

Preface

This book:

- is for first-time users of CUBLOC.
- avoided the usual manual approach; it has many figures and photos to make it fun and easy to approach CUBLOC. rather than listing commands as in dictionary entries, provides a thematic learning scenario where the reader learns the usage of a command.
- begins with a simple example of turning on an LED and gradually advances to a rather sophisticated example.
- One will especially begin with studying the basics of BASIC and Ladder Logic languages, then how to use them in specific programming situations.

It is the wish of the author that as many people as possible will become able CUBLOC programmers.



This book was written for :

- 1 Beginners of CUBLOC
- 2 Those who are familiar with PLC (Programmable Logic Controller) but new to BASIC
- 3 Who has been interested in Embedded Control
- 4 Those who are versed in electric circuits but weak at software programming
- 5 Who wants to control something

CUBLOC

Some Notes from the Author:

You need not know a thing about CUBLOC before you read this book; but you should be familiar with using PCs or Windows environment.

Since this book covers quite a lot about electronic circuits and components, those who are not familiar with those can easily read this book.

Most of the examples of this book assume you have the CUBLOC Start Kit 280.

Thus, we recommend that you purchase the Start Kit with this book.

You may not need to prepare other things before you dive into this book.

But you must be prepared with passion to dig in.

Now, let us lead you to the "World of CUBLOC"

ContentS

Chapter 1: What is CUBLOC?

006 CUBLOC?

Chapter 2: BASIC Programming

014 Lab 1: Twinkle, Twinkle, Little LED

Setting up CUBLOC Studio	14
How to use CUBLOC Studio	16
Troubleshooting: When downloading does not work	19

026 Lab 2: Turning on LED with a switch

CUBLOC Study Board 1: Circuit diagram	29
In-depth Analysis: How to use CUBLOC Studio	35

036 Lab 3: Sequential On-Off of LEDs

046 Lab 4: Digital Piano

058 Lab 5: Analog Input

070 Lab 6: Keypad Input

078 Lab 7: Counter Input

088 Lab 8: RS232 Communication

098 Lab 9: 7-segment Display

104 Lab 10: LCD Display

"Shall we start off, too?"



CUBLOC is an acronym for Comfile U(micro) BASIC Ladder On Chip,
which means it is Comfile's micro-chip with BASIC and Ladder Logic built in.



Part 01

What is CUBLOC ?

This chapter explains what CUBLOC is and why we should study it.

Chapter 1: What is CUBLOC?

CUBLOC?

It is a micro-computer developed by Comfile Technology, Inc. for embedded control.

The PCs we use in daily life is fit for home and office uses; but not for embedded control.

For instance, just imagine one would have to have a PC built into a microwave oven!! It would not only be too big, but it would be too expensive to sell.

So we need a much smaller computer than a PC, which has the minimal features to control devices such as microwaves and cars. CUBLOC is such a micro-computer packed with control features. CUBLOC supports Ladder Logic control language popularly used for internal control of devices, and also supports BASIC language to implement expanded functions.

Does that mean I have to study both? (whining :-)
No. Of course, it would be wonderful if you know both. But even if you know just one of these, you can use CUBLOC in many ways.

What is Embedded Control?

Embedded control is a CPU (Central Processing Unit) that is hidden inside a device/machine or under a hood to control every part of the device. Most electronic devices we use everyday has such an internal control device inside.

For example, products such as cell phones, washer and remote controls have an internal control device called Micro Controller or MCU (Micro Controller Unit).



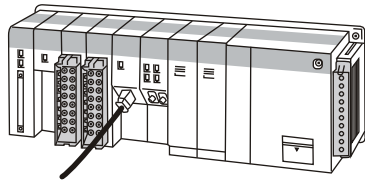
Internal Control using an MCU

MCUs are used inside mass-produced home appliances and communication devices. Though their price is relatively low, they are very difficult to develop.

Conventionally, to develop an MCU, one has to know Assembly Language and C Language, and needs equipments such as an ICE (In-Circuit Emulator) Programmer. Above all, one has to spend very many hours to understand the innards of an MCU. And even if one understands how an MCU works, to develop a working application source, one has to undergo numerous trials and errors. This is NOT a good way to develop something quick and easy.

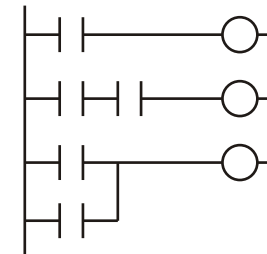
Internal Control using an MCU

PLC (Programmable Logic Controller) is an internal control device commonly used for factory automation and industrial machines. As shown in the figure below, a PLC has a terminal to connect sensors or motors to control relatively large equipments.

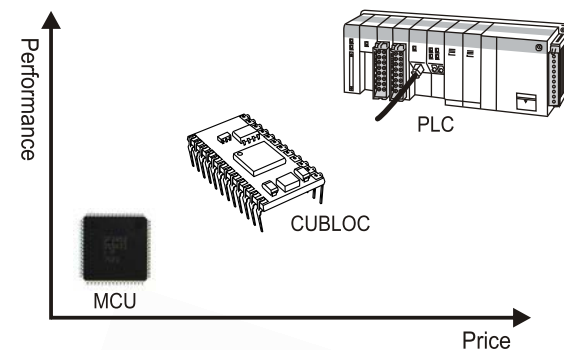


Also in elevators PLCs are used. The PLC in an elevator control all sorts of mechanical elements (e.g., motor and door) and collects various kinds of sensory inputs to move the users to wanted floors and open/close the door.

PLCs use a programming language called Ladder Logic, where through an RS-232 communication port a Ladder Logic program can be downloaded to a PLC from a PC and the PLC monitored on the PC.



LADDER LOGIC



Internal Control using a CUBLOC

CUBLOC is an embedded controller that lies between PLC and MCU in terms of features. It is small like an MCU that it can be installed on a PCB (Printed Circuit Board), and like a PLC a PC and download a program to it and monitor it.

It by default supports the Ladder Logic language as a PLC does, and in addition supports BASIC which is a sequentially processed programming language.

Thus, it can be seen as a new kind of embedded controller that took only the advantages from MCU and PLC.



Chapter 1: What is CUBLOC?

The Background behind CUBLOC's Birth

Long time ago (1996), we at Comfile Technology was making development equipments for MCUs.



PICBASIC



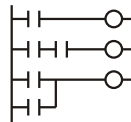
ICE



MCU

Since MCU is difficult to use and it was a controller only for experts, we created a product called PICBASIC. (1997)

PICBASIC spread like a wildfire throughout the automation field. Some of the PICBASIC users wanted a micro controller that could be controlled by Ladder Logic codes; so we made a product called TinyPLC. (1999)



Ladder Logic



TinyPLC Module

Since the TinyPLC users were mostly who worked in the factory automation field using PLCs, they wanted to use only Ladder Logic codes to implement complicated functions such as key input, display and communication. But such functions were not implementable using Ladder Logic codes only.



CUBLOC Core Module

On the other hand, PICBASIC users envied the Ladder Logic's capability to do real-time logic control. Thus, our company, Comfile Technology, finally came up with CUBLOC which combines the merits of the two sides: implementation of sophisticated functions and real-time control. (2004)

Application Areas of CUBLOC

CUBLOC can be used anywhere automatic control is needed because CUBLOC has all features of an MCU plus features of a PLC.

Yet, since it is not as cheap as an MCU, it cannot be used in mass-produced home appliances.

In general, it better be used for single product manufacture and low-volume production of products such as industrial machines, automation controllers, lab controllers, test jigs, etc.

In fact, using an MCU for such purpose is very cumbersome. (Of course, one will still have to use an MCU if there were no modular processor such as CUBLOC.)

But now with CUBLOC one can carry out the development in an easy and speedy manner.



Application Example



This is a real application made with CUBLOC.

a Perm Machine at a beauty parlor

For different perm styles, CUBLOC provides control for different perm temperatures and perm durations, respectively.

Since this deals with a field sensitive to the latest fad, the product allows for users to download the latest hairstyle program from the Internet.

For the development of this product, CB280, XPort and GHLCD were used.

Development Environment

You don't need special equipments to develop a CUBLOC application.

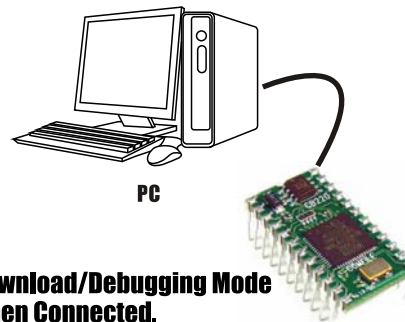
As with general PLCs, one just needs an RS232 cable to connect to a PC.

If your notebook computer does not have an RS232 port, you can use an RS232-to-USB adapter cable.

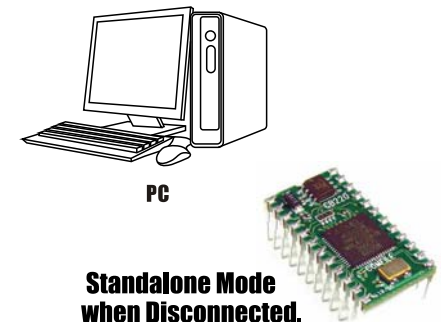


On your PC, you should install CUBLOC Studio, an integrated development environment (freeware) downloadable from Comfile Technology's website www.comfiletech.com.

When a PC and a CUBLOC are connected, they enter a mode where download and debugging become possible; when not connected, they enter the standalone mode.



**Download/Debugging Mode
when Connected.**



**Standalone Mode
when Disconnected.**

Chapter 1: What is CUBLOC?

How CUBLOC Works

The program (i.e., source code whether in BASIC or Ladder Logic) composed with CUBLOC Studio on the PC by the user is compiled into an intermediate language format. We call this middle language, Token.

This Token is downloaded through the RS232 cable to CUBLOC's flash memory.

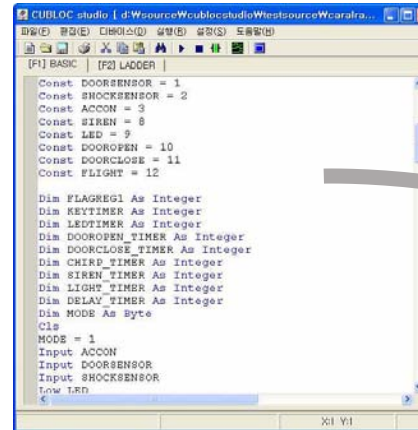
The main CPU (Central Processing Unit) on CUBLOC interprets this Token and carries out the commands.

The Token downloaded to a flash memory does not disappear even when the power is turned off because of the nature of flash memories. (We say that the flash memories are not *volatile*-i.e., its contents do not disappear when the power is turned off.)

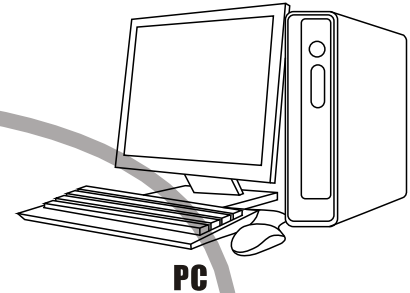
So the RS232 cable needs connect the PC and the CUBLOC; but once the application development is over, the cable can be removed.

One can use the final version of the downloaded Token on the CUBLOC. And when the application needs to be modified or updated, one can then re-connect the PC and CUBLOC with the RS232 cable for modification of the source code and re-compilation.

Source Code



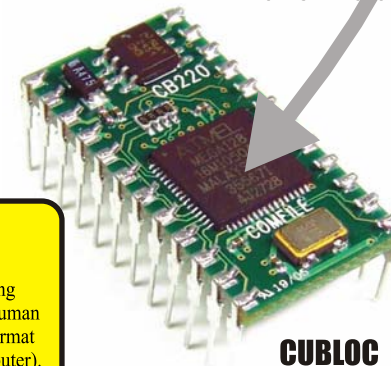
```
[F1] BASIC [F2] LADDER  
Const DOORSENSOR = 1  
Const SHOCKSENSOR = 2  
Const ACCOM = 3  
Const SIREN = 8  
Const LED = 9  
Const DOOROPEN = 10  
Const DOORCLOSE = 11  
Const FLIGHT = 12  
  
Dim FLAGREG1 As Integer  
Dim NEXTIMER As Integer  
Dim LEDTIMER As Integer  
Dim DOOROPEN_TIMER As Integer  
Dim DOORCLOSE_TIMER As Integer  
Dim CHIRD_TIMER As Integer  
Dim SIREN_TIMER As Integer  
Dim LIGHT_TIMER As Integer  
Dim DELAY_TIMER As Integer  
Dim MODE As Byte  
Cls  
MODE = 1  
Input ACCOM  
Input DOORSENSOR  
Input SHOCKSENSOR  
Low LED
```



PC

Token

98 ab 3c 12 03 cc
75 32 95 33 A2 00



CUBLOC



Smart Note:

Compiling a source code means translating the source code (which makes sense to human users) into a computer-understandable format (which is readily executable by the computer).



The Structure of CUBLOC

Please see the pictures on the right. CB220 is the core module. And the big chip in the middle is the CUBLOC's *main chip*.

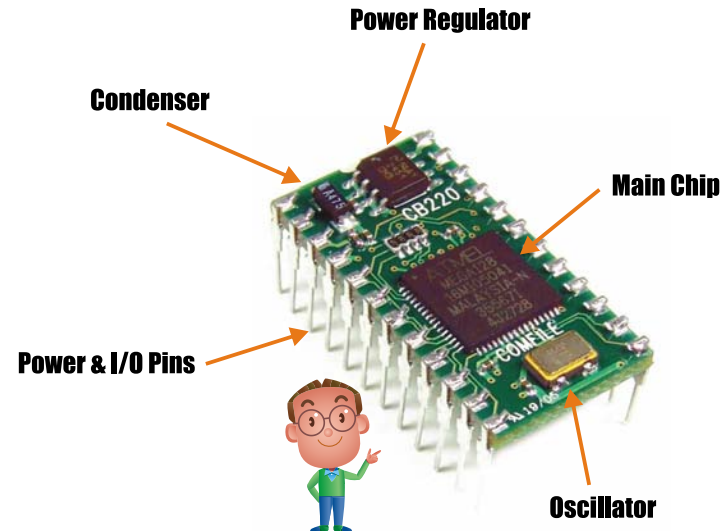
This chip is a "one-chip microcomputer" (meaning it has most components of a PC so it can run as a self-sufficient computer) which includes the OS (operating system) and the interpreter (which interprets the Token so that the commands can be carried out). The flash memory is also included in the main chip.

Other chips on the core module are the oscillator, the power regulator (which stabilizes the voltage), resistances, condensers, etc.

Most functions of CUBLOC comes from the main chip and other chips help the main chip carry out these functions.

To the power pin, 5V voltage is connected; and to the I/O pins the chips the user needs are connected, e.g., switch, relay, LED, etc.

Other CUBLOC modules (CB280 and CB290) differ only in terms of number of I/O pins and memory size. Their inner structures are either similar or the same.



CB220

I/O : 16
FLASH : 80K
RAM : 3K

CB280

I/O : 49
FLASH : 80K
RAM : 3K

CB290

I/O : 91
FLASH : 80K
RAM : 28K



This book mainly uses CB280.

Chapter 1: What is CUBLOC?

CUBLOC Studio

CUBLOC Studio is an *integrated development environment* software.

One can download the CUBLOC Studio program from Comfile Technology's website www.comfiletech.com in the *download* section.

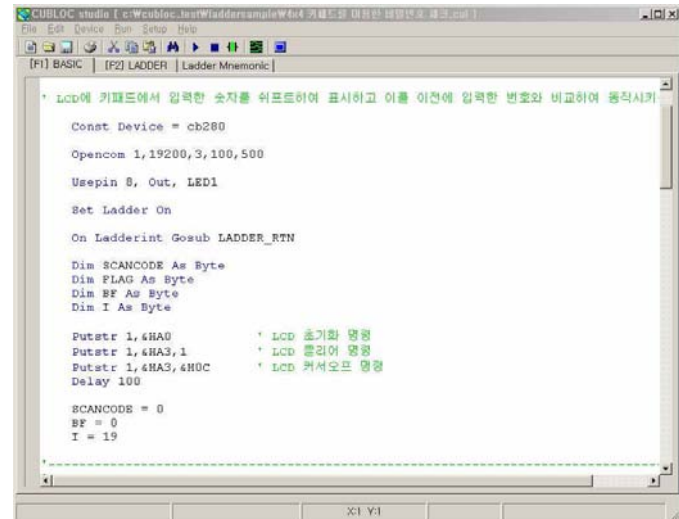
If you have purchased the Start Kit, you can install the Studio program from the included CD-ROM.

CUBLOC Studio is a freeware, which can be freely used and distributed. (Check back once in a while at the website for an updated version.)

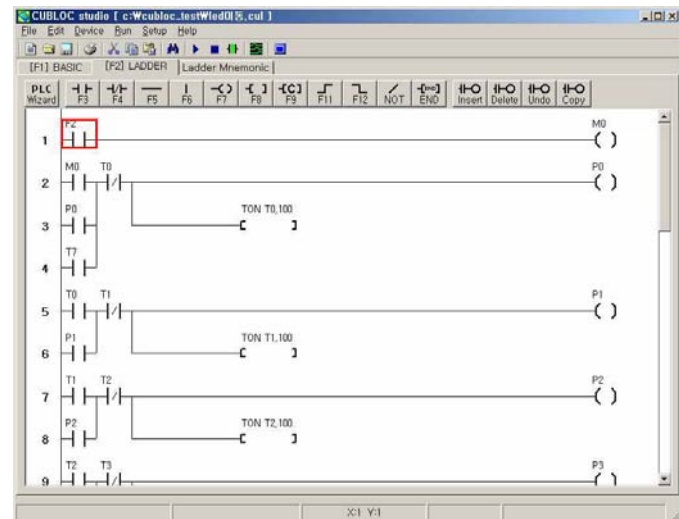
Since CUBLOC Studio integrates BASIC compiler, Ladder Logic compiler, editor and debugger, without other programs one can use CUBLOC.



Cubloc
Studio



BASIC editor screen



Ladder Logic editor screen

02

Part

BASIC Programming



In this chapter you will learn how to program in CUBLOC BASIC language (to control CUBLOC) through entry-level examples.

One who would like to start with Ladder Logic can read Chapter 3 first.

2 BASIC Programming

OPERATIONAL SUMMARY

The first program to compose is a simple program that will make an LED twinkle.

First, you need to install the Studio program on the PC. And you need to study how you can use the Studio program.

You also need to connect the PC to the CUBLOC with an RS232 cable, and design the circuit for this experiment.

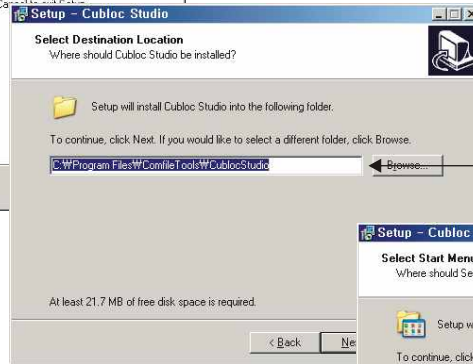
You will be learning what to do to utilize CUBLOC.

LAB. 1 Twinkle, Twinkle, Little LED

Setting up CUBLOC Studio



1 Start setting up by double-clicking the Setup file.



2 You may change the installation folder if you want; if not, just click Next.



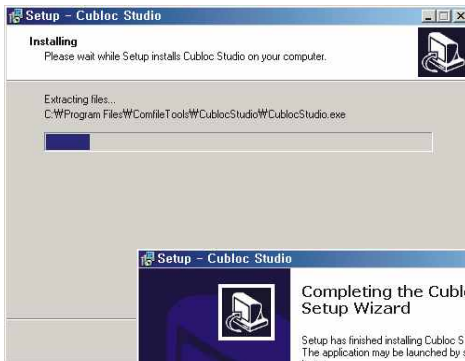
You can choose the name of the CUBLOC Studio folder that will be included in the PC's Startup menu.

3



If you check the checkbox, a shortcut icon of CUBLOC Studio will be installed on the desktop.

4



5 Installation is in progress.

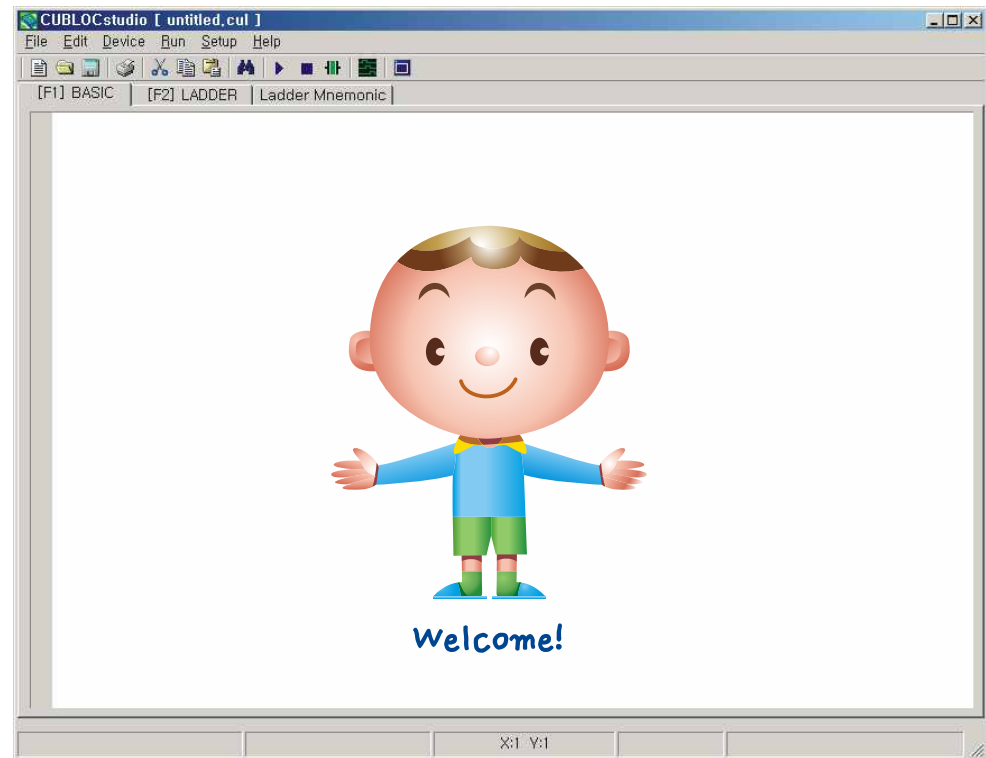


6 Installation is completed.



Cubloc Studio

7 If you double-click the CUBLOC Studio icon on the desktop screen ...



8 Ta-dah! CUBLOC Studio window appears!

How to use CUBLOC Studio

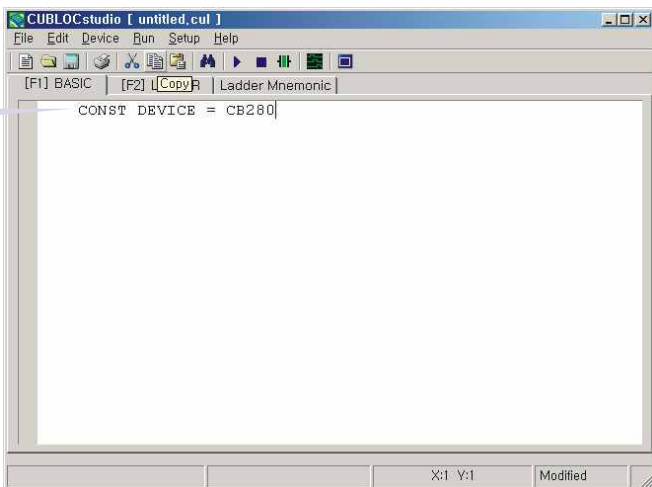
Let's learn the basics of using CUBLOC Studio.

It is very easy to learn how to use CUBLOC Studio to do BASIC programming.

Just think it as a word-processing program.

You first compose the source program, compile it and then you run it.

Let's first learn how to compose a source program.



Did you confirm that what you write is what you see?

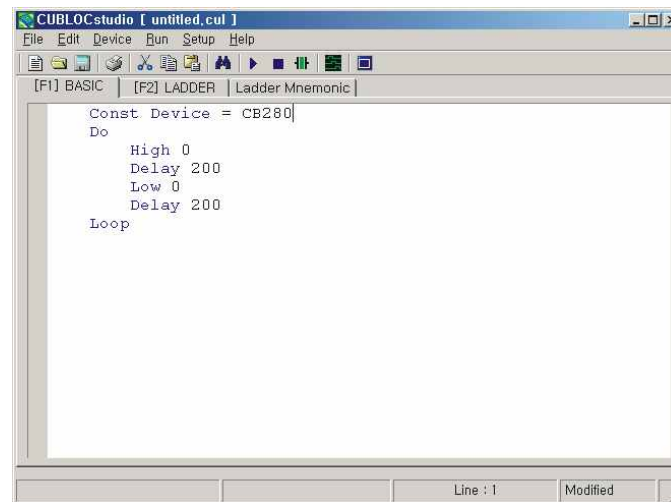
Think of CUBLOC Studio as a "word processor" and input any word.

If you just practice a little bit using the DEL key to delete characters and practice moving the cursor around using the arrow keys, you will soon be able to draft a wanted document (i.e., source program).

The BASIC program
you just composed is
called a "source program?"

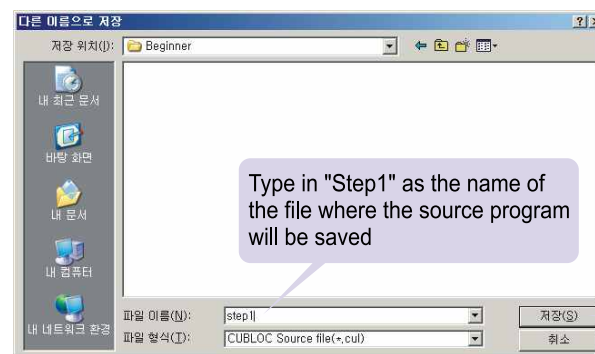
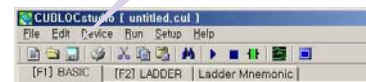


If you have finished practicing, now type in a source program as below:



If you finished typing it in, you should now save it.

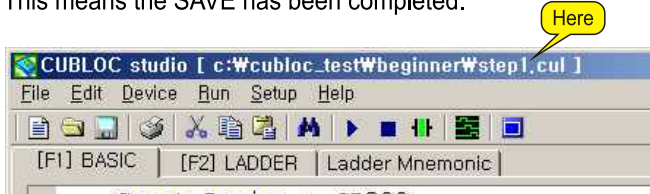
Click the icon shown here



Then, if you click here,
the file will be saved



If you look at the top of the screen, its labeling has been changed to step1.cul. This means the SAVE has been completed.

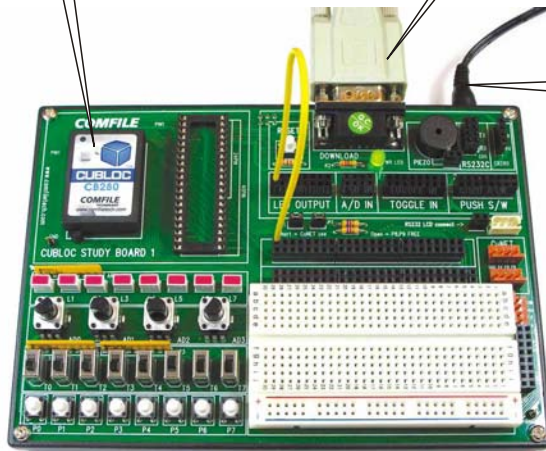


Next, we will check the hardware connection.

CUBLOC is installed here. Do not install it upside down.

RS232 cable

DC 9V adapter
Use the one whose center has the + polarity



Is your hardware (i.e., CUBLCO) ready as shown above?
After you power on the Study Board, check that the LED is turned on.



For this experiment, Port 0 of the CUBLOC has to be connected to LED's Port 0 with a wire.

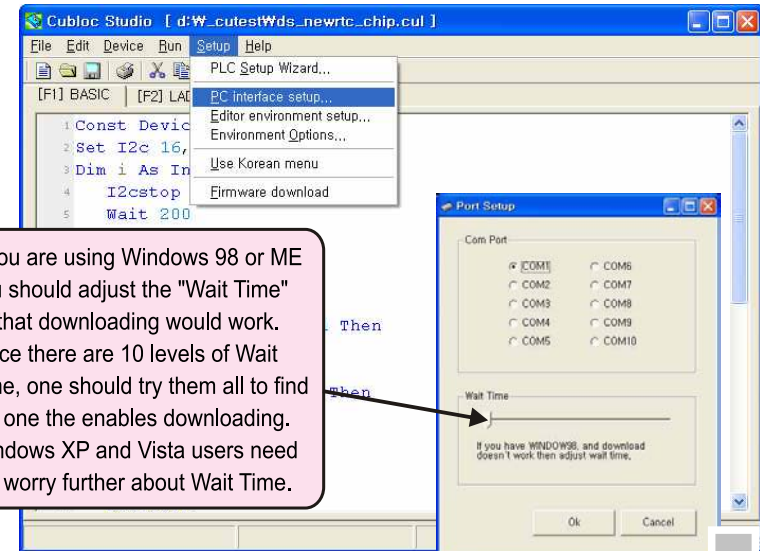
Look at the picture carefully and accurately insert the wire's both ends in correct positions.

The Study Board provides an environment where one can connect any two ports with a wire without soldering, which makes it really easy to do the experiments.



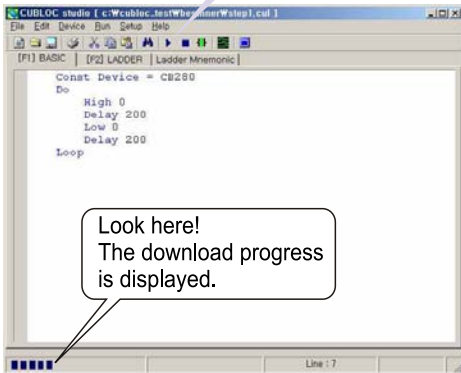
Once the hardware connection is ready, one needs to set up the PC interface.

Usually people use COM1 port, but if you would like to use another port, you can simply replace COM1 with your choice in the menu below.



If you are using Windows 98 or ME you should adjust the "Wait Time" so that downloading would work. Since there are 10 levels of Wait Time, one should try them all to find the one the enables downloading. Windows XP and Vista users need not worry further about Wait Time.

If you click the PLAY icon, download will begin.



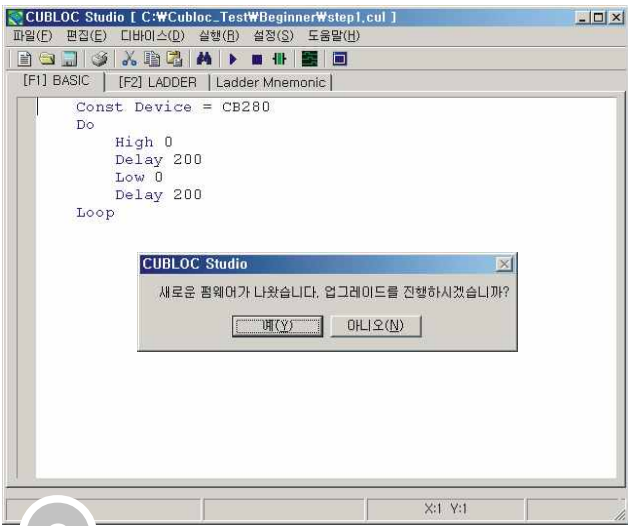
Look here!
The download progress is displayed.

You can also
press Ctrl + R!

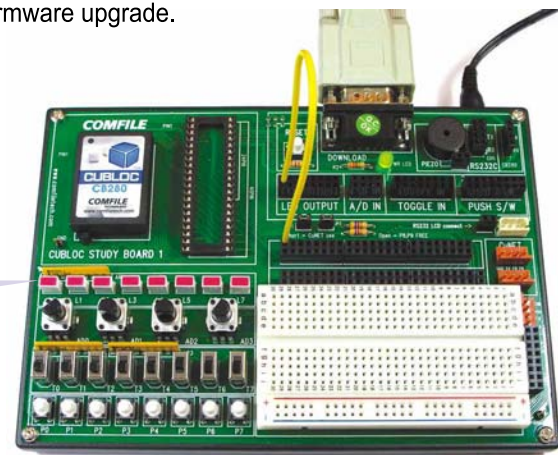


Once the file name is determined and file saved, the next time you press the PLAY button you don't have to give the file name again.

After you installed CUBLOC Studio, as shown below sometimes a popup window appears and asks whether you would like to upgrade your firmware. If you would like to upgrade, you should click "YES" and wait for the upgrade process to finish.



If the LED blinks, it indicates a success. Yet, if the LED does not blink, do not get disappointed but look for the cause. Wrong wire-connection would be most likely, but there could be other reasons. If it still does not work, try it again after a firmware upgrade.



This should blink

Wah!
I Succeeded!!



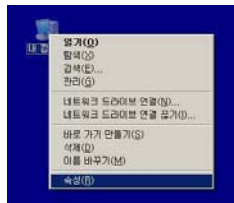
Firmware Upgrade is...

a process of replacing the interpreter program and the OS (operating system) stored inside CUBLOC to improve its functions or make up for some of its lacking aspects.

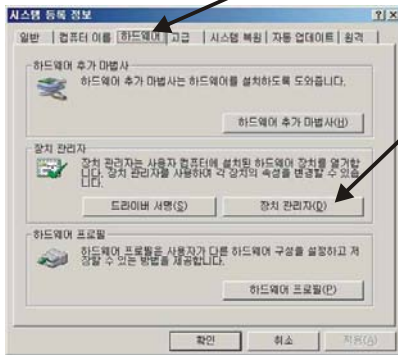
Troubleshooting

Download would not work?

Then first check Windows' communication port driver by taking the following steps:

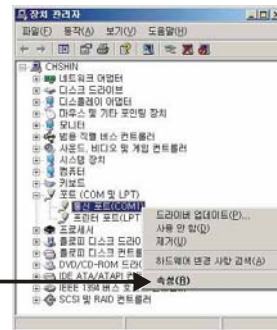


- 1 Click "My Computer" icon on the Desktop with the right mouse-button and when the menu window appears, select the Properties menu item.



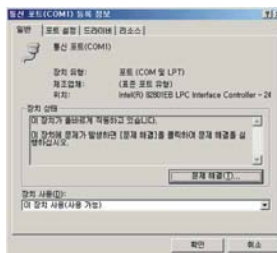
- 2 Select the "Hardware" tab on the pop-up window.

- 3 Click the "Device Manager".



On the Device Manager window, click "COM & LPT", and click the "COM1" port under it with the right mouse-button. On the pop-up window, click "Properties".

- 4



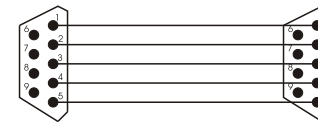
On the pop-up window's "General" tab, if it says "This device is working properly."

- 5

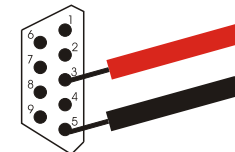
Now, let's learn how to check an RS232 cable to see if it works properly.

First, you should always use a 1:1 RS232 cable.

There won't be a problem if you are using the cable that came with the Start Kit; but if you are using one made by you or one used with another equipment, you should check to make sure it is a 1:1 cable.



Look at the DSUB connector that connects to the Study Board. If you look at the inside carefully, you will see small numbers next to the holes.



While connected with a PC, when the 5th (GND) and the 3rd (TX) terminals the voltage difference should be -10 to -12 V.

If it is a wrong cable or there is a breach inside, a wrong voltage will be produced.



then the Windows driver is indeed working properly. Otherwise, consult a computer expert around you or solve this problem by reading a related book.

```

Const Device = CB280  ← Declaration of the CUBLOC model number
Do
  High 0  ← Makes Port 0 high
  Delay 200 ← 200 milliseconds delay
  Low 0  ← Makes Port 0 low
  Delay 200
Loop  ← Repeat the commands between DO and LOOP infinitely.

```

The first line declares the Model number of the CUBLOC being used. You just need to write down the model number of the CUBLOC model you are using.

Since we are running an experiment with CB280, you should write:
CONST DEVICE = CB280

Next comes the DO-LOOP structure. This is a conditional loop structure. Since there is no condition to satisfy, it repeats the enclosed commands indefinitely.

There are HIGH and LOW commands within the loop. The HIGH command puts a port in a high state, and the LOW command puts a port in a low state. As the commands are executed, the ports takes on the specified output states.

Lastly there is a DELAY command. As its name says, it adds a time delay. Since the execution speed of CUBLOC is so fast that, if one specified only HIGH and LOW commands, it will not give enough time for a person to see the LED's blinking. Therefore, we need to extend the time of both HIGH and LOW states by giving small delays. The number given after the DELAY command the amount of delay given in milliseconds.



NEW COMMANDS

Const Device

CONST DEVICE = device name

This command should come first at the beginning of a source code to declare the device name. Device names such as CB220, CB280, CB290, CT1720, etc. can be used.

Do, Loop

Do while [condition]
Command
[Exit Do]
Loop

Do
Command
[Exit Do]
Loop While [condition]

The DO-LOOP statement repeatedly executes the included commands while the condition is true. When the condition is specified after the LOOP command, the condition is tested after the enclosed commands are executed. If an EXIT DO phrase is used within the loop, the execution stops there and exits the loop. If the DO-LOOP construct is used without a condition, it becomes an infinite loop.

Do Until [condition]
Command
[Exit Do]
Loop

Do
Command
[Exit Do]
Loop Until [condition]

This variation executes the loop while the condition is false. The following are examples.

Do while A=0
 LOW 8
 HIGH 8
Loop

Do
 LOW 8
 HIGH 8
Loop Until A<>0



HELPFUL TIPS

CUBLOC's I/O Ports

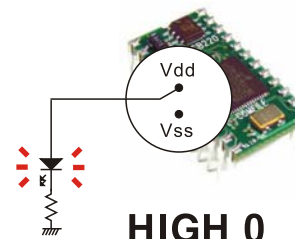


SOUT	1	•	•	17	VDD	TX1	33	•	•	49	TTLTX1
SIN	2	•	•	18	VSS	RX1	34	•	•	50	TTLRX1
ATN	3	•	•	19	RES	AVDD	35	•	•	51	AVREF
VSS	4	•	•	20	N/C	N/C	36	•	•	52	P48
P0	5	•	•	21	P16	P24	37	•	•	53	P31
P1	6	•	•	22	P17	P25	38	•	•	54	P30
P2	7	•	•	23	P18	P26	39	•	•	55	P29
P3	8	•	•	24	P19	P27	40	•	•	56	P28
P4	9	•	•	25	P20	P47	41	•	•	57	P32
P5	10	•	•	26	P21	P46	42	•	•	58	P33
P6	11	•	•	27	P22	P45	43	•	•	59	P34
P7	12	•	•	28	P23	P44	44	•	•	60	P35
P8	13	•	•	29	P15	P43	45	•	•	61	P36
P9	14	•	•	30	P14	P42	46	•	•	62	P37
P10	15	•	•	31	P13	P41	47	•	•	63	P38
P11	16	•	•	32	P12	P40	48	•	•	64	P39

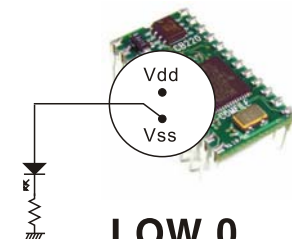
Most of the pins on CB280 are I/O pins. For CUBLOC, the I/O pins handle the communication with the external devices. If the CPU were man's brain, the I/O pins correspond to man's limbs.

The I/O ports can deal with either inputs or outputs, which is why they are called "I/O", an abbreviation of "Input/Output". When CUBLOC is initially powered on, all I/O ports become input ports. Only after a BASIC command such as HIGH or LOW is used, any specified port is readied with an output value.

Port 0 is somewhat special. When it is made high using the "HIGH 0" BASIC command, it becomes (internally) connected to V_{dd} (5V). On the other hand, if a "LOW 0" command is issued, Port 0 is connected to Gnd (V_0).



HIGH 0



LOW 0

High

High port

This command makes the output of the specified port high.
Example: High 8 'Port 8 outputs high state.'

Low

Low port

This command makes the output of the specified port low.
Example: Low 8 'Port 8 outputs low state.'

Delay

Delay time

This command delays the execution of CUBLOC by specified time. Since the execution speed of CUBLOC is very fast, sometimes its execution needs to be delayed deliberately. The time is in milliseconds. For example, if 500 is specified, it denotes a delay time of 500 milliseconds. But, since this does not means exactly 500 milliseconds will be delayed, this is not an appropriate command for rigorous time control.

Example: Delay 500 '500 milliseconds, i.e., the execution is delayed by 0.5 second.'

One can use the PAUSE command instead of DELAY. Pause is synonymous to Delay.

Example: Pause 100 '100 milliseconds, i.e., the execution is delayed by 0.1 second.'

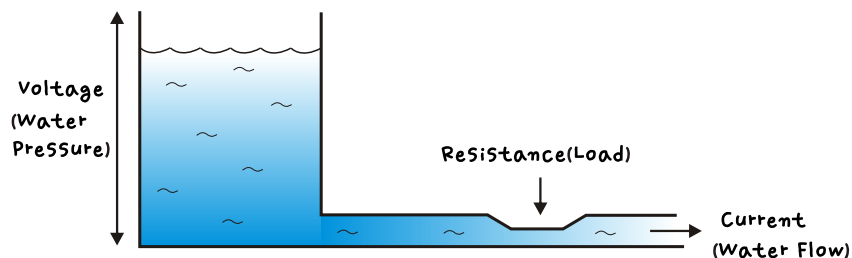
Let's get some Hardware Basics!

For those who are newbies in electronics, everything will be unfamiliar. But if you progress step by step, nothing will be too difficult. Don't worry because the electronics knowledge dealt with in this book is very elementary.



HELPFUL TIPS

Voltage, Current & Resistance



Commonly, electric current is compared to water flow.

The above figure shows water flowing out through a hose at the bottom of a container. The more the water in the container, faster the water flow is.

This is why more current flows when the voltage is higher.

If something hinders the flow of the water, the water will not flow as freely without the hindrance. Such hindrance is resistance (or load).

If resistance hinders the electric flow, some might think, 'Then why do we need it?' Some chips of an electric circuit should only get specified current. If they get more than needed, they burn out or become dysfunctional. Resistance comes handy in such situation to regulate the maximum current that can be supplied to such chips.

Therefore, resistance is an indispensable element of a circuit.

PARTS STORY

RESISTANCE

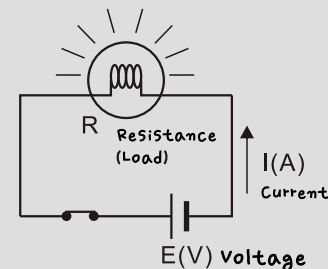
Resistance hinders the flow of the current.

Thus, bigger the resistance, worse the current flows.

When electric current is put on a resistance element, difference of voltage occurs between the two ends of the resistance.

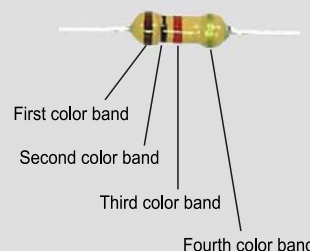
This can be explained by the famous *Ohm's Law*.

$$R = \frac{E(V)}{I(A)}$$



Resistance value is denoted by color bands.

The 3 colors of the 4 colors (excluding the gold color) denote resistance values. The first two colors denote assigned digits and the third color denotes the exponent of 10.



	First	Second	Third	Fourth
Black	0	0	10 ⁰	
Brown	1	1	10 ¹	
Red	2	2	10 ²	
Orange	3	3	10 ³	
Yellow	4	4	10 ⁴	
Green	5	5	10 ⁵	
Blue	6	6	10 ⁶	
Violet	7	7	10 ⁷	
Grey	8	8	10 ⁸	
White	9	9	10 ⁹	
Gold			0.1	± 5%
Silver			0.2	± 10%

Example: If the band colors were brown, black, red and gold from the left, this will denote $10 \times 10^2 = 1K \text{ Ohms}$.

"It is convenient to memorize the colors in order as BBRO YGBV GWGS."

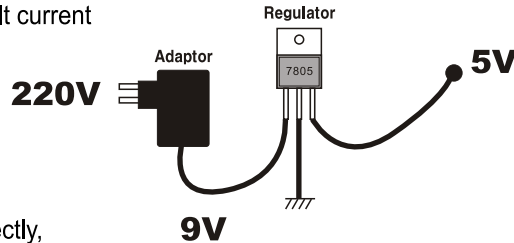




HELPFUL TIPS

Obtaining 5 Volts

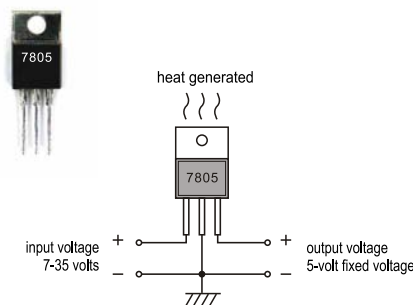
To use CUBLOC, one needs 5-volt current. To get it, one should first obtain 9V DC (Direct Current) current from 220V AC (Alternating Current) by using an adaptor; then the 9-volt current should be lowered to 5-volt current using a voltage regulator.



Aren't you curious as to why the voltage is not lowered from 220 to 5 directly, but done so via 9 volts?

The reason is to have a buffering stage in the middle to get a stable voltage. The 9-volt DC adaptor often outputs voltages varying from 8 to 12 volts. When the unstable voltage passes through the regulator, it is converted to a stable 5-volt current.

One of the representative 5-volt regulators is a chip called 7805. When a circuit is configured as shown in the next figure, one can obtain a stable 5-volt output. Since the 7805 chip generates heat in this process, one needs to heat sink.



There are other 5-volt fixed-voltage generators besides 7805 chip.

Since the Study Board also has a fix-voltage current circuit to supply stable 5-volt current.

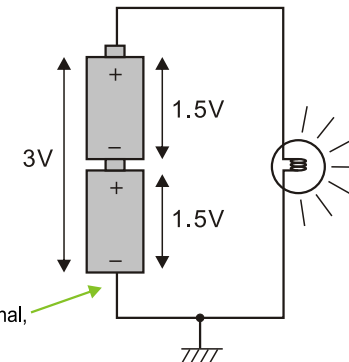


HELPFUL TIPS

The Identity of a Ground

GND (as an abbreviation of Ground) denotes the base voltage, i.e., 0-volt state. When we talk about a mountain's height, we say "elevation XX meters". Here the "elevation" means "elevated from the sea-water surface level."

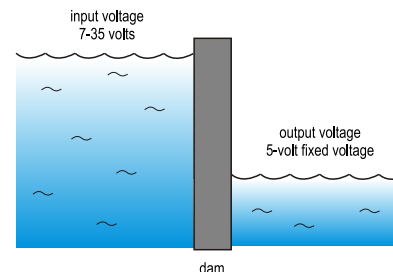
Similarly, when we say "5 volts", it means "5 volts from ground voltage". The ground on batteries is the terminal denoted by the - (minus) sign.



The GND sign looks like this.



Here is the 0V terminal, i.e., GND.



7805 chip can be called an "electronics dam"

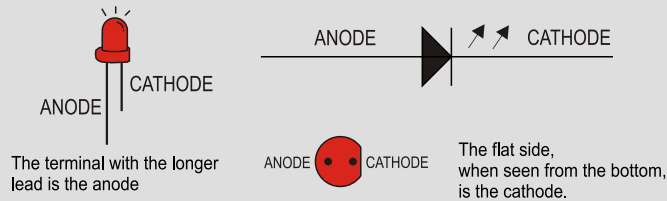
PARTS STORY

LED

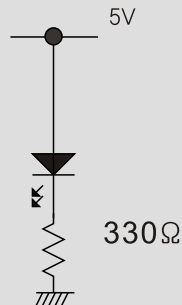
LED (pronounced L.E.D.) stands for light-emitting diode. One can easily find LEDs such as an LED lamp or a "7-segment" LED which displays an alphanumeric figure.



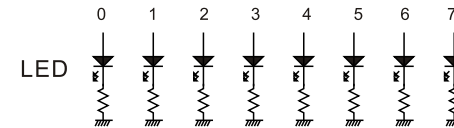
Its operating principle is very simple. When one applies 2 volts to the + terminal (called "anode") and GND to the terminal (called "cathode"), the LED turns on.



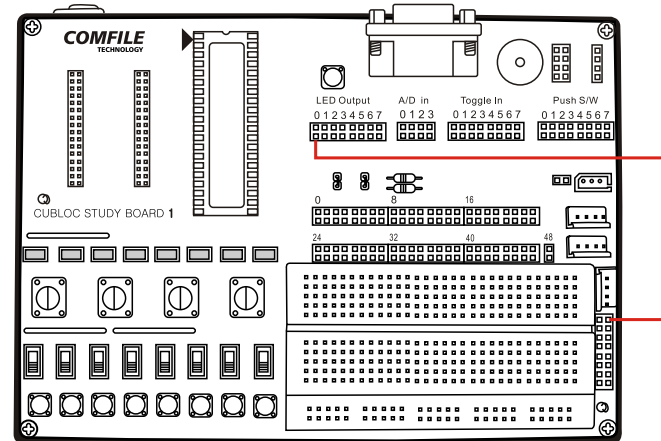
Then what should one do to operate an LED with a 5-volt current? This is when a resistance plays an important role, since an LED only needs 2 volts. When one serially connects a resistance to a 5-volt current, the resistance will consume 3 volts and convey the remaining 2 volts to the LED. In this case, a resistance of about 330 ohms is appropriate.



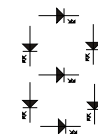
Since CUBLOC's Study board already has resistances and LEDs built-in as in the next figure, one only needs to connect the LED terminals to a voltage source using wires.



The Study Board has 8 LEDs. Change the wire connection to turn on different LEDs. (Refer to p. 17.)



"Seven segments" is a device consisting of 7 LEDs deployed in an 8 shape. Imagine which LED segments should be turned on to depict each digit (0 to 9).



0 1 2 3 4 5 6 7 8 9



Well done!

There is a saying: To begin is half done.
If you have succeeded in the first CUBLOC project,
this means you have resolved many issues by yourself.
I hope you run all the way to the "finish line"
at the current pace.



APPLIED TASK

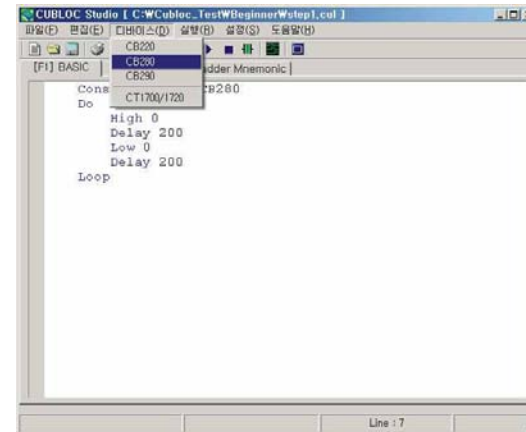
How about modifying the given source program by yourself?
Here I pose some tasks for you. Are you ready for the challenges!

- 1 Change the source code so that the flashing period of the LED is shortened or lengthened.
- 2 What should you do to turn on and off the LED every 1 second?
- 3 Make 2 LEDs flash at the same time.
- 4 Make 2 LEDs flash alternatingly.
(I.e., one should be on when the other is off, and vice versa.)



If you choose a device model from the DEVICE menu, a corresponding "CONST DEVICE = ..." statement will be inserted at the beginning of the source code you are composing.

If there is a CONST DEVICE statement already, just its device name will be replaced with the one you chose.



COMMANDS LEARNED IN THIS CHAPTER

CONST DEVICE
DO, LOOP
HIGH, LOW
DELAY

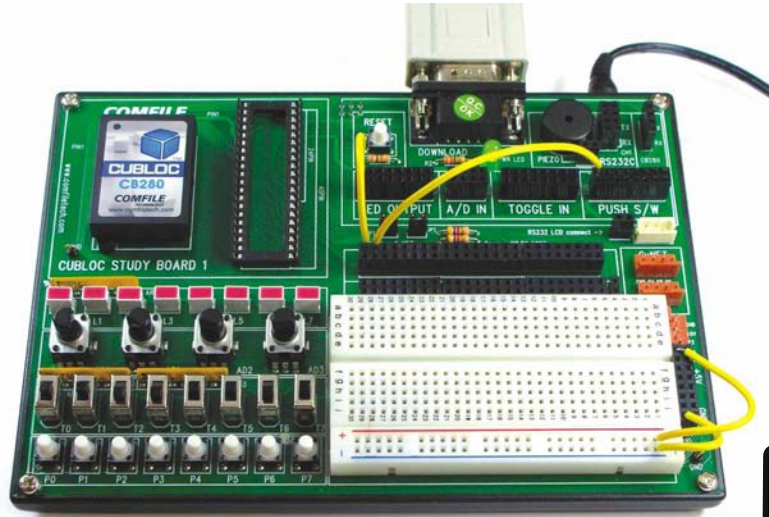
OPERATIONAL SUMMARY

1. Compose a program that will turn an LED on and off using a switch.
2. Learn about digital states of I/O ports.
3. Also learn about the roles of variables in BASIC, and how they are declared and used.

LAB.2 Turning and LED on with a switch

Circuit Configuration

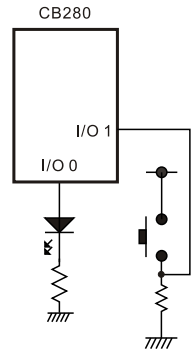
Connect the CUBLOC's I/O Port 1 to the switch and I/O Port 0 to an LED.



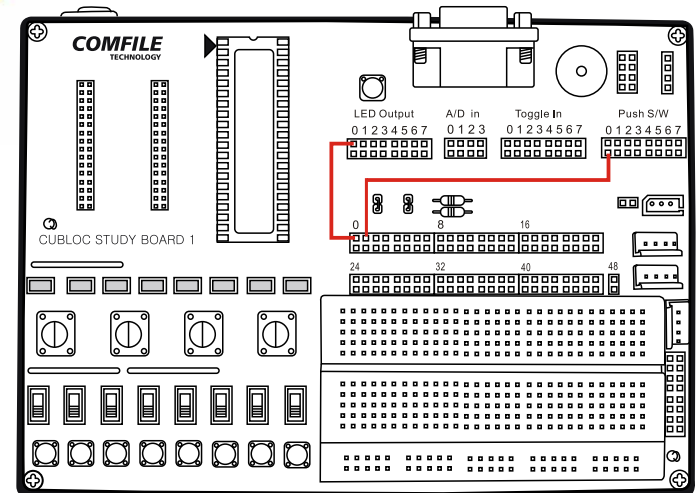
Circuit Diagram

The circuit configuration, when drawn as a circuit diagram, looks like the next figure.

Only 2 I/O ports of CB280 are used; the rest are not used.



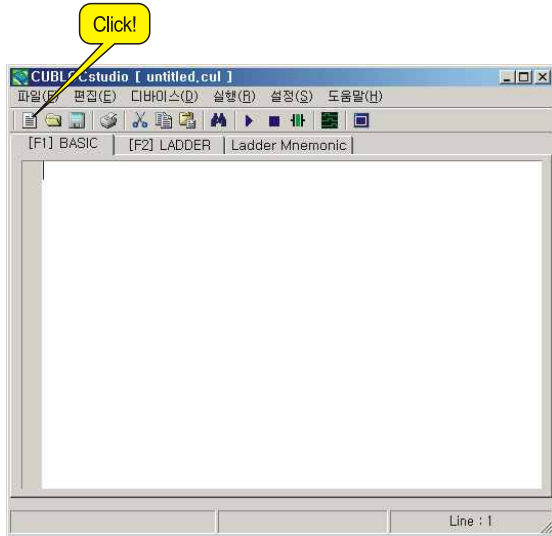
Make the wire connection as the red line indicate in the next figure!



Inputting the Source Program

We will input the source program.

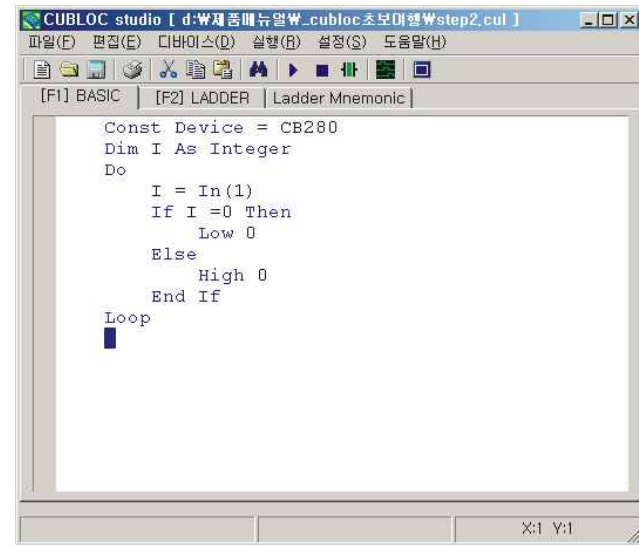
Do you by any chance see the previous program loaded in your programming window? If so, click the NEW button to completely delete the previous program and input the new code.



Whenever you push the RUN button, automatically the current program code will be saved with the given file name. Only when a file name is not give would you be asked for a file name for saving!

Now enter the source code.

It is important that you do not make any typos while entering the code.



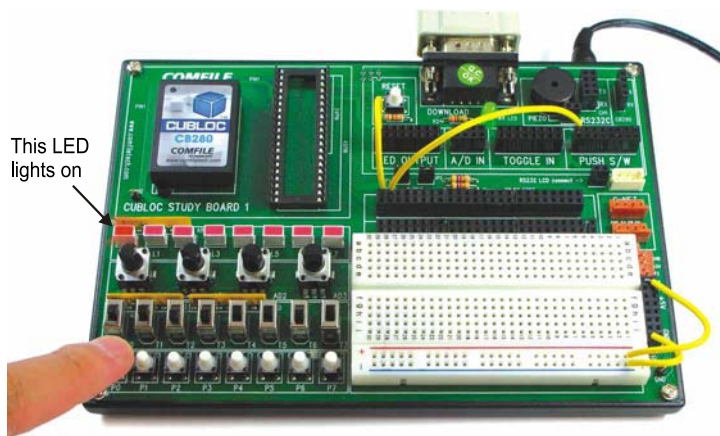
If you have finished inputting the source code, click the RUN button! A pop-up window will appear! Type in STEP2 as the new file name, under which to the program file will be saved.



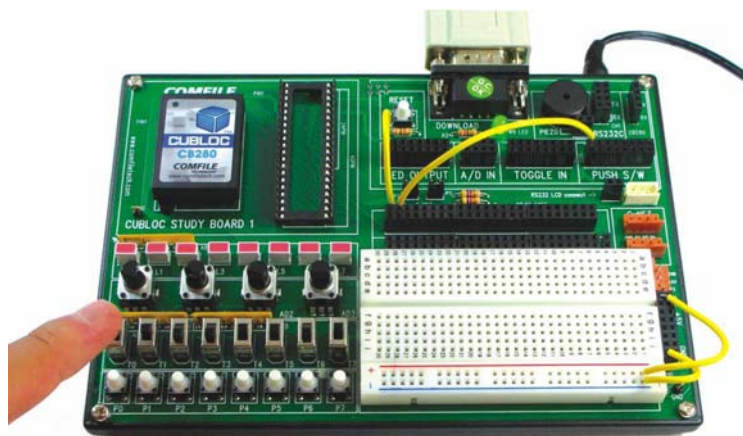
When saving is completed, the program will be automatically downloaded and run.

Confirmation Run

Let us confirm whether the program runs correctly.



▲ When the switch is pushed, the LED is turned on.

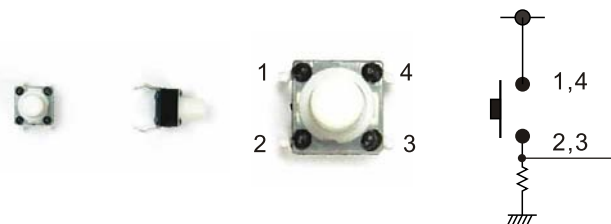


▲ When the switch is released, the LED is turned off.

PARTS STORY

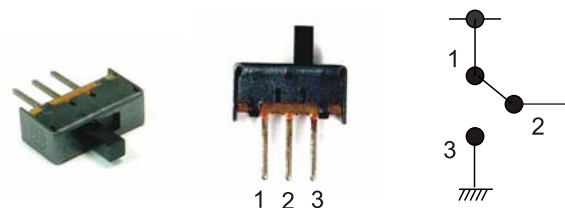
Switch

There are two kinds of switches on the Study Board. One is push switch, where the contact is made only while the switch is pushed down.



The Study Board is designed so that a voltage of HIGH (5 V) will be applied only during the switch is pushed down.

The other kind is toggle switch, where the changed state is maintained once the switch is toggled, i.e., either in contact or separated.



The Study Board is designed so that a HIGH (5 V) voltage state is maintained when a toggle switch is switched on; and a LOW (0 V) state is maintained when the toggle switch is switched off.

Depending on the situation, a push switch is needed or a toggle switch is necessary. The push switch is also called "tact switch."

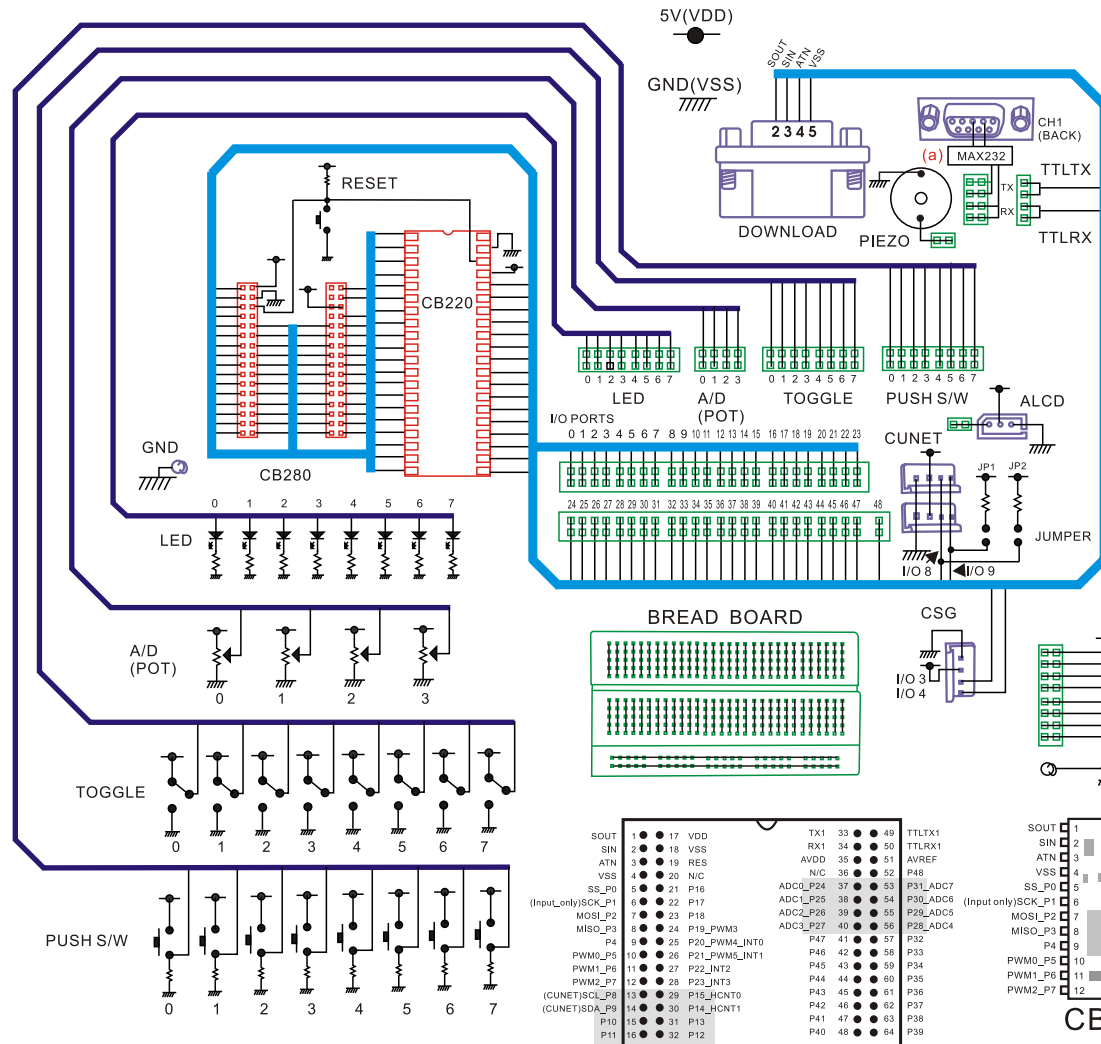
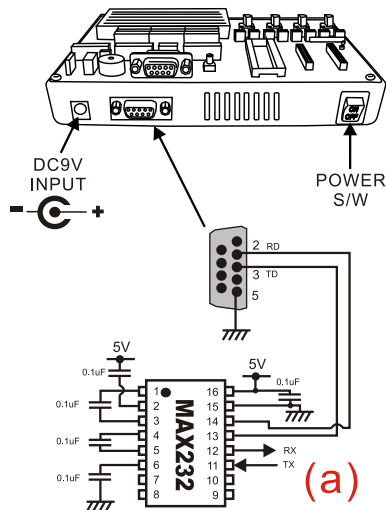


CUBLOC Study Board-1 Circuit Diagram

To experiment with only the core module of CUBLOC, one needs a board equipped with a basic circuit.

And one also needs I/O devices such as LEDs and switches.

A single board that meets all these basic requirements is a Study Board.



CB280

CB220

Explanation of the Source Program-1

```

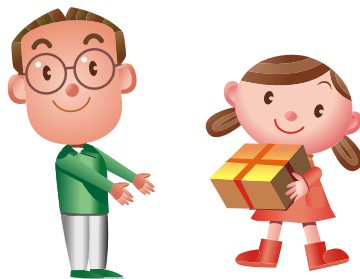
Const Device = CB280  ← All programs should start with a "Const Device" statement.
Dim I As Integer  ← Declare variable I as an integer.
Do
    I = In(1)  ← Store the value of Port 1's input state at variable I. (0 or 1 will be stored.)
    If I = 0 Then  ← If I value is 0 (LOW)...
        Low 0  ← Make Port 0's state LOW (0).
    Else  ← Otherwise, i.e., if I=1 (which means Port 1 value was HIGH)
        High 0  ← Make Port 0's state HIGH.
    End If
Loop

```

In BASIC programming language, a variable is used to store a value temporarily. But to start using a variable as a storage place, one has to secure its storage space first! It is the DIM command that used for such purpose. When DIM command is used to declare a variable's name and its type, CUBLOC allocates an appropriate storage space for the variable.



Please use the DIM command before you keep a value with us!



NEW COMMANDS

Dim

DIM

Format: DIM <variable> AS <variable type>

This command declares a new variable. With CUBLOC one can declare one of the 5 variable types.

BYTE : 8-bit unsigned integer (represents 0 to 255)
 INTEGER : 16-bit unsigned integer (represents 0 to 65535)
 LONG : 32-bit signed integer (-2147483648 to +2147483647)
 SINGLE : 32-bit real number (-3.4×10^{38} to $+3.4 \times 10^{38}$)
 STRING : a series of characters (up to 127 characters (bytes))

An integer is a number without the fraction part. E.g., 12 is an integer.
 A real number is a number with the fraction part. E.g., 3.14 is a real number.
 A string is a train of characters. E.g., "ABCDE" is a string.

The only variable types that can store negative numbers are:
 LONG and SINGLE. BYTE and INTEGER can only store positive numbers.

A variable name must start with an Alphabet character.
 Example: Dim ABC as Integer

When declaring a string type, the maximum storage capacity should be specified.
 Example: Dim STR as String * 12 ' 12-byte storage space is secured.

If the maximum storage capacity is not specified when declaring a string type, by default 64 bytes will be allocated.
 Example: Dim STR as String ' 64 bytes of storage space is allocated.



Q: Can infinitely many variables be declared with CUBLOC?

A: No. As CUBLOC's storage space is finite, variables cannot be declared infinitely. The memory size differs among different CUBLOC models. CB280 has the maximum storage space of 1948 bytes.



HELPFUL TIPS

Decimals(Digits) and Hexadecimals

The numbers we use in everyday life is decimals.

The units we use for counting money is decimal system and we also use it when we say, "The height is 182 centimeters, and the weight is 80 kilograms."

Decimals consist of numbers from 0 to 9.

But the numbers computers use are 0 and 1, i.e., computers use binary counting system. They use only 0s and 1s to make a number.

A binary number looks like: 10101110101010.

Since binary numbers are too hard for humans to grasp, people came up with a new scheme to read 4 binary places as one chunk. A 4-place binary number makes a one-place hexadecimal (16) number. Hexadecimal consist of numbers from 0 to 9 and alphabets A to F, a total of 16 symbols. Examples of hexadecimal numbers are: 1AEF and FFFF.

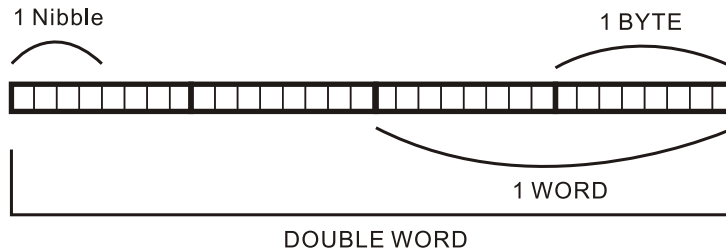
Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



HELPFUL TIPS

Byte and Bit

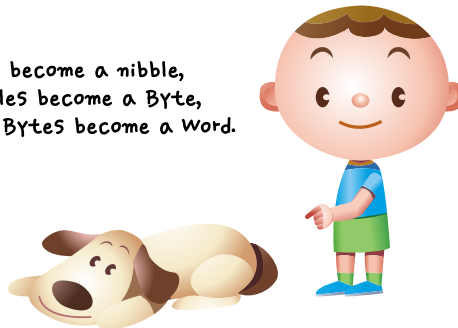
The smallest unit of information storage is called bit, which can store 0 or 1. 8 bits are called a Byte, and though not used frequently, 4 bits are called a nibble.



16 bits are called a Word and 32 bits are called a Double Word.



Bits become a nibble,
nibbles become a Byte,
and Bytes become a Word.



Explanation of the Source Program-2

```
I = IN(1)
If I =0 Then
    Low 0
Else
    High 0
End If
```

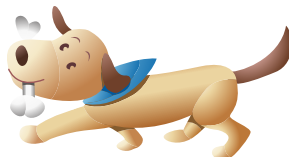
IF statement checks a condition and, depending on the result, selects a statement or statements to run. For example, in the above IF statement, if the I value is 0, Low 0 is executed; otherwise, High 0 is executed. That is, if the condition is tested to be true, the code between THEN and ELSE is executed; if false, the code between ELSE and END IF is executed.

Since I stored the value of In(1), either Low 0 or High 0 is executed depending on the In(1) value.

In is a function that changes a specified pin to be in input state and reads the state of that I/O pin. Thus, the result is either 0 or 1.

In the case of the previous circuit, since a switch is connected to Pin 1, In(1) eventually means reading the value of the switch's state.

Consequently, the above code sets Pin 0 to LOW or HIGH depending on the switch's state.



NEW COMMANDS

IF

Depending on the truth value of the conditional expression, which statement(s) should be executed is determined.

IF statements can be expressed in many forms.

IF statement format 1

IF <condition> THEN <statement>

If <condition> is true <statement> is executed. If not, nothing happens.

IF statement format 2

IF <condition> THEN <statement1> ELSE <statement2>

If <condition> is true, <statement1> is executed. If not, <statement2> is executed

IF statement format 3

**IF <condition> THEN
 <statement>**

END IF

If <condition> is true <statement> is executed.
If not, nothing happens.

"END IF" *must* be used, and nothing should be written after THEN.

IF statement format 4

**IF <condition> THEN
 <statement1>**

ELSE

<statement2>

END IF

If <condition> is true, <statement1> is executed.
If not, <statement2> is executed.

"END IF" *must* be used, and nothing should be written after THEN.

IF statement format 5

**IF <condition> THEN
 <statement1>**

**ElseIF <conditions2> THEN
 <statement2>**

END IF

If <condition1> is true, <statement1> is executed. If <condition1> is true,

<statement2> is executed.

One can use multiple ELSEIFs.



In()

<Variable> = In(<Port no.>)

This function makes the specified port into an input port, reads the state of the I/O pin of the Input port, and stores it in the <Variable>.

Example: A = In(1) 'Read the state of I/O port 1 and store it in variable A.

The variable to store the result (state) should be of one of integer types: Byte, Integer or Long.

The state value can't be stored in SINGLE or STRING. type variable.

The result value is either 0 or 1.

A job well done

We have studied about digital inputs after covering digital outputs.

Also, we covered declaration and usage of a variable, which is important in BASIC language. We even cover IF Statement.

Since these commands are frequently used in BASIC, make sure you master it before you go to the next Section.



APPLIED TASK

- 1 Last time, you have made a program that turns on an LED when a push switch is pushed. This time, make a program that does the opposite, i.e., turns on an LED when the switch is released and turns off while the switch is pushed down.
- 2 Make an LED flash while the switch is pushed down.
- 3 Make all 8 LEDs turn on when the switch is pushed down.
- 4 Do the above [1], [2] and [3] this time using a toggle switch.

COMMANDS LEARNED IN THIS CHAPTER

DIM
IF
IN



IN-DEPTH STUDY OF COMMANDS: Array Declaration using DIM

In the previous chapter, we have learned how to secure memory space to store variable values. How should we do so for a series of values?

```
Const Device = CB280
Dim A1 as Integer
Dim A2 as Integer
Dim A3 as Integer
Dim A4 as Integer
Dim A5 as Integer
Dim A6 as Integer
Dim A7 as Integer
Dim A8 as Integer
Dim A9 as Integer
Dim A10 as Integer
Dim A11 as Integer
```

If we do so as above, we will soon face a limit because we will have to type in 100 declarations for 100 variables! Thus, in BASIC language, there is a feature called Array to secure multiple, contiguous memory space.

```
Const Device = CB280
Dim A(12) as Integer
```

The above declares A(0) through A(11) with one statement, where each array element, e.g., A(0), can be used like a variable.

Other merit of an array is that a variable can be used as the index of the array. E.g., in A(B), a variable can be used instead of 0. For example,

```
Const Device = CB280
Dim A(12) as Integer
Dim B as Integer
B = 0
A(0) = 12
A(B) = 12
```

the last two statements has the same effect because B has the value 0.

Thus, the use of an array can greatly facilitate storing or reading a series of values into or out of variables.

When the declaration "Dim A(12) as Integer" is made, array elements A(0) to A(11) are created. Beware that A(12) is not created and does not exist.

One can also declare a 2-dimensional array such as in
Dim A(12, 8) as Integer

A(6)

In that sense, "Dim A(12) as Integer" declared a one-dimensional array.

In a similar manner, one can also declare a 3-dimensional array, e.g., "Dim A(12, 8, 6) as Integer"

A(3,6)

A(3,3,6)

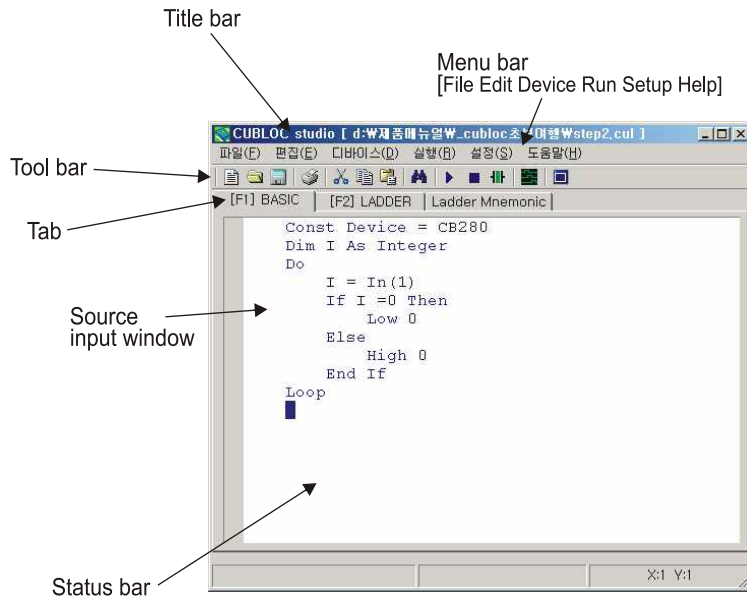
In CUBLOC, one can declare up to 8-dimensional arrays!



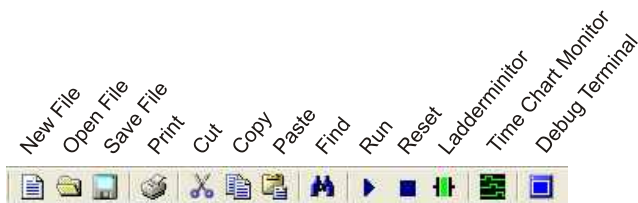


IN-DEPTH ANALYSIS: How to use CUBLOC Studio

Let us learn how to use CUBLOC Studio more in detail.



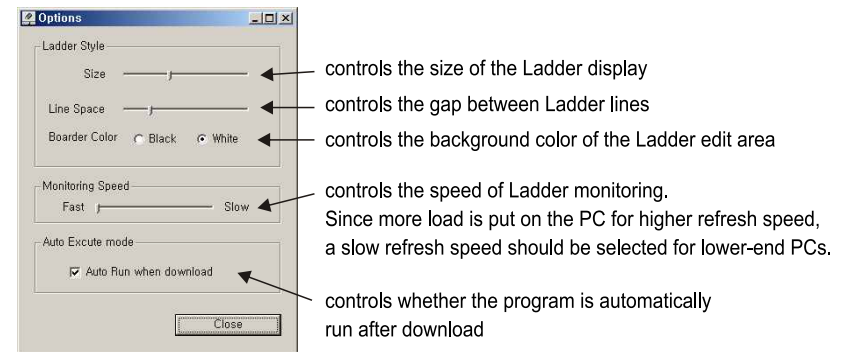
Frequently used features are gathered at the Tool bar as icons. One can use each feature by clicking the pertinent icon.



Shortcut keys that can be used during BASIC source editing are as follows:

Shortcut Key	Operation
CTRL-Z	Undo
CTRL-O	Open
CTRL-S	Save (It is best to save for backup frequently during editing.)
CTRL-C	Copy
CTRL-X	Cut
CTRL-V	Paste
CTRL-F	Find
CTRL-HOME	Move to the beginning of the file
CTRL-END	Move to the beginning of the file

In the Setup menu, if you select the Studio Option menu item, a popup window such as below will appear. This popup window can be used as follows:



CUBLOC Studio's Ladder source input method is dealt with in detail in Chapter 3: Ladder Logic Programming. For specific use of the menu, please refer to the CUBLOC User's Manual. This book consists of contents which one can follow if he/she knows just how to use the icons on the tool bar.

2 BASIC Programming

OPERATIONAL SUMMARY

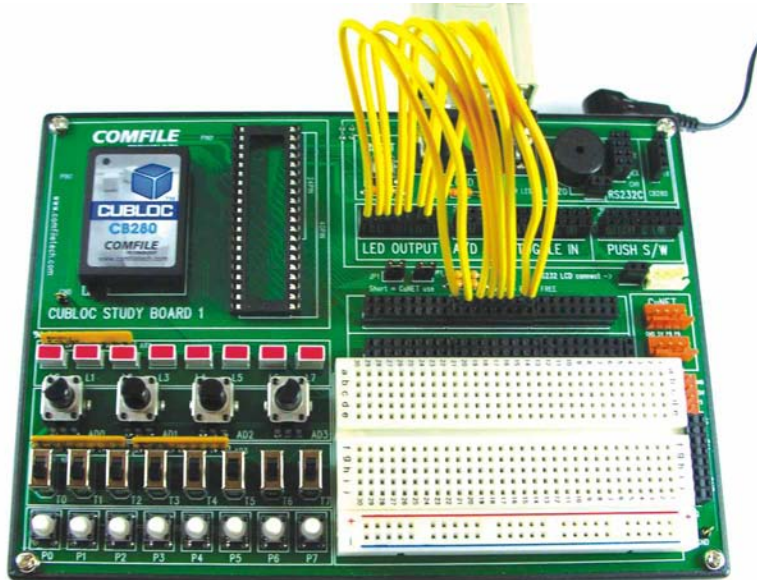
We will compose a program that sequentially flashes the 8 LEDs.

We will also learn how to handle 8 bits together, arithmetic operations in BASIC language and how to debug a program.

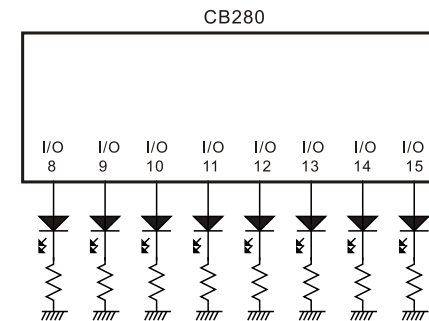
LAB.3 Sequential Flashing of LEDs

Circuit Configuration

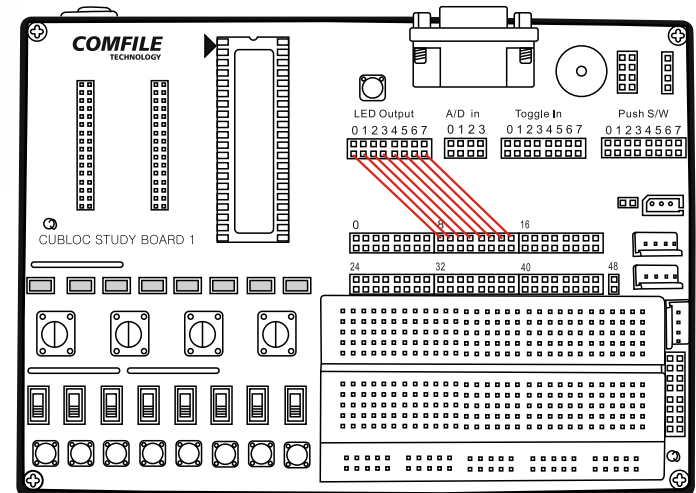
Connect all 8 I/O ports (from port 8 to port 15) to the 8 LEDs.



Circuit Diagram



When you do the wiring, first make sure the board is power OFF. After the wiring is finished, then power ON the CUBLOC again.





Step 1

Shall we warm up before we enter the main subject?

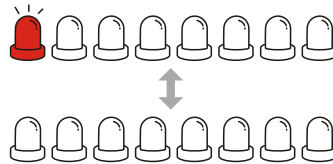
Let's rerun the program we created in Lab 1 by just changing the port number.

One LED will flash.

```

Const Device = CB280
Do
  High 8
  Delay 200
  Low 8
  Delay 200
Loop

```



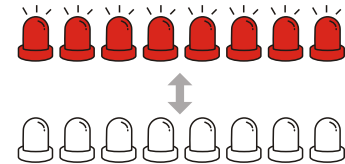
Step 3

If we use the BYTEOUT command, we can express the same operation much more compactly.

```

Const Device = CB280
Do
  Byteout 1,255
  Delay 200
  Byteout 1,0
  Delay 200
Loop

```



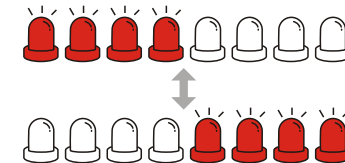
Step 4

Shall we flash 4 LEDs at a time in turn? By changing the numbers (i.e., arguments) of the BYTEOUT command, it can be implemented easily.

```

Const Device = CB280
Do
  Byteout 1,&HF0
  Delay 200
  Byteout 1,&H0F
  Delay 200
Loop

```



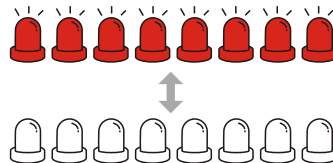
Step 2

Now, let's make all 8 LEDs flash.

```

Const Device = CB280
Do
  High 8
  High 9
  High 10
  High 11
  High 12
  High 13
  High 14
  High 15
  Delay 200
  Low 8
  Low 9
  Low 10
  Low 11
  Low 12
  Low 13
  Low 14
  Low 15
  Delay 200
Loop

```

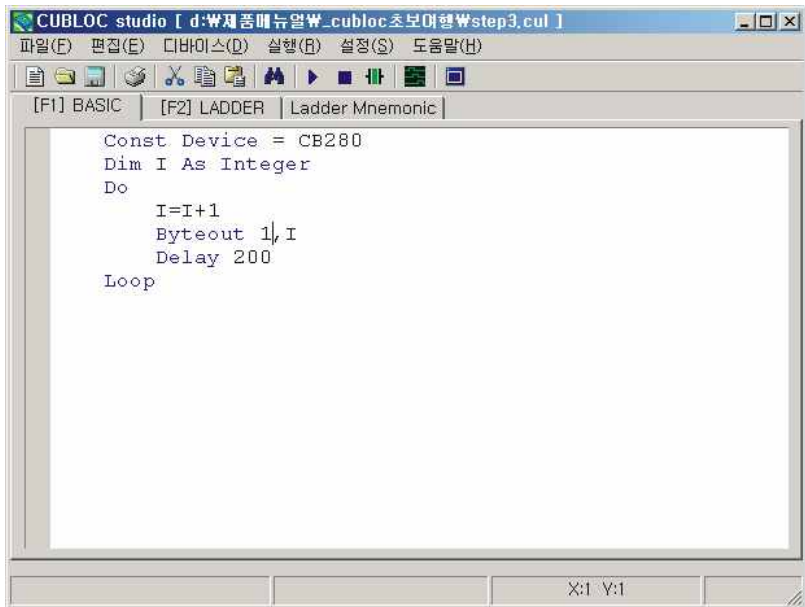


Experiment by changing the numbers coming after the BYTEOUT command. You will see how the flashing pattern of the LEDs change!

Source Program

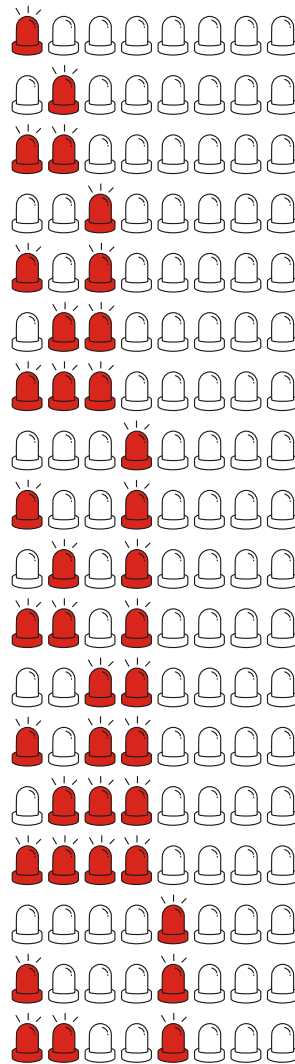
Let's get back to the main track and compose a program that sequentially flashes the 8 LEDs.

Now look at the screen below and type in the source program accordingly.



Confirm the Run Result

Click the RUN button and confirm the result. If the LEDs are flashed sequentially, that means the program was composed correctly.



It seems that the LEDs are flashed in an orderly fashion following some rule, right? Why would that be? Think about the reason.





Source Program

```

Const Device = CB280
Dim I As Byte ← Declare variable I as a Byte type.
Do
    I=I+1 ← Increment I value by 1.
    Byteout 1,I ← Output variable I value to Block 1 ports.
    Delay 200
Loop ← Infinitely repeat the contents between Do and Loop.

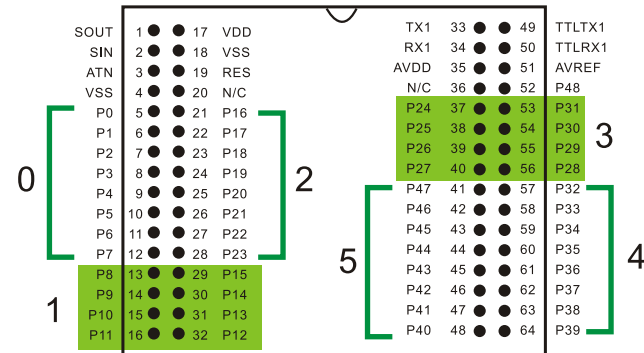
```

Most of the program is what we have learned already and only Byteout would be a new command.

Byteout outputs a value to 8 ports at the same time. In CUBLOC, 8 ports are bundled together to be called a Port Block. See the figure in the right column.

In CB280, there are 5 port blocks. Ports 8 through 15 belong to Port Block 1. Therefore, "BYTEOUT 1, I" outputs I value to Port Block 1. Since variable I is declared as a Byte type, it has an 8-bit information. The 8-bit information will be seen at the 8 ports of Port Block 1.

The arithmetic expression "I = I + 1" adds 1 to I and stores the value in I. As a result, the value stored in variable I is increased by 1.



NEW COMMANDS

Byteout

BYTEOUT <Port Block No.> <Value>

This command outputs <Value> to the ports of <Port Block No.>.

All the ports of the < Port Block No.> turn into output ports.

In place of the <Value>, an Integer type variable or a constant should be used.

Integer type variable: Byte, Integer and Long types.

Integer type constant: a directly written number such as "1234".

Real constant: a number with the decimal point.

Integer constant: a number without the decimal point.



HELPFUL TIPS

Many ways to express Integer type Constants

Besides expressing an Integer type Constant in usual decimal (10) system, it can be expressed as a binary (2) number or a hexadecimal (16) number.

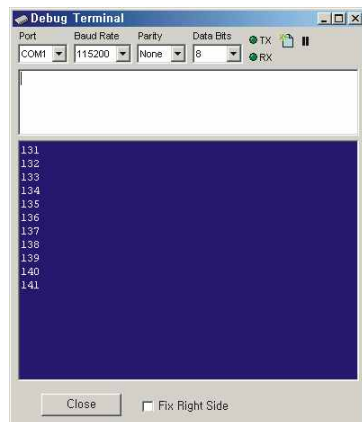
Decimal: 1234, 7573

Hexadecimal: &H12AB, &HABCD, 0xABCD, 0x123

Binary: &B10101111, 0b1010101

DEBUG

```
Const Device = CB280
Dim I As Byte
Delay 500
Do
    I=I+1
    Byteout 1,I
    Debug Dec I,CR
    Delay 200
Loop
```



Run a program after you inserted a Debug command!
A popup window labeled "Debug Terminal" will appear and some values will be shown inside it. This is how you do debugging with CUBLOC.

Debugging is to catch a "bug" in a program and correcting it. Since bugs are not easily seen, one has to examine value(s) of variable(s) of a running program. For this purpose, the command DEBUG is used.

When a Debug statement is used, the CUBLOC Studio pops up a Debug Terminal window and displays the debugged variable values inside the window. Looking at the terminal, the user makes a judgment as to whether the program he/she composed is working properly as the user intended. The "Dec" after the Debug command means the number following is a decimal number. And the CR at the end of the Debug statement stands for "Carriage Return," meaning after outputting a value, the next value should be outputted on a new line.

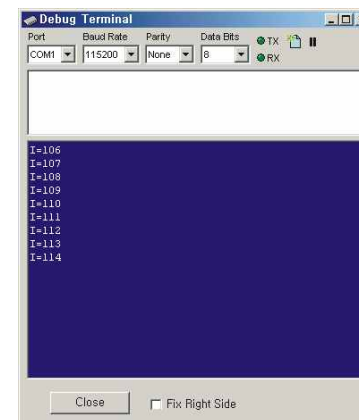


HELPFUL TIPS

Using DELAY before DEBUG

Before the first DEBUG command is used, a DELAY 500 statement must be used. Since CUBLOC's execution speed is faster than a normal PC, a Debug statement will be executed even before the PC displays a Debug Terminal popup window. If that happens, the Debug Terminal window will have nothing displayed. Therefore, we need to delay the CUBLOC a little bit so that the PC can catch up with the CUBLOC.

```
Const Device = CB280
Dim I As Byte
Delay 500
Do
    I=I+1
    Byteout 1,I
    Debug Dec? I,CR
    Delay 200
Loop
```



Run the same program after you put a ? mark after the DEC as shown above. In the Debug Terminal, the displayed lines will be such as "I = 123", which can be better understood.



NEW COMMANDS

Debug

Debug <data>

This command outputs a variable value to the Debug Terminal window. It has many usages as shown below.

Examples;

Debug Dec A	'Display variable A's value as a decimal number.
Debug Hex A	'Display variable A's value as a hexadecimal number.
Debug "Result :",Dec A	'Display variable A's value following the character string "Result: ".
Debug Dec5 A	'Display variable A's value as a 5-place decimal number.
	'If A is a 123, fill the preceding two places with blanks, i.e., 123.
Debug Hex8 A	'Display variable A's value as a 8-place hexadecimal number.
	'Fill the preceding zeroes with blanks.
Debug Dec ? A	'Display variable A's value as a decimal number preceded by "A =".
Debug Float A	'Display variable A's value as a real number.
Debug Clr	'Clear the Debug Terminal window.
Debug Goxy,4,3	'Set the cursor position on the Debug Terminal window.



Arithmetic Expressions

```

Const Device = CB280
Dim I As Integer, B as Integer
Delay 500
A = 5
B = 300
A = A * B
Debug Dec A, CR

```

Multiply B to A and store the result in A



If this program is run, as a result 1500 will be displayed on the Debug Terminal. The "Delay 500" statement is there to prevent the faster CUBLOC running the Debug statement before the slower PC brings up the popup Debug Terminal window.

In BASIC language many kinds of arithmetic operations are possible. Besides the common addition, subtraction, multiplication and division, operations such as Mod, << and >> are also possible.

The following are the arithmetic operators that can be used with CUBLOC.

^ * / MOD + - << >>

From the left to right in the above sequence, more an operator is on the left hand side, its operation precedence is higher. That is, when many operator types are mixed, the operations are carried out in accordance with their precedences: from higher to lower.

^ is the exponential operator, Mod is the remainder operator, << and >> are bit-shift operators, left and right, respectively.

$$A = 3 + 5 * 4$$

In this case, the result is 23 because $5 * 4$ is carried out first and then 3 is added to the resulting 20.

Mod operator returns a remainder, not the result of division.

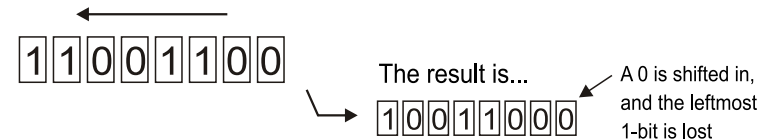
$$A = 12 \text{ MOD } 5$$

In the above operation, 2 is stored in A.

<< and >> are bit-shift operators, which are not commonly used.

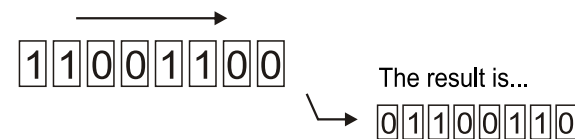
But in CUBLOC BASIC they are very useful operators.

For example, "A << 1" means "to left bit-shift the variable A value in binary by 1 bit place." Here the newly shifted in bit value is 0 and the shifted out value disappears.



On the contrary,

"A >> 1" means "to right bit-shift the variable A value in binary by 1 bit place."

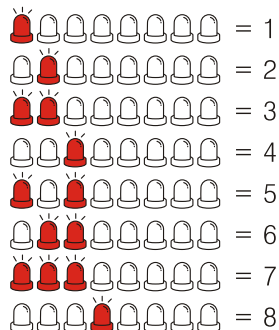


"A >> 3" means "to right bit-shift the variable A value in binary by 3 bit places."

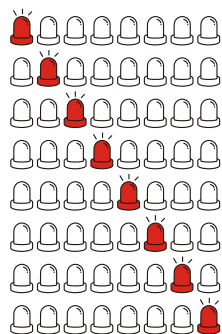
Applied Programming

Did you figure out the principle behind the flashing sequence of the LEDs on p. 38?

The changing flashing pattern denoted a binary number increased by 1.



This time we will compose a program that flashes an LED from left to right in turn.



```
Const Device = CB280
```

```
Do
Byteout 1,1
Delay 200
Byteout 1,2
Delay 200
Byteout 1,4
Delay 200
Byteout 1,8
Delay 200
Byteout 1,&H10
Delay 200
Byteout 1,&H20
Delay 200
Byteout 1,&H40
Delay 200
Byteout 1,&H80
Delay 200
Loop
```

This outputs 1 on Port Block 1.
If the connection is made in the order of 0, 1, 2 ..., only one LED on the left end will be turned on.

The same operation can be programmed alternatively like this:

```
Const Device = CB280
Dim I As Byte
Dim J As Byte
Do
J = 1
For I =0 To 7
Byteout 1,J
J = J << 1
Delay 200
Next
Loop
```

The contents between FOR and NEXT are repeated 8 times while I value changes from 0 to 7

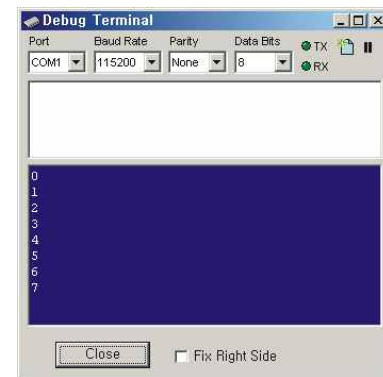
Though it performs the same operation, the program looks quite different from the former one.

This program especially used the FOR-NEXT loop!

FOR-NEXT loop repeats the loop only for the pre-specified times.

The variable I that comes after the keyword FOR is called parameter, which is used for counting how many time the loop will be executed. Also, the I value can be used inside the loop.

```
Const Device = CB280
Dim I As Byte
Delay 200
For I =0 To 7
Debug Dec I,cr
Next
```



The above is a program composed to help one understand how the FOR-NEXT loop works. As one sees the numbers output on the Debug Terminal, one can easily understand how the FOR-NEXT loop works.



NEW COMMANDS

For, Next

**FOR <Variable> = <Initial Value> to <Final Value> [<Step>]
<statement>**

...
**[EXIT FOR]
NEXT**

This is a command structure that repeats a contents for the pre-specified times. One gives the parameter variable the initial value and provides the contents to be repeated. When the execution point reaches NEXT, the parameter variable is incremented. If its value reaches the <Final Value>, looping stops.

When there is no <Step> specified, the parameter is incremented by 1 by default; if <Step> is specified, the parameter value is incremented by the <Step> value.

If there is an EXIT FOR in the middle of the loop, the looping can be exited at that point.

```
Dim A as Long
For A =10 TO 0 Step -1
    Debug Dec A,cr
Next
```

If one wants to specify a negative <Step> value, the parameter <Variable> must be declared as a LONG integer.

```
Dim K as Byte
For K = 0 to 300
    Debug Dec K,cr
Next
```

This has to be changed to either
INTEGER or LONG.

Since the K value has to be incremented up to 300, if K is declared as BYTE, this program will not work properly and will repeated infinitely. In this kind of a case, K must be declared either as INTEGER or LONG.



Fun Trip

to the **CUBLOC World**

TRY THIS!

- 1 Make the LEDs flash one at a time from left to right and then right to left, indefinitely.
- 2 Do the same as above but this time 2 LEDs at a time.
- 3 Make the left and right flashing above faster.

- No model answers are separately provided -

A job well done

AS you watch the LED outputs, you would have understood how binary numbers are represented with bits.

You would have also learned the operators of BASIC and how to use the Debug command. Since these will be used frequently in the future, please make sure you understand them completely.



COMMANDS LEARNED IN THIS CHAPTER

Debug

Byteout

For-Next

Operators: +, -, *, /, <<, >>, Mod

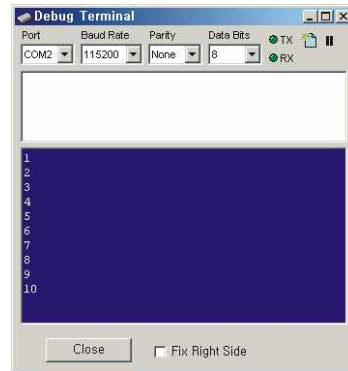
43



IN-DEPTH ANALYSIS: FOR-NEXT command

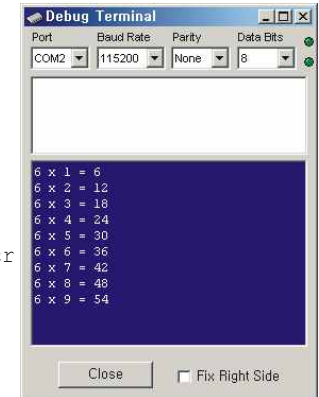
Let us study more on FOR-NEXT.
Create a program that outputs 1 to 10
on the Debug Terminal. Of course,
this kind will be too easy for you, right?

```
Const Device = CB280
Dim A as Integer
Delay 500
For A=1 to 10
    Debug Dec A,cr
Next
```



Now, let us display the 6's column of the
Multiplication Table.

```
Const Device = CB280
Dim A As Integer
Delay 500
For A=1 To 9
    Debug "6 x ",Dec A," = ", Dec 6 * A,cr
Next
```



If a comma (,) is used in the Debug
statement, one can display multiple
characters or numbers. A character string
should be surrounded by double quotes.

This time let us do the same except
that the parameter value will be
incremented by 2 each time.

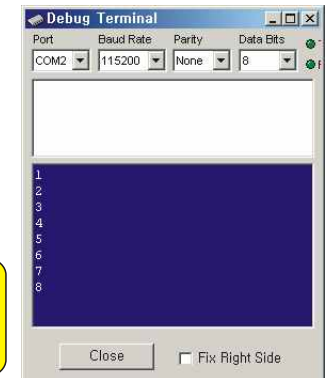
```
Const Device = CB280
Dim A as Integer
Delay 500
For A=1 to 10 Step 2
    Debug Dec A,cr
Next
```



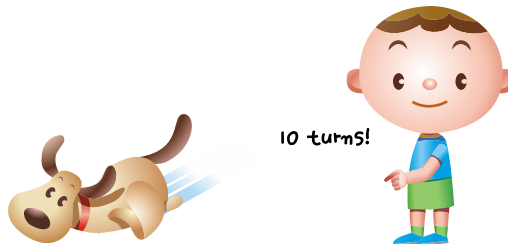
Now, let us display 1 to 10 but make the
program exit the loop when the parameter
value reaches 8.

```
Const Device = CB280
Dim A As Integer
Delay 500
For A=1 To 10
    Debug Dec A,cr
    If a=8 Then Exit For
Next
```

When the A
value reaches 8,
the program will
exit the loop



I believe now you have a better grasp of FOR-NEXT's diverse usages.
As seen, diverse combination of FOR-NEXT's parameter, Step value and
Exit condition can produce many kinds of results.





IN-DEPTH ANALYSIS: Precedence in Operator Arithmetic

With CUBLOC, besides arithmetic operations such as addition and subtraction, there are logical operation and comparison operation. These kinds can be distinguished by the operator used.

- Arithmetic operators: Commonly used mathematical operations such as addition and subtraction: +, -, etc.
- Comparison operator: Operators used with commands such as If, While: <, >, =, etc.
- Logical operators: Operators used for logical operation between two binary numbers: AND, OR, etc.

ARITHMETIC OPERATORS

Kinds	Mathematical Expression	Expression in BASIC	Usage Example
Addition	+	+	3+4+5 6+A
Subtraction	-	-	10-3 63-B
Multiplication	x	*	2 * 4 16 * C
Division	÷	/	1120 / 23 B / E
Exponentiation	5 ³	^	5^3 A^2
Remainder	None	MOD	120 MOD 3

Expression of Fractions

Since a fraction cannot be expressed in its conventional notation in BASIC language, in BASIC it is denoted using the division operator (/). When the numerator or the denominator is an arithmetic expression, it should be surrounded by parentheses.

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

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

In an arithmetic operation, many operators can be used. Since each operator has its own precedence relative to other operators, no matter how complicated an operation may be, its computation is carried out from the operation of the highest precedence at the time

Step 1: Expression within parentheses

Step 2: A sign preceding a number

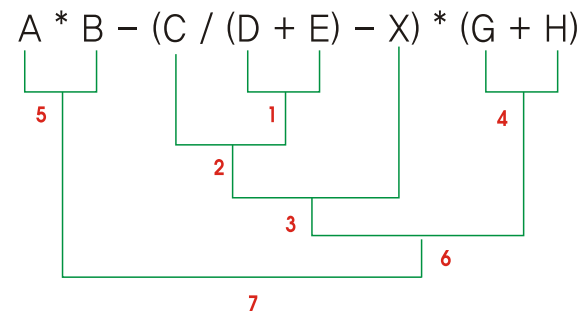
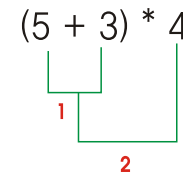
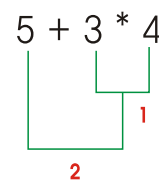
Step 3: Exponentiation (^)

Step 4: Multiplication, Division and Mod (remainder) (*, /, MOD)

Step 5: Addition and Subtraction (+, -)

Step 6: Shift-left and Shift-right (<<, >>)

When there are multiple operators with same precedence, the operation flows from left to right.



OPERATIONAL SUMMARY

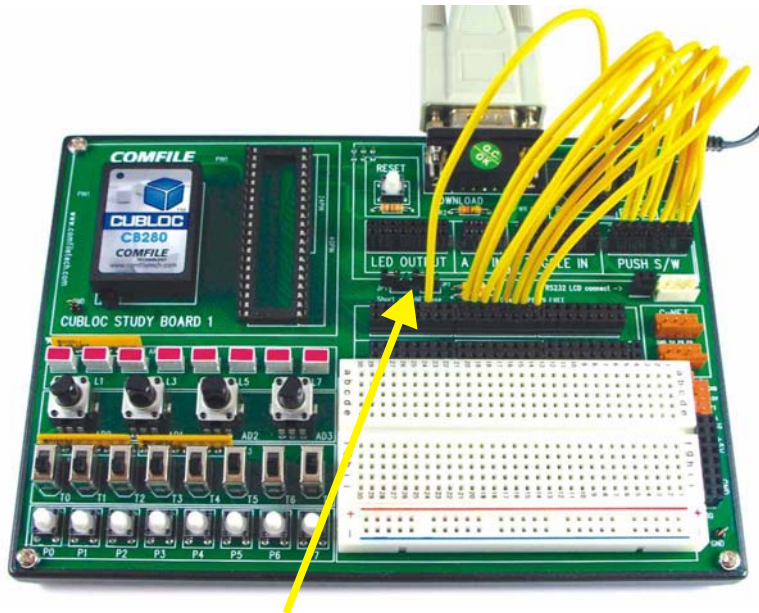
Let's make a digital piano with CUBLOC. Let me calm down those who begin to aspire, 'Wow! I'm gonna get a digital piano!!' You will be able to play a single scale of Do, Re, Mi, Fa, Sol, La, Ti, Do, and that using a piezoelectric speaker ("piezo" in short from now on).

We will be studying the tone generation principle, how to use PWM port, and how to use SELECT and CASE statements.

LAB.4 Digital Piano

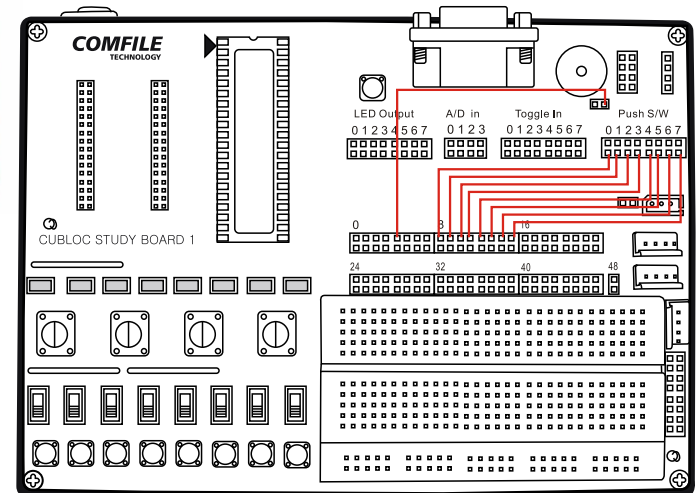
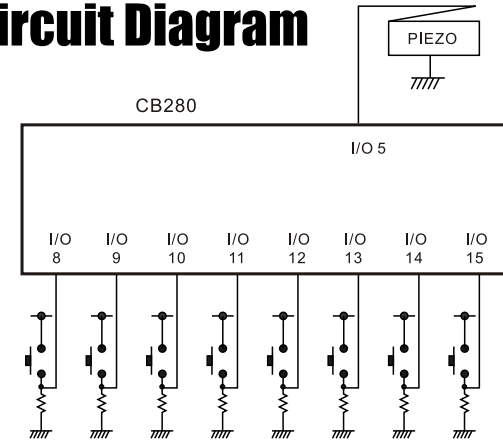
Circuit Configuration

Connect all of I/O ports 8 to 15 to the Push Switch, and connect I/O port 5 to the piezo



Make sure you leave both of CUNET use jumpers JP1 and JP2 open!

Circuit Diagram





Source Program-1

Enter the source program as show below:

```

Const Device = CB280
Const SoundCh = 0
Dim A As Byte
Low 5
Do
  A = Bytein(1)
  Select Case A
  Case 1
    Freqout SoundCh, 4403 '도
  Case 2
    Freqout SoundCh, 3923 '레
  Case 4
    Freqout SoundCh, 3495 '미
  Case 8
    Freqout SoundCh, 3299 '파
  Case &h10
    Freqout SoundCh, 2939 '솔
  Case &h20
    Freqout SoundCh, 2618 '라
  Case &h40
    Freqout SoundCh, 2333 '시
  Case &h80
    Freqout SoundCh, 2202 '도
  Case Else
    Pwmoff 0
  End Select
Loop

```

Confirmation Run

After you click to RUN button to download the source program, push any of the push buttons! If you can hear a sound through the piezo, that means the program works.

If you can't hear any sound, check the wire connects one more time! Make sure you left both of CUNET use jumpers JP1 and JP2 open!

Explanation of the Source Program-1

In the beginning part of the source program, you can see a statement for declaring a constant "Const SoundCh = 0". Constant is a "number who value is fixed" in the source program. A constant can be directly written in a statement; but to enhance the understanding of those who read the program later (e.g., to describe the meaning of the constant in this case), usually even constants are declared separately.



NEW COMMANDS

Const

Const <Constant Name> = <Constant Value>

This is a command that declares a constant. If one writes.

```
Const Relay = 5
```

from then on in that source program the word Relay means constant 5.

The same command can be expressed as

```
Relay Con 5
```

A device declaration such as "Const Device = CB280" is a statement that assigns a device name (character string constant) "CB280" to a system constant DEVICE. That is, CUBLOC Studio recognizes which model it is using by checking the DEVICE constant.



HELPFUL TIPS

How are sounds generated?

Sound is a resonance that is being conveyed in the form of a wave in the air. Our voices and speaker sounds are also resonance in the air being delivered to our ears. With the same principle, we can generate sound using a piezo.

Piezo consists of two thin metal films put together. When electric current is run to these two films, the two films begin to stick together and separate, creating waves in the air. This is how speakers work, too.



speakers



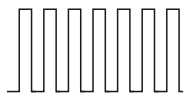
piezo

Speakers are used with audio components and TVs, as they make bigger sounds than piezos. Since piezos make small sounds, they are used for generating simple key-touch sounds.

To repeatedly supply and stop current to a speaker or a piezo, a pulse signal has to be generated in CUBLOC.



Type A



Type B

There are two types of pulse signals. With Type A pulse, one can generate diverse sounds such as human voice or musical sounds.

This is too much to be implemented in CUBLOC. Therefore, CUBLOC generates simple square pulses such as Type B.

STEP 1

Currently the piezo is connected to Port 5. A pulse can be generated if high and low voltages are outputted at Port 5. Let's do it!

```
Const Device = CB280
Low 5
Do
  Reverse 5
  Delay 1
Loop
```



NEW COMMANDS

Reverse

Reverse <Port No.>

This command reverses the state of a specified port.

If it was already HIGH, it is made LOW; and vice versa.

Since this command does not make the port into an output port automatically, before this command is used a HIGH or a LOW command should be used to make the port into an output port.

```
Low 5
Reverse 5 'Port 5 takes on HIGH state.
```


STEP 2

With the above experiment, one may simply hear a continuing, single beep. CUBLOC has a more sophisticated way to generate a pulse signal. That is to use the PWM.

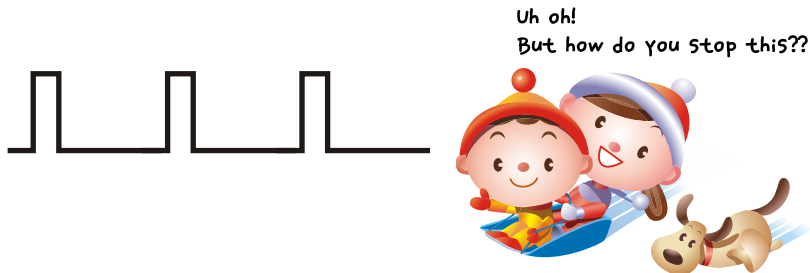
```
Const Device = CB280
Low 5
Pwm 0,200,5000
```

You probably are not familiar with the term PWM, which stands for Pulse Width Modulation. One can simply think it as a pulse generation device.

Shall we see the difference between the previous two programs? Step 1 program has the disadvantage that it has to keep looping during pulse generation. That is, CUBLOC cannot do anything else during pulse generation.

On the other hand, if one uses the PWM, just issuing the PWM command makes the specified port generate the wanted waveform, CUBLOC can do other work. Purely automatic!

In addition, the PWM can also adjust the width of the pulses. The PWM statement "PWM 0, 10, 1000" makes just 10 out of 1000 waveform points HIGH (while the other 990 points remain LOW).



On the other hand, "PWM 0, 900, 1000" makes just 900 out of 1000 waveform points HIGH (while the remaining 10 points remain LOW).



As can be seen, though the above two output pulses have the same frequencies, their high pulse widths can be different, resulting in two distinguished output waves.

PWM can be used in many areas. It can control the brightness of LEDs and lamps, and control the speed of motors. In the next example, it will be used to generate sound.

PWM



NEW COMMANDS

PWM <Channel No.> <Duty> <Pulse Period>

This command begins outputting a pulse-type waveform.

For the <Channel No.> one has to specify the PWM channel number. Note that it is not a port number. Since the PWM channel number is different for each CUBLOC model, refer to the pinout diagram of each model.

Before you use the PWM command, you must turn the port connected to the PWM channel into an output state. <Pulse Period> is any number with the maximum value being 65535. The larger this value, the lower the frequency of the resulting wave. The <Duty> value cannot be bigger than the <Pulse Period> because it is the width of the HIGH state portion of the pulse wave.

PWM 0, 200, 1024 'Within the pulse period of 1024, make 200 HIGH

PWM channels 0, 1 and 2 should use the same pulse period because they share the same timer internally. The same goes with PWM channels 3, 4 and 5. But the periods of PWM channel group 0, 1 and 2, and group 3, 4 and 5 may differ.

Once you initiated a PWM wave generation using the PWM command, as long as the CUBLOC is on the pulses will keep generating. You can stop the PWM pulse generation using the command PWMOFF.

But when you see the "Digital Piano" source program on p. 47, there is no PWM command but only FreqOut commands.

FreqOut can be seen as a command similar to PWM, a simplified version. It's just that the "duty ratio" is 50% such that the HIGH and LOW edges have the same 1:1 width ratio.



As can be seen in the figure above, a waveform where the HIGH and the LOW width are the same are said to have the 50% duty ratio. Since waves with 50% duty ratio are used for sound generation, we have used FreqOut instead of PWM for convenience' sake.

Then how do we generate different sounds such as "Do Re Mi Fa Sol La T Do"? That is very simple. The A tone of the 4th Octave (on the piano) has the frequency of 440 Hz (Hertz). And similarly each sound has its own frequency. When these frequencies are inserted into the formula described in the FreqOut command explanation, we can build a table that gives the specification of each sound.



HELPFUL TIPS

Multi-function Port

Each I/O port of CUBLOC has the other port besides the usual I/O port such that the other port can accept PWM or ADC input. Such port is called a Multifunction Port. At the same time, this means Port 5 can no more be used as an I/O port once it is used as a PWM port. That is, for a multifunction port, one has to choose between the I/O function, and the extra function. There is no explicit command to make the choice but one implicitly does so by making the I/O port into an input or output state depending on the functional situation.



NEW COMMANDS

Pwmoff

PWMOFF <Channel No.>

This command stops outputting a PWM wave at <Channel No.>.

This command is used to stop a wave generation started by the FREQOUT command.

```
Low 5
Pwm 0, 50, 100 ' Start generating a PWM wave
Delay 2000
Pwmoff 0 ' Stop the wave 2 seconds after it started generating
```

Freqout

FreqOut <Channel No.> <Freq>

This command generates a different wave depending on the given<Freq>. You do not directly write the actual frequency at <Freq> but insert a value there, which is computed by the following formula.

$$\text{Freq} = \frac{230400}{\text{Freq}}$$

This formula computes an accurate value only when Freq value is equal to or greater than 100.

```
Freqout 0, 5236 ' This generates a frequency of 440 Hz
```

Do	Re	Mi	Fa	Sol	La	Ti	Do	Re	Mi
4403	3923	3495	3299	2939	2618	2333	2202	1961	1851



Source Program-2

```

Const Device = CB280
Const SoundCh = 0
Dim A As Byte
Low 5
Do
    A = ByteIn(1)
    Select Case A
        Case 1
            Freqout SoundCh, 4403 'Do
        Case 2
            Freqout SoundCh, 3923 'Re
        Case 4
            Freqout SoundCh, 3495 'Mi
        Case 8
            Freqout SoundCh, 3299 'Fa
        Case &h10
            Freqout SoundCh, 2939 'Sol
        Case &h20
            Freqout SoundCh, 2618 'La
        Case &h40
            Freqout SoundCh, 2333 'Ti
        Case &h80
            Freqout SoundCh, 2202 'Do
        Case Else
            Pwmoff 0
    End Select
Loop

```

PWM Channel 0 is made into an output state

If no key is pressed, the PWM pulse generation is stopped

Wow! This is a rather long program. The "Freqout SoundCh, 4403" statement generates the Do sound. Before this command is executed, there is a "Select Case A" conditional statement. "Select Case ..." is useful when one would like to execute different options depending on a variable value. The same can be implemented using IF ... ELSEIF, but using Select Case is much more crisp and compact.

ByteIn command does the opposite of the ByteOut command. It reads the values of an entire port block, i.e., 8 bits. Since all ports of Port Block 1 are connected to the PUSH switch in this example, using a single command A = ByteIn(1) reads all states of the PUSH switch.

Since only that switch is turned on (i.e., bit status = 1) when a switch is pressed, a Case statement can be used to simulate such situation.



0	0	0	0	0	0	0	1	= 1
0	0	0	0	0	0	1	0	= 2
0	0	0	0	0	1	0	0	= 4
0	0	0	0	1	0	0	0	= 8
0	0	0	1	0	0	0	0	= &H10
0	0	1	0	0	0	0	0	= &H20
0	1	0	0	0	0	0	0	= &H40
1	0	0	0	0	0	0	0	= &H80

Select, Case

Select Case <variable>

Case <value 1>

<statement 1>

Case <value 2>

<statement 2>

Case <value 3>

<statement 3>

Case Else

<statement else>

End Select

When the <variable> value is equal to <value 1>, <statement 1> is executed; if equal to <value 2>, <statement 2> is executed, and so on. When there is no match, <statement else> after <Case Else> is executed.

Also Is and an inequality (< or >) can be used to make a value comparison

```

Select Case K
Case Is      'If K is less than 10.
    R =
Case Is 4    'If K is less than 40.
    R = 1
Case Is 5    'If K is less than 50.
    R = 2
Case Is 100  'If K is less than 100.
    R = 3
Case else    'Otherwise...
    R = 4
End Select

```

Applied Programming-1

Shall we alter the program a bit to make it more fun?

Now we shall change the program so that pressing a key will create a "Ding-Dong" chime.

```
Const Device = CB280
Const SoundCh = 0
Dim A As Byte
Low 5
Do
  A = Bytein(1)
  Select Case A
    Case 1
      Freqout SoundCh,4403 'Do
      Delay 200
      Freqout SoundCh,3495 'Mi
      Delay 200
      Freqout SoundCh,2202 'Do
      Delay 200
    Case 2
      Freqout SoundCh,3923 'Re
    Case 4
      Freqout SoundCh,3495 'Mi
    Case 8
      Freqout SoundCh,3299 'Fa
    Case &h10
      Freqout SoundCh,2939 'Sol
    Case &h20
      Freqout SoundCh,2618 'La
    Case &h40
      Freqout SoundCh,2333 'Ti
    Case &h80
      Freqout SoundCh,2202 'Do
    Case Else
      Pwmoff 0
  End Select
Loop
```

When this portion is added, whenever Switch 1 is pressed, a "Ding-Dong-Ding" sound will be made.

Modify the program in your own way to produce many other kinds of sounds!



NEW COMMANDS

Bytein

<Variable> = Bytein(<Port Block No.>)

This command changes a port block into an input state, and reads the 8 bits of values from the 8 input ports into a Byte variable.

Here the variable must be of an Integer type.

Note that the all the 8 ports of the port block change into the input state.

If one of the ports to be affected was being used as an output port, that port will no longer be able to function as an output port after this command is executed.

Applied Programming-2

Let us think about how we can input the frequency values of Do, Re and Mi more conveniently. In the previous example, we replaced the variables such as Do and Re with numbers such as 4403 and 3923, respectively. But if we use the Const command, things can be done more conveniently.

If one assigns frequency values to mnemonic words such as Do and Re using the Const command, from then on one can compose the source code using those meaningful words instead of manually inputting the respective frequencies.

```
Const Device = CB280
Const SoundCh = 0
Const 도 = 4403
Const 레 = 3923
Const 미 = 3495
Const 파 = 3299
Const 솔 = 2939
Const 라 = 2618
Const 시 = 2333
Const 하도 = 2202
```

```
Dim A As Byte
Low 5
Do
  A = Bytein(1)
```

```
Select Case A
  Case 1
    Freqout SoundCh,도
  Case 2
    Freqout SoundCh,레
  Case 4
    Freqout SoundCh,미
  Case 8
    Freqout SoundCh,파
  Case &h10
    Freqout SoundCh,솔
  Case &h20
    Freqout SoundCh,라
  Case &h40
    Freqout SoundCh,시
  Case &h80
    Freqout SoundCh,하도
  Case Else
    Pwmoff 0
  End Select
Loop
```

We used Do, Re, Mi ... instead of numbers.



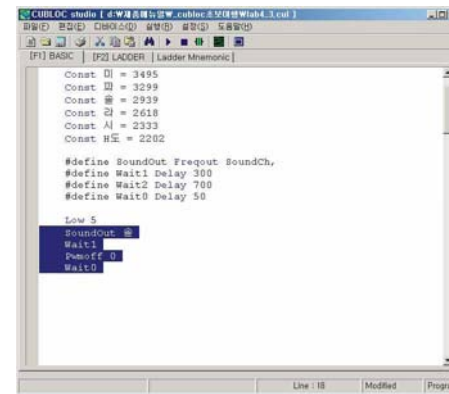
Applied Programming-3

Next is a program that performs a Korean children's song "School Bell Rings". The melody is played in a manner where, after playing one sound, there is a delay, and then the next sound is played. The program may be somewhat long; but type it in. You can listen to the music using CUBLOC.



When you are inputting the source, it can be tedious to input the same or similar portions more than once, right?

Copy and Paste come handy in such a situation.

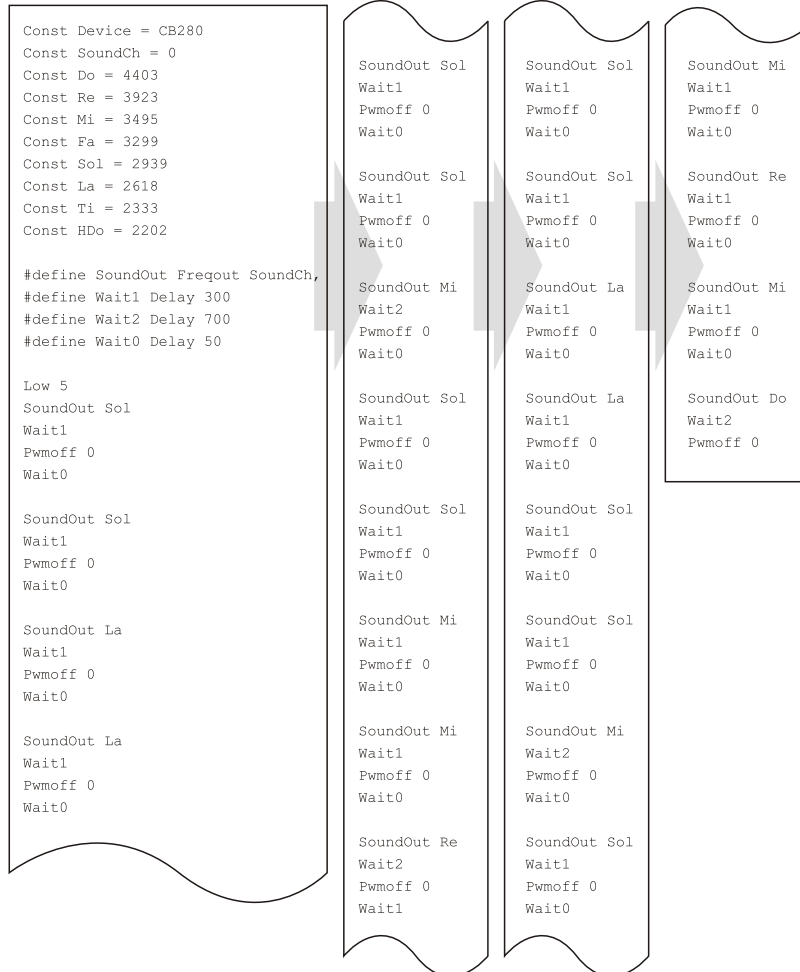
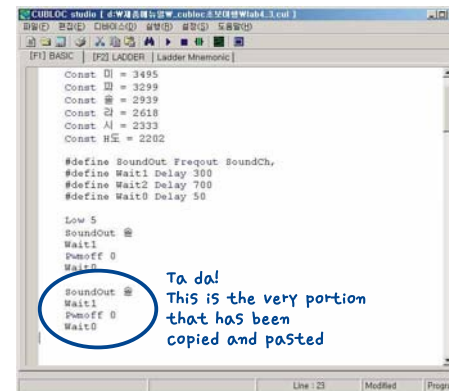


Place the cursor at the beginning of the repeated part, press the SHIFT key (and keep it pressed down), and move the cursor to the end of the repeated part. Then the repeated part will be highlighted as it is selected.

If you press CTRL + C (i.e., while pressing the CTRL key down, press the c key), the selected portion is copied into the PC's system memory, usually called buffer.

Now, move the cursor to the position where you would like to paste the copied part.

Then press CTRL + V and the copied part will be copied into the new location. Now you can simply modify the slightly differing part(s)!



If you see the above program, a new command appears: `#define`. This command facilitates inputting of frequently repeated commands.

Using this feature, one can just say "SoundOut Sol" without having to type in a FreqOut lengthy command each time.

Since the new compact expression conveys the meaning of performing the Sol sound more clearly, it makes both programming and later understanding it easier.

At a glance, it may look similar to the Const command; but Const deals only with constants no characters and blanks.

The source program may look even after using `#define`. If we abbreviate the repeating parts as subroutines, the source can be expressed even more compactly.

Now, look at the source code on the right hand side. It is far shorter than the previous version.

The "Sub SoundOut ..." portion is where a subroutine is declared.

When such a subroutine is defined, whenever the program execution hits the name "SoundOut," the execution point jumps to the subroutine, to execute the subroutine, and then returns to previous execution point in the "main" program.

And since one can also pass along a parameter (or parameters) when a subroutine is called, the subroutine can be run with different parameters.



NEW COMMANDS

#define

#define <statement1> <statement2>

This command simply replaces <statement1> with <statement2>.

Since <statement2> can even include blanks, this command is useful in making abbreviations for long command sequences. E.g.,

```
#define Playsound Freqout 0, 4938
           statement1      statement2
```

```
Const Device = CB280
Const SoundCh = 0
Const Do = 4403
Const Re = 3923
Const Mi = 3495
Const Fa = 3299
Const Sol = 2939
Const La = 2618
Const Ti = 2333
Const HDo = 2202
```

```
Const W1 = 300
Const W2 = 700
```

```
Low 5
SoundOut Sol,W1
SoundOut Sol,W1
SoundOut La,W1
SoundOut La,W1
SoundOut Sol,W1
SoundOut Sol,W1
SoundOut Mi,W2
SoundOut Sol,W1
```

```
SoundOut Sol,W1
SoundOut Mi,W1
SoundOut Mi,W1
SoundOut Re,W2
Delay w2
SoundOut Sol,W1
SoundOut Sol,W1
SoundOut La,W1
SoundOut La,W1
SoundOut Sol,W1
SoundOut Sol,W1
SoundOut Mi,W2
SoundOut Sol,W1
SoundOut Mi,W1
SoundOut Re,W1
SoundOut Mi,W1
SoundOut Do,W2
```

```
End
```

```
Sub SoundOut(Fr As Integer, Wt As Integer)
    Freqout soundch,Fr
    Delay wt
    Pwmoff 0
    Delay 50
End Sub
```




NEW COMMANDS

Sub, Endsub

SUB <Subroutine Name> [(<Parameter1> As <Type1>) [<Parameter2> As <Type2>] ...]
<statement1>

...
END SUB

This command declares a subroutine, which takes on an independent form. The subroutine's parameter(s) can be passed along and their types (Byte, Integer, etc.) can also be declared.

The code for subroutine must be placed *after* the main routine of the BASIC program. The end of the main routine is marked by the END command. When you call a subroutine, you just need to write its name as if you are using a command with some parameter(s) optionally following it in parentheses.

```
SoundOut Do,W2 ' as if using a regular command
End ' There must be an END command at the end of the main routine.
Sub SoundOut(Fr As Integer, Wt As Integer)
    Freqout soundch,Fr
    Delay wt
    Pwmoff 0
    Delay 50
End Sub ' There must be an END SUB command at the end of a subroutine.
```

One can also write a subroutine without any parameter as below.

```
Sub
    Delay
End Sub
```

As above, if one specifies nothing inside the parentheses, it becomes a parameterless subroutine. One just needs to call this subroutine by writing its name DelayLong to run it.

Don't forget! To start defining a subroutine, you should first end the main routine with the END command. The CUBLOC compiler distinguishes the main routine and a subroutine by detecting the END command. Thus, the END command is very important.



Fun Trip

to the **CUBLOC World**

APPLIED TASK

- 1 Change the program so that it will perform a different song.
- 2 Make it so that pressing a key will produce a touch sound.
- 3 Make the program sound an alarm (as an ambulance does).

Job well done!

You have learned the principle of how a sound is output along with how to use the PWM and FREQOUT commands. Make sure you master the Subroutine part so that you can compose compact source programs from now on. This technique will be used frequently.



COMMANDS LEARNED IN THIS CHAPTER

REVERSE

CONST

PWM

PWMOFF

SELECT.CASE

#DEFINE

BYTEIN

FREQOUT

SUB



IN-DEPTH ANALYSIS: Sub, Function Command

The Sub command defines a subroutine. Function is a similar command that defines a function. The difference is that a function has a return value while a subroutine does not.

```
Const Device = CB280
Dim A as Integer
Delay 500p
A = AplusB( 10, 5)
Debug Dec A
End
```

With parameters 10 and 5, we call a function named AplusB.

15 will be outputted on the Debug Terminal window.

```
Function AplusB( PA As Integer, PB As Integer)
    AplusB = PA + PB
End Function
```

If some value is stored in the function name, that value will be the function's return value.

Some value can be stored in the function name within the function's definition. Then that value becomes the function's return value. In fact, the return value is passed along when the last statement of the function definition, i.e., END FUNCTION, is executed.

One can also declare the type of the return value as below. All types (Byte, Integer, Long, Single and String) can be used, and when omitted the type defaults to Long.

```
Function AplusB( PA As Integer, PB As Integer) as Integer
    AplusB = PA + PB
End Function
```

This part declares the type of the function's return value.

We call the source codes of subroutines and functions are *subprograms*. Subprograms can define their own variables. Variable defined only in subprograms are called *local variables* while as variables defined in the main routine are called *global variables*. They are used in a similar manner; but their respective *scopes* (i.e., the range of their effectiveness) are different.

```
Const Device = CB280
Dim A as Integer
Delay 500p
A = AplusB( 10, 5)
Debug Dec A
End
```

Variable A is a global variable.

```
Function AplusB( PA As Integer, PB As Integer)
    Dim C As Integer
    C = PA + PB
    AplusB = C
End Function
```

Variable C is a local variable.

A global variable can be used all over the source program. It can be used even within a subprogram.

On the other hand, a local variable can only be used within its own subprogram where it was defined.

I can go all around the world!



Global BUS

I can only drive around L.A.!



L.A. Local BUS

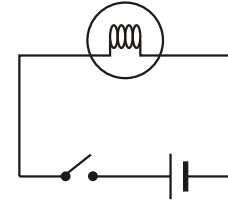


**Fun Trip**to the **CUBLOC World****HELPFUL TIPS**

Analog and Digital

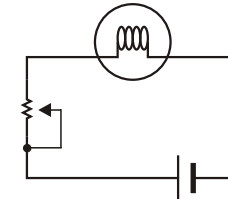
In digital system, binary notation is used to express only two states: on or off.

As in the right-hand side figure, binary notation usually expresses two states: on or off (or bright or dark).



On the other hand, as in the second right-hand side figure, if a variable-resistance is used to do the "dimmer" control of the light bulb's brightness, the result can no more be expressed by mere two states of on or off. We need some way to accurately express all levels of brightness between 100% on and off.

Analog is a quantity (as in this situation) which has to be represented by a continuous-state variable.



Information we access in everyday life are in general analog: temperature, weight, height, etc. Since computers are based on digital processing, they cannot direct use analog input; so analog input has to be converted to digital input. Analog-to-Digital Converter (A/D Converter or ADC in short) handles this task.

CUBLOC's ADC reads a certain voltage and converts it to a digital number. For example, it reads a voltage in the 0 to 5 volts range and outputs a number between 0 and 1023. 0 V is converted to 0, 2.5 V is converted to 511, and 5 V to 1023. CUBLOC's ADC "converted voltage into a number."

Source Program

Type in the source program as below.

```

CUBLOC studio [ d:\자료실\테스트\cubloc\초보여행\test.cul ]
파일(F) 편집(E) 디바이스(D) 실행(R) 설정(S) 도움말(H)
[F1] BASIC [F2] LADDER Ladder Mnemonic

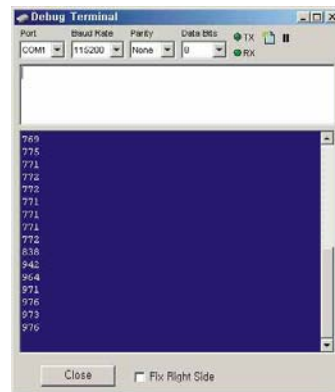
Const Device = CB280
Dim I As Integer
Do
    I = Adin(0)
    Debug Dec I, CR
    Delay 200
Loop
  
```

Confirmation Run

Click the RUN button to download and run the source program.

Now turn the first volume dial.

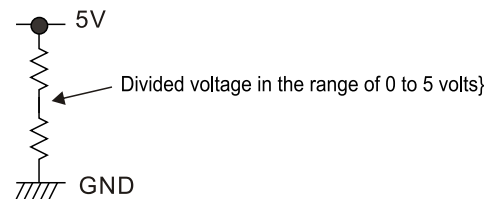
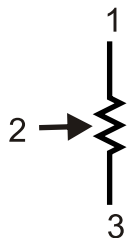
You will see how the numbers displayed on the Debug Terminal change.



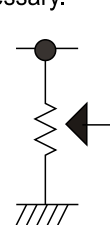
PARTS STORY

Volume Dial (a variable resistance)

Differing from usual resistance whose resistance value is fixed, a variable resistance can vary its resistance value. As can be seen in the picture below, there is a handle (or a dial) on a variable resistance.



In a circuit as shown, if two different resistances are serially connected and voltage is applied to its two ends, between the two resistances, an intermediate voltage occurs related to the ratio of the two resistances. If one wants to produce a voltage between 0 and 5 volts, one just needs to find two appropriate resistances to produce such a *divided voltage*. If one does not want to pick two different resistances every time but would like to have a varying voltage value, then you need a variable resistance. A variable resistance is used when a third voltage, i.e., a divided voltage, is necessary.





Adin

<variable> = ADIN(<channel>)

After performing A/D conversion at a specified channel, its value is stored in the variable. Since the A/D channel number and the port number can differ among different CUBLOC models, one should check the pinout diagram.

The I/O pin to be used must be left in an input state for later A/D usage. Since the default state of CUBLOC's I/O pins are input state after CUBLOC powers on, unless they are changed to output state in the middle all channels are ready for A/D input after the CUBLOC is powered on.

```
Const Device = CB280
Dim A as Integer
Input 24      ' Channel 0 is Port 24
A = Adin(0)   ' A/D conversion at Channel 0 and store the value in variable A.
```

TAdin

<variable> = TADIN(<channel>)

After performing A/D conversion at a specified channel, its value is stored in the variable.

This command performs almost the same as the ADIN command.

The only difference is that the average of 10 converted values are returned for better stability. This technique is effective when the input values sway due to noise, etc.

Note that the conversion time takes a bit longer with this method.

```
Const Device = CB280
Dim A as Integer
Input 24      ' Channel 0 is Port 24
A = TAdin(0)  ' Perform A/D conversion at Channel 0 and store the value
               in variable A.
```

Explanation of the Source Program

```
Const Device = CB280
Dim I As Integer
Do
    I = Adin(0)
    Debug Dec I, CR
    Delay 200
Loop
```

ADIN command performs the A/D conversion. In the parentheses the A/D channel number is specified. The location of A/D channel's ports is different among different CUBLOC models. In case of CB280, A/D Channel 0 is Port 24.



TIPS More Stable A/D Conversion

When you run the previous program, you will see that the numbers displayed on the Debug Terminal slightly varies even when you did not make any changes.

This is caused by the minor errors occurring during the A/D conversion and the minute change of the voltage.

To reduce such errors, people use the technique of displaying the average value over multiple converted values. With CUBLOC, when the TADIN command is executed, automatically the average of 10 converted values is returned.

In the program above, one can see that more stable values will be displayed by simply replacing ADIN with TADIN.

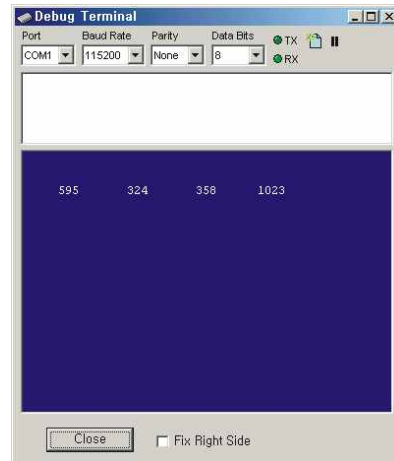
Applied Programming-1

The previous program displayed the A/D converted value of only one volume dial.

What if one wants to see the converted values of all 4 volume dials?

```
Const Device = CB280
Dim I As Integer
Do
    I = Adin(0)
    Debug goxy,3,2
    Debug Dec5 I
    I = Adin(1)
    Debug goxy,13,2
    Debug Dec5 I
    I = Adin(2)
    Debug goxy,23,2
    Debug Dec5 I
    I = Adin(3)
    Debug goxy,33,2
    Debug Dec5 I
    Delay 200
Loop
```

When one inputs the program on the left-hand side, 4 numbers will be displayed on the Debug Terminal as below. And when the 4 volume dials are turned, the numbers will change on the screen.



HELPFUL TIPS

Formatted Display of Numbers

DEC is a "converter" that converts a number into a decimal number. E.g., "DEBUG DEC A" outputs the value in variable A as a decimal number.

```
Debug Dec A 'A=10
Debug Dec A 'A=100
```

10
100

In the above, if the number to be outputted is 10, it takes up 2 places; if 100, it takes up 3 places. Since the number of places a number occupies when outputted can vary, this may create an awkward situation. Thus, CUBLOC provides a way to format the outputted numbers. "Formatted Output" means a number will take up a pre-specified number of places no matter how big or small the number is.

```
Debug Dec5 A 'A=10
Debug Dec5 A 'A=100
Debug Dec5 A 'A=100
```

10
100
10000

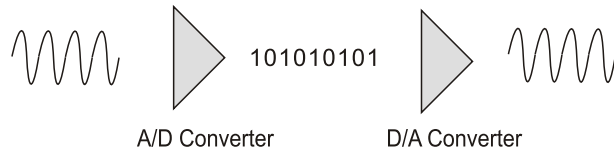
By simply attaching a total number of places to "Dec" without any blank, the outputted numbers are displayed in right-adjusted fashion.



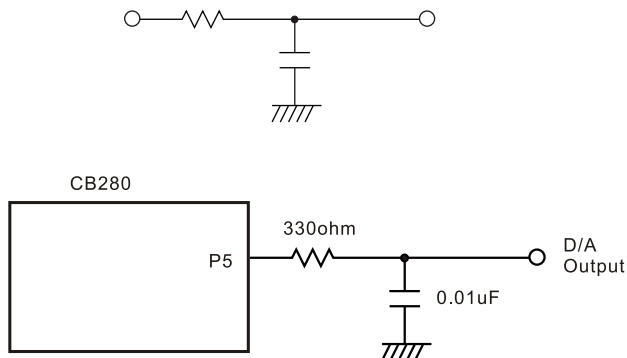
If you would like to display the numbers in a right-adjusted manner, you can use "Dec10".

Applied Programming-2

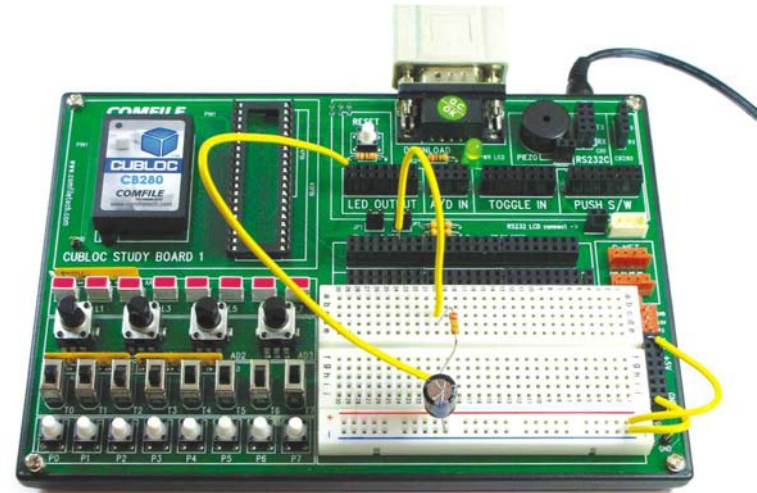
There are many applications where an analog quantity is A/D converted and then D/A converted before its output. E.g., the CD (Compact Disc) we use to listen to music is digital recording of analog sounds. To listen to this digital recording, we need D/A conversion.



Remember the PWM command, we learned before?
 The PWM command can be used as a D/A converter.
 Shall we create a program that D/A-converts A/D-converted values using the PWM?
 With CUBLOC one can implement D/A conversion using only minimal components such as resistances and condensers.



Let's conduct an experiment after we configure the Study Board as below.
 Let us use a breadboard to connect a condenser and a resistance.
 Also connect to an LED to visually confirm the result.



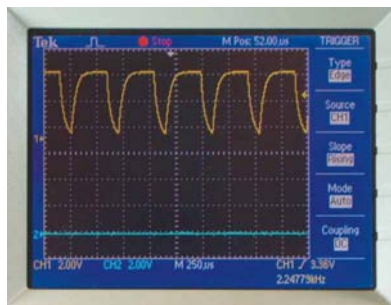
```
Const Device = CB280
Dim I As Integer
Low 5
Do
    I = Tadin(0)
    Pwm 0,I,1024
    Delay 200
Loop
```

Now, input this program, run it, and turn the volume dial. The LED should light up as you turn the dial.

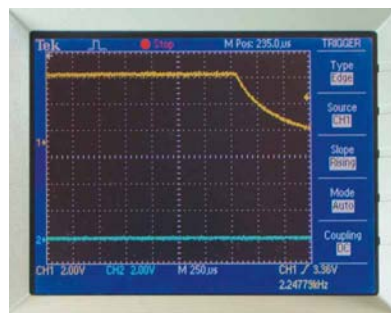
After you A/D-converted the dialed volume value (which was initially analog), that converted value is outputted using a PWM, and then this value is smoothed using a condenser and a resistance, finally to light up the LED.

Condenser is an electric part that temporarily stores electricity. There are condensers that store electricity for a very long time, too, which are called "rechargeable batteries." Most condensers store electricity for a short period of time: several microseconds or nanoseconds. Then some may think, 'What use could it be if it can store it for such a short time?' But that is the very point of its usefulness! Condensers are useful because they can hold up only a very small amount of current.

In the previous circuit, to convert the PWM signal into a constant voltage signal, a condenser is needed to store electricity. But if one uses a condenser with too small a capacity, a waveform on the right-hand side will be produced.



If a condenser with too large a capacity is used, the output wave will go on even after the input PWM signal is stopped as shown on the right-hand side.



Therefore, when a condenser of an appropriate capacity is used, the PWM wave (i.e., signal) will be converted to a stable voltage, and when the PWM wave is stopped, the output wave will also soon stop, as the electricity stored in the condenser depletes in a timely manner, thus making a useful D/A converter.

There are many kinds of condensers:

Mylar condenser, ceramic condenser, electrolyte condenser, monolithic condenser, multilayer ceramic condenser, etc.

In the digital circuit that deals with CUBLOC, ceramic, electrolyte and multilayer ceramic condensers are used.

The rest you need not know, as they are used in very complicated, high-frequency, RF (radio frequency) or analog circuits. (Knowing them can only be a headache.)



On the left-hand side are condensers that can usually be seen in power circuits. Since they have polarity (i.e., they have distinguished + and - terminals), make sure you connect them correctly. Since these kinds have bigger capacity than ceramic condensers, they are used in power circuits.



On the left-hand side is a condenser that has a relatively small capacity and has no polarity. It is used as a "bypass condenser" which is attached close to a semiconductor's power terminal.

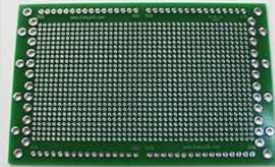
Bypass condenser is a condenser that is used in preparation for an instantaneous power consumption. It has a capacity of about 0.01 microfarad and is attached somewhere near the power pin.



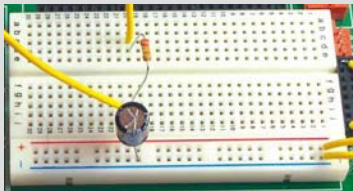
PARTS STORY

Breadboard

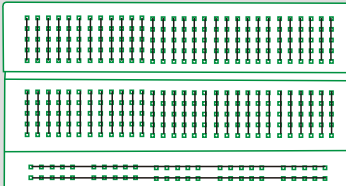
Though the Studyboard has many components such as switches and LEDs already built in, if you would like to carry out an experiment which requires components that are not already included, you will need to create a separate circuit using something like a *universal board*.



But using a universal board is cumbersome, as one needs to manually solder and it is difficult to revise if one makes a mistake. Therefore, the Studyboard has a built-in *breadboard* instead. On a breadboard's many holes one can insert components such as resistances, condensers and other chips. Inside the breadboard are lines running vertically and horizontally that are invisible from the outside as illustrated below. One can create a circuit considering this line connections.



Breadboard is very useful for initial development.



APPLIED TASK

- 1 Create a program where the 8 LEDs display the A/D-converted value when the volume dial is turned.
- 2 Create a program where the pitch of the piezo's sound changes when the volume dial is turned.

Job well done!

You have learned the difference between analog and digital, also the methods to convert one into the other.

CUBLOC is a "microcomputer" based on the digital principles.

Analog is like everyday quantities such as temperature, humidity, voice, brightness, etc. It is very important to know the difference between analog and digital to use CUBLOC well.



COMMANDS LEARNED IN THIS CHAPTER

ADIN
TADIN
DEBUG CLR
DEBUG GOXY
DEC5



IN-DEPTH ANALYSIS: DEBUG Command

The Debug command comes very handy in CUBLOC Basic, as it has many usages.

One should NOT think it merely display a variable's value on the PC screen!

It confirms whether some part of the program was executed or not

If one places a Debug statement among a piece of source code, which is never run, naturally the Debug statement will display nothing on the Debug Terminal.

```

Do
  Byteout 1,A
Loop
Debug "END"      ' This Debug command is not executed.

```

If you are not as to whether a certain part of a source code is executed or not, place a Debug command there and have anything displayed. If that part of the code is executed, it will be displayed; and if not, nothing will be displayed.

It is utilized as the program's display window

For those who do not have an LCD or a 7-segment display, the Debug Terminal can be used as your display instead.

To specify your display position, you can use "Debug goxy, x, y" where x and y are x and y coordinates, respectively.

To clear the screen, you can use "Debug Clr".

This way, you can freely output characters and numbers on a screen larger than a usual LCD.

As you create CUBLOC programs, you get to use the Debug command here and there.

But after you complete the program, you won't need them any more.

At this time, you don't need to go after all occurrence of Debug to delete them.

You just need to write the statement "Set Debug Off" at the beginning of the program, which cancels the effects of all Debug commands.

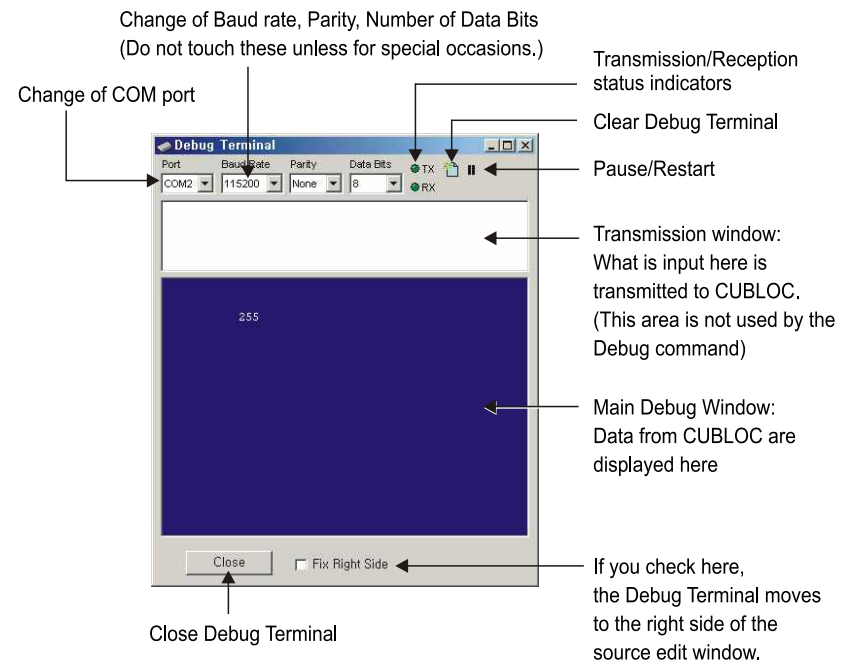
It is as if they don't exist any more. Later, when you need its effect again, you can just delete this one statement.

```

Const Device = CB280
Set Debug off      'All Debug commands stop working

```

How to use the Debug Terminal





HELPFUL TIPS

Basic Grammar for BASIC & Its libraries

CUBLOC BASIC language has over 140 commands.

As seen in several of the previous example, some commands are closely related to the hardware configuration. For instance, ADIN and PWM do not belong to standard BASIC language. We call these kinds of commands "Libraries." Libraries are such that one does not need to memorize all of them because they can be used by simply reading their user guide when they are needed.

The commands you should know are the ones like IF, THEN, FOR, NEXT, etc., which are part of the standard BASIC language.

Below is a table summarizing the basic commands that one must know. Most them were already covered. If you can conceive that these basic commands form the backbone of BASIC and actual operations happen through libraries, you will be able to learn BASIC more easily.

Variable Declaration	Dim, Byte, Integer, Long, Single, String	These commands secure appropriate memory space for specified variable types.
Branch Command	Goto	These commands change the point of program execution.
Subprogram	Gosub, Return, Sub, Function	These commands call subroutines and define/declare them.
Loop Control	For, Next, Do, Loop, While, Until	These commands manage repeated execution within the program.
Conditionals	If, Then, Select	Depending on whether the given condition is met, these commands execute one of the optional source code.
Constant Declaration	Const,	These commands give often-used number a name/label.
Operators	+, -, /, *, Mod	These commands perform arithmetic and other operations.
Interrupts	On Gosub, Set	These commands declare the kind of interrupt (i.e., a sudden event) and subroutine that will handle the interrupt, or change a setting.



HELPFUL TIPS

CUBLOC's BASIC Language

CUBLOC's BASIC Language has a grammar very similar to Microsoft's Visual BASIC. Yet, since there is no 100% compatibility between the two, not all Visual BASIC programs will run on CUBLOC. In fact, CUBLOC BASIC is not as complicated as Visual BASIC. This is because Visual BASIC has numerous commands to control all kinds of PC components while there is not much to control on CUBLOC. Since CUBLOC is *embedded controller*, it is equipped only with necessary commands.

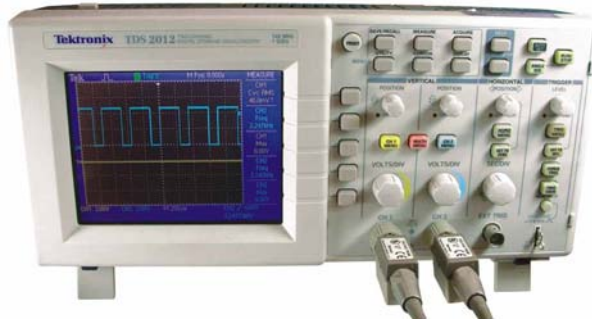
CUBLOC BASIC is easier to learn than other versions of BASIC languages because it has a compact vocabulary (i.e., commands).



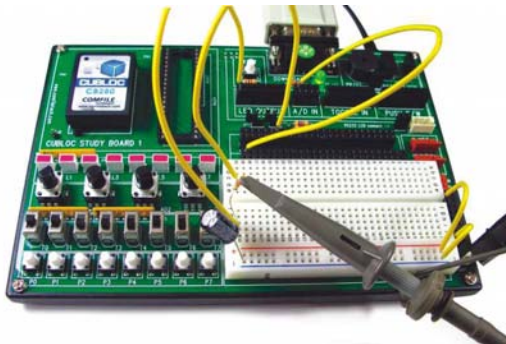


Let's use Oscilloscopes!

Most software programmers and PLC (Programmable Logic Controller) users are not familiar with using oscilloscopes. Oscilloscope is a device that makes wave changes visible and is indispensable to electronic engineers.

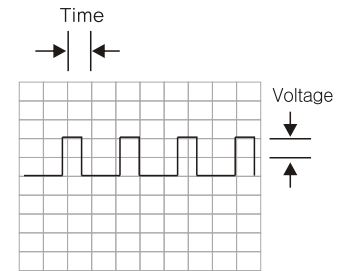
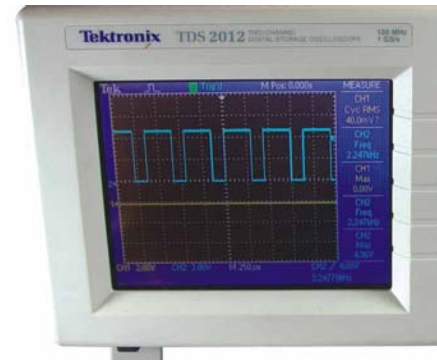


The above is a photograph of a Tektronix oscilloscope. Beside this company's, there are many other brands and kinds. It is very simple to use.

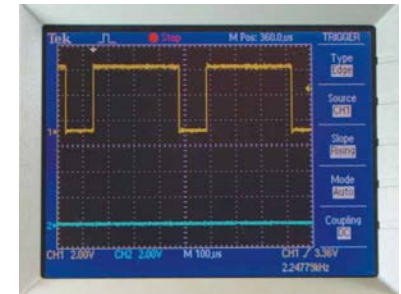
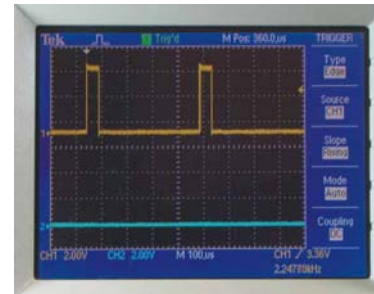


First, have the big foot of the probe clutched at GND (i.e., ground the big foot), and put the end of the probe to a spot where you would like to make a measurement.

The vertical axis of the screen indicates voltage and the horizontal axis represents time. If you see the screen, you can tell how much quantity each grid represents.



Shall we measure the PWM wave we generated with one of the previous programs? If we adjust the voltage knob and the seconds knob to make the waveform more understandable, waveforms as shown on the left-hand side below will show.



In the previous program, when the volume dial was turned the duty ratio of the wave changed, right? Now, let's try that right now. If we use an oscilloscope, we can witness the changing duty ratio right in front of our eyes! Such changes and waveforms cannot be seen using a tester; an oscilloscope must be used for such purpose. Make it a habit to use an oscilloscope to see the waveform.



HELPFUL TIPS

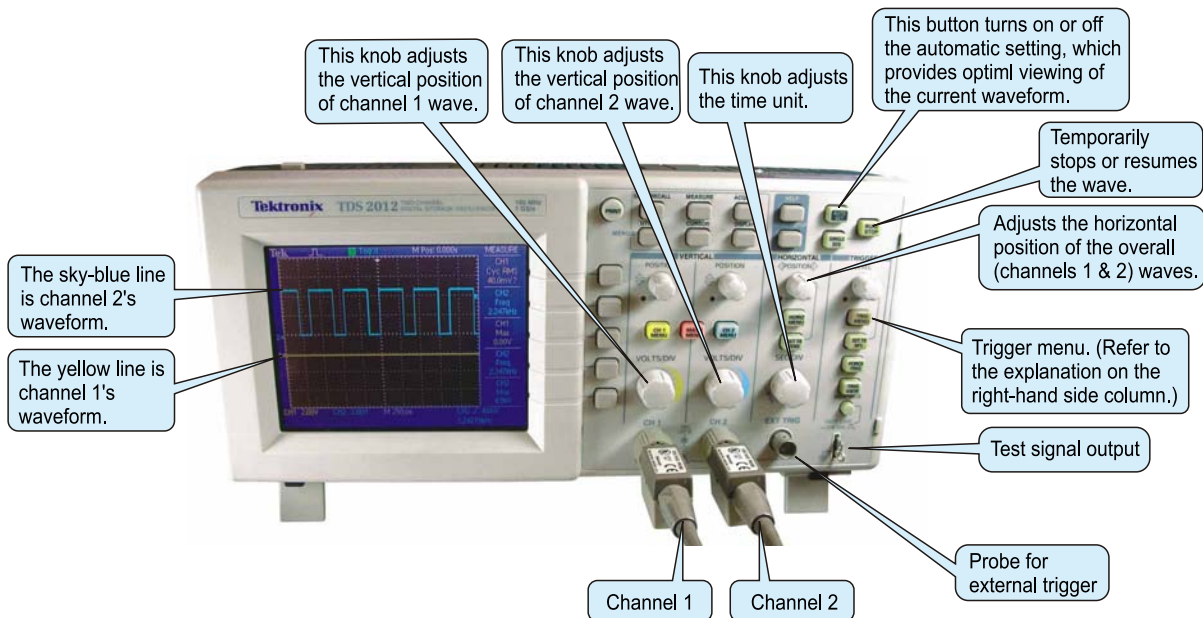
How to use an Oscilloscope-Summary



Fun Trip

to the CUBLOC World

This is a summary of how to operate frequently used components of an oscilloscope.
Oscilloscopes made by other makers(other than Tektronix) can be operated in a similar manner.



HELPFUL TIPS

Trigger?

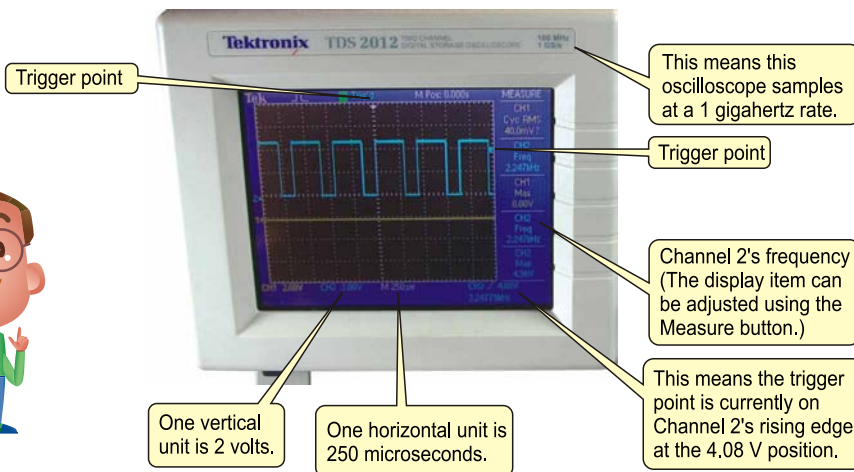
"Trigger" is a small lever which can be pulled to fire a gun. Here in this context it refers to a point from which a signal is detected. For a repeating sine wave with a fixed period, trigger does not take on a significant meaning.

Yet, if you would like to capture a point where a signal in low state suddenly rises to a high pulse, you need to specify a trigger point. You need to specify which position on which channel, and whether it is on the rising edge or the falling edge. To set up a trigger point, one needs to push the Trigger Menu button and set the details.

When the oscilloscope finds a trigger point, it pauses the screen for a short time so that the user can confirm it, or the screen can be stopped indefinitely.

Though oscilloscope is a useful device to observe waveform directly, if one cannot accurately pinpoint the point one needs to look at, it will be of no use.

Thus, it is important to familiarize oneself with the Trigger Menu so that the exact point of interest can be accurately located.



There's a limit as to what a tester can do.

To become an advance user, an oscilloscope is a must have!



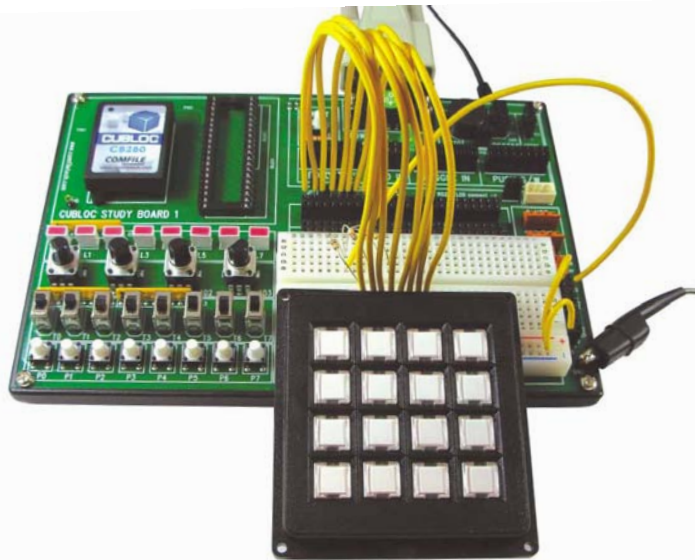
OPERATIONAL SUMMARY

What must be done to receive 16 inputs simultaneously? Should one attach a switch to each I/O port as in the "Digital Piano" example? Doing so would be inefficient, as too many ports are used. In this lab, we will study multi-switch input circuit and program that utilizes the Key-Matrix feature.

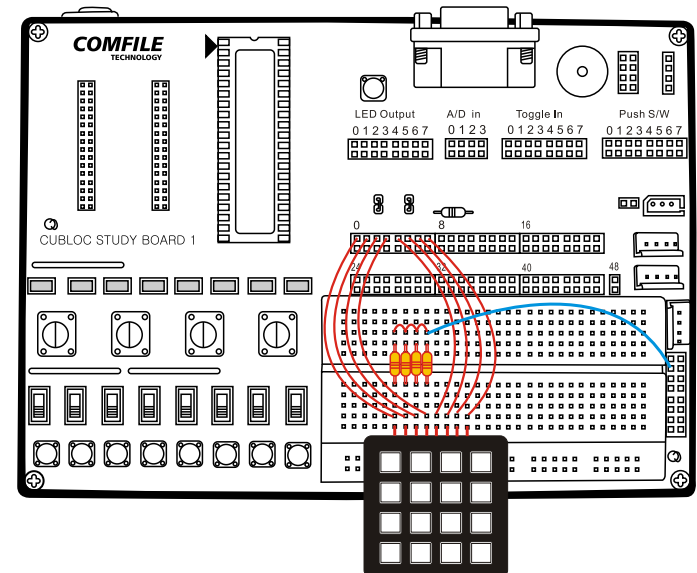
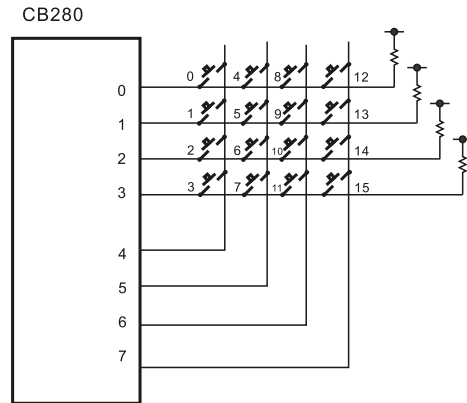
LAB.6 Keypad Input

Circuit Configuration

Connect a 4x4 keypad to the breadboard and connect 8 pins to Port Block 0 in order. Ports 0, 1, 2 and 3 should be pulled up using 10 kOhm resistances, respectively. Since a 4x4 keypad is not included in the Start Kit, one needs to separately acquire it at www.comfiletech.com.

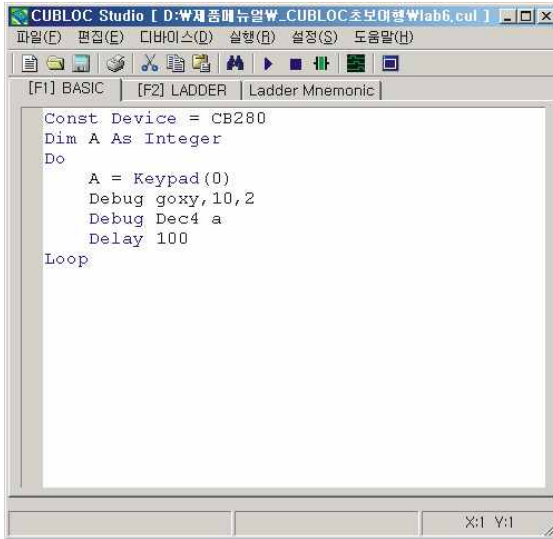


Circuit Diagram



Source Program

Input the source program as in the screen below.



Explanation of the Source Program

Const Device = CB280

Dim A As Integer

Do

A = Keypad(0)

Debug goxy,10,2

Debug Dec4 a

Delay 100

Loop

Read key matrix from Port Block 0.

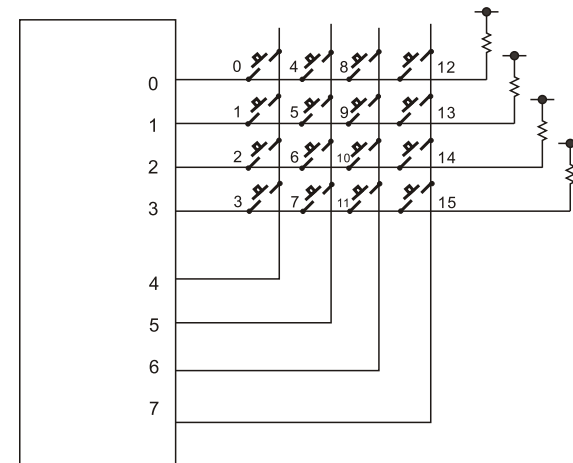
Move the output (x, y) position on the Debug Terminal to (10, 2).

Display variable A's value as a 4-place decimal number.

Delay execution for 100 milliseconds.

KEYPAD command reads the states from a key matrix (i.e. keypad).

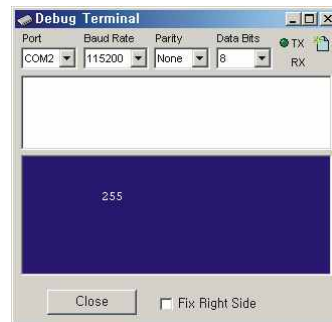
Key matrix is a configuration where multiple keys are meshed in a matrix formation as in the next figure.



To use the KEYPAD command, one has to use only one port block and the input ports (i.e., the lower 4 bits) have to be connected to individual pull-up resistances. When a key is pushed on the keypad, the "scan code" corresponding to that key is outputted.

Confirmation Run

If you push a key on the keypad and a number is displayed on the Debug Terminal, then you have succeeded. If no key is pressed, 255 is supposed to be displayed.



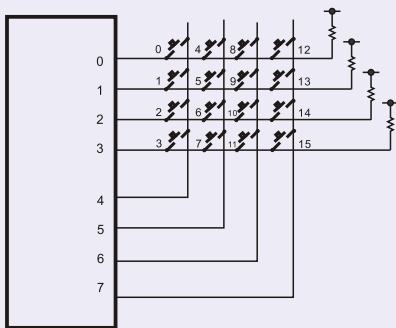


NEW COMMANDS

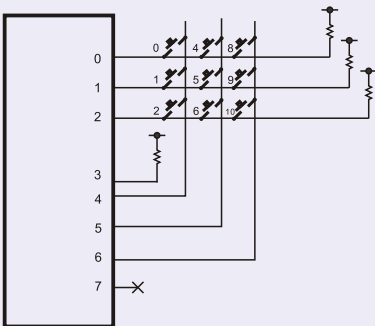
Keypad

<Variable> = KEYPAD(<Port Block no.>)

This command reads a key code value from a key matrix connected to one of the port blocks. If no key is pressed on the key matrix, value 255 is returned. If there is a key pressed, the scan code of the corresponding key is returned. The input side ports (i.e., the lower 4 bits) must be connected to pull-up resistances, respectively.



Up to a key matrix of 4x4 size can be connected. If a 3x4 or 2x3 matrix is connected, any unused input port must be connected to a pull-up resistance, and any unused output port must be left in the current "open" state.



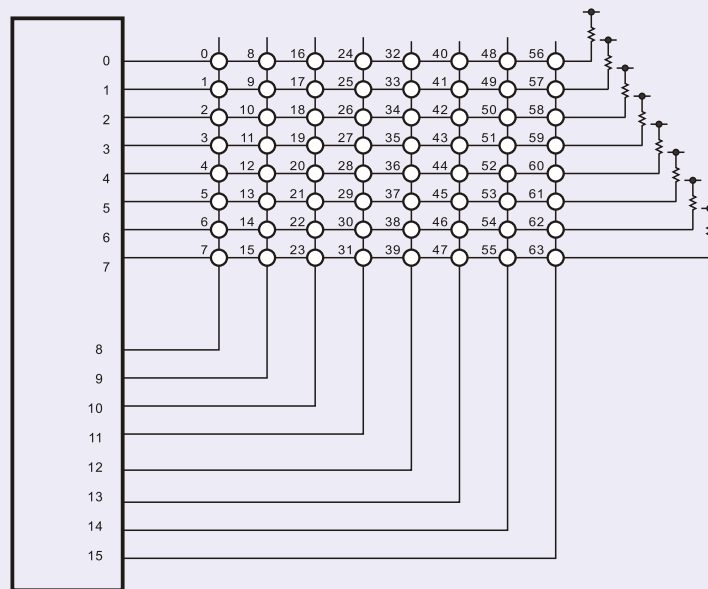
Then how would one connect a 8x8 key matrix? CUBLOC is equipped with a command to accommodate matrices larger than 4x4: EKEYPAD. Its functions are very similar to the KEYPAD command. But to accommodate an 8x8 matrix, one needs to use 2 port blocks.

EKeypad

<Variable> = EKEYPAD(<Input Port Block no.>, <Output Port Block no.>)

This command reads a key code from a key matrix as large as 8x8. To use this command at least 2 port blocks (i.e., 16 ports) should be used. The input port blocks, whether used or unused, must be connected to pull-up resistances. And unused output ports can be left in the current open state.

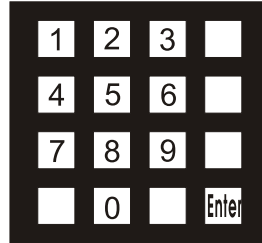
Example: A = EKEYPAD(0, 1)





Applied Programming-1

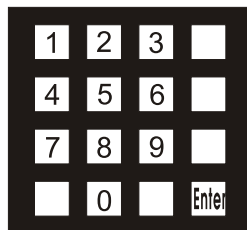
Let us create a program which will



This has the same numbers arrangement as on the phone's dial-pad. One has to match the numbers and scan code values, respectively, If one uses IF statements to accomplish this goal, it will be too inefficient.

```
ScanCode = Keypad(0)
If ScanCode = 0 then A = 1
If ScanCode = 1 then A = 4
If ScanCode = 2 then A = 7
```

Could there be a simpler, easier way?
One could use a constant array.



Actual keypad labels

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

Corresponding scan codes

Scan codes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Keypad values	1	4	7	11	2	5	8	0	3	6	9	12	13	14	15	16

Now let us use a constant array to create a program that displays the corresponding keypad key when a key scan code is pressed.

```
Const Device = CB280
Const Byte Ktbl = (1,4,7,11,2,5,8,0,3,6,9,12,13,14,15,16)
Dim A As Integer
Dim K As Integer
Do
    A = Keypad(0)
    If a = 255 Then
        k = 255
    Else
        k = Ktbl(a)
    End If
    Debug goxy,10,2
    Debug Dec4 k
    Delay 100
Loop
```

The keyboard for IBM brand PCs use the same key matrix method!



The statement beginning with “Const Byte” is where the constant array is defined. As such, a constant array is used when one needs tabular translation. Constant arrays can be used like general (i.e., variable) arrays with the exception that its content cannot be changed.



NEW COMMANDS

Const Array

**Const <Type> <Array Name>
= (<a list of constants>, [<a list of constants>])**

“Constant” is a value that does not change during the program's execution. Defining multiple constants in an array form is called “Constant array.” Constant arrays are used for table lookups or conversion of mass data. The elements of a constant array can be used in the same manner as those of a variable array except that the value of each element cannot be altered.

```
Dim A as Long, B as Long
Const Byte data1 = (31,25,102,34,1)
A = data1(0) '31 is stored in A.
B = data1(1) '25 is stored in B.

Const Byte data2 = (" CUBLOC SYSTEM")
A = Data2(0) 'The ASCII value of "C", &h43, is stored in A.
```

If one stores a (character) string as a Byte type constant array, one can reach each character as an element of the array. In the above situation, “C” is in data2(0) and “U” is in data2(1).

When declaring the type of the constant array, one should keep the data size in mind. When elements include a negative number, the array type should be Long. When there is a value larger than 255, the array type should be Integer or Long.

```
Dim A as Long, B as Long
Const Integer data3 = (1234, 342,1, 12000, 21)
Const Long data4 = (1238238, -128328, -1, 0, 12323)
Const Single Data5 = (32.1, 3.14, 0.12)
```

Each element of the constant array can be a string. This is different from the case above where each element was a character. In this case, the maximum length among the string elements should be specified. Any string element longer than the specified maximum length will be truncated.

```
Const String * 6 data6 = ("COMFILE", "BASIC", "ERROR")
```

We have just converted the key scan codes into the values we want, respectively; we have not completed the program, yet. For a number to be stored at a variable, the number on the pressed should be displayed continuously and when the Enter key is pressed the number should be stored at the variable.



We see that multiple keys are placed in a matrix form, and each key when pressed are designed to trigger a special function.



HELPFUL TIPS

The Differences between the Constant Array and General Array

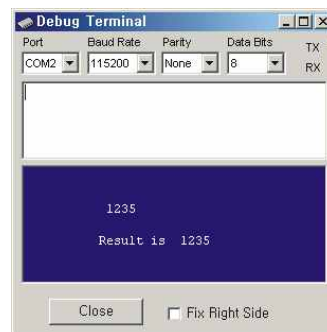
Items	Array	Const Array
Storage place	Data Memory(SRAM)	Program Memory(Flash memory)
When the value is recorded	When the program is run	When downloaded
May be changed while the program runs	yes	no
Use Purpose	Storage of data that may change during execution	Storage of constants that does not change during execution
What happen when power is off	The information is lost	The information is preserved



```

CUBLOC studio [ d:\W제 품매뉴얼\W_cubloc초보매뉴얼\lab6.2.cul ]
파일(F) 편집(E) 디버깅(D) 실행(R) 설정(S) 도움말(H)
[F1] BASIC [F2] LADDER Ladder Mnemonic
Const Device = CB280
Const Byte Ktbl = (1, 4, 7, 11, 2, 5, 8, 0, 3, 6, 9, 12, 13, 14, 15, 16)
Dim A As Integer
Dim K As Integer
Dim Res As Long
Dim M As Long
Ramclear
Do
  A = Keypad(0)
  If A < 10 Then
    ' Only number keys are handled here.
    k = Ktbl(A)
    M = M << 4
    M = M + k
    ' Stand by until the key is released.
    Do While Keypad(0) < 255
      Loop
    Res = Bcd2bin(M)
    Debug goxy, 10, 2
    Debug Dec5 Res
  ElseIf A = 15 Then
    ' Display the result when the ENTER key is pressed.
    Debug goxy, 10, 4
    Debug "Result is ", Dec5 Res
  End If
Loop
  
```

After the program is started, when a key is pressed a number is displayed on the Debug Terminal, and when the ENTER key is pressed "Result is " and the same resulting number is outputted on the next line.



```
Const Device = CB280
```

```
Const Byte Ktbl = (1, 4, 7, 11, 2, 5, 8, 0, 3, 6, 9, 12, 13, 14, 15, 16)
```

```
Dim A As Integer
```

```
Dim K As Integer
```

```
Dim Res As Long
```

```
Dim M As Long
```

```
Ramclear
```

This command resets all variable values to 0 (zero). On the other hand the Dim command only secures the memory space for variables being declared, and does not clear their values as the RAMCLEAR command does.

```
Do
```

```
  A = Keypad(0)
```

```
  If A < 10 Then
```

```
    '
```

```
    'Only number keys are handled here.
```

```
    '
```

```
      k = Ktbl(A)
```

```
      M = M << 4
```

```
      M = M + k
```

```
    '
```

```
    'Stand by until the key is released.
```

```
    '
```

```
      Do While Keypad(0) < 255
```

```
        Loop
```

```
      Res = Bcd2bin(M)
```

```
      Debug goxy, 10, 2
```

```
      Debug Dec5 Res
```

```
  ElseIf A = 15 Then
```

```
    '
```

```
    'Display the result when the ENTER key is pressed.
```

```
    '
```

```
      Debug goxy, 10, 4
```

```
      Debug "Result is ", Dec5 Res
```

```
  End If
```

```
Loop
```

This is a comment. If you start a statement with an ' (apostrophe), everything following on that line is treated as a comment, i.e., its content is not interpreted as part of the program, thus not affecting the program's running.

While the condition after the WHILE command is true, the statement(s) between the DO and the LOOP are repeated. In this case, since there is no statement between the two, this simply pauses the execution very briefly. This is to prevent further key input until the key scan code result is 255, i.e., the currently pressed key is released.

While the key is being inputted, it is recorded in the BCD (binary-coded decimal) format. Since BCD format represents one decimal place (i.e., 0 to 9) with 4 bits, it is easier to process. The final result should be converted to a binary value using the BCD2BIN function before it is stored on the memory so that it can be used with other binary values in binary operations!



NEW COMMANDS

Ramclear

RamClear

This command clears all variables' value to be 0 (zero).

Bcd2bin

<variable> = BCD2Bin(<BCD code>)

This command is a function that converts a BCD-coded number to its binary value. (Binary value is its usual format as it would be stored in a CUBLOC variable.) That is, in A = 100, 100 is stored as a binary number in variable A. That way, its value can be used readily in any binary operation.

BCD code is a representation that expresses one decimal place with 4 bits, a Nibble. Converting a binary number into a decimal number needs a separate procedure; but doing so with a BCD number is very simple, as one just needs to read groups of 4 bits (i.e., one nibble unit).

3	4	5	1
0000	1101	0111	1011
0	D	7	B

3	4	5	1
0011	0100	0101	0001
3	4	5	1

Expressing a decimal number 3451 as a binary number is as follows. If you look at its binary representation, it is hard to tell it is 3451 in decimal. You will have to do the binary to decimal conversion.

But if 3451 is represented in BCD format as below, one can tell the 4-bit groups represent 3, 4, 5, 1 rather easily. For such easy readability we use BCD representation.

There is a reverse function of BCD2Bin, call Bin2BCD, also, which converts a binary number into a BCD representation.

APPLIED TASK

- 1 To number keys 0 to 9, add A, B, C, D, E and F so that a hexadecimal number can be inputted.
- 2 Make pressing a key produce a beep sound.
- 3 Implement a "repeat" feature, where keep pressing a key will result in repeated input of the same key.

Job well done!

Keypad input is the most primitive and basic user interface used in many devices. Make sure you understand the principle of key input and its implementation method.



COMMANDS LEARNED IN THIS CHAPTER

KEYPAD
EKEYPAD
CONST BYTE
CONST LONG
CONST INTEGER

CONST STRING
RAMCLEAR
BD2BIN
BIN2BCD



OPERATIONAL SUMMARY

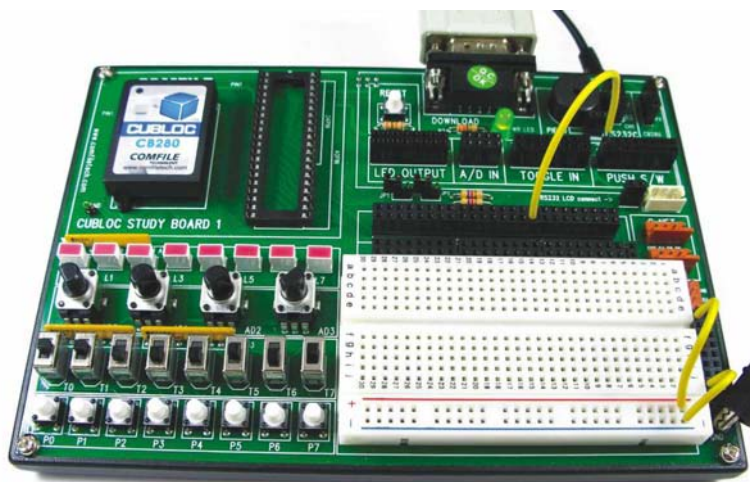
What should one do to count externally generated pulses? You can count them using the commands you have learned so far. But such method requires checking the I/O ports all the time and CUBLOC won't be able to do anything else. If CUBLOC is just counting pulses and can't do anything else, what an inefficient use of CUBLOC would it be?

Therefore, CUBLOC has a feature that automatically counts the pulses. If one uses this feature, CUBLOC can do other tasks, too. In LAB7, we will learn issues related to the count feature.

LAB.7 Counter Input

Circuit Configuration

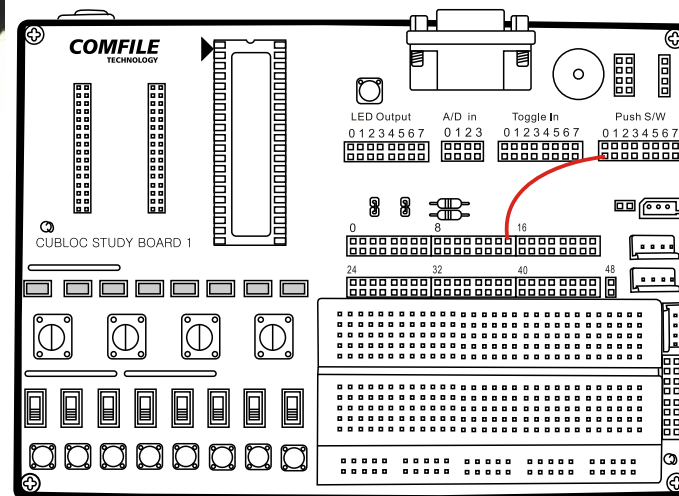
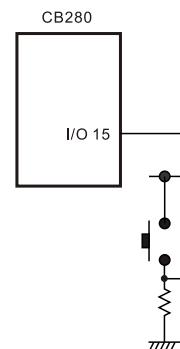
Connect the 15th I/O pin to the push switch.
The 15th I/O is the High Count 1 terminal.



SOUT	1	●	17	VDD	TX1	33	●	49	TTLTX1
SIN	2	●	18	VSS	RX1	34	●	50	TTLRX1
ATN	3	●	19	RES	AVDD	35	●	51	AVREF
VSS	4	●	20	N/C	N/C	36	●	52	P48
SS_P0	5	●	21	P16	ADC0_P24	37	●	53	P31_ADC7
(Input_only)SCK_P1	6	●	22	P17	ADC1_P25	38	●	54	P30_ADC6
MOSL_P2	7	●	23	P18	ADC2_P26	39	●	55	P29_ADC5
MISO_P3	8	●	24	P19_PWM3	ADC3_P27	40	●	56	P28_ADC4
P4	9	●	25	P20_PWM4_INT0	P47	41	●	57	P32
PWM0_P5	10	●	26	P21_PWM5_INT1	P46	42	●	58	P33
PWM1_P6	11	●	27	P22_INT2	P45	43	●	59	P34
PWM2_P7	12	●	28	P23_INT3	P44	44	●	60	P35
(CUNET)SCL_P8	13	●	29	P15_HCNT0	P43	45	●	61	P36
(CUNET)SDA_P9	14	●	30	P14_HCNT1	P42	46	●	62	P37
P10	15	●	31	P13	P41	47	●	63	P38
P11	16	●	32	P12	P40	48	●	64	P39

CB280


Circuit Diagram





Source Program

Enter the source program as shown below.



```

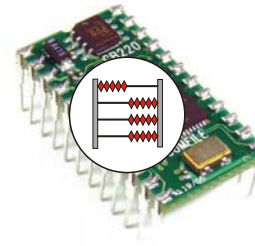
Const Device = CB280
Dim A As Integer
Input 15
Do
    A = Count(1)
    Debug goxy,10,2
    Debug dec5 A
    Delay 100
Loop
  
```

Explanation of the Source Program

```

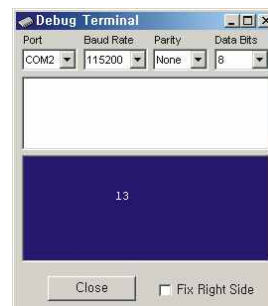
Const Device = CB280
Dim A As Integer
Input 15
Do
    A = Count(1)
    Debug goxy,10,2
    Debug dec5 A
    Delay 100
Loop
  
```

CUBLOC's internal counter is keeping the number of pulses coming in from high-speed counter channel 1. When function COUNT is executed, the internal counter returns the count it has kept so far. The program displays such return value on the Debug Terminal.



Confirmation Run

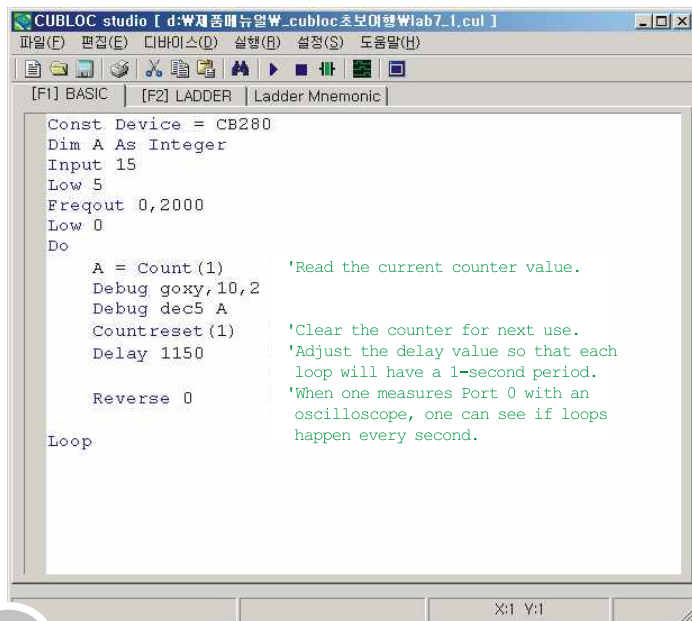
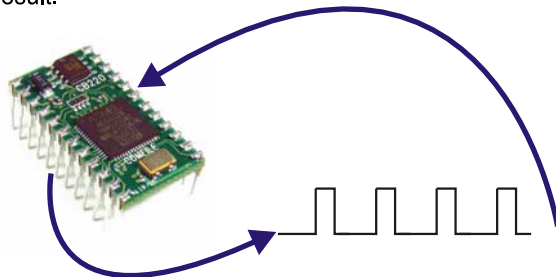
If you see the number displayed on the Debug Terminal increasing when you push the PUSH switch, your program has been correctly created.



Applied Programming-1

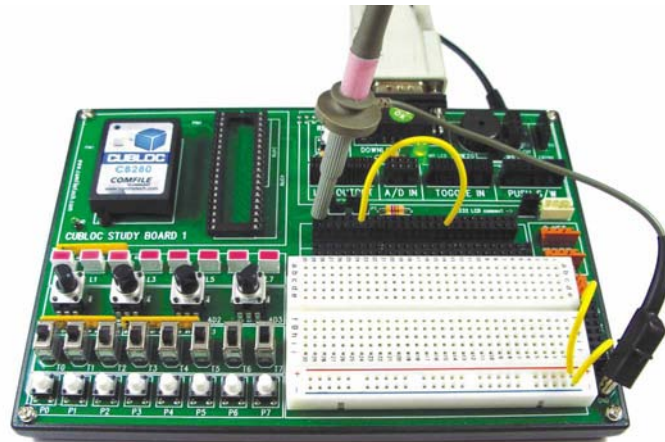
Let's create a program that counts the pulses incoming from outside. But wait a second. Previously we said CUBLOC has a feature that automatically generates pulses, right?

After we generate pulses using the FREQOUT command, let count them, and display the result.

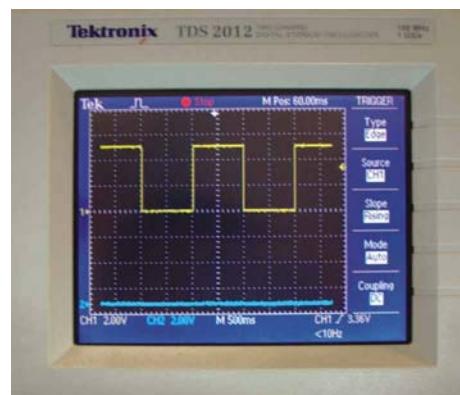


Enter this program and run it.

Connect Port 15 (i.e., the counter) and Port 5 (i.e., the PWM).



Making each loop's period 1 second is the most important goal in this program. Setting the delay to 1000 will not make the period exactly 1 second. Thus, we need to have Port 0 output a reversed voltage on each loop, and adjust the delay value while measuring the period between each loop. In this case, it seems that the period becomes 1 second when the delay is set to 1150.



Since it is written at the bottom of the oscilloscope's screen that each horizontal grid represents 500 milliseconds, 2 grids represents 1000 milliseconds, i.e., 1 second.



Applied Programming-2

Is there any other way to enforce the period to be 1 second besides adjusting the delay value?

When you used Delay command for this purpose, how was it inconvenient? That's right. Since DELAY does not enforce an accurate timing, to calibrate exact time)e.g., 1 second(trial-and-error method has to be used. And during the delay CUBLOC cannot do anything else!

Enter the following program and run it.

```
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) 'Read the current counter value.
Debug goxy, 10, 2
Debug dec5 A
Countreset(1) 'Clear the counter for next use.
Reverse 0 'Adjust the delay value so that each loop will have a 1-second period.
Return 'When one measures Port 0 with an oscilloscope, one can see if loops happen every second.
```

Now on the Debug Terminal 1150 is displayed. This means that 1150 pulses have been inputted during one second.

That is, the pulse has a frequency of 1.15 KHz. If you read the FREQOUT's description, the frequency is 1.15 KHz when "Freq 0, 2000" command is issued.

Wow! We have made a frequency counter using CUBLOC!

Frequency of a pulse generated outside CUBLOC can also be measured. Experiment what values are outputted when the arguments to Freqout command are changed! See if Freqout 0, 5236 outputs 440 Hz.

We see that we can implement a frequency counter merely by reading off value of the CUBLOC's built-in counter every one second.



NEW COMMANDS

Count

<variable> = COUNT(<Channel No.>)

This command returns a value stored in the counter.

The <channel no.> can be either 0 or 1. Since the port at which the counter channel is located differs among different CUBLOC models, one needs to check out the model's pinout. In case of CB280, channel 1 is at Port 15.

The following program shows the same run result but it uses the Timer interrupt to run the GetFreq subroutine every 1 second, which measures the frequency of the incoming pulse.

```
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
```

On Timer statement is a declaration indicating a beginning of an interrupt. When this command is issued, CUBLOC executes Gosub GetFreq at the specified time interval. The time unit of On Timer is 10 msec (milliseconds). Since 100 is specified here, the time interval is 1000 msec or 1 second. The Gosub statement will be executed every 1 second.

Based on the On Timer command this routine will be run just once every 1 second. This routine is called an "interrupt routine." After you write down all the statements to be executed, then the RETURN command must be added at the end.

The DO-LOOP follows the On Timer)100(command. Since the DO-LOOP has not contents inside, it is an infinite loop that does nothing. That is, it enters the CUBLOC into a standby mode.

The actual work is done in GetFreq. What this routine does was already explained above. It reads the counter value and displays it.

If some commands were included in the DO-LOOP, this program will carry out those commands while running the GetFreq routine every one second. That is, CUBLOC need not hold its breath while measuring the frequency; such can be done while handling other tasks.

This is what makes the timer interrupt attractive.

Yet, one precaution one has to take is that the timer interrupt routine must finish within the specified timer interval. For example, just imagine what would happen if there were a command such as "Delay 2000" inside the GetFreq routine!

Since the routine execution will take 2 seconds, raising an interrupt every 1 second would become meaningless. As a result, this program will keep running the interrupt routine.

When such a mishap happens, the behavior of the program deviates from its original purpose, the interrupt routine must finish within the timer interval.



NEW COMMANDS

On Timer

On Timer(<Interval>) Gosub <Routine Label>

This command generates an interrupt for every <interval> to execute the specified routine. The <interval> unit is 10 msec. E.g., 1 stands for 10 msec. The maximum interval value is 65535. When this command is specified more than once, only the last one becomes effective (while the previous ones lose their effects).



HELPFUL TIPS

Label Names

Labels are used as one way to indicate a location within a program.

A label can be used with a GOTO or a GOSUB command.

A label name can be either in English or Korean, and must be followed a colon (:). As a label, one cannot use ones such as Dim or For, for these reserved names are already used by BASIC.

Good labels: ABC:, Lab1:

Bad labels: 123:, Goto:, For:



NEW COMMANDS

Gosub, Return

Gosub <label>
<label>:
<statement>

...

Return

The Gosub command is a subroutine call command that can be used within a main routine. One can use Sub-End Sub to make it a subroutine, or it can be made subroutine simply by just using a label within the main routine. With Gosub, one cannot pass along parameters as in Sub, or declare local variables for later use.

```
Const Device = CB280
Delay 500
Gosub ABC
Do
Loop
```

Call the ABC subroutine.

```
ABC:
Debug "Done"
Return
```

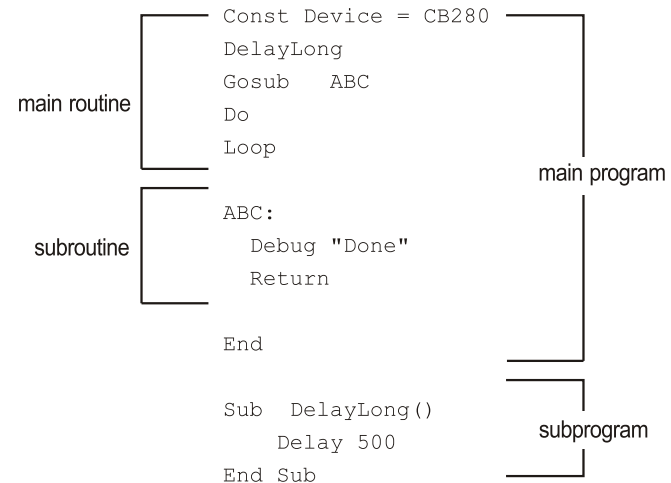
RETURN must be used at the end of a subroutine.



HELPFUL TIPS

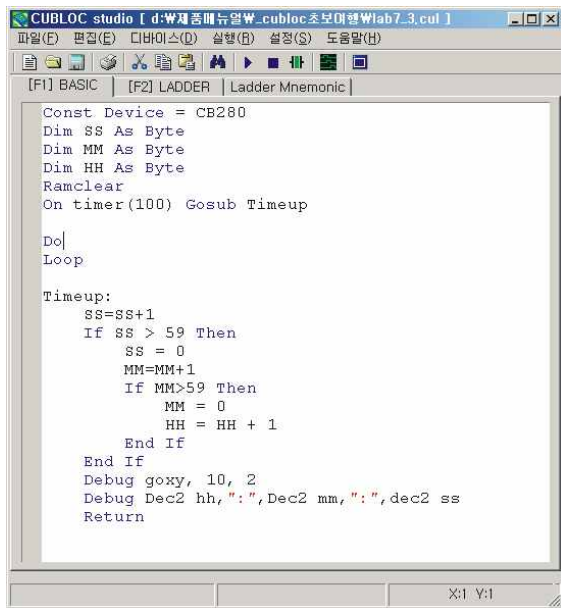
The order of program creation

When one creates a source program, in general the main routine is written first, and then subroutine(s) are written. And if there is a subprogram, put an END command at the end of the subprogram.



Timer Interrupt Application Source

To better understand timer interrupt, let us create a program that implements a timer.



```

CUBLOC studio [ d:\재능교육\cubloc초보대결\lab7.3.cul ]
파일(F) 편집(E) 디버깅(D) 실행(R) 설정(S) 도움말(H)
[F1] BASIC [F2] LADDER Ladder Mnemonic
Const Device = CB280
Dim SS As Byte
Dim MM As Byte
Dim HH As Byte
Ramclear
On timer(100) Gosub Timeup

Do
Loop

Timeup:
  SS=SS+1
  If SS > 59 Then
    SS = 0
    MM=MM+1
    If MM>59 Then
      MM = 0
      HH = HH + 1
    End If
  End If
  Debug goxy, 10, 2
  Debug Dec2 hh, ":", Dec2 mm, ":", dec2 ss
  Return
  
```

Once you enter the above program and run it, currently lapsed time is displayed on the Debug Terminal. This can happen, as the timer interrupt happens every 1 second to increment some of the variables' values and display them.



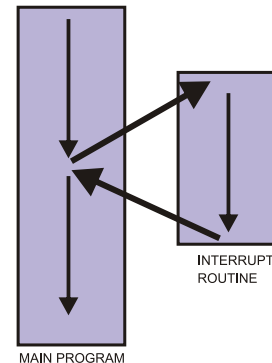
HELPFUL TIPS

What is an Interrupt?

Since BASIC is a “sequential language,” which processes its program from the beginning to the end in the order it is written, it is difficult for it to handle emergency situations where something out of the order has to be taken care of. Interrupt is a concept that was born in response to such need.

While you were doing a homework, if you get a phone call, you stop doing the homework and take the call. And, when you finish the call, you resume doing the homework. Interrupt has a similar concept.

Beside the timer interrupt just explained, CUBLOC is equipped with other type of interrupt capabilities.



The next table summarizes the interrupts that are possible in CUBLOC.

Name of the Interrupt	Description
On Timer	generates interrupts periodically at a set interval.
On Int	generates an interrupt based on state changes happening at an external pin.
On Recv	generates an interrupt when there is an incoming signal on an RS232 interface.
On LadderInt	generates an interrupt when there is an interrupt request from a LadderLogic program.
On Pad	generates an interrupt when there is a pad communication request from, say, a keypad.



If you want to suppress the use of interrupts entirely, use the command SET ONGLOBAL OFF. If you would like to allow interrupts again, use the command SET ONGLOBAL ON. The initial default state is ON.

About The Counter Input Channel

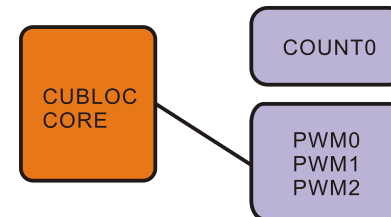
CUBLOC has 2 high-speed counter input channels. We have created a program that used Counter Channel 1. But what should we do to use Counter Channel 0? Besides changing the source code from Count(1) to Count(0), there is one more thing to do.

```
Const Device = CB280
Dim R as Integer
Set Count0 on      'Activate Counter Channel 0.
R = Count(0)
```

SOUT	1	●	17	VDD	TX1	33	●	49	TTLTX1
SIN	2	●	18	VSS	RX1	34	●	50	TTLRX1
ATN	3	●	19	RES	AVDD	35	●	51	AVREF
VSS	4	●	20	N/C	N/C	36	●	52	P48
SS-P0	5	●	21	P16	AD0-P24	37	●	53	P31-AD7
SCK-P1	6	●	22	P17	AD1-P25	38	●	54	P30-AD6
MOSI-P2	7	●	23	P18	AD2-P26	39	●	55	P29-AD5
MISO-P3	8	●	24	P19-PWM3	AD3-P27	40	●	56	P28-AD4
P4	9	●	25	P20-PWM4-INT0	P47	41	●	57	P32
PWM0-P5	10	●	26	P21-PWM5-INT1	P46	42	●	58	P33
PWM1-P6	11	●	27	P22-INT2	P45	43	●	59	P34
PWM2-P7	12	●	28	P23-INT3	P44	44	●	60	P35
CUNET_SDL-P8	13	●	29	P15-HCOUNT1	P43	45	●	61	P36
CUNET_SDA-P9	14	●	30	P14-HCOUNT0	P42	46	●	62	P37
P10	15	●	31	P13	P41	47	●	63	P38
P11	16	●	32	P12	P40	48	●	64	P39

In CUBLOC, the high-speed Counter Channel 1 is always available; but it is not so with Counter Channel 0. To use it, one has to activate it using the command SET COUNT0 ON because initially (by default) COUNT0 is set to OFF.

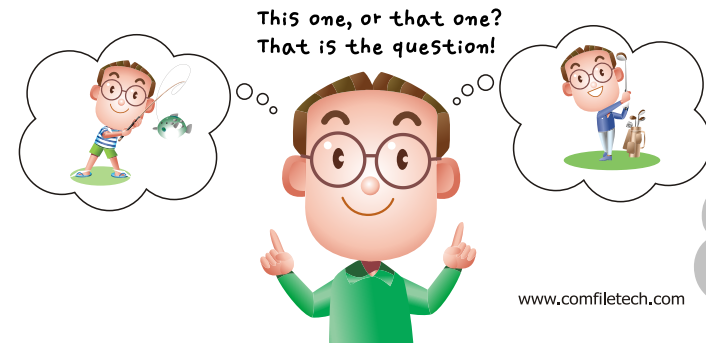
There are things you have to give up. Those are three PWM channels: 0, 1 and 2. CUBLOC usually has 6 PWM channels which are used in PWM or FREQOUT commands. Since the first 3 channels share common resources with Counter Channel 0, one can use only one of the two options at one time; i.e., either PWM channels 0, 1 and 2, or Counter Channel 0.



As shown above, initially, PWM channels 0, 1 and 2 are available to the CUBLOC core. By issuing the command SET COUNT0 ON, the availability of PWM channels 0, 1 and 2 is cancelled and now COUNT0 becomes available to the CUBLOC core.

When command SET COUNT0 OFF is issued, PWM channels 0, 1 and 2 become available again.

Thus, one has to choose between the two: either PWM channels 0, 1 and 2, or Counter Channel 0. But note that even when you give up the first 3 PWM channels, still PWM channels 3, 4 and 5 are available.





NEW COMMANDS

Set

Set <reserved word> ON or OFF

Set <reserved word> <parameter(s)>

This is command used to set a certain environment. After the command SET one has to write a preset <reserved word>. For example, to determine whether COUNT0 is activated or not, after SET one writes COUNT0 and then ON or OFF. E.g.,

```
Set Count0 On
```

There are many other reserved words.

Below are SET commands related to banning or allowing interrupts.

```
Set Onglobal Off ' Disallow all interrupts.
Set OnInt Off ' Disallow all external pin interrupts (INT).
Set Ontimer Off ' Disallow timer interrupts.
Set Onrecv Off ' Disallow all interrupts based on RS232 reception.
Set OnPad Off ' Disallow all pad input interrupts.
Set OnLadderInt Off ' Disallow all interrupts from LadderLogic.
```

By changing the OFF to ON in above examples, each type of disallowed interrupts are now allowed.

A command that determines the start of LadderLogic is also a SET command:

```
Set Ladder On 'Begin running of a LadderLogic program.
```

The command that blocks running of DEBUG commands is also a SET command:

```
Set Debug Off ' Do not interpret (i.e., process) the DEBUG commands in the
source program.
```

Some of the SET command do not use ON or OFF but numbers as parameters.

```
Set Display 2,0,1,50 ' Display-related setting
```

This command is related to LCD display setting, which we will learn later in detail.

APPLIED TASK

- 1 Create a program that displays the counter-inputted number on an LED.

Great job!!

We have studied how to measure frequency using the counter input and the timer interrupt. Interrupts and timers are frequently used in actual situations.



COMMANDS LEARNED IN THIS CHAPTER

Count
OnTimer
Gosub, Return
Set Count0



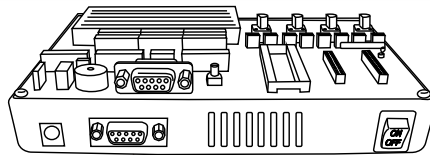
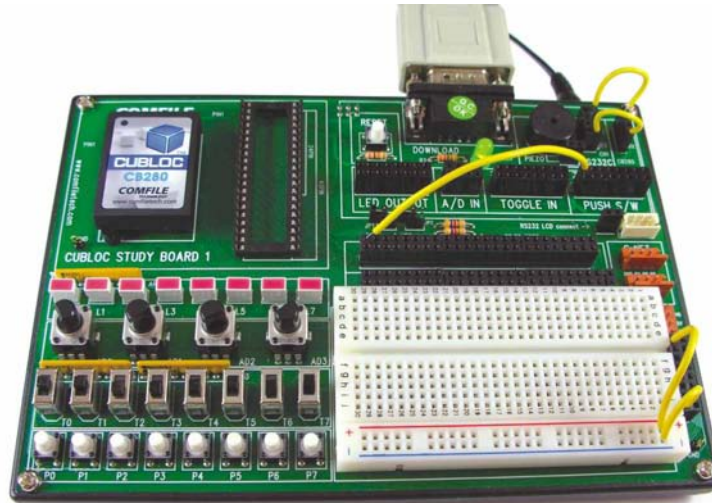
OPERATIONAL SUMMARY

What should be done if some data has to be received from the PC and later some other data has to be sent back to it? The most common way is to use the RS232 communication protocol. In this lab, we will conduct an experiment where the CUBLOC exchanges data with a PC using CUBLOC's RS232 feature.

LAB.8 RS232 Communication

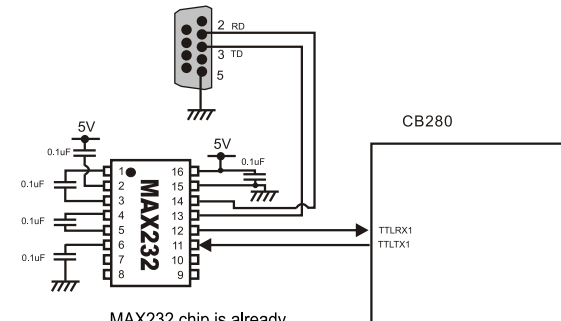
Circuit Configuration

CB280 has an I/O pin dedicated for RS232 communication. This pin can only be used for RS232 Channel 1's transmission (TX) and reception (RX). To use CB280's RS232 Channel 1 on the Study Board, the connection shown below has to be made. And Port 0 has to be connected to Push Switch 0.

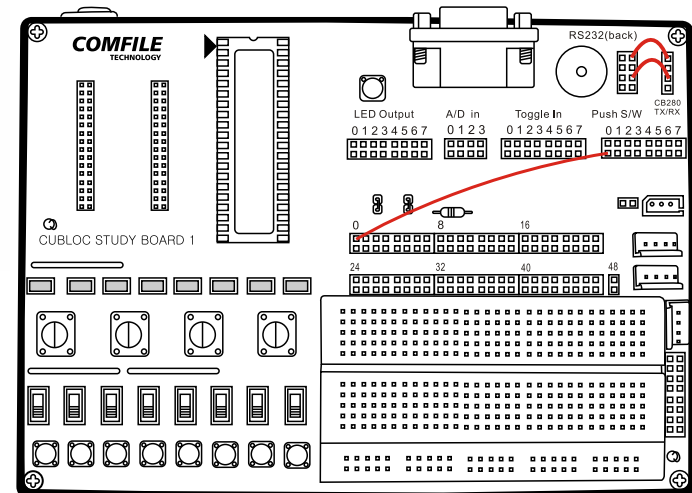


Once the CB280's TX and RX are connected as shown above, the RS232 port on the backside of the Study Board can now be used as CB280's Channel 1 port!

Circuit Diagram

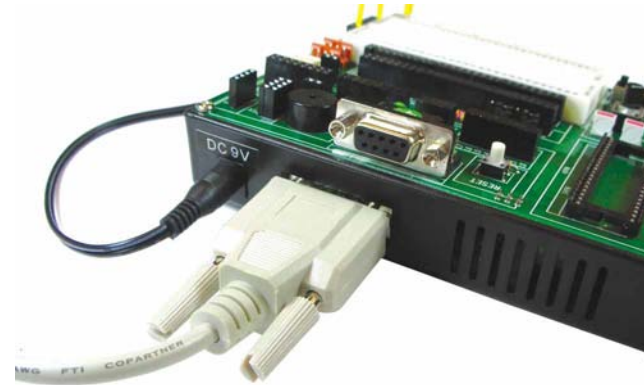
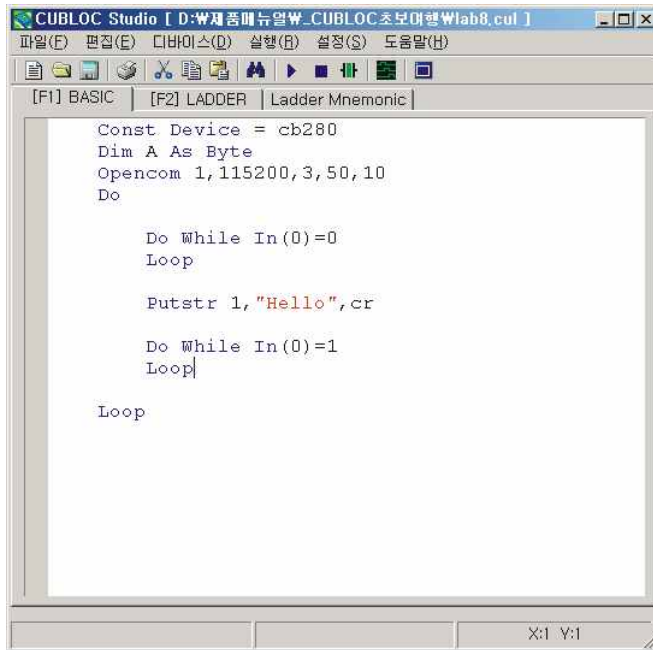


MAX232 chip is already included on the Study Board.



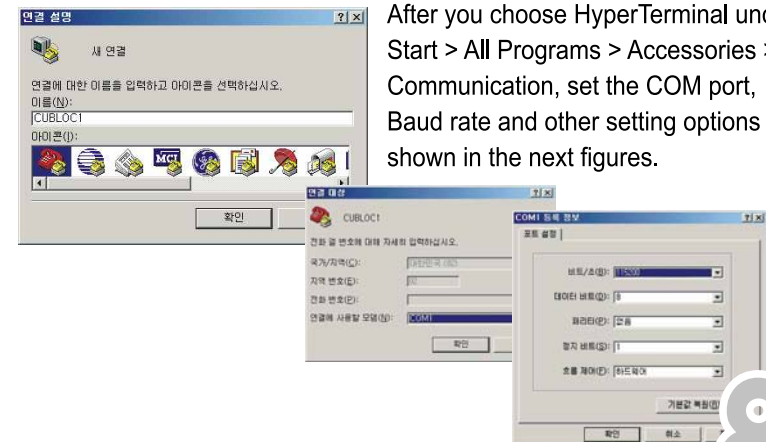
Source Program

Enter the source program as shown below.



Currently PC's COM port is connected to CUBLOC's download port. Since most PCs only have one COM port, we will just connect the PC's COM port with CUBLOC's Channel 1 port as shown in the above picture.

And on the PC, we will open up the HyperTerminal program. HyperTerminal is a program which lets one visually monitor the communication status and it comes with the Window XP as a standard feature.



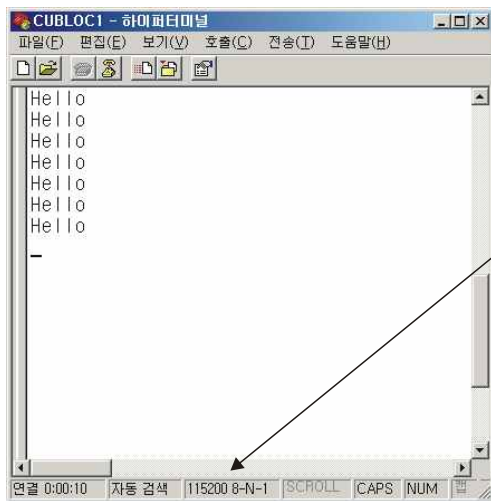
After you choose HyperTerminal under Start > All Programs > Accessories > Communication, set the COM port, Baud rate and other setting options as shown in the next figures.

Confirmation Run-1

This is a program that outputs "Hello" via the RS232 Channel 1 when the push switch is pressed. To confirm the run result of this program, one has to connect CUBLOC's Channel 1 port and PC using an RS232 cable.

Confirmation Run-2

If you press the PUSH switch, you will see the character string “Hello” being displayed on HyperTerminal.



If you see here, you can confirm the status of the HyperTerminal. If you cannot see the status here well, check that the communications options are correctly set to 115200-8-N-1.

Explanation of the Source Program-1

```
Const Device = cb280
Dim A As Byte
Opencom 1,115200,3,50,10
Do
```

This command opens the RS232 Channel 1. This sets the Baud rate to 11520, communication protocol to 3 (i.e., 8 bits, No parity, 1 Stop bit), Reception Buffer capacity to 50 bytes, and Transmission Buffer capacity to 10 bytes.

```
Do While In(0)=0
Loop
```

Wait (i.e., Loop) while the I/O Port 0 (i.e. the status of the PUSH switch) is 0 (i.e., Low). That is, wait until the switch is turned ON.

```
Putstr 1,"Hello",cr
```

```
Do While In(0)=1
Loop
```

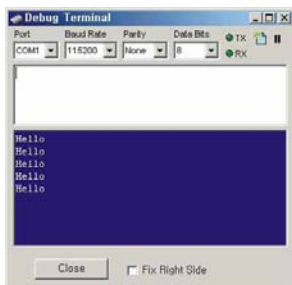
Wait (i.e., Loop) while the I/O Port 0 (i.e. the status of the PUSH switch) is 1 (i.e., High). That is, wait until the switch is turned OFF, i.e., until one takes his/her finger off the push switch.

```
Loop
```

With the OPENCOM command one declares the beginning of an RS232 communication.

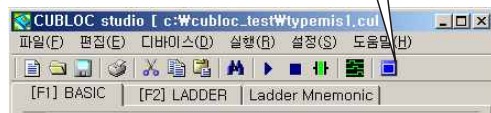
In CUBLOC, RS232 communication is implemented using a buffer. Therefore, one has to declare in advance the sizes of the buffers to be used and the communication protocol.

This program outputs a character string “Hello” through the RS232 communication port when the PUSH switch is pressed. Putstr is very the command that outputs a character string via the RS232 port.



Even without the HyperTerminal, one can confirm the same result on CUBLOC's Debug Terminal.

If you click here, the Debug Terminal will be displayed.





NEW COMMANDS

Opencom

OpenCom <Channel No.>, <Baud Rate>, <Protocol>, <Receive Buffer Size>, <Send Buffer Size>

Channel: The channel to be used

Baud Rate: one of 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400

Protocol: 3 = 8, N, 1 (8 bits, No parity, 1 stop bit)
19 = 8, E, 1 (8 bits, Even parity, 1 stop bit)
27 = 8, O, 1 (8 bits, Odd parity, 1 stop bit)

This command is for initial setting of RS232 channel before its use.

Thus, this command MUST be used before beginning RS232 communication. The Baud rate is the speed of the channel, i.e., how fast it receives or transmits information.

Putstr

PutStr <Channel No.> <Character String>

This command transmits a character string through the specified communication channel. Actually when this command is carried out, it just stores the character string in the transmission buffer.

Then CUBLOC automatically sends the character string through the channel one byte at a time. Since CUBLOC keeps transmitting until the buffer is empty, the user need not worry about the implementation details.

Putstr 1, "Comfile", Dec A, CR 'Send "Comfile" and variable A's value in decimal number format.

Put

Put <Channel No.> <Data> <No. Bytes to be sent>

This command sends data through a specified RS232 channel. For <Data> one can put a constant or a variable. As for <No. Bytes to be sent> one can use 1 through 4. E.g., for a Byte type data, one should write 1; and for Long type, one should write 4.

Explanation of the Source Program-2

Put command sends only one variable value.

The previously mentioned 'PutStr 1, "Hello", CR' can be re-written using PUT as follows:

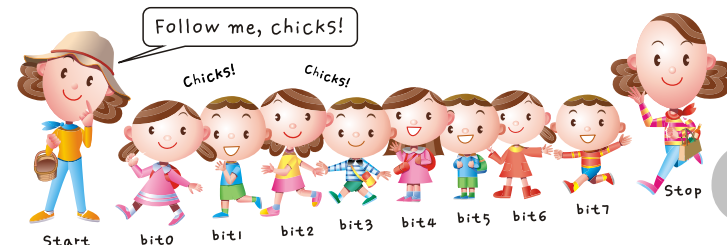
```
Put 1, asc("H"), 1
Put 1, asc("e"), 1
Put 1, asc("l"), 1
Put 1, asc("l"), 1
Put 1, asc("o"), 1
Put 1, 13, 1
Put 1, 10, 1
```

= Putstr 1, "Hello", cr

The "1" at the end of the PUT commands mean just send 1 byte. The ASC function returns a character's ASCII value. In the HyperTerminal, as character is displayed based on its ASCII value. The same goes with the previous learned Debug Terminal. In ASCII code, character A has the value &h41 and character B has the value &h42. That is, when &h42 is sent, "B" is outputted.

(ASCII stands for American Standard Code for Information Interchange, and it is a notation system to represent Alphabets and commonly used special characters such as \$, %, #, &, etc.)

13 and 10 are control codes for a carriage return. (In the olden typewriter days, the "carriage return" lever was used to roll up the paper by one line so that the characters will be typed on the next line.) These control codes moves the cursor one line down and at the beginning of the next line so that the next character will be outputted at the beginning of the next line.

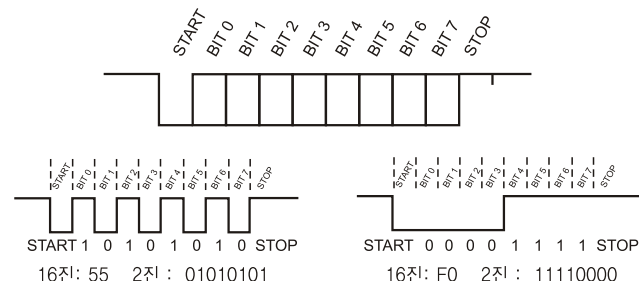


ASCII Code	Character	ASCII Code	Character	ASCII Code	Character	ASCII Code	Character
00H	NUL	20H	SPACE	40H	@	60H	`
01H	SOH	21H	!	41H	A	61H	a
02H	STX	22H	“	42H	B	62H	b
03H	ETX	23H	#	43H	C	63H	c
04H	EOT	24H	\$	44H	D	64H	d
05H	ENQ	25H	%	45H	E	65H	e
06H	ACK	26H	&	46H	F	66H	f
07H	BEL	27H	•	47H	G	67H	g
08H	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

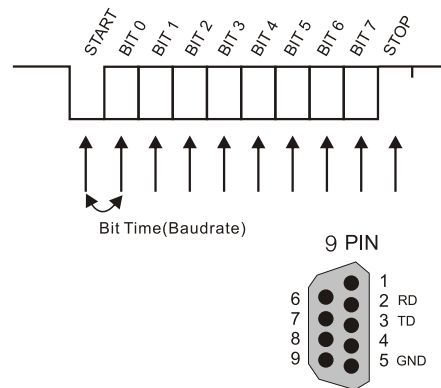


HELPFUL TIPS

The way RS232 communication is implemented is by sending information via 1 line of TX (transmission). But how can 8 bits be sent through a 1-bit line? The secret is in time management. By time-slicing, bits of information can be sent via RS232 transmission of a single-bit line.



RS232 communication is implemented by analyzing the RS232 waveform from the opposite position. First, the start position is found. Then the status of the transmission line is checked periodically. This period is inversely proportional to the frequency, which is the so-called Baud Rate. The higher the baud rate, the more data can be transmitted



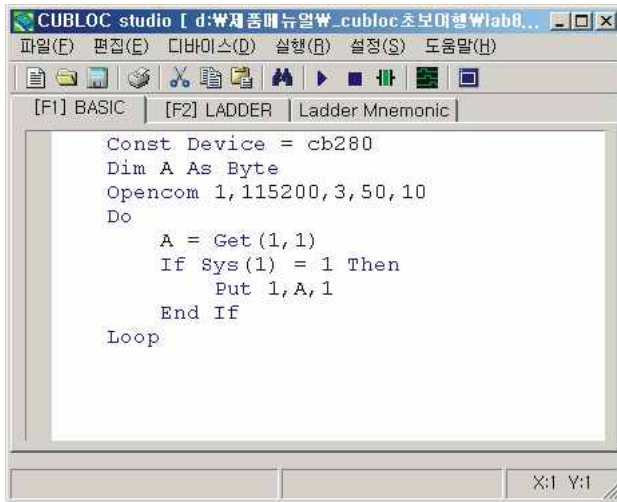
When one examines the RS232 port of a PC, it consists of 9 pins; but only 2 pins (TX and RX) are used to send and receive data, and the rest of the pins are “test pins” used to test the send /receive status and thus frequently not used. CUBLOC supports the use of only the 2 pins: TX and RX.



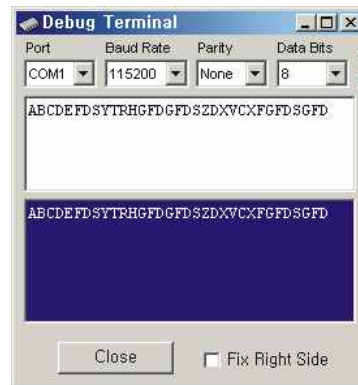
English character A's
ASCII value is 41H!

Applied Programming-1

This time let us create an RS232 reception program. After creating the program below, download it to the CUBLOC. Don't forget to switch back the RS232 cable to the DOWNLOAD position.



Let's confirm that the above program is working correctly by checking it on the Debug Terminal. Plug the RS232 cable into Channel 1, place the cursor to the upper window of the Debug Terminal, then if you type anything in the upper window, it should be copied onto the lower window.



Let's take a close look into the source program.

```
Const Device = cb280
Dim A As Byte
Opencom 1,115200,3,50,10
Do
  A = Get(1,1)
  If Sys(1) = 1 Then
    Put 1,A,1
  End If
Loop
```

Open the RS232 port

Read one byte of data from the receive buffer

If there is some incoming data, the value is SYS(1) is set to 1. Then the incoming data is sent out without any change.

The Do-Loop keeps executing the Get function. Get function read a byte of data from the RS232 receive buffer. If there is no incoming data, nothing can be read off the receive buffer. Since "garbage" (i.e., any random value) could be stored at variable A, if the A value is sent out (even when there is no incoming data), random garbage could be transmitted.

Therefore, to check whether there was some incoming data, function SYS(1) is used. SYS is one of CUBLOC's built-in system function that monitors CUBLOC's various elements. Right after the GET function is executed, SYS(1) returns the number of incoming data bytes, which lets us know if there was any incoming RS232 data.

If there was any incoming data, the same data is sent out using the PUT function. And therefore the COPYING behavior shown on the Debug Terminal described above results.



NEW COMMANDS

Get

<variable> = GET(<channel no.>, <no. of received bytes>)

This function reads data from the RS232 receive buffer. After the RS232 channel is activated using the OPENCOM command, all incoming data is automatically accumulated in the receive buffer, which can be read using the GET function. If the incoming data is not read out of the receive buffer before the buffer fills up, the incoming data after that point will be lost. The number of data in the receive buffer can be found out by the BLEN function.

`A = BLEN(1, 0)` 'This returns the number of data in Channel 1's receive buffer (0).
To indicate the send buffer 1 can be used instead of 0.

For the GET function, the number of bytes to be received has to be specified, which can be from 1 to 4. Here variable A should be of the data type that suits the number of bytes to be assigned to A. E.g., for 2-byte data A should be declared as the INTEGER type and for 4-byte data A should be declared as the LONG type. Note that CUBLOC implements the LOW BYTE FIRST policy; therefore, when a data is stored into a LONG type or a BYTE type, it has to be stored starting from the lower bytes.

`A = Get(1, 2)` 'Read 2 bytes of data from Channel 1's receive buffer.

Getstr

<string variable> = GETSTR(<channel no.>, <no. of received bytes>)

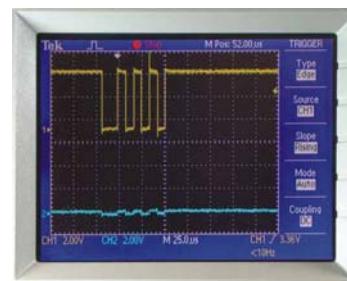
This performs a similar function as the earlier GET function; but the result is a character string. Therefore the number of received bytes could be larger than 4. This function is usually used to receive a large quantity data.

`St1 = Getstr(1, 20)` 'Read 20 bytes of data and store it in St1.

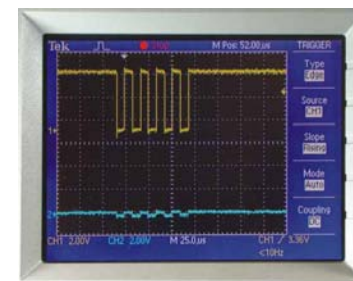


What would be the actual shape of RS232 wave?

Let take a look using an oscilloscope. The next photos show in what forms RS232 waves are sent or received.



The RS232 Waveform of &hAA

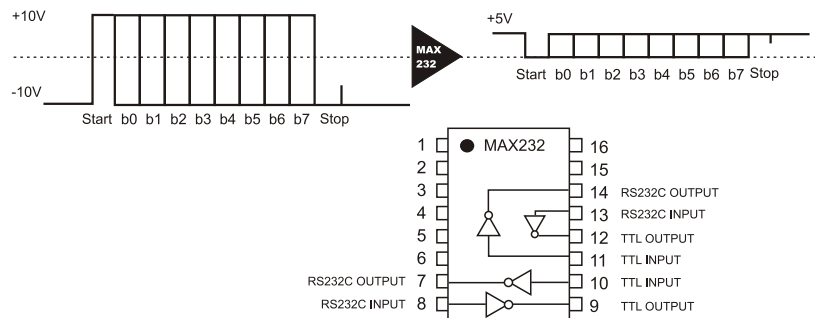


The RS232 Waveform of &h55



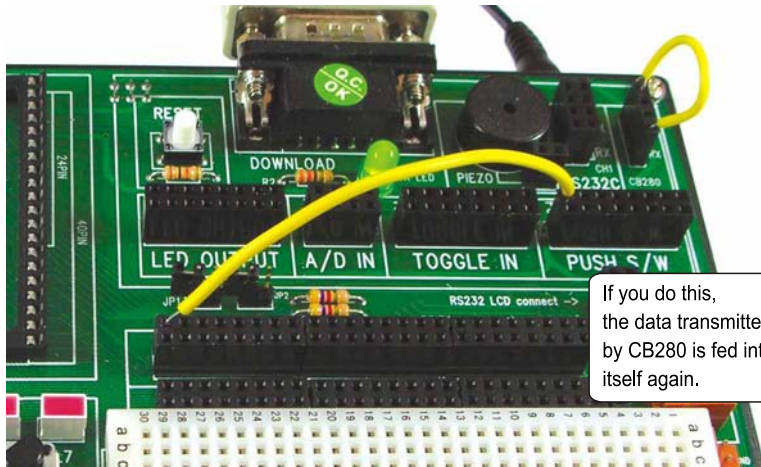
Q: What does the MAX232 chip do?

A: On most print boards that uses RS232 there is a MAX232 chip. This chip is a level converter chip which converts the 5-volt RS232 signal to a +/- 12-volt signal, and vice versa. The RS232 signals out of PCs are all 12-volt signals. If this signal is to be fed into CUBLOC, it has to be converted into a +/- 5 V signal, and that is what MAX232 does. Otherwise, if one feeds the 12 V signal directly into CUBLOC, it will burn! So be careful!!



Applied Programming-2

This time we will create an RS232 reception program using the reception interrupt. As the picture below, connect the RS232 cable to the DOWNLOAD port, and then connect the CB280's TX and RX together.



If you do this, the data transmitted by CB280 is fed into itself again.

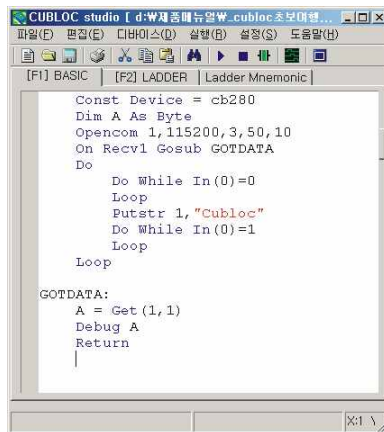
```
Const Device = cb280
Dim A As Byte
Opencom 1,115200,3,50,10
On Recv1 Gosub GOTDATA
Do
    Do While In(0)=0
    Loop
    Putstr 1,"Cubloc"
    Do While In(0)=1
    Loop
Loop
```

If there is any data received, the program execution point is jumped (GOSUBed) to the GOTDATA label.

This part is the same as the portion in the above source program. If any key is pressed, the word "Cubloc" is transmitted to Channel 1.

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

This is the reception interrupt routine, which exactly displays the received data on the Debug Terminal.



Now input the source program shown here and run it. Whenever you press the PUSH switch, if the word "Cubloc" is printed on the Debug Terminal, then the program is working correctly.

Reception interrupt is a frequently used feature in CUBLOC's RS232. In the previous program, the RS232 data reception is checked inside the DO-LOOP. But such implementation (or programming) means the CUBLOC cannot do any other work but just keep checking whether some RS232 data has been received until one finally arrives.

Therefore we need the RS232 reception interrupt. CUBLOC would be processing other tasks; but when there is an incoming RS232 data, it will stop doing other tasks and GOSUB (i.e., jump) to the label previously specified using the ON RECV command.

When the RETURN command is encountered at the end of the reception interrupt routine, the program execution goes back to where it jumped previously, resuming the task CUBLOC was previously processing.



NEW COMMANDS

On Recv1

ON RECV1 GOSUB <label>

This command declares an interrupt should occur when data arrives in the RS232 receive buffer. After this command is issued, an interrupt occurs when any RS232 data is received and the program execution flow "Gosubs" (i.e., jumps) to <label> to execute the interrupt routine.

While the interrupt routine is being carried out, even if another piece of data arrives at the RS232 receive buffer, another interrupt is not generated. Only when the current interrupt routine is completed and the program execution returns to the GOSUB place in the source program can another interrupt occur.

Set Until

SET UNTIL <channel no.>, <no. of reception bytes>[, <termination code>]

If only ON RECV declaration is available, an interrupt happens even one only one byte of data arrives, which can result in frequent inhibition of the main program execution.

Therefore, usually the ON RECV command is used together with the SET UNTIL command. By using this command, one can specify the minimum number of bytes required to cause an interrupt. For instance, if one wrote:

```
SET UNTIL 1, 5
```

at least 5 bytes of data has to arrive before an interrupt can be generated. If one wrote

```
SET UNTIL 1, 99, "S"
```

an interrupt will occur if either 99 bytes of data arrive or a character string "S" arrives.

```
Dim A(5) As Byte
Opencom 1,19200,3,100,50
On Recv1 GotData_rtn
Set Until 1,99,"S"
```

The termination byte can only be a single byte, and when it is not a character it can be a number as follows:

```
SET UNTIL 1, 99, 5
```

'An interrupt will occur if either 99 bytes of data arrive or number 5 arrives.

Now you can use the Receive interrupt; but something seems to be missing. Since a receive interrupt will happen even when one byte of data is received, there is a chance that interrupts may happen too frequently.

Therefore, SET UNTIL command can be used after the ON RECV command to prescribe the minimum number of bytes that have to arrive before an interrupt can happen. Usually this number is called the "packet size" which sets the size of a group of data that will be sent or received as a unit.

As long as the sender knows the minimum number of bytes to be sent, it is better to process multiple bytes at a time.

```
Const Device = cb280
```

```
Dim A As Byte
```

```
Dim St1 As String * 6
```

declares a character string that will temporarily store the received data.

```
Opencom 1,115200,3,50,10
```

```
On Recv1 Gosub GOTDATA
```

```
Set Until 1,6
```

An interrupt will happen when 6 bytes of data are received.

```
Do
```

```
Do While In(0)=0
```

```
Loop
```

```
Putstr 1,"Cubloc"
```

```
Do While In(0)=1
```

```
Loop
```

```
Loop
```

```
GOTDATA:
```

```
st1 = Getstr(1,6)
```

```
Debug st1
```

```
Return
```

Since this interrupt deals with a 6-byte buffer, a Getstr command which can read 6 bytes of data at a time is used. The fetched data is stored in the character string st1 and is displayed using a Debug command.



The other function that the SET UNTIL has is that it can set the termination code/character. When a special character or number is declared as the termination code and when that code is included in the received data, an interrupt is generated.

```
Const Device = CB280
Dim A As Byte
Dim St1 As String * 7
Opencom 1,115200,3,50,10
On Recv1 Gosub GOTDATA
Set Until 1,10,"!"
Do
    Do While In(0)=0
        Loop
        Putstr 1,"Cubloc!"
        Do While In(0)=1
            Loop
        Loop
    Loop
```

The exclamation mark is set as the termination character such that an interrupt is generated when it is detected in the incoming input data stream. At this time, the packet size should be set sufficiently large.

The exclamation mark (i.e., the termination code) is deliberately placed at the end of the transmitted data.

```
GOTDATA:
    st1 = Getstr(1,7)
    Debug st1
    Return
```

When the termination code is detected in the incoming data stream, an interrupt is generated. Since the data stream up to the termination code is read, it is included when the Debug statement prints the read data.



HELPFUL TIPS

There are more to learn about communications, including how to form a packet, Modbus, etc. Yet, since this book is for beginners, we will stop at just learning the concepts. More in-depth knowledge should be obtained when needed by each reader.

APPLIED TASK

- 1 Create a program that delivers A/D converted value to a PC using the CUBLOC's RS232 communication feature.
- 2 And on the PC display the transmitted A/D converted value using the HyperTerminal.

Job well-done!

Communication connects you to the world. Just like people exchange information with others in different places using phones or the Internet, CUBLOC exchanges data with other devices using the RS232 communication interface. If CUBLOC lacks any feature, one can get the help of other devices using the communication interface.



COMMANDS LEARNED IN THIS CHAPTER

OpenCom
PutStr
Put
Sys

Get
GetStr
On Recv
Set Until

OPERATIONAL SUMMARY

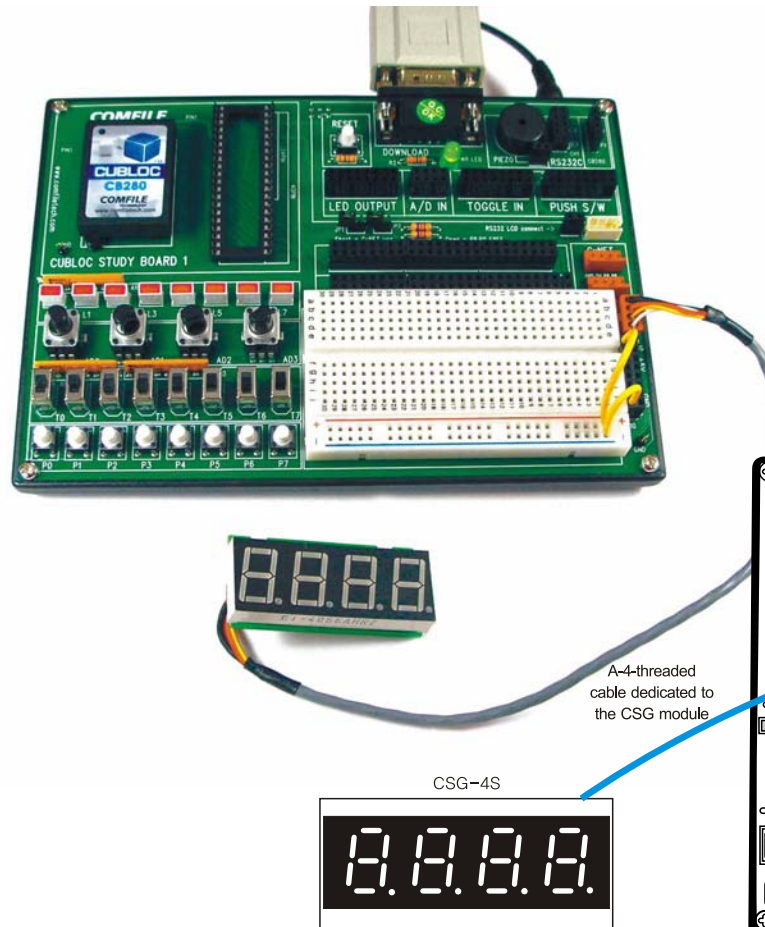
A 7-segment can be easily found in our daily lives. It is a LED display used for displaying simple numbers. A 7-segment can be driven directly by CUBLOC; but such method requires a rather complicated method called dynamic display. This method requires many I/O pins and the CUBLOC cannot handle any other task because it has to keep syncing during the display. Therefore, we will use a separate hardware called CSG module to implement a 7-segment display. Using a CSG module has multiple benefits: One can save many I/O pins, and the CUBLOC can keep doing its main task, as it needs not be dedicated for the 7-segment display.



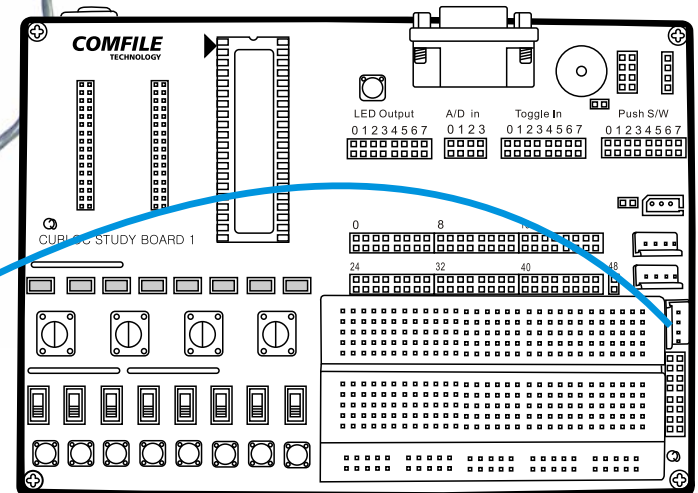
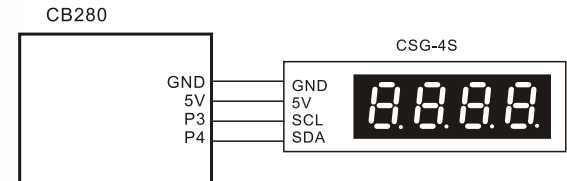
LAB.9 7-segment Display

Circuit Configuration

Connect the CSG module to the CSG port located at the left end of the Study Board.



Circuit Diagram



Confirmation Run

If numbers with increasing values are displayed on the CSG module, the program is properly operating. If not, check the dip switch setting on the backside of the module one more time!

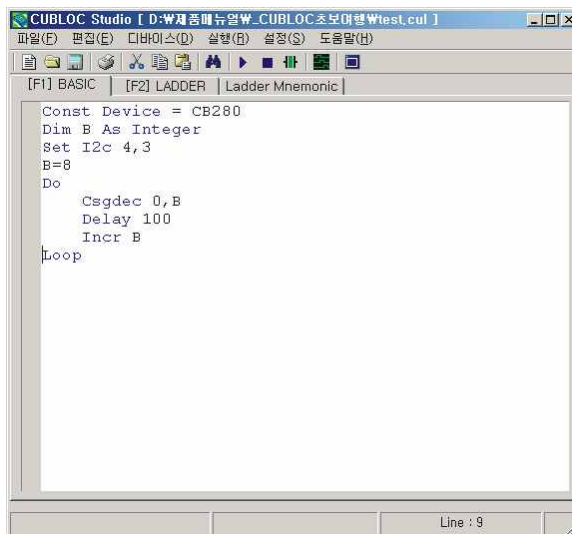
The Backside of the CSG Module

Before use, let us take a quick look at the backside of the CSG module. There are dip switches, which determine the address of the CSG module. As shown in the picture, if only the 2nd dip switch is turned ON and the rest are OFF, that sets the address to 0.

Let's deal with detailed description of the addressing later; for now, let's compose a source program with this setting.



Source Program



Explanation of the Source Program

```
Const Device = CB280
```

```
Dim B As Integer
```

```
Set I2c 4,3
```

This is a command that declares the use of I2C port.

```
B=8
```

```
Do
```

```
    Csgdec 0,B
```

This command displays data on the CSG module in the decimal format.

```
    Delay 100
```

```
    Incr B
```

This command increments the value of B by 1, i.e., it has the same effect as B = B + 1.

```
Loop
```

To use a CGS-related command, one has to declare the use of I2C port in advance, using the SET I2C command.

The CSGDEC is the actual command that transfers data to the CSG module. The first argument 0 means address 0, and the transferred data is stored at variable B.



NEW COMMANDS

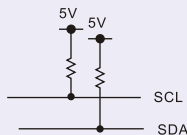
Set I2C

SET I2C <data pin>, <clock pin>

This command declares and sets the data pin and clock pin to be used in the I2C communications. In all I2C-related command that follows, no further information on data pin and clock pin need to be provided because the information declared by this command is referred to.

Thus, this command does not take any actual effect but retains the information as to which pin will be used as the data pin.

All I/O ports (which are bidirectional) of CUBLOC can be used as a clock pin or data pin. And as shown in the figure below, all pins used for the I2C communication should have a pull-up resistance attached to them.



Csgdec

CSGDEC <address>, <data>

This command transfers data to the CSG module at the specified address, and displays the value in decimal format on the module. For this command to work properly, the SET I2C command MUST be issued first to declare the data pin and clock pin.

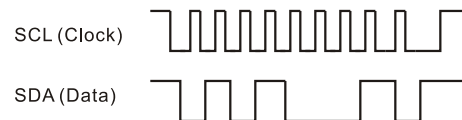
In general a pull-up resistance is necessary for using an I/O pin as an I2C communication pin. But, since this command uses an I/O pin only as an output pin, it is not necessary. Yet, for other usages of an I/O pin as an I2C communication pin, think as a MUST to attach a pull-up resistance.



HELPFUL TIPS

What is I2C Communication?

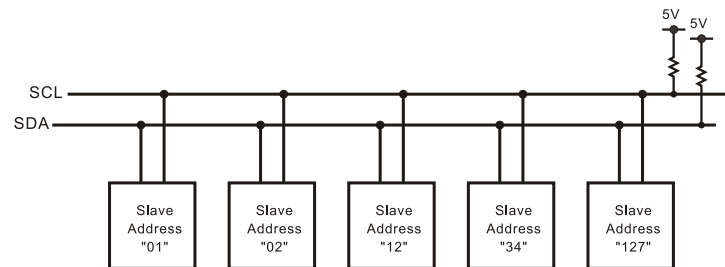
I2C communication is a communication protocol, where only 2 lines a clock line and a data line are used.



While RS232 communication protocol is used primarily for computer-to-computer communications and device-to-device communications, I2C communication protocol is frequently used for chip-to-chip communications and board-to-board communications.

Customarily, for I2C communication, a master and a slave are designated. The master initiates the communication and sends the data, while the slave receives the sent data and performs pertinent actions in a passive manner.

In the example below, a CUBLOC becomes the master and the CSG module becomes the slaves. In general, the slaves have their own unique addresses so that multiple slaves can be connected to a single I2C line.

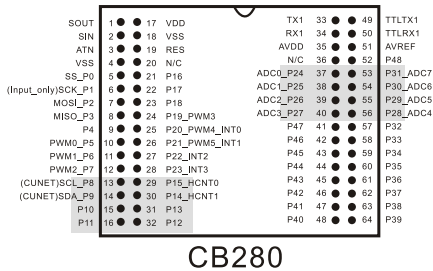


Yet, the master can communicate with only one slave at a time by selecting its address.

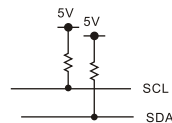

HELPFUL TIPS

I2C and RS232

The I2C communication does not need to set Baud rate, parity or number of bits. Further, while RS232 communication entails some inconvenience because in CUBLOC only the designated communication port can be used; but for I2C communication, all I/O ports can be used bi-directionally. Thus, more devices can be connected with I2C communication than with RS232 protocol.



Since one I2C line can connect up to 128 devices, and all I/O ports can be used, how many devices can be connected altogether?

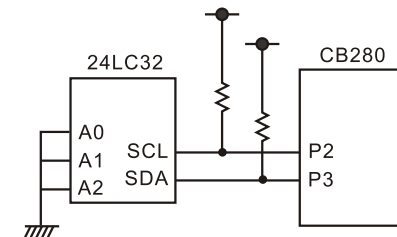


Q: Where is the I2C communication protocol used?

A: Mainly for chip-to-chip communication.

For CUBLOC, I2C protocol is used frequently to make for up CUBLOC's missing functions. For example, CUBLOC only has a 4 KB EEPROM. When a much bigger 256 KB EEPROM has to be used, what can be done?

One can use CUBLOC's I2C protocol to hook up an external 256 KB EEPROM for reading and writing data.



Though the I2C communication protocol seems to be better in all aspects, since CUBLOC supports the I2C communication only in master mode, this means this protocol does not support receiving external data at an arbitrary time point. In this respect, RS232 is better.

When CUBLOC becomes the master to send data out I2C communication protocol is better. But to receive data from outside, I2C protocol cannot be used, as a reception data buffer has to be managed and interrupt handling is necessary.



Also by using the I2C protocol, one can connect to a multi-function chip, e.g., A/D converter, D/A converter, chips that provide supplementary I/O functions.

Note that CUBLOC uses I2C protocol to connect CSG and CLCD, too.

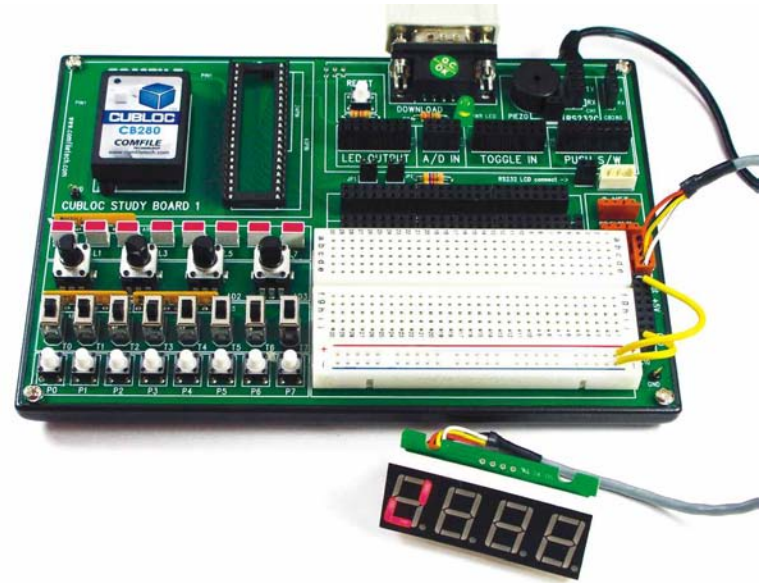
As seen, CUBLOC frequently uses I2C communication protocol.

Applied Programming-1

If one wants to turn on only one LED, what should one do? There is the CSGXPUT command. Though there was the previously introduced CsgDec command, it can only output decimal numbers. For more diverse applications, one needs better control over the seven LCD segments.

```
Const Device = CB280
Dim B As Integer
Set I2c 4,3
B=0
Do
  Csgxput 0,0,B
  Delay 100
  Incr b
Loop
```

On the 7-segment at 0th place (i.e., leftmost segment) in the CSG module of address 0, display the value of variable B.



The 8 LEDs of a 7-segment has a respective number from A to H. As in the previous experiment of turning on and off 8 LEDs, these 8 LEDs can be turned on or off individually. (Note that even though it is called a “7-segment”, it actually has 8 LED segments because a decimal point is also included.) Using this function, one can display even non-numbers, e.g., alphabets or other symbols.



NEW COMMANDS

Csgxput

CSGXPUT <module address>, <7-segment place>, <data>

This command makes it possible to turn on any of the 8 LEDs in a 7-segment module in the CSG module. Since individual LEDs can be turned on or off freely, this command is used to display non-number figures that cannot be expressed with CsgDec or CsgHex.



NEW COMMANDS

Csgnput

CSGNPUT <module address>, <7-segment place>, <data>

This command displays a number in one of the places in the CSG module. The <7-segment place> is 0 for the leftmost digit and 3 for the rightmost digit.

Since the <data> is a number in ASCII code, it ranges from &H30 to &H39(in hexadecimal), which corresponds to decimal numbers 0 to 9. Yet, as exceptions &H3A to &H3F are expressed simply as A to F, respectively.

If other ASCII codes are used as <data>, nothing is displayed.

APPLIED TASK

- Using the CSGXPUT command, display alphabets A, b, E and F.

COMMANDS LEARNED IN THIS CHAPTER

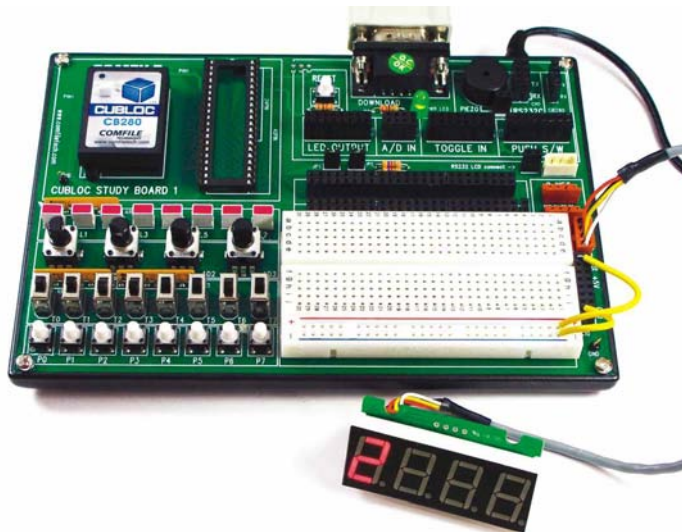
SET I2C
CSGDEC
CSGXPUT
CSGNPUT

Applied Programming-2

There is a command CSGNPUT, which similar to CSGXPUT, but is used for displaying numbers only. By using the CSGNPUT command, one can display an arbitrary number on the wanted place of the CSG module.

```
Const Device = CB280
Dim B As Integer
Set I2c 4,3
B=&H30
Do
    Csgnput 0,0,B
    Delay 100
    Incr b
    If B>&h3f then B = &h30
Loop
```

On the 7-segment at 0th place (i.e., leftmost segment) in the CSG module of address 0, display the value of variable B as a number.



OPERATIONAL SUMMARY

LCD (Liquid Crystal Display) is the display device that is used most widely in recent electronic products. Cell phones, MP3 players, TVs for cars and even facsimiles use LCDs. In this lab, we will learn how to display on an LCD.

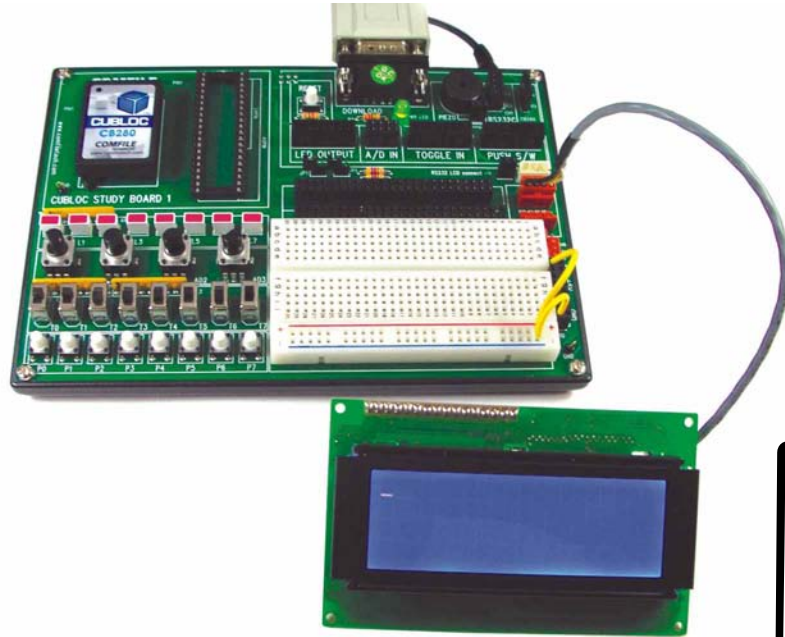


As in the case of the 7-segment, we are not going to have CUBLOC drive the LCD directly, but will use a separate CLCD module to drive it. Directly driving an LCD is a burden, as requires at least 7 I/O pins and one has to study the specification of each LCD model. In this lab, we will simply connect the CLCD module to the CUNET port, and then use LOCATE, PRINT and other known commands to operate the LCD.

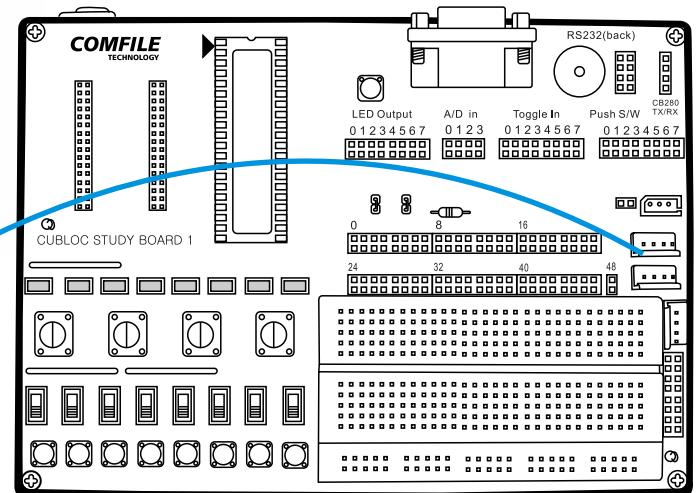
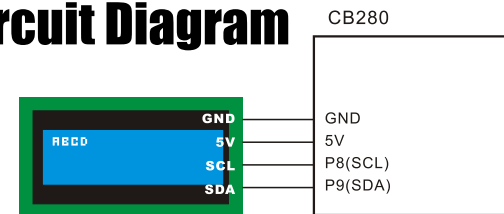
LAB. 10 LCD Display

Circuit Configuration

Connect the CLCD module to the CUNET port of the CUBLOC's Study Board.



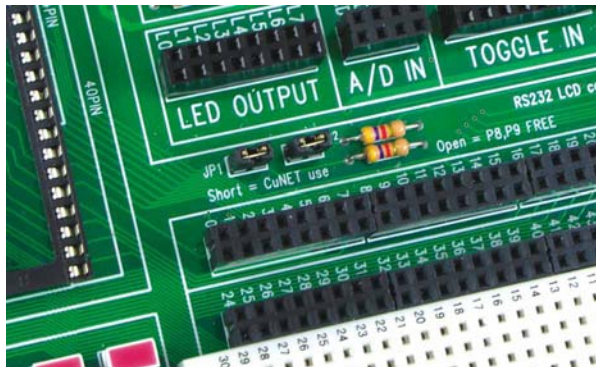
Circuit Diagram



If one looks at the backside of the CLCD module, a small PCB can be seen as the photo below. One can adjust the address using the DIP switches on this board. As in the photo below, set all DIP switches to the OFF position to set the address to 0 (zero).



Then short circuit the JP1 jumper (CuNET Use) of the CUBLOC's Study Board.



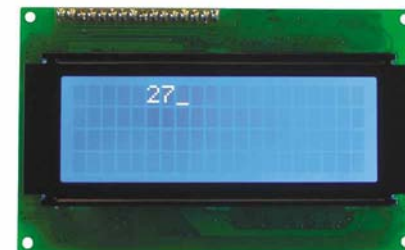
Source Program

```

CUBLOC studio [ d:\재품매뉴얼\cubloc초보머랭\lab10.cul ]
파일(F) 편집(E) 디바이스(D) 실행(R) 설정(S) 도움말(H)
[F1] BASIC [F2] LADDER Ladder Mnemonic

Const Device = CB280
Dim A As Integer
Set Display 2,0,0,50
Cls
Delay 20
A = 0
Do
  Locate 5,0
  Print Dec a
  Delay 200
  Incr a
Loop
  
```

When this source program is inputted and run, a series of increasing numbers will be displayed on the LCD as shown below.



Explanation of the Source Program

```
Const Device = CB280
```

```
Dim A As Integer
```

```
Set Display 2,0,0,50
```

```
Cls
```

```
Delay 20
```

```
A = 0
```

```
Do
```

```
    Locate 5,0
```

```
    Print Dec a
```

```
    Delay 200
```

```
    Incr a
```

```
Loop
```

The display usage should be declared before the display module is used.

After issuing a "Clear the display" command, wait a little bit, as CUBLOC's execution is faster than the LCD.

Set the cursor's position. A character is "printed" at the current cursor position.

A delay is given to maintain the current state of the display for a while.

A text LCD such as CLCD can be comfortably run by a small number of commands such as Cls, Locate and Print. Before using the CLCD, a SET DISPLAY command has to be issued in advance similarly as an OpenCom command has to be issued before using RS232.



NEW COMMANDS

Cls

CLS

This command clears the screen. After this command is issued, the CUBLOC has to wait until the screen is actually cleared. If the next command such as Locate or Print is sent from CUBLOC to CLCD without waiting, CLCD will not be able to receive the next command properly. A delay of about 20 milliseconds (e.g., DELAY 20) is sufficient.

Set Display

SET DISPLAY type, transmission method, address, buffer size

Type: 0 = ALCD, 1 = GHLCD (GHB3224), 2 = CLCD

Transmission Method: 0 = CuNET, 1 = RS232 CH1

Address: LCD's address when CUNET, Baud Rate when RS232

Buffer Size: The size of the transmission buffer, which should range from 50 to 128.

For the Type argument, the LCD type is specified. For ALCD, write 0; for GHB3224 series, write 1; and for CLCD, write 2. When a new series of LCDs developed, further numbers such as 3 or 4 could be newly assigned.

There are 2 kinds of transmission method: One can use the RS232 Channel 1 or the CuNET port. When RS232 is chosen, one should write the Baud rate as the argument value. Note that when the RS232 Ch. 1 is used for LCD transmission, the RS232 cannot be used for other purpose.

Therefore, it may be a good idea to use the CuNET port as the display port.

When the CuNET port is used, use the LCD's DIP switches to specify the chosen address. Both the RS232 and CuNET use a transmission buffer to send data.

Let me explain why a transmission buffer is needed. While CUBLOC is fast, a display device is slow. If CUBLOC has to wait until a display carries out a command, CUBLOC will be slowed down, too.

Therefore, CUBLOC stores a command in a transmission buffer and proceeds to the next line of code. The data(e.g., command) in the buffer is automatically released one byte at a time, matching the pace of the display device until the buffer is empty.

This way, CUBLOC can keep its original fast pace while the display runs at its own pace. If the buffer size is too small, it cannot store all the data sent from CUBLOC. Thus, a value between 50 and 128 should be specified.