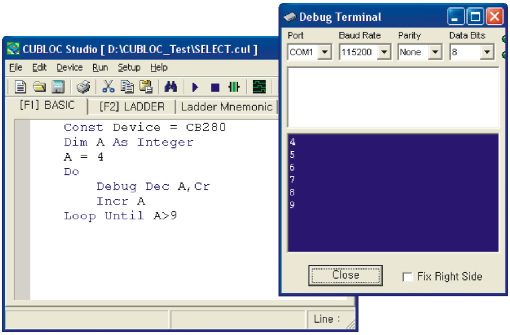
DEMO PROGRAM



Port: COM1 · Baud Rate: 115200 · Parity: None · Data Bits: 8

```
Const Device = CB280
Dim A As Integer
A = 4
Do While A < 10
     Debug Dec A,Cr
     Incr A
Loop
```

```
4
5
6
7
8
9
```



Port: COM1 · Baud Rate: 115200 · Parity: None · Data Bits: 8

```
Const Device = CB280
Dim A As Integer
A = 4
Do
     Debug Dec A,Cr
     Incr A
Loop Until A>9
```

```
4
5
6
7
8
9
```

DTZero

DTZero variable

variable : Variable for decrement. (No String or Single)

`DTZero` decrement variable by 1 as long is variable is greater than 0. If variable is 0, `DTZero` does nothing. This differs from the Decr command, which will underflow the variable and wrap around to the type's highest value

```
DTZero A      ' Decrement A by 1 unless A is 0
```

EERead( )

variable = EERead (Address, ByteLength)

variable : Variable to store result (No String or Single)
address : 0 to 4095
byteLength : Number of bytes to read (1 to 4)

Read data from the specified address in the EEPROM.

```
Dim A As Integer
Dim B As Integer
A = 100
EEWrite 0,A,2  ' Store 2 bytes of A in Address 0.
B = EEReed(0,2)      ' Read 2 bytes from Address 0 and store in B.
```

EEWrite

EEWrite address, data, byteLength

address : 0 to 4095
data : Data to write to EEPROM (up to Long type values)
byteLength : Number of Bytes to write (1 to 4)

Store data in the specified address in EEPROM. This is very useful for storing configuration or calibration data.

```
Dim A As Integer
Dim B As Integer
A = 100
EEWrite 0,A,2       ' Store A in Address 0.
B = EERead(0,2)     ' Read from Address 0 and store in B.
```

When writing to the EEPROM, it takes about 3 to 5 milliseconds.
When reading from the EEPROM, it takes less than 1 millisecond.
There is a physical limit of around 100,000 writes to each location within the EEPROM.

If you are using the EEPROM for data acquisition or data that requires a lot of writes, we recommend using a module with battery-backup memory, such as the CB290 or CB405, instead. One alternative is an RS-232 Compact Flash (CF) or Secure Digital (SD) memory interface module.

The following table compares SRAM and EEPROM.

Type	Battery Backup SRAM	EEPROM
Life of Data	Depends on battery capacity	40 Years
Maximum Writes	Infinite	About 100,000
Writing Time	0 ms	3 to 5 ms
General use	Store often-used variable information over a power outage. Example: daily production counter.	Important data that needs to survive even a backup battery failure. Example: Product Serial Number

EKeyPad()

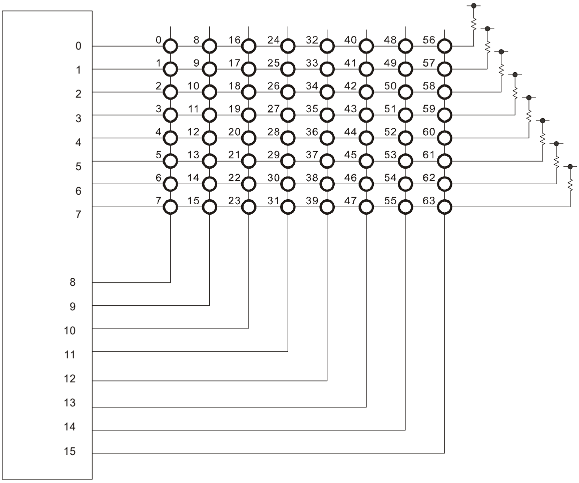
variable = EKeyPad(portblockIn, portblockOut)

variable : Variable to store results (Returns Byte)
portblockIn : Port Block to receive input (0 to 15)
portblockOut : Port Block to output (0 to 15)

`EKeyPad` extends `KeyPad` to read up to 64 key inputs. Two port blocks are used to read a keypad matrix of up to 8x8 lines. The input port block and the output port block must be selected separately.

A pullup resistor (2.2K to 10K) should be connected between each input port and 5V. For ports not used within the input port block, a pullup resistor must be used. Unused ports may not be used for other purposes when using this command.

Ports not used within the output port block can be left unconnected. Unconnected ports may not be used for other purposes. The following is an example showing port block 0 as the input port block and port block 1 as the output port block:



If no keys are pressed, 255 will be returned. Otherwise, the pressed key’s scan code will be returned.

For...Next

For...Next will loop the commands within itself for a set number of times.

```
For Variable = StartingValue To EndingValue [Incremental Step]
       Commands
       [Exit For]
Next
```

In the below example, an Incremental Step is not set. By default each loop increments by 1.

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

For K=10 To 0 Step -1       ' Negative Step, step from 10 to 0.
     Debug Dp(K),CR
Next
```

An Exit For command can be used within the For...Next loop to exit at any time.

```
For K=0 To 10
   Debug Dp(K),CR
   If K=8 Then Exit For      ' If K equals 8 exit the For...Next loop.
Next
```

When choosing a variable to use for the For...Next loop, please make sure the chosen variable is able to cover the desired range. Byte variables can cover 0 to 255. For larger values, a variable with a larger range must be chosen.

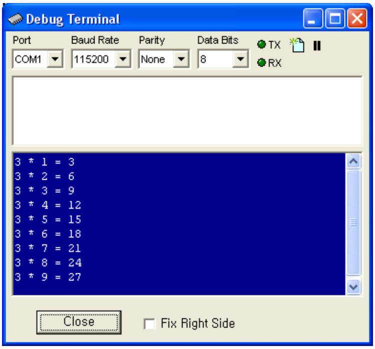
```
Dim K As Byte
For K=0 To 255
   Debug Dp(K),CR
Next
```

When using a negative Step, please choose a Long variable type if the loop will go below 0.

```
Dim LK As Long
For LK=255 To 0 Step -1      ' This will reach -1 as last step
   Debug Dp(LK),CR
Next
```

DEMO PROGRAM

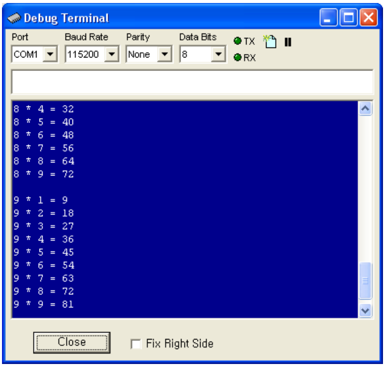
```
Const Device = CB280
Dim A As Integer
For A=1 To 9
    Debug "3 * "
    Debug Dec A
    Debug " = "
    Debug Dec 3*A,Cr
Next
```



Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
```

```
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
```



Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72

9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
```

FreePin

FreePin I/O

I/O : I/O port number

This command will reassign an I/O port back to BASIC, if it had previously been assigned to Ladder Logic with `UsePin`.

FreqOut

FreqOut channel, freqValue

channel : PWM channel (0 to 15)

freqValue : Frequency value between 1 and 65535

FreqOut output the specified frequency to the specified PWM channel. Please make sure to specify the PWM channel, not the I/O port number. For the CB220 and CB280, ports 5, 6, and 7 correspond to PWM Channel 0, 1, and 2, respectively.

The following chart shows several example freqValues and their corresponding frequencies. The highest possible frequency can be achieved by setting freqValue to 1 and the lowest possible can be achieved by setting freqValue to 65535. A freqValue of 0 does not produce any output.

FreqValue	Frequency
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	Frequency
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

You can calculate freqValue using the following formula:

freqValue = 2,304,000 / Desired Frequency

Before using this command, please set the specified PWM port to output mode and set to a logic high or logic low state. To stop the output, use the PWMOff command.

The following is an example:

```
Const Device = CB280
Dim i As Integer
Low 5                ' Set Port 5 to low and output.
i = 1
FreqOut 0,10         ' Produce a 209.3Khz wave
Do                   ' Infinite loop
Loop
```

Since FreqOut uses the same resources as the PWM, there are some restrictions to consider. PWM channels 0, 1, and 2 use the same timer. If

PWM channel 0 is used for a `FreqOut` command, PWM channels 0, 1, and 2 cannot be used for a PWM command.

Likewise, PWM Channels 3, 4, and 5 are linked. If `FreqOut` is used on PWM channel 3, PWM Channels 3, 4, and 5 cannot be used for a PWM command.

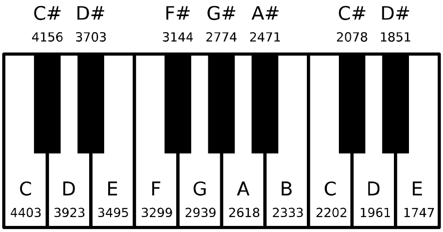
You can produce different frequencies on PWM channels 0 and 3.

To summarize, two different frequencies can be produced at one time, and when using the `FreqOut` command, a PWM command cannot be used on the same channel.

The following is a chart that correlates `freqValue` to musical notes:

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

```
FreqOut 0,5236           ' Note A in Octave 4(440Hz)
FreqOut 0,1469           ' Note G in Octave 5
```



Get( )

variable = Get(channel, length)

variable : Variable to store results (Cannot use String or Single)
channel : RS-232 Channel (0 to 3)
length : Length of data to receive in bytes (1 to 4)

Read data from the RS-232 port. The `Get` command actually reads from the receive buffer. If there is no data in the receive buffer, it will quit without waiting for data and return 0. The `Blen` command can be used to check if there is any data in the receive buffer before trying to read data.

The length of data to be read must be between 1 and 4. If receiving a single `Byte`, it would be 1. If receiving a `Long`, it would be 4. For larger amounts of data, use `GetStr` or `GetA`.

TIPS

Use `Sys(1)` after `Get` or `GetStr` to verify how much data was actually read. If 5 bytes are received but only 4 bytes are verified, 1 byte is lost.

```
Const Device = CB280
Dim A as Byte
OpenCom 1,115200,3,50,10
On Recv1 GoSub GOTDATA
Do
     Do While In(0) = 0
     Loop          ' Wait for a button press (Connect P0)
     Put 1,Asc("H"),1
     Put 1,Asc("E"),1
     Put 1,Asc("L"),1
     Put 1,Asc("L"),1
     Put 1,Asc("O"),1
     Put 1,13,1    ' HELLO + Chr (13) + Chr (10)
     Put 1,10,1
     Do While In(0) = 1
     Loop
Loop

GOTDATA:
     A=Get(1,1)
     Debug A
     Return
```

GetA

GetA channel, arrayName, byteLength

channel : RS-232 Channel (0 to 3)

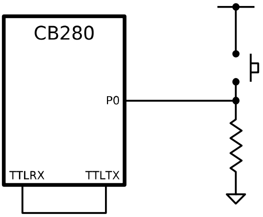
arrayName : Array to store received data (Byte type only)

byteLength : Number of bytes to store (1 to 65535)

The command `GetA` can be used to store received RS-232 data in a `Byte` array. Data will be stored starting from the first element of the array. Check the receive buffer with `BLen` before reading to avoid overflowing the buffer.

```
Const Device = CB280
Dim A(10) As Byte
OpenCom 1,115200,3,50,10
Set Until 1,8, 10  '10 is a required dummy value, it will be ignored
On Recv1 Gosub GOTDATA
Do
     Do While In(0) = 0
     Loop           ' Wait until press button (Connect P0)
     PutStr 1,"CUBLOC",Cr
     Do While In(0) = 1
     Loop
Loop

GOTDATA:
     GetA 1,A,8
     Debug A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7)
     Return
```

In order to run the program above, please connect Rx to Tx as shown below.



GetA2

GetA2 channel, arrayName, byteLength, stopChar

channel : RS-232 Channel (0 to 3)
arrayName : Array to store received data (Byte type only)
byteLength : Number of Bytes to store (1 to 65535)
stopChar : Stop character ASCII code

GetA2 is the same as GetA except it will stop reading data at the when it encounters stopChar, even if the data received is less than byteLength. If stopChar is not found, then it will operate just like GetA.

stopChar is included in the received data.

You can use the Sys(1) command to check the number of bytes read:

```
Dim A(10) As Byte
OpenCom 1,19200,0,50,10
GetA2 1,A,20,10  ' Read until stop character ASCII code 10 is found
                 ' or 20 bytes have been read
```

GetCrc

GetCrc variable, arrayName, byteLength

variable : String variable to store results (Integer type)
arrayName : Array with data(Must be a Byte array)
byteLength : number of bytes to calculate CRC

`GetCrc` calculates a Circular Redundancy Check (CRC) when using MODBUS RTU Master Mode. `GetCrc` will return a 16-bit integer CRC value of the specified array, `arrayName`. You can set the number of bytes to use for the CRC calculation from the the array starting at 0.

```
 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 ' Store in variable B the CRC for 6 bytes of array A
Debug Hex B,Cr
```

NOTE: Please use byte arrays when using this function.

GetPad( )

variable = GetPad(length)

variable: Variable to store results

length: Length of data to receive

Reads `length` bytes from the pad buffer allocated with `Set Pad`.

```
Const Device = CB280

Dim X As Integer
Dim Y As Integer

Set Pad 0,4,20       'Mode 0, Packet size 4 bytes, Buffer 20 bytes
On Pad GoSub ABC     'Define ABC as interrupt service routine

Do                   'Run forever
Loop

ABC:
   X = GetPad(2)     'First 2 bytes is the x coordinate
   Y = GetPad(2)     'Second 2 bytes is y coordinate
   Debug Dec X, Cr   'Print x coordinate
   Debug Dec Y, Cr   'Print y coordinate
   Return
```

GetStr( )

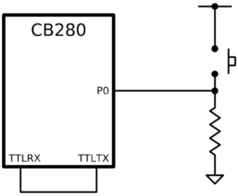
variable = GetStr(channel, length)

variable : String Variable to store results
channel : RS-232 Channel
length : Length of data to receive

Same as `Get`, except the variable to store results can only be a `String` and the length of data is not limited to 4 bytes.

```
Const Device = CB280
Dim A As String * 10
OpenCom 1,115200,3,50,10
Set Until 1,8, 10  '10 is a required dummy value, it will be ignored
On Recv1 GoSub GOTDATA
Do
             Do While In(0) = 0
             Loop                ' Wait until press button (Connect P0)
             PutStr 1,"CUBLOC",Cr
             Do While In(0) = 1
             Loop
Loop

GOTDATA:
             A = GetStr(1,8)
             Debug A
             Return
```

In order to run the program above, please connect Rx to Tx as shown below.



GetStr2( )

variable = GetStr(channel, byteLength, stopChar)

variable : String variable to store results
channel : RS-232 Channel
byteLength : Length of data to receive
stopChar : Stop character ASCII code

Same as the `GetStr` command, except it will stop reading data when it encounters `stopChar`, even if the data received is less than byteLength. If `stopChar` is not found, then it will operate just like `GetStr`.

GoSub...Return

The GoSub command can call a subroutine. The Return command must be used at the end of the subroutine.

```
GoSub ADD_VALUE

ADD_VALUE:
     A = A + 1
     Return
```

GoTo

The `GoTo` command will instruct the current program to jump to a specified label. This is part of every BASIC language, but its use should be limited in an effort to adopt a structural programming paradigm. Be especially careful when using GoTo within a GoSub or subroutine since improperly terminating a subroutine can have undesired effects.

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

LAB1:
    I = 3
```

About Labels...

A label can be set with a colon (:) to specify a point for `GoTo` or `GoSub` to begin execution.

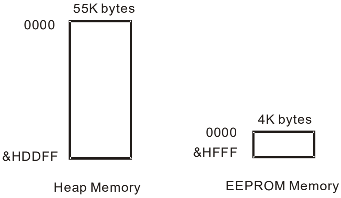
```
ADD_VALUE:
LINKPOINT:
```

Labels cannot use reserved keywords, numbers, or include blank spaces. The following labels are not permitted:

```
Ladder:                 'Reserved keyword
123:                    'Number.
About 10:               'Blank space.
```

Heap Memory Access

Heap memory access is a special feature only available on the CB405 module. The user may use 55KB of heap memory from address 0 through 56831 (&H0000 through &HDDFF). The heap can be used to store large data for graphics, temperature tables, etc. With a backup battery, the heap can be used for data logging and other persistent uses.



There are five heap-related functions:

Function	Syntax	Feature
`HeapClear`	`HeapClear`	Erase the entire heap.
`HRead`	`variable = HRead(address, length)`	Read the designated number of bytes set by l`ength` from heap address, `address`, and store into `variable`.
`HWrite`	`HWrite address, variable, length`	Store `length` number of bytes of `variable` to the heap address, `address`.
`HeapW`	`HeapW address, variable`	Store one byte, `variable`, to the heap address, `address`.
`Heap`	`variable = Heap(address)`	Read one byte from the heap address, `address`, and store into `variable`.

HRead( )

variable = HRead (address, byteLength)

variable : Variable to store results
address : Heap address
byteLength : Number of bytes to read, constant or variable (1 to 4)

Read the designated number of bytes set by `length` from heap address, `address`, and store into `variable`.

HWrite

HWrite address, data, byteLength

address : Heap address
data : Constant or variable with data (whole numbers only)
byteLength : number of bytes to write

Store `length` number of bytes of `variable` to the heap address, `address`.

```
Dim A As Integer
Dim B As Integer
A = 100
HWrite 0,A,2          ' Write integer A to address 0.
B = HRead(0,2)        ' Read from address 0 and store in B.
```

NOTE

`EERead` and `EEWrite` have same syntax as `HRead` and `HWrite`.

Function	Memory Type	Feature
`EERead`, `EEWrite`	EEPROM	Retains data during power cycles without a battery. The `EEWrite` command takes about 5ms. 4KB of available memory
`HRead`, `HWrite`	SRAM	Retains data during power cycles with a backup battery. Without a backup battery, data is lost. `HWrite` command takes about 20 micro-seconds to execute, much faster compared to `EEWrite`. 55KB of available memory

HeapClear

HeapClear

Set all 55KB of the heap to zero.

Heap( )

variable = Heap (address)

variable : Variable to store results
address : Heap address

Returns 1 byte of data from the specified heap address.

HeapW

HeapW address, data

address : Heap memory address
data : Constant or variable with data (Byte only)

Write 1 byte of data to the specified heap address.

Heap Memory Addressing

The heap is divided into byte unit addresses. When a `Long` variable is stored, 4 bytes are stored, and 4 memory addresses are used.

```
  HWrite 0, &H1234ABCD, 4
```

0	&HCD
1	&HAB
2	&H34
3	&H12

As you can see in the above table, when a `Long` variable is stored in heap address 0, four memory addresses are taken.

```
  HWrite 0, &HABCD, 2        ' Write &HCD to address 0 and &HAB to address
1
  HWrite 1, &H6532, 2        ' Overwrite address 1 with &H32 and write
                       ' &H65 to address 2
```

DEMO PROGRAM

```
Const Device = CB405
Dim A As Byte
Dim i As Long,J As Long

i = &HABCD1234
HeapClear
HWrite 0,i,4
Do
  HeapW 56830,100
  HeapW 56831,123

  Debug Dec Heap(56830),Cr
  Debug Dec Heap(56831),Cr
  J = HRead(0,4)
  Debug Hex J,Cr
  Delay 100
Loop
```

High

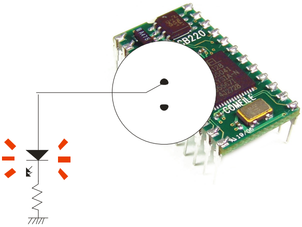
High port

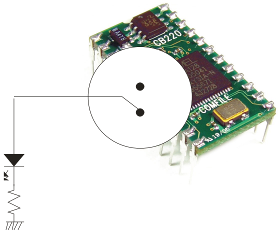
port : I/O port number

Set the port to a logic high state, 5V.

```
Output 8      ' Set Port 8 to output state.
High 8         ' Set Port 8 to HIGH (5V).
```

When a port is set to high, the port is internally connected to VDD (5V). If it’s set to Low, the port is internally connected to VSS (0V). This allows either source or sink interfacing to external components (up to 25ma for source or sink).

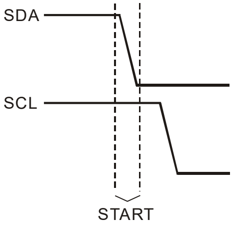




I2CStart

I2CStart

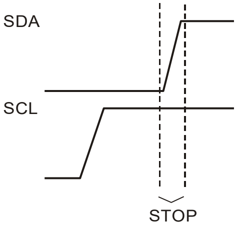
Sets I2C SDA and SCL to start mode. After this command, SDA and SCL go low.



I2CStop

I2CStop

Set I2C SDA and SCL to stop mode. After this command, SDA and SCL go high.



I2CRead( )

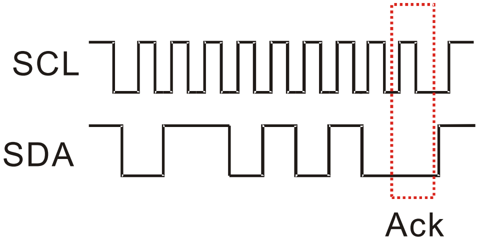
variable = I2CRead(dummy)

variable : Variable to store results. (No String or Single)

dummy : Dummy value.

Read a byte from the I2C ports set by `SET I2C` command. Use any value for the dummy value.

```
A = I2CRead(0)
```



This command will send an ACK signal back to the slave I2C device. After reading a byte, an SCL pulse will be sent while SDA is kept low.

I2CReadNA( )

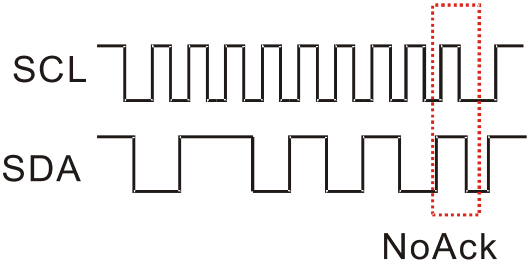
Variable = I2CReadNA (dummy)

variable : Variable to store results. (No String or Single)

dummy : Dummy value. (Normally 0)

Functions the same as the `I2CRead` command but without acknowledgment.

```
A = I2CReadNA(0)
```



I2CWrite( )

variable = I2CWrite data

variable : Acknowledge (0=Acknowledgment, 1=No Acknowledgment)
data : data to send (Byte value : 0 to 255)

Sends one byte of data through I2C. This command creates an ACK pulse and returns 0 if there is an acknowledgment and 1 if there isn’t. No acknowledgment indicates that there was a communication error (possibly due to incorrect wiring). The following is an example illustrating how this can be used to trigger an error processing function:

```
If I2CWrite(DATA)=1 Then GoTo ERR_PROC
```

If you don’t need to check for an ACK, you can just use any variable to receive the ACK status as shown below:

```
A = I2CWrite(DATA)
```

One byte of data transfer takes approximately 60 microseconds.

Please refer to Chapter 8 "About I2C…" for a detailed description of I2C communications.

If...Then...ElseIf...EndIf

You can use `If…Then…ElseIf...Else…EndIf` conditional statements to control execution of your program.

```
If  Condition1 Then [Expression1]
      [Expression2]
[ElseIf Condition2 Then
      [Expression3]]
[Else
      [Expression4]]
[End If]
```

Usage 1

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

Usage 2

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

Usage 3

```
If  A<10  Then   ' When using more than 1 line,
   B=1           ' do not put any Expressions after "Then".
End If
```

Usage 4

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

Usage 5

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

Usage 6

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

In( )

variable = In(port)

variable : The variable to store result (No String or Single)
port : I/O port number (0 to 255)

Read the current state of the specified port. This function reads the state of the I/O port and stores it in the specified variable. When this command is executed, Cubloc will automatically set the port to input and read from the port. You do not need to use the `Input` command to set the port beforehand when using this command.

```
Dim A As Byte
A = In(8)     ' Read the current state of port 8
              ' and store in variable A(0 or 1)
```

TIP

By default, all I/O ports are set to a high-Z (high impedance) input state at power ON. When a port is set to output, it will either output a high or low signal; high is 5V and low is 0V or GND (ground).

Incr

Incr variable

variable : Variable to increment. (No String or Single)

Increment the variable by 1.

```
Incr A                    'Increment A by 1.
```

Input

Input port

port : I/O port number (0 to 255)

Sets the specified port to a high-Z (high impedance) input state.
All I/O ports of Cubloc modules are set to a high-Z input state by default at power on.

High impedance means that the value of resistance is so high that it’s neither high nor low; it won’t affect a circuit attached to the port.

```
Input 8      'Set port 8 to a high-Z input state.
```

KeyIn

variable = KeyIn(ort, debouncingTime)

variable : Variable to store results (No String or Single)

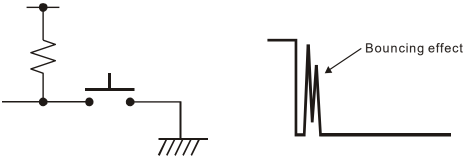
port : Input port (0 to 255)

deboucingTime : Debouncing time (1 to 65535)

The command `KeyIn` removes the input signal's contact bounce before reading an input. You can use `KeyIn` only with active low inputs as shown below. For active high inputs, please use `KeyInH`. When a button press is detected, `KeyIn` will return 0; otherwise, it will return 1.

If you use 10 for the deboucing time, the Cubloc will debounce for 10 ms. Contact bounce usually stops after 10ms, so a 10ms debouncing time will suffice for most applications

```
A = KeyIn(1,10) 'Read from port 1 after waiting 10ms for debouncing.
```



KeyInH

variable = KeyInH(port, debouncingTime)

variable : Variable to store results (No String or Single)
port : Input port (0 to 255)
deboucingTime : Debouncing time (0 to 65535)

`KeyInH` is for active high inputs. For active low inputs, the `KeyIn` command should be used.

When a button press is detected, `KeyInH` will return 1; otherwise, it will return 0.

```
A = KeyInH(1,100) 'Read from port 1 after waiting 100ms for debouncing.
```

Keypad

variable = Keypad(portBlock)

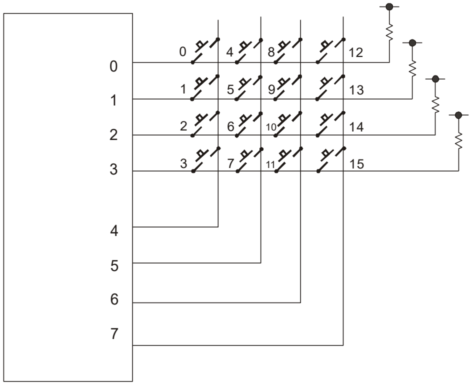
variable : Variable to store results (Returns Byte, No String or Single)

portBlock : Port block (0 to 15)

Use this command to read input from a matrix keypad. One port block can be used to read a 4 by 4 keypad input. The keypad rows should be connected to the lower 4 bits of the Port Block, and the keypad columns should be connected to upper 4 bits of the port block.

Pullup resistors (2.2K to 10K) should be connected to the lower 4 bits of the port block. A resistor should be connected even if a row is not being used.

Please refer to the diagram below:



```
A = Keypad(0) ' Read the status of keypad connected to Port Block 0
```

If no keys are pressed, 255 will be returned. Otherwise, the pressed key's scan code will be returned.

Low

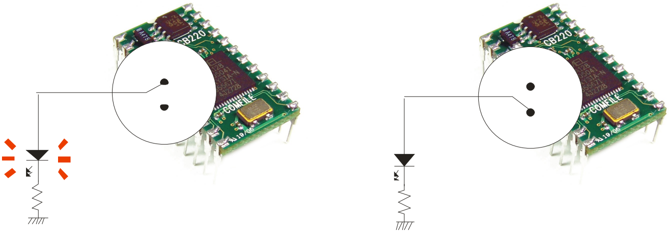
Low port

port : I/O port number (0 to 255)

This command sets the port's I/O mode to output and outputs logic low or 0V (GND).

```
Output 8              'Set port 8's I/O mode to output
LOW 8                 'Set logic low (0V) on port 8.
```

When a port is set to logic high, the port is internally connected to VDD (5V). If it is set to a logic low, the port is internally connected to VSS (0V). This allows either source or sink interfacing to external components (up to 25mA for source or sink).



MemAdr( )

variable = MemAdr (targetVariable)

variable : Variable to store results (No String or Single)

targetVariable : Variable whose physical memory address should be returned

The `MemAdr` command will return the memory location of the specified target variable. This can be useful when used with the `Peek` and `Poke` commands; operations similar to C pointer manipulation can be performed.

```
Dim A as Single
Dim Adr as Integer
Adr = MemAdr(A)     'Return the physical address of A.
```

`MemAdr` does not work with arrays.

Ncd

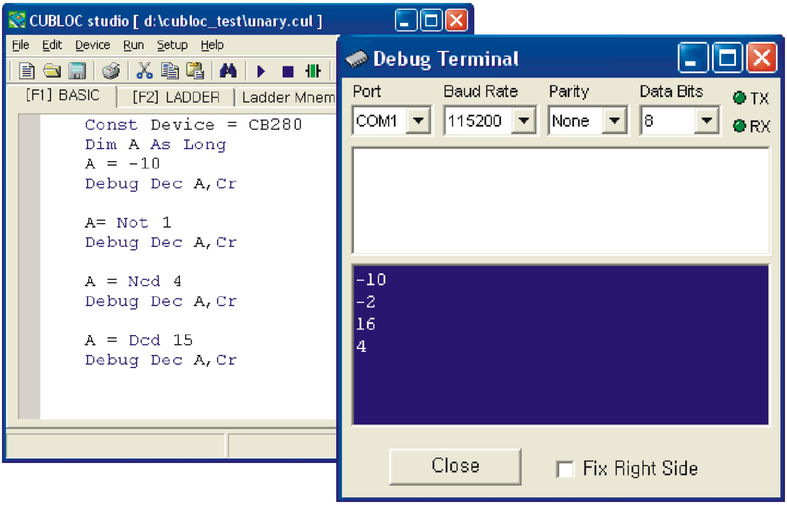
variable = Ncd bitPosition

variable : Variable to store results. (No String or Single)

bitPosition : Position of bit (0 to 31)

The Ncd command is used to return a value with the specified bit set to 1.

```
I = Ncd 0 'Result is 00000001  = 1
I = Ncd 1 'Result is 00000010  = 2
I = Ncd 2 'Result is 00000100  = 4
I = Ncd 3 'Result is 00001000  = 8
I = Ncd 4 'Result is 00010000  = 16
I = Ncd 5 'Result is 00100000  = 32
I = Ncd 6 'Result is 01000000  = 64
I = Ncd 7 'Result is 10000000  = 128
```



```
Const Device = CB280
Dim A As Long
A = -10
Debug Dec A,Cr

A= Not 1
Debug Dec A,Cr

A = Ncd 4
Debug Dec A,Cr

A = Dcd 15
Debug Dec A,Cr
```

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
-10
-2
16
4
```

Nop

Nop

This command does nothing, but consumes one command cycle. It is useful for tuning small intervals.

```
Low 8
Nop
High 8       'Output very short pulse to port 8. (About 50 micro Sec)
Nop
Low 8
```

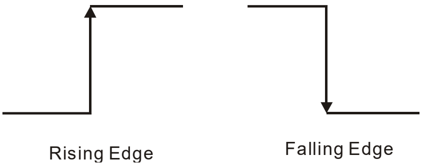
On Int

On Int0 GoSub label
On Int1 GoSub label
On Int2 GoSub label
On Int3 GoSub label

This command must be called before accepting external interrupts. Cubloc has 4 external interrupt ports. The interrupt ports can be set to sense input on the rising edge, falling edge, or both.

`Set OnInt`x command must be used with this command in order for the interrupt to work.

*CB220 has no external interrupt inputs.



```
Dim A As Integer
On Int0 GoSub GETINT0
Set Int0 0            'Falling Edge Input
Do
Loop

GETINT0:
A=A+1                 'Record number of interrupts
Return
```

On LadderInt GoSub

On LadderInt GoSub label

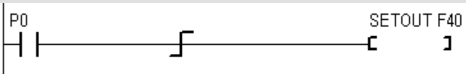
If Register F40 turns on in Ladder Logic, and the `On LadderInt GoSub` command is used, then the processor will jump to the routine (`label`) specified by `On LadderInt`.

This can be used when a Ladder Logic program needs to trigger a specific procedure in BASIC.

Please use the `SetOut` and `DIFU` command to write 1 to the register F40. When the BASIC interrupt routine is finished, register F40 can be cleared by writing a zero to it.

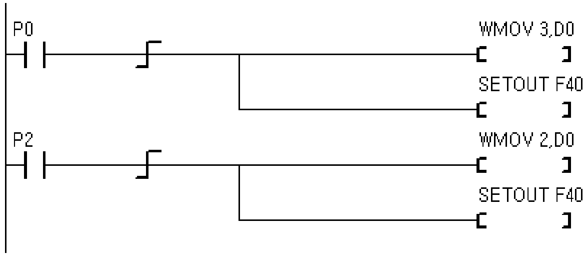
During the interrupt service routine(ISR) execution, writing a 1 to register F40 will not allow another interrupt to occur. If register F40 is cleared from BASIC, it signals the end of the ISR and can process another interrupt.

```
  UsePin 0,In
  Set Ladder On
  Set Display 0,0,16,77,50
  On LadderInt GoSub msg1_rtn
  Dim i As Integer
  Low 1

  Do
    i=i+1
    ByteOut 1,i
    Delay 200
  Loop
msg1_rtn:
  Locate 0,0
  Print "ON LadderInt",Dec i
  Reverse 1
  Return
```



When P0 turns on, register F40 turns on and the msg1_rtn interrupt routine in BASIC is executed. In the ISR, a `String` is printed to the LCD.

Although there is only one register, F40, with which call an ISR in BASIC from Ladder Logic, we can use data register D to process many different types of interrupts.

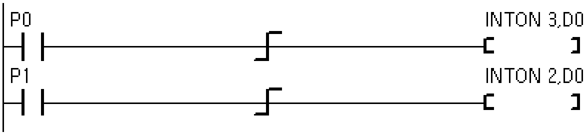


Given the ladder above, when P0 turns on, D0 gets 3 and the interrupt routine is executed. If P2 turns on, D0 gets 2 and the interrupt routine is executed. In the ISR, the user can then process the type of interrupt based on the value stored in D0.

```
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
```

For a short version of the ladder above the user can use an `INTON` command, which accomplishes both WMOV and SETOUT in one command.

The following is an equivalent shortened version of the ladder above:



On Pad

On Pad GoSub label

The `On Pad` interrupt will jump to the specified label when the keypad/touchpad port receives a packet (packets sizes are assigned by the `Set Pad` command). Please be sure to use a `Return` command after the label.

```
Const Device = Ct1720
Dim TX1 As Integer, TY1 As Integer
Contrast 450
Set Pad 0,4,5
On Pad GoSub GETTOUCH
Do
Loop

GETTOUCH:
  TX1 = GetPad(2)
  TY1 = GetPad(2)
  CircleFill TX1,TY1,10
  PulsOut 18,300
  Return
```

On Recv

On Recv0 GoSub label
On Recv1 GoSub label
On Recv2 GoSub label
On Recv3 GoSub label

When data is received on one of Cubloc's RS-232 channels, this command will jump to the specified label (`Recv0` for channel 0, `Recv1` for channel 1, etc...). The processor will automatically check for received data and trigger interrupts when this command is used.

```
Dim A(5) As Byte
OpenCom 1,19200,0, 100, 50
On Recv1 DATARECV_RTN     ' Jump to DATARECV_RTN when RS232 channel
                          ' 1 receives any data

Do
Loop         ' Infinite Loop

DATARECV_RTN:
  If BLen(1,0) > 4 Then
   A(0) = Get(1,1) ' Read 1 Byte.
   A(1) = Get(1,1) ' Read 1 Byte.
   A(2) = Get(1,1) ' Read 1 Byte.
   A(3) = Get(1,1) ' Read 1 Byte.
   A(4) = Get(1,1) ' Read 1 Byte.
  End If
Return       ' End of interrupt routine
```

IMPORTANT

When a `Recv` interrupt service routine (ISR) is being executed, subsequent `Recv` interrupts will not be received. After the ISR is finished executing, if there is still data being received, another `Recv` interrupt will be generated, and the ISR will execute again.

On Timer()

On Timer(interval) GoSub label

interval : Interrupt interval 1=10ms, 2=20ms......65535=655350ms

1 to 65535 can be used

`On Timer` can be used to repeatedly execute an interrupt routine on a specified interval. Set the desired interval in increments of 10 milliseconds, and a label to jump to when interrupt occurs.

```
On Timer(100) GoSub TIMERTN
Dim i As Integer

I = 0

Do
Loop

TIMERTN:
Incr i             ' i is incremented by 1 every second.
Return
```

IMPORTANT

Be aware that ithe interrupt service routine (ISR) must require less time to execute than the interval itself. If the interval is set to 10ms, the ISR must execute within 10 ms; otherwise collisions can occur.

OpenCom

OpenCom channel, baudRate, protocol, recvSize, sendSize

channel : RS232 Channel (0 to 3)
baudRate : BaudRate (Do not use a variable)
protocol : Protocol (Do not use a variable)
recvSize : Receive buffer size (Max. 1024, Do not use a variable)
sendSize : Send buffer size (Max. 1024, Do not use a variable)

This command must be used to enable RS-232 communication.

The Cubloc has 2 or 4 channels for RS-232C communication, depending on the model. Channel 0 is typically used for monitoring and downloading, but it can also be used for RS-232 communication if monitoring is not needed. Downloading will still work regardless.

You may use any value between 2400 to 230400, but we are recommed one of the following values.

2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400

For the protocol parameter, please refer to the table below:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			Parity		Stop Bit	Bit	# of Bits
			0	0 = NONE	0=1 Stop Bit	0	0 = 5 bit
			0	1 = Reserve*	1=2 Stop Bits	0	1 = 6 bit
			1	0 = Even		1	0 = 7 bit
			1	1 = Odd		1	1 = 8 bit

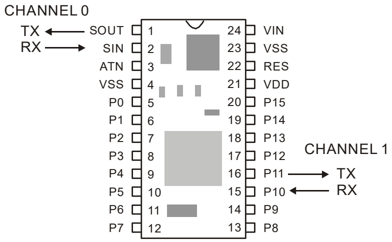
The following table shows typical settings based on the previous table:

Bits	**Parity**	**Stop Bit**	**Value to Use**
8	NONE	1	3
8	EVEN	1	19 (Hex = 13)
8	ODD	1	27 (Hex = 1B)
7	NONE	1	2
7	EVEN	1	18 (Hex = 12)
7	ODD	1	26 (Hex = 1A)

```
OpenCom 1, 19200, 3, 30, 20  'Set to 8-N-1
```

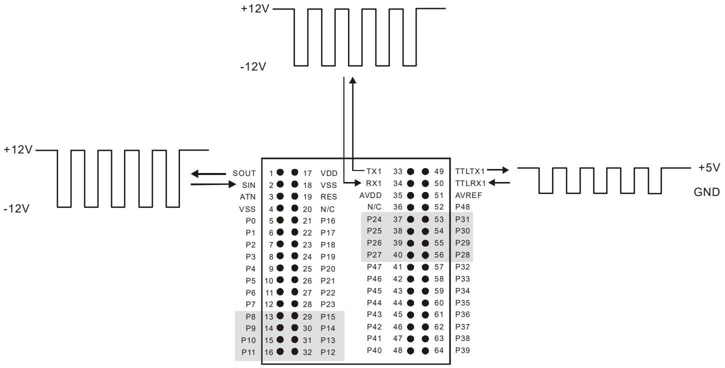
The send and receive buffer size can be set with the `OpenCom` command. Each buffer can be as large as 1024 bytes, but be aware that the send and receive buffers consume data memory. So the larger the buffers, the fewer number of variables can be used in the program. Receive buffer sizes from 30 to 100 and send buffer sizes from 30 to 50 will suffice for most programs.

For the CB220 module, pins 1 and 2 can be used for channel 0. Ports 10 and 11 can be used for channel 1.



For the CB280 module, there are dedicated RS-232 ports. For channel 1, there are 2 types of outputs: +/-12V and TTL (+5/0V).

Please make sure to use only one of them at a time.



*If necessary, the `Set RS232` command can reset the RS-232 channel's configuration during program execution.

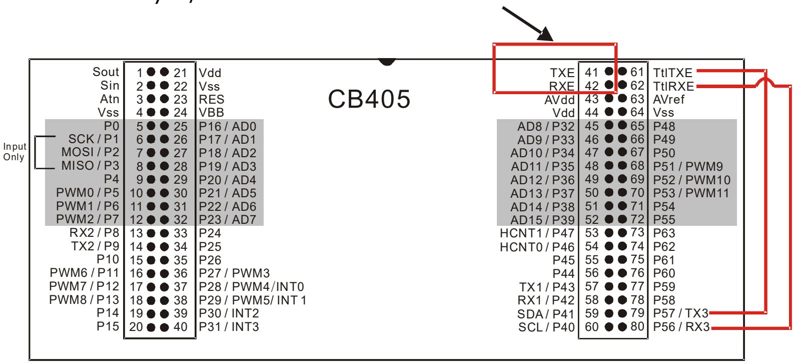
CB400/CB405 RS232 How-to

The following is a table of the 5V TTL signal pins for the CB405:

Channel	I/O Port	5V TTL
1	P42	RX
	P43	TX
2	P8	RX
	P9	TX
3	P56	RX
	P57	TX

The CB400/CB405 has an internal MAX232 that can be used to convert any of the 5V TTL signals to +/- 12V level signals. The following is an example for connecting channel 3:

This allows any +/- 12V RS232 device to connect to TXE and RXE.



Out

Out port, value

port : I/O port number (0 to 255)
value : Value to output to the specified I/O port (1 or 0)

This command outputs a 1 (logic high) or a 0 (logic low) to the specified Port. When you execute this command, the Cubloc will automatically set the port's I/O mode to output. If using the `Out` command it is not necessary to use the `Output` command to set the port beforehand.

```
Out 8,1             'Output a logic high signal on port 8.
             '(This is same as using command High 8)

Out 8,0             'Output a logic low signal on port 8.
             '(This is same as using Low 8)
```

Output

Output port

port : I/O port number (0 to 255)

Set the port's I/O mode to output. All Cubloc module's I/O ports are set to High-Z input by default at power on.

```
Output 8      'Set port 8's I/O mode to output.
```

You can also use the `High` or `Low` commands to set a port's I/O mode to output. When using the `Output` command, the port's output (logic high or logic low) is not clearly defined. Therefore, it recommended to use the `High` or `Low` commands to set a port's I/O mode so the port's output is properly initialized.

```
Low 8         'Set port 8's I/O mode to output and output a logic low.
```

OutStat( )

variable = OutState(port)

variable : Variable to store results. (No String or Single)

port : I/O port number (0 to 255)

Reads the current output value for the specified port. This command is different from the `In` command; it reads a port's current output, not input.

```
Dim A As Byte
A = OutStat(0) 'Read from port 0 and store the output in A.
```

Pause

Pause value

Same as `Delay`

Peek( )

variable = Peek (address, length)

variable : Variable to store result.s (No String or Single)
address : RAM address.
length : Number of bytes to read (1 to 4)

Reads the specified length of data starting from the specified data memory address.

Poke

Poke address, value, length

address : RAM address
value : The value to write
length : Number of bytes to write (1 to 4)

Write the specified length of data starting at the specified data memory address.

```
Const Device = CB280
Dim F1 As Single, F2 As Single
F1 = 3.14
EEWrite 10,Peek(MemAdr(F1),4),4
Poke MemAdr(F2),EERead(10,4),4

Debug Float F2,CR
```

PulsOut

PulsOut port, period

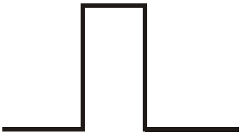
port : Output port (0 to 255)

period : Pulse period (1 to 65535)

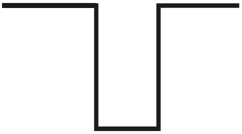
This is a subroutine that outputs a pulse. To create a high pulse, the output port must be set to logic low beforehand. To create a Low pulse, the output Port must be set to logic high before hand.

If you set the pulse period to 10, you will create a pulse of about 2.6ms. Likewise, a Pulse Period of 100 will be about 23ms.

```
Low 2
PulsOut 2, 100 '23mS high pulse
```



```
High 2
PulsOut 2, 100  '23mS low pulse
```



`PulsOut` is a premade system subroutine.

```
Sub PulsOut(pt As Byte, ln As Word)
    Dim dl1 as integer
    Reverse pt
    For dl1=0 to ln
    Next
    Reverse pt
End Sub
```

Put

Put channel, data, byteLength

channel : RS-232 channel (0 to 3)
data : Data to send (up to Long type value)
byteLength : Length of data in bytes (1 to 4)

This command sends data on the specified RS-232 channel. For `data`, variables and constants can be used. To send a `String`, please use the `PutStr` command instead.

IMPORTANT
The `OpenCom` command must be used beforehand

```
OpenCom 1,19200,0,50,10
Dim A As Byte
A = &HA0
Put 1,A,1 ' Send &HA0 (0xA0)
          ' to RS232 Channel 1.
```

The data is first stored in the send buffer set by `OpenCom`. The Cubloc BASIC Interpreter will automatically keep transmitting the data until the send buffer is empty.

If the send buffer is full when the `Put` command is executed, `Put` will overwrite the data currently in the buffer. The `BFree` command can be used to check the send buffer beforehand to prevent overwriting existing data in the send buffer.

```
If BFree(1,1) > 2 Then  ' If send buffer has at least 2 bytes free
     Put 1,A,2
End If
```

`Bfree` is used to check how much free space the send buffer has before writing any data to it.

TIP

After using `Put` or `PutStr`, the function `Sys(0)` can be used to verify that the data has been stored in the send buffer.

```
OpenCom 1,19200,0,50,10
PutStr 1,"COMFILE"
Debug Dec Sys(0)    ' If output is 7, all data has been stored
                    ' in the send buffer
```

*Please refer to the `On Recv` interrupt service routine for receiving data using the hardware serial buffer.

PutA

PutA channel, array, byteLength

channel : RS-232 channel. (0 to 3)
array : The Byte array to send
byteLength : Number of bytes to send (1 to 65535)

The command PutA is used to send a Byte array on the specified RS-232 channel. The array data will be sent starting from the first element of the array.

```
Dim A(10) As Byte
OpenCom 1,19200,0,50,10
PutA 1,A,10          ' Send 10 Bytes of Array A
```

IMPORTANT

If byteLength is larger than the array, Cubloc will send random values.

*Please refer to the On Recv interrupt routine for receiving data using the hardware serial buffer.

PutA2

PutA2 channel, array, byteLength, stopChar

channel : RS-232 channel. (0 to 3)
array : The byte array to send
byteLength : Number of bytes to send (1 to 65535)
stopChar : Stop character ASCII code

Same as the `PutA` command, except it will stop transmission when the specified stop character in encountered in the array (`stopChar` will be the last character to be sent).

PutStr

PutStr channel, data…

channel : RS-232 channel. (0 to 3)
data : String data (String variable , String constant or Constant)

Sends String data on the specified RS-232 channel.

```
OpenCom 1,19200,0,50,10
PutStr 1,"COMFILE TECHNOLOGY", Dec I, Cr
```

Similar to the `Put` command, `PutStr` puts data to be sent in the RS-232 port's send buffer. Afterward, the Cubloc BASIC interpreter takes care of the actual sending. Use the `BFree` command to prevent overloading the send buffer.

Pwm

Pwm channel, duty, period

channel : PWM channel (0 to 15)
duty : Duty cycle (must be less than the period)
period : Maximum of 65535

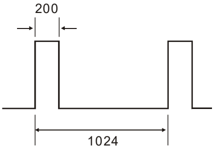
Outputs a pulse waveform whose shape is determined by the values of `duty` and `period`. Be aware that the PWM channel is different from the I/O port number. For the CB280, ports 5, 6, and 7 are used for PWM 0, 1, and 2. Before using `Pwm`, make sure to set the ports' I/O mode to output, and set their output to a known state (logic low or logic high).

Depending on the value of `period`, a PWM signal of up to 16 bit precision is generated. A period of 1024 is a 10 bit pulse and a period of 65535 is a 16 bit pulse. The pulse's actual frequency in Hz can be computed with the following formula.

Frequency = 2,304,000 / `period`

`duty` must to be less than `period`. The pulse's output will remain active for a fraction of the period given by `duty`/`period`.

`Pwm` is independently hardware driven within the Cubloc. Once the `Pwm` command is executed, it will keep running until the `PwmOff` command is called.



```
Low 5                ' Set port 5 output and output LOW signal.
Pwm 0,200,1024       ' Output 10-bit pulse with duty of 200 and
                     ' width of 1024
```

IMPORTANT

PWM channels 0, 1, and 2 must use the same value for `period` since they share the same resources, but their duty cycles can be different. PWM channels 3, 4, and 5 also must use the same value for `period`, but again, their duty cycles can be different.

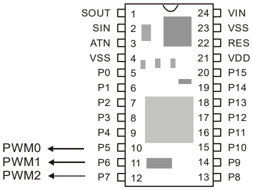
PwmOff

PwmOff channel

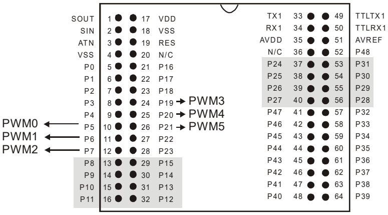
channel : PWM channel. (0 to 15)

Stops the PWM output on the specified PWM channel.

The following illustrates the PWM channels available on each module:



For CB220, 3 PWM channels are provided on ports P5, P6, and P7.



Please refer to the table below for PWM channels and their corresponding I/O ports.

	CB220	CB280	CB290	CT17X0	CB400/ CB405
PWM0	I/O 5	I/O 5	I/O 5	I/O 8	I/O 5
PWM1	I/O 6	I/O 6	I/O 6	I/O 9	I/O 6
PWM2	I/O 7	I/O 7	I/O 7	I/O 10	I/O 7
PWM3		I/O 19	I/O 89	I/O 11	I/O 27
PWM4		I/O 20	I/O 90	I/O 12	I/O 28
PWM5		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

RamClear

RamClear

Clear Cubloc BASIC's RAM. BASIC's data memory can hold garbage values at power on. `RamClear` can be used to initialize all data memory to zero.

*There are Cubloc modules that support battery backup of the RAM. If you don't use the `RamClear` command in these modules, Cubloc will retain the values in RAM between power cycles.

Reset

Reset

Restarts the Cubloc BASIC program from the beginning. It does not clear the data memory, so any variables that have been declared will retain their values. `RamClear` should be used if this behavior is not desirable.

Reverse

Reverse port

port : I/O port number. (0 to 255)

Reverse the specified port output: logic-high to logic-low, or logic-low to logic-high.

```
Output 8     ' Set Port 8 to output.
Low 8        ' Set output to logic low.
Reverse 8    ' Reverse from logic low to logic high.
```

Rnd( )

variable = Rnd(0)

The command `Rnd` returns a random number from 0 through 32767. The number passed to `Rnd` has no effect, but is required.

```
Dim A As Integer
A = Rnd(0)
```

Internally, this function is pseudorandom; it creates a random number based on previous values. When powered off and turned back on again, the same pattern of random values is generated. Thus, this function is not a true random number generator.

Select...Case

```
Select Case variable
          [Case value [,value],…
                    [Statement 1]]
          [Case value [,value],…
                    [Statement 2]]
          [Case Else
                    [Statement 3]]
End Select
```

If `variable` meets the `value` or expression following `Case`, the code beneath `Case` is executed.

```
Select Case A
        Case 1
              B = 0
        Case 2
              B = 2
        Case 3,4,5,6           ' Use Comma(,) for more than 1 value.
              B = 3
        Case Is < 1            ' Use < for logical operations.
              B = 3
        Case Else              ' Use Else for all other cases.
              B = 4
End Select

Select Case K
        Case  Is < 10          ' If less than 10
              R = 0
        Case Is < 40           ' If less than 40
              R = 1
        Case Is < 80
              R = 2
        Case Is < 100
              R = 3
        Case Else
              R = 4
End select
```

Set Debug

Set Debug On|Off

`Set Debug` is set to `On` by default.

You can use this command to control debugging functions in BASIC.

When you don't need any debugging features, you can use this command to turn off all `Debug` commands instead of modifying every instance of of the `Debug` command. When this command is used, no `Debug` command will be compiled; they are simply discarded from the program.

Debug Command How-to

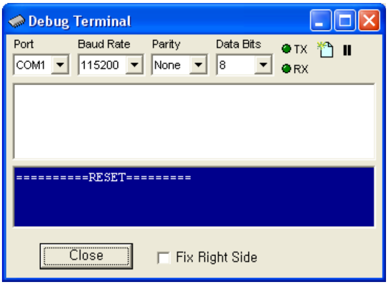
When used correctly, the `Debug` command can help the user identify and fix bugs in a program. During a program's execution, variables can be monitored and verified, an LCD can be simulated, and do other tasks can be performed to increase development productivity.

1. How to Check if program is being reset

Sometimes, do to programming errors, a program can be automatically reset. This condition can be verified using `Debug`.

Simply put a Debug statement at the beginning of your program, such as '`Debug "=========Reset========="` ' as shown below:

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

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



Debug Terminal

Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

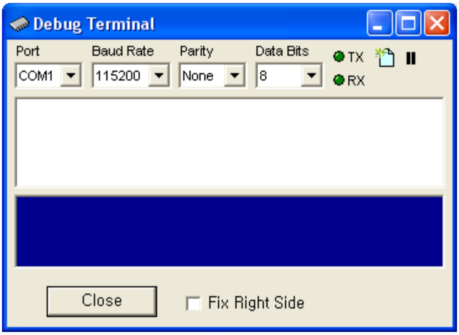
```
==========RESET=========
```

If the program is being reset, the `"=========Reset========="` line will be printed more than once.

2. How to check if a particular point of the program is being executed

Simply insert a `Debug` command at the point in question, as shown below:

```
Const Device = CB280
Do
     High 0
     Delay 200
     Low 0
     Delay 200
Loop
Debug "This Part!"
```

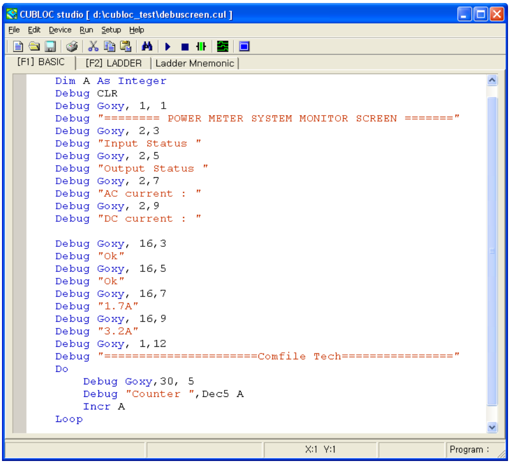


Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

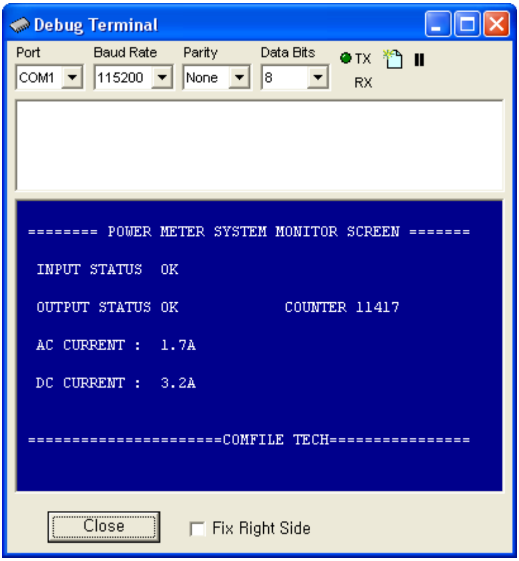
(The debug statement above will never execute, as the program will execute the `Do…Loop` indefinitely)

3. How to simulate an LCD

You can simulate an LCD using the Debug terminal. Simply use `GoXY, XX, YY` to access a particular location on the terminal as shown below:


CUBLOC studio [d:\cubloc_test\debuscreen.cul]
File Edit Device Run Setup Help
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
X:1 Y:1
Program :

```
Dim A As Integer
Debug CLR
Debug Goxy, 1, 1
Debug "======== POWER METER SYSTEM MONITOR SCREEN ======="
Debug Goxy, 2,3
Debug "Input Status "
Debug Goxy, 2,5
Debug "Output Status "
Debug Goxy, 2,7
Debug "AC current : "
Debug Goxy, 2,9
Debug "DC current : "

Debug Goxy, 16,3
Debug "Ok"
Debug Goxy, 16,5
Debug "Ok"
Debug Goxy, 16,7
Debug "1.7A"
Debug Goxy, 16,9
Debug "3.2A"
Debug Goxy, 1,12
Debug "=====================Comfile Tech================"
Do
    Debug Goxy,30, 5
    Debug "Counter ",Dec5 A
    Incr A
Loop
```



Port: COM1 | Baud Rate: 115200 | Parity: None | Data Bits: 8

```
======== POWER METER SYSTEM MONITOR SCREEN =======

 INPUT STATUS  OK

 OUTPUT STATUS OK             COUNTER 11417

 AC CURRENT :  1.7A

 DC CURRENT :  3.2A


=====================COMFILE TECH================
```

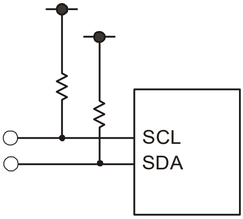
Use the command `Debug Clr` to clear the debug window. At any time during development, you can remove all `Debug` statements from the program's compilation by using the command `Set Debug Off`.

Set I2C

Set I2C dataPort, clockPort

dataPort : SDA, data send/receive port. (0 to 255)
clockPort : SCL, clock send/receive port. (0 to 255)

The `Set I2C` command sets the data port (SDA) and clock port (SCL), for I2C communication. Once this command is executed, both ports' I/O modes become output, and their outputs become logic high. For I2C communication please use ports capable of both input and output and 4.7K resistors as shown below.



I2C communication requires ports capable of both input and output, but some ports are only capable of either input or output. Please check the port specifications in the data sheet for the model you are using.

Set Int

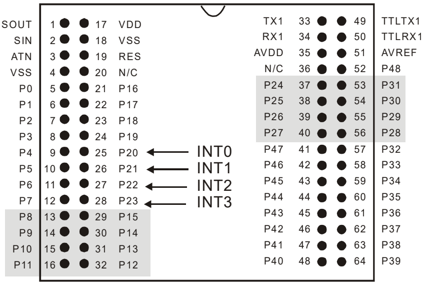
Set Intx mode

x : 0 to 3, External interrupt channel

mode : 0=Falling edge, 1=Rising edge, 2=Changing edge

This command must be used with the `On Int` command in order to receive external interrupts. The mode of interrupt can be set to trigger on either the falling edge, rising edge, or changing edge.

```
Set Int0 0    ' Set external interrupt 0 to trigger on the falling edge.
```



Set Ladder

Set Ladder On|Off

Ladder is set to `Off` by default. Use this command to turn enable Ladder Logic.

The following is an example of a simple BASIC program for starting Ladder logic:

```
Const Device = CB280 'Device Declaration

UsePin 0,In,START    'Port Declaration
UsePin 1,In,RESETKEY
UsePin 2,In,BKEY
UsePin 3,Out,MOTOR

Alias M0=RELAYSTATE  'Aliases
Alias M1=MAINSTATE

Set Ladder On        'Start Ladder

Do
Loop                 'BASIC program will run in infinite loop.
```

Set Modbus

Set Modbus mode, slaveAddress, returnInterval
mode : 0=ASCII, 1=RTU
slaveAddress : Slave address (1 to 254)
returnInterval : Return interval (1 to 255)

Cubloc supports the Modbus protocol in combination with Ladder Logic functions. Modbus can be used on RS-232 channel 1 only.

To enable Modbus slave mode, use the `Set Modbus` command. This command will enable the Modbus slave. It must come after an `OpenCom` command to set up RS-232 communication on RS-232 channel 1. The baud rate, stop bit, and parity settings can be set with `OpenCom`.

```
OpenCom 1,115200,3,80,80    ' Please set receive buffer
                            ' to at least 50.
Set Modbus 0,1,100          ' ASCII mode, Slave address = 1
```

After this command, Cubloc responds automatically. Cubloc supports Modbus commands 1, 2, 3, 4, 5, 6, 15, and 16.

Command	Command Name
01, 02	Bit Read
03, 04	Word Write
05	1 Bit Write
06	1 Word Write
15	Multiple Bit Write
16	Multiple Word Write

Please refer to Chapter 9 for detailed description of Modbus with ASCII and RTU examples.

The return interval is Cubloc's delay time for responding to the master Modbus device. If the return interval is set to be too fast, the master device might not be able to receive all data. The default setting is 1, which is about 200 micro-seconds. A value of 100 is about 4.5ms and a value of 255 is about 11ms.

Set OnGlobal

Set OnGlobal On|OFF

`OnGlobal` is `On` by default.

This command turns on or off the ability to process ALL interrupts.

When `OnGlobal` is set to `Off` and then set to `On`, all interrupt settings in effect before `OnGlobal` was set to `Off` will be still be in effect.

```
Set OnGlobal Off      ' Turn ALL interrupts OFF.
```

If you don’t use any interrupts, you can turn off all interrupts using `Set OnGlobal Off` to increase Cubloc's execution speed.

Set OnInt

Set OnIntx On|Off

x : 0 to 3, External interrupt channel

At power on, `Set OnIntx` is `On` by default.

This command turns on or off the ability to receive individual external interrupts. `x` corresponds to the interrupt number supported by the device. For example `OnInt1` is used for interrupt 1.

When `Set OnIntx` is `On` for a specific interrupt, the interrupt service routine (ISR) set using the `On Intx` command will be executed when the corresponding interrupt occurs. If `Set OnIntx` is `Off`, then the ISR will not be executed when the corresponding external interrupt occurs. See also the `Set Intx` command which controls external interrupts.

```
Set OnInt0 On
Set OnInt1 On
Set OnInt1 Off
Set OnInt2 Off
Set OnInt3 On
```

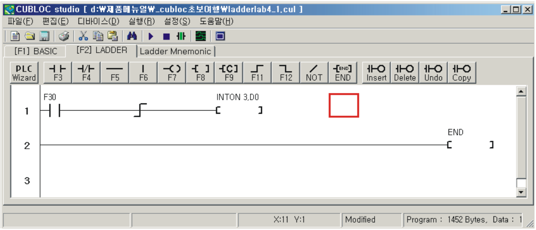
Set OnLadderInt

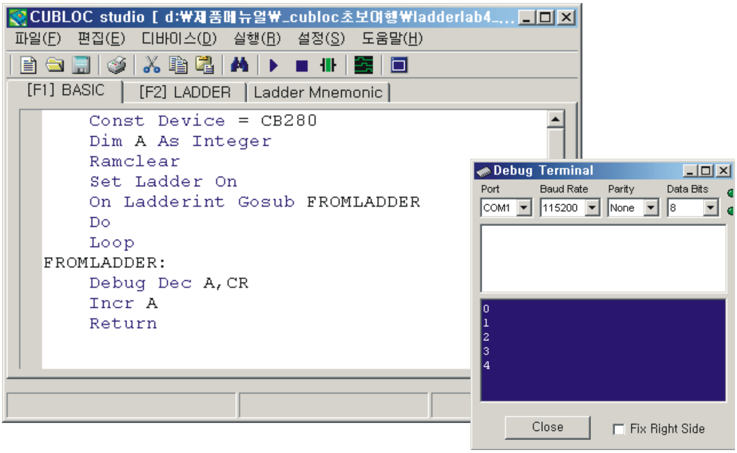
Set OnLadderInt On|Off

At power on, `Set OnLadderint` is `On` by default.

This command turns on or off the ability to receive Ladder Logic interrupts.

When `Set OnLadderInt` is `On`, the interrupt service routine (ISR) set using the `On LadderInt` command will be executed when the corresponding interrupt occurs. If `Set OnGlobal` is `Off`, then the ISR will not be executed when the Ladder Logic interrupt occurs. See also the `On LadderInt` command.


CUBLOC studio [d:₩제품메뉴얼₩_cubloc초보여행₩ladderlab4_1.cul]
파일(F) 편집(E) 디바이스(D) 실행(R) 설정(S) 도움말(H)
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
PLC Wizard
F3
F4
F5
F6
F7
F8
F9
F11
F12
NOT
END
Insert
Delete
Undo
Copy
1
2
3
F30
INTON 3,D0
END
X:11 Y:1
Modified
Program : 1452 Bytes, Data : 1


CUBLOC studio [d:₩제품메뉴얼₩_cubloc초보여행₩ladderlab4_...
파일(F) 편집(E) 디바이스(D) 실행(R) 설정(S) 도움말(H)
[F1] BASIC
[F2] LADDER
Ladder Mnemonic
Debug Terminal
Close
□ Fix Right Side

```
      Const Device = CB280
      Dim A As Integer
      Ramclear
      Set Ladder On
      On Ladderint Gosub FROMLADDER
      Do
      Loop
FROMLADDER:
      Debug Dec A,CR
      Incr A
      Return
```

Port: COM1 Baud Rate: 115200 Parity: None Data Bits: 8

```
0
1
2
3
4
```

Set OnPad

Set OnPad On|Off

At power on, `Set OnPad` is `On` by default.

This command turns on or off the ability to receive `On Pad` interrupts.

When `Set OnPad` is `On`, the interrupt service routine (ISR) set using the `On Pad` command will be executed when the corresponding interrupt occurs. If `Set OnPad` is `Off`, then the ISR will not be executed when the interrupt occurs. See also the `Set Pad` and `On Pad` commands.

Set OnRecv

Set OnRecv0 On|Off
Set OnRecv1 On|Off
Set OnRecv2 On|Off
Set OnRecv2 On|Off

At power on, `Set OnRecv` is `On` by default.

This command turns on or off the ability to receive `On Recv` interrupts. An `On Recv` interrupt occurs after data is received on the serial port AND stored in serial port's the receive buffer.

When `Set OnRecv` is `On`, the interrupt service routine (ISR) set using the `On Recv` command will be executed when the corresponding interrupt occurs. If `Set OnRecv` is `Off`, then the ISR will not be executed when the interrupt occurs. See also the `On Recv` command.

```
Set OnRecv1 On
Set OnRecv1 Off
```

Set OnTimer

Set OnTimer On|Off

At power on, `Set OnTimer` is `On` by default.

This command turns on or off the ability to receive `On Timer` interrupts. An interrupt occurs at every time interval set by the `On Timer` command.

When `Set OnTimer` is `On`, the interrupt service routine (ISR) set using the `On Timer` command will be executed when the corresponding interrupt occurs. If `Set OnTimer` is `Off`, then the ISR will not be executed when the interrupt occurs. See also the `On Timer` command.

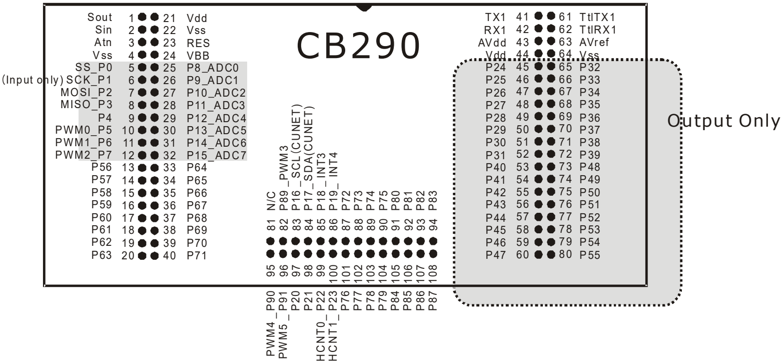
Set OutOnly

Set OutOnly On|Off

The CB290/CT1721 output ports are in a high impendence (High-Z) state at power on in order to prevent the output of data prior to initialization.

`Set OutOnly On` must be used to enable the CB290 / CT1721's output-only ports.

```
Const Device = CB290
Set OutOnly On
Low 24
```



Model	Output only port
CB290	P24 to P55
CT1720 / CT1721	P24 to P55

Set Pad

Set Pad mode, packetSize, bufferSize

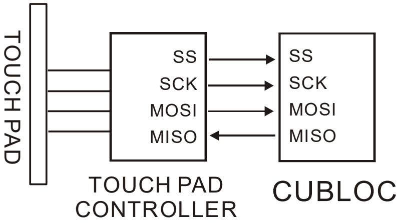
mode : Bit mode (0 to 255)

packetSize : Packet size (1 to 255)

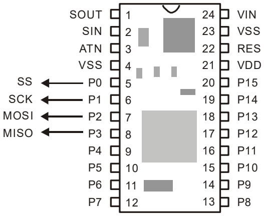
bufferSize : Receive buffer size (1 to 255)

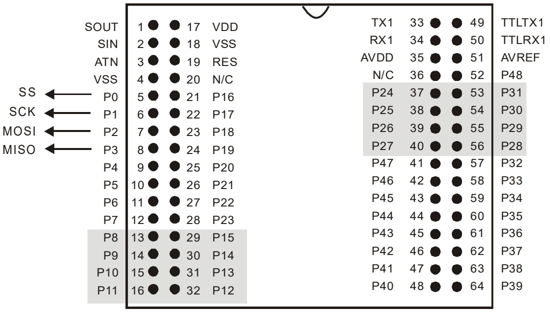
The Cubloc has a dedicated port for keypad / touchpad inputs similar to a PC's keyboard or mouse port. This port can be used with the `Set Pad` command to create interrupts when input is received on a keypad, touchpad, etc.,. This port is as an SPI slave.

To use pad communication, you must use a `Set Pad` command at the beginning of your program. Pad communication uses 4 wires. SCK is used as the clock signal, SS as the slave select, MOSI as the master out/slave in, and MISO as master in/slave out.



I/O ports P0 through P3 can be used for pad communication.

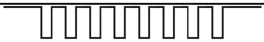
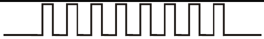
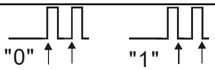




The `packetSize` parameter sets the packet size need to cause an interrupt. For example, the Cutouch panel requires 4 bytes to be received before an interrupt is generated.

The `bufferSize` parameter is the total size of the receive buffer. The buffer size must be at least 1 more than the packet size. A larger buffer will essentially give you more time to process the interrupt service routine. The buffer size is usually set to 5 or 10 times the packet size.

The `mode` parameter will set the receiving mode of the received data. Please refer to the following table:

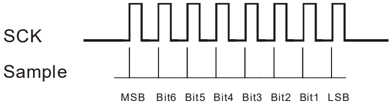
Mode	**Value**	**Bit Pattern**	**Diagram**
LSB First	&H20	0010 xxxx	
MSB First	&H00	0000 xxxx	
SCK Low-Edge Triggered	&H08	xxxx 1xxx	
SCK High-Edge Triggered	&H00	xxxx 0xxx	
Sampling after SCK	&H04	xxxx x1xx	
Sampling before SCK	&H00	xxxx x0xx	

The receiving modes can be added together. For example, for MSB first, High-Edge Triggered SCK, and sampling after SCK:

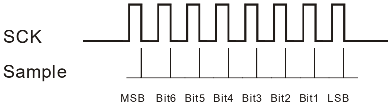
0x00 + 0x00 + 0x04 = 0x04

Here are some common examples:

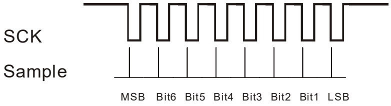
&H00



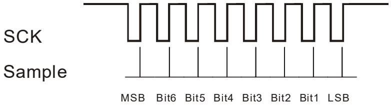
&H04



&H08



&H0C



For pad communications, you can use Comfile's keypads and touch screens.

The `Set Pad` command will automatically set the I/O mode of ports P0 through P3; the user doesn't have to set them.

Set RS232

Set RS232 channel, baudRate, protocol

channel : RS-232 Channel (0 to 3)
baudRate : baudRate (Do not use a variable)
protocol : Protocol (Do not use a variable)

You can only use the `OpenCom` command once to open a serial port. `Set RS232` is used to change the baud rate and/or protocol of a serial port at run time.

For the protocol parameter, please refer to the table below:

Bit 7	Bit 6	Bit 5	Bit4	Bit3	Bit2	Bit1	Bit0
			Parity 0	0 = NONE	Stop Bit 0= 1 Stop Bit 1= 2 Stop Bits	Bit 0	# of Bits 0 = 5 bit
			0	1 = Reserve*		0	1 = 6 bit
			1 1	0 = Even 1 = Odd		1 1	0 = 7 bit 1 = 8 bit

The following table shows typical settings based on the previous table:

Bits	Parity	Stop Bit	Value to Use
8	NONE	1	3
8	EVEN	1	19 (Hex = 13)
8	ODD	1	27 (Hex = 1B)
7	NONE	1	2
7	EVEN	1	18 (Hex = 12)
7	ODD	1	26 (Hex = 1A)

```
OpenCom 1, 19200, 3, 30, 20     'Open Rs232 channel 1
Set RS232 1, 115200, 19         'Change the baud rate & parity
```

Set RS485

Set RS485 channel, portNumber

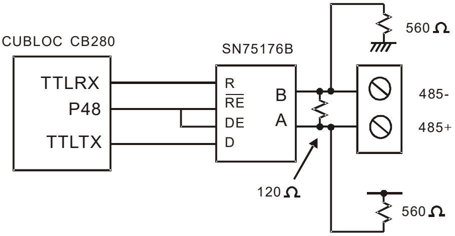
channel : RS-232 channel (0 to 3)

portNumber : Transmit enable port number

RS485 allows you to link multiple Cublocs up to a distance of 1.2km. With RS485, there must be 1 master and the rest must be slave devices. You can use a chip such as the SN75176B or use an RS232 to RS485 converter module.

With RS485, transmitting and receiving data cannot occur simultaneously. RS485 is known for being stable under noisy conditions.

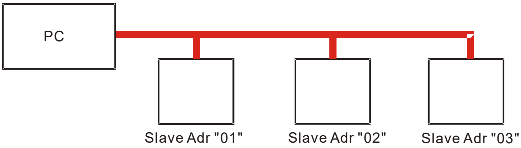
You can refer to the following circuit schematic for connecting TTL signals from a CB280 to the RS485 chip, SN75176B:



RS485 communication needs a "Transmit Enable" signal to control when the device is sending or receiving. There can only be one device transmitting while all the other devices are in receiving mode.

Example:

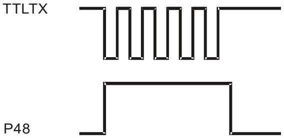
When the PC is transmitting, all the slave devices can only receive data.



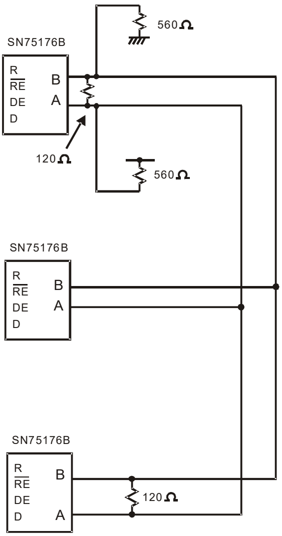
The `Set RS485` command allows the Cubloc or Cutouch to control the data line whenever it wants to send or receive data. While the data is being sent, the transmit enable pin will output active high. This will be done automatically by the Cubloc RTOS.

*NOTE: If you are using an RS232-to-RS485 converter and it supports automatic mode, then you don't need to use this command.

```
Set RS485 1,48         ' Set P48 as the transmit enable pin
```



When using the `Set RS485` command, the port chosen may not be used for other purposes.



1: Please refer to the diagram on the left when connecting multiple Cublocs or Cutouchs using RS485.

Please use a 120 Ohm terminating resistor for the device at the end.

The two 560 Ohm pull-up and pull-down resistors are required for proper communication.

Set Spi

Set Spi clk, mosi, miso, mode

clk : port for clock output

mosi : port for data (Master output, Slave Input)

miso : port for data (Master input, Slave output)

mode: communication mode

bit 3: 0 = MSB start, 1 = LSB start

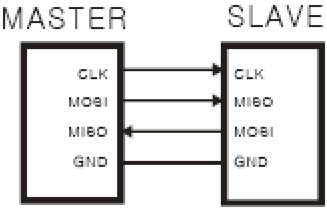
bit 2: 0 = wait at clock LOW, 1 = wait at clock HIGH

bit 1: Output sampe point; 0 = before rising edge, 1 = after rising edge

bit 0: Input sample point; 0 = before rising edge, 1 = after rising edge

Ex) `Set Spi 9,8,7,0`

```
Const Device = CB280
Dim Dtin As Byte
Set Spi 9,8,7,0
Dtin = Spi(Dtout,32)
```



Set Until

Set Until channel, packetLength[, stopChar]

channel : RS232 channel. (0 to 3)
packetLength : Length of packet (0 to 255)
stopChar : Character to catch
charCheck: (Optional) Whether to use stopChar
0 = Check Char (default)
1 = Don't Check Char

This is a conditional statement you can put right after the `On Recv` command. Normally, the `On Recv` interrupt is generated when a single byte is received via the specified serial port. `Set Until` can be used to only generate an interrupt when either `packetLength` bytes are received, or `stopChar` is detected.

`packetLength` is used to interrupt `stopChar` never arrives. To interrupt only on `packetLength`, set `charCheck` to 1.

You MUST use this command with the `On Recv` command.
The following is an example:

```
Dim A(5) As Byte
OpenCom 1,19200,0, 100, 50
On Recv1 DATARECV_RTN
Set Until 1,99,"S"
```

As you can see above, the packet size is 99 bytes. In other words, if character "S" is not received within 99 bytes, an interrupt will occur.

The stop character may also be written in decimal form as shown below:

```
Set Until 1,100,4
```

In the following example, an interrupt be generated only when the packet length is greater than or equal to 5. `charCheck` is 1 so `stopChar` is ignored.

```
Set Until 1,5,0,1
```

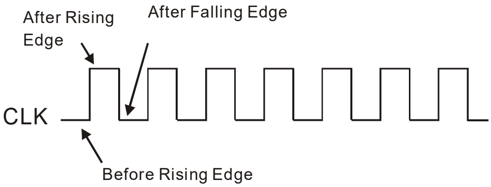
ShiftIn( )

variable = ShiftIn(clock, data, mode, bitLength)

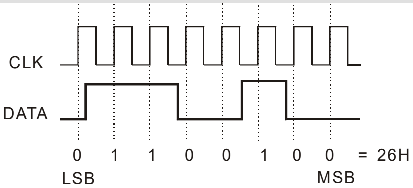
variable : Variable to store results. (No String or Single)
clock : Clock port. (0 to 255)
data : Data port. (0 to 255)
mode : 0 = LSB first (Least Significant Bit first), after rising edge
1 = MSB first (Most Significant Bit first), after rising edge
2 = LSB first (Least Significant Bit first), after falling edge
3 = MSB first (Most Significant Bit first), after falling edge
4 = LSB first (Least Significant Bit first), before rising edge
5 = MSB first (Most Significant Bit first), before fising edge
bitLength : Length of bits (1 to 16)

The `ShiftIn` command receives a shift input. It uses 2 ports, CLOCK and DATA, to communicate.

The `ShiftIn` and `ShiftOut` commands can be used to communicate with SPI, Microwire, and similar communication protocols. When using EEPROM, ADC, or DAC devices that require SPI communication, this command can be used.



```
Dim A AS Byte
A = ShiftIn(3,4,0,8)          ' Port 3 is CLOCK, port 4 is DATA,
                              ' mode is 0, 8 bits to be received.
```



ShiftOut

ShiftOut clock, data, mode, variable, bitLength

clock : Clock port. (0 to 255)

data : Data port. (0 to 255)

mode : 0 = LSB first (Least Significant Bit first)

1 = MSB first (Most Significant Bit first)

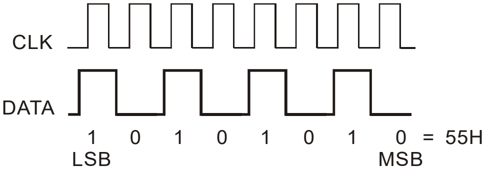
2 = MSB first(Most Significant Bit first) , create ACK (For I2C)

variable : Variable to store data (up to 65535)

bitLength : Bit length (1 to 16)

This command performs a shift output. There are 3 modes. Mode 2 is for the I2C protocol. In I2C communication, an ACK signal is required for every 8 bits.

```
ShiftOut 3,4,0,&H55,8 ' Port 3 = Clock,
                     ' Port 4 = Data, Mode = 0, send 0x55
                     ' 8 bits to be sent,
```



Spi

InData = Spi(OutData, Bits)

InData: Input data
OutData: Input data
Bits: Number of bits (1 to 32)

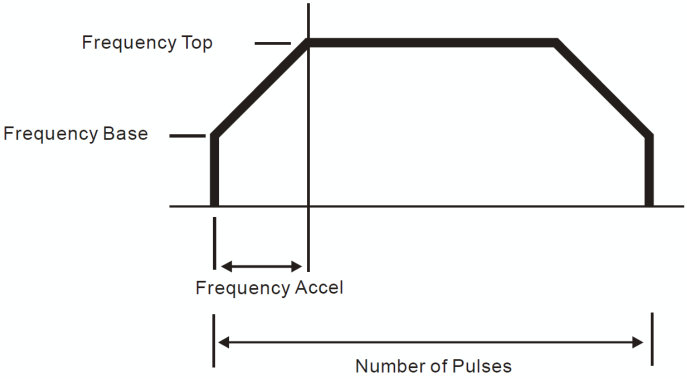
This command sends and receives data simultaneously. In contrast, the `ShiftOut` and `ShiftIn` commands only support either sending data (`ShiftOut`) or receiving data (`ShiftIn`), but not both. The `Spi` command can be used on any I/O port.

The `Set Spi` command must be used prior to the `Spi` command to ensure the I/O ports are defined prior to sending or receiving data.

StepAccel

StepAccel channel, port, freqBase, freqTop, FreqAccel, qty

channel : StepPulse channel (StepAccel supports only 0)
port : Output port
freqBase : The starting stepper frequency (Up to FreqTop)
freqTop : The frequency after acceleration is finished (Up to 3.3KHz)
freqAccel : The acceleration in steps per second
qty : # of pulses to output (up to 2147483647



This command outputs a set number of pulses at a set frequency (up to 3.3kHz) with acceleration. The `StepAccel` command supports only 1 channel, so 0 must be used for the `channel` parameter.

You can use any of the available I/O ports on the Cubloc. When the `StepAccel` command is executed, the specified port's I/O mode is automatically set to output. Even after the command has finished generating pulses, the port's I/O mode remains output.

The output frequency can be set from 1hz to 3.3KHz.

This command will run in the background independently, so system resources can be used for other tasks.

StepPulse

StepPulse channel, port, freq, qty

channel : StepPulse channel(0 or 1)

port : Output port

freq : Output frequency (Up to 15kHz)

qty : # of pulses to output (up to 2147483647)

This command outputs a set number of pulses at a set frequency (up to 15kHz). `FreqOut` and `Pwm` can also output pulses, but the number of pulses cannot be controlled and only the dedicated PWM ports can be used. With `StepPulse`, any output port can be used, and the number of pulses and pulse frequency can be controlled.

Depending on the Cubloc module used, the number of available channels may change. Please refer to the following table for module specific information.

Module	**Channels**	**Channel**	**PWM Channels that cannot be used during use of the command**
CB220, 280, 290, CT17XX	1	0	Channel 0: PWM 3, 4, 5
CB400, CB405	2	0 or 1	Channel 0: PWM 3, 4, 5 Channel 1: 6, 7, 8

`StepPulse` uses the Cubloc processor’s PWM counters. When using this command, PWM3, PWM4, and PWM5 cannot be used.

For the CB400/CB405, when using channel 1, PWM6, PWM7, and PWM8 cannot be used. With the CB2XX series, only channel 0 may be used. With the the CB400/CB405, `StepPulse` can be used on 2 different channels simultaneously.

You can use any of the available I/O ports on the Cubloc. When the `StepPulse` command is executed, the specified port's I/O mode is automatically set to output. Even after the command has finished generating pulses, the port's I/O mode remains output.

The output frequency can be set from 1hz to 15kHz.

This command will run in the background independently, so system resources can be used for other tasks.

StepStat( )

variable = StepStat (Channel)

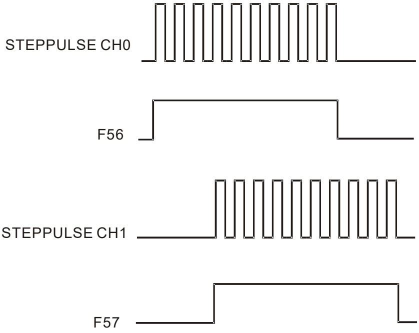
variable : Variable to store results

channel : StepPulse channel(0 or 1)

`StepStat` allows you to monitor how many pulses have been generated since the last `StepPulse` command.

`StepStat` will return double the number of pulses remaining to be generated. If there are 500 pulses left to output, `StepStat` will return 1000.

You can also check the output status of pulses using _F(56) or F56 in Ladder Logic. When channel 0 is generating pulses, _F(56) will be logic high, 1. When channel 1 is generating pulses, _F(57) will be logic high, 1. If no pulses are being generated the F registers will be logic low, 0.



StepStop

StepStop channel

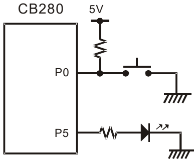
channel : StepPulse channel (0 or 1)

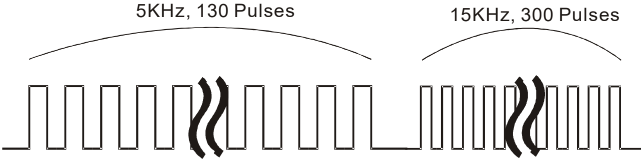
The `StepStop` command immediately stops pulse output on the specified channel.

DEMO PROGRAM

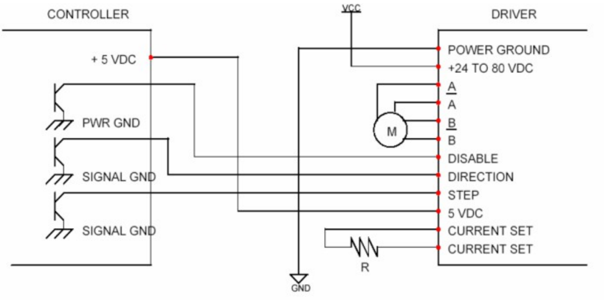
```
Const Device = CB280
Do
    Do While In(0) = 1
    Loop

    StepPulse 0, 5, 5000, 300

    Do While In(0) = 0
    Loop
Loop
```

When the port 0 switch is pressed, port 5 will output 300 pulses at the speed of 5kHz. The following is a circuit diagram for the code above:





A stepper motor controller can be created using a stepper motor and stepper motor driver as shown below.



Connect 3 Cubloc I/O ports to the stepper motor driver. The DISABLE and DIRECTION pins are only to enable and set the direction of the stepper motor.

Please refer to your stepper motor specifications for how many pulses are required to move the stepper motor one rotation.

Sys( )

variable = Sys(address)

variable : Variable to store results. (No String or Single)
address : Address. (0 to 255)

Use the `Sys(0)` and `Sys(1)` commands are used to read the status of the RS-232 buffers for both channel 0 and channel 1.

`Sys(0)` returns the actual number of bytes written to the RS-232 transmit buffer after executing commands `Put` or `PutStr`.

`Sys(1)` returns the actual number of bytes read from the RS-232 receive buffer after executing commands `Get` or `GetStr`.

`Sys(5)` returns the value of the system timer which increments approximately every 10ms. The value can only be read; not changed. The timer will increment up to 65,535 and then reset to 0. This timer can be used in applications that require and extra timer.

`Sys(6)` returns the address of the top of the stack in data memory. At power on, this value is 0. As variables are declared or subroutines and functions are called, this value will increase. This can be used provide an indication of a program's data memory usage at runtime.

All other values are undefined.

TADIn()

variable = TADIn(Channel)

variable : Variable to store results. (No String or Single)
channel : AD channel number (Not Port number, 0 to 15)

This command is similar to `ADIn`, but returns the average of 10 values read with the `ADIn` command. When working under noisy environments, using `TADIn` could help in obtaining more precise results.

`TADIn` is a pre-made function:

```
Function TADIn(num As Byte) As Integer
     Dim ii As Integer, ta As Long
     ta = 0
     For ii = 0 To 9
            ta = ta + ADIn(num)
     Next
     TADIn = ta / 10
End Function
```

Time( )

variable = Time (address)

variable : Variable to store results. (No String or Single)

address : Address of time value (0 to 6)

The CT1721C, CB290 has an internal Real Time Clock (RTC) chip. You can use the `Time` and `TimeSet` commands to write and read time values to and from the RTC. Time information such as the current time of day, day of the week, and year can be written to the RTC, and read from it in real-time.

If a backup battery is used, time is kept current even when the module powers off.

The following is a chart showing the addresses of the RTC and its corresponding values.

* You cannot use these commands for the CB220 nor CB280 since they do not have an RTC.

<table>
<tr><th>Address</th><th>Value</th><th>Range</th><th colspan="6">Bit Structure</th></tr>
<tr><td>0</td><td>Second</td><td>0 to 59</td><td>▨</td><td colspan="3">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>1</td><td>Minute</td><td>0 to 59</td><td>▨</td><td colspan="3">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>2</td><td>Hour</td><td>0 to 23</td><td>▨</td><td>▨</td><td colspan="2">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>3</td><td>Date</td><td>01 to 31</td><td>▨</td><td>▨</td><td colspan="2">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>4</td><td>Weekday</td><td>0 to 6</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td><td>1st digit place</td></tr>
<tr><td>5</td><td>Month</td><td>1 to 12</td><td>▨</td><td>▨</td><td>▨</td><td>2nd digit</td><td colspan="2">1st digit place</td></tr>
<tr><td>6</td><td>Year</td><td>00 to 99</td><td colspan="4">2nd digit place</td><td colspan="2">1st digit place</td></tr>
</table>

Please refer to the chart below for the weekdays' corresponding numerical values:

Sunday	0
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6

System Real Time Clock RTC)

This feature will allow you to use the system timer of a Cubloc as an RTC. You can use `Time` and `TimeSet` commands to access the following addresses:

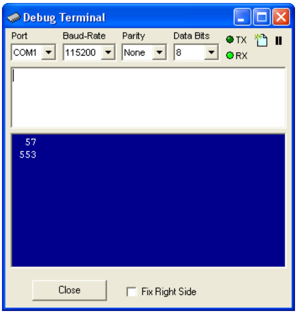
Address	Returning Value	Range
10	Seconds	0 to 59
11	Minutes	0 to 59
12	Hours	0 to 65535
13	Continuous Seconds	0 to 65535

Address 10 will increment its value by 1 every second. When its value becomes 60, address 11 will increment its value by 1. When address 11's value becomes 60, address 12 will increment its value by 1. When address 12's value becomes 65535, it will reset back to 0. At power on, all addresses are set to 0. The `TimeSet` command can be used to set the time at the beginning of user's program.

The system RTC (addresses 10 to 13)'s values are stored as raw binary values, unlike the CB290's and CB405's on-chip RTC . There is no need to convert the values using `BCD2Bin` and `Bin2BCD`.

The system RTC uses the processor's system timer so there can be a slight time difference (< 1%) after a 24 hour period.

```
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
```



Port: COM1
Baud-Rate: 115200
Parity: None
Data Bits: 8

```
 57
553
```

Address 13 will increment its value by 1 every second, just like address 10, except when it reaches 65,535 it will reset to 0. Addresses 10 through 13 must be used with Cubloc Studio version 2.0.X and above.

TimeSet

TimeSet address, value

address : Address of time value (0 to 6)

value : Time value. (0 to 255)

Use the `TimeSet` command to store new time values.

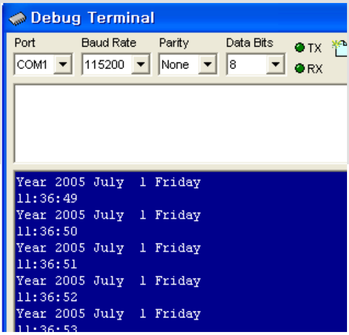
<table>
<tr><th>Address</th><th>Value</th><th>Range</th><th colspan="6">Bit Structure</th></tr>
<tr><td>0</td><td>Second</td><td>0 to 59</td><td>▨</td><td colspan="3">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>1</td><td>Minute</td><td>0 to 59</td><td>▨</td><td colspan="3">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>2</td><td>Hour</td><td>0 to 23</td><td>▨</td><td>▨</td><td colspan="2">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>3</td><td>Date</td><td>01 to 31</td><td>▨</td><td>▨</td><td colspan="2">2nd digit place</td><td colspan="2">1st digit place</td></tr>
<tr><td>4</td><td>Day</td><td>0 to 6</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td><td>▨</td><td>1st digit place</td></tr>
<tr><td>5</td><td>Month</td><td>1 to 12</td><td>▨</td><td>▨</td><td>▨</td><td>10</td><td colspan="2">1st digit place</td></tr>
<tr><td>6</td><td>Year</td><td>00 to 99</td><td colspan="4">2nd digit place</td><td colspan="2">1st digit place</td></tr>
</table>

The following is an example showing how to set the time, and output the current time to the debug window:

```
Const Device=CB290
     Dim i As Byte
     TimeSet 0,0          'Sec
     TimeSet 1,&H32       'Min
     TimeSet 2,&H11       'Hour
     TimeSet 3,&H1        'Date
     TimeSet 4,&H5        'Day of the week
     TimeSet 5,&H6        'Month
     TimeSet 6,&H5        'Year

     Do
            i = Time(6)
            Debug "Year ","200",Hex i, " "
            i = Time(5)
            Select Case i
            Case 0
                  Debug "January"
            Case 1
                  Debug "February"
            Case 2
                  Debug "March"
            Case 3
                  Debug "April"
            Case 4
                  Debug "May"
            Case 5
                  Debug "June"
            Case 6
```

```
                  Debug "July"
            Case 7
                  Debug "August"
            Case 8
                  Debug "September"
            Case 9
                  Debug "November"
            Case 10
                  Debug "December"
            End Select
            i = Time(3)
            Debug " ", Hex2 i         'Print date
            Debug " "

            i = Time(4)
            Select Case i
            Case 0
                  Debug "Sunday "
            Case 1
                  Debug "Monday "
            Case 2
                  Debug "Tuesday "
            Case 3
                  Debug "Wednesday "
            Case 4
                  Debug "Thursday "
            Case 5
                  Debug "Friday "
            Case 6
                  Debug "Saturday "
            End Select
            Debug Cr

            i = Time(2)
            Debug Hex2 i,":"
            i = Time(1)
            Debug Hex2 i,":"
            i = Time(0)
            Debug Hex i,Cr
            Delay 1000
Loop
```

Debug Terminal Screenshot:



Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

```
Year 2005 July  1 Friday
11:36:49
Year 2005 July  1 Friday
11:36:50
Year 2005 July  1 Friday
11:36:51
Year 2005 July  1 Friday
11:36:52
Year 2005 July  1 Friday
11:36:53
```

UDelay

UDelay time

time : Interval (1 to 65535)

`UDelay` is a more precise delay. The delay is initialized to about 70 micro-seconds. Every unit added to it will add 14 to 18 micro-seconds. For example, `UDelay` 0 would be about 70 micro-seconds. `UDelay` 1 would be about 82 to 84 micro-seconds.

```
UDelay 100    ' Delay about 1630 micro-seconds.
```

When an interrupt occurs or Ladder Logic code is being executed at the same time, this delay function might be affected. During this delay, BASIC interrupts are enabled and could cause further delay when using this command.

To prevent interference from Ladder Logic or BASIC interrupts, consider stopping Ladder Logic and all interrupts before using this command.

UsePin

UsePin port, In/Out[, aliasName]

port : I/O port number. (0 to 255)
In/Out : "In" or "Out"
aliasName : Alias for the port (Optional)

This command is used to set a port's I/O mode and alias for Ladder Logic programs. This is required before the ports can be used in Ladder Logic.

```
UsePin 0,In,START
UsePin 1,Out,RELAY
UsePin 2,In,BKEY
UsePin 3,Out,MOTOR
```

Use the `FreePin` command to return reassign the port back to BASIC.

UTMax

UTMax variable

variable : Variable to increment. (No String or Single)

Increment the variable by 1 until the variables maximum is reached. When the maximum is reached, the variable is no longer incremented. The maximum here refers to the variable's type’s maximum value. For `Byte` the maximum would be 255, and for `Integer` the maximum would be 65535.

```
UTMax A       ' Increment A by 1
```

Wait

Wait time

time : interval variable or constant (in milliseconds) 10 to 2,147,483,640

Wait for the specified time in milliseconds.

This command will generate a delay using the system clock. This delay function is accurate to 10ms. It is much more precise than the `Delay` command.

```
Wait 10                  ' Delay 10 ms.
Wait 15                  ' Delay 10 ms.
Wait 110                 ' Delay 110 ms.
Wait 115                 ' Delay 110 ms.
```

WaitTx

WaitTx channel

channel : RS-232 channel. (0 to 3)

This `WaitTx` command will wait until the send buffer is flushed.

Without `WaitTx`, the following is necessary:

```
OpenCom 1,19200,0, 100, 50
PutStr 1,"ILOVEYOU",CR

Do While BFree(1,1)<49  ' Wait until all data has been sent
Loop
```

Using `WaitTx`, the process of sending data is simpler as shown below:

```
OpenCom 1,19200,0, 100, 50
PutStr 1,"ILOVEYOU",CR

WaitTx 1                  ' Wait until all data has been sent
```

When this command is waiting, other interrupts may be called. In other words, this command will not affect other parts of the Cubloc system.

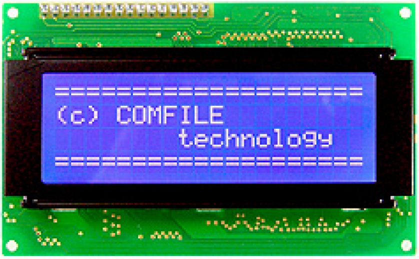
MEMO

Chapter 7: Cubloc Display Library

The Cubloc integrated display functions make it easy to control Comfile LCD products such as the GHLCD or CLCD. Drawing lines, circles, boxes, and printing strings can all be done with a single line of code.

Character LCD : CLCD

The CLCD products are blue or green LCDs that can display characters and numbers. A control board on the back of the device receives data and controls the attached LCD panel.



The CLCD receives data via RS-232 or the CuNet I2C communication protocol.

Set Display

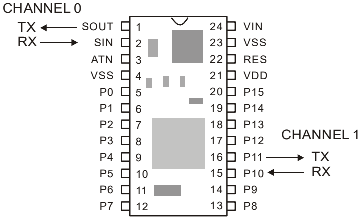
Set Display type, method, baud, bufferSize

type : 0=RS-232LCD, 1= GHB3224, 2=CLCD
method : Communication method 0=CUNET, 1=RS-232 CH1
baud : Slave address when method = 0
baudRate when method = 1
bufferSize : Send buffer size (up to 128)

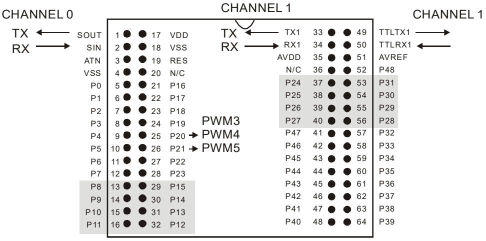
This command is used to initialize the display settings. It can only be used once. All displays will communicate using the method set here.. This command configures the type of LCD, the communication method, the baud rate, and the buffer size. CLCDs will use method 0.

Method = 1 (RS232 Channel 1)

This method only supports the use of RS-232 channel 1 for display. For the CB220, port 11(TX) is used.



For the CB280, pin 33 or pin 49 can be used. Pin 49 outputs a 12V level signal and pin 33 outputs a 5V level signal.



The following baud rates are supported:

2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400.

The recommended buffer size is around 50 to 128. If the send buffer size too small, data will not be displayed correctly. If the send buffer size is too big, it will take up unnecessary memory.

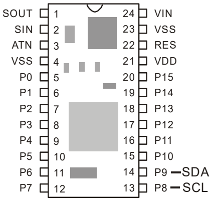
```
Set Display 0,1,19200,50    ' Set Baud rate to 19200 and
                            ' send buffer to 50.
```

The `Set Display` command can only be used once at the beginning of the program.

Method = 0 (Use CUNET)

CUNET is an implemntation of the I2C protocol that is part of the Cubloc.

For the CB220 , use I/O port 8 (Clock) and I/O port 9 (Data).



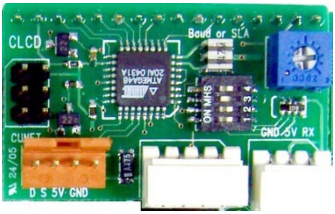
CUNET can be used with displays that support it. CUNET does not use baud rate settings, it uses slave address settings instead.

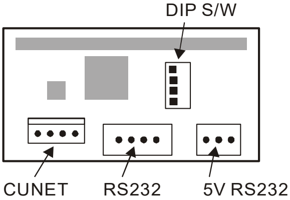
```
Set Display 2,0,1,50  'CLCD, Slave address of 1, Send buffer of 50
```

Although multiple devices can be connected via I2C, for CUNET displays **only ONE device may be attached.**

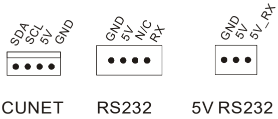
CLCD Module

On the back of the CLCD, a control board is attached. This control board receives CuNET signals and prints to the CLCD.

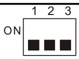
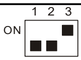
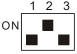
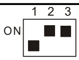
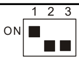
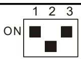
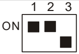
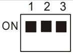




The CLCD can also communicate using RS-232. There are two RS-232 connectors, one for 3-pin 5V level signals and the other for 4-pin +/- 12V level signals.



Use the CLCD DIP switches to set the I2C slave address. The 4th DIP switch is not used.

DIP Switch	RS232 Baud rate	I2C Slave Address
	2400	0
	4800	1
	9600	2
	19200	3
	28800	4
	38400	5
	57600	6
	115200	7

CuNET and RS-232 communication can both be used. If both are connected, please make sure when one of them is processing, the other is not.

The following is the CLCD command table:

Command	Example (hex)	Bytes	Execution Time	Explanation
ESC ’ C’	1B 43	2	15mS	Clear screen. A 15ms delay must be executed after this command.
ESC 'S’	1B 53	2		Cursor On (Default)
ESC 's’	1B 73	2		Cursor Off
ESC 'B’	1B 42	2		Backlight On (Default)
ESC 'b’	1B 62	2		Backlight Off
ESC 'H’	1B 48	2		Locate 0,0
ESC 'L’ X Y	1B 4C xx yy	4	100 uS	Change the position of the cursor.
ESC 'D’ 8byte	1B 44 Code 8bytes	11		Character codes 8 through 15 are 8 custom characters that the user is free to create and use.
1	01	1		Move to beginning of row 1
2	02	1		Move to beginning of row 2
3	03	1		Move to beginning of row 3
4	04	1		Move to beginning of row 4

If the data received is not a command, the CLCD will display it on the screen.

When using RS-232, the maximum baud rate settings for 12V(4-pin) levels is 38400. For TTL 5V levels (3-pin), a baud rate up to 115200 can be used.

The following is an example of code using the CB280 to connect to a CLCD module using the CUNET protocol. When you execute this program, the CLCD will display incrementing numbers.

```
Const Device = CB280
Set Display 2,0,1,50  ' Set the SLAVE ADDRESS to 1 by
                     ' manipulating the DIP switch.
Dim i As Integer
Delay 100            ' Delay for start up of CLCD
Cls
Delay 200            ' Delay for initializing and clearing CLCD
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
```

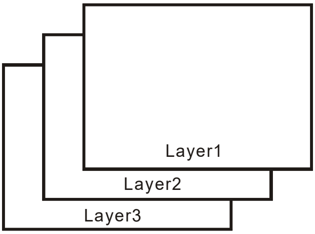
* The slave address specified in the `Set Display` command should match that of the CLCD .

GHLCD Graphic LCD : GHB3224C

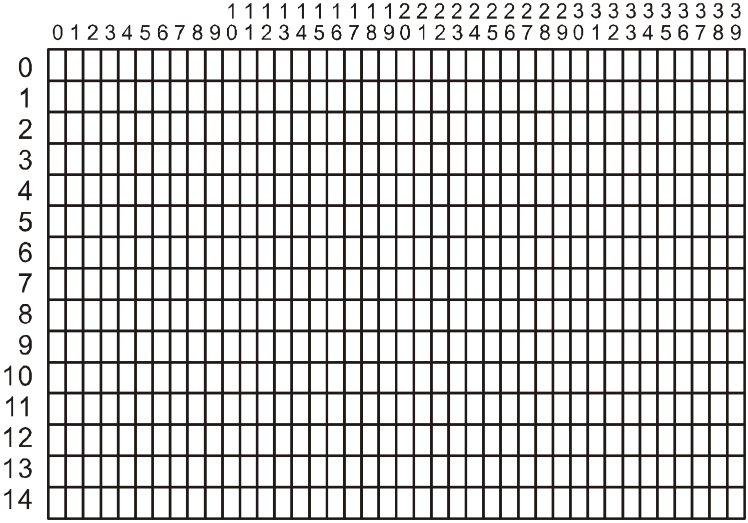
The GHLCD is an LCD that features the ability to display characters and graphics on three different layers. Unlike the CLCD, the GHLCD supports many different commands for easily drawing lines, circles, and boxes. There are also commands to copy, cut, and paste graphics, and a BMP downloader for downloading images to the GHLCD.

The GHB3224C model is a blue and white STN type LCD with a display area of 320 by 240 pixels. There are 3 layers. The first layer is for text and the other 2 layers can be used for graphics.

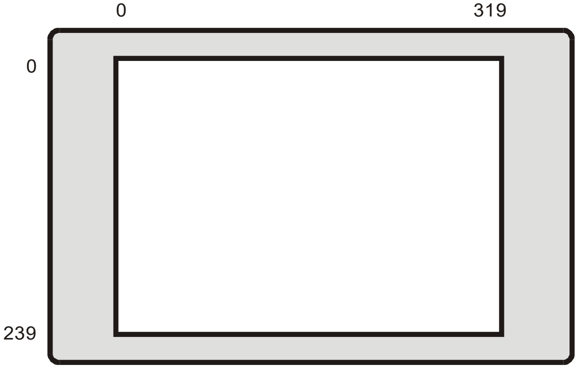
* GHB3224C Library is 99% compatible with Cutouch modules.



The text layer is a 40x15 grid as illustrated below. Each character size is 8 by 16 pixels.



The GHLCD series features a 320 by 240 pixels for graphics.



Please note that graphics or characters will appear randomly when trying to print outside the specified range of pixels shown here.

GHB3224C supports CuNET.

The GHB3224C model supports CuNET. When using Cubloc with the GHCLD, using CuNET instead of serial communications will free up the serial port for other uses.

GHB3224C CuNET settings:

```
Set Display 1,0,1,50   'GHLCD, CUNET, Set Address to 1,
                       'Send buffer to 50.
```

*Warning : The CuNET slave address and display slave address must match. The display lave address can be set with the DIP switch.

Cls

Cls

Initialize the LCD and clear all layers.
(Set a little bit of delay to give the LCD time to initialize.)

```
Cls
Delay 200
```

Clear

Clear layer

Erase the specified layer(s).

```
Clear 1  ' Erase (Text) Layer 1.
Clear 2  ' Erase (Graphic) Layer 2.
Clear 0  ' Erase all layers.  Same as CLS.
```

CsrOn

CsrOn

Turn the cursor on. (Default is OFF).

CsrOff

CsrOff

Turn the cursor off.

Locate

Locate x,y

x : x coordinate of LCD
y : y coordinate of LCD

Set the position of the text cursor. After the CLS command, the LCD defaults to position 0,0.

```
Locate 1,1           ' Move cursor to 1,1
Print "COMFILE"
```

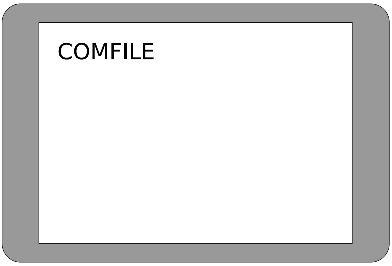
Print

Print text

text : String, String variable, or String constant containing text to be printed

This command prints characters on the text layer. To print characters to the graphic layer, the `GPrint` command can be used.

```
Locate 1,1    ' Move to position 1,1
Print "COMFILE"
```



```
COMFILE
```

To connect multiple `String`s, you can use a comma as shown below:

```
Print "ABC","DEF","GHI"    ' Same as PRINT "ABCDEFGHI".
```

Use `CR` for carriage return (new line).

```
Print "California",CR      ' Print California and go to the next line.
```

CLCDOut

CLCDOut x, y, text

x: x coordinate of LCD

y: y coordinate of LCD

text : String, String variable, or String constant containing text to be printed

The `CLCDOut` command was added to Cubloc Studio in v3.1.2. This command combines the Locate and Print commands into one statement.

```
CLCDOUT 3,2, "COMFILE"      ' Print "Comfile" at position 3,2
```

Using the `Locate` and `Print` command separately can occasionally result in characters being printed to the wrong location. The `CLCDOut` command was created to address this problem.

```
Const Device = CB280
Dim I As Integer
Set Display 2,0,0,80

Do
    CLCDOUT 1,1,"COMFILE",Dec I
    Incr I
    Delay 200
Loop
```

This command can only be used with CLCD modules manufactured after December 2010. Users of existing CLCD modules manufactured before December 2010 should contact Comfile Technology for an upgrade.

This command is not supported by the GHLCD devices.

After executing this command, the Print command's behavior is undefined. After the first `CLCDOut` command is executed, `CLCDOut` should be used exclusively, instead of `Print`, from that point on.

Layer

Layer layer1Mode, layer2Mode, layer3Mode

Layer1Mode : Set layer 1 mode (0=off, 1=on, 2=flash)
Layer2Mode : Set layer 2 mode (0=off, 1=on, 2=flash)
Layer3Mode : Set layer 3 mode (0=off, 1=on, 2=flash)

Set the mode of the specified layer. The `flash` mode will flash the layer at 16Hz. When the LCD is powered on, layer 1 and 2 defaults to `on` and Layer 3 defaults to `off`.

This command can be used to hide the process of drawing lines, circles, etc.,. Set the layer to `off` when drawing, then set the layer `on` when drawing is complete.

GLayer

GLayer layerNumber

LayerNumber : Set the graphic layer. (0,1,2)

There are 3 layers on the GHLCD GHB3224 series. Any of the layers may be used as a graphic layer. Graphic commands such as `Line`, `Circle`, and `Box` can be used to draw on a graphic layer. Normally, layer 1 is used for text while layer 2 is used for graphics. Layers 2 and 3 have slightly different characteristics. We recommend layer 2 for graphics that require a lot of erasing.

Layer 1 can also be used as a graphic layer. In this case, you can even erase text characters with graphic commands. To set Layer 3 to a graphic layer, use the command `Layer 3 On`.

Overlay

Overlay overlayMode

overlayMode : Logical mode (0=OR, 1=AND, 2=XOR)

This command determines the drawing logic mode between layer 1 and layer 2. Usually, layer 1 is for text and layer 2 is for graphics. By using this command, the user can specify the overlay mode when layer 1 and layer 2 are displaying on the same position. The default is `XOR`, which will invert when layer 1 and layer 2 print to the same positions. `OR` will allow graphics on both layers to overlap. `AND` will display graphics only where they overlap.

Contrast

Contrast value

value : Contrast value (1 to 1024)

This command controls the contrast of the LCD. Use this command with care, the contrast setting is sensitive. You will most likely need to adjust the contrast wheel on the back of the LCD after using this command.

```
Contrast 450
Delay 500
```

Please add a 500ms delay after each call to this command to prevent an overrun in the graphics buffer.

Light

Light value

value : Backlight 0=Off, 1=On

This command is used to turn the backlight on and off. The default is on.

Note: This command is obsolete for all devices manufactured after December 2012 due to a change in the backlight hardware.

WMode

WMode value

value : 0=Fast, 1=Slow

This command is used to select the method for refreshing the screen. To draw graphics or text, data must be written to memory. Slow mode only writes to memory between screen refreshes, so the screen redraws at a slower rate, but produces no imperfections in the display. Fast mode writes to memory at any time, so the screen refreshes at a faster rate, but can cause imperfections in the display, such as flickering and noise.

Font

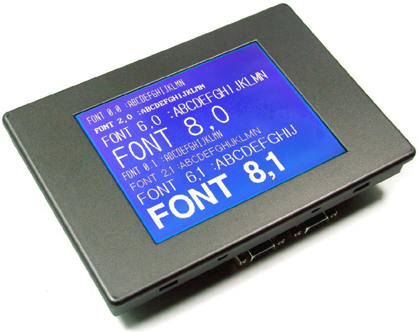
Font fontSize, eFontWidth

fontSize : 0 to 8 Font Selection

eFontWidth : 0 = fixed width, 1=variable width

This command sets the size and with of the font used to print characters. The GHB3224 LCD has 4 different font sizes and 2 different widths.

Font Size	Font
0,1	10 x 16
2,3,4,5	16 x 16
6,7	24 x 24
8	48 x 48



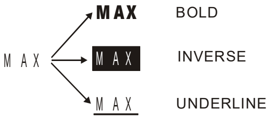
```
    Const Device = CB290
    Cls
    Delay 100
    Font 0,0
    Glocate 10,10
    GPrint "FONT 0,0 :ABCDEFGHIJKLMN"
    Font 2,0
    Glocate 10,30
    GPrint "FONT 2,0 :ABCDEFGHIJKLMN"
    Font 6,0
    Glocate 10,50
    GPrint "FONT 6,0 :ABCDEFGHIJKLMN"
    Font 8,0
    Glocate 10,72
    GPrint "FONT 8,0 "
    Font 0,1
    Glocate 10,120
    GPrint "FONT 0,1 :ABCDEFGHIJKLMN"
    Font 2,1
    Glocate 10,140
    GPrint "FONT 2,1 :ABCDEFGHIJKLMN"
    Font 6,1
    Glocate 10,160
    GPrint "FONT 6,1 :ABCDEFGHIJ"
    Font 8,1
    Glocate 10,185
    GPrint "FONT 8,1 "
```

Style

Style bold, inverse, underline

bold : 0=Normal, 2 or 3 =Bold
inverse : 0=Normal, 1=Inverse
underline : 0=Normal, 1=Underline

This command is used to add bold, inverse, or underline decorations to your fonts.



CMode

CMode value

value : 0=Box type, 1=Underline type

This command sets the type of cursor to use. The default is the underline type.

■ 0 : BOX Type

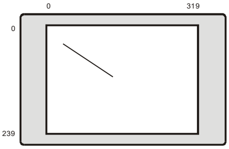
— 1 : Under Line Type

Line

Line x1, y1, x2, y2

This command draws a line from `x1`, `y1` to `x2`, `y2`.

```
Line  10,20,100,120  ' Draw a line
```

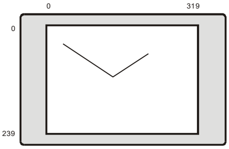


LineTo

LineTo x, y

This command draws a line from the last point to `x, y`.

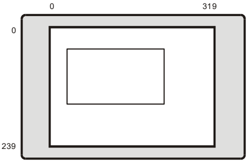
```
LineTo  200,50

'  Continue  drawing  a  line  from  the  last
point
```



Box

Box x1, y1, x2, y2

This command draws a rectangle with the upper-left coordinates of `x1, y1` and the bottom right coordinates of `x2, y2`.

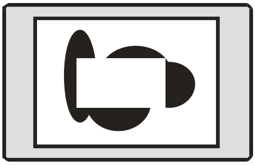


```
Box   10,20,200,100    ' Draw a box
```

BoxClear

BoxClear x1, y1, x2, y2

This command clears a rectangle with upper-left coordinates of `x1, y1` and bottom-right coordinates of `x2, y2`.



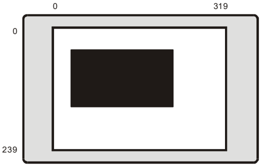
```
BoxClear   10,20,200,100    ' Clear box
```

BoxFill

BoxFill x1, y1, x2, y2,logic

logicOperator : 0=OR, 1=AND, 2=XOR

This command draws a rectangle with upper-left coordinates of `x1, y1` and bottom-right coordinates of `x2, y2` and fill by logically combining with other `BoxFill` areas.



0 - OR will display `BoxFill` areas that overlap and `BoxFill` areas that don't overlap.
1 - AND will display only the `BoxFill` areas that overlap.
2 - XOR will display only the `BoxFill` areas that don't overlap.

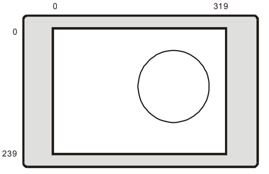
```
BoxFill   10,20,200,100,0    ' Draw and fill
box
```

Circle

Circle x, y, r

This command draws a circle with center coordinates of x, y and radius r.

```
Circle  200,100,50   ' Draw circle
```

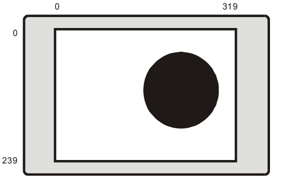


CircleFill

CircleFill x, y, r

Draw and fill a circle with x, y as the center and with r as the radius.

```
CircleFill 200,100,50

 ' Draw and fill circle
```

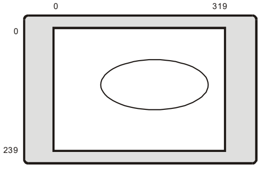


Ellipse

Ellipse x, y, r1, r2

This command draws an ellipse with center `x`, `y`, horizontal radius `r1`, and vertical radius `r2`.

```
Ellipse  200,100,100,50    ' Draw ellipse
```

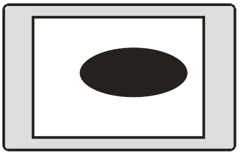


ElFill

ElFill x, y, r1, r2

This command draws and fills an ellipse with center `x`, `y`, horizontal radius `r1`, and vertical radius `r2`.

```
ElFill  200,100,100,50

' Draw and fill ellipse
```

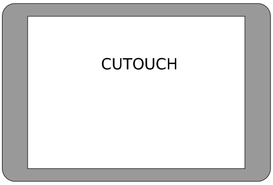


GLocate

GLocate x, y

This command specifies the graphical text position on the current graphic layer.

```
GLocate 128,32      ' locate new position
GPrint "CUTOUCH"
```

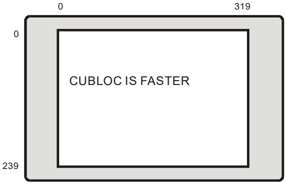


GPrint

GPrint string

This command prints a string on the graphic layer. You have more freedom printing text in the graphic layer, as you can use `GLocate` to specify the exact position. Then you can use the `GPrint` command to print a string at the specified location.

```
GPrint "CUBLOC IS FASTER",CR
  ' Print String and go to next line(CR)
```

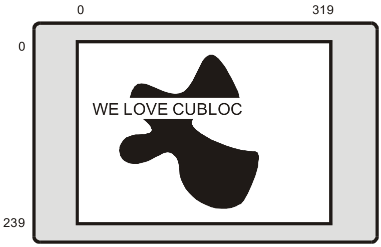


DPrint

DPrint string

`DPrint` is similar to `GPrint`, except it over-writes the current graphics.

```
DPRint "WE LOVE CUBLOC",CR  ' Print String and go to next line
```



This command prints faster than `GPrint` since it simply overwrites the background. When trying to display animations or numbers that change rapidly, such as a moving ball or the current time, `DPrint` will allow smoother transitions.

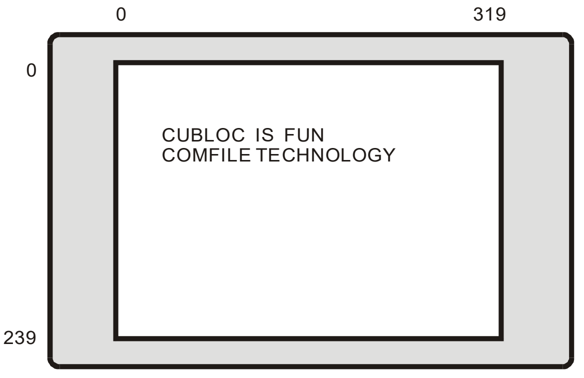
`DPrint` can only be used with X-Axis values that are a multiple of 8. For example, you can use `GLocate` 8,2 or `GLocate` 16,101 but not `Glocate` 10, 30.

Offset

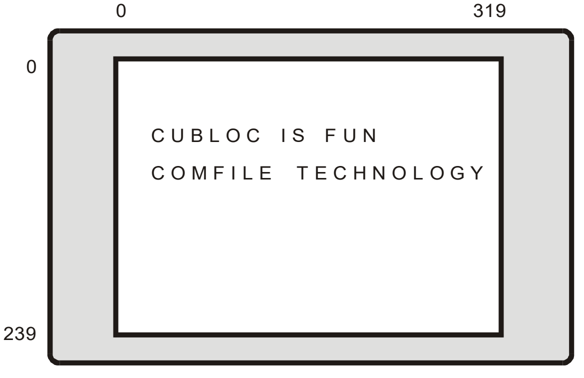
Offset x, y

This command offsets printed characters on the graphic layer. `x` is the spacing between characters, and `y` is the spacing between lines. The default value is 0. Both `x` and `y` can be set independently.

```
Offset 0,0    ' Default offset
```



```
Offset 3,3    ' Set x and y offset to 3.
```



Pset

PSet x, y

This command draws a dot at coordinates `x`, `y`.

```
PSet  200,100    ' Draws a dot
```

Color

Color value

value : 0 = white, 1 = black

This command sets the current drawing color. The default value is 0 (white).

```
Color  0         ' Set color to white.
```

LineStyle

LineStyle value

This command sets the line style used when drawing lines and shapes. You can make dotted lines by increasing `value`. The default value is 0, a solid line.

```
LineStyle 1       ' Use dotted lines
```

DotSize

DotSize value, style

value : the size of the dot

style : the style of the dot: 0 = rectangular, 1 = circular

This command sets the dot size used when drawing lines and shapes. Value sets the size of the dot, and style makes the dot either rectangular (0), or circular (1).

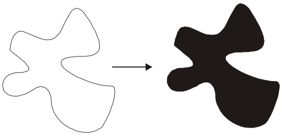
```
DotSize 1,1       ' Set dot size to 1 and dot style to circular.
```

Paint

Paint x, y

This command fills the area enclosing `x, y`.

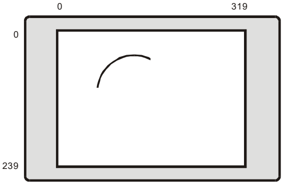
```
Paint 100,100   ' Fill the area enclosing
                ' 100,100
```



Arc

Arc x, y, r, start, end

This command draws an arc. `x, y` specifies the arc's center, r specifies the car's radius, start specifies the angle at which to `start` drawing, and `end` specifies the angle at which to stop drawing.



```
Arc 150, 150, 100, 200, 290    ' Draw an
arc
    '  from 200 to 290 degrees.
```

DefChr

DefChr code, data

code : Custom character code (&hDB30 to &hDBFF)
data : 32byte bitmap data

This command creates custom characters. A character of size 16x16 pixels can be created and stored in the LCD memory. Then the character can be used just like any other regular character using the `Print` or `GPrint` and `DPrint`. A total of 207 custom characters can be stored in the memory. The characters are not preserved when powered off.

```
DefChr &HDB30,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,_
       &HAA,&HAA,&HAA,&H55,&HAA,&HAA,&HAA,&HAA,_
       &HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,_
       &HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA,&HAA

Print Chr(&HDB30)
```

Bmp

Bmp x, y, fileNumber, layer

x, y : Coordinates where bitmap should display
fileNumber : Bitmap File number
layer : Layer on which to display the bitmap.

The GHB3224 has flash memory for storing bitmap files. The BMP Downloader is used to download bitmaps to the LCD. Once the bitmaps files are stored in flash memory, this command can be used to display them on the LCD.

*The GHB3224 has 102,400 bytes of flash memory for storing bitmap files which can store approximately 10 320x240 full screen images.

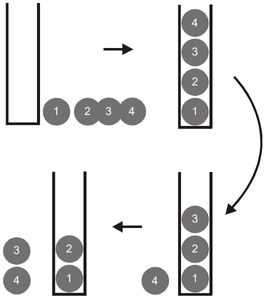
Graphic Data Push and Pop Commands

The GHB3224 has a separate stack for storing graphic data. The current screen can be pushed and popped on and off the stack. By saving and restoring the screen to the stack, copy, cut, and paste features can be implemented.

The `GPush` and `GPop` can be used for precise cutting of the current screen while `HPush` and `HPop` can be used for higher speed operations with less precision.

The stack is LIFO (last in first out), so it will pop the last screen that was pushed onto the stack.

There is approximately 32KB of stack memory on the GHB3224 which can store approximately 3 to 4 full screens. Please refer to the picture below for a depiction of how the stack works:

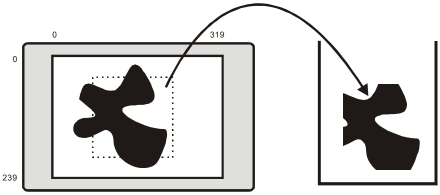


GPush

GPush x1, y1, x2, y2, layer

Push the rectangular area with upper-left coordinates x1, y1 and lower-right coordinates x2, y2 onto the stack.

```
GPush 10,20,200,100,2
```



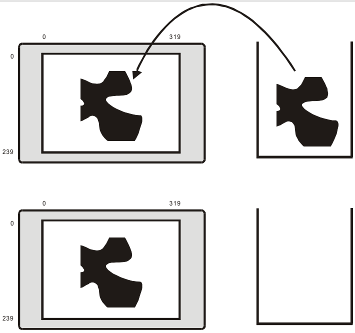
GPop

GPop x, y, layer, logic

logic =0 : OR
logic =1 : AND
logic =2 : XOR
logic =3 : Clear screen then pop

This command pops a screen from stack and displays it on the specified layer at coordinates x,y. The logic parameter specifies how the popped screen should be combined with the data currently displayed on the specified layer.

```
GPop 120,20,2,0
```



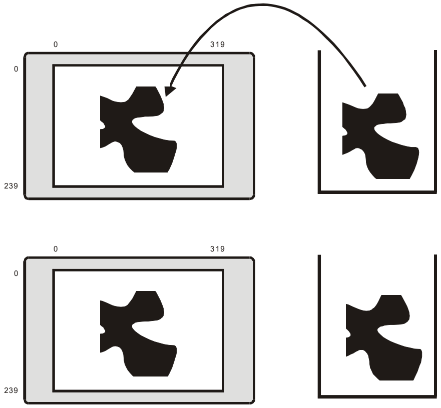
GPaste

GPaste x, y, layer, logic

logic =0 : OR
logic =1 : AND
logic =2 : XOR
logic =3 : Clear screen then pop

This command pastes a screen from the stack and displays it on the specified layer at coordinates x,y. The logic parameter specifies how the popped screen should be combined with the data currently displayed on the specified layer.

This command is identical to `GPop` except it will not remove the screen from stack. Therefore, you can use this command if the current item in the stack must be used again.

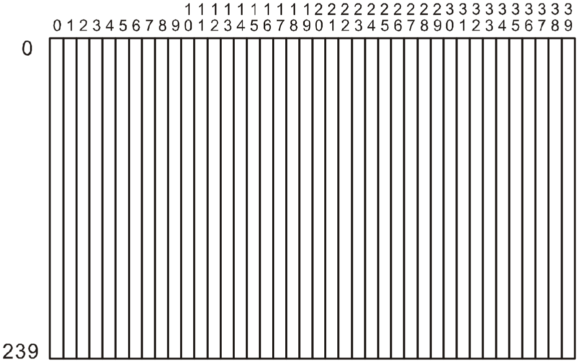


HPush

HPush x1, y1, x2, y2, layer

The `HPush`, `HPop`, and `HPaste` commands are similar to `GPush`, `GPop`, and `GPaste` except the x coordinate in multiples of 8 as shown below:

The 320 pixels is divided into 40 columns, each 8 pixels wide.



```
HPush 6,20,12,100,2
```

HPop

HPop x, y, layer

This command is the same as `GPop`, except the x coordinate must be from 0 to 39.

```
HPop 10,20,2,0
```

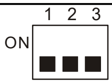
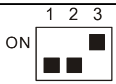
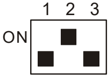
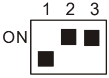
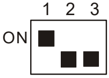
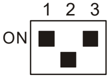
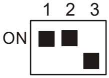
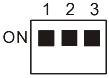
HPaste

HPaste x, y, layer

This command is the same as `GPaste` except the x coordinate must be from 0 and 39.

GHB3224C DIP Switch Settings

On the back of the GHB3224B, there are DIP switches to set the RS232 baud rate and I2C slave address. GHB3224 DIP Switch number 4 is not used.

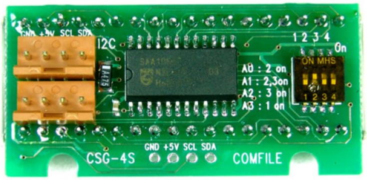
DIP Switch	RS232 Baud Rate	I2C Slave Address
	2400	0
	4800	1
	9600	2
	19200	3
	28800	4
	38400	5
	57600	6
	115200	7

Only one communication method (either CuNET or RS232) can be used at a time.

Seven Segment Display: CSG

A seven segment display can be used to display numbers. Eight LEDs are used for most seven segment displays as shown below, allowing decimal points to be displayed as well.

Using a seven segment display requires specialized circuits to control and refresh the segment matrix. This increases in complexity with each digit added. In the interest of convenience and simplicity, we have developed an easy to use seven segment display called the CSG module.



As you can see above, the front has a 4 digit seven segment display and the back has two I2C connections. After connecting the CSG to a Cubloc, you can use the commands in the following table to easily and quickly display the numbers you want.

Command	Explanation	Example Usage
`CSGDec` SlaveAdr, Data	Output decimal value.	`CSGDec` 0, 1
`CSGHex` SlaveAdr, Data	Output hex as decimal value	`CSGHex` 0, 1
`CSGNPut` SlaveAdr, Digit, Data	Control digit places	`CSGNPut` 0, 0, 8
`CSGXPut` SlaveAdr, Digit, Data	Control digit places and output data as binary number	`CSGXPut` 0, 0, 9

CSGDec

The CSGDec command is used to print decimal values to the display.

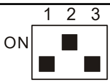
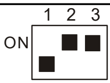
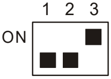
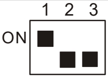
```
Const Device = CB280
Set I2c 9,8         '←-- must be used before the CSGDec command
Dim b As Integer
b=8
Do
        CSGDec 0,b  '←-- CSGDec command
        Delay 100
        b = b + 1
        If b=0 Then b=200
Loop
```

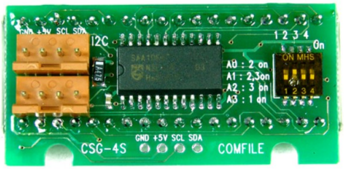
To use CSG commands, the Set I2C command must be used beforehand.

Slave Address

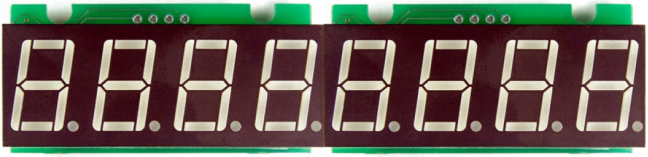
The I2C slave address can be set using the DIP switches on the back of the CSG module. A total of 4 addresses can be set per I2C line pair.

CSG DIP Switch:

DIP Switch	Slave Address
	0
	1
	2
	3



To display more than 4 digits, use 2 CSG modules as shown below and set different slave addresses for each.



CSGNPut

CSGNPut slaveAddress, digit, data

slaveAddress : CSG module I2C slave address
digit : Digit position (0 to 3)
data : Data (&H30 to &H39, &H41 to &H46)

This command displays the desired digit on the specified CSG module. The most significant bit of the data parameter controls the decimal point.

You can use &H30 through 39 and &H41 through &H46 only.

&H30 displays "0"
&H31 displays "1"
:
&H39 displays "9"
&H41 displays "A"
&H42 displays "b"
:
&H46 displays "F"

CSGXPut

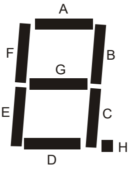
CSGXPut slaveAddress, digit, data

slaveAddress : CSG module I2C slave address

digit : Digit (0 through 3)

data : Bit array with each position corresponding to an LED segment

This command turns on the LED at the specified position. When displaying anything other than numbers, this command can be used to control each LED segment individually.



Bit	7	6	5	4	3	2	1	0
LED	H	G	F	E	D	C	B	A

To print an "L", positions D, E, and F must be turned on, corresponding to but positions 3, 4, and 5. Therefore the bit value would be 0011 1000, which in hexadecimal is &H38.

```
CSGXPut 0, 0, &H38    'Display an 'L'
```

CSGDec

CSGDec slaveAddress, data

slaveAddress : CSG I2C slave address

data : Decimal value

This command prints a decimal value to the CSG.

CSGHex

CSGHex slaveAddress, data

slaveAddress : CSG slave address

data : Hexadecimal value to display

This command prints a hexadecimal value to the CSG.

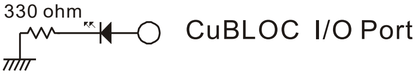
MEMO

Chapter 8: Interfacing

Input/Output Circuits

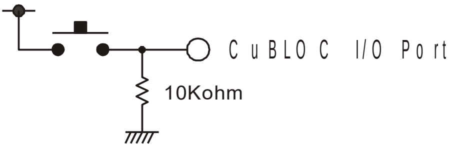
How to connect LEDs

Connect an LED as shown below, and output a logic high to the connected I/O port to turn the LED ON.



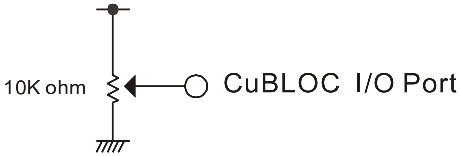
How to connect push buttons

Connect a push button as shown below, and set the connected port's I/O mode to input. When the button is pressed, the Cubloc will read logic high; otherwise it will read logic low.



How to connect a potentiometer

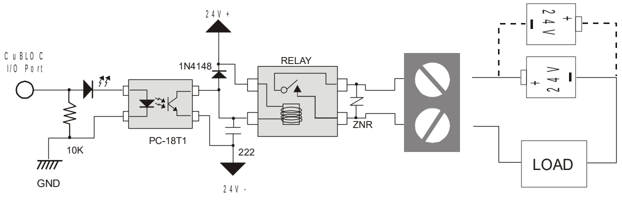
Connect the potentiometer as shown below to an A/D I/O port and use the `ADIn` command to read the position of the potentiometer.



The Cubloc core module uses 5V power. When using a larger voltage, please use an appropriate voltage converter or regulator.

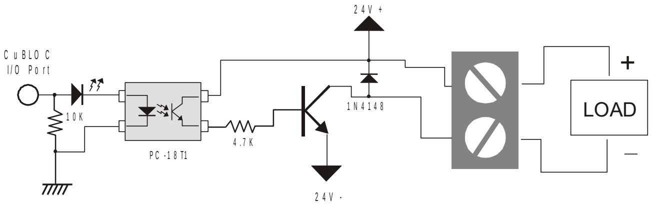
How to Connect an Output Relay.

The following diagram shows how to connect an output relay to a Cubloc I/O port. A photocoupler can be used to separate isolate 24V and 5V circuits and protect against noise. Noise coming from 24V circuit will not affect the 5V circuit and vice-versa.



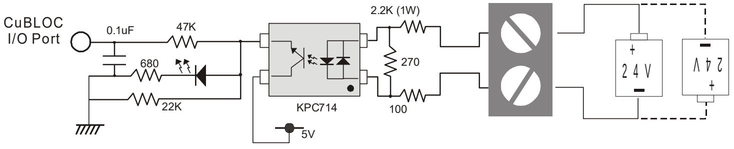
How to Connect an NPN TR Output

This circuit diagram shows an NPN TR photocoupler separating the 5V circuit from the LOAD.



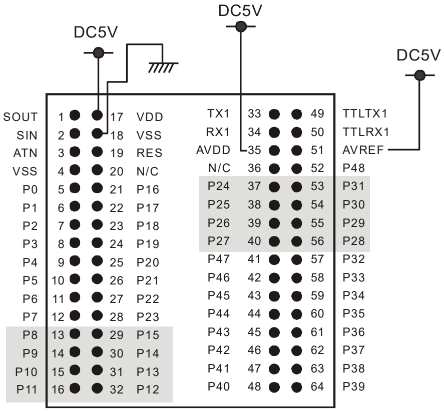
How to Connect a DC24V Input

Use a double polarity photocoupler to convert 24V signals to 5V signals. When input is received, the Cubloc will receive a logic high(5V) signal.



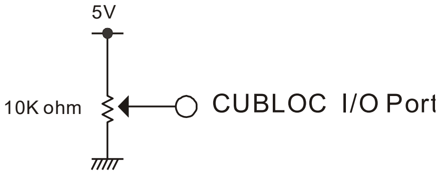
How to connect an AD Input

To connect an AD input to the CB280, the AVDD and AVREF pins must be connected to a 5V source. AVDD supplies power to the ADC of the Cubloc and AVREF is the reference voltage that the ADC uses to do conversions. If a 5V source is connected to the AVREF pin, input voltages from 0 to 5V will be converted and if a 3V source is connected to the AVREF pin, input voltages from 0 to 3V will be converted.

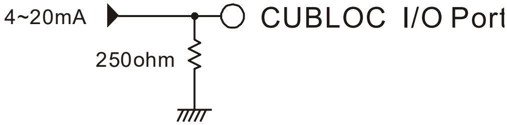


The CB220's AVDD and AVREF are internally connected to 5V.

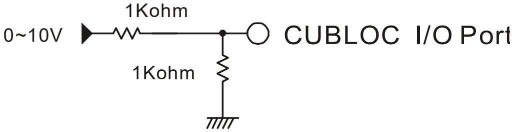
The following is the simplest AD input circuit using a potentiometer. When you turn the knob, the voltage will be converted by the Cubloc ADC to a digital value from 0 to 1023.



The following illustrates a 4-20mA signal connected to the ADC input port. You can use a 230 Ohm and 20 Ohm resistor in series instead of a 250 Ohm resistor.

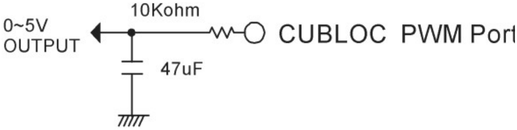


For an input voltage from 0 to 10V, use 2 resistors as shown below. This is called a voltage divider.



How to use a PWM as Digital-to-Analog converter

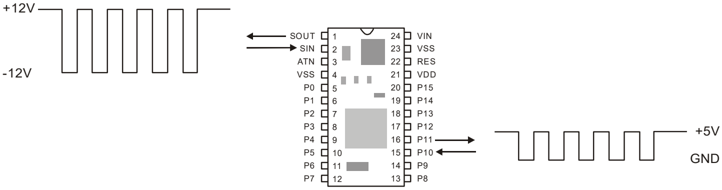
The Cubloc has 6 PWM ports. If you use a simple circuit like that shown below, you can make a digital-to-analog converter.



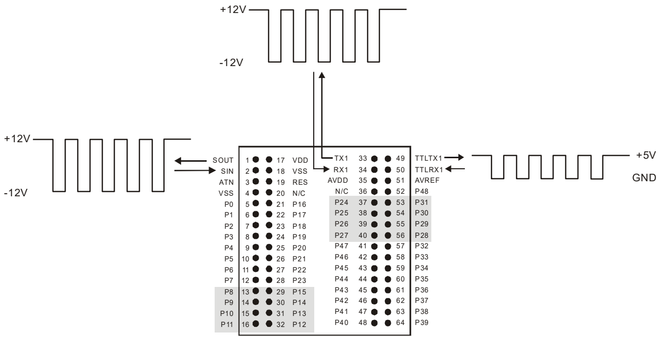
RS-232 HOWTO

A PC's RS-232 interface typically use +/-12V signals, while the Cubloc's RS-232 interface uses 5V signals. To avoid the need to make a separate circuit to convert the +/-12V signals from the PC to the 5V signals of the Cubloc, and vise-versa, the Cubloc was built with both a +/-12V RS-232 interface (channel 0 – the download port) and a 5V RS-232 interface (channel 1).

The +/-12V RS-232 interface (channel 0) is provided on pins 1 and 2. For the CB220/320 a 5V RS-232 interface id provided on pins 10 and 11.



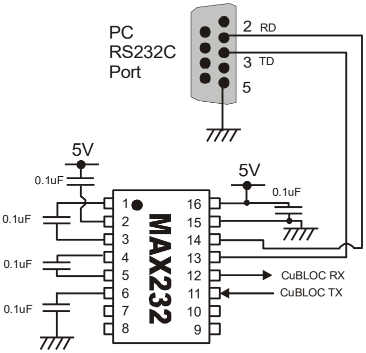
For the CB280/380, RS-232 channel 1 can make use of either 5V or +/-12V signals, but the pins are different. For a 5V interface, use pins 49 and 50. For a +/12V interface use pins 33 and 34.



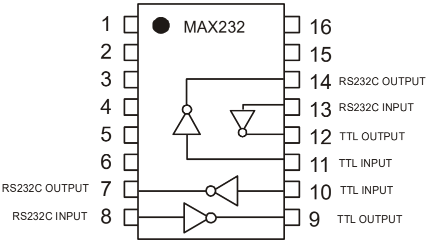
Downloading programs to the Cubloc is very easy, since the PC's RS-232 interface can connect directly to pins 1 and 2 of the Cubloc. For RS-422 and RS-485, 5V signals are provided on RS-232 channel 1.

For the CB280, a +/-12V RS-232 interface is provided in addition to the 5V interface, but only one can be used at a time.

The following shows a simple circuit diagram to convert a 12V RS-232 interface to a 5V RS-232 interface using a MAX232 chip.



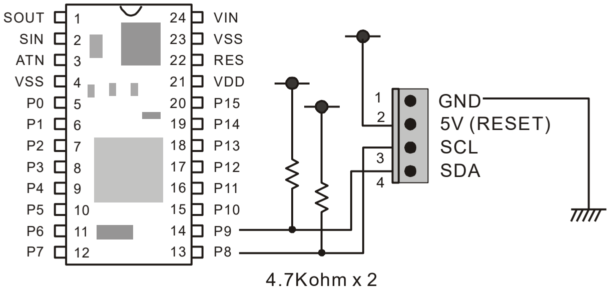
The MAX232 is a very useful chip for converting between 5V and +/-12V RS-232 signals.



CUNET

CUNET is a communication protocol for Cubloc peripherals such as the CLCD, GHLCD, CSG modules. With just 2 pins, SCL and SDA, up to 127 devices can be communicated with simultaneously. CUNET uses Cubloc's I2C protocol to perform the communication.

To use CUNET, please be sure to add pull up resistors(4.7K each) to the SCL and SDA lines. SCL and SDA pins are in an open-collector configuration, protecting against outside noise. It automatically removes pulses less than 50ns.

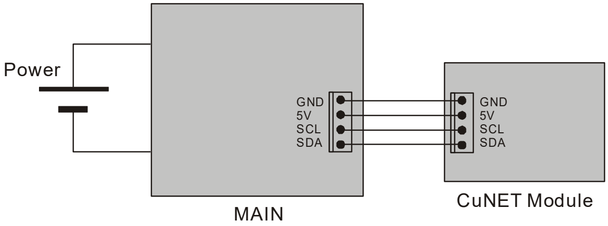


When using CUNET, the connector's pin 1 must be connected to ground, pin 2 to 5V or RESET, pin 3 to SCL, and pin 4 to SDA. This 4 pin configuration is standard for all CUNET interfaces.

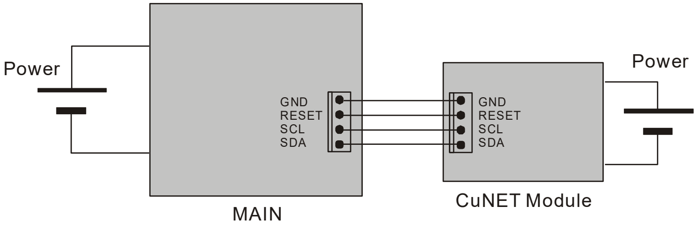
When using CUNET, the Cubloc core module will act as the master and the connected device will act as as the slave. All CUNET devices will respond to the Cubloc while in an idle state.

CUNET is a master-slave protocol. Slaves cannot initiate communication with the master. For externally-initiated communication, you must use PAD communication. PAD can receive inputs from other external devices without the need for the Cubloc to initiate communication. Please refer to the `On Pad` command for more detailed information.

When the device is to be powered from the CUNET bus, pin 2 of the CUNET device should be connected to 5V on the main module,:



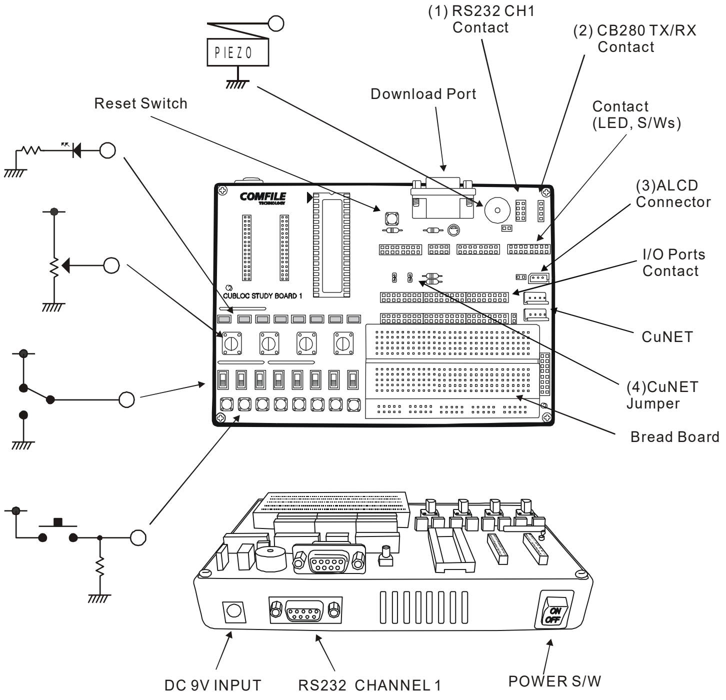
Pin 2 of a CUNET device can be connected to RESET on the main module when a separate power supply is powering the CUNET device. (Active-low to RESET causes the Cubloc to reset)



CUNET cables can be up to 3 feet in length. For longer distances (up to approximately 1 mile), he Phillips I2C long distance interface chip (P82B96 or P82B715) can be used.

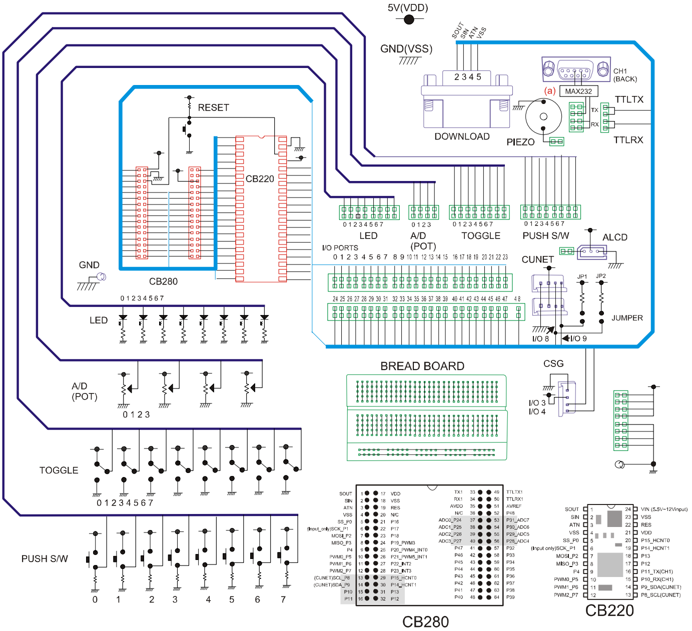
Cubloc Study Board Circuit Diagram

The Cubloc Study Board is useful for first timers and developers of the Cubloc. Simple experiments can be created using switches, LED, RS-232 communication, I2C, piezo, ADC, toggle switches, and LCDs.



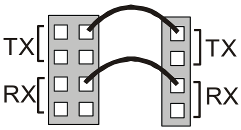
When 9V is supplied, the 5V voltage regulator inside the Study Board will provide 5V to the main module and all peripherals. DC adapter polarity can be used either way. For normal operation, please use a 9V adapter with at least 200mA of current.

Cubloc Study Board Schematic



(1) To use RS-232 channel 1, please connect wires to the appropriate pins on the interface labeled RS-232C in the upper right hand corner.

(2) For the CB280, connect RS-232 channel 1 as shown below:



(3) When using CuNET, all jumpers must be shorted. If using pin 8 and 9 for general I/O, all jumpers should be left open.

About I²C...

The Cubloc provides an easy set of commands to communicate using the I²C protocol. I²C communication is a widely used protocol, primarily for communicating with ADC, EEPROM, DAC, and external I/O chips.

I²C uses two lines, SDA and SCL, and devices must operate in either master or slave mode. The Cubloc can only be used as a master.

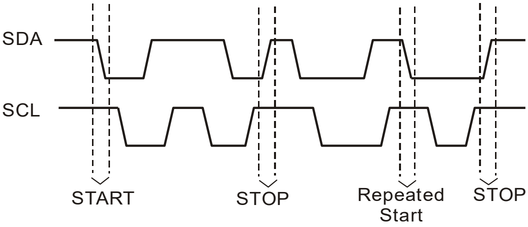
Be sure to configure I2C using the `Set I2C` command before using any other I²C commands.

I²C's START, STOP

When SCL(Clock) and SDA(Data) are logic high, I²C is in an idle state. If a START command is executed during the idle state, I²C begins.

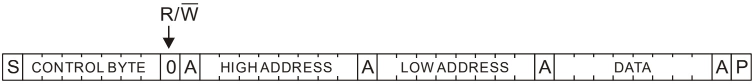
When SCL and SDA are both logic low, I²C is in a busy state. If a STOP command is executed during the busy state, I²C stops.

There is also a Repeated Start in I²C. If a START command is executed during a busy state, I²C restarts.



Using an EEPROM through I²C

We will go through an example showing I²C communication between a Cubloc and a 24LC32 EEPROM. The following is a diagram from the EEPROM's data sheet. It shows how to send data to the EEPROM.



S : Start
A : Acknowledge
P : Stop

The first bit is for the Start command. The 4 upper bits of CONTROL BYTE must be 1010 and the 3 lower bits are for selecting the chip's address. The user may change the EEPROM chip's address by configuring the chip.

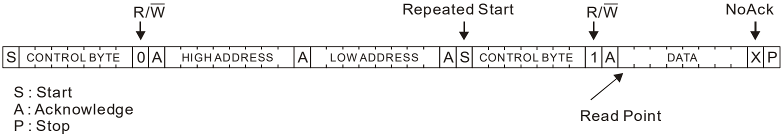
For a read, 1 can be written into R/W and for a write, 0 can be written into R/W. A is for acknowledgment of the 8 bits (1 byte) sent. Then HIGH ADDRESS, LOW ADDRESS and DATA can be sent. When all data is sent, the Stop command can be transmitted.

It takes about 5ms for each EEPROM write.

The following is an EEPROM write sequence in Cubloc's BASIC code:

```
Set I2C 9,8    ' Set P9 as SDA, P8 as SCL
I2CStart
If I2CWrite(&b10100000) = 1 Then ERR_PROC    ' Chip Address = A0
If I2CWrite(adr.Byte1) = 1 Then ERR_PROC     ' Write address to write to
If I2CWrite(adr.LowByte) = 1 Then ERR_PROC
If I2CWrite(data) = 1 Then ERR_PROC       ' Write byte
I2CStop
Delay 5          ' Wait for the write operation to finish
```

Next, we will look at how to read 1 byte from the EEPROM. Although it might look more complex than writing 1 byte, we will soon find out that they are very similar.



Read Point is where the actual data will be read from the EERPOM. The first part of the command is for setting the address to read data from.

```
Set I2C 9,8
I2CStart
If I2CWrite(&b10100000) = 1 Then ERR_PROC   ' Chip Address = A0
If I2CWrite(adr.Byte1) = 1 Then ERR_PROC    ' Write address to read from
If I2CWrite(adr.LowByte) = 1 Then ERR_PROC
I2CStart                                    ' Repeat start
If I2CWrite(&b10100001) = 1 Then ERR_PROC   ' Read command.
Data = I2CRead(0)                           ' Store result to data
I2CStop
```

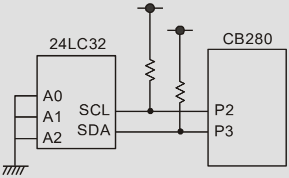
And now, we will look at how to read multiple bytes from the EEPROM. If we don't send a STOP command, we can keep reading from the EEPROM since it automatically increments its address. In this way, we can set the starting address only once, and then read the rest of the data much faster.

```
Set I2C 8,9
I2CStart
If I2CWrite(&b10100000) = 1 Then ERR_PROC    ' Chip address = A0
If I2CWrite(adr.Byte1) = 1 Then ERR_PROC     ' Write address to read from
If I2CWrite(adr.LowByte) = 1 Then ERR_PROC
I2CStart                                     ' Repeat start
If I2CWrite(&b10100001) = 1 Then ERR_PROC    ' Read command.
For i = 0 To 10
   adata(i) = I2CRead(0)    ' Read 10 bytes continuously,
                            ' adata is an array
Next
I2CStop
```

I²C example

The following example shows a CB280 and 24LC32 EEPROM connected. A value will be written to a specified address of the EEPROM and then read back to display on the DEBUG window of Cubloc Studio.

```
     Const Device = cb280
     Dim adr As Integer
     Dim data As Byte
     Dim a As Byte
     data = &ha1
     adr = &h3
     Set I2C 3,2
     Do
            ' Write 1 Byte
            I2CStart
            If I2CWrite(&b10100000)= 1 Then Goto err_proc
            a=I2CWrite(adr.byte1)
            a=I2CWrite(adr.lowbyte)
            a=I2CWrite(data)
            I2CStop
            Delay 1000

            ' Read 1 Byte
            I2CStart
            a=I2CWrite(&b10100000)
            a=I2CWrite(adr.Byte1)
            a=I2CWrite(adr.LowByte)
            I2CStart
            a=I2Cwrite(&b10100001)
            a=I2CRead(0)
            I2CStop

            ' Print Results
            Debug Hex a,cr
            Delay 500
     Loop

err_proc:
     Debug "Error !"
     Do
     Loop
```



More About I²C… (Advanced)

I²C is a common protocol used by many devices today. Cubloc uses I²C as one of its primary communication protocols.

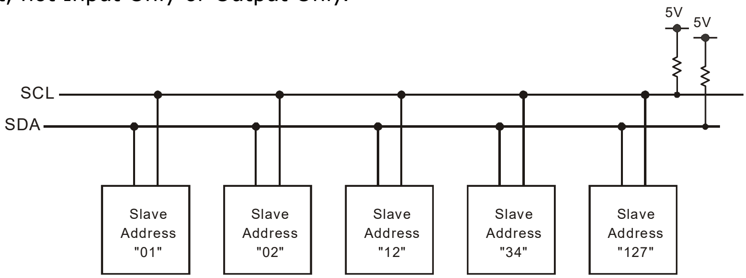
CuNET is built upon the I²C protocol. The main advantage of CuNET is that it's hardware controlled for LCDs. (Not CSG modules or I/O ports)

I²C commands such as `I2CWrite` and `I2CRead` are software commands. An advantage of I²C is that it does not require receive interrupts like serial communications; the clock line is controlled by the master device. This allows the Cubloc to multi-task, not creating any situations where the processor can "freeze" indefinitely while attempting to communicate. The Cubloc can simply request data when it wants to; it does not have to wait for the I²C slave device to respond.

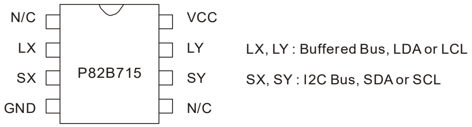
As a result, a Cubloc CB280 module can interface with up to 24 separate I²C buses! (That's buses, you can add multiple I²C devices per I²C bus!)

The Cubloc simulates a master I²C device. Therefore, all other connected I²C devices must operate as slaves.

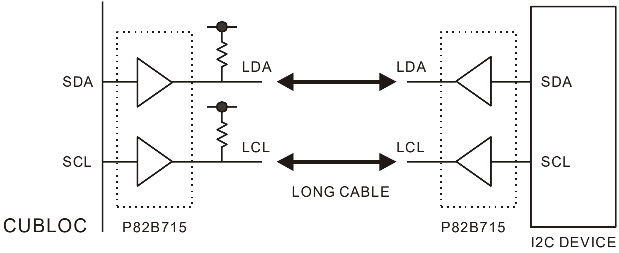
*Note: The I/O port used for I²C communication must be an Input/Output port, not Input Only or Output Only.



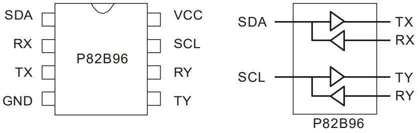
Even though the maximum range for a typical I²C bus is around 12 feet, a long distance extender chip such as the P82B715 can be used to extend the bus to almost ¾ of a mile. A P82B96 can also be used as a buffer to protect the I²C devices from electrical surges and interference.

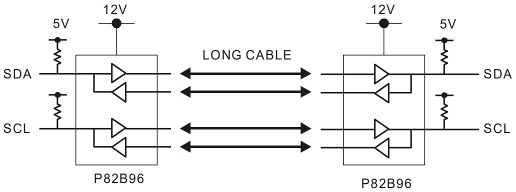

N/C
LX
SX
GND
P82B715
VCC
LY
SY
N/C
LX, LY : Buffered Bus, LDA or LCL
SX, SY : I2C Bus, SDA or SCL

Extend to almost ¾ of a mile using the P82B715.



By using the P82B96, ground and power can be isolated on each end.

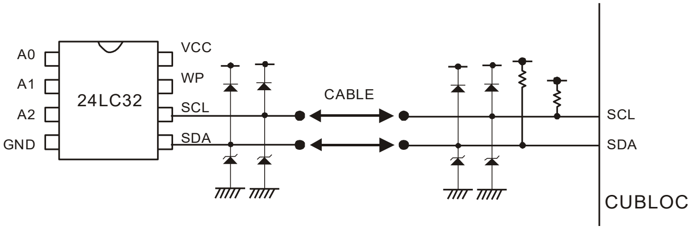




Please refer to Phillips website for more information on the specific chips discussed here: http://www.standardics.nxp.com/.

If you are using I²C **within 12 feet**, we recommend using the following protection circuit:

If the I²C devices are connected with no buffers, electrical interference can cause damage either the Cubloc or the I²C slave device. By using diodes as shown below, you can protect against most of the electrical interference. If the devices are in a heavy, industrial environment, we recommend using P82B96 chips as buffers.



Chapter 9: MODBUS

About Modbus...

Modbus is a protocol developed by Modicon as an interface to their PLCs.

It is usually used with devices like touchscreens, HMI devices, and SCADA software; most of which now support Modbus.

In Modbus, there are master and slave devices. The master provides command while the slave receives and responds to commands. The slave can only send data to the master when requested; it cannot initiate communication on its own.

Each slave has a unique address called a slave address. The master, using those slave addresses, can talk to one slave at a time.

For 1 to 1 connections, RS-232 can be used. For 1 to N connections, RS-485 can be used.

The master sends messages in units of "frames". Each frame contains the slave address, command, data, and checksum codes. The slave receives a trame, analyzes it, performs the requested function, and responds. When responding to the master, slaves also respond in frames.

There are two Modbus transmission modes, ASCII and RTU. RTU is binary and can be implemented in fewer bytes than ASCII, making it a little more compact. ASCII uses a Longitudinal Redundancy Check (LRC) for error checking while RTU uses a Cyclic Redundancy Check (CRC).

The table below shows a Modbus frame in both ASCII and RTU:

Field	Hex	ASCII	RTU
Header		: (colon)	None
Slave Address	0X03	0 3	0X03
Command	0X01	0 1	0X01
Start Address HI	0X00	0 0	0X00
Start Address LO	0X13	1 3	0X13
Length HI	0X00	0 0	0X00
Length LO	0X25	2 5	0X25
Error Check		LRC (2 Bytes)	CRC(2 Bytes)
Ending Code		CR LF	None
Total Bytes		17 Bytes	8 Bytes

ASCII uses a colon (:) to mark the start of a start of a frame and ends the frame with a Carriage Return (CR) or a Line Feed(LF).

START	SLAVE ADR	FUNCTION	DATA	LRC	END
: (COLON)	2 Bytes	2 Bytes	n Bytes	2 Bytes	CR,LF

RTU requires no special characters to mark the start and end of a frame. It uses 4 bytes of silence (a delay) to indicate the start and finish.

START	SLAVE ADR	FUNCTION	DATA	CRC	END
T1-T2-T3-T4	1 Byte	1 Byte	N Bytes	2 Byte	T1-T2-T3-T4

Cubloc support

NOTE: *Starting with Cubloc Studio v3.3.1, the CB400, CB405 and CB405RT no longer support Modbus ASCII. However, Modbus RTU is still supported, so please use Modbus RTU instead.*

Cubloc supports Modbus functions 1,2,3,4,5,6,15, and 16.

Function Code	Function Name
01, 02	Bit Read
03, 04	Word Write
05	1 Bit Write
06	1 Word Write
15	Multiple Bit Write
16	Multiple Word Write

In Modbus, there are addresses that correspond to Cubloc registers. Cubloc's registers P, M, F, C, T, and D can be accessed using the addresses in the following table:

Bit Units		Word Units	
Address	Register	Address	Register
0000H	P		
1000H	M		
2000H	Not Used		
3000H	Not Used		
4000H	F		
		5000H	T
		6000H	C
		7000H	D
		8000H	WP
		9000H	WM
		0A000H	WF

Function Code 01: Read Coil Status
Function Code 02: Read Input Status

These functions are used to read the bit status of a PLC’s Register. The following is an example that reads registers P20 through P56 from slave address of 3.

Query:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X01	1	0 1	2
Start Address HI	0X00	1	0 0	2
Start Address LO	0X14	1	1 4	2
Length HI	0X00	1	0 0	2
Length LO	0X25	1	2 5	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

For Modbus ASCII, the LRC is the 2’s complement of the 8-bit sum of all packet values except the colon, CR, and LF.

For the table above, 0x03 + 0x01 + 0x13 + 0x25 = 0x3C.

To find the 2’s complement of 0x3C, we can write it in binary first.

0011 1100

Then invert the bits.

1100 0011

Then add one:

1100 0011 + 0000 0001 = 1100 0100 = 0xC4

The LRC is therefore 0xC4.

The table below illustrates this frame in Modbus ASCII.

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	13	10

The response frame to the query above would be:

Response:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X01	1	0 1	2
Byte Count	0X05	1	0 5	2
Data 1	0X53	1	5 3	2
Data 2	0X6B	1	6 B	2
Data 3	0X01	1	0 1	2
Data 4	0XF4	1	F 4	2
Data 5	0X1B	1	1 B	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

If you look at the response to the query, you can see that bits 20 through 27 make up one byte.

Data 1's least significant bit corresponds to P20 and Data 1's most significant bit corresponds to P27. Likewise, Data 2 corresponds to P28 through P35, Data 3 to P36 through P43, Data 4 to P44 through 51, and Data 5 to P52 through P59. We don't need P57 through P59, so the most significant 3 bits of Data 5 can be disregarded.

Function Code 03: Read Holding Registers
Function Code 04: Read Input Registers

These functions can read 1 word (16 bits), and is usually used for counters, timers, and data registers. The following shows an example that reads slave address 3's registers D0 through D2.

Query:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X03	1	0 3	2
Start Address HI	0X70	1	7 0	2
Start Address LO	0X00	1	0 0	2
Length HI	0X00	1	0 0	2
Length LO	0X03	1	0 3	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

The query above asked for a length of 3. And since 1 word is 2 bytes, we will get 6 bytes total in the response.

Response:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X03	1	0 3	2
Byte Count	0X06	1	0 6	2
Data 1 LO	0X03	1	0 3	2
Data 1 HI	0XE8	1	E 8	2
Data 2 LO	0X01	1	0 1	2
Data 2 HI	0XF4	1	F 4	2
Data 3 LO	0X05	1	0 5	2
Data 3 HI	0X33	1	3 3	2
Length LO	0X03	1	0 3	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

Function Code 05: Force Single Coil

This function can be used to remotely set a register's value in units of bits. To turn on a bit register, 0xFF00 must be set. To turn off a bit register, 0x0000 must be sent. Any other values will be ignored. The following is an example showing slave address 3's P1 register being turned on.

Query:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X05	1	0 5	2
Start Address HI	0X01	1	0 1	2
Start Address LO	0X00	1	0 0	2
Length HI	0XFF	1	F F	2
Length LO	0X00	1	0 0	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

The response is typically an echo of the request indicating that the operation was successful.

Response:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X05	1	0 5	2
Start Address HI	0X01	1	0 1	2
Start Address LO	0X00	1	0 0	2
Length HI	0XFF	1	F F	2
Length LO	0X00	1	0 0	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

Function Code 06: Preset Single Registers

PLC’s can remotely control the status of its registers in word (2 bytes) units through this function code.

The following is an example showing slave address 3’s D1 being written.

Query:

Field	RTU	Bytes	ASCII	Bytes
Header		1	: (colon)	1
Slave Address	0X03	2	0 3	2
Function Code	0X06	2	0 6	2
Start Address HI	0X70	2	0 7	2
Start Address LO	0X01	2	0 1	2
Length HI	0X12	2	1 2	2
Length LO	0X34	2	3 4	2
Error Check	CRC	2	LRC	2
Ending Code		2	CR LF	2

The response is typically an echo of the request indicating that the operation was successful.

Response:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X06	1	0 6	2
Start Address HI	0X70	1	0 1	2
Start Address LO	0X01	1	7 0	2
Length HI	0X12	1	1 2	2
Length LO	0X34	1	3 4	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

Function Code 15: Force Multiple Coils

PLC’s can remotely control the status of its registers in units of multiple bits through this function code. The following is an example showing slave address 3’s P20 through P30 being turned on/off.

Query:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X0F	1	0 F	2
Start Address HI	0X00	1	0 0	2
Start Address LO	0X14	1	1 4	2
Length HI	0X00	1	0 0	2
Length LO	0X0B	1	0 B	2
Byte Count	0X02	1	0 2	2
Data 1	0XD1	1	D 1	2
Data 2	0X05	1	0 5	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

The following table shows how Data in the above query is divided. P20 corresponds to the least significant bit of the first byte sent, and P27 corresponds to the most significant bit. There will be a total of 2 bytes sent in this manner. Unused bits can be set to zero.

Bit	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0	1
Reg.	P27	P26	P25	P24	P23	P22	P21	P20						P30	P29	P28

Response:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X0F	1	0 F	2
Start Address HI	0X00	1	0 0	2
Start Address LO	0X14	1	1 4	2
Length HI	0X00	1	0 0	2
Length LO	0X0B	1	0 B	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

Function Code 16: Preset Multiple Registers

PLC's can remotely control the status of registers in units of multiple words through this function code. The following is an example showing slave address 3's D0 through D2 being written.

Query:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X10	1	1 0	2
Start Address HI	0X70	1	7 0	2
Start Address LO	0X00	1	0 0	2
Length HI	0X00	1	0 0	2
Length LO	0X03	1	0 3	2
Byte Count	0X06	1	0 6	2
Data 1 HI	0XD1	1	D 1	2
Data 1 LO	0X03	1	0 3	2
Data 2 HI	0X0A	1	0 A	2
Data 2 LO	0X12	1	1 2	2
Data 3 HI	0X04	1	0 4	2
Data 3 LO	0X05	1	0 5	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

Response:

Field	RTU	Bytes	ASCII	Bytes
Header			: (colon)	1
Slave Address	0X03	1	0 3	2
Function Code	0X10	1	1 0	2
Start Address HI	0X70	1	7 0	2
Start Address LO	0X00	1	0 0	2
Length HI	0X00	1	0 0	2
Length LO	0X03	1	0 3	2
Error Check	CRC	2	LRC	2
Ending Code			CR LF	2

Error Check

If there is an error in the data from the master, the slave will send back an error code.

Field	Hex	ASCII	Bytes
Header		: (colon)	1
Slave Address	0X03	0 3	2
Function Code	0X81	8 1	2
Error Code	0X09	0 9	2
Error Check		LRC	2
Ending Code		CR LF	2

The list of possible error codes is as follows:

Code	Error Name	Explanation
01	ILLEGAL FUNCTION	When a non-supported function code is received.
02	ILLEGAL DATA ADDRESS	When an incorrect address is received.
03	ILLEGAL DATA VALUE	When bad data is received.
09	LRC UNMATCH	When LRC is incorrect.

The error check is only for MODBUS ASCII; there is no error check in RTU. Modbus RTU uses a CRC to check for errors in transmission.

Using Modbus with HMI / SCADA Addresses

The following describes the address mechanism to be used from HMI / SCADA software.

Data Type	Range
Coil	1–999
Input Status	10001 – 19999
Input Register	30001 – 39999
Holding Register	40001 – 49999

To determine the appropriate address to use from HMI / SCADA software, refer to the following table.

Word Region (Holding/Input Registers) Relevant Function Codes: 3,4,6,16	
Area of the Cubloc to be addressed	Address to be entered in the HMI / SCADA software
D area (D0 ~ D511)	40001 ~ 40512
T area (T0 ~ T255)	41001 ~ 41256
C area (C 0~ C255)	42001 ~ 42256
WM area (WM0 ~ WM255)	43001 ~ 43256

Bit Region (Coil, Input Status) Relevant Function Codes : 1,2,4,15	
Area of the Cubloc to be addressed	Address to be entered in the HMI / SCADA software
P area (P0 ~ P127)	1 ~128 or 10001 ~10128
M area (M0 ~ M2047)	4097 ~ 6144 or 14097 ~ 16144

Modbus ASCII Master

There are no special commands needed to set the Cubloc as a Modbus master. One simply needs to use Cubloc's RS-232 commands like Get and Put.

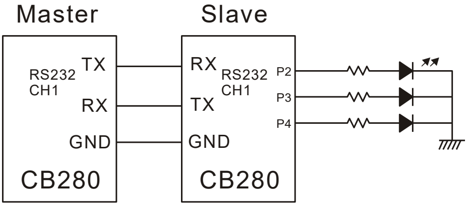
The following is an example illustrating the Cubloc running as a Modbus ASCII master:

```
'Master Source

Const Device = cb280
     Dim RDATA As String * 80
     Dim a As Byte, ct As Byte
     Dim b As String * 17
     Dim Port As Integer

     OpenCom 1,115200,3,80,80
     On Recv1 Gosub GETMODBUS   ' Data Receive Interrupt routine
     Set Until 1,60,10          ' When Ending Code (10)
                                ' on Channel 1 is discovered,
                                ' create an interrupt
     Do
            For Port=2 To 4
                   BitWrite Port, 1   'Turn P0,P1,P2 ON!
                   Delay 100
            Next
            For Port=2 To 4
                   BitWrite Port, 0   'Turn P0,P1,P2 OFF!
                   Delay 100
            Next

     Loop

GETMODBUS:
     If BLen(1,0) > 0 Then      ' If buffer empty then
            A=BLen(1,0)         ' Store the buffer length in A!
            Debug "GOT RESPONSE: "
            B=GetStr(1,A)       ' Store received data in B
            Debug B
     End If
     Return

End
     Sub BitWrite(K As Integer, D As Integer)
            Dim LRC As Integer
            PutStr 1,":0305"
            PutStr 1,Hp(k,4,1)
            If D=0 Then
                   PutStr 1,"0000"
```

```
                  LRC = -(3+5+K.Byte1+K.Byte0)     'Calculate LRC
            Else
                  PutStr 1,"00FF"
                  LRC = -(3+5+K.Byte1+K.Byte0+0xFF) ' LRC
            End If
            PutStr 1,Hex2(LRC),13,10  'Send

    End Sub
```

Modbus ASCII Slave

```
' Slave Source
    Const Device = cb280
    OpenCom 1,115200,3,80,80
    Set Modbus 0,3
    UsePin 2, Out
    UsePin 3, Out
    UsePin 4, Out
    Set Ladder On
```



When the slave finishes processing the data sent by the master, the return packet from the slave will cause a jump to the label GETMODBUS. The Set Until command is used to check for the ending code LF (10).

The GetStr command is used to store all received data in RDATA. The data in RDATA can be analyzed for any communication errors.

If the slave is not connected, the program will never jump to GETMODBUS.

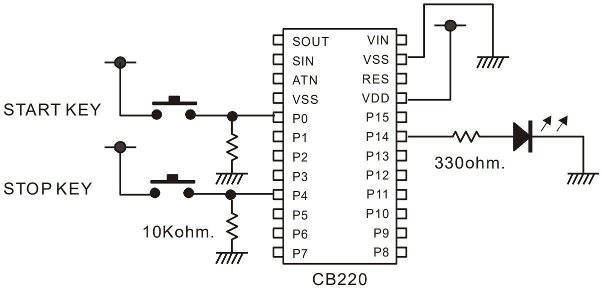
MEMO

Chapter 10: Application Notes

NOTE 1. Switch Input

Let’s say you are developing some kind of machine controlled by a Cubloc. The first thing you need is a user interface. Our task today is to build a machine that will receive input from a switch and process it to perform a task.

We will make a "start" and "stop" button that will turn a lamp on and off.

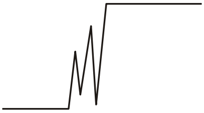


As you can see above, ports P0 and P4 will be connected to a pull-down resistor (resistor attached to ground). The CB220 will read these switches as logic-low or "off" when the switch is not pressed. To find out if these switches are pressed or unpressed, we can use Cubloc BASIC command `In`.

<Filename: startstopkey.cul>

```
Const Device = cb220

Dim a As Byte
Do
        If In(0) = 1 Then a = 1
        If In(4) = 1 Then a = 0
        Out 14,a
Loop
```

When the switch is pressed, a "bouncing" effect occurs from the switch’s mechanical spring.



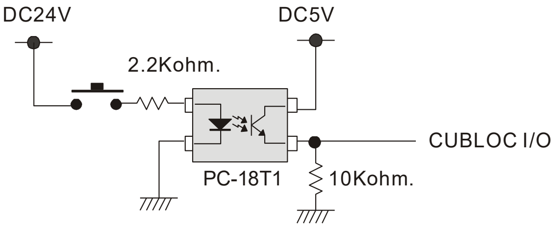
The above picture shows how bouncing can confuse Cubloc controller with alternating high and low signal levels. To get rid of this bouncing effect, a capacitor and resistor can be added to filter it out.

A simpler method is to use the `KeyInH` command instead of `In` which will remove the bouncing effect through software.

```
Const Device = cb220

Dim a As Byte
Do
        If KeyInH(0,20) = 1 Then a = 1
        If KeyInH(4,20) = 1 Then a = 0
        Out 14,a
Loop
```

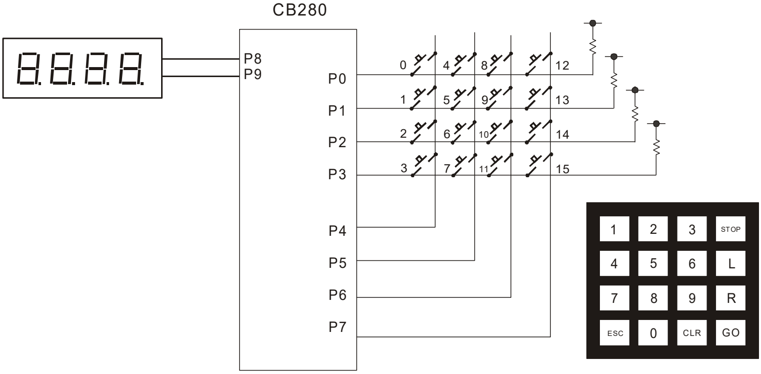
The 2nd parameter of `KeyInH(0, 20)` a delay before reading the input so the signal bounce has time to settle. This delay is called the debouncing time. The software will wait for 20ms before reading the input.

In industrial environments, there can be a lot of electromagnetic noise which can affect switch signals. To prevent this noise from affecting the circuit, a circuit diagram similar to one shown below can be used. Using a photocoupler, the user is able to raise the voltage and minimize the effect the noise has on the switch input.



NOTE 2. Keypad Input

This application demonstrates interfacing to a 4x4 keypad and displaying the results on a 4 digit seven segment module (CSG module)



The CSG module is a 4 digit seven segment LED module that can be connected via CUNET or the I2C interface to display numbers and custom characters.

<Filename: csgprint.cul>

```
Const Device = CB280
Set I2c 9,8
Dim i As Byte
Do
       CsgDec 0,i
       i = i + 1
Loop
```

If you connect the CSG to the CuNet connector and execute the program above, the CSG module will show incrementing numbers.

The key matrix can be easily read using the `KeyPad` command. If you look carefully at the keypad, you will see that the scan code does not match the actual key pressed. In order to read the correct key, we will use a `Byte` array (`KEYTABLE`) to map the scan codes to the appropriate key.

```
Const Device = CB280
Set I2c 9,8
Dim i As Integer
Dim K As Integer

Const Byte KEYTABLE = (1,4,7,10,2,5,8,0,3,6,9,11,12,13,14,15)
Do
       i = Keypad(0)
       If i < 16 Then
              i = KEYTABLE(i)
              Csgdec 0,i
       End If
Loop
```

And now we will make a simple program that receives input. When a number key input is received, it is displayed to the CSG module as a 4 digit number. The number is stored int the variable `K`, which is in BCD format. We then use the `BCD2Bin` command to convert the BCD value back into binary.

```
Const Device = CB280
Set I2c 9,8
Dim i As Integer
Dim K As Integer
Dim M As Integer
K = 0
Const Byte KEYTABLE = (1,4,7,10,2,5,8,0,3,6,9,11,12,13,14,15)
Do
       i = Keypad(0)
       If i < 16 Then
              i = KEYTABLE(i)
              If i < 10 Then
                    K = K << 4
                    K = K + i
                    Csghex 0,K
              End If
              '
              '     WAIT UNTIL KEY DEPRESS
              '
              Do While KeyPad(0) < 255
              Loop
```

```
                M = BCD2Bin(K)
                Debug Dec M,CR
            End If
    Loop
```

When there is no input, the returned scan code is 255. Using the statment `Do While keypad(0) < 255`, we wait until a key is released which will return a scan code of 255. This is to allow the processor to stop reading input while a key is being pressed. Otherwise, the processor might receive multiple key inputs since the Cubloc's execution speed is very fast.

Using `_D(0) = M`, you can pass the scan code to Ladder Logic's D0 register. If you need to use a keypad in Ladder Logic, you can make minor modifications to this code to quickly get your results.

NOTE 3. Temperature Sensor

There are many uses for devices that sense temperature. Refrigerators, heaters, air conditioners, automobiles, etc.. all make use of devices that sense temperature.

There are several different kinds of temperatures sensors. There are PT100, NTC, and PTC thermistors, and other chip-type sensors such as the DS1620.

We will take a look at the NTC thermistor and interface it with the Cubloc.

The NTC thermistor is a temperature-sensitive resistor. Depending on the temperature, the value of resistance will change. By reading the value of this resistance, we can determine the temperature.

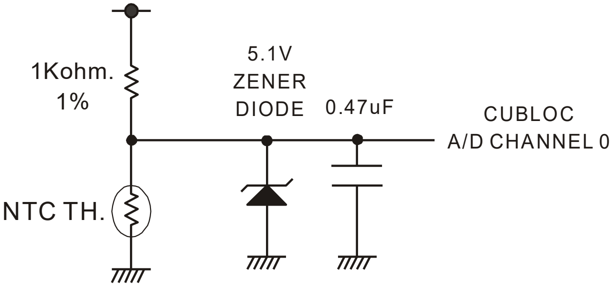
A common NTC thermistor resembles a diode. With this thermistor, we can sense temperatures between -30 and 250 degrees Celsius.

You can acquire an R-T (Resistance – Temperature) conversion table from the maker of the thermistor. The following is a diode-type 10Kohm NTC thermistor R-T conversion chart and table.

Temperature	Minimum	Average	Maximum
0	31260.0	32610.0	33987.7
1	29725.7	30993.7	32286.7
2	28275.6	29466.8	30680.6
3	26904.5	28023.9	29163.6
4	25607.8	26660.0	27730.3
5	24381.0	25370.2	26375.7
6	23220.0	24150.1	25094.9
7	22120.9	22995.7	23883.7
8	21080.1	21903.1	22737.7
9	20094.1	20868.5	21653.3
10	19159.9	19888.7	20626.7
11	18274.4	18960.5	19654.6
12	17434.8	18080.8	18733.8
13	16638.5	17246.9	17861.4
14	15883.1	16456.1	17034.4
15	15166.2	15706.0	16250.4
16	14485.7	14994.4	15506.9
17	13839.6	14318.9	14801.5
18	13225.9	13677.7	14132.2
19	12642.8	13068.7	13496.9
20	12088.7	12490.3	12893.6
21	11561.9	11940.6	12320.7

22	11061.0	11418.2	11776.4
23	10584.6	10921.6	11259.2
24	10131.3	10449.3	10767.5
25	9700.0	10000.0	10300.0
26	9281.3	9572.5	9864.0

For connecting the sensor to the Cubloc, please refer to the following circuit diagram. To protect against voltage surges, a zener diode is recommended, especially if the thermistor is attached to a long probe wire.



As you can see in the circuit diagram, we will be using an ADC (Analog-to-Digital) converter to read the voltage across the sensor. The A/D converter will convert the voltage into a value between 0 and 1024.

The most important part of this application note is the following table which converts the temperature and voltage to an A/D value between 0 and 1024. (Only some of the temperatures are shown.)

Temp	Resistance	Voltage	A/D value
-30	175996.6	4.971750865	1018
-29	165473.9	4.969965259	1018
-28	155643.6	4.968080404	1017
-27	146456.3	4.966091647	1017
-26	137866.4	4.963994167	1017
-25	129831.7	4.961782976	1016
-24	122313.4	4.959452909	1016
-23	115275.4	4.956998627	1015
-22	108684.3	4.954414614	1015
-21	102509.3	4.951695171	1014
-9	52288.3	4.90617073	1005
-8	49549.7	4.901087406	1004
-7	46970.5	4.895769279	1003
-6	44540.6	4.890207868	1002
-5	42250.5	4.884394522	1000
-4	40091.5	4.878320427	999
-3	38055.4	4.871976604	998
-2	36134.4	4.865353924	996
-1	34321.5	4.858443112	995

0	32610.0	4.851234752	994
1	30993.7	4.8437193	992
2	29466.8	4.835887094	990
3	28023.9	4.827728362	989
4	26660.0	4.819233234	987
5	25370.2	4.810391755	985
6	24150.1	4.801193902	983
7	22995.7	4.79162959	981
8	21903.1	4.781688696	979
9	20868.5	4.771361072	977
10	19888.7	4.760636561	975
11	18960.5	4.749505017	973
12	18080.8	4.737956327	970
13	17246.9	4.725980424	968
14	16456.1	4.713567319	965
15	15706.0	4.700707114	963
16	14994.4	4.68739003	960
17	14318.9	4.673606431	957
18	13677.7	4.659346849	954
19	13068.7	4.644602011	951
20	12490.3	4.629362861	948
21	11940.6	4.613620595	945
22	11418.2	4.597366683	942
23	10921.6	4.580592903	938
24	10449.3	4.563291365	935
25	10000.0	4.545454545	931
26	9572.5	4.527075313	927
27	9165.6	4.508146964	923
28	8778.3	4.488663246	919
29	8409.4	4.468618396	915
30	8058.1	4.448007162	911
31	7723.3	4.426824842	907
32	7404.3	4.405067304	902
33	7100.2	4.382731022	898
34	6810.2	4.359813102	893
35	6533.7	4.336311306	888
36	6269.8	4.312224084	883
37	6018.0	4.287550592	878
38	5777.7	4.262290722	873
39	5548.3	4.236445118	868
50	3606.1	3.914475937	802
51	3472.1	3.881948015	795
52	3343.7	3.848917708	788
53	3220.8	3.815397329	781
54	3103.1	3.781399998	774
55	2990.2	3.746939622	767
56	2882.1	3.712030877	760
57	2778.4	3.676689176	753
58	2679.0	3.640930651	746
59	2583.6	3.604772114	738
81	1220.4	2.748157207	563
82	1181.9	2.7084025	555
83	1144.8	2.668747011	547
84	1109.0	2.629210536	538
85	1074.5	2.589812422	530
86	1041.3	2.550571543	522

87	1009.2	2.511506263	514
88	978.3	2.472634416	506
89	948.5	2.433973277	498
90	919.8	2.395539544	491
91	892.0	2.357349316	483
92	865.3	2.319418079	475
93	839.4	2.281760687	467
94	814.5	2.244391354	460
95	790.4	2.207323646	452
96	767.1	2.170570465	445
97	744.7	2.134144055	437
98	723.0	2.098055989	430
99	702.0	2.062317177	422
100	681.8	2.026937858	415
101	662.2	1.99192761	408
102	643.3	1.957295352	401
103	625.0	1.92304935	394
104	607.3	1.889197225	387
105	590.2	1.855745964	380
106	573.7	1.822701928	373
107	557.7	1.790070865	367
108	542.2	1.757857926	360
109	527.2	1.726067674	353
239	33.5	0.162295782	33
240	33.0	0.159800146	33
241	32.5	0.157350769	32
242	32.0	0.154946682	32
243	31.5	0.152586936	31
244	31.0	0.150270604	31
245	30.5	0.147996779	30
246	30.0	0.145764577	30
247	29.6	0.143573131	29
248	29.1	0.141421596	29
249	28.7	0.139309144	29
250	28.2	0.137234968	28

```
     '
     '      NTC THERMISTOR READ TABLE
     '      10K DIODE TYPE
     '
     Const Device = cb280

Const Integer TH_TABLE = (992,990,989,987,985,983,981,979,977,975,
                   973,970,968,965,963,960,957,954,951,948,
                   945,942,938,935,931,927,923,919,915,911,
                   907,902,898,893,888,883,878,873,868,862,
                   857,851,845,839,833,827,821,815,808,802,
                   795,788,781,774,767,760,753,746,738,731,
                   723,716,708,700,692,684,677,669,661,652,
                   644,636,628,620,612,604,596,587,579,571,
                   563,555,547,538,530,522,514,506,498,491,
                   483,475,467,460,452,445,437,430,422,415)

     Dim a As Integer,b As Integer
```

```
    Do
            b = Tadin(0)
            If b > 990 Or b < 400 Then
                  Debug "Out of Range"    'Check short or open th.
            End If
            For a=0 To 100
                  If b > TH_TABLE(a) Then Exit For
            Next
            Debug Dec a,cr
            Delay 500
    Loop
```

<Filename: ntcth.cul>

By using the TADIn command for AD conversion, the Cubloc will automatically calculate the average of 10 A/D conversion reads for more precise results. The sample program shown here will be able to sense between 0 and 100 degrees. A minor modification to the code would allow a larger temperature range.

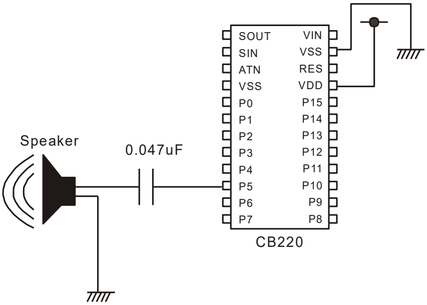
The formula for acquiring the A/D conversion value from the R-T table is as follows:

$$V = \frac{5}{(1000 + THR)} \times THR$$

THR is the resistance value. 1000 is for a 1K Ohm resistor and 5 is for 5 volts. The 10 bit A/D converter of the Cubloc will return a value between 0 and 1024. Therefore, to get the A/D value, you must multiply the result, V, by 204.8. A chart can be made by plotting this formula in a spreadsheet.

NOTE 4. Sound Bytes

This application note will demonstrate a few examples that generate sound with the Cubloc. Sound can be generated using the Cubloc's I/O port or PWM channel. With a PWM channel, sounds of varying frequency can be generated.



```
Const Device = CB220
Dim PLAYSTR As String
Low 5
Freqout 0,5236         'Create a sound with frequency of 440Hz
Delay 500              'Delay
Pwmoff 0               'Stop Sound by turning off PWM
```

The example above shows the CB220's PWM channel 0 of CB220 being used with the `FreqOut` command to produce a sound.

With commands like `FreqOut` and `Delay`, simple sounds can be created.

```
Const Device = CB220
Low 5
FreqOut 0,4403
Delay 200
FreqOut 0,3703
Delay 200
FreqOut 0,3114
Delay 200
FreqOut 0,2202
Delay 200
PwmOff 0
```

By changing the frequencies, a simple program can be made that plays musical notes.

Octave 4							Octave 5						
A	B	C	D	E	F	G	A	B	C	D	E	F	G
A	B	C	D	E	F	G	H	I	J	K	L	M	N

To express one note, 2 characters are used: the first character is for the frequency of the note and second character is for the length of the note.

```
Const Device = CB220

Dim PLAYSTR As String
Low 5
PLAYSTR = "G5E3E3G3E3C5"
PLAY 0,PLAYSTR

Do
Loop
End

Sub PLAY(CH As Byte,NOTE As String)
     Dim PL As Byte
     Dim CHAR As Byte
     Const Integer PLAYTABLE = (5236,4665,4403,3923,3495,3299,2939,
            2618,2333,2202,1961,1747,1649,1469,0)
     For PL=1 To Len(NOTE) Step 2
            CHAR = Asc(Mid(NOTE,PL,1)) - &H41
            FreqOut CH,PLAYTABLE(CHAR)
            CHAR = Asc(Mid(NOTE,PL+1,1)) - &H30
            Delay CHAR*100
     Next
     PwmOff CH
End Sub
```

When using the PWM port for other purposes, the `FreqOut` command cannot be used. To get around this limitation, any regular I/O port can be used to create sound.

The following example shows how to make an alert sound with I/O port P4. This program uses the `Reverse` and `UDelay` commands to alternate the I/O port between logic-high and logic-low

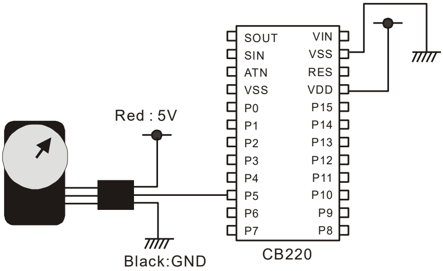
```
Const Device = CB220

Low 4
Do
SOUND 4,110,60
SOUND 4,80,60
SOUND 4,40,160
Loop
End

Sub SOUND(PN As Byte,FR As Byte,LN As Byte)
           Dim SI As Byte,SJ As Byte
           For SJ = 0 To LN
                 Reverse PN
                 UDelay FR
                 Reverse PN
                 UDelay FR
           Next
End Sub
```

NOTE 5. RC Servo Motor

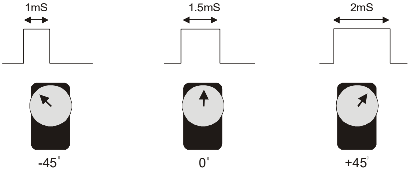
RC servo motors (or "servos" for short) are used by many hobbyists to make remote control cars, planes, etc.,. In recent years, they have been used for robot arms, legs, and wheels.

With the Cubloc's PWM outputs, a servo can be easily added to any project.

A servo typically has three wires. The black wire is ground and red wire is for power. The other yellow wire is for receiving a PWM signal. A typical pulse rate is approximately 60 pulses per second.



The servo will move to a location set by the PWM's pulse and duty cycle, and will hold its position.



Every RC servo motor is different, but as an example, pulses of 1ms, 1.5ms, and 2ms might stop a RC servo motor to -45 degrees, 0 degrees, and +45 degrees respectively.

```
Const Device = CB280
Low 5
Pwm 0,2500,32768
```

When the code above is executed, a 1ms pulse will be generated from I/O port P5 and the RC servo motor will position itself to -45 degrees.

```
Const Device = CB280
Low 5
Pwm 0,4000,32768
```

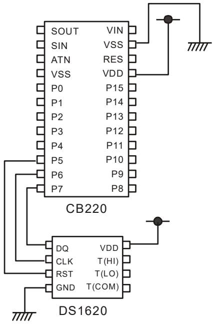
When the code above is executed, a 1.5ms pulse will be generated from I/O port P5 and the RC servo will position itself to 0 degrees.

By simply changing the duty value of PWM command, the RC servo can easily be controlled. For the CB220, 3 RC servos can be controlled simultaneously while the CB280 and CB290 can control up to 6 RC servos. The CB400/CB405 can control up to 12 servos.

Warning: When the RC servo is in operation, it will need about 500mA of current. Please make sure to use a power supply of at least 500mA.

NOTE 6. Digital Thermometer

The DS1620 is a digital thermometer. The chip has an internal temperature conversion table so the user does not have to make a separate table. Temperatures between -55 and 125 degrees Celsius can be measured by the DS1620 in units of 0.5 degrees.

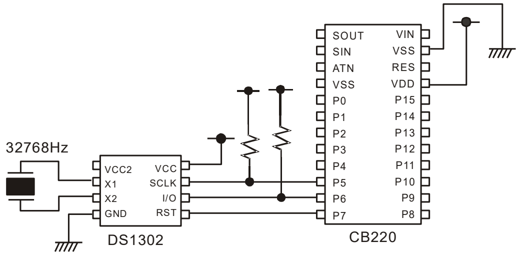


```
Const Device = CB220
Const iorst = 7
Const ioclk = 6
Const iodq = 5
Dim i As Integer
Delay 100
High iorst          ' init ds1620
ShiftOut ioclk,iodq,0,12,8
ShiftOut ioclk,iodq,0,3,8
Low iorst
High iorst

ShiftOut ioclk,iodq,0,&hEE,8
Low iorst
Do
        High iorst
        ShiftOut ioclk,iodq,0,&haa,8
        i = ShiftIn(ioclk,iodq,4,9)
        i = i
        debug dec i,cr
        Low iorst
        Delay 100
Loop
```

The final value received must be divided by 2 to obtain the correct temperature.

NOTE 7. DS1302 RTC

The DS1302 RTC (Real Time Clock) is a chip that acts as an electronic time keeper. It has the ability to keep time and date in real-time. We will show you how to implement this clock chip into your application.



Pin	Function	I/O Direction	Explanation
RST	Reset	Input	Data transfer when High
SCLK	System Clock	Input	Clock signal
I/O	Data Input/Output	Input / Output	Data input/output

```
Const Device = CB220
     Const iorst = 7
     Const iodio = 6
     Const ioclk = 5
     Dim i As Integer
     Dim adr As Byte
     High iorst
     ShiftOut ioclk,iodio,0,&h8e,8
     ShiftOut ioclk,iodio,0,0,8
     Low iorst
     Delay 1
     High iorst
     ShiftOut ioclk,iodio,0,&h80,8
     ShiftOut ioclk,iodio,0,&H50,8
     Low iorst

     Do
            High iorst
            adr = &h81
            ShiftOut ioclk,iodio,0,adr,8
            i = ShiftIn(ioclk,iodio,4,8)
            Debug Hex i,cr
            Low iorst
            Delay 1000
     Loop
```

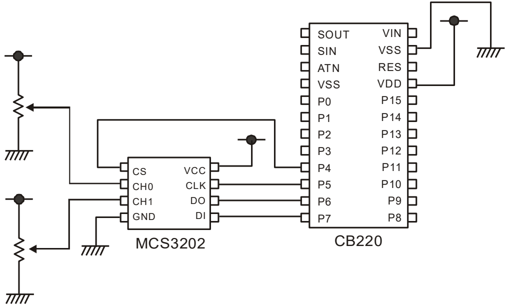
The code above will read address 0, the seconds value, and display it in the debug window. At the beginning of the program, writes are enabled to the DS1302 chip and address 0 is set to 50 seconds. Within the `Do Loop`, data is read from the DS1302. The DS1302 has 6 addresses as shown below:

ADDRESS 0 (sec)	CH	10 SEC			SEC			
ADDRESS 1 (min)	0	10 MIN			MIN			
ADDRESS 2 (hour)	12/24	0	10 A/P	HR	HR			
ADDRESS 3 (date)	0	0	10DATE		DATE			
ADDRESS 4 (month)	0	0	0	10M	MONTH			
ADDRESS 6 (day)	0	0	0	0	0	DAY		
ADDRESS 6 (year)	10 YEAR				YEAR			

These addresses can be used to read and write to the DS1302. Please note that the data is in BCD code format.

NOTE 8. MCP3202 12 Bit A/D Conversion

The Cubloc has a 10 bit A/D converter. For greater resolution, meaning greater precision, you can use a chip like the MCP3202. The MCP3202 is a 12 bit A/D converter that supports the SPI protocol. Here we will show you how to implement this 12 bit A/D converter into your project.

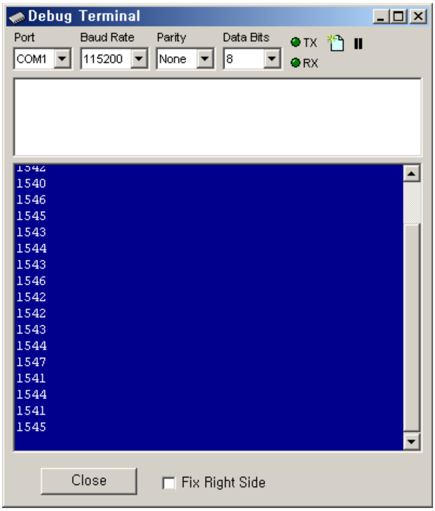


Pin	Function	I/O Direction	Explanation
CS	Chip Select	Input	Low for data communication
CLK	Clock	Input	Clock signal
DI	Data Input	Input	Data input from MCP3202
DO	Data Output	Output	Data output from MCP3202

```
Const Device = CB220
Const iodi = 7
Const iodo = 6
Const ioclk = 5
Const iocs = 4
Dim i As Byte
Dim ad As Integer
Do
        Low iocs
        i = &b1011 'Channel 0
        'i = &h1111 'Channel 1
        Shiftout ioclk,iodi,0,i,4
        ad = Shiftin(ioclk,iodo,3,12)
        High iocs
        Debug Dec ad,cr
        Delay 100
Loop
```

The MCP3202 will convert voltage coming into CH0 and CH1 to a digital value and retain it. SPI communication can then be used to read the value from the MCP3202.

The voltage measured on the MCP320 CH0 and CH1 pins must not be greater than the voltage supplied to the MCP3202. The result of the A/D conversion is displayed in the debug window.



Port: COM1
Baud Rate: 115200
Parity: None
Data Bits: 8

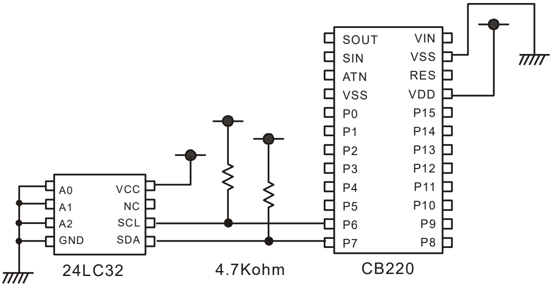
```
1542
1540
1546
1545
1543
1544
1543
1546
1542
1542
1543
1544
1547
1541
1544
1541
1545
```

NOTE 9. Reading from and Writing to an EEPROM

Typically EEPROMs store between 0.5 to 64KB of data. Data is retained even after powering off. For example, if you wanted to retain a temperature setting for a temperature controller, you could simply store the value of the temperature in the EEPROM in case of a power outage.

The Cubloc has an internal EEPROM of 4KB. This EEPROM can be used to store small amounts of data. If a larger EEPROM is needed, the 24LC512 can be used to store up to 64KB of data.

The following example will demonstrate how to access the 24LC32 4KB EEPROM using the I2C protocol. Serial EEPROMs usually support either SPI or I2C. I2C EEPROM part numbers begin with 24XXXX and SPI EEPROM part numbers begin with 93XXX.



```
    Const Device = CB220
    Dim adr As Integer
    Dim data As Byte
    Dim a As Byte
    data = &ha6
    adr = &h3
    Set I2C 7,6
    Do
            I2CStart
            If I2CWrite(&b10100000)= 1 Then Goto err_proc
            a=I2CWrite(adr.byte1)
            a=I2CWrite(adr.lowbyte)
            a=I2CWrite(data)
            I2CStop
            Delay 1000
            I2CStart
            a=I2CWrite(&b10100000)
            a=I2CWrite(adr.byte1)
            a=I2CWrite(adr.lowbyte)
            I2CStart
            a=I2CWrite(&b10100001)
            a=I2CRread(0)
            I2CStop
            Debug Hex a,cr
            ADR = ADR + 1
            DATA = DATA + 1
     Loop

err_proc:
     Debug "Error !"
     Do
     Loop
```

This example program will write a number to the EEPROM and read it back. When this program runs correctly, numbers will increment in the debug window. This program can be easily modified to support other EEPROMs.

Note: Typically, EEPROMs need a delay of about 5ms after a write operation.

MEMO

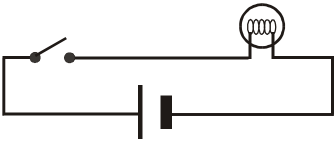
Chapter 11: Ladder Logic

WARNING

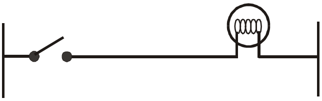
If you do not use the `SET LADDER ON` command in BASIC, Ladder Logic will not be executed.

Ladder Logic Basics

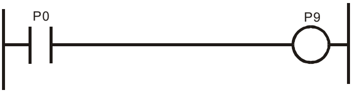
The following is an example of an electrical circuit with one switch and a lamp.



If you take out the power, the following results:

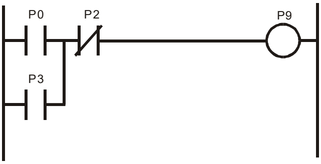


If you express the above circuit using Ladder Logic, the following results:

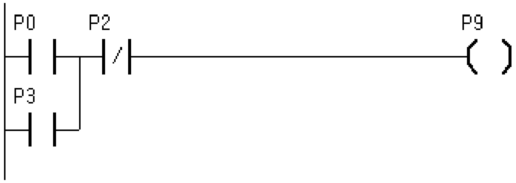


As you can see, Ladder Logic is basically an easy way to model electrical circuits in software. The switch corresponds to the input port P0 and the lamp corresponds to the output port P9.

There are many ways to connect other devices such as timers, counters, etc.,. The following is an example illustrating OR and AND logic using Ladder Logic:



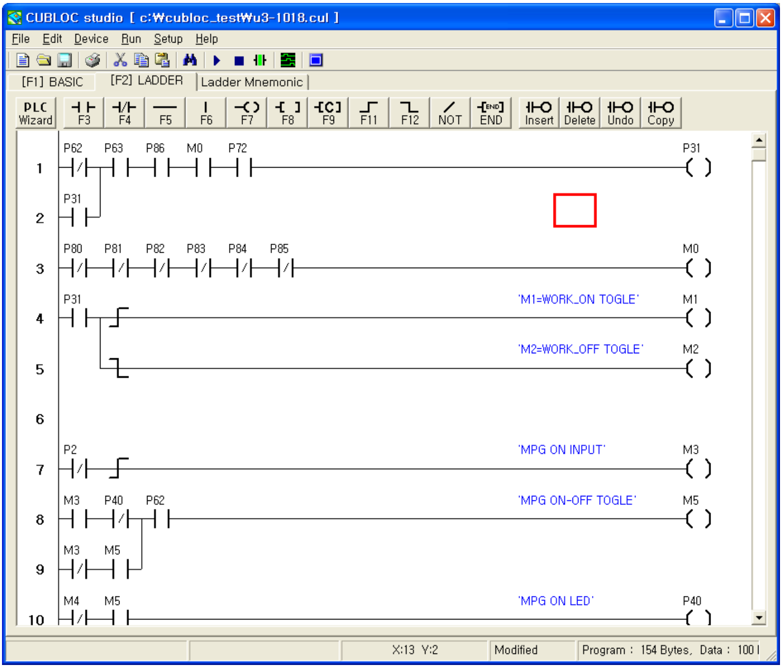
In this circuit diagram, P0 and P3 are ORed and that result is ANDed with P2 If you express the circuit diagram above in Ladder Logic, it will be as follows:



In Cubloc Studio, the right side is not shown. In Cubloc Ladder Logic, P0, P1, P2 are called "Registers".

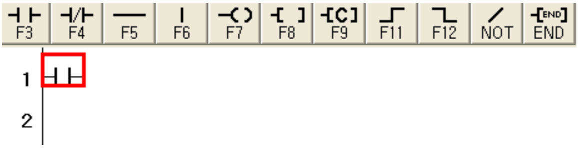
Creating Ladder Logic Programs

The screen below shows you how Ladder Logic programs are created in Cubloc Studio.

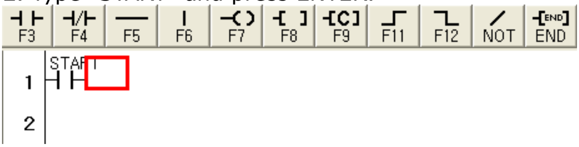


The red box shown above is the Ladder Logic cursor. You may use the keyboard's up, down, left, and right keys or the mouse to control the red box. After moving to the desired position, you can use the keys F3 through F12 to place the desired symbol. You can also enter text for each symbol.

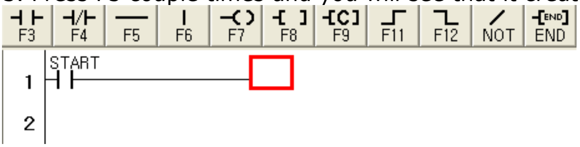
1. Press F3 to make a contact.



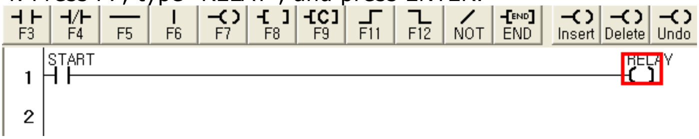
2. Type "START" and press ENTER.



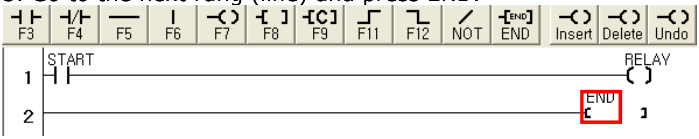
3. Press F5 couple times and you will see that it creates a line.



4. Press F7, type "RELAY", and press ENTER.



5. Go to the next rung (line) and press END.

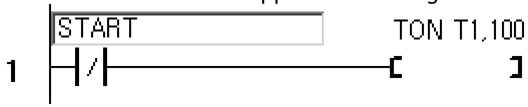


At the very end of the Ladder Logic program, you must always put an END command.

Editing Ladder Logic Text

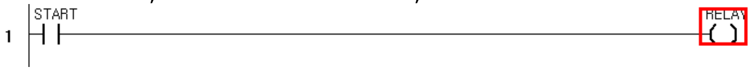
Editing Text

To edit existing text, place the cursor in the desired location and press ENTER. A text box will appear for editing the text.

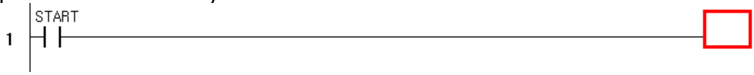


Erasing a Cell

To erase a cell, select it with the cursor, and...

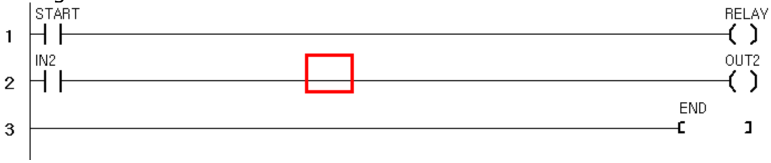


press the SPACE key.

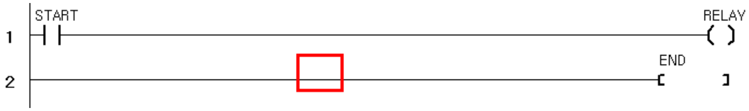


Erasing a Rung (one line)

A rung is a row in Ladder.

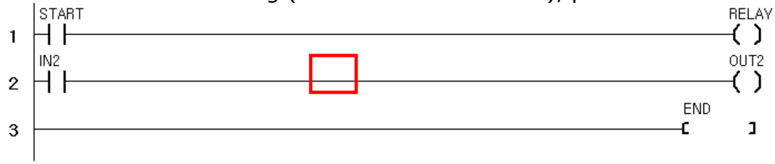


You can press CTRL-D to erase a rung. This actually moves the rung to a buffer.



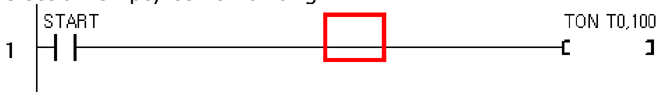
Rung Recovery

To recover an erased rung (recall it from the buffer), press CTRL-U.

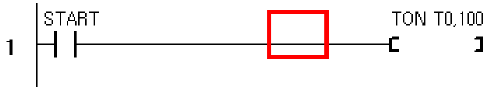


Cell Insert and Delete

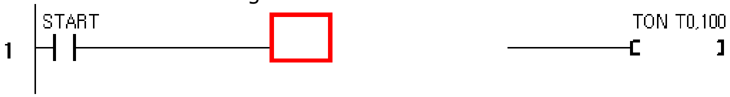
Select an empty cell on a rung.



If you press the DEL key, the cell is erased and items on the right are pulled one cell to the left.

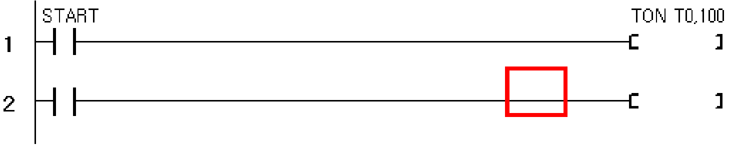


If you press the INS key, a blank cell is inserted and items on the right are moved one cell to the right.



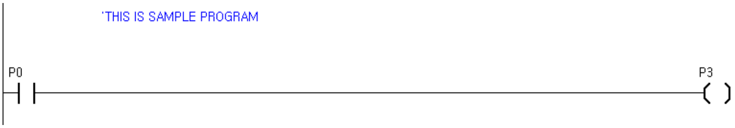
Rung Copy

When the same style of rung is needed, you can press CTRL-A and it will copy the rung above and pasted it one rung below.



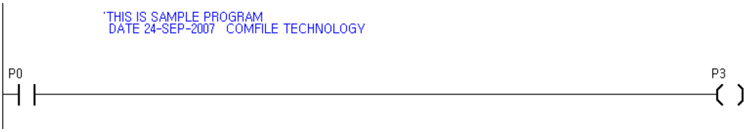
Comments

You can enter comments by adding an apostrophe (').



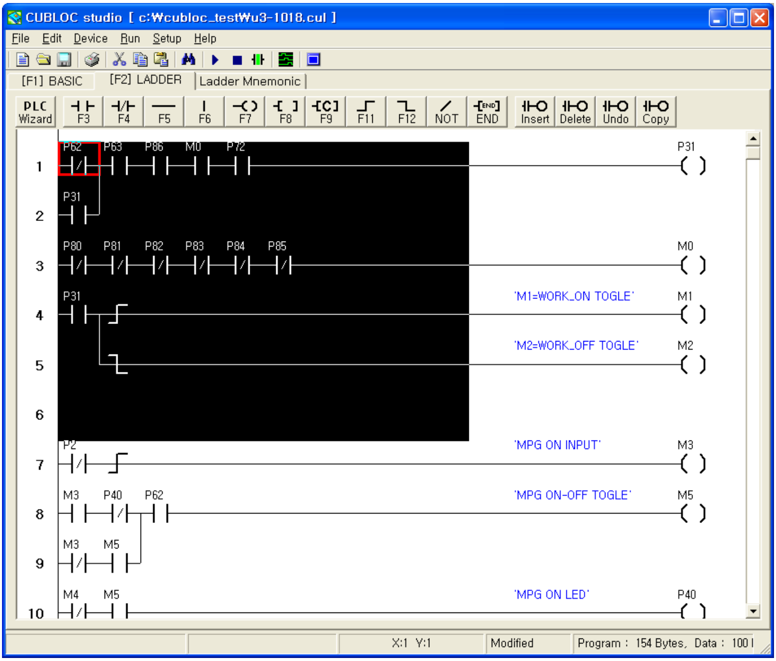
You can use a semi-colon (;) to drop to the next line.
For example:

"This is Sample Program; Date 24-Sep-2007 Comfile Technology"



Ladder Logic Block Copy and Paste

You can make a block of ladder logic and copy and paste to different parts of the program.

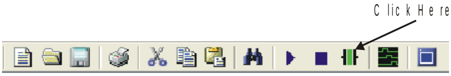


Click and drag the mouse to select the block you wish to copy. Press CTRL-C to copy and CTRL-V to paste. Similar to text editing, you can press CTRL-X and CTRL-V to cut and paste also.

NOTE: Please be aware that the Ladder Logic editor does not support undo.

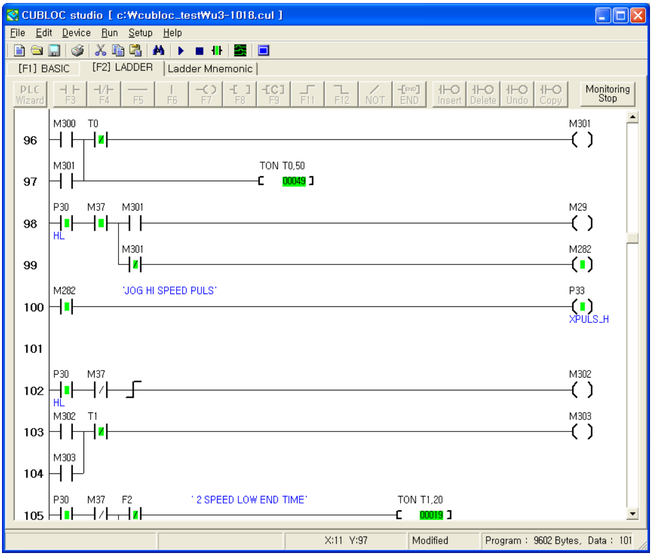
Monitoring

Cubloc Studio supports real-time monitoring of Ladder Logic.



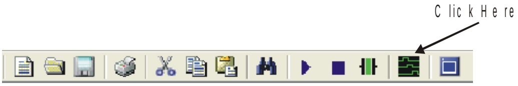
Status of contacts that are ON will be displayed in green. Timer and counter values will be displayed in decimal.

The monitoring speed can be controlled by going to "Setup" -> "Ladder Logic Environment Options..." and adjusting the "Monitoring Speed" slider. If the monitoring speed is too fast, it can negatively affect Cubloc's communications as monitoring consumes Cubloc's resources. We recommend a monitoring speed of 5.



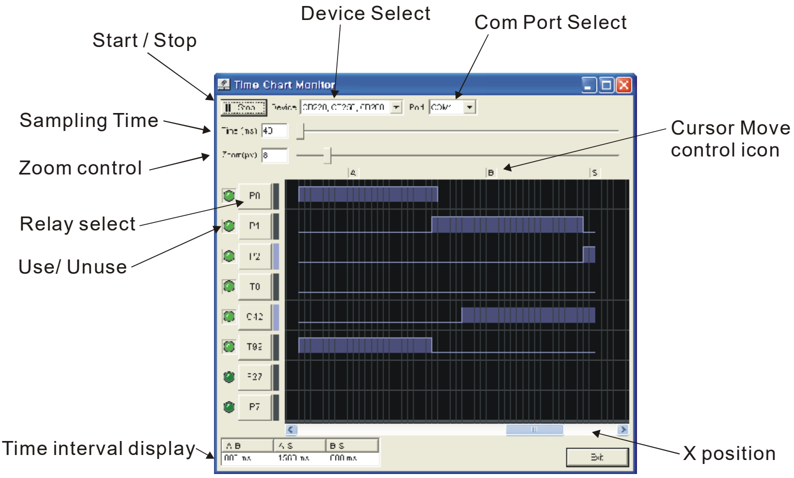
NOTE: Please be sure to stop monitoring before editing or downloading.

Time Chart Monitoring



With Time Chart Monitoring, Ladder Logic contacts can be analyzed in a time chart. The minimum width of the time chart is 40ms. The "Zoom" control can be used to measure the width of each pulse after stopping.

Up to 8 Registers can be monitored at one time.



Device [illegible]
Port COM
Time (ms) 40
Zoom(px) 8

A B	A S	B S
907 ms	1507 ms	600 ms

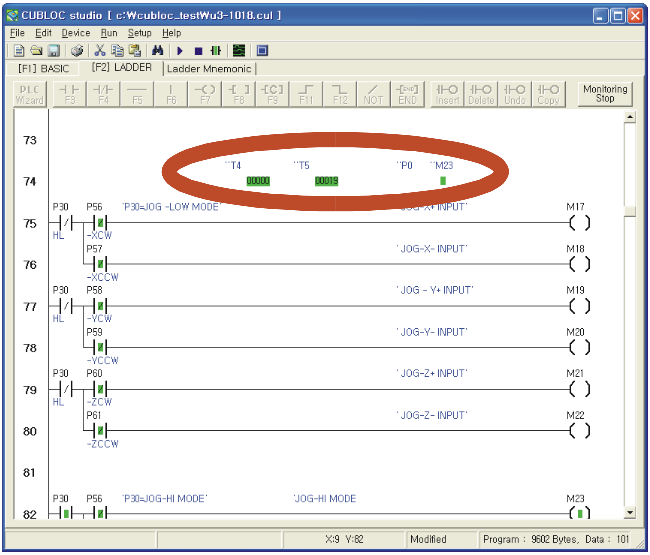
To use the Time Chart Monitor, you must turn Debug off in BASIC. To do this, simply add the `Set Debug Off` command at the very beginning of the BASIC program.

NOTE: While using the Time Chart Monitor, Ladder Monitoring may not be used.

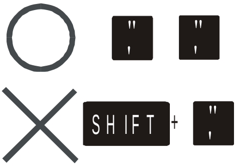
Watch Point

The Watch Point feature is useful when, in a long Ladder Logic program, two registers cannot be viewed simultaneously in the same screen due to their location in the ladder. Two apostrophes ('') are used to add a watch point.

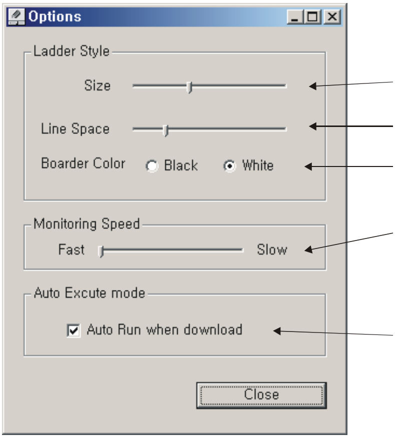
Examples:
''P0   ''P1   ''D0



NOTE: To set a watch point it's two apostrophes(''), not a quotation mark(") that must be used.



Options Window



Options

Ladder Style

Size

Line Space

Boarder Color ○ Black ● White

Monitoring Speed

Fast Slow

Auto Excute mode

☑ Auto Run when download

Close

LADDER size adjust

LADDER line space adjust

LADDER background color

LADDER monitorring speed setting

Auto run when download

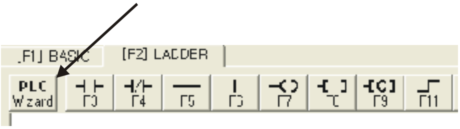
If you select "Auto Run when download", the program will automatically reset itself after downloading. This can become a problem for machines that are sensitive to resets. To precisely control when to reset a program turn this option off.

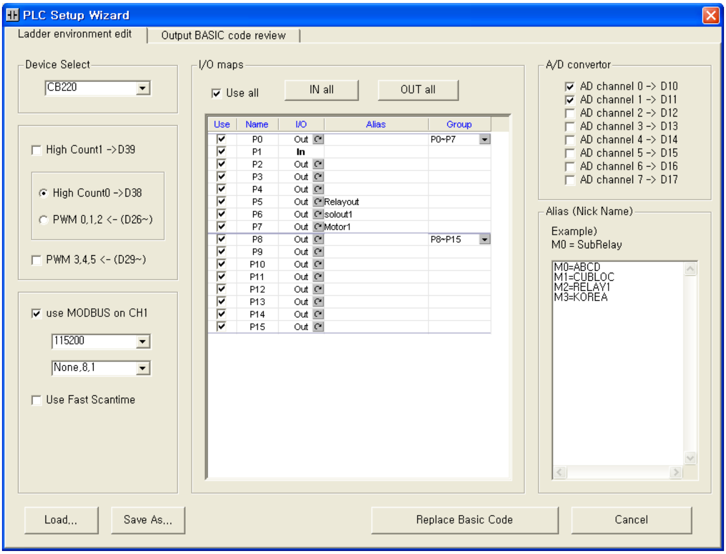
In the "Help" menu, Cubloc Studio upgrade and version information can be found.

PLC Setup Wizard

To use Ladder Logic in Cubloc, you must create some minimal BASIC code. Although very simple, this can be hard for first-timers. You can use the PLC Setup Wizard and setup the I/Os you will be using and create the BASIC source automatically.

PLC SETUP WIZARD





Device Select

CB220

□ High Count1 ->D39

● High Count0 ->D38

○ PWM 0,1,2 <- (D26~)

□ PWM 3,4,5 <- (D29~)

☑ use MODBUS on CH1

115200

None,8,1

□ Use Fast Scantime

I/O maps

☑ Use all IN all OUT all

Use	Name	I/O	Alias	Group
☑	P0	Out		P0~P7
☑	P1	**In**		
☑	P2	Out		
☑	P3	Out		
☑	P4	Out		
☑	P5	Out	Relayout	
☑	P6	Out	solout1	
☑	P7	Out	Motor1	
☑	P8	Out		P8~P15
☑	P9	Out		
☑	P10	Out		
☑	P11	Out		
☑	P12	Out		
☑	P13	Out		
☑	P14	Out		
☑	P15	Out		

A/D convertor

☑ AD channel 0 -> D10
☑ AD channel 1 -> D11
□ AD channel 2 -> D12
□ AD channel 3 -> D13
□ AD channel 4 -> D14
□ AD channel 5 -> D15
□ AD channel 6 -> D16
□ AD channel 7 -> D17

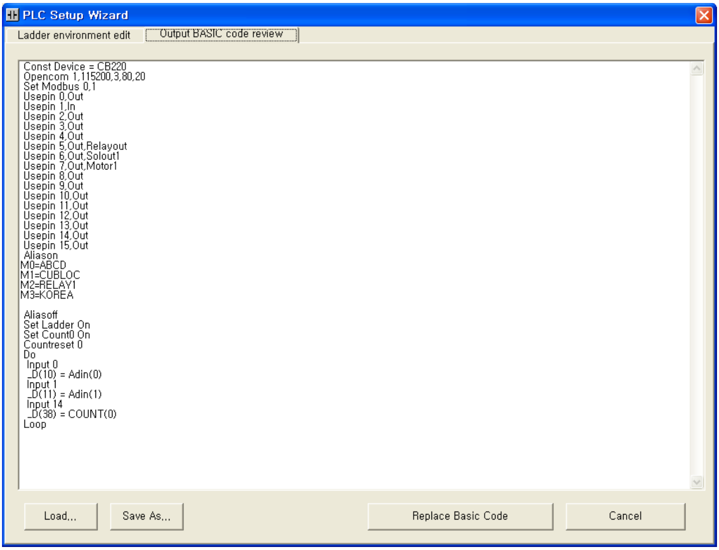
Alias (Nick Name)

Example)
M0 = SubRelay

```
M0=ABCD
M1=CUBLOC
M2=RELAY1
M3=KOREA
```

As you can see in above screen, device name, I/O mode, alias, and other features can be set simply by clicking.

You can set aliases for registers, turn Modbus on, and set the Modbus baud rate. You can always review the current BASIC code generated in real-time by navigating to the "Output BASIC code review" tab.



```
Const Device = CB220
Opencom 1,115200,3,80,20
Set Modbus 0,1
Usepin 0,Out
Usepin 1,In
Usepin 2,Out
Usepin 3,Out
Usepin 4,Out
Usepin 5,Out,Relayout
Usepin 6,Out,Solout1
Usepin 7,Out,Motor1
Usepin 8,Out
Usepin 9,Out
Usepin 10,Out
Usepin 11,Out
Usepin 12,Out
Usepin 13,Out
Usepin 14,Out
Usepin 15,Out
 Aliason
M0=ABCD
M1=CUBLOC
M2=RELAY1
M3=KOREA

 Aliasoff
Set Ladder On
Set Count0 On
Countreset 0
Do
 Input 0
 _D(10) = Adin(0)
 Input 1
 _D(11) = Adin(1)
 Input 14
 _D(38) = COUNT(0)
Loop
```

To get values for A/D, PWM, or COUNT, read from the D registers. For ADC0, the AD value is stored in D(10). Reading from register D(10) will return the AD value for ADC0.

To output PWM3, simply write to register D(29).

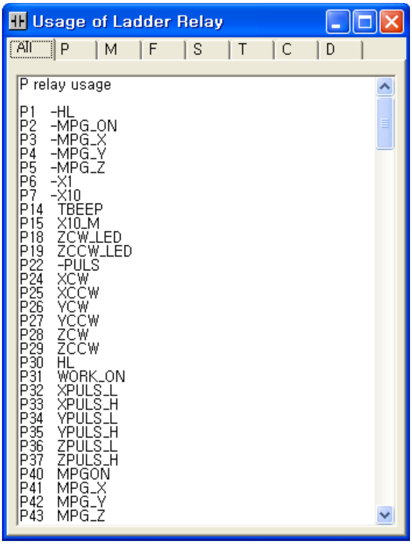
For HIGH COUNT1, simply read from register D(39). If necessary, the register for storing and writing values can be changed by modifying the BASIC code.

When finished making changes, click the "Replace Basic Code" button to produce the final BASIC code. Please be aware that any existing BASIC code will be overwritten.

Changes can also be saved to a file by clicking the "Save As..." button. Click the "Load..." button to restore from a saved file.

Usage of Ladder Register

With this feature, the aliases of all registers can be seen. A great deal of time can be saved using this feature while debugging and developing the final product. Go to **Run->View Register Usage** to open this window.



```
P relay usage

P1   -HL
P2   -MPG_ON
P3   -MPG_X
P4   -MPG_Y
P5   -MPG_Z
P6   -X1
P7   -X10
P14   TBEEP
P15   X10_M
P18   ZCW_LED
P19   ZCCW_LED
P22   -PULS
P24   XCW
P25   XCCW
P26   YCW
P27   YCCW
P28   ZCW
P29   ZCCW
P30   HL
P31   WORK_ON
P32   XPULS_L
P33   XPULS_H
P34   YPULS_L
P35   YPULS_H
P36   ZPULS_L
P37   ZPULS_H
P40   MPGON
P41   MPG_X
P42   MPG_Y
P43   MPG_Z
```

Registers

CB220, CB280 Registers

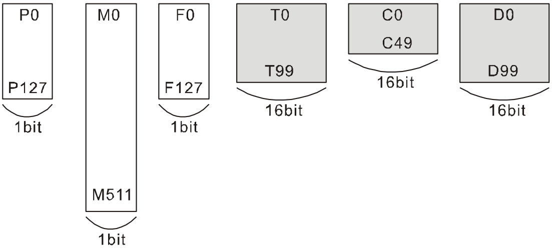
The following is a chart showing the registers for the CB220 and CB280.

Register Name	Range	Units	Feature
Input/Output Register P	P0 to P127	1 bit	Interface w/ External devices
Internal Registers M	M0 to M511	1 bit	Internal Registers
Special Register F	F0 to F127	1 bit	System Status
Timer T	T0 to T99	16 bit (1 Word)	For Timers
Counter C	C0 to C49	16 bit (1Word)	For Counters
Step Enable S	S0 to S15	256 steps (1 Byte)	For Step Enabling
Data Memory D	D0 to 99	16bit (1 Word)	For Storing Data

P, M, and F registers are in bit units whereas T, C, and D are in word units. To access P, M, and F registers in word units, use WP, WM, and WF respectively.

Register Name	Range	Units	Feature
WP	WP0 to 7	16 bit (1 Word)	Register P Word Access
WM	WM0 to WM31	16 bit (1 Word)	Register M Word Access
WF	WF0 to WF7	16 bit (1 Word)	Register F Word Access

WP0 consists of registers P0 through P15. P0 is the least significant bit of WP0 and P15 is the most significant bit. These registers are often used with commands like `WMOV`.



CB290 and CB405 Registers

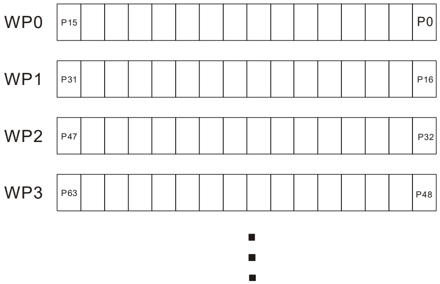
The following is a chart showing the registers for the CB290 and CB405. The CB290 and CB405 have more M, C, T, and D registers than the CB220 and CB280.

Register Name	Range	Units	Feature
Input/Output Register P	P0 to P127	1 bit	Interface w/ External devices
Internal Registers M	M0 to M2047	1 bit	Internal Registers
Special Register F	F0 to F127	1 bit	System Status
Timer T	T0 to T255	16 bit (1 Word)	For Timers
Counter C	C0 to C255	16 bit (1 Word)	For Counters
Step Enable S	S0 to S15	256 steps (1 Byte)	For Step Enabling
Data Memory D	D0 to 511	16 bit (1 Word)	For Storing Data

P, M, and F Registers are in bit units whereas T, C, and D are in word units. To access P, M, and F registers in word units, use WP, WM, and WF respectively.

Register Name	Range	Units	Feature
WP	WP0 to 7	16 bit (1 Word)	Register P Word Access
WM	WM0 to WM63	16 bit (1 Word)	Register M Word Access
WF	WF0 to WF7	16 bit (1 Word)	Register F Word Access

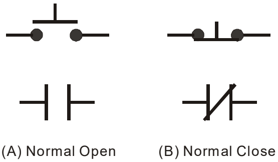
WP0 contains P0 through P15. P0 is the least significant bit of WP0 and P15 is the most significant bit. These registers are often used with commands like `WMOV`.



Ladder Symbols

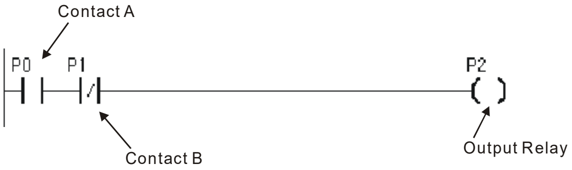
Contact A, Contact B

Contact A is "Normally Open" and closes when a signal is received. On the other hand, Contact B is "Normally Closed" and opens when a signal is received.



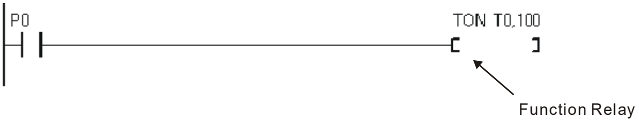
Input, Output Register Symbol

Input/Output registers are the most basic symbols among the registers in Ladder Logic.



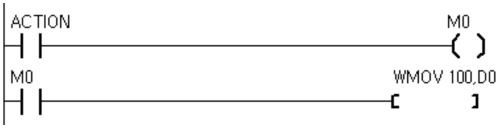
Function Registers

Function registers include timers, counters, and other mathematical operation registers.



Internal Registers

Internal registers (M) only operate within the program. Unless connected to an actual external port, they are only used internally. M registers can be used as input or output symbols.



P Registers Not Used as I/O Ports

Cubloc supports P registers from P0 to P127. P registers are directly connected to I/O ports one to one. However, most models of Cubloc have less than 128 I/O ports. Those P registers not connected to an I/O port can be used as internal registers (M registers).

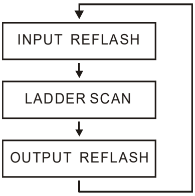
Using I/Os

Cubloc I/O ports can be used by both BASIC and Ladder Logic. Without defined settings, all I/O ports are controlled in BASIC. The `UsePin` command must be used to set the I/O ports for use in Ladder Logic.

```
UsePin 0,IN
UsePin 1,OUT
```

The above code sets P0 as input and P1 as output for use in Ladder Logic.

Those ports declared with the `UsePin` command will be re-flashed during a ladder scan. Re-flashing means that, prior to a ladder scan, those ports declared as IN with the `UsePin` command will have their input read and copied to the port's corresponding P register. After the ladder scan, those ports declared as OUT with the `UsePin` command will have the corresponding P register's value written to the port's output.



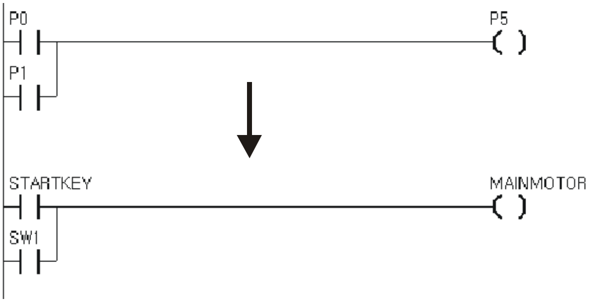
In BASIC, the commands `In` and `Out` an be used to control I/O ports. This method directly accesses the I/O ports when reading and writing. In order to avoid collisions between the two, I/O ports should be used in either BASIC or Ladder Logic, but not both. Once a port is declared with the `UsePin` command, it can only be used in Ladder Logic and cannot be directly accessed in BASIC, except through the ladder registers.

```
UsePin 0,IN, START
UsePin 1,OUT, RELAY
```

Aliases (such as START or RELAY as shown above) can be assigned using the `UsePin` command to improve readability.

Use of Aliases

When creating Ladder Logic using "Register numbers" such as P0, P1, and M0, aliases can be used to help simplify programs.



In order to use an alias, they must be declared in BASIC.

```
Alias M0 = MAINMOTOR
Alias M2 = STATUS1
Alias M4 = MOTORSTOP
```

Aliases can be declared using either the `UsePin` or the `Alias` commands.

Starting Ladder Logic

Cubloc executes BASIC first. To enable Ladder Logic, use the command `Set Ladder On`. After this command is executed, Ladder Logic will begin running and perform ladder scans every 10 milliseconds.

If the `Set Ladder On` command is not used, Ladder Logic will not run and no ladder scans will be performed

```
Set Ladder On
```

Declaring the Device to Use

The type of device being used must be declared in BASIC at the beginning of every program using the `Const Device` command. The following are examples illustrate this for the CB220 and the CB280.

```
Const Device = CB220        ' Use the CB220.
```

or

```
Const Device = CB280        ' Use the CB280.
```

Using Ladder Logic Only

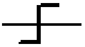
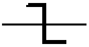
If only using Ladder Logic a minimal amount of BASIC code is required. The BASIC code must contain, at a minimum, a device declaration, the `Set Ladder On` command, and a `Do...Loop` to keep the program executing. Port declarations, and aliases must also be declared in BASIC.

The following example illustrates such a program.

```
Const Device = CB280        'Device Declaration

UsePin 0,In,START           'Port Declaration
UsePin 1,In,RESETKEY
UsePin 2,In,BKEY
UsePin 3,Out,MOTOR

Alias M0=RELAYSTATE         'Aliases
Alias M1=MAINSTATE

Set Ladder On               'Start Ladder

Do
Loop                        'BASIC program will run in infinite loop.
```

Ladder Logic Commands

Low-Level Commands

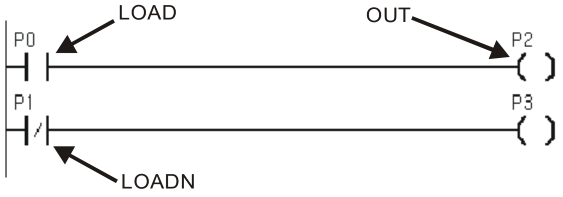
Command	Symbol	Description
LOAD	┤ ├	Contact A (Open by default)
LOADN	┤/├	Contact B (Closed by default)
OUT	—( )	Output
NOT	— / —	NOT (Invert the result)
STEPSET	-[ ]	Step Controller Output (Step Set)
STEPOUT	-[ ]	Step Controller Output (Step Out)
MCS	-[ ]	Master Control Start
MCSCLR	-[ ]	Master Control Stop
DIFU		Set ON for 1 scan time when a logic-high signal is received
DIFD		Set ON for 1 scan time when logic-low signal is received
SETOUT	-[ ]	Set and Hold Output ON
RSTOUT	-[ ]	Set and Hold Output OFF
END	-[ ]	End of Ladder Logic
GOTO	-[ ]	Jump to the Specified Label
LABEL	-[ ]	Label Declaration
CALLS	-[ ]	Call a Subroutine
SBRT		Declare a Subroutine
RET		End Subroutine
TND		Conditional Exit Command

High-Level Commands

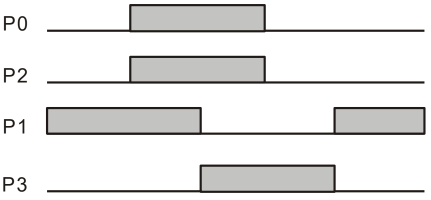
Command	Parameter	Description
Data Commands		
WMOV	s,d	Move Word
DWMOV	s,d	Move Double Word
WXCHG	s,d	Swap Word
DWXCHG	s,d	Swap Double Word
FMOV	s,d,n	Move Data Multiple Times (Fill Data)
GMOV	s,d,n	Move a Group of Data
Increment/Decrement Commands		
WINC	d	Increment a Word by 1
DWINC	d	Increment a Double Word by 1
WDEC	d	Decrement a Word by 1
DWDEC	d	Decrement a Double Word by 1
Math Commands		
WADD	s1,s2,d	Word Add
DWADD	s1,s2,d	Double Word Add
WSUB	s1,s2,d	Word Subtract
DWSUB	s1,s2,d	Double Word Subtract
WMUL	s1,s2,d	Word Multiplication
WDIV	s1,s2,d	Word Division
DWDIV	s1,s2,d	Double Word Division
Logical Commands		
WAND	s1,s2,d	Word AND
DWAND	s1,s2,d	Double Word AND
WOR	s1,s2,d	Word OR
DWOR	s1,s2,d	Double Word OR
WXOR	s1,s2,d	Word XOR
DWXOR	s1,s2,d	Double Word XOR
Bit Shift Commands		
WROL	d	Shift Word 1 bit to the Left
DWROL	d	Shift Double Word 1 bit to the Left
WROR	d	Shift Word 1 bit to the Right
DWROR	d	Shift Double Word 1 bit to the Right

LOAD, LOADN, OUT

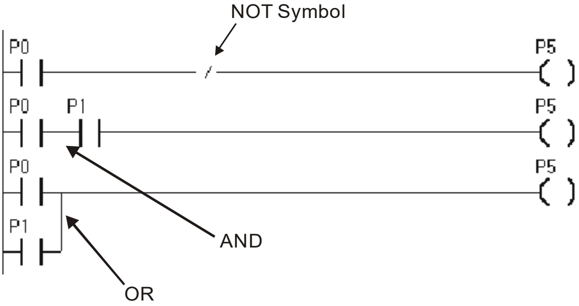
LOAD is for contacts that are open by default and LOADN is for contacts that are closed by default.



Registers that can be used	P	M	F	S	C	T	D	Constants
LOAD LOADN	O	O	O	O	O	O		
OUT	O	O						



NOT, AND, OR

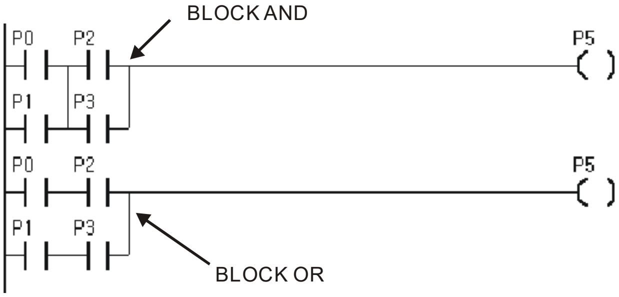


NOT inverts the results. On the first rung in the example above, if P0 is ON then P5 will be OFF.

AND is performed on two registers when they are placed horizontally beside each other on the same rung. On the second rung in the example above, both registers P0 and P1 must be ON in order for P5 to be ON.

An OR operation is performed when two registers are placed vertically beside each other on the same rung. On the third rung in the example above, when either P0 or P1 is ON, P5 will be ON.

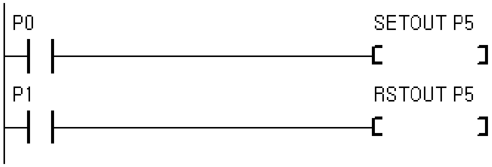
When two or more AND or OR operations are used on a single rung, as illustrated in the following example, it is called a BLOCK AND or a BLOCK OR.



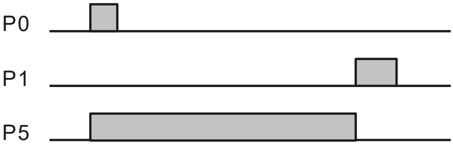
SETOUT, RSTOUT

SETOUT turns a register ON, and holds it on even if its input condition changes. Contrary to SETOUT, RSTOUT turns a register OFF and holds it OFF even if its input condition changes.

On the first rung in the example below, SETOUT will turn P5 ON when P0 turns ON, and will hold P5 ON even if P0 turns OFF. In the second rung of the example below, RSTOUT will turn P5 OFF when P1 is ON, and will hold P5 OFF even when P1 turns OFF.



Registers that can be used	P	M	F	S	C	T	D	Constants
SETOUT	O	O	O					
RSTOUT	O	O	O					

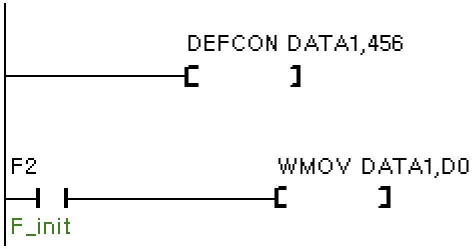


DEFCON

Cubloc Studio 3.3.1 and later

DEFCON *name, value*

Defines a named constant



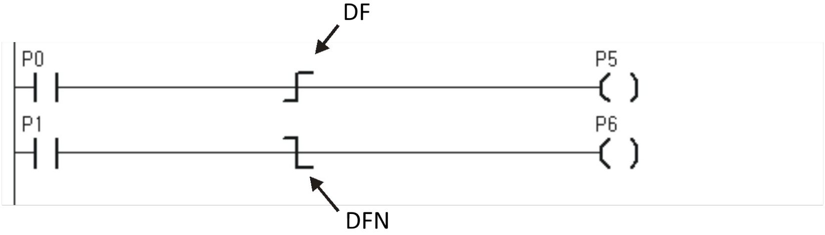
DEFCON provides a convenient way to reuse a value within a ladder diagram. If the value changes, only the constant's declaration, the DEFCON statement, would need to change.

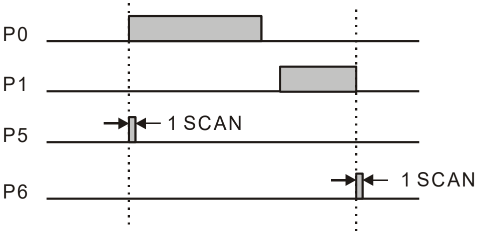
Please note you cannot use reserved words as constant names. For example DEFCON WMOV 100 is not legal since WMOV is a reserved command name.

DF, DFN

DF turns an output ON for the duration of one ladder scan when the input changes from OFF to ON.

Conversely, DFN turns an output ON for the duration of one ladder scan when the input changes from ON to OFF.



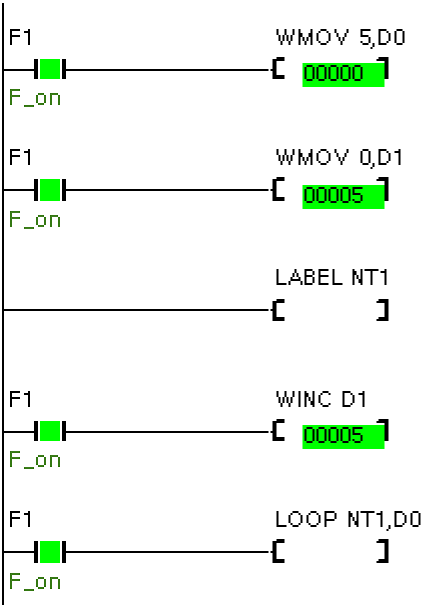


LOOP

Cubloc Studio 3.3.1 and later

LOOP *label, register*

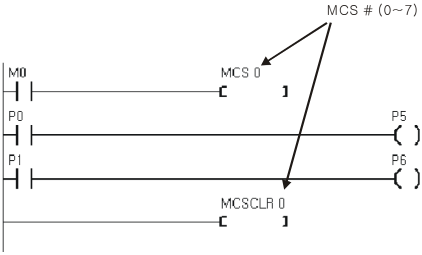
Repeatedly jumps to *label* until the value in register reaches 0. *register* is automatically decremented with each iteration.



In the example above, D0 is initialized to 5. This will be the number of times to repeat the loop. D1 is initialized to 0. This is the number we wish to change. D1 is incremented with each iteration and D0 is decremented with each iteration. Therefore, when D0 reaches 0, LOOP no longer jumps to NT1 and D1 will contain the value 5.

MCS, MCSCLR

MCS and MCSCLR are used to conditionally execute rungs in a ladder. Rungs between MCS x and MCSCLR x are executed only when the input to MCS x is ON. If the input to MCS x is OFF, the rungs between MCS x and MCSCLR x will not be executed. Using this command, an block of Ladder Logic can be conditionally executed.



In the example above, when M0 turns ON, the rungs between MCS 0 and MCSCLR are executed normally. If M0 is OFF, P5 and P6 will not be processed.

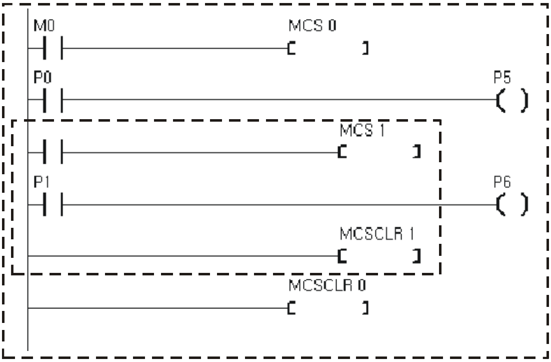
MCS numbers can be assigned from 0 to 7. MCS numbers should be used incrementally from 0 to 7. MCS 1 must be nested inside MCS 0 and MCS 2 must be nested inside MCS 1. When MCS 0 is OFF, none of the MCS's nested inside of MCS 0 will be executed.

When the input to an MCS is OFF, all outputs within that MCS block will turn OFF, timers withing the MCS block will be reset, and counters within the MCS block will stop counting.

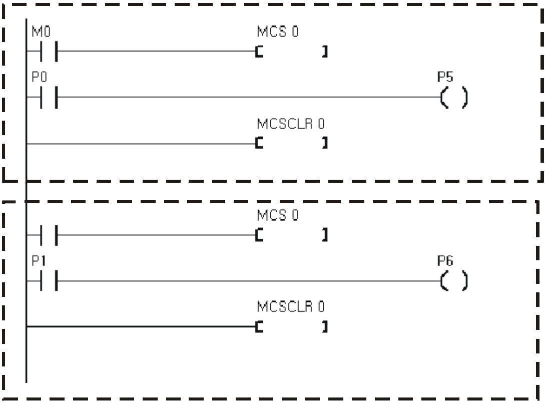
The following table describes the behavior of various commands nested within an MCS block

Command	When MCS is ON	When MCS is OFF
OUT	Normal Operation	OFF
SETOUT	Normal Operation	State is retained
RSTOUT	Normal Operation	State is retained
Timer	Normal Operation	Reset to default value
Counter	Normal Operation	State is retained, but counting is stopped
Other Commands	Normal Operation	Not executed

The following example shows MCS 1 nested within MCS 0.

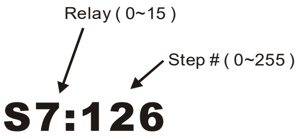


MCS numbers can be used repeatedly, but cannot be nested within one another.



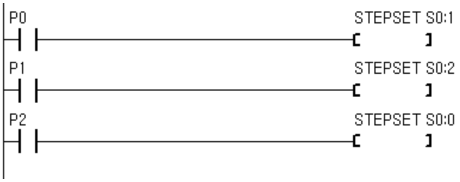
Step Control

S Registers are used for step control. Step control arguments must follow the following format.

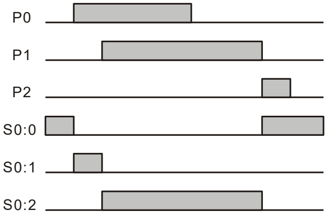


Step control can be performed in two ways: "Normal Step" and "Reverse Step". STEPSET is used for "Normal Step".

STEPSET

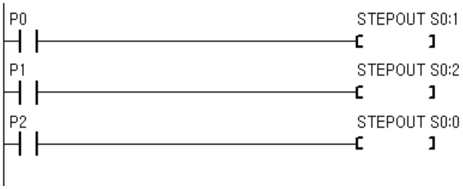


STEPSET turns ON the current step, but only if the previous step is ON. STEPSET is processed one step at a time, in sequence, according to the step number. For example, in the ladder above, when P1 turns ON, S0:2 is turned ON, but only if S0:1 is ON. After S0:2 turns ON, S0:1 is turns OFF. When P2 turns ON, S0:0 is turned ON and all other steps are turned OFF. S0:0, or step 0, is used to reset. All other steps are processed in order.

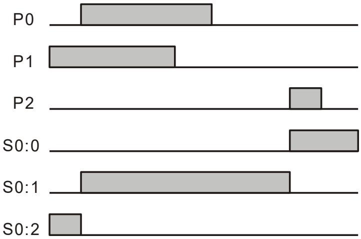


STEPOUT

STEPOUT differs from STEPSET in two ways. First, it is not evaluated in sequence, and second, when a step is turned ON, it remains in that state regardless of its input condition. When using STEP OUT and a step is turned ON, all other steps are turned OFF, so only one step is ON at any given time.



In the ladder above, when P1 turns ON, S0:2 will turn ON. When P0 turns on S0:1 turns ON, and S0:2 turns OFF and S0:1 turns ON. When PO turns OFF, S0:1 is unchanged as each step, when turned ON, remains ON regardless of its input condition. Finally when P2 turns ON, S0:0 turns ON and S0:1 turns OFF.



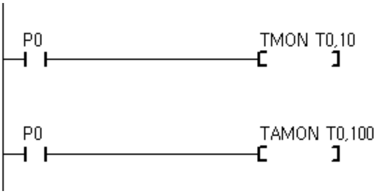
TMON, TAMON

Cubloc Studio 3.3.1 and later

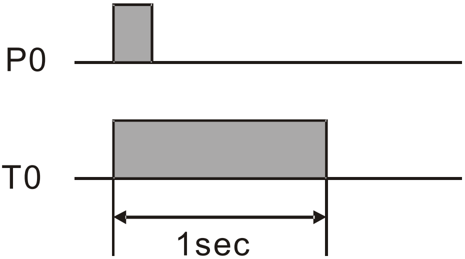
TMON *arg*, *timeout*
TAMON *arg*, *timeout*

Implements a timer that turns *arg* ON when the input turns ON, and remains ON until *timeout* elapses. After *timeout* elapses, *arg* will remain on regardless of the input.

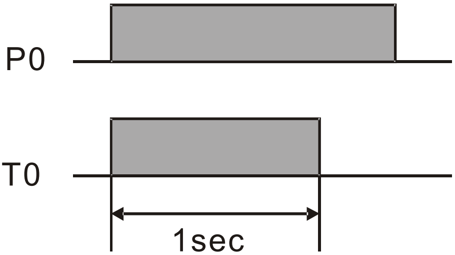
Type of Timer	Time units	Maximum Time
TMON	0.01 sec	655.35 sec
TAMON	0.1 sec	6553.5 sec



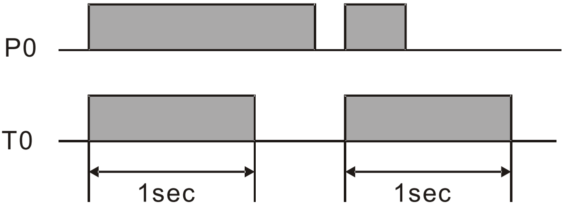
Usable Registers	P	M	F	S	C	T	D	Y	A	B	Constants
arg					O	O	O	O			O



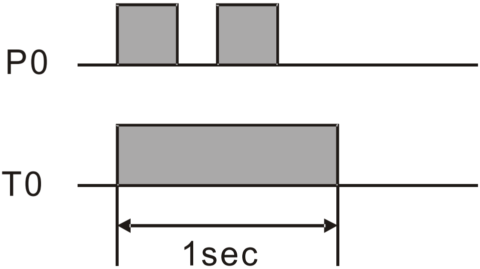
If the input (P0) turns OFF, the output (T0) still remains ON.



After the timeout elapses, the output (T0) will turn OFF even if the input (P0) remains ON.



If the input (P0) turns ON after the timeout has elapsed, the output (T0) will turn ON again.

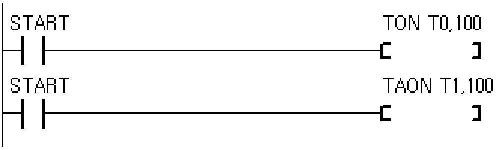


If the input (P0) turns OFF and then ON before the timeout has elapsed, the output (T0) will still remain ON only for the specified timeout.

TON, TAON

TON and TAON provide the capability to delay the effect of an input changing from OFF to ON for a specified amount of time. When the input turns ON, a timer begins incrementing from its initial value to the specified value. When the specified value is reached, the output is turned ON. TON increments every 0.01 seconds and TAON increments every 0.1 seconds.

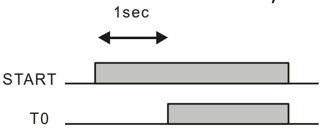
Type of Timer	Time units	Maximum Time
TON	0.01 sec	655.35 sec
TAON	0.1 sec	6553.5 sec



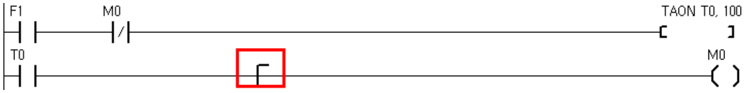
For TON and TAON there are 2 parameters. The first parameter can be any value from T0 through T99, and the second parameter can be any numeric constant or data register, such as D0.

Usable Registers	P	M	F	S	C	T	D	Constants
TON, TAON					O	O	O	O

In the ladder above, when START turns ON, timer T0 increases from 0 to 100 in increments of 0.01 seconds. When 100 is reached, T0 will turn ON. Here, 100 is equal to a time of 1 second for TON, and 10 seconds for TAON.



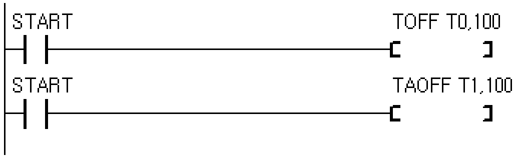
When START turns OFF, the timer is reset to its initial value and T0 will turn OFF. TON and TAON will reset their values upon powering OFF. Using KTON and KTAON, the battery backup feature can be used to retain the timer's values between power cycles. The example below illustrates how to reset TAON.



TOFF, TAOFF

TOFF and TAOFF provide the capability to delay the effect of an input changing from ON to OFF for a specified amount of time. When input turns ON, the output turns ON immediately, but when the input turns OFF, the output is kept ON for the set amount of time, and then turned OFF. TOFF increases in increments of 0.01 seconds and TAOFF increases in increments 0.1 seconds.

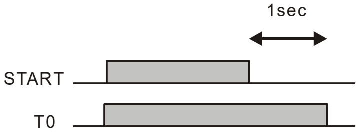
Type of Timer	Time units	Maximum Time
TOFF	0.01 sec	655.35 sec
TAOFF	0.1 sec	6553.5 sec



For TON and TAON there are 2 parameters. The first parameter can be any value from T0 through T99, and the second parameter can be any numeric constant or data register, such as D0.

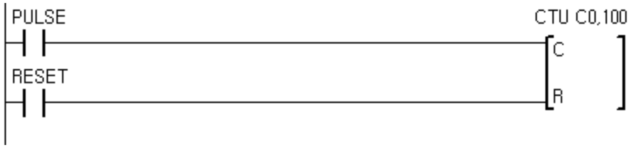
Usable Registers	P	M	F	S	C	T	D	Constants
TOFF, TAOFF					O	O	O	O

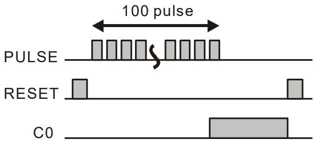
In the ladder above, when START turns ON, the timer T0 immediately turns ON. When START turns OFF, the timer will start decreasing from 100 to 0. When 0 is reached, T0 will turn OFF. Here, 100 is equal to a time of 1 second for TOFF, and 10 seconds for TAOFF.



CTU

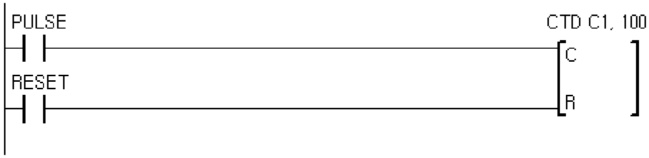
This CTU command is an UP counter. When an input is received the counter is incremented by one. When the counter reaches a specified value, the specified register will turn ON There is also a reset input so the counter can be reset as needed.

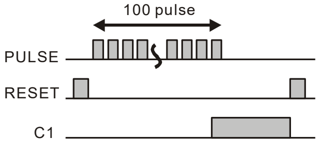




CTD

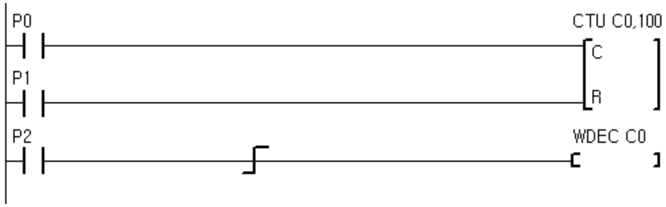
This CTD command is a DOWN counter. When an input is received the counter is decremented by one. When the counter reaches 0, the specified register will turn ON. There is a also a reset input so the counter can be reset as needed.



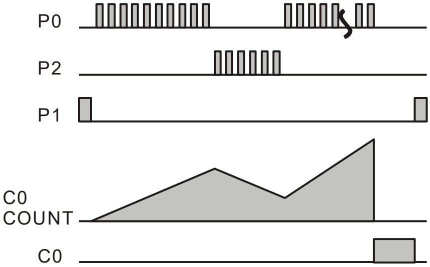


UP/DOWN COUNTER

Below is a simple example of how an UP counter can be used to make an UP/DOWN Counter.

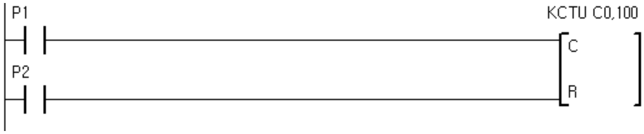


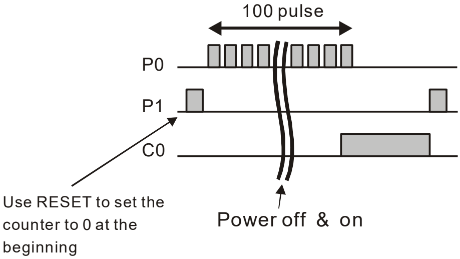
P0 is for counting UP, P2 is for counting DOWN, and P1 is for resetting the counter. When the counter reaches 100, C0 turns ON.



KCTU

This command is exactly same as the CTU command, except this command will be able to remember its count when the module is powered off. By comparison, the CTU command will lose its count when the module is powered off. This command can only be used with modules that support battery backup such as the CB290, CB405, and CuTouch.





When using this command for the very first time, use the RESET signal to initialize the counter. Otherwise the counter could start at some random value.

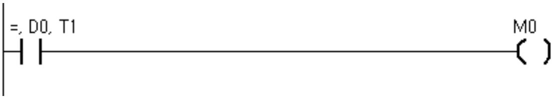
KCTD

This command is exactly same as the CTD command, except this command will be able to remember its count when the module is powered off. By comparison, the CTD command will lose its count when the module is powered off. This command can only be used with modules that support battery backup such as the CB290, CB405, and CuTouch.

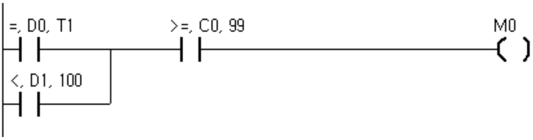
Comparison Logic

Compare 2 word(16 bit) or 2 double word(32 bit) values and turn on an output when the conditions are satisfied.

Comparison Command	Data Types	Explanation
=, s1, s2	Word(16 bit)	Turns on when s1 and s2 are equal.
<>, s1, s2	Word(16 bit)	Turns on when s1 and s2 are not equal.
>, s1, s2	Word(16 bit)	Turns on when s1 > s2.
<, s1, s2	Word(16 bit)	Turns on when s1 < s2.
>=, s1, s2	Word(16 bit)	Turns on when s1 >= s2.
<=, s1, s2	Word(16 bit)	Turns on when s1 <= s2.
D=, s1, s2	DWord(32 bit)	Turns on when s1 and s2 are equal.
D<>, s1, s2	DWord(32 bit)	Turns on when s1 and s2 are not equal.
D>, s1, s2	DWord(32 bit)	Turns on when s1 > s2.
D<, s1, s2	DWord(32 bit)	Turns on when s1 < s2.
D>=, s1, s2	DWord(32 bit)	Turns on when s1 >= s2.
D<=, s1, s2	DWord(32 bit)	Turns on when s1 <= s2.



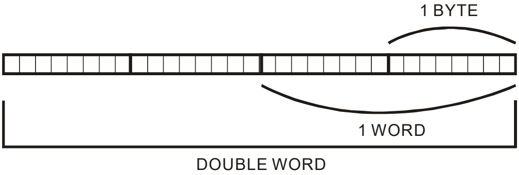
You can mix comparison logic as shown below:



When either D0=T1 or D1<100 and if C0>=99, M0 will turn ON. In other words, either D0 has to equal to value of T1 or D1 has to be less than 100 while C0 must be greater than or equal to 99.

Storing Words and Double Words

A byte is 8 bits, a word is 16 bits, and a double word is 32 bits.



Memory is divided into bytes, so words and double words must be split across multiple bytes. Therefore, there are two ways to store words and double words: Least significant byte first (Little Endian) or most significant byte first (Big Endian). Cubloc stores its data in Little Endian.

In the memory map to the right, 1234H is stored in memory address 0 and 12345678H is stored in memory address 5. Note how 34H (the least significant byte of 1234H is stored first in memory address 0, and 78H, the least significant byte of 12345678H is stored first in memory address 5.

0	34
1	12
2	
3	
4	
5	78
6	56
7	34
8	12
9	

The Registers C, T, D are in word units. To store double words data, 2 words are required, therefore 2 registers are required. Below is an example of storing a double word, 12345678H. D1 gets 1234H and D0 gets 5678H.

D0	5678
D1	1234
D2	
D3	
D4	

Binary, Decimal, Hexadecimal

To program well, we need to know binary, decimal, and hexadecimal numbers. The following chart shows the relationships between these notations.

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

In Cubloc's Ladder Logic, we express binary and hexadecimal numbers in the following manner:

Binary:	00101010B
Hexadecimal:	0ABCDH

We put a B at the end of the binary number and an H at the end for hexadecimal numbers. To clearly identify that a hexadecimal value is a number, we can put a 0 in front of the hexadecimal number.
(E.g. : 0ABH, 0A1H, 0BCDH)

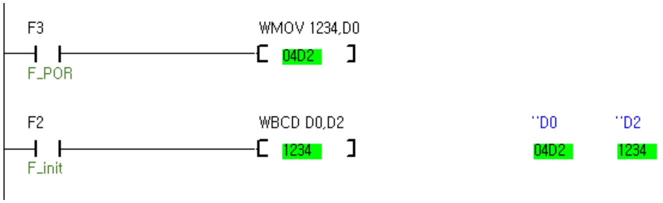
*BASIC is slightly different from Ladder Logic in the way binary and hexadecimal numbers are expressed. &B100010 is used for binary and &HAB is used for hexadecimal.

WBCD

Cubloc Studio 3.3.1 and later

WBCD s, d

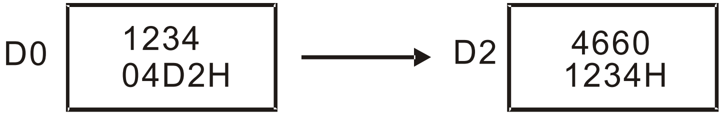
Converts the 16 bit binary value in `s` to a BCD (Binary Coded Decimal) value ans stores the result in `d`.



```
F3                        WMOV 1234,D0
─┤ ├──────────────────────[ 04D2 ]
F_POR

F2                        WBCD D0,D2            ''D0    ''D2
─┤ ├──────────────────────[ 1234 ]              04D2    1234
F_init
```

Values shown are in hexadecimal

Usable Registers	P	M	F	S	C	T	D	Y	A	B	Constants
s, d					O	O	O	O		O	

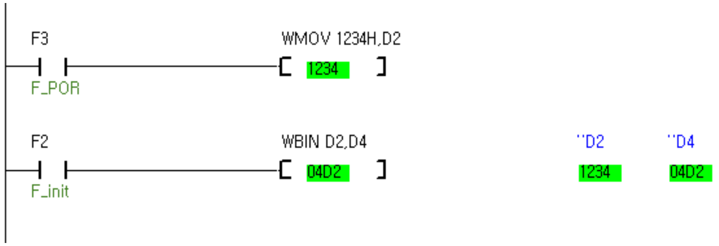


WBIN

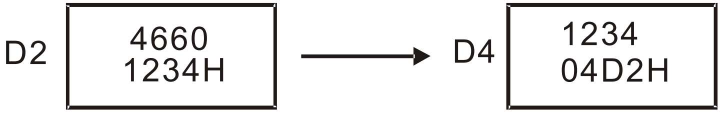
Cubloc Studio 3.3.1 and later

WBIN s, d

Converts the 16 bit BCD value in `s` to binary and stores the result in `d`. Values shown are in hexadecimal.



Usable Register	P	M	F	S	C	T	D	Constants
s, d					O	O	O	

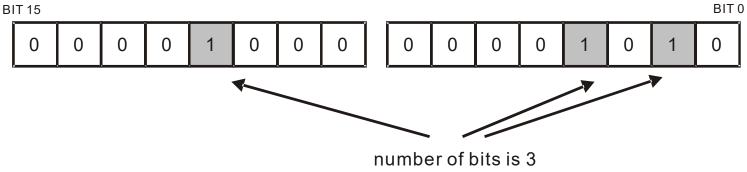


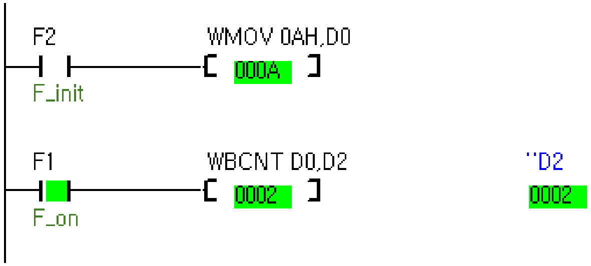
WBCNT, DWBCNT

Cubloc Studio 3.3.1 and later

WBCNT s, d
DWBCNT s, d

Counts the number of active bits (1s) in `s` in stores the result in `d`. WBCNT operates words (16 bits) and DWBCNT operates on double words (32 bits).





```
F2          WMOV 0AH,D0
            [ 000A ]
F_init

F1          WBCNT D0,D2                 ''D2
            [ 0002 ]                    0002
F_on
```

Values shown are in hexadecimal

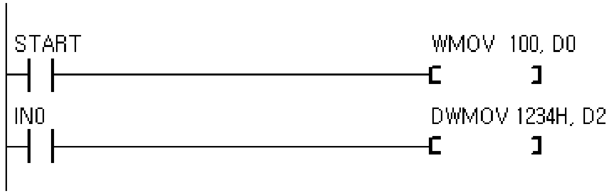
Usable Register	P	M	F	S	C	T	D	Constants
s					O	O	O	
d							O	

WMOV, DWMOV

WMOV s, d
DWMOV s, d

The WMOV command moves 16 bit data from `s` to `d`. DWMOV is used for 32 bit data.

Usable Register	P	M	F	S	C	T	D	Constants
s (Source)					O	O	O	O
d (Destination)					O	O	O	



When the input START turns ON, D0 will get 100. When IN0 turns ON, D2 will get 1234H.

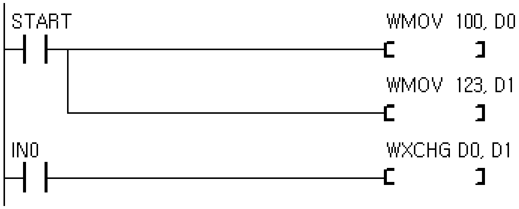
D0	100
D1	
D2	1234H
D3	0
D4	

WXCHG, DWXCHG

WXCHG s, d
DWXCHG s, d

These commands swap data between `s` and `d`. WXCHG is for swapping word values and DWXCHG is swapping double word values.

Usable Registers	P	M	F	S	C	T	D	Constants
s					O	O	O	
d					O	O	O	



When START turns ON, D0 gets 100 and D1 gets 123. When IN0 turns ON, D0 and D1 swap their data. The result is as shown below:

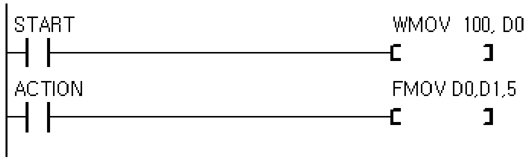
D0	123
D1	100
D2	
D3	
D4	

FMOV

FMOV s, d, n

This command stores `s` in `d`, `n` number of times, in subsequent memory locations. This command is usually used for initializing or clearing memory.

Usable Registers	P	M	F	S	C	T	D	Constants
s					O	O	O	
d					O	O	O	
n								O



The result of the ladder above is shown below:

D0	100
D1	100
D2	100
D3	100
D4	100
D5	100

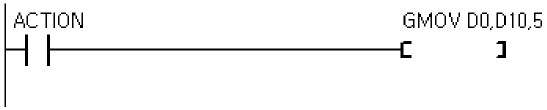
*NOTE: `n` must be less than 255.

GMOV

GMOV s, d, n

This command takes n values, starting from s, and copies them to d. To avoid collisions, source and destination memory should not overlap.

Usable Registers	P	M	F	S	C	T	D	Constants
s					O	O	O	
d					O	O	O	
n								O



The result of the ladder above is shown below:

D0	12
D1	34
D2	56
D3	78
D4	90
D5	
D6	
D7	
D8	
D9	
D10	12
D11	34
D12	56
D13	78
D14	90
D15	
D16	

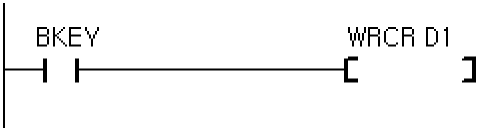
*NOTE: n must be less than 255.

WCMP, DWCMP

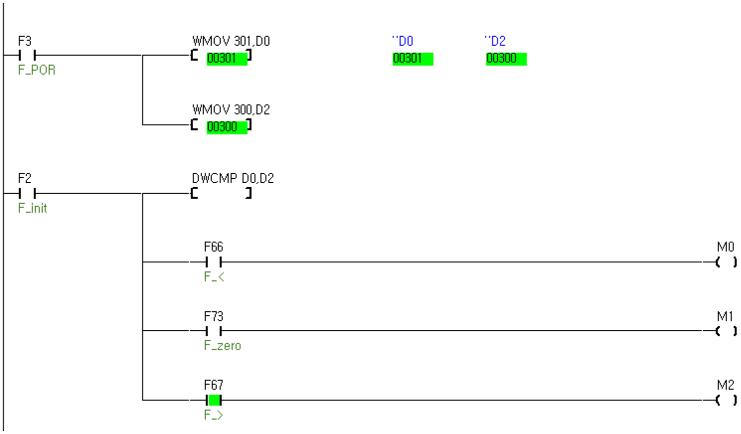
Cubloc Studio 3.3.1 and later

WCMP arg1, arg2
DWCMP arg1, arg2

Compares `arg1` and `arg2`. WCMP operates on words (16 bits) and DWCMP operates on double words (32 bits).



Usable Registers	P	M	F	S	C	T	D	Constants
arg1, arg2					O	O	O	O



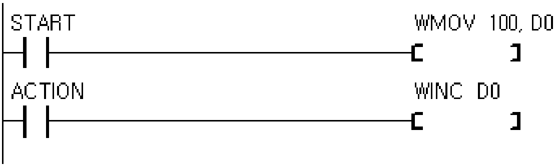
Flags:
F73 (zero flag) - ON if s1 and s2 are equal
F66 (less than flag) - ON if s1 is less than s2
F67 (greater than flag) - ON if s1 is greater than s2

WINC, DWINC, WDEC, DWDEC

WINC d
DWINC d
WDEC d
DWDEC d

WINC increments the word value in `d` by one.
DWINC increments the double word value in `d` by one.
WDEC decrements the word value in `d` by one.
DWDEC decrements the double word value in `d` by one.

Usable Registers	P	M	F	S	C	T	D	Constants
d					O	O	O	



The result of the ladder above is shown below:

D0	99
D1	
D2	
D3	

WADD, DWADD

WADD s1, s2, d
DWADD s1, s2, d

These commands adds `s1` and `s2` and store the result in `d`. WADD adds word values and DWADD adds double word values.

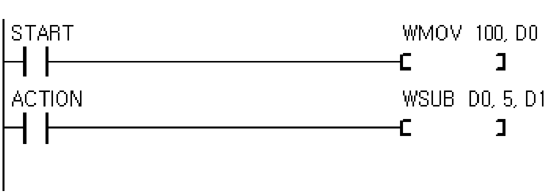
Usable Registers	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
d					O	O	O	

WSUB, DWSUB

WSUB s1, s2, d
DWSUB s1, s2, d

These commands subtracts `s2` from `s1` and store the result in `d`. WADD subtracts word values and DWADD subtracts double word values.

Usable Registers	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
d					O	O	O	



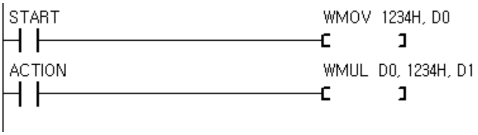
In the ladder above, D1 will get 95.

WMUL

WMUL s1, s2, d

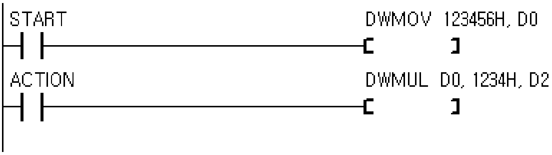
These commands multiply `s1` and `s2` and store the result in `d`. WMUL multiplies word values.

Usable Registers	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
d					O	O	O	



In the ladder above, the result of 1234H * 1234H is stored as the double word 14B5A90H in D1.

D0	1234H
D1	5A90H
D2	14BH



In the ladder above, the result of 123456H * 1234H is stored as 4B60AD78H in D2.

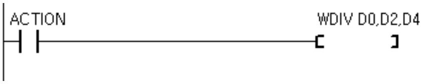
D0	3456H
D1	0012H
D2	0AD78H
D3	4B60H
D4	0
D5	0

WDIV, DWDIV

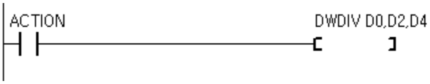
WDIV s1, s2, d
DWDIV s1, s2, d

These commands divide `s1` and `s2` and store the result in `d` and the remainder in `d`+1. WDIV divides word values and DWDIV divides double word values.

Usable Registers	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
d					O	O	O	



D0	1234H
D1	
D2	3
D3	
D4	611H
D5	1



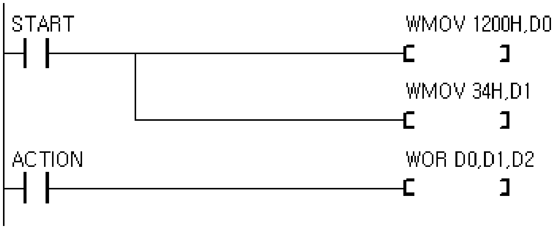
D0	5678H
D1	1234H
D2	7
D3	0
D4	0C335H
D5	299H
D6	5
D7	0

WOR, DWOR

WOR s1, s2, d
DWOR s1, s2, d

These commands logically OR `s1` and `s2` and store the result in `d`. WOR divides word values and DWOR divides double word values.

Usable Registers	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
d					O	O	O	



The result of above ladder diagram:

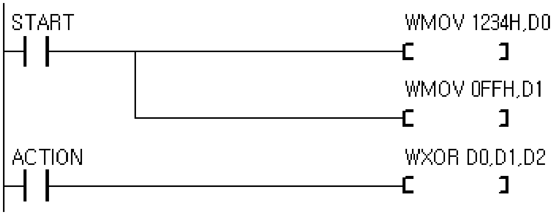
D0	1200H
D1	34H
D2	1234H

WXOR, DWXOR

WXOR s1, s2, d
DWXOR s1, s2, d

These commands logically XOR s1 and s2 and store the result in d. WXOR divides word values and DWXOR divides double word values.

Usable Registers	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
d					O	O	O	



The following is result of above LADDER:

D0	1234H
D1	0FFH
D2	12CBH

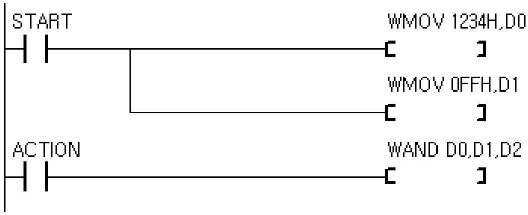
When you want to invert specific bits, you can use XOR logical operation.

WAND, DWAND

WAND s1, s2, d
DWAND s1, s2, d

These commands logically AND s1 and s2 and store the result in d. WAND divides word values and DWAND divides double word values.

Registers that may be used	P	M	F	S	C	T	D	Constants
s1					O	O	O	O
s2					O	O	O	O
D					O	O	O	



The results of execution of LADDER above:

D0	1234H
D1	0FFH
D2	34H

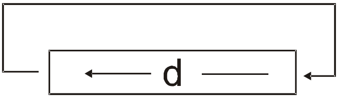
You can use AND operation when you want to use specific bits only.

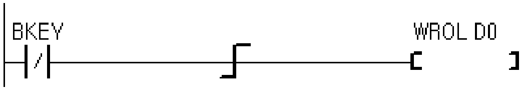
WROL, DWROL

WROL d
DWROL d

This commands rotates the bits in d 1 bit to the left (least significant bit to most significant bit). Bits on the far left are appended on the far right. WROL operates on word values and DWROL operates on double word values.

Registers that may be used	P	M	F	S	C	T	D	Constants
D					O	O	O	





If D0 has 8421H, the following results:

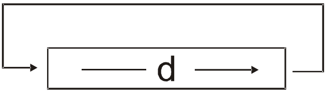
D0	0843H
D1	

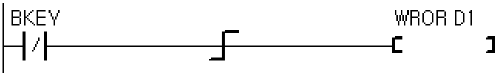
WROR, DWROR

WROR d
DWROR d

This commands rotates the bits in d 1 bit to the right (most significant bit to least significant bit). Bits on the far right are appended on the far left. WROR operates on word values and DWROR operates on double word values.

Registers that may be used	P	M	F	S	C	T	D	Constants
d					O	O	O	





If D1 has 8421H, the following results:

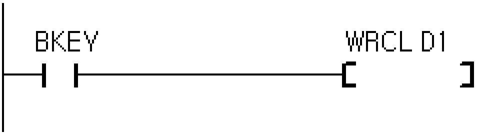
D0	
D1	0C210H

WRCL, DWRCL

Cubloc Studio 3.3.1 and later

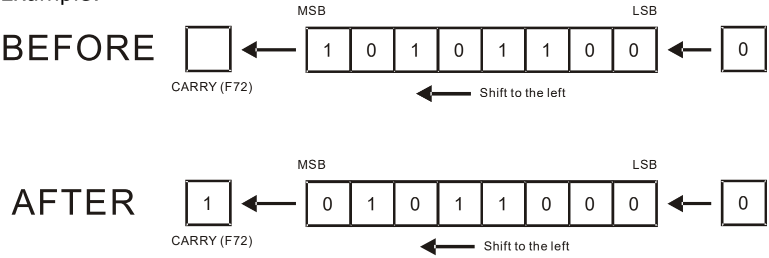
WRCL *arg*
DWRCL *arg*

Shifts *arg* one bit to the left and fills the carry bit (F72). WRCL operates on words (16 bits) and DWRCL operates on double words (32 bits).



Usable Registers	P	M	F	S	C	T	D	Constants
arg					O	O	O	O

Example:



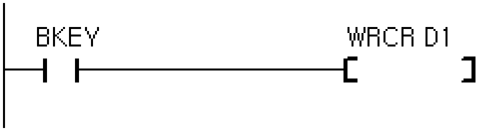
The most significant bit is moved to the carry flag, while the least significant bit is populated with a 0.

WRCR, DWRCR

Cubloc Studio 3.3.1 and later

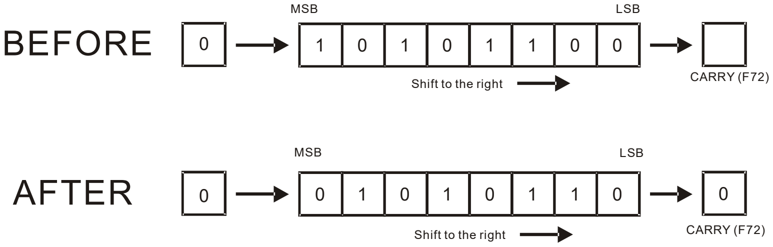
WRCR arg
DWRCR arg

Shifts `arg` one bit to the right and fills the carry bit (F72). WRCR operates on words (16 bits) and DWRCR operates on double words (32 bits).



Usable Registers	P	M	F	S	C	T	D	Constants
arg					O	O	O	O

Example:



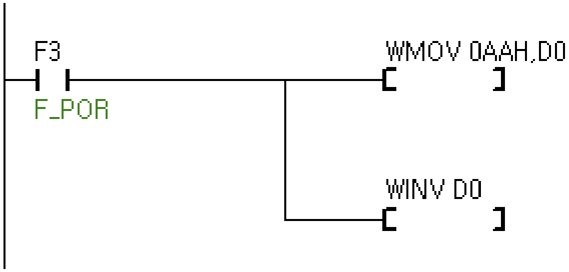
The least significant bit is moved to the carry flag, while the most significant bit is populated with a 0.

WINV, DWINV

Cubloc Studio 3.3.1 and later

WINV arg
DWINV arg

Toggles, inverts, each bit in `arg` making each 1 a 0, and each 0 a 1. WINV operates words (16 bits) and DWINV operates on double word (32 bits).



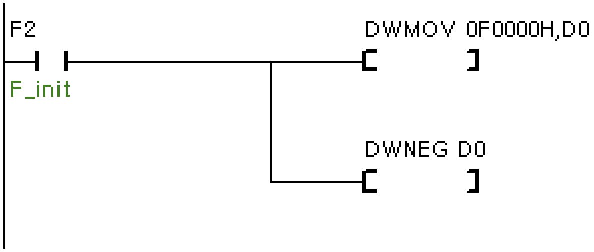
Usable Registers	P	M	F	S	C	T	D	Constants
arg					O	O	O	O

WNEG, DWNEG

Cubloc Studio 3.3.1 and later

WNEG arg
DWNEG arg

Toggles the sign (+/-) of `arg`. WNEG operates words (16 bits) and DWNEG operates on double word (32 bits).

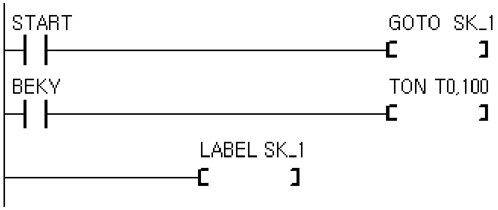


Usable Registers	P	M	F	S	C	T	D	Constants
arg					O	O	O	O

GOTO, LABEL

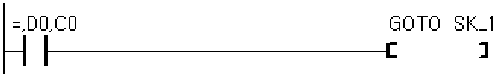
GOTO label
LABEL label

The GOTO command causes a jump to `label`. The LABEL command is used to name a position in the program to jump to.



In the ladder above, when START turns ON, the program will jump to label SK_1

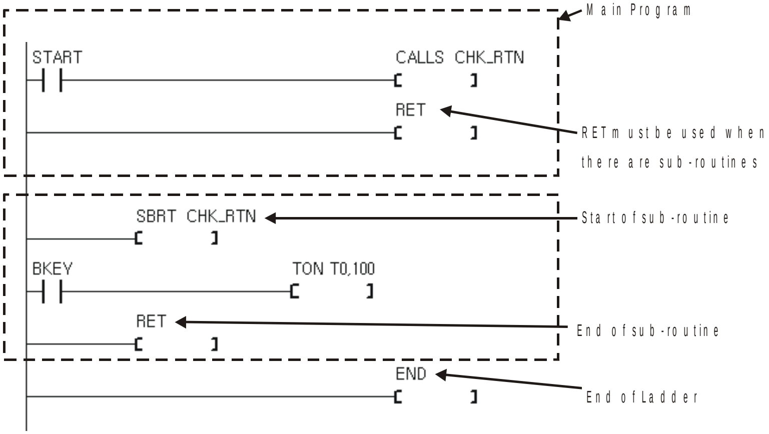
In the ladder below, when D0 equals C0, the program will jump to SK_1.



CALLS, SBRT, RET

CALLS label
SBRT label

The CALLS command calls a subroutine with the specified label.
SBRT marks the subroutine's starting point and RET marks the subroutine' end.



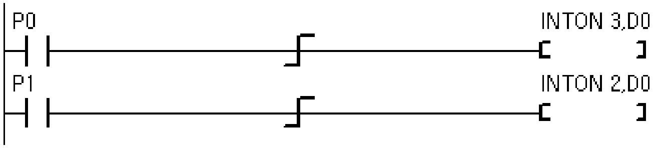
Please be aware that when adding sub-routines to your program, RET must be appended to the end of main program, before the first subroutine. END must be placed at the very end of the main program after the last subroutine.

INTON

INTON s,d

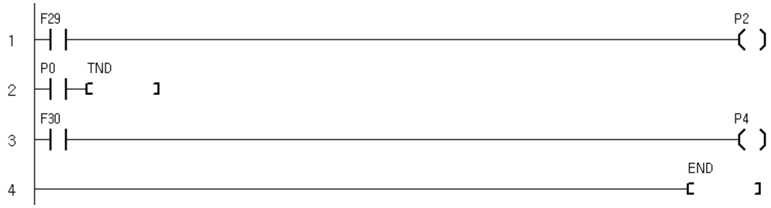
INTON is the same as the WMOV command except it triggers an interrupt in BASIC.

Usually egisters	P	M	F	S	C	T	D	Constants
s (Source)					O	O	O	O
d (Destination)					O	O	O	

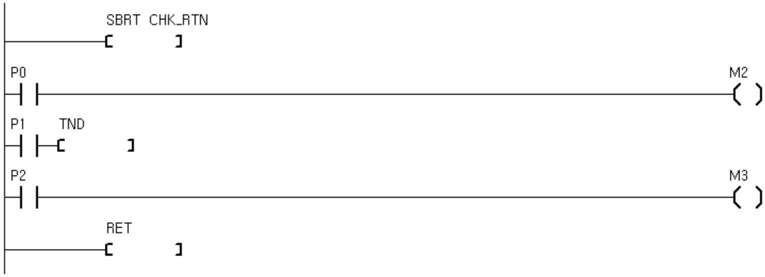


TND

TND is a conditional exit command. TND is typically used to abort a ladder scan.



In the ladder above, when P0 turns ON the ladder scan will be aborted.



It can also be used to exit from subroutines when a certain condition is met. In the ladder above, when P1 turns ON, the subroutine will be aborted, but the ladder scan will continue.

Special Registers

Cubloc contains a set of special registers that can be used to obtain the Cubloc's current status or for timing functions and applications.

Special Register	Explanation
F0	Always OFF
F1	Always ON
F2	Turn on 1 SCAN time at POWER UP (Set Ladder On).
F3	Reserved
F4	Reserved
F5	Reserved
F6	Reserved
F7	Reserved
F8	1 SCAN every 10ms
F9	1 SCAN every 100ms
F10	Reserved
F11	Reserved
F12	Reserved
F13	Reserved
F14	Reserved
F15	Reserved
F16	Repeat ON/OFF every 1 Scan time. Period 2 Scan times.
F17	Repeat ON/OFF every 2 Scan times. Period 4 Scan times.
F18	Repeat ON/OFF every 4 Scan times. Period 8 Scan times.
F19	Repeat ON/OFF every 8 Scan times. Period 16 Scan times.
F20	Repeat ON/OFF every 16 Scan times. Period 32 Scan times.
F21	Repeat ON/OFF every 32 Scan times. Period 64 Scan times.
F22	Repeat ON/OFF every 64 Scan times. Period 128 Scan times.
F23	Repeat ON/OFF every 128 Scan times. Period 256 Scan times.
F24	Repeat ON/OFF every 10ms (20ms period)
F25	Repeat ON/OFF every 20ms (40ms period)
F26	Repeat ON/OFF every 40ms (80ms period)
F27	Repeat ON/OFF every 80ms (160ms period)
F28	Repeat ON/OFF every 160ms (320ms period)
F29	Repeat ON/OFF every 320ms (640ms period)
F30	Repeat ON/OFF every 640ms (1.280s period)
F31	Repeat ON/OFF every 2.56 seconds (5.12s period)
F32	Repeat ON/OFF every 5.12 seconds (10.24s period)
F33	Repeat ON/OFF every 10.24 seconds (20.48s period)
F34	Repeat ON/OFF every 20.48 seconds (40.96s period)
F35	Repeat ON/OFF every 40.96 seconds (81.92s period)
F36	Repeat ON/OFF every 81.92 seconds (163.84s period)
F37	Repeat ON/OFF every 163.84 seconds (327.68s period)
F38	Repeat ON/OFF every 327.68 seconds (655.36s period)
F39	Repeat ON/OFF every 655.36 seconds (1310.72s period)
F40	Call LADDERINT in BASIC
F41	Reserved

Special Register	Explanation
F42	Reserved
F56	StepPulse Channel 0 Status
F57	StepPulse Channel 1 Status
F65 (F_ERR)	Error
F66 (F_<)	< Result of WCMP, DWCMP
F67 (F_>)	> Result of WCMP, DWCMP
F72 (F_CARRY)	Carry Flag
F73 (F_ZERO)	Zero Flag

* Turning F40 ON will create a `LadderInt` in BASIC. Please refer to the `On LadderInt GoSub` command for details.

* Turning F2 ON causes one ladder scan immediately when the `Set Ladder On` command is executed in BASIC.

* Reserved registers should not be used.

MEMO

Chapter 12: Cutouch

Preface

The Cutouch is a fully integrated graphical touchscreen device containing a Cubloc embedded computer. In recent years, touchscreens have found increasing use in the field of industrial automation. However, most touchscreen devices require connection to an external PLC, and require learning yet another complex interface description method. In addition, the cost of touchscreen interfaces has remained relatively high.

The Cutouch is a complete touchscreen controller featuring both a graphical user interface and direct I/O, reducing complexity and cost.

The embedded BASIC language can be used to draw graphics and print characters to the LCD, and process touchscreen coordinates. BASIC makes it easy to interface to various types of sensors, read analog values, process text, perform mathematical operations, and perform custom RS-232 communication; tasks that are difficult to accomplish with a traditional PLC. Ladder Logic is also available and can execute alongside BASIC to perform sequential processing and real-time logic as done traditional PLCs.

The Cutouch has reprogrammable flash memory for BASIC and Ladder Logic programs. An RS-232 serial port is used to download and debug code using a Windows PC, but once disconnected from the PC, the Cutouch will operate as a stand-alone device.

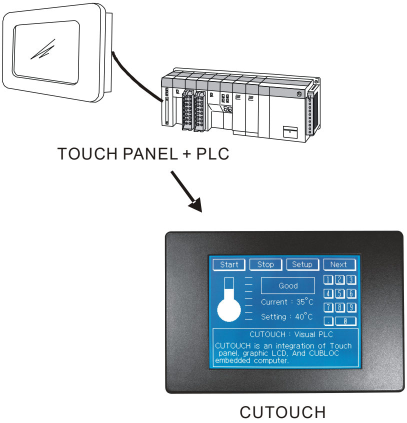
If you are thinking about developing a device that uses a touchscreen, consider the Cutouch. The integrated approach saves time, and lets you concentrate on solutions instead of problems.

Comfile Technology Inc.

What is Cutouch?

The Cutouch is quite different from traditional touchscreens. Traditional touchscreens are not a complete integrated product for solution development. They are usually only capable of displaying graphics and capturing touch input. Most touchscreens require an external controller in order to affect the real world through I/O.

The Cutouch combines a traditional PLC with a touchscreen graphical LCD. By integrating user input, display output, and control, developers can now use one device as a complete control system.



CUTOUCH is an integration of Touch panel, graphic LCD. And CUBLOC embedded computer.

Cutouch Specifications

Processor	Cutouch CT1721
Microprocessor	Dual Core Atmega128 @ 18.432Mhz
Program Memory (Flash)	80KB
Data Memory (RAM)	24KB(BASIC)+4KB(Ladder Logic)
EEPROM	4KB EEPROM
Program Speed	36,000/sec
General Purpose I/O	- 82 I/O lines (TTL & 24V DC)
Serial Ports for Communication	- 2 High-speed hardware-independent serial ports - Configurable Baud rates: 2400bps to 230,400 bps
Analog Inputs	8 channel 10-bit ADCs Configurable Voltage: **0 to 5V** OR **0 to 10V**
Analog Outputs	- 6 Channel 16-bit PWMs (DACs) - Output Voltage Range: **0 to 5V** - Configurable Frequencies: **35hz to 1.5Mhz**
External Interrupts	4 Channels
High Speed Counters	2 Channel 16-bit Counters
Power	- Required Power: **24V DC** - Current Consumption w/ ports unloaded: @ 24V w/ Backlight ON: 170mA @ 24V w/ Backlight OFF: 70mA @ 12V w/ Backlight ON: 340mA 12V w/ Backlight OFF: 130mA
RTC (Real Time Clock)	Yes
Timers	– 1 User Configurable Timer - Configurable Interval Units = 10ms
RTC Back-up	*Yes, a 1 Farad rechargeable Super-Capacitor **is included**.
Data Memory Back-up	No
Operating Temperature	0 °C to 70 °C
Package	Integrated Touch-screen Panel w/ 2mm Headers and 2.5mm RCABLE Headers

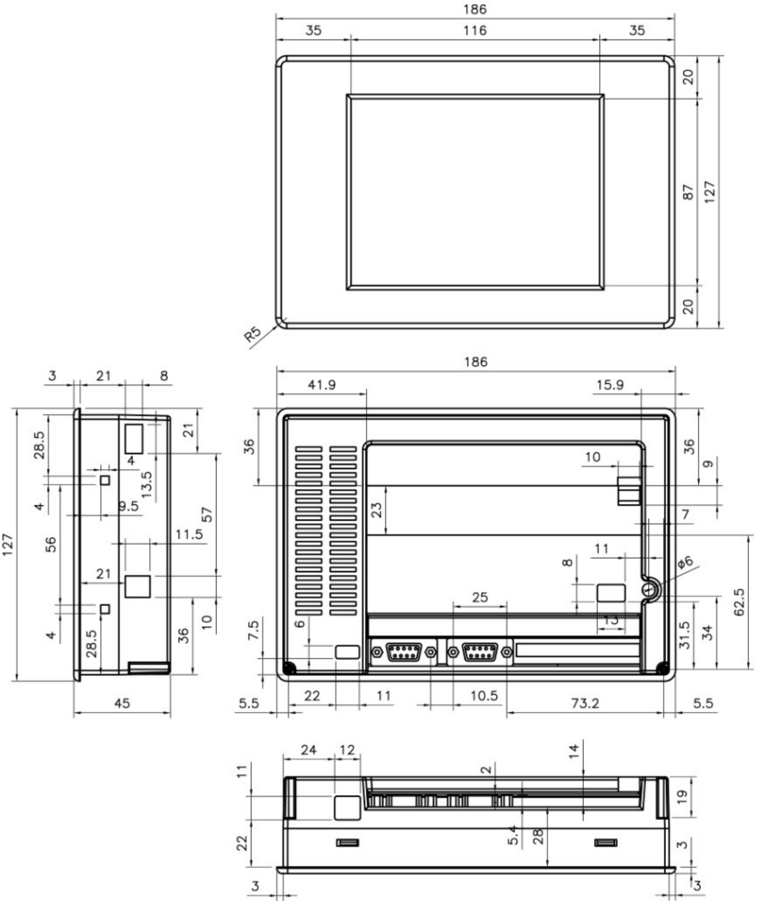
Hardware Requirements

The **Cubloc Studio** software used to develop programs for the Cutouch will run on a computer with Windows 7, Vista, 2000, or 98 installed. If you would like to use it in Linux/Unix/Macintosh environment, virtual machine software, such as Vmware, will be needed to host the Windows operating system.

An RS-232 port is also required, or you may use a USB-to-RS232C converter. Programs can be downloaded, monitored, and debugged when connected to a PC via RS-232.

When the Cutouch is disconnected from the PC, it functions as a stand-alone device. The main program is stored in the Cutouch's flash memory, and will be retained even between power cycles. Programs can be downloaded and erased thousands of times.

Cutouch Dimensions



Cutouch I/O Ports

Name	I/O	Port Block	Description
P0	I/O	Block 0	ADC0
P1	I/O		ADC1
P2	I/O		ADC2
P3	I/O		ADC3
P4	I/O		ADC4
P5	I/O		ADC5
P6	I/O		ADC6
P7	I/O		ADC7
P8	I/O	Block 1	PWM0
P9	I/O		PWM1
P10	I/O		PWM2
P11	I/O		PWM3
P12	I/O		PWM4 / INT0
P13	I/O		PWM5 / INT1
P14	I/O		INT2
P15	I/O		INT3 (RS485 Transmit enable)
P16	I/O		High-speed Counter 0
P17	IN		High-speed Counter 1
P18	OUTPUT		BUZZER
P19~P23			N/C
P24~31	OUTPUT	Block 3	8 Output only port –NPN TR
P32~39	OUTPUT	Block 4	8 Output only port – NPN TR
P40~47	OUTPUT	Block 5	8 Output only port – NPN TR
P48~55	OUTPUT	Block 6	8 Output only port – NPN TR
P56~63	INPUT	Block 7	8 Input only port- DC24V
P64~71	INPUT	Block 8	8 Input only port - DC24V
P72~79	INPUT	Block 9	8 Input only port - DC24V
P80~87	INPUT	Block 10	8 Input only port - DC24V

N/C = No Connection

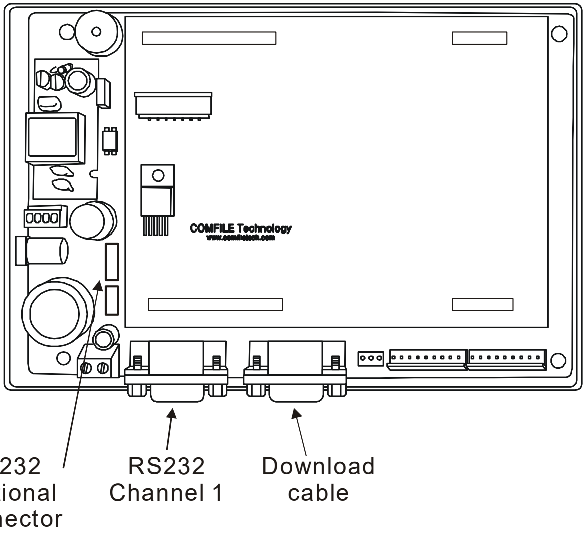
The Cutouch CT1720 I/O Ports are TTL 5V.

The Cutouch Add-On Board allows opto-isolated 24V DC inputs and 24V TR outputs for J1 through J4.

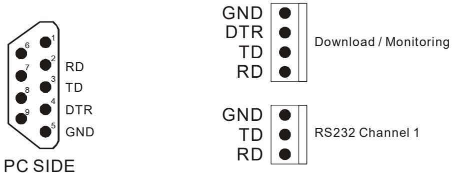
The Cutouch CT1721 is a combination of the CT1720 plus the Add-On Board.

* Please be careful to not input more than 5V into a Cutouch TTL ports as it can damage the product.

There are extra RS232 headers as shown below:



The Download Channel is a 4-pin connector and RS-232 Channel 1 is a 3 pin connector. They can be connected to a PC as shown below:



Backup Battery

The Cutouch will maintain data in its volatile memory when powered off by using a backup battery. If backup is not needed, the program should clear the memory at the beginning of the program. Use `RamClear` to clear all data memory at the start of your program.

*The Cutouch comes with a self-charging 1.0F super-capacitor that can last about a day (up to 30hrs). You can replace it with a 10.0F super-capacitor to extend the duration to about 300 hours (12.5 days). Adding a battery can provide additional backup time depending on capacity. To add a backup battery, please connect to the ports labeled “External Battery,” under the super-capacitor (not visible when the back cover is in place).

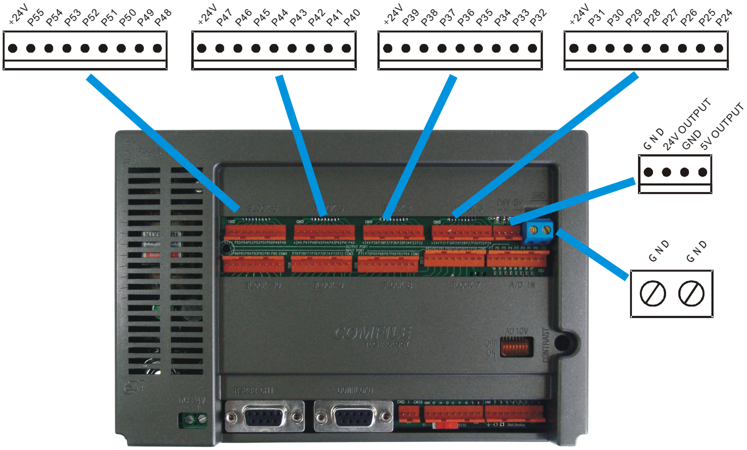
```
Const Device = CT1720
Dim TX1 As Integer, TY1 As Integer
TX1 = 0
TY1 = 0        ' Clear just this variable
RamClear       ' Clear all RAM
```

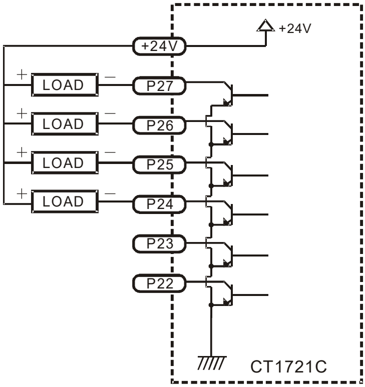
In Ladder Logic, all registers S, M, C, T, and D are retained by the backup battery. Register P is cleared by default when the device is powered on. If you only want to clear parts of a register, rather than all registers, use the following method:

```
Const Device = CT1720
Dim i As Integer
For i=0 to 32       ' Clear only Register M0 to M32
    _M(i) = 0
Next
Set Ladder On
```

Cutouch Output ports

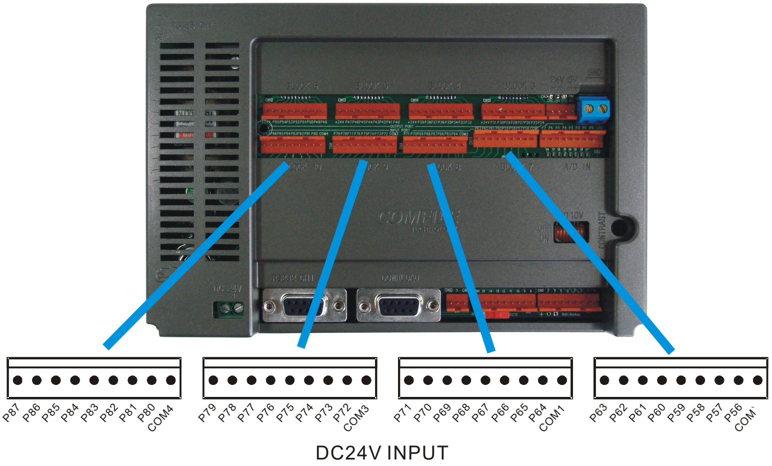
CT1721C has 32 NPN TR output ports. (Sink DC output)

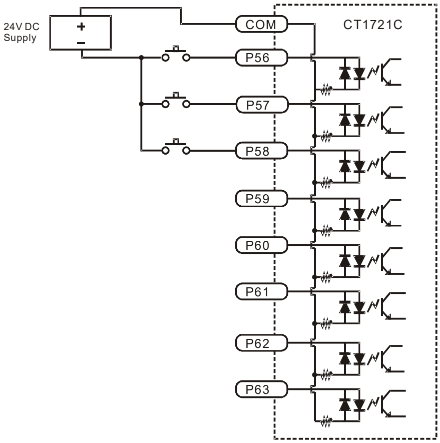

+24V P55 P54 P53 P52 P51 P50 P49 P48
+24V P47 P46 P45 P44 P43 P42 P41 P40
+24V P39 P38 P37 P36 P35 P34 P33 P32
+24V P31 P30 P29 P28 P27 P26 P25 P24
GND
24V OUTPUT
GND
5V OUTPUT
GND
GND
COMFILE



Cutouch Input ports

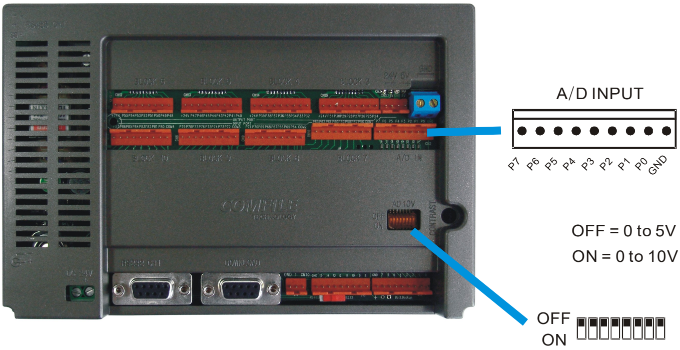
CT1721C has 32 opto-isolated DC input.



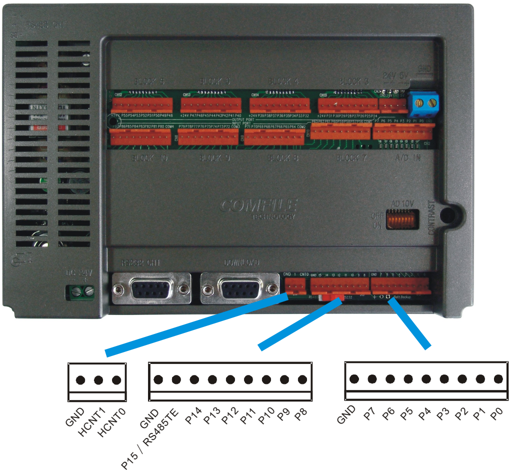


Cutouch A/D and TTL I/O Ports

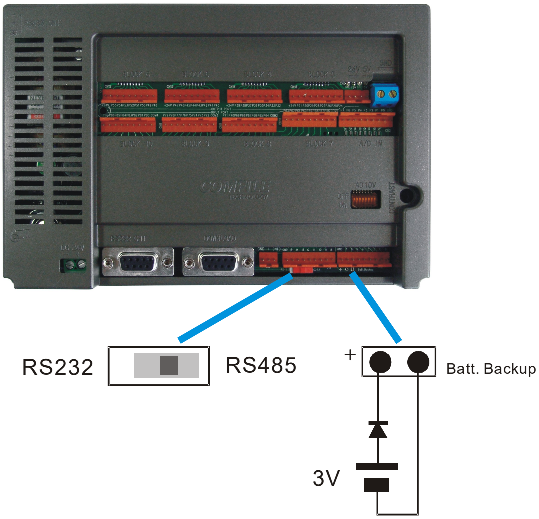
CT1721C has 8 A/D ports.



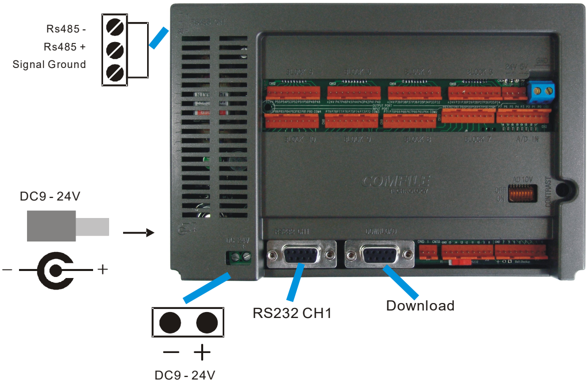
CT1721C has 16 TTL I/O ports and 2 High speed Counter.



Cutouch jumper & connector

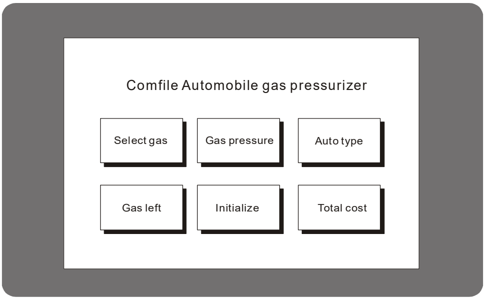


This battery supports Data Sram memory only.



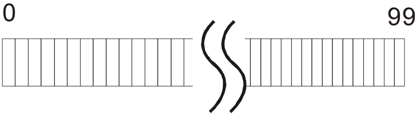
Menu System Library

The Cutouch supports extra commands easily create and manipulate menus. With the menu system library, a menu system as shown below can be made in less than 5 minutes.



MENU Commands

The Cutouch has enough memory to support approximately 100 menu buttons. The `MenuSet` command is used to set the x and y coordinates, as well as the style of the menu. The `MenuTitle` command can be used to name the menu. When touch input is received, the `MenuCheck` command can be used to determine which menu button was pressed.



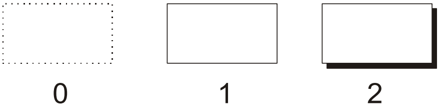
Each menu button's position and style can be changed using the `MenuSet` command. Up to 100 buttons can be displayed on a single screen. Each button can be assigned a different function on a different screen, resulting in virtually unlimited menus and buttons.

MenuSet

MenuSet index, style, x1, y1, x2, y2

index : Menu index number
style : Button style; 0=none, 1=box, 2=box with shadow
x1,y1,x2,y2 : Menu button position

The `index` value must be a number from 0 through 99. `Style` is the shape of the button, where 0 is for no box, 1 is for a box, and 2 is for a shadowed box.



`x1, y1, x2, y2` are the x and y axis positions of the left upper and lower right corners. When this command is executed, the set part of the screen becomes part of the button's area.

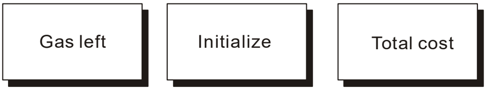
MenuTitle

MenuTitle index, x, y, string

index :Menu index number
x,y : Title location based on left upper corner of button
string : Name of the menu

`MenuSet` only draws the box itself. Use the `MenuTitle` command to set the name of the menu like shown here:

```
     MenuTitle 0,13,13,"Gas Left"
     MenuTitle 1,16,13,"Initialize"
     MenuTitle 2,13,13,"Total Cost"
```



MenuCheck( )

variable = MenuCheck(index, touchX, touchX)

variable : Variable to store results (1 if selected, 0 if unselected)
index : Menu index number
touchX : x coordinate of the touch
touchY : y coordinate of the touch

Use this command to determine which menu button has been touched. `touchX` and `touchY` are the x and y coordinates of where the user touched the screen. If the coordinates of the touch match the coordinates of the button, 1 is returned, otherwise 0 is returned.

```
If MenuCheck(0,TX1,TY1) = 1 Then
        MenuReverse 0
        Beep 18,180
End If
```

MenuReverse

MenuReverse index

index : Menu index number

This command causes the menu button identified by index to have it's colors reversed. This is useful to provide visual feedback to a user, indicating that a menu button has been touched.

Initialize
Total cost

Menu( )

Variable = Menu(index, position)

variable : Variable to store results (1 = selected, 0 = unselected)
index : Menu index
position : Position (0=x1, 1=y1, 2=x2, 3=y2)

To find the current coordinates of Menu buttons set using the `MenuSet` command, use `Menu()` function to return the current status of the specified menu. 0 will read x2, 1 will read y1, 2 will read x2, and 3 will read y2.

```
If Menu(0,1) < 100 THEN    ' If Menu button 0' s Y1 is less than 100
```

WaitDraw

WaitDraw

This command will wait for a drawing command to finish before resuming execution.

```
ElFill  200,100,100,50     ' Fill an ellipse
WaitDraw                   ' Wait until drawing is finished.
```

This command is especially useful for animations, and if graphics are displayed at a high rate.

The Cutouch has an internal buffer for receiving graphic commands from the internal Cubloc controller. If this buffer fills up and data is sent to it, the existing data in the buffer can be overwritten. In order to avoid these situations, use the `WaitDraw` command to wait until the buffer has enough space before sending graphic commands.

If graphics need to be drawn repeatedly, `WaitDraw` can be used to prevent overrunning the buffer, which may appear as noise on the LCD.

This command can only be used with the Cutouch.

Touch Pad Input Example

`Set Pad`, `On Pad`, and `GetPad` can all be used to determine which menu buttons were touched by the user.

All `Pad` commands are used for receiving and processing touch input. We can use `On Pad` interrupts to capture touch events. The following is an example program that uses the touch pad:

```
     '
     '   DEMO FOR Cutouch
     '
     Const Device = CT1720
     Dim TX1 As Integer, TY1 As Integer
     Set Pad 0,4,5              '← (1) Activate Touch PAD Input
     On Pad GoSub abc           '← (2) Declare pad interrupts
     Do
     Loop
abc:
     TX1 = GetPad(2)            '← (3) Interrupt Service routine
     TY1 = GetPad(2)
     CircleFill TX1,TY1,10      '← (4) Draw a circle where it
                                '        was touched
Return
```

(1) `SET PAD 0, 4, 5` : This command will activate the pad inputs. (Syntax: `Set Pad mode, packetSize, bufferSize`). The Cutouch has a separate touch controller that will sense touch input and send it back to the CPU using the SPI protocol. This touch controller will create a pad signal that is equal to mode = 0 (Most significant bit, rising edge sampling). Input packets are 4 bytes each (x and y coordinates each get 2 bytes). Buffer size is 5; one more than the actual packet size.

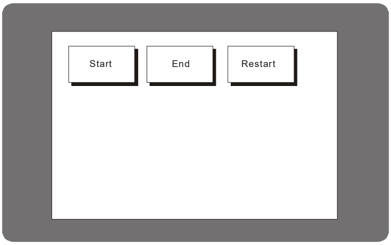
(2) `On Pad GoSub abc`: This command is for PAD interrupt declaration. When a pad event occurs, the program will jump to label `abc`.

(3) This is the interrupt service routine. When a pad event occurs, this part of the code will be executed, until `Return`. `GetPad` will read the data received from touch pad, 2 bytes for the x coordinate and 2 bytes for the y coordinate.

(4) Draw a circle where touch input was received.
When this program is executed, a circle will appear wherever the screen is touched. This program can be used as a skeleton for touch programs.

The following is an example combining menu and pad commands. When a menu button is pressed, a beep will sound and the menu button will colors reversed.

```
     '
     '   DEMO FOR Cutouch
     '
     Const Device = CT1720
     Dim TX1 As Integer, TY1 As Integer
     Dim k As Long
     Contrast 550
     Set Pad 0,4,5
     On Pad GoSub abc
     MenuSet 0,2,8,16,87,63
     MenuTitle 0,13,13,"Start"
     MenuSet 1,2,96,16,176,63
     MenuTitle 1,13,13,"End"
     MenuSet 2,2,184,16,264,63
     MenuTitle 2,13,13,"Restart"
     Low 18
     Do
     Loop
abc:
     TX1 = GetPad(2)
     TY1 = GetPad(2)
     CircleFill TX1,TY1,10
     If MenuCheck(0,TX1,TY1) = 1 Then
            MenuReverse 0
            PulsUut 18,300      ' Send out beep to piezo
     End If
     If MenuCheck(1,TX1,TY1) = 1 Then
            MenuReverse 1
            PulsOut 18,300
     End If
     If MenuCheck(2,TX1,TY1) = 1 Then
            MenuReverse 2
            PulsOut 18,300
     End If
     Return
```



Cutouch Sample Programs

SAMPLE 1

Let’s make a simple counter that will print to the screen. The source files used here are in the Cubloc Studio installation directory. (Usually C:\Program Files\Comfile Tools\CublocStudio)

3241

<Filename : CT001.CUL>

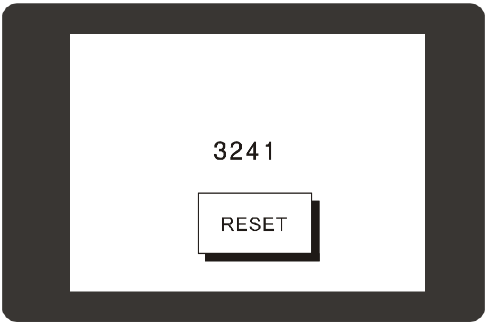
```
Const Device = Ct1720
Dim i As Integer
Contrast 550  ' LCD CONTRAST SETTING

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

Please adjust the screen’s contrast accordingly using the `Contrast` command.

* Depending on the model, the contrast may be able to be adjusted using a knob on the back of the Cutouch, providing the ability to change the contrast manually.

SAMPLE 2

The following sample program will display a "RESET" button and will increment the number displayed on the screen every time the button is pressed.



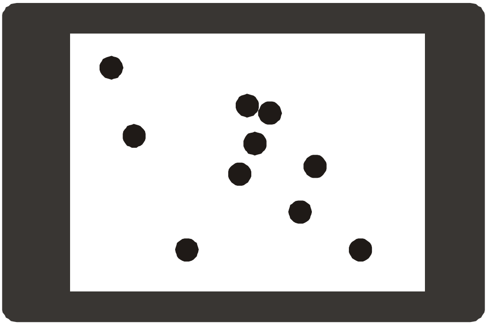
<Filename : CT002.CUL>

```
     Const Device = Ct1720
     Dim i As Integer
     Dim TX1 As Integer, TY1 As Integer
     Contrast 550
     Set Pad 0,4,5
     On Pad GoSub GETTOUCH
     MenuSet 0,2,120,155,195,200
     MenuTitle 0,20,14,"RESET"

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

GETTOUCH:
     TX1 = GetPad(2)
     TY1 = GetPad(2)
     If MenuCheck(0,TX1,TY1) = 1 Then
            PulsOut 18,300
            I = 0
     End If
     Return
```

The SET PAD command activates touch input. The ON PAD command jumps to a label when touch input is received. The MENUSET command sets the desired touch input area and the MENUTITLE command sets the name of the button itself. `PulsOut` outputs a BEEP sound to the piezo speaker.

SAMPLE 3

This sample draws a circle wherever the screen is touched.

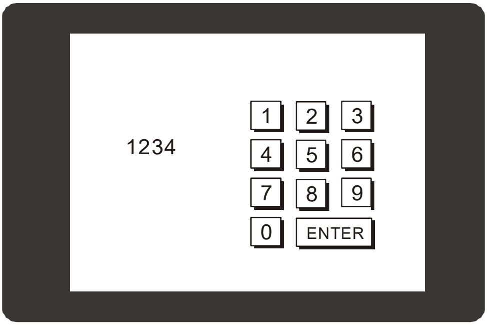


<Filename : CT003.CUL>

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

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

SAMPLE 4

This sample make a virtual keypad that accept numerical input.



<Filename : CT004.CUL>

```
     Const Device = Ct1720
     Dim TX1 As Integer, TY1 As Integer
     Dim I As Integer
     Contrast 550
     Set Pad 0,4,5
     MenuSet
     MenuSet 0,2,165,50,195,75
     MenuTitle 0,11,4,"1"
     MenuSet 1,2,205,50,235,75
     MenuTitle 1,11,4,"2"
     MenuSet 2,2,245,50,275,75
     MenuTitle 2,11,4,"3"
     MenuSet 3,2,165,85,195,110
     MenuTitle 3,11,4,"4"
     MenuSet 4,2,205,85,235,110
     MenuTitle 4,11,4,"5"
     MenuSet 5,2,245,85,275,110
     MenuTitle 5,11,4,"6"
     MenuSet 6,2,165,120,195,145
     MenuTitle 6,11,4,"7"
     MenuSet 7,2,205,120,235,145
     MenuTitle 7,11,4,"8"
     MenuSet 8,2,245,120,275,145
     MenuTitle 8,11,4,"9"
     MenuSet 9,2,165,155,195,180
     MenuTitle 9,11,4,"0"
     MenuSet 10,2,205,155,275,180
     MenuTitle 10,17,4,"ENTER"  I =0
     Do
     Loop
GETTOUCH:
     TX1 = GetPad(2)
     TY1 = GetPad(2)
     If MenuCheck(0,TX1,TY1) = 1 Then
```

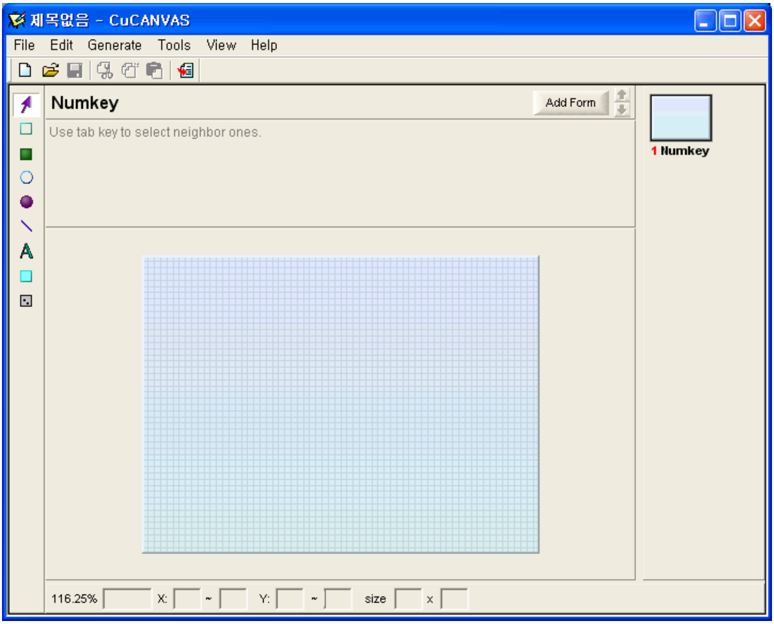
```
            I = I << 4
            I = I + 1
            PulsOut 18,300
    Elseif MenuCheck(1,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 2
            PulsOut 18,300
    Elseif MenuCheck(2,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 3
            PulsOut 18,300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

The final value "I" is stored in BCD format. The `Bcd2Bin` command can be used to convert it back to a binary number.

SAMPLE 5

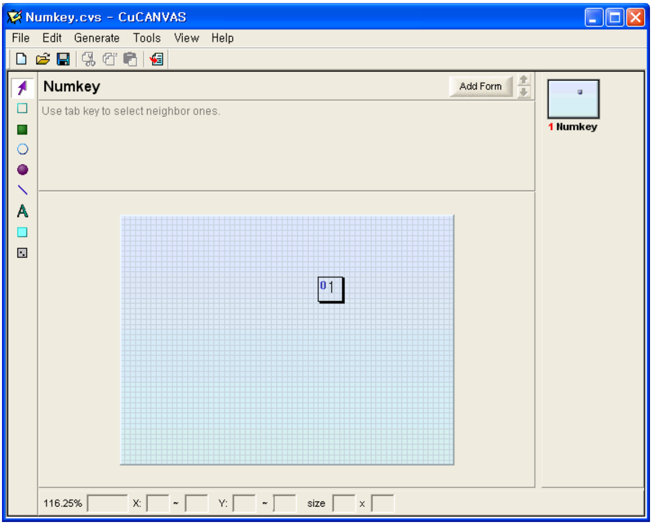
This sample uses CUCANVAS to make some menus. To create the virtual keypad shown in the previous page, it would take a long time to manually code it and place buttons. CUCANVAS can save time.

Run CUCANVAS and press the "Add Form" button in the upper right hand corner. Enter a desired name for the new form ("NumKey" is used in this example).

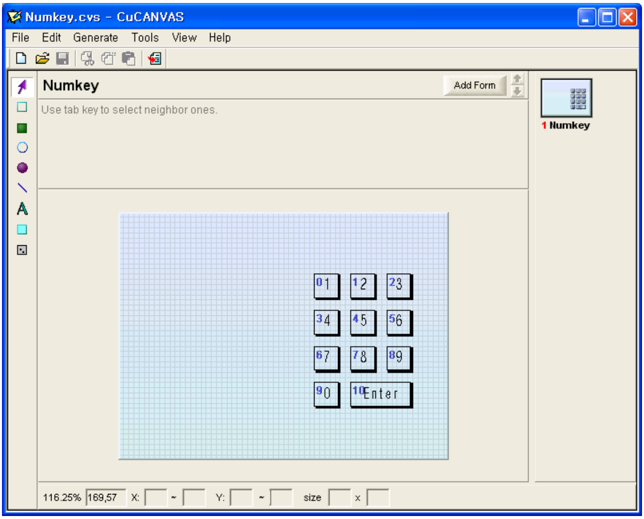


On the left side of CUCANVAS, there a toolbar with an arrow, box, filled box, circle, filled circle, line, text, and menu box. Select the last button, menu box, and draw a small box on the screen.

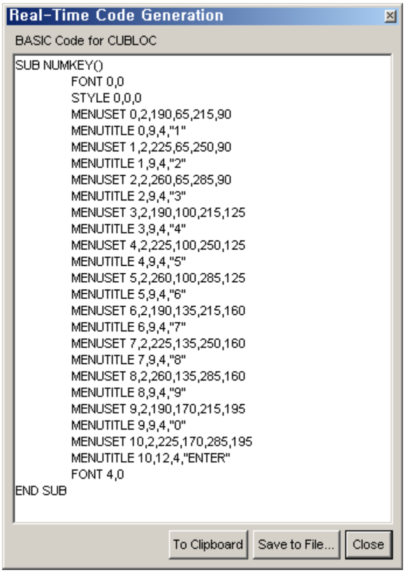
The 0 on the button means the menu number is 0. In the actual screen, this number will not be displayed. Type "1" in the Title field on the top. There is now a "1" button.


Numkey.cvs - CuCANVAS
File Edit Generate Tools View Help
Numkey
Add Form
Use tab key to select neighbor ones.
0 1
1 Numkey
116.25%
X:
~
Y:
~
size
x

A keypad like the one shown below can be created in less than 5 minutes by creating each button in the same way the "1" button was made.

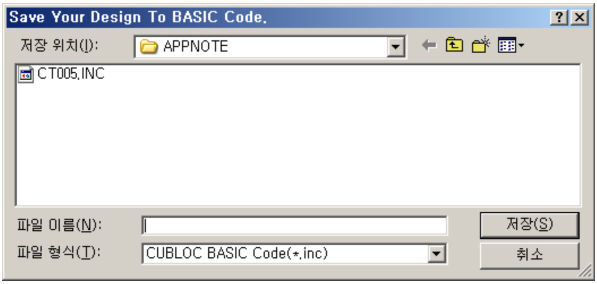


Now for the fun part. Simply click "Generate" on the menu bar, and click "View Basic Code." CUCANVAS will generate a subroutine that includes the buttons just created. Simply copy (Ctrl+C) and paste (CTRL+V) this code to Cubloc Studio. A complete menu has just been created in a matter of minutes. The "To Clipboard" button will also copy the generated code to the clipboard.



```
SUB NUMKEY()
        FONT 0,0
        STYLE 0,0,0
        MENUSET 0,2,190,65,215,90
        MENUTITLE 0,9,4,"1"
        MENUSET 1,2,225,65,250,90
        MENUTITLE 1,9,4,"2"
        MENUSET 2,2,260,65,285,90
        MENUTITLE 2,9,4,"3"
        MENUSET 3,2,190,100,215,125
        MENUTITLE 3,9,4,"4"
        MENUSET 4,2,225,100,250,125
        MENUTITLE 4,9,4,"5"
        MENUSET 5,2,260,100,285,125
        MENUTITLE 5,9,4,"6"
        MENUSET 6,2,190,135,215,160
        MENUTITLE 6,9,4,"7"
        MENUSET 7,2,225,135,250,160
        MENUTITLE 7,9,4,"8"
        MENUSET 8,2,260,135,285,160
        MENUTITLE 8,9,4,"9"
        MENUSET 9,2,190,170,215,195
        MENUTITLE 9,9,4,"0"
        MENUSET 10,2,225,170,285,195
        MENUTITLE 10,12,4,"ENTER"
        FONT 4,0
END SUB
```

Instead of copying and pasting repetitive menu creations, include files can be used. Click the "Save to File" button and save the code as an include (*.inc) file.



저장 위치(I): APPNOTE

CT005.INC

파일 이름(N):

파일 형식(T): CUBLOC BASIC Code(*.inc)

Include files make it easy to change the interface of a program without a lot of cut and paste operations within the main code.

The following program is exactly same as SAMPLE 4 except an include file is used for the virtual keypad:

<Filename : CT005.CUL>

```
     Const Device = Ct1720
     Dim TX1 As Integer, TY1 As Integer
     Dim I As Integer
     Contrast 550
     Set Pad 0,4,5
     On Pad GoSub GETTOUCH
     NUMKEY        ' Execute the Sub-routine in INCLUDE file
     I =0
     Do
     Loop

GETTOUCH:
     TX1 = GetPad(2)
     TY1 = GetPad(2)
     If MenuCheck(0,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 1
            PulsOut 18,300
     Elseif MenuCheck(1,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 2
            PulsOut 18,300
     Elseif MenuCheck(2,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 3
            PulsOut 18,300
     Elseif MenuCheck(3,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 4
            PulsOut 18,300
     Elseif MenuCheck(4,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 5
            PulsOut 18,300
     Elseif MenuCheck(5,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 6
            PulsOut 18,300
     Elseif MenuCheck(6,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 7
            PulsOut 18,300
     Elseif MenuCheck(7,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 8
            PulsOut 18,300
     Elseif MenuCheck(8,TX1,TY1) = 1 Then
            I = I << 4
            I = I + 9
            PulsOut 18,300
     Elseif MenuCheck(9,TX1,TY1) = 1 Then
            I = I << 4
            PulsOut 18,300
     Elseif MenuCheck(10,TX1,TY1) = 1 Then
            I = 0
            PulsOut 18,300
     End If
     Locate 3,3
     Print HEX4 I

     Return

     End

#INCLUDE "CT005.INC"
```

We must place the #include command at the end of the code, as the generated code is in the form of a subroutine, which must come after the `End` statement in the main program.

CUCANVAS can be downloaded at www.cubloc.com. CUCANVAS is free to use with Cutouch products.

SAMPLE 6

This sample demonstrates how to set up a paging and menu system.

Switching between screens is quite simple. Maintain a variable that keeps track which screen is currently being displayed. While switching to a new screen, always update this variable. Use the variable to determine which set of Menucheck tests should be run for a particular screen. Subroutines are very useful for compartmentalizing the code.

<Filename : CT1721treemenu.cub >

```
Const Device = CT1720
Ramclear

Set Pad 0,4,5
On Pad Gosub ProcessTouch

Dim TX1 As Integer
Dim TY1 As Integer

Dim CurrentScreen As Byte
   #define _MAINMENU 0
   #define _SUBMENU1 1
   #define _SUBMENU2 2
   #define _SUBMENU3 3
   #define _SUBMENU4 4

MAIN
CurrentScreen = _MAINMENU

Do
   If CurrentScreen = _MAINMENU Then
      Set Onpad Off
      DisplayTime
      Set Onpad On
   Endif

   Delay 250
Loop

ProcessTouch:
   TX1 = GetpadX()
   TY1 = GetpadY()

   Select Case CurrentScreen
      Case _MAINMENU
         ProcessMainMenu
      Case _SUBMENU1
         ProcessSubMenu1
      Case _SUBMENU2
```

```
         ProcessSubMenu2
      Case _SUBMENU3
         ProcessSubMenu3
      Case _SUBMENU4
         ProcessSubMenu4
   End Select
Return
End


Sub ProcessMainMenu()
   If Menucheck(0,TX1,TY1) = 1 Then
      FlashMenu 0
      CurrentScreen = _SUBMENU1
      Cls
      SUBMENU1

   Elseif Menucheck(1,TX1,TY1) = 1 Then
      FlashMenu 1
      CurrentScreen = _SUBMENU2
      Cls
      SUBMENU2

   Elseif Menucheck(2,TX1,TY1) = 1 Then
      FlashMenu 2
      CurrentScreen = _SUBMENU3
      Cls
      SUBMENU3

   Elseif Menucheck(3,TX1,TY1) = 1 Then
      FlashMenu 3
      CurrentScreen = _SUBMENU4
      Cls
      SUBMENU4
   Endif
End Sub

Sub ProcessSubMenu1()
   If Menucheck(0,TX1,TY1) = 1 Then
      FlashMenu 0
      Beeper 1

   Elseif Menucheck(1,TX1,TY1) = 1 Then
      FlashMenu 1
      CurrentScreen = _MAINMENU
      Cls
      MAIN
   Endif
End Sub

Sub ProcessSubMenu2()
   If Menucheck(0,TX1,TY1) = 1 Then
      FlashMenu 0
      Beeper 2

   Elseif Menucheck(1,TX1,TY1) = 1 Then
      FlashMenu 1
      CurrentScreen = _MAINMENU
      Cls
      MAIN
   Endif
End Sub

Sub ProcessSubMenu3()
   If Menucheck(0,TX1,TY1) = 1 Then
      FlashMenu 0
      Beeper 3

   Elseif Menucheck(1,TX1,TY1) = 1 Then
      FlashMenu 1
      CurrentScreen = _MAINMENU
      Cls
      MAIN
   Endif
End Sub

Sub ProcessSubMenu4()
   If Menucheck(0,TX1,TY1) = 1 Then
      FlashMenu 0
      Beeper 4

   Elseif Menucheck(1,TX1,TY1) = 1 Then
      FlashMenu 1
      CurrentScreen = _MAINMENU
      Cls
      MAIN
   Endif
End Sub

Sub Beeper(Num As Byte)
   Dim i As Byte

   For i = 1 To Num
      High 18
      Delay 100
      Low 18
      Delay 200
   Next
End Sub


Sub FlashMenu(Num As Byte)
   Menureverse Num
   Delay 150
   Menureverse Num
End Sub
```

```
    Sub DisplayTime()
       Glocate 16,220
       Font 0,0
                                                                 Dprint
Dp(Bcd2bin(Time(2)),2,1),":",Dp(Bcd2bin(Time(1)),2,1),":",Dp(Bcd2bin(Ti
me(0)),2,1)
    End Sub

    #include "CT1721treemenu.inc"
```

Use CuCanvas to Generate your menus.

<Filename : CT1721treemenu .inc>

```
         SUB MAIN()
       FONT 6,1
       STYLE 0,0,0
       GLOCATE 96,24
       GPRINT "Main Menu"
       FONT 0,1
       MENUSET 0,2,40,96,144,128
       MENUTITLE 0,20,8,"Submenu 1"
       MENUSET 1,2,168,96,280,128
       MENUTITLE 1,24,8,"Submenu 2"
       MENUSET 2,2,40,152,144,184
       MENUTITLE 2,20,8,"Submenu 3"
       MENUSET 3,2,168,152,280,184
       MENUTITLE 3,24,8,"Submenu 4"
       LINESTYLE 0
       DOTSIZE 0,0
       COLOR 1
       BOX 80,16,232,56
       FONT 4,0
    END SUB

    SUB SUBMENU1()
       FONT 6,1
       STYLE 0,0,0
       GLOCATE 96,24
       GPRINT "Submenu 1"
       LINESTYLE 0
       DOTSIZE 0,0
       COLOR 1
       BOX 80,16,232,56
       FONT 2,1
       MENUSET 0,2,88,88,224,120
       MENUTITLE 0,22,8,"Beep 1 time"
       MENUSET 1,2,0,208,72,239
       MENUTITLE 1,10,7,"<BACK"
       FONT 4,0
    END SUB
```

```
SUB SUBMENU2()
   FONT 6,1
   STYLE 0,0,0
   GLOCATE 96,24
   GPRINT "Submenu 2"
   LINESTYLE 0
   DOTSIZE 0,0
   COLOR 1
   BOX 80,16,232,56
   FONT 2,1
   MENUSET 0,2,88,88,224,120
   MENUTITLE 0,16,8,"Beep 2 times"
   MENUSET 1,2,0,208,72,239
   MENUTITLE 1,10,7,"<BACK"
   FONT 4,0
END SUB

SUB SUBMENU3()
   FONT 6,1
   STYLE 0,0,0
   GLOCATE 96,24
   GPRINT "Submenu 3"
   LINESTYLE 0
   DOTSIZE 0,0
   COLOR 1
   BOX 80,16,232,56
   FONT 2,1
   MENUSET 0,2,88,88,224,120
   MENUTITLE 0,16,8,"Beep 3 times"
   MENUSET 1,2,0,208,72,239
   MENUTITLE 1,10,7,"<BACK"
   FONT 4,0
END SUB

SUB SUBMENU4()
   FONT 6,1
   STYLE 0,0,0
   GLOCATE 96,24
   GPRINT "Submenu 4"
   LINESTYLE 0
   DOTSIZE 0,0
   COLOR 1
   BOX 80,16,232,56
   FONT 2,1
   MENUSET 0,2,88,88,224,120
   MENUTITLE 0,16,8,"Beep 4 times"
   MENUSET 1,2,0,208,72,239
   MENUTITLE 1,10,7,"<BACK"
   FONT 4,0
END SUB
```

Chapter 13: CB405RT

CB405RT

- CB405 core module
- equipped with a built-in accurate RTC (DS3231)
- equipped with a built-in 16-bit A/D converter (8 channels)

The CB405RT is a product, which adds an RTC and a 16-bit ADC to the original CB405. The external dimensions and pinout are the same as the CB405 with the exception that ports P32 to P37 are used for the 16-bit ADC and therefore cannot be used.

The CB405RT has a built-in realtime clock (DS3231), which automatically compensates for clock errors due to temperature changes, sporting improved accuracy over the existing RTC.

When a supercapacitor is connected to VBB, data memory is preserved even when the power supply is interrupted. Since the supercapacitor is charged automatically when the device is powered, the supercapacitor can compensate for the power supply if interrupted, eliminating the need for batteries.

CB405RT-Related Commands

CB405RT-related commands can be used on in Cubloc Studio V2.6.B or in later versions.

First, the RTC-related commands

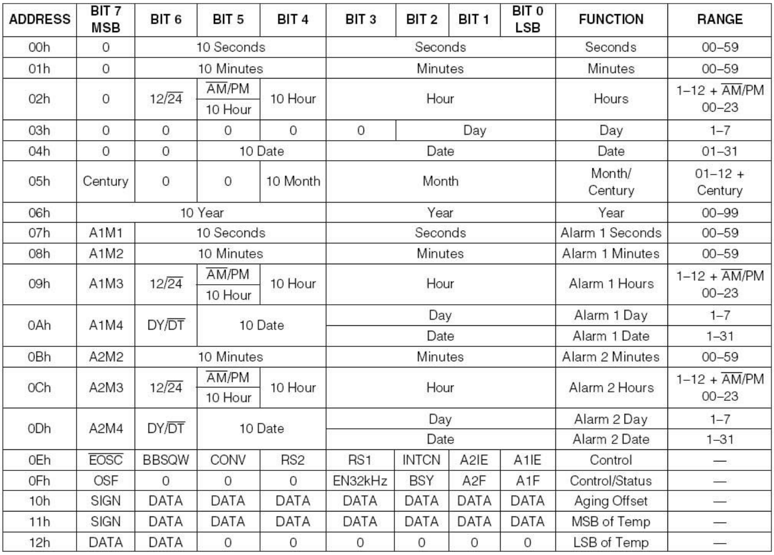
RTCRead( )

variable = RTCRead(address)

variable : variable to store the result (Byte type)
address : RTC's address

If you see the address table of RTC chip (DS3231) below, you will notice that the time data are stored from addresses 0 through 6. The rest of the addresses are related to the alarm and RTC chipsetting; thus, we recommend those interested to refer to the DS3231 databook.

`RTCRead` command literally reads data from the RTC chip.



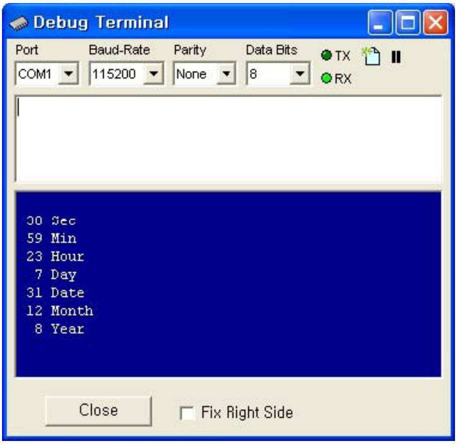
ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	$12/\overline{24}$	$\overline{\text{AM}}$/PM 10 Hour	10 Hour	Hour				Hours	1–12 + $\overline{\text{AM}}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	$12/\overline{24}$	$\overline{\text{AM}}$/PM 10 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{\text{AM}}$/PM 00–23
0Ah	A1M4	DY/$\overline{\text{DT}}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	$12/\overline{24}$	$\overline{\text{AM}}$/PM 10 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{\text{AM}}$/PM 00–23
0Dh	A2M4	DY/$\overline{\text{DT}}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{\text{EOSC}}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

CAUTION:

To use CB405RT, in the beginning of the program, one must replace `Const Device = CB405` with `#Include "CB405RT"`.

```
#include "CB405RT"    ' Use "#include" instead of the "Const
                       ' Device" statement.
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
```

When the above example program is executed, the following debug window will appear.



```
30 Sec
59 Min
23 Hour
 7 Day
31 Date
12 Month
 8 Year
```

RTCWrite

RTCWrite address, data

address : RTC chip's address

data : the variable or constant to store

This command writes a new data value to the specified address of the RTC chip.

In other RTC chips, clock inaccuracies can occur due to temperature changes. The RTC chips, which work properly at room temperature, begin to tick abnormally as their crystals oscillations are affected by temperature.

The DS3231 is an RTC chip that avoids such problems. It has a thermistor (i.e., a temperature sensor) along with a 32 KHz crystal. The oscillation frequency is adapted based on the temperature sensed by the thermistor. As a result, the crystal oscillations don't fluctuate and the clock keeps better time.

But be aware that although the DS3231 keeps great time under varying temperatures, it is still not 100% perfect and should be verified for time critical applications.

HADIn( )

variable = HADIn(channel)

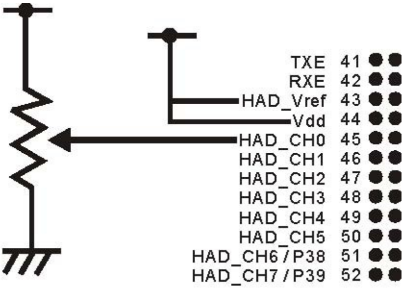
variable : Variable to store the result (Integer or Long type)

channel : AD channel

This function stores the result of 16-bit A/D conversion at the specified variable.

```
#include "CB405RT"
Do
 Debug Goxy,1,1,Dec5 HADIn(0)
 Wait 500
Loop
```

For the above example program to work, a 5V source should be connected to HAD_Vref and a volume resistance should be connected to the HAD_CH0 as shown below. Note that only voltages ranging from 1V to 5V can be connected to HAD_Vref and only voltages ranging from 0V to HAD_Vref can be applied to HAD_CH0 through HAD_CH7.



The 16-bit ADC gives a result ranging from 0V to Vref expressed as one of the 65536 equally divided intervals. It is capable of much more accurate measurement compared to a 10-bit ADC.

HADIn2( )

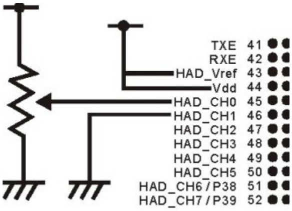
variable = HADIn2(channel combination)

variable : Variable to store the result (Integer or Long type)
channel : Combination: AD channel combination code

This command is for a differential A/D input. Two channels are paired up, and the A/D converter converts the difference between the two voltages.

Channel combination code	+ Input	- Input
0	CH0	CH1
1	CH2	CH3
2	CH4	CH5
3	CH6	CH7
4	CH1	CH0
5	CH3	CH2
6	CH5	CH4
7	CH7	CH6

```
#include "CB405RT"
Do
 Debug Goxy,1,1,Dec5 Hadin2(0)
 Wait 500
Loop
```



TIP

For a more accurate measurement, one should start with a reliable power source. A Linear power source (e.g., using 7805, etc.) is better for A/D input than a switching power source circuit (e.g., using LM2576). Also, when the A/D input pins are extended and exposed outside the board, a protection circuit (where additional chips such as the isolation circuit, etc. are involved) should be added to protect the core module from external noise. Keep in mind that, especially when a voltage over 5V could be applied to the A/D input port, repairable damage can occur to the module

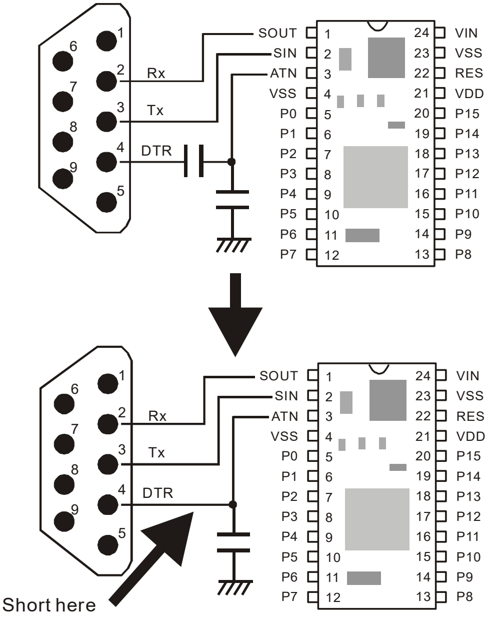
Appendix A: ASCII CODE

Code	char.	Code	char.	Code	char.	Code	char.
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

Appendix B: Note for BASIC STAMP users

When using Parallax's Basic Stamp compatible development board, please be aware of the following:

There is a capacitor on the Basic Stamp compatible development boards which may cause a download error in Cubloc Studio. Please short (or take out) the extra capacitor connected to the DTR pin of the board as shown below. The Cubloc already has this capacitor on the chip itself.



Short here

Ladder Logic Special Registers

Special Register	Explanation
F0	Always OFF
F1	Always ON
F2	Turn on 1 SCAN time at POWER UP (Set Ladder On).
F3	Reserved
F4	Reserved
F5	Reserved
F6	Reserved
F7	Reserved
F8	1 SCAN every 10ms
F9	1 SCAN every 100ms
F10	Reserved
F11	Reserved
F12	Reserved
F13	Reserved
F14	Reserved
F15	Reserved
F16	Repeat ON/OFF every 1 Scan time.
F17	Repeat ON/OFF every 2 Scan times.
F18	Repeat ON/OFF every 4 Scan times.
F19	Repeat ON/OFF every 8 Scan times.
F20	Repeat ON/OFF every 16 Scan times.
F21	Repeat ON/OFF every 32 Scan times.
F22	Repeat ON/OFF every 64 Scan times.
F23	Repeat ON/OFF every 128 Scan times.
F24	Repeat ON/OFF every 10ms
F25	Repeat ON/OFF every 20ms
F26	Repeat ON/OFF every 40ms
F27	Repeat ON/OFF every 80ms
F28	Repeat ON/OFF every 160ms
F29	Repeat ON/OFF every 320ms
F30	Repeat ON/OFF every 640ms
F31	Repeat ON/OFF every 2.56 seconds
F32	Repeat ON/OFF every 5.12 seconds
F33	Repeat ON/OFF every 10.24 seconds
F34	Repeat ON/OFF every 20.48 seconds
F35	Repeat ON/OFF every 40.96 seconds
F36	Repeat ON/OFF every 81.92 seconds
F37	Repeat ON/OFF every 163.84 seconds
F38	Repeat ON/OFF every 327.68 seconds
F39	Repeat ON/OFF every 655.36 seconds
F40	Call LADDERINT in BASIC
F41	Reserved
F42	Reserved
F65 (F_ERR)	Error
F66 (F_<)	< Result of WCMP, DWCMP
F67 (F_>)	> Result of WCMP, DWCMP
F72 (F_CARRY)	Carry Flag
F73 (F_ZERO)	Zero Flag

MEMO

Index